# Overview of systemd for RHEL 7

Actualizado27 de Agosto de 2015 a las 13:11 - English

The systemd system and service manager is responsible for controlling how services are started, stopped and otherwise managed on Red Hat Enterprise Linux 7 systems. By offering on-demand service start-up and better transactional dependency controls, systemd dramatically reduces start up times. As a systemd user, you can prioritize critical services over less important services.

Although the systemd process replaces the init process (quite literally, /sbin/init is now a symbolic link to /usr/lib/systemd/systemd) for starting services at boot time and changing runlevels, systemd provides much more control than the init process does while still supporting existing init scripts. Here are some examples of the features of systemd:

- **Logging**: From the moment that the initial RAM disk is mounted to start the Linux kernel to final shutdown of the system, all log messages are stored by the new systemd journal. Before the systemd journal existed, initial boot messages were lost, requiring that you try to watch the screen as messages scrolled by to debug boot problems.
  Now, all system messages come in on a single stream and are stored in the **/run** directory. Messages can then be consumed by the rsyslog facility (and redirected to traditional log files in the **/var/log** directory or to remote log servers) or displayed using the **journalctl** command across a variety of attributes.

- **Dependencies**: With systemd, an explicit set of dependencies can be defined for each service, instead of being implied by boot order. This allows a service to start at any point that its dependencies are met. In this way, many services can start at the same time, making the boot process faster. Likewise, complex sets of dependencies can be set up, so the exact requirements of a service (such as storage availability or file system checking) can be met before a service starts.

- **Cgroups**: Services are identified by Cgroups, which allow every component of a service to be managed. For example, the older System V init scripts would start a service by launching a process which itself might start other child processes. When the service was killed, it was hoped that the parent process would do the right thing and kill its children. By using Cgroups, all components of a service have a tag that can be used to make sure that all of those components are properly started or stopped.

- **Activating services**: Services don't just have to be always running or not running based on runlevel, as they were previous to systemd. Services can now be activated based on path, socket, bus, timer, or hardware activation. Likewise, because systemd can set up sockets, if a process handling communications goes away, the process that starts up in its place can pick up the next message from the socket. To the clients using the service, it can look as though the service continued without interruption.

- **More than services**: Instead of just managing services, systemd can manage several different unit types. These unit types include:
  - **Devices**: Create and use devices.
  - **Mounts and automounts**: Mount file systems upon request or automount a file system based on a request for a file or directory within that file system.
  - **Paths**: Check the existence of files or directories or create them as needed.
  - **Services**: Start a service, which often means launching a service daemon and related components.

  - **Services**: Start a service, which often means launching a service daemon and related components.
  - **Slices**: Divide up computer resources (such as CPU and memory) and apply them to selected units.
  - **Snapshots**: Take snapshots of the current state of the system.
  - **Sockets**: Set up sockets to allow communication paths to processes that can remain in place, even if the underlying process needs to restart.
  - **Swaps**: Create and use swap files or swap partitions.
  - **Targets**: Manage a set of services under a single unit, represented by a target name rather than a runlevel number.
  - **Timers**: Trigger actions based on a timer.

- **Resource management**

  - The fact that each systemd unit is always associated with its own cgroup lets you control the amount of resources each service can use. For example, you can set a percent of CPU usage by service which can put a cap on the total amount of CPU that service can use -- in other words, spinning off more processes won't allow more resources to be consumed by the service. Prior to systemd, nice levels were often used to prevent processes from hogging precious CPU time. With systemd's use of cgroups, precise limits can be set on CPU and memory usage, as well as other resources.
  - A feature called slices lets you slice up many different types of system resources and assign them to users, services, virtual machines, and other units. Accounting is also done on these resources, which can allow you to charge customers for their resource usage.

# Booting RHEL 7 with systemd

When you boot a standard X86 computer to run RHEL 7, the BIOS boots from the selected medium (usually a local hard disk) and the boot loader (GRUB2 for RHEL 7) starts the RHEL 7 kernel and initial RAM disk. After that, the systemd process takes over to initialize the system and start all the system services.

Although there is not a strict order in which services are started when a RHEL 7 (systemd) system is booted, there is a structure to the boot process. The direction that the systemd process takes at boot time depends on the **default.target** file. A long listing of the **default.target** file shows you which target starts when the system boots:

```
# cd /etc/systemd/system # ls -l default.target lrwxrwxrwx. 1 root root 16 Aug 23 19:18
default.target -> /lib/systemd/system/graphical.target
```

You can see here that the **graphical.target** (common for desktop systems or servers with graphical interfaces) is set as the **default.target** (via a symbolic link). To understand what targets, services and other units start up with the graphical target, it helps to work backwards, as systemd does, to build the dependency tree. Here's what to look for:

- **graphical.target**: The **/lib/systemd/system/graphical.target** file includes these lines:

  ```
  Requires=multi-user.target Wants=display-manager.service Conflicts=rescue.service
  rescue.target After=multi-user.target rescue.service rescue.target display-
  manager.service AllowIsolate=yes
  ```

  This tells systemd to start everything in the multi-user.target before starting the graphical target. Once that's done, the "Wants" entry tells systemd to start the **display-**

This tells systemd to start everything in the multi-user.target before starting the graphical target. Once that's done, the "Wants" entry tells systemd to start the **display-manager.service** service (**/etc/systemd/system/display-manager.service**), which runs the GNOME display manager (**/usr/sbin/gdm**).

- **multi-user.target**: The **/usr/lib/systemd/system/multi-user.target** starts the services you would expect in a RHEL multi-user mode. The file contains the following line:

```
Requires=basic.target
```

  This tells systemd to start everything in the **/usr/lib/systemd/system/basic.target** target before starting the other multi-user services. After that, for the multi-user.target, all units (services, targets, etc.) in the **/etc/systemd/system/multi-user.target.wants** and **/usr/lib/systemd/system/multi-user.target.wants** directories are started. When you enable a service, a symbolic link is placed in the **/etc/systemd/system/multi-user.target.wants** directory. That directory is where you will find links to most of the services you think of as starting in multi-user mode (printing, cron, auditing, SSH, and so on). Here is an example of the services, paths, and targets in a typical **multi-user.target.wants** directory:

```
# cd /etc/systemd/system/multi-user.target.wants abrt-ccpp.service
hypervkvpd.service postfix.service abrtd.service hypervvssd.service remote-
fs.target abrt-oops.service irqbalance.service rhsmcertd.service
abrt-vmcore.service ksm.service rngd.service abrt-xorg.service ksmtuned.service
rpcbind.service atd.service libstoragemgmt.service rsyslog.service auditd.service
libvirtd.service smartd.service avahi-daemon.service mdmonitor.service
sshd.service chronyd.service ModemManager.service sysstat.service crond.service
netcf-transaction.service tuned.service cups.path nfs.target vmtoolsd.service
```

- **basic.target**: The **/usr/lib/systemd/system/basic.target** file starts the basic services associated with all running RHEL 7 systems. The file contains the following line:

```
Requires=sysinit.target
```

  This points systemd to the **/usr/lib/systemd/system/sysinit.target**, which must start before the **basic.target** can continue. The **basic.target** target file starts the firewalld and microcode services from the **/etc/systemd/system/basic.target.wants** directory and services for SELinux, kernel messages, and loading modules from the **/usr/lib/systemd/system/basic.target.wants** directory.

- **sysinit.target**: The **/usr/lib/systemd/system/sysinit.target** file starts system initialization services, such as mounting file systems and enabling swap devices. The file contains the following line:

```
Wants=local-fs.target swap.target
```

  Besides mounting file systems and enabling swap devices, the sysinit.target starts targets, services, and mounts based on units contained in the **/usr/lib/systemd/system/sysinit.target.wants** directory. These units enable logging, set kernel options, start the udevd daemon to detect hardware, and allow file system decryption, among other things. The **/etc/systemd/system/sysinit.target.wants** directory contains services that start iSCSI, multipath, LVM monitoring and RAID services.

udevd daemon to detect hardware, and allow file system decryption, among other things. The **/etc/systemd/system/sysinit.target.wants** directory contains services that start iSCSI, multipath, LVM monitoring and RAID services.

- **local-fs.target**: The local-fs.target is set to run after the local-fs-pre.target target, based on this line:

  ```
  After=local-fs-pre.target
  ```

  There are no services associated with the local-fs-pre.target target (you could add some to a "wants" directory if you like). However, units in the **/usr/lib/systemd/system/local-fs.target.wants** directory import the network configuration from the initramfs, run a file system check (fsck) on the root file system when necessary, and remounting the root file system (and special kernel file systems) based on the contents of the /etc/fstab file.

Although the boot process is built by systemd in the order just shown, it actually runs, in general, in the opposite order. As a rule, a target on which another target is dependent must be running before the units in the first target can start. To see more details about the boot process, see the bootup man page (**man 7 bootup**).

# Using the systemctl Command

The most important command for managing services on a RHEL 7 (systemd) system is the **systemctl** command. Here are some examples of the **systemctl** command (using the nfs-server service as an example) and a few other commands that you may find useful:

- **Checking service status**: To check the status of a service (for example, nfs-server.service), type the following:

  ```
  # systemctl status nfs-server.service nfs-server.service - NFS Server Loaded:
  loaded (/usr/lib/systemd/system/nfs-server.service; disabled) Active: active
  (exited) since Wed 2014-03-19 10:29:40 MDT; 57s ago Process: 5206
  ExecStartPost=/usr/libexec/nfs-utils/scripts/nfs-server.postconfig (code=exited,
  status=0/SUCCESS) Process: 5191 ExecStart=/usr/sbin/rpc.nfsd $RPCNFSDARGS
  $RPCNFSDCOUNT (code=exited, status=0/SUCCESS) Process: 5188 ExecStartPre=/usr
  /sbin/exportfs -r (code=exited, status=0/SUCCESS) Process: 5187 ExecStartPre=/usr
  /libexec/nfs-utils/scripts/nfs-server.preconfig (code=exited, status=0/SUCCESS)
  Main PID: 5191 (code=exited, status=0/SUCCESS) CGroup: /system.slice
  /nfs-server.service Mar 19 10:29:40 localhost.localdomain systemd[1]: Starting
  NFS Server... Mar 19 10:29:40 localhost.localdomain systemd[1]: Started NFS
  Server.
  ```

- **Stopping a service**: To stop a service, use the stop option as follows:

  ```
  # systemctl stop nfs-server.service
  ```

- **Starting a service**: To start a service, use the start option as follows:

  ```
  # systemctl start nfs-server.service
  ```

- **Enabling a service**: To enable a service so it starts automatically at boot time, type the following:

- **Enabling a service**: To enable a service so it starts automatically at boot time, type the following:

```
# systemctl enable nfs-server.service
```

- **Disable a service**: To disable a service so it doesn't start automatically at boot time, type the following:

```
# systemctl disable nfs-server.service
```

- **Listing dependencies**: To see dependencies of a service, use the list-dependencies option, as follows:

```
# systemctl list-dependencies nfs-server.service nfs-server.service ├─nfs-idmap.service ├─nfs-mountd.service ├─nfs-rquotad.service ├─proc-fs-nfsd.mount ├─rpcbind.service ├─system.slice ├─var-lib-nfs-rpc_pipefs.mount └─basic.target ├─alsa-restore.service ├─alsa-state.service ...
```

- **Listing units in targets**: To see what services and other units (service, mount, path, socket, and so on) are associated with a particular target, type the following:

```
# systemctl list-dependencies multi-user.target multi-user.target ├─abrt-ccpp.service ├─abrt-oops.service ├─abrt-vmcore.service ├─abrt-xorg.service ├─abrtd.service ├─atd.service ├─auditd.service ├─avahi-daemon.service ├─brandbot.path ├─chronyd.service ├─crond.service ...
```

- **List specific types of units**: Use the following command to list specific types of units (in these examples, service and mount unit types):

```
# systemctl list-units --type service UNIT LOAD ACTIVE SUB DESCRIPTION abrt-ccpp.service loaded active exited Install ABRT coredump hook abrt-oops.service loaded active running ABRT kernel log watcher abrt-xorg.service loaded active running ABRT Xorg log watcher abrtd.service loaded active running ABRT Automated Bug Reporting accounts-daemon.service loaded active running Accounts Service ... # systemctl list-units --type mount UNIT LOAD ACTIVE SUB DESCRIPTION -.mount loaded active mounted / boot.mount loaded active mounted /boot dev-hugepages.mount loaded active mounted Huge Pages File System dev-mqueue.mount loaded active mounted POSIX Message Queue File Syst mnt-repo.mount loaded active mounted /mnt/repo proc-fs-nfsd.mount loaded active mounted RPC Pipe File System run-user-1000-gvfs.mount loaded active mounted /run/user/1000/gvfs ...
```

- **Listing all units**: To list all units installed on the system, along with their current states, type the following:

```
# systemctl list-unit-files UNIT FILE STATE proc-sys-fs-binfmt_misc.automount static dev-hugepages.mount static dev-mqueue.mount static proc-sys-fs-binfmt_misc.mount static ... arp-ethers.service disabled atd.service enabled auditd.service enabled ...
```

- **View service processes with systemd-cgtop**: To view processes associated with a particular service (cgroup), you can use the **systemd-cgtop** command. Like the **top** command (which sorts processes by such things as CPU and memory usage), **systemd-cgtop** lists running processes based on their service (cgroup label). Once command (which sorts processes by such things as CPU and memory usage), **systemd-cgtop** lists running processes based on their service (cgroup label). Once systemd-cgtop is running, you can press keys to sort by memory (m), CPU (c), task (t), path (p), or I/O load (i). Here is an example:

```
# systemd-cgtop
```

- **Recursively view cgroup contents**: To output a recursive list of cgroup content, use the systemd-cgls command:

```
# systemd-cgls ├─user.slice │ ├─user-1000.slice │ │ ├─session-5.scope │ │ │
├─2661 gdm-session-worker [pam/gdm-password] │ │ │ ├─2672 /usr/bin/gnome-keyring-
daemon --daemonize --login │ │ │ ├─2674 gnome-session --session gnome-classic │ │
│ ├─2682 dbus-launch --sh-syntax --exit-with-session │ │ │ ├─2683 /bin/dbus-
daemon --fork --print-pid 4 --print-address 6 --session │ │ │ ├─2748 /usr/libexec
/gvfsd ...
```

- **View journal (log) files**: Using the **journalctl** command you can view messages from the systemd journal. Using different options you can select which group of messages to display. The **journalctl** command also supports tab completion to fill in fields for which to search. Here are some examples:

```
# journalctl -h View help for the command # journalctl -k View kernel messages
from current boot # journalctl -f Follow journal messages (like tail -f) #
journalctl -u NetworkManager View messages for specific unit (can tab complete)
```

# Comparing systemd to Traditional init

Some of the benefits of systemd over the traditional System V init facility include:

- systemd never loses initial log messages
- systemd can respawn daemons as needed
- systemd records runtime data (i.e., captures stdout/stderr of processes)
- systemd doesn't lose daemon context during runtime
- systemd can kill all components of a service cleanly

Here are some details of how systemd compares to pre-RHEL 7 init and related commands:

- **System startup**: The systemd process is the first process ID (PID 1) to run on RHEL 7 system. It initializes the system and launches all the services that were once started by the traditional **init** process.

- **Managing system services**: For RHEL 7, the **systemctl** command replaces **service** and **chkconfig**. Prior to RHEL 7, once RHEL was up and running, the **service** command was used to start and stop services immediately. The **chkconfig** command was used to identify at which run levels a service would start or stop automatically.
  Although you can still use the **service** and **chkconfig** commands to start/stop and enable/disable services, respectively, they are not 100% compatible with the RHEL 7 systemctl command. For example, non-standard service options, such as those that start databases or check configuration files, may not be supported in the same way for RHEL 7

enable/disable services, respectively, they are not 100% compatible with the RHEL 7 systemctl command. For example, non-standard service options, such as those that start databases or check configuration files, may not be supported in the same way for RHEL 7 services.

- **Changing runlevels**: Prior to RHEL 7, runlevels were used to identify a set of services that would start or stop when that runlevel was requested. Instead of runlevels, systemd uses the concept of targets to group together sets of services that are started or stopped. A target can also include other targets (for example, the multi-user target includes an nfs target).
  There are systemd targets that align with the earlier runlevels. However the point of targets is not to necessarily imply a level of activity (for example, runlevel 3 implied more services were active than runlevel 1). Instead targets just represent a group of services, so it's appropriate that there are many more targets available than there are runlevels. The following list shows how systemd targets align with traditional runlevels:

```
Traditional runlevel New target name Symbolically linked to... Runlevel 0 |
runlevel0.target -> poweroff.target Runlevel 1 | runlevel1.target ->
rescue.target Runlevel 2 | runlevel2.target -> multi-user.target Runlevel 3 |
runlevel3.target -> multi-user.target Runlevel 4 | runlevel4.target -> multi-
user.target Runlevel 5 | runlevel5.target -> graphical.target Runlevel 6 |
runlevel6.target -> reboot.target
```

- **Default runlevel**: The default runlevel (previously set in the **/etc/inittab** file) is now replaced by a default target. The location of the default target is **/etc/systemd/system /default.target**, which by default is linked to the multi-user target.

- **Location of services**: Before systemd, services were stored as scripts in the **/etc/init.d** directory, then linked to different runlevel directories (such as **/etc/rc3.d**, **/etc/rc5.d**, and so on). Services with systemd are named something.**service**, such as firewalld.service, and are stored in **/lib/systemd/system** and **/etc/systemd/system** directories. Think of the **/lib** files as being more permanent and the **/etc** files as the place you can modify configurations as needed.
  When you enable a service in RHEL 7, the service file is linked to a file in the **/etc/systemd /system/multi-user.target.wants** directory. For example, if you run **systemctl enable fcoe.service** a symbolic link is created from **/etc/systemd/system/multi-user.target.wants/fcoe.service** that points to **/lib/systemd/system/fcoe.service** to cause the fcoe.service to start at boot time.
  Also, the older System V init scripts were actual shell scripts. The systemd files tasked to do the same job are more like .ini files that contain the information needed to launch a service.

- **Configuration files**: The **/etc/inittab** file was used by the init process in RHEL 6 and earlier to point to the initialization files (such as **/etc/rc.sysinit**) and runlevel service directories (such as **/etc/rc5.d**) needed to start up the system. Changes to those services was done in files (usually named after the service) in the **/etc/sysconfig** directory. For systemd in RHEL 7, there are still files in **/etc/sysconfig** used to modify how services behave. However, services can be modified by adding files to the **/etc/systemd** directory to override the permanent service files in the **/lib/systemd** directories.

# Transitioning to systemd

If you are used to using the **init** process and System V init scripts prior to RHEL 7, there are a few things you should know about transitioning to systemd:

If you are used to using the **init** process and System V init scripts prior to RHEL 7, there are a few things you should know about transitioning to systemd:

- **Using RHEL 6 commands**: For the time being, you can use commands such as **service**, **chkconfig**, **runlevel**, and **init** as you did in RHEL 6. They will cause appropriate systemd commands to run, with similar, if not exactly the same, results. Here are some examples:

```
# service cups restart Redirecting to /bin/systemctl restart cups.service #
chkconfig cups on Note: Forwarding request to 'systemctl enable cups.service'.
```

- **System V init Scripts**: Although not encouraged, System V init scripts are still supported. There are still some services in RHEL 7 that are implemented in System V init scripts. To see System V init scripts that are available on your system and the runlevels on which they start, use the **chkconfig** command as follows:

```
# chkconfig --list ... iprdump 0:off 1:off 2:on 3:on 4:on 5:on 6:off iprinit
0:off 1:off 2:on 3:on 4:on 5:on 6:off iprupdate 0:off 1:off 2:on 3:on 4:on 5:on
6:off netconsole 0:off 1:off 2:off 3:off 4:off 5:on 6:off network 0:off 1:off
2:on 3:on 4:on 5:on 6:off rhnsd 0:off 1:off 2:on 3:on 4:on 5:on 6:off ...
```

Using **chkconfig**, however, will not show you the whole list of services on your system. To see the systemd-specific services, run the **systemctl list-unit-files** command, as described earlier.

# More information on systemd

For more information on systemd, refer to the Red Hat Enterprise Linux 7 System Administrator's Guide