Red Hat Enterprise Linux 8

Using systemd unit files to customize and optimize your system

Optimize system performance and extend configuration with systemd

Last Updated: 2024-03-20

Red Hat Enterprise Linux 8 Using systemd unit files to customize and optimize your system

Optimize system performance and extend configuration with systemd

Legal Notice

Abstract

Modify the systemd unit files and extend the default configuration, examine the system boot performance and optimize systemd to shorten the boot time.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the Jira website.
2. Click **Create** in the top navigation bar.
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. WORKING WITH SYSTEMD UNIT FILES

The **systemd** unit files represent your system resources. As a system administrator, you can perform the following advanced tasks:

- Create custom unit files
- Modify existing unit files
- Work with instantiated units

1.1. INTRODUCTION TO UNIT FILES

A unit file contains configuration directives that describe the unit and define its behavior. Several **systemctl** commands work with unit files in the background. To make finer adjustments, you can edit or create unit files manually. You can find three main directories where unit files are stored on the system, the **/etc/systemd/system/** directory is reserved for unit files created or customized by the system administrator.

Unit file names take the following form:

```
<unit_name>.<type_extension>
```

Here, *unit_name* stands for the name of the unit and *type_extension* identifies the unit type.

For example, you can find an **sshd.service** as well as an **sshd.socket** unit present on your system.

Unit files can be supplemented with a directory for additional configuration files. For example, to add custom configuration options to **sshd.service**, create the **sshd.service.d/custom.conf** file and insert additional directives there. For more information on configuration directories, see Modifying existing unit files.

The **systemd** system and service manager can also create the **sshd.service.wants/** and **sshd.service.requires/** directories. These directories contain symbolic links to unit files that are dependencies of the **sshd** service. **systemd** creates the symbolic links automatically either during installation according to [Install] unit file options or at runtime based on [Unit] options. You can also create these directories and symbolic links manually.

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

Many unit file options can be set using the so called **unit specifiers** – wildcard strings that are dynamically replaced with unit parameters when the unit file is loaded. This enables creation of generic unit files that serve as templates for generating instantiated units. See Working with instantiated units.

1.2. SYSTEMD UNIT FILES LOCATIONS

You can find the unit configuration files in one of the following directories:

Table 1.1. systemd unit files locations

Directory	Description
/usr/lib/systemd/system/	**systemd** unit files distributed with installed RPM packages.
/run/systemd/system/	**systemd** unit files created at run time. This directory takes precedence over the directory with installed service unit files.
/etc/systemd/system/	**systemd** unit files created by using the **systemctl enable** command as well as unit files added for extending a service. This directory takes precedence over the directory with runtime unit files.

The default configuration of **systemd** is defined during the compilation and you can find the configuration in the **/etc/systemd/system.conf** file. By editing this file, you can modify the default configuration by overriding values for **systemd** units globally.

For example, to override the default value of the timeout limit, which is set to 90 seconds, use the **DefaultTimeoutStartSec** parameter to input the required value in seconds.

```
DefaultTimeoutStartSec=required value
```

1.3. UNIT FILE STRUCTURE

Unit files typically consist of three following sections:

The [Unit] section

Contains generic options that are not dependent on the type of the unit. These options provide unit description, specify the unit's behavior, and set dependencies to other units. For a list of most frequently used [Unit] options, see Important [Unit] section options.

The [Unit type] section

Contains type-specific directives, these are grouped under a section named after the unit type. For example, service unit files contain the **[Service]** section.

The [Install] section

Contains information about unit installation used by **systemctl enable** and **disable** commands. For a list of options for the **[Install]** section, see Important [Install] section options .

Additional resources

- Important [Unit] section options
- Important [Service] section options
- Important [Install] section options

1.4. IMPORTANT [UNIT] SECTION OPTIONS

The following tables lists important options of the [Unit] section.

Table 1.2. Important [Unit] section options

Option [a]	Description
Description	A meaningful description of the unit. This text is displayed for example in the output of the **systemctl status** command.
Documentation	Provides a list of URIs referencing documentation for the unit.
After[b]	Defines the order in which units are started. The unit starts only after the units specified in **After** are active. Unlike **Requires**, **After** does not explicitly activate the specified units. The **Before** option has the opposite functionality to **After**.
Requires	Configures dependencies on other units. The units listed in **Requires** are activated together with the unit. If any of the required units fail to start, the unit is not activated.
Wants	Configures weaker dependencies than **Requires**. If any of the listed units does not start successfully, it has no impact on the unit activation. This is the recommended way to establish custom unit dependencies.
Conflicts	Configures negative dependencies, an opposite to **Requires**.

[a] For a complete list of options configurable in the [Unit] section, see the **systemd.unit(5)** manual page.

[b] In most cases, it is sufficient to set only the ordering dependencies with **After** and **Before** unit file options. If you also set a requirement dependency with **Wants** (recommended) or **Requires**, the ordering dependency still needs to be specified. That is because ordering and requirement dependencies work independently from each other.

1.5. IMPORTANT [SERVICE] SECTION OPTIONS

The following tables lists important options of the [Service] section.

Table 1.3. Important [Service] section options

Option [a]	Description

Option [a]	Description
Type	Configures the unit process startup type that affects the functionality of **ExecStart** and related options. One of: * **simple** – The default value. The process started with **ExecStart** is the main process of the service. * **forking** – The process started with **ExecStart** spawns a child process that becomes the main process of the service. The parent process exits when the startup is complete. * **oneshot** – This type is similar to **simple**, but the process exits before starting consequent units. * **dbus** – This type is similar to **simple**, but consequent units are started only after the main process gains a D-Bus name. * **notify** – This type is similar to **simple**, but consequent units are started only after a notification message is sent via the sd_notify() function. * **idle** – similar to **simple**, the actual execution of the service binary is delayed until all jobs are finished, which avoids mixing the status output with shell output of services.
ExecStart	Specifies commands or scripts to be executed when the unit is started. **ExecStartPre** and **ExecStartPost** specify custom commands to be executed before and after **ExecStart**. **Type=oneshot** enables specifying multiple custom commands that are then executed sequentially.
ExecStop	Specifies commands or scripts to be executed when the unit is stopped.
ExecReload	Specifies commands or scripts to be executed when the unit is reloaded.
Restart	With this option enabled, the service is restarted after its process exits, with the exception of a clean stop by the **systemctl** command.
RemainAfterExit	If set to True, the service is considered active even when all its processes exited. Default value is False. This option is especially useful if **Type=oneshot** is configured.

Option [a]	Description

[a] For a complete list of options configurable in the [Service] section, see the **systemd.service(5)** manual page.

1.6. IMPORTANT [INSTALL] SECTION OPTIONS

The following tables lists important options of the [Install] section.

Table 1.4. Important [Install] section options

Option [a]	Description
Alias	Provides a space-separated list of additional names for the unit. Most **systemctl** commands, excluding **systemctl enable**, can use aliases instead of the actual unit name.
RequiredBy	A list of units that depend on the unit. When this unit is enabled, the units listed in **RequiredBy** gain a **Require** dependency on the unit.
WantedBy	A list of units that weakly depend on the unit. When this unit is enabled, the units listed in **WantedBy** gain a **Want** dependency on the unit.
Also	Specifies a list of units to be installed or uninstalled along with the unit.
DefaultInstance	Limited to instantiated units, this option specifies the default instance for which the unit is enabled. See Working with instantiated units.

[a] For a complete list of options configurable in the [Install] section, see the **systemd.unit(5)** manual page.

1.7. CREATING CUSTOM UNIT FILES

There are several use cases for creating unit files from scratch: you could run a custom daemon, create a second instance of some existing service as in Creating a custom unit file by using the second instance of the sshd service

On the other hand, if you intend just to modify or extend the behavior of an existing unit, use the instructions from Modifying existing unit files.

Procedure

1. To create a custom service, prepare the executable file with the service. The file can contain a custom-created script, or an executable delivered by a software provider. If required, prepare a PID file to hold a constant PID for the main process of the custom service. You can also include

environment files to store shell variables for the service. Make sure the source script is executable (by executing the **chmod a+x**) and is not interactive.

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

```
# touch /etc/systemd/system/<name>.service

# chmod 664 /etc/systemd/system/<name>.service
```

Replace *<name>* with a name of the service you want to created. Note that the file does not need to be executable.

3. Open the created ***<name>*.service** file and add the service configuration options. You can use various options depending on the type of service you wish to create, see Unit file structure. The following is an example unit configuration for a network-related service:

```
[Unit]
Description=<service_description>
After=network.target

[Service]
ExecStart=<path_to_executable>
Type=forking
PIDFile=<path_to_pidfile>

[Install]
WantedBy=default.target
```

- *<service_description>* is an informative description that is displayed in journal log files and in the output of the **systemctl status** command.
- the **After** setting ensures that the service is started only after the network is running. Add a space-separated list of other relevant services or targets.
- *path_to_executable* stands for the path to the actual service executable.
- **Type=forking** is used for daemons that make the fork system call. The main process of the service is created with the PID specified in *path_to_pidfile*. Find other startup types in Important [Service] section options.
- **WantedBy** states the target or targets that the service should be started under. Think of these targets as of a replacement of the older concept of runlevels.

4. Notify **systemd** that a new ***<name>*.service** file exists:

```
# systemctl daemon-reload

# systemctl start <name>.service
```

WARNING

Always execute the **systemctl daemon-reload** command after creating new unit files or modifying existing unit files. Otherwise, the **systemctl start** or **systemctl enable** commands could fail due to a mismatch between states of **systemd** and actual service unit files on disk. Note, that on systems with a large number of units this can take a long time, as the state of each unit has to be serialized and subsequently deserialized during the reload.

1.8. CREATING A CUSTOM UNIT FILE BY USING THE SECOND INSTANCE OF THE SSHD SERVICE

If you need to configure and run multiple instances of a service, you can create copies of the original service configuration files and modifying certain parameters to avoid conflicts with the primary instance of the service.

Procedure

To create a second instance of the **sshd** service:

1. Create a copy of the **sshd_config** file that the second daemon will use:

```
# cp /etc/ssh/sshd{,-second}_config
```

2. Edit the **sshd-second_config** file created in the previous step to assign a different port number and PID file to the second daemon:

```
Port 22220
PidFile /var/run/sshd-second.pid
```

   See the **sshd_config**(5) manual page for more information about **Port** and **PidFile** options. Make sure the port you choose is not in use by any other service. The PID file does not have to exist before running the service, it is generated automatically on service start.

3. Create a copy of the **systemd** unit file for the **sshd** service:

```
# cp /usr/lib/systemd/system/sshd.service /etc/systemd/system/sshd-second.service
```

4. Alter the created **sshd-second.service**:

   a. Modify the **Description** option:

```
Description=OpenSSH server second instance daemon
```

   b. Add **sshd.service** to services specified in the **After** option, so that the second instance starts only after the first one has already started:

```
After=syslog.target network.target auditd.service sshd.service
```

c. Remove the **ExecStartPre=/usr/sbin/sshd-keygen** line, the first instance of **sshd** includes key generation.

d. Add the **-f /etc/ssh/sshd-second_config** parameter to the **sshd** command, so that the alternative configuration file is used:

```
ExecStart=/usr/sbin/sshd -D -f /etc/ssh/sshd-second_config $OPTIONS
```

e. After the modifications, the **sshd-second.service** unit file contains the following settings:

```
[Unit]
Description=OpenSSH server second instance daemon
After=syslog.target network.target auditd.service sshd.service

[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D -f /etc/ssh/sshd-second_config $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

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

5. If using SELinux, add the port for the second instance of **sshd** to SSH ports, otherwise the second instance of **sshd** will be rejected to bind to the port:

```
# semanage port -a -t ssh_port_t -p tcp 22220
```

6. Enable **sshd-second.service** to start automatically on boot:

```
# systemctl enable sshd-second.service
```

7. Verify if the **sshd-second.service** is running by using the **systemctl status** command.

8. Verify if the port is enabled correctly by connecting to the service:

```
$ ssh -p 22220 user@server
```

Make sure you configure firewall to allow connections to the second instance of **sshd**.

1.9. FINDING THE SYSTEMD SERVICE DESCRIPTION

You can find descriptive information about the script on the line starting with **#description**. Use this description together with the service name in the **Description** option in the [Unit] section of the unit file. The header might contain similar data on the **#Short-Description** and **#Description** lines.

1.10. FINDING THE SYSTEMD SERVICE DEPENDENCIES

The Linux standard base (LSB) header might contain several directives that form dependencies between services. Most of them are translatable to systemd unit options, see the following table:

Table 1.5. Dependency options from the LSB header

LSB Option	Description	Unit File Equivalent
Provides	Specifies the boot facility name of the service, that can be referenced in other init scripts (with the "$" prefix). This is no longer needed as unit files refer to other units by their file names.	–
Required-Start	Contains boot facility names of required services. This is translated as an ordering dependency, boot facility names are replaced with unit file names of corresponding services or targets they belong to. For example, in case of **postfix**, the Required-Start dependency on $network was translated to the After dependency on network.target.	**After**, **Before**
Should-Start	Constitutes weaker dependencies than Required-Start. Failed Should-Start dependencies do not affect the service startup.	**After**, **Before**
Required-Stop, **Should-Stop**	Constitute negative dependencies.	**Conflicts**

1.11. FINDING DEFAULT TARGETS OF THE SERVICE

The line starting with **#chkconfig** contains three numerical values. The most important is the first number that represents the default runlevels in which the service is started. Map these runlevels to equivalent systemd targets. Then list these targets in the **WantedBy** option in the [Install] section of the unit file. For example, **postfix** was previously started in runlevels 2, 3, 4, and 5, which translates to multi-user.target and graphical.target. Note that the graphical.target depends on multiuser.target, therefore it is not necessary to specify both. You might find information about default and forbidden runlevels also at **#Default-Start** and **#Default-Stop** lines in the LSB header.

The other two values specified on the **#chkconfig** line represent startup and shutdown priorities of the init script. These values are interpreted by **systemd** if it loads the init script, but there is no unit file equivalent.

1.12. FINDING FILES USED BY THE SERVICE

Init scripts require loading a function library from a dedicated directory and allow importing configuration, environment, and PID files. Environment variables are specified on the line starting with **#config** in the init script header, which translates to the **EnvironmentFile** unit file option. The PID file specified on the **#pidfile** init script line is imported to the unit file with the **PIDFile** option.

The key information that is not included in the init script header is the path to the service executable, and potentially some other files required by the service. In previous versions of Red Hat Enterprise Linux, init scripts used a Bash case statement to define the behavior of the service on default actions, such as **start**, **stop**, or **restart**, as well as custom-defined actions. The following excerpt from the **postfix** init script shows the block of code to be executed at service start.

```
conf_check() {
    [ -x /usr/sbin/postfix ] || exit 5
    [ -d /etc/postfix ] || exit 6
    [ -d /var/spool/postfix ] || exit 5
}

make_aliasesdb() {
 if [ "$(/usr/sbin/postconf -h alias_database)" == "hash:/etc/aliases" ]
 then
  # /etc/aliases.db might be used by other MTA, make sure nothing
  # has touched it since our last newaliases call
  [ /etc/aliases -nt /etc/aliases.db ] ||
   [ "$ALIASESDB_STAMP" -nt /etc/aliases.db ] ||
   [ "$ALIASESDB_STAMP" -ot /etc/aliases.db ] || return
  /usr/bin/newaliases
  touch -r /etc/aliases.db "$ALIASESDB_STAMP"
 else
  /usr/bin/newaliases
 fi
}

start() {
 [ "$EUID" != "0" ] && exit 4
 # Check that networking is up.
 [ ${NETWORKING} = "no" ] && exit 1
 conf_check
 # Start daemons.
 echo -n $"Starting postfix: "
 make_aliasesdb >/dev/null 2>&1
 [ -x $CHROOT_UPDATE ] && $CHROOT_UPDATE
 /usr/sbin/postfix start 2>/dev/null 1>&2 && success || failure $"$prog start"
 RETVAL=$?
 [ $RETVAL -eq 0 ] && touch $lockfile
        echo
 return $RETVAL
}
```

The extensibility of the init script allowed specifying two custom functions, **conf_check()** and **make_aliasesdb()**, that are called from the **start()** function block. On closer look, several external files and directories are mentioned in the above code: the main service executable **/usr/sbin/postfix**, the **/etc/postfix/** and **/var/spool/postfix/** configuration directories, as well as the **/usr/sbin/postconf/** directory.

systemd supports only the predefined actions, but enables executing custom executables with **ExecStart**, **ExecStartPre**, **ExecStartPost**, **ExecStop**, and **ExecReload** options. The **/usr/sbin/postfix** together with supporting scripts are executed on service start. Converting complex init scripts requires understanding the purpose of every statement in the script. Some of the statements are specific to the operating system version, therefore you do not need to translate them. On the other hand, some adjustments might be needed in the new environment, both in unit file as well as in the service executable and supporting files.

1.13. MODIFYING EXISTING UNIT FILES

If you want to modify existing unit files proceed to the **/etc/systemd/system/** directory. Note that you should not modify the your the default unit files, which your system stores in the **/usr/lib/systemd/system/** directory.

Procedure

1. Depending on the extent of the required changes, pick one of the following approaches:
   - Create a directory for supplementary configuration files at **/etc/systemd/system/*<unit>*.d/**. This method is recommended for most use cases. You can extend the default configuration with additional functionality, while still referring to the original unit file. Changes to the default unit introduced with a package upgrade are therefore applied automatically. See Extending the default unit configuration for more information.
   - Create a copy of the original unit file from **/usr/lib/systemd/system/`directory in the `/etc/systemd/system/** directory and make changes there. The copy overrides the original file, therefore changes introduced with the package update are not applied. This method is useful for making significant unit changes that should persist regardless of package updates. See Overriding the default unit configuration for details.
2. To return to the default configuration of the unit, delete custom-created configuration files in the **/etc/systemd/system/** directory.
3. Apply changes to unit files without rebooting the system:

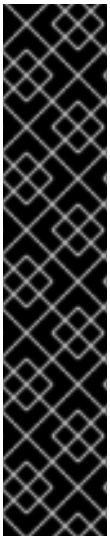
   ```
   # systemctl daemon-reload
   ```

   The **daemon-reload** option reloads all unit files and recreates the entire dependency tree, which is needed to immediately apply any change to a unit file. As an alternative, you can achieve the same result with the following command:

   ```
   # init q
   ```

4. If the modified unit file belongs to a running service, restart the service:

   ```
   # systemctl restart <name>.service
   ```



IMPORTANT

To modify properties, such as dependencies or timeouts, of a service that is handled by a SysV initscript, do not modify the initscript itself. Instead, create a **systemd** drop-in configuration file for the service as described in: Extending the default unit configuration and Overriding the default unit configuration .

Then manage this service in the same way as a normal **systemd** service.

For example, to extend the configuration of the **network** service, do not modify the **/etc/rc.d/init.d/network** initscript file. Instead, create new directory **/etc/systemd/system/network.service.d/** and a **systemd** drop-in file **/etc/systemd/system/network.service.d/*my_config*.conf**. Then, put the modified values into the drop-in file. Note: **systemd** knows the **network** service as **network.service**, which is why the created directory must be called **network.service.d**

1.14. EXTENDING THE DEFAULT UNIT CONFIGURATION

You can extend the default unit file with additional systemd configuration options.

Procedure

1. Create a configuration directory in **/etc/systemd/system/**:

   ```
   # mkdir /etc/systemd/system/<name>.service.d/
   ```

   Replace *<name>* with the name of the service you want to extend. The syntax applies to all unit types.

2. Create a configuration file with the **.conf** suffix:

   ```
   # touch /etc/systemd/system/name.service.d/<config_name>.conf
   ```

   Replace *<config_name>* with the name of the configuration file. This file adheres to the normal unit file structure and you have to specify all directives in the appropriate sections, see Unit file structure.

   For example, to add a custom dependency, create a configuration file with the following content:

   ```
   [Unit]
   Requires=<new_dependency>
   After=<new_dependency>
   ```

   The *<new_dependency>* stands for the unit to be marked as a dependency. Another example is a configuration file that restarts the service after its main process exited, with a delay of 30 seconds:

   ```
   [Service]
   Restart=always
   RestartSec=30
   ```

   Create small configuration files focused only on one task. Such files can be easily moved or linked to configuration directories of other services.

3. Apply changes to the unit:

   ```
   # systemctl daemon-reload
   # systemctl restart <name>.service
   ```

Example 1.1. Extending the httpd.service configuration

To modify the **httpd.service** unit so that a custom shell script is automatically executed when starting the Apache service, perform the following steps.

1. Create a directory and a custom configuration file:

   ```
   # mkdir /etc/systemd/system/httpd.service.d/
   ```

```
# touch /etc/systemd/system/httpd.service.d/custom_script.conf
```

2. Specify the script you want to execute after the main service process by inserting the following text to the **custom_script.conf** file:

```
[Service]
ExecStartPost=/usr/local/bin/custom.sh
```

3. Apply the unit changes::

```
# systemctl daemon-reload
```

```
# systemctl restart httpd.service
```

NOTE

The configuration files from the **/etc/systemd/system/** configuration directories take precedence over unit files in **/usr/lib/systemd/system/**. Therefore, if the configuration files contain an option that can be specified only once, such as **Description** or **ExecStart**, the default value of this option is overridden. Note that in the output of the **systemd-delta** command, described in Monitoring overridden units ,such units are always marked as [EXTENDED], even though in sum, certain options are actually overridden.

1.15. OVERRIDING THE DEFAULT UNIT CONFIGURATION

You can make changes to the unit file configuration that will persist after updating the package that provides the unit file.

Procedure

1. Copy the unit file to the **/etc/systemd/system/** directory by entering the following command as **root**:

```
# cp /usr/lib/systemd/system/<name>.service /etc/systemd/system/<name>.service
```

2. Open the copied file with a text editor, and make changes.

3. Apply unit changes:

```
# systemctl daemon-reload
# systemctl restart <name>.service
```

1.16. CHANGING THE TIMEOUT LIMIT

You can specify a timeout value per service to prevent a malfunctioning service from freezing the system. Otherwise, the default value for timeout is 90 seconds for normal services and 300 seconds for SysV-compatible services.

Procedure

To extend timeout limit for the **httpd** service:

1. Copy the **httpd** unit file to the **/etc/systemd/system/** directory:

```
# cp /usr/lib/systemd/system/httpd.service /etc/systemd/system/httpd.service
```

2. Open the **/etc/systemd/system/httpd.service** file and specify the **TimeoutStartUSec** value in the **[Service]** section:

```
...
[Service]
...
PrivateTmp=true
TimeoutStartSec=10

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

3. Reload the **systemd** daemon:

```
# systemctl daemon-reload
```

4. **Optional.** Verify the new timeout value:

```
# systemctl show httpd -p TimeoutStartUSec
```

NOTE

To change the timeout limit globally, input the **DefaultTimeoutStartSec** in the **/etc/systemd/system.conf** file.

1.17. MONITORING OVERRIDDEN UNITS

You can display an overview of overridden or modified unit files by using the **systemd-delta** command.

Procedure

- Display an overview of overridden or modified unit files:

```
# systemd-delta
```

For example, the output of the command can look as follows:

```
[EQUIVALENT] /etc/systemd/system/default.target → /usr/lib/systemd/system/default.target
[OVERRIDDEN] /etc/systemd/system/autofs.service →
/usr/lib/systemd/system/autofs.service

--- /usr/lib/systemd/system/autofs.service       2014-10-16 21:30:39.000000000 -0400
+++ /etc/systemd/system/autofs.service  2014-11-21 10:00:58.513568275 -0500
@@ -8,7 +8,8 @@
 EnvironmentFile=-/etc/sysconfig/autofs
 ExecStart=/usr/sbin/automount $OPTIONS --pid-file /run/autofs.pid
 ExecReload=/usr/bin/kill -HUP $MAINPID
-TimeoutSec=180
```

```
+TimeoutSec=240
+Restart=Always

 [Install]
 WantedBy=multi-user.target

[MASKED]     /etc/systemd/system/cups.service → /usr/lib/systemd/system/cups.service
[EXTENDED]   /usr/lib/systemd/system/sssd.service →
/etc/systemd/system/sssd.service.d/journal.conf

4 overridden configuration files found.
```

1.18. WORKING WITH INSTANTIATED UNITS

You can manage multiple instances of a service by using a single template configuration. You can define a generic template for a unit and generate multiple instances of that unit with specific parameters at runtime. The template is indicated by the at sign (@). Instantiated units can be started from another unit file (using **Requires** or **Wants** options), or with the **systemctl start** command. Instantiated service units are named the following way:

```
<template_name>@<instance_name>.service
```

The *<template_name>* stands for the name of the template configuration file. Replace *<instance_name>* with the name for the unit instance. Several instances can point to the same template file with configuration options common for all instances of the unit. Template unit name has the form of:

```
<unit_name>@.service
```

For example, the following **Wants** setting in a unit file:

```
Wants=getty@ttyA.service getty@ttyB.service
```

first makes systemd search for given service units. If no such units are found, the part between "@" and the type suffix is ignored and **systemd** searches for the **getty@.service** file, reads the configuration from it, and starts the services.

For example, the **getty@.service** template contains the following directives:

```
[Unit]
Description=Getty on %I
...
[Service]
ExecStart=-/sbin/agetty --noclear %I $TERM
...
```

When the **getty@ttyA.service** and **getty@ttyB.service** are instantiated from the above template, **Description**= is resolved as **Getty on ttyA** and **Getty on ttyB**.

1.19. IMPORTANT UNIT SPECIFIERS

You can use the wildcard characters, called **unit specifiers**, in any unit configuration file. Unit specifiers substitute certain unit parameters and are interpreted at runtime.

Table 1.6. Important unit specifiers

Unit Specifier	Meaning	Description
%n	Full unit name	Stands for the full unit name including the type suffix. **%N** has the same meaning but also replaces the forbidden characters with ASCII codes.
%p	Prefix name	Stands for a unit name with type suffix removed. For instantiated units %p stands for the part of the unit name before the "@" character.
%i	Instance name	Is the part of the instantiated unit name between the "@" character and the type suffix. **%I** has the same meaning but also replaces the forbidden characters for ASCII codes.
%H	Host name	Stands for the hostname of the running system at the point in time the unit configuration is loaded.
%t	Runtime directory	Represents the runtime directory, which is either **/run** for the **root** user, or the value of the XDG_RUNTIME_DIR variable for unprivileged users.

For a complete list of unit specifiers, see the **systemd.unit(5)** manual page.

1.20. ADDITIONAL RESOURCES

- How to set limits for services in RHEL and systemd
- How to write a service unit file which enforces that particular services have to be started
- How to decide what dependencies a systemd service unit definition should have

CHAPTER 2. OPTIMIZING SYSTEMD TO SHORTEN THE BOOT TIME

As a system administrator, you can optimize performance of your system and shorten the boot time. You can review the services that **systemd** starts during boot and evaluate their necessity. Disabling certain services to start at boot can improve the boot time of your system.

2.1. EXAMINING SYSTEM BOOT PERFORMANCE

To examine system boot performance, you can use the **systemd-analyze** command. By using certain options, you can tune systemd to shorten the boot time.

Prerequisites

- Optional: Before you examine **systemd** to tune the boot time, list all enabled services:

```
$ systemctl list-unit-files --state=enabled
```

Procedure

Choose the information you want to analyze:

- Analyze the information about the time that the last successful boot took:

```
$ systemd-analyze
```

- Analyze the unit initialization time of each **systemd** unit:

```
$ systemd-analyze blame
```

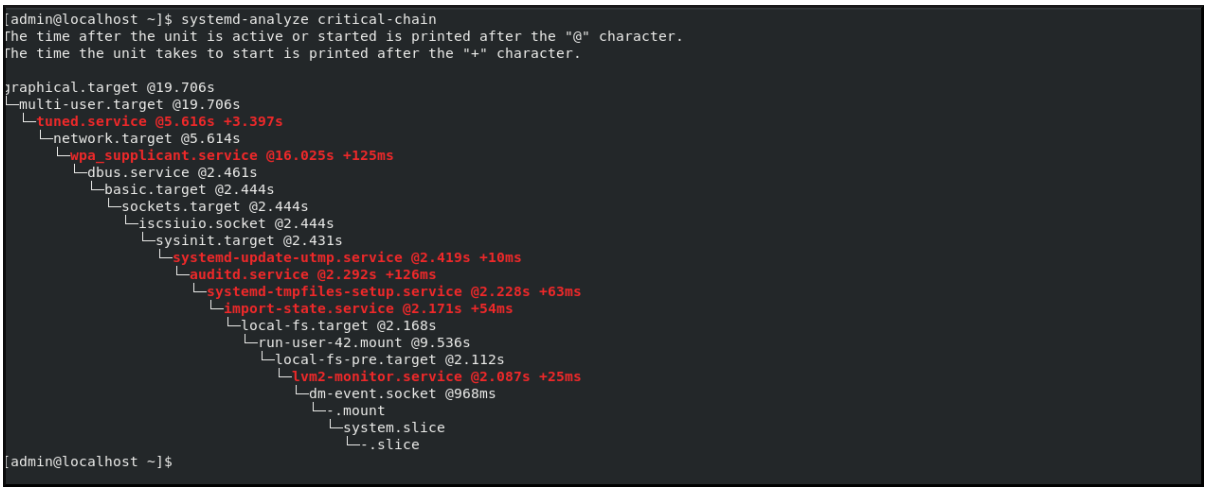
  The output lists the units in descending order according to the time they took to initialize during the last successful boot.

- Identify critical units that took the longest time to initialize at the last successful boot:

```
$ systemd-analyze critical-chain
```

  The output highlights the units that critically slow down the boot with the red color.

Figure 2.1. The output of the systemd-analyze critical-chain command



Additional resources

- **systemd-analyze (1)** man page

2.2. A GUIDE TO SELECTING SERVICES THAT CAN BE SAFELY DISABLED

You can shorten the boot time of your system by disabling certain services that are enabled on boot by default.

- List enabled services:

```
$ systemctl list-unit-files --state=enabled
```

- Disable a service:

```
# systemctl disable <service_name>
```

Certain services must stay enabled so that your operating system is safe and functions in the way you need.

Refer to the following table as a guide to selecting the services that you can safely disable. The table lists all services enabled by default on a minimal installation of Red Hat Enterprise Linux.

Table 2.1. Services enabled by default on a minimal installation of RHEL

Service name	Can it be disabled?	More information

Service name	Can it be disabled?	More information
auditd.service	yes	Disable **auditd.service** only if you do not need audit messages from the kernel. Be aware that if you disable **auditd.service**, the **/var/log/audit/audit.log** file is not produced. Consequently, you are not able to retroactively review some commonly-reviewed actions or events, such as user logins, service starts or password changes. Also note that auditd has two parts: a kernel part, and a service itself. By using the **systemctl disable auditd** command, you only disable the service, but not the kernel part. To disable system auditing in its entirety, set **audit=0** on kernel command line.
autovt@.service	no	This service runs only when it is really needed, so it does not need to be disabled.
crond.service	yes	Be aware that no items from crontab will run if you disable crond.service.
dbus-org.fedoraproject.FirewallD1.service	yes	A symlink to **firewalld.service**
dbus-org.freedesktop.NetworkManager.service	yes	A symlink to **NetworkManager.service**
dbus-org.freedesktop.nm-dispatcher.service	yes	A symlink to **NetworkManager-dispatcher.service**
firewalld.service	yes	Disable **firewalld.service** only if you do not need firewall.
getty@.service	no	This service runs only when it is really needed, so it does not need to be disabled.
import-state.service	yes	Disable **import-state.service** only if you do not need to boot from a network storage.
irqbalance.service	yes	Disable **irqbalance.service** only if you have just one CPU. Do not disable **irqbalance.service** on systems with multiple CPUs.
kdump.service	yes	Disable **kdump.service** only if you do not need reports from kernel crashes.

Service name	Can it be disabled?	More information
loadmodules.service	yes	This service is not started unless the **/etc/rc.modules** or **/etc/sysconfig/modules** directory exists, which means that it is not started on a minimal RHEL installation.
lvm2-monitor.service	yes	Disable **lvm2-monitor.service** only if you do not use Logical Volume Manager (LVM).
microcode.service	no	Do not be disable the service because it provides updates of the microcode software in CPU.
NetworkManager-dispatcher.service	yes	Disable **NetworkManager-dispatcher.service** only if you do not need notifications on network configuration changes (for example in static networks).
NetworkManager-wait-online.service	yes	Disable **NetworkManager-wait-online.service** only if you do not need working network connection available right after the boot. If the service is enabled, the system does not finish the boot before the network connection is working. This may prolong the boot time significantly.
NetworkManager.service	yes	Disable **NetworkManager.service** only if you do not need connection to a network.
nis-domainname.service	yes	Disable **nis-domainname.service** only if you do not use Network Information Service (NIS).
rhsmcertd.service	no	
rngd.service	yes	Disable **rngd.service** only if you do not need much entropy on your system, or you do not have any sort of hardware generator. Note that the service is necessary in environments that require a lot of good entropy, such as systems used for generation of X.509 certificates (for example the FreeIPA server).
rsyslog.service	yes	Disable **rsyslog.service** only if you do not need persistent logs, or you set **systemd-journald** to persistent mode.
selinux-autorelabel-mark.service	yes	Disable **selinux-autorelabel-mark.service** only if you do not use SELinux.
sshd.service	yes	Disable **sshd.service** only if you do not need remote logins by OpenSSH server.

Service name	Can it be disabled?	More information
sssd.service	yes	Disable **sssd.service** only if there are no users who log in the system over the network (for example by using LDAP or Kerberos). Red Hat recommends to disable all **sssd-*** units if you disable **sssd.service**.
syslog.service	yes	An alias for **rsyslog.service**
tuned.service	yes	Disable **tuned.service** only if you do need to use performance tuning.
lvm2-lvmpolld.socket	yes	Disable **lvm2-lvmpolld.socket** only if you do not use Logical Volume Manager (LVM).
dnf-makecache.timer	yes	Disable **dnf-makecache.timer** only if you do not need your package metadata to be updated automatically.
unbound-anchor.timer	yes	Disable **unbound-anchor.timer** only if you do not need daily update of the root trust anchor for DNS Security Extensions (DNSSEC). This root trust anchor is used by Unbound resolver and resolver library for DNSSEC validation.

To find more information about a service, use one of the following commands:

```
$ systemctl cat <service_name>
```

```
$ systemctl help <service_name>
```

The **systemctl cat** command provides the content of the respective **/usr/lib/systemd/system/*<service>*** service file, as well as all applicable overrides. The applicable overrides include unit file overrides from the **/etc/systemd/system/*<service>*** file or drop-in files from a corresponding **unit.type.d** directory.

Additional resources

- The **systemd.unit(5)** man page
- The **systemd help** command that shows the man page of a particular service

2.3. ADDITIONAL RESOURCES

- **systemctl**(1) man page
- **systemd**(1) man page
- **systemd-delta**(1) man page
- **systemd.directives**(7) man page

- **systemd.unit**(5) man page
- **systemd.service**(5) man page
- **systemd.target**(5) man page
- **systemd.kill**(5) man page
- systemd Home Page