

RHEL 7 Update systemd

July 2014

OVERVIEW

- RHEL 7.0 will ship with systemd, a new init system that replaces upstart.
- But systemd is more than a SysVinit/upstart replacement
- It is a system and service manager for Linux.
- It can work as a drop-in replacement for sysvinit.
- It replaces inetd and xinetd for most scenarios

```
# ps --pid 1
PID TTY          TIME CMD
  1 ?             00:00:01 systemd
```

Key Concepts

- UNITS:
 - Services, Sockets,
 - Devices, Mounts, Automounts, Swaps
 - Timers, Paths,
 - Targets, Snapshots
- Slices
 - Unit/Service Dependency Tracking
 - Process tracking with Service information

Benefits

- Dependency tracking for units and processes
- No more (sleep 60; do something) loops
- Properly kill daemons
- Minimal boot times
- Debugging – no early boot messages are lost
- Easy to learn and backwards compatible.
- Autospawn and Respawn for Services
- Tight integration with cgroups, the default interface in the future

Systemd - Units

- Naming convention is: name.type
 - `httpd.service`, `sshd.socket`, or `dev-hugepages.mount`
- Service – Describe a daemon's type, execution, environment, and how it's monitored.
- Socket – Endpoint for interprocess communication. File, network, or Unix sockets.
- Target – Logical grouping of units. Replacement for runlevels.
- Device – Automatically created by the kernel. Can be provided to services as dependents.
- Mounts, automounts, swap – Monitor the mounting/unmounting of file systems.

Systemd - Units

- Snapshots – save the state of units – useful for testing
- Timers – Timer-based activation
- Paths – Uses inotify to monitor a path
- Slices – cgroup hierarchy for resource management.
- Scopes – Organizational units that groups services' worker processes.

Systemd – Dependency Resolution

- Example:
 - Wait for block device
 - Check file system for device
 - Mount file system
- `nfs-lock.service`:
 - `Requires=rpcbind.service network.target`
 - `After=network.target named.service rpcbind.service`
 - `Before=remote-fs-pre.target`

What about my System-V init scripts?

- systemd maintains 99% backwards compatibility with initscripts and the exceptions are well documented.
- While we do encourage everyone to convert legacy scripts to service unit files, it's not a requirement.
 - Hint: we'll show you how to do this in a few minutes.
- Incompatibilities are listed here:
<http://www.freedesktop.org/wiki/Software/systemd/Incompatibilities/>
- Converting SysV Init Scripts:
<http://0pointer.de/blog/projects/systemd-for-admins-3.html>

Faster Boot times

- Lennart Poettering says that “Fast booting isn't the goal of systemd, it's a result of a well designed system.”
- As virt/cloud demand continues, the desire for light-weight, reliable/resilient, and fast images grows.
 - A stripped down image can boot in ~2 seconds.
 - Less CPU cycles burned during the boot process
 - Important for highly dense and dynamic environments.
 - Even more important for containers.

The Basics: Managing Services

Managing Services - Unit Files

- Via Init:
 - Init scripts are stored in `/etc/init.d` & called from `/etc/rc*`
- Via systemd:
 - Maintainer files: `/usr/lib/systemd/system/`
 - User modifications: `/etc/systemd/system/`
- Note: unit files under `/etc/` will take precedence over `/usr`

Managing Services - Start/Stop

- Via Init:
 - `$ service httpd {start,stop,restart,reload}`
- Via systemctl:
 - `$ systemctl {start,stop,restart,reload} httpd.service`
- Notes:
 - systemctl places the “action” before the service name.
 - If a unit isn't specified, .service is assumed.
 - `systemctl start httpd == systemctl start httpd.service`
 - Tab completion works great with systemctl, install bash-completion
 - systemctl can connect to remote hosts over SSH using “-H”

Managing Services - Status

- Via Init:
 - `$ service httpd status`
- Via systemctl:
 - `$ systemctl status httpd.service`
- List loaded services:
 - `systemctl -t service`
- List installed services:
 - `systemctl list-unit-files -t service` (similar to `chkconfig --list`)
- View state:
 - `systemctl --state failed`

Managing Services - Enable/Disable

- Via Init:
 - `$ chkconfig httpd {on,off}`
- Via systemctl:
 - `$ systemctl {enable, disable, mask, unmask} httpd.service`
- mask – “This will link these units to /dev/null, making it impossible to start them. This is a stronger version of disable, since it prohibits all kinds of activation of the unit, including manual activation. Use this option with care.”

Runlevels...

gone.

What Runlevels?

- Runlevels == Targets
- “Runlevels” are exposed via target units
- /etc/inittab is no longer used
- Target names are more relevant:
 - multi-user.target vs. runlevel3
 - graphical.target vs. runlevel5
- Set the default via: ``systemctl enable graphical.target --force``
- Change at run-time via: ``systemctl isolate [target]``

Runlevel Names

Runlevel	Systemd Target	Description
0	poweroff.target, runlevel0.target	System halt
1	rescue.target, runlevel1.target	Single user mode
3 (2,4)	multi-user.target, runlevel3.target	Multi-user, non graphical
5	graphical.target, runlevel5.target	Multi-user, graphical
6	reboot.target, runlevel6.target	System reboot

```
ls /lib/systemd/system/runlevel*target -l
lrwxrwxrwx. 1 root root 15 Jul 3 21:37 /lib/systemd/system/runlevel0.target -> poweroff.target
lrwxrwxrwx. 1 root root 13 Jul 3 21:37 /lib/systemd/system/runlevel1.target -> rescue.target
lrwxrwxrwx. 1 root root 17 Jul 3 21:37 /lib/systemd/system/runlevel2.target -> multi-user.target
lrwxrwxrwx. 1 root root 17 Jul 3 21:37 /lib/systemd/system/runlevel3.target -> multi-user.target
lrwxrwxrwx. 1 root root 17 Jul 3 21:37 /lib/systemd/system/runlevel4.target -> multi-user.target
lrwxrwxrwx. 1 root root 16 Jul 3 21:37 /lib/systemd/system/runlevel5.target -> graphical.target
lrwxrwxrwx. 1 root root 13 Jul 3 21:37 /lib/systemd/system/runlevel6.target -> reboot.target
```

Customizing Service Unit Files

Customizing Service Unit Files

- Unit files can be altered or extended by placing “drop-ins” under: `/etc/systemd/system/foobar.service.d/*.conf`
- Changes are applied on top of maintainer unit files.

```
# cat /etc/systemd/system/httpd.service.d/50-httpd.conf
[Service]
Restart=always
StartLimitInterval=10
StartLimitBurst=5
StartLimitAction=reboot
CPUShares=2048
Nice=-10
OOMScoreAdjust=-1000
```

Customizing Service Unit Files

- Run `systemctl daemon-reload` after making changes to notify systemd
- Drop-ins will be shown from `systemctl status`

```
# systemctl status httpd.service
httpd.service - The Apache HTTP Server
Loaded: loaded (/usr/lib/systemd/system/httpd.service;
enabled)
Drop-In: /etc/systemd/system/httpd.service.d
└─50-httpd.conf
```

Customizing Service Unit Files - Tips!

- Changes to unit files under `/usr/lib/systemd/system/` could be overwritten by updates. DON'T DO IT!
- `/etc` service files will take precedence over `/usr`
- Simply delete the drop-in to revert to defaults. Don't forget to run ``systemctl daemon-reload``
- `systemd-delta` – will show what is overridden and extended between `/usr` & `/etc`.
- `man 5 systemd.service`, `man 5 systemd.exec`

Resource Management

Making Cgroups Easier

- View cgroup hierarchy via `systemd-cgls`
- View usage stats via `systemd-cgtop` (use for tuning)
- Default hierarchy
 - `system.slice` – contains system services
 - `user.slice` – contains user sessions
 - `machine.slice` – contains virtual machines and containers
- Services can be promoted to their own slice if necessary.

Resource Management – Configuration

- `systemctl` can configure and persist cgroup attributes
- `systemctl set-property httpd.service CPUShares=2048`
- Add `--runtime` to not persist the settings:
 - `systemctl set-property --runtime httpd.service \ CPUShares=2048`
- Alternatively settings can be placed in unit files
 - `[Service]`
 - `CPUShares=2048`

Converting Init Scripts

Remember what an init-file looks like?

```
#!/bin/bash
#
# httpd          Startup script for the Apache HTTP Server
#
# chkconfig: - 85 15
# description: The Apache HTTP Server is an efficient and extensible \
#               server implementing the current HTTP standards.
# processname: httpd
# config: /etc/httpd/conf/httpd.conf
# config: /etc/sysconfig/httpd
# pidfile: /var/run/httpd/httpd.pid
#
### BEGIN INIT INFO
# Provides: httpd
# Required-Start: $local_fs $remote_fs $network $named
# Required-Stop: $local_fs $remote_fs $network
# Should-Start: distcache
# Short-Description: start and stop Apache HTTP Server
# Description: The Apache HTTP Server is an extensible server
#               implementing the current HTTP standards.
### END INIT INFO

# Source function library.
. /etc/rc.d/init.d/functions

if [ -f /etc/sysconfig/httpd ]; then
    . /etc/sysconfig/httpd
fi

# Start httpd in the C locale by default.
HTTPD_LANG=${HTTPD_LANG-"C"}

# This will prevent initlog from swallowing up a pass-phrase prompt if
# mod_ssl needs a pass-phrase from the user.
INITLOG_ARGS=""

# Set HTTPD=/usr/sbin/httpd.worker in /etc/sysconfig/httpd to use a server
# with the thread-based "worker" MPM; BE WARNED that some modules may not
# work correctly with a thread-based MPM; notably PHP will refuse to start.
```

```

# Path to the apachectl script, server binary, and short-form for messages.
apachectl=/usr/sbin/apachectl
httpd=${HTTPD-/usr/sbin/httpd}
prog=httpd
pidfile=${PIDFILE-/var/run/httpd/httpd.pid}
lockfile=${LOCKFILE-/var/lock/subsys/httpd}
RETVAL=0
STOP_TIMEOUT=${STOP_TIMEOUT-10}

# check for 1.3 configuration
check13 () {
    CONFFILE=/etc/httpd/conf/httpd.conf
    GONE="(ServerType|BindAddress|Port|AddModule|ClearModuleList|"
    GONE="${GONE}AgentLog|RefererLog|RefererIgnore|FancyIndexing|"
    GONE="${GONE}AccessConfig|ResourceConfig)"
    if LANG=C grep -Eiq "^[[:space:]]*($GONE)" $CONFFILE; then
        echo
        echo 1>&2 " Apache 1.3 configuration directives found"
        echo 1>&2 " please read /usr/share/doc/httpd-2.2.22/migration.html"
        failure "Apache 1.3 config directives test"
        echo
        exit 1
    fi
}

# The semantics of these two functions differ from the way apachectl does
# things -- attempting to start while running is a failure, and shutdown
# when not running is also a failure. So we just do it the way init scripts
# are expected to behave here.
start() {
    echo -n "Starting $prog: "
    check13 || exit 1
    LANG=$HTTPD_LANG daemon --pidfile=${pidfile} $httpd $OPTIONS
    RETVAL=$?
    echo
    [ $RETVAL = 0 ] && touch ${lockfile}
    return $RETVAL
}

```

```

# When stopping httpd, a delay (of default 10 second) is required
# before SIGKILLing the httpd parent; this gives enough time for the
# httpd parent to SIGKILL any errant children.
stop() {
    echo -n $"Stopping $prog: "
    killproc -p ${pidfile} -d ${STOP_TIMEOUT} $httpd
    RETVAL=$?
    echo
    [ $RETVAL = 0 ] && rm -f ${lockfile} ${pidfile}
}
reload() {
    echo -n $"Reloading $prog: "
    if ! LANG=$HTTPD_LANG $httpd $OPTIONS -t >&/dev/null; then
        RETVAL=6
        echo $"not reloading due to configuration syntax error"
        failure $"not reloading $httpd due to configuration syntax error"
    else
        # Force LSB behaviour from killproc
        LSB=1 killproc -p ${pidfile} $httpd -HUP
        RETVAL=$?
        if [ $RETVAL -eq 7 ]; then
            failure $"httpd shutdown"
        fi
    fi
    echo
}

# See how we were called.
case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    status)
        status -p ${pidfile} $httpd
        RETVAL=$?
        ;;
    restart)
        stop
        start
        ;;

```

```

condrestart|try-restart)
    if status -p ${pidfile} $httpd >&/dev/null; then
        stop
        start
    fi
;;
force-reload|reload)
    reload
;;
graceful|help|configtest|fullstatus)
    $apachectl $@
    RETVAL=$?
;;
*)
    echo $"Usage: $prog {start|stop|restart|condrestart|try-restart|force-reload|reload|status|fullstatus|graceful|help|configtest}"
    RETVAL=2
esac

exit $RETVAL

```

Contrast that with a systemd unit file syntax

```
[Unit]
Description=The Apache HTTP Server
After=network.target remote-fs.target nss-lookup.target

[Service]
Type=notify
EnvironmentFile=/etc/sysconfig/httpd
ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND
ExecReload=/usr/sbin/httpd $OPTIONS -k graceful
ExecStop=/usr/sbin/httpd $OPTIONS -k graceful-stop
KillSignal=SIGCONT
PrivateTmp=true

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

Test Unit File

- Copy the unit file
 - `cp [myapp].service /etc/systemd/system/`
- Alert systemd of the changes:
 - `systemctl daemon-reload`
- Start service
 - `systemctl start [myapp].service`
- View status
 - `systemctl status [myapp].service`

The Journal

The Journal - Logging with systemd

- “The journal is a component of systemd, that captures Syslog messages, Kernel log messages, initial RAM disk and early boot messages as well as messages written to STDOUT/STDERR of all services, indexes them and makes this available to the user”
- Indexed
- Formatted
- Errors in red
- Warnings in bold
- Security
- Reliability
- Intelligently rotated

Journal

- Does not replace rsyslog in RHEL 7
 - rsyslog is enabled by default
- Use rsyslog for traditional logging w/ enterprise features
- The journal is not persistent by default at the moment but a ring-buffer in /run/log/journal.
- Collects event metadata
- Stored in key-value pairs
 - man page: systemd.journal-fields(7)
- journalctl - utility for to viewing the journal.
 - Simple (or complex) filtering
 - Interleave units, binaries, etc

Using the Journal

- Enable persistence: ``mkdir /var/log/journal``
- View from boot: ``journalctl -b``
- Tail -f and -n work as expected:
 - `journalctl -f ; journalctl -n 50`
- Filter by priority: ``journalctl -p [level]``

0	emerg
1	alert
2	crit
3	err
4	warning
5	notice
6	debug

Using the Journal

- Other useful filters:
 - `--since=yesterday` or `YYYY-MM-DD (HH:MM:SS)`
 - `--until=YYYY-MM-DD`
 - `-u [unit]`
 - Pass binary e.g. `/usr/sbin/dnsmasq`
- View journal fields
 - `journalctl [tab] [tab]` ← bash-completion rocks!!
- Entire journal
 - `journalctl -o verbose` (useful for `grep`)

Troubleshooting the Boot Process

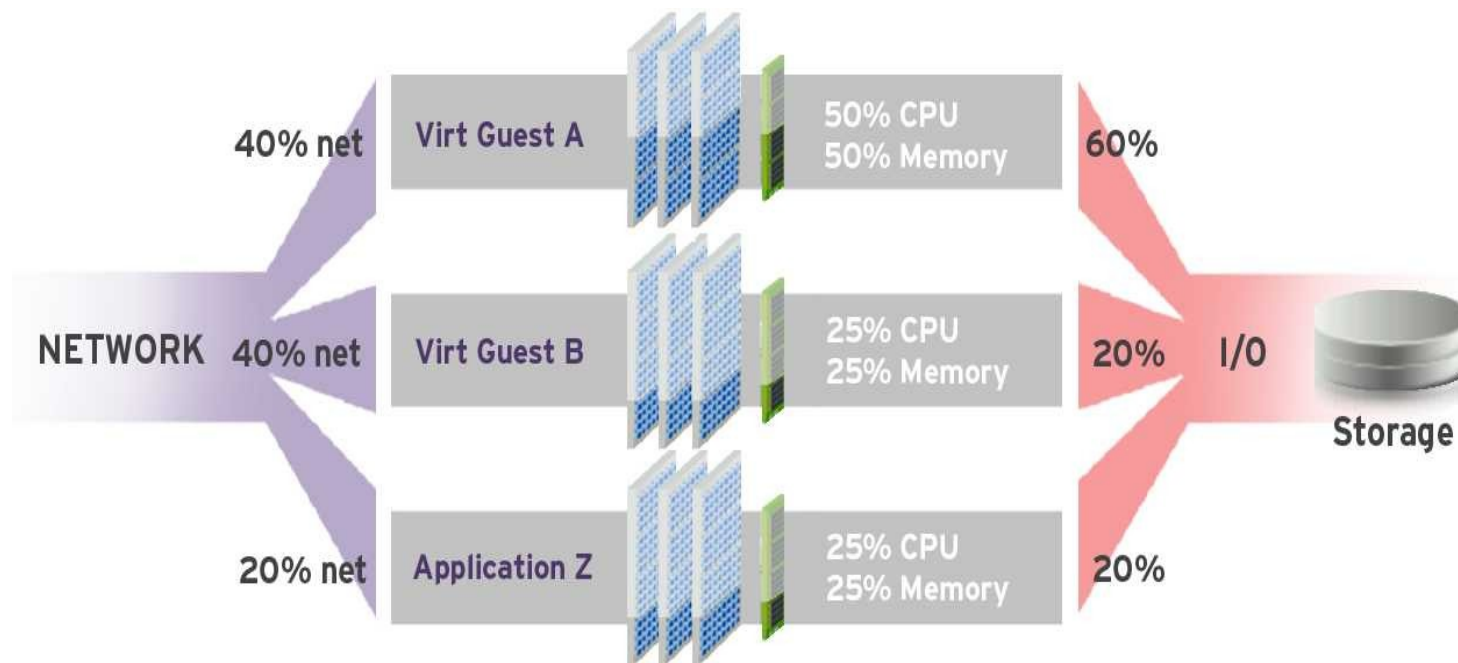
Booting

- Boot process is too fast, interactive boot append:
`systemd.confirm_spawn=1`
- `/var/log/boot.log` – still works the same
- Enable debugging from grub by appending:
 - `systemd.log_level=debug systemd.log_target=kmsg
log_buf_len=1M`
 - Or send debug info to a serial console: `systemd.log_level=debug
systemd.log_target=console console=ttyS0`
- Enable early boot shell on `tty9`
 - `systemctl enable debug-shell.service`
 - `ln -s /usr/lib/systemd/system/debug-shell.service \`
`/etc/systemd/system/sysinit.target.wants/`
- `systemctl list-jobs`

Resource Management

Control Groups Made Simple

Resource Management with cgroups can reduce application or VM contention and improve throughput and predictability



Slices, Scopes, Services

- In RHEL7 systemd manages cgroups, new concept of Scopes/Slices:
 - Slice – Unit type for creating the cgroup hierarchy for resource management.
 - Scope – Organizational unit that groups a services' worker processes.
 - Service – Process or group of processes controlled by systemd

Control Groups - Usability Improvements: Scopes

Systemd puts all related worker PIDs into cgroup called a 'scope'.

- Services
 - Apache processes in same services/apache scope
 - Mysql processes in same services/Mysql scope
 - Apache/Mysql get an equal "slice" of the system
- Users accounts
 - All users get an equal "slice"
- Machines
 - All containers/VMs get an equal "slice"
- No service/user/machine can dominate system

Control Groups - Usability Improvements: Slices

Special unit file for assigning resource constraints

Slices get assigned to scopes

- Systemd automatically assigns services to system.slice
- You can override resource with Unit file configuration
 - MemoryLimit=1g
- Command Line
 - `#> systemctl set-property httpd.service CPUShares=524 MemoryLimit=500M`
- Systemd will assign Containers to machine.slice
 - You can override by editing
 - `/etc/systemd/system/big-machine.slice`