systemd

1. Fundamentos

systemd "system Daemon" es, al igual que init System V o upstart, un sistema de arranque alternativo al init clásico. Está diseñado específicamente para el núcleo de Linux. Al igual que el init clásico, es el primer proceso arrancado por el núcleo una vez concluida la inicialización del mismo.

En la corta guerra entre los sucesores de init SystemV, ganó systemd, frente a upstart. Aparte de sus grandes cualidades y su adopción por el conjunto de editores profesionales para sus distribuciones, fue sobre todo la decisión de Debian en febrero de 2014 de adoptar systemd lo que venció las últimas resistencias de Canonical, fundador de Ubuntu y autor de upstart. Todas las distribuciones mayores (Red Hat, SuSE, Ubuntu, Debian) y sus derivados utilizan systemd.

El sustituto de init systemd es fruto de una reflexión profunda sobre la ejecución de servicios, combinando todos los sistemas operativos. Su autor, Lennart Poettering (a quien debemos también PulseAudio y Avahi) ha comparado en particular init System V, upstart y el sistema de arranque de Mac OS X y ha sacado conclusiones interesantes sobre las ventajas y limitaciones de cada uno de ellos. Se puede acceder a sus conclusiones en la siguiente página web: http://0pointer.de/blog/projects/why.html

Son bastante constructivas.

En su defensa, Lennart hace algunas constataciones interesantes sobre la duración del arranque de un sistema y el número de procesos que arrancan antes de llegar a la pantalla de login. Especialmente, invita a observar cuál es el PID del primer proceso iniciado por el usuario justo después del arranque de la máquina. Ronda el 150 en Mac OS, pero es alrededor de 1800 en una distribución clásica. Por lo tanto, hay un problema. Systemd permite acelerar el arranque de servicios, siempre basado en eventos y de forma asíncrona. Pero va más allá, especialmente anticipándose a las necesidades de varios servicios para acelerar su carga en el futuro, averiguar y gestionar el estado exacto de cada proceso que forme parte de un servicio controlando estos sobre todo gracias a una interfaz CLI y otra gráfica. Además, aboga por sustituir el shell script por un lenguaje compilado.

A pesar de que systemd utiliza mecanismos eficientes, hace invocaciones a funciones (API) únicamente presentes en el núcleo de Linux como, entre otras, el concepto de cgroups que permite agrupar procesos en un mismo grupo (cada proceso asociado a un servicio se ubica en un mismo grupo). Por ello, systemd ya no es compatible con POSIX y, por lo tanto, depende de Linux. A no ser que se contribuya con aportaciones grandes y pesadas a los núcleos de otros Unix, systemd no es portable.

2. Unidades objetivo y servicios

En systemd, el concepto de nivel de ejecución ya sólo existe para la compatibilidad con System V. El estado deseado por el sistema después de haber ejecutado los servicios se llama **unidad objetivo** o **target unit**. El componente básico de systemd es la **unidad** o **unit**. Existen varias: servicios, sockets, periféricos, objetivos, etc. Un objetivo es básicamente el punto de sincronización entre unidades en espera. De este modo, dispondrá, por ejemplo, de unidades objetivo correspondientes a cada nivel de ejecución, pero puede llamarlas como desee.

Un objetivo puede agrupar a otros. Puede activar o definir varios objetivos al mismo tiempo y no limitarse a uno solo, único y pesado. Por ejemplo, un objetivo contendrá la activación de los servicios de red, otro los servicios de audio y el último para el inicio del entorno gráfico, etc. Un objetivo se puede agrupar en otro.

3. Configuración

Las definiciones de las distintas unidades están en `/lib/systemd/system` . Las unidades objetivo o los servicios que deben ser gestionados por el sistema durante el arranque están en `/etc/system/system` , en forma de enlaces simbólicos o copias de los archivos anteriores. Estas rutas pueden ser ubicadas en diferentes entornos según las distribuciones. La ruta de la configuración de sistema se obtiene mediante la siguiente línea de comando:

```
# pkg-config systemd --variable=systemdsystemconfdir
```

En CentOS 7 (Red Hat) : `/etc/systemd/system`

En Ubuntu 16.04 LTS : `/lib/systemd/system`

Se trata de la configuración del sistema. Otros directorios pueden utilizarse:

```
/usr/local/lib/systemd/system
/usr/lib/systemd/system
```

La configuración de usuario se define por lo general en `/etc/systemd/system` .

En todos los casos, la configuración de usuario toma prevalencia sobre la configuración de sistema.

4. Objetivos

a. Equivalencia con init System V

A pesar de ser un atajo simple, la noción de objetivo es la de acercarse al nivel de ejecución clásico, ya que un objetivo es un estado a obtener. Un archivo objetivo lleva el sufijo **.target**. La ventaja de systemd es que a conseguido harmonizar los diferentes objetivos entre las distribuciones. Aquí tenemos una tabla comparativa de los objetivos entre System V y systemd:

Runlevel	Target Unit	Descripción
0	runlevel0.target, poweroff.target	Extinción de la máquina
1	runlevel1.target, rescue.target	Boot en modo single
2	runlevel2.target, multi-user.target	Consola, multi-usuario
3	runlevel3.target, multi-user.target	Consola, multi-usuario
4	runlevel4.target, multi-user.target	Consola, multi-usuario
5	runlevel5.target, graphical.target	Gráfico, multi-usuario
6	runlevel6.target, reboot.target	Reboot del equipo

b. Conocer el objetivo por defecto

El comando siguiente proporcionar el objetivo por defecto, es decir el estado en el cual el sistema alcanza al arrancar:

```
$ systemctl get-default
graphical.target
```

El objetivo por defecto es un simple enlace simbólico llamado default.target, al archivo objetivo seleccionado.

```
$ cd /etc/systemd/system/
$ ls -l default.target
 lrwxrwxrwx. 1 root root 40 abr. 2 09:18 default.target ->
```

```
/usr/lib/systemd/system/graphical.target
```

c. Cambiar el objetivo por defecto

Para cambiar el objetivo por defecto, utilice el comando **systemctl** como sigue:

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

Este comando tiene por efecto la modificación del objetivo del enlace simbólico. El hecho de cambiar el objetivo por defecto no modifica el estado actual del sistema.

d. Pasar de un objetivo a otro

El equivalente de un telinit es el comando **systemctl isolate**:

```
# systemctl isolate graphical.target
```

e. Modo seguro y emergencia

Dos objetivos particulares tienen un acceso directo desde el comando **systemctl**: `rescue` y `emergency`.

```
# systemctl rescue
# systemctl emergency
```

En los dos casos, se trata de alias de los comandos **isolate** vers les cibles respectives (rescue.target y emergency.target). El modo `rescue` es el modo mono usuario. El modo `emergency` se dedica a los casos críticos; proporciona el modo de funcionamiento más pequeño posible: el sistema de archivos raiz se monta en modo solo lectura, no se puede acceder a ningún otro sistema de archivos ni a la red.

f. Objetivos activos y dependencias

Un objetivo puede estar constituido por varios objetivos. Definir objetivos "reducidos" permite agruparles de forma más sencilla y reutilizarlos. Para listar los objetivos actualmente activos:

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

UNIT                LOAD ACTIVE SUB    DESCRIPTION
basic.target        loaded active active Basic System
cryptsetup.target   loaded active active Local Encrypted Volumes
getty.target        loaded active active Login Prompts
graphical.target    loaded active active Graphical Interface
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-pre.target  loaded active active Network (Pre)
network.target      loaded active active Network
nss-lookup.target   loaded active active Host and Network Name Lookups
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
slices.target       loaded active active Slices
sockets.target      loaded active active Sockets
swap.target         loaded active active Swap
sysinit.target      loaded active active System Initialization
time-sync.target    loaded active active System Time Synchronized
timers.target       loaded active active Timers

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

20 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

Encontramos por supuesto el objetivo por defecto, graphical.target, pero no está solo. Esto significa que la activación de un objetivo puede provocar la activación de otros. Se emplea también un mecanismo de dependencias. Aquí vemos el contenido del objetivo

graphical.target:

```
# cat /usr/lib/systemd/system/graphical.target
[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
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
```

Este archivo contiene:

- La línea **Requires=** proporciona la lista de las unidades dependientes obligatorias del objetivo.
- La línea **After=** indica el orden en que las unidades deben ser cargadas (pero estas no son obligatorias).

También encontramos un directorio multi-user.target.wants, que contiene una lista de las unidades, con frecuencia enlaces simbólicos, que deben ser cargados durante la activación del objetivo.

```
# ls -1 graphical.target.wants
accounts-daemon.service
rtkit-daemon.service
switcheroo-control.service
udisks2.service
upower.service
...
```

g. Listar todos los objetivos

Agregando `--all` al comando de listado, obtenemos todos los objetivos disponibles en el sistema.

```
# systemctl list-units --type target --all
```

5. Servicios

Los servicios terminan con el sufijo `.service`. Al ser unidades como las otras, se controlan con el mismo comando **systemctl**. Tienen el mismo rol que los servicios init System V, y pocas cosas más, se utilizan de manera idéntica. Sin embargo, donde los servicios System V son con frecuencia simples scripts shell (bash ou sh), las unidades de servicios son archivos de texto estandarizados que describen las reglas y comandos vinculados a ese servicio.

Las herramientas como `service` o `chkconfig` son inútiles, y son sustituidas por el comando **systemctl**.

a. Acciones

El comando **systemctl** accepta las acciones habituales, como **start**, **stop** o **restart**. A título comparativo, aquí tenemos una tabla que presenta las acciones posibles y sus equivalentes con el comando service (o el servicio System V mismo);

servicio	systemctl	Descripción
`service name start`	`systemctl start name`	Arranca el servicio.
`service name stop`	`systemctl stop name`	Detiene el servicio.
`service name restart`	`systemctl restart name`	Reinicia el servicio.
`service name condrestart`	`systemctl try-restart name`	Reinicia el servicio solo si está arrancado.
`service name reload`	`systemctl reload name`	Vuelve a cargar la configuración.

`service name status`	`systemctl status name`
	
	`systemctl is-active name`
		Indica si el servicio está arrancado.

`service --status-all`	`systemctl list-units --type service -all`	Muestra el estado de todos los servicios.

Obtendrá, por ejemplo, una lista de los servicios activos como ésta:

```
# 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
```

```
  abrtd.service        loaded active running ABRT Automated Bug Reporting Tool
  atd.service          loaded active running Job spooling tools
  auditd.service       loaded active running Security Auditing Service
  avahi-daemon.service  loaded active running Avahi mDNS/DNS-SD Stack
  chronyd.service      loaded active running NTP client/server
  crond.service        loaded active running Command Scheduler
  cups.service         loaded active running CUPS Printing Service
  dbus.service         loaded active running D-Bus System Message Bus
...
```

Notará que, si la lista sobrepasa el tamaño de su consola, se espera una tecla para pasar a la siguiente. Para evitar este funcionamiento, utilice el siguiente parámetro:

```
$ systemctl list-units --type=service --no-pager
```

Sólo los servicios activos (vinculados al objetivo actual o arrancados de forma específica) se mostrarán. Para tener una lista más completa, agregue `--all`:

```
# systemctl list-units --type=service --no-pager --all
```

Para obtener la lista completa de los servicios, sea cual sea su estado:

```
# systemctl list-unit-files --type service | wc -l
251
```

b. Estado

El estado proporcionado por `systemctl` es por lo general más completo que el provisto por System V, donde son los mismos scripts y syslog quienes permiten obtener las trazas. Así, constatará más abajo que una gran cantidad de información se proporciona y en particular las trazas del servicio. En el caso siguiente ssh, lo que puede ser muy util en caso de que el servicio no arranque.

```
# systemctl status sshd

 sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2021-04-21 22:38:05 CEST; 11min ago
     Docs: man:sshd(8)
           man:sshd_config(5)
 Main PID: 679 (sshd)
    Tasks: 1 (limit: 2324)
   Memory: 5.6M
      CPU: 169ms
   CGroup: /system.slice/sshd.service
           +-679 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups

abr 21 22:38:03 localhost.localdomain systemd[1]: Starting OpenSSH server daemon...
abr 21 22:38:05 localhost.localdomain sshd[679]: Server listening on 0.0.0.0 port 22.
abr 21 22:38:05 localhost.localdomain sshd[679]: Server listening on :: port 22.
abr 21 22:38:05 localhost.localdomain systemd[1]: Started OpenSSH server daemon.
abr 21 22:45:17 localhost.localdomain sshd[799]: Connection closed by authenticating
user alejandro 192.168.1.22 port 57511 [preauth]
abr 21 22:45:20 localhost.localdomain sshd[797]: Connection closed by authenticating
user alejandro 192.168.1.22 port 57510 [preauth]
abr 21 22:45:45 localhost.localdomain sshd[801]: Accepted password for alejandro from
192.168.1.22 port 57518 ssh2
abr 21 22:45:45 localhost.localdomain sshd[801]: pam_unix(sshd:session): session opened
for user alejandro(uid=1000) by (uid=0)
~
```

Con respecto a estas trazas, puede consultar **journalctl** para una descripción más detallada.

c. Activación

La activación o desactivación de un servicio permite especificar su estado deseado en la carga del objetivo. Sin embargo, estos cambios no tendrán efecto inmediato en el servicio (iniciar o detener), esto requerirá la ejecución de un `start/stop`.

Al igual que para el comando **service**, aquí tenemos una tabla de correspondencia para el

comando **chkconfig**:

chkconfig	systemctl	Descripción.
`chkconfig name on`	`systemctl enable name`	Activar un servicio.
`chkconfig name off`	`systemctl disable name`	Desactivar un servicio.

`chkconfig --list name`	`systemctl status name`
	
	`systemctl is-enabled name`
		Indica el estado del servicio.

`chkconfig --list`	`systemctl list-unit-files --type=service`
	
	`systemctl list-dependencies --before / --after name`
		Lista de los servicios, con sus dependencias.

La activación de un servicio solo añade un enlace simbólico en la unidad concernente en el directorio "wants" correspondiente al objetivo actual, como muestran los comandos siguientes:

```
# systemctl disable sshd
Removed symlink /etc/systemd/system/multi-user.target.wants/sshd.service.
# systemctl enable sshd
Created symlink from /etc/systemd/system/multi-
user.target.wants/sshd.service to /usr/lib/systemd/system/sshd.service.
```

Si ha ido siguiendo el hilo, habrá visto que estábamos en objetivo graphical.target, y sin embargo los comandos anteriores han activado y desactivado el servicio para el objetivo multi-user.target. La razón es que el objetivo multi-user.target es una dependencia del primero.

d. Ocultación

Para o desactivar un servicio no impide que sea ejecutado de forma manual o como dependencia. Podemos también constatar que un servicio que se pensaba que estaba completamente detenido se encuentra de nuevo en curso de ejecución. La solución es ocultarlo con `mask` :

```
# systemctl mask sshd
```

Una secuencia completa para deshacerse de forma permanente de un servicio es:

```
# systemctl stop sshd
# systemctl disable sshd
# systemctl mask sshd
```

El servicio puede volver a hacerse visible con `unmask`.

e. Dependencias

Al igual que los objetivos, los servicios tienen dependencias. Estas son facilmente visibles con `list-dependencies`. Observe que el sentido de `after/before` ne debe ser comprendido como "antes de haber arrancado sshd", si no "sshd arranca después de estas unidades". Un árbol de dependencias se construye, he indica todo lo necesario para el arranque del servicio sshd.

```
# systemctl list-dependencies --after sshd
sshd.service
      sshd-keygen.service
      system.slice
```

```
systemd-journald.socket
basic.target
  rhel-import-state.service
  systemd-ask-password-plymouth.path
  paths.target
    brandbot.path
    cups.path
    systemd-ask-password-console.path
    systemd-ask-password-wall.path
  slices.target
    -.slice
    system.slice
    user.slice
  sockets.target
    avahi-daemon.socket
```

En una Fedora 31 hay 176 dependencias.

Estas dependencias se definen, de manera recursiva, en cada uno de los archivos de la unidad, como es el caso de los objetivos. Las líneas Wants, After, y WantedBy definen las dependencias.

```
[Unit]
Description=OpenSSH server daemon
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target sshd-keygen.target
Wants=sshd-keygen.target

[Service]
Type=notify
EnvironmentFile=-/etc/crypto-policies/back-ends/opensshserver.config
EnvironmentFile=-/etc/sysconfig/sshd-permitrootlogin
EnvironmentFile=-/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS $CRYPTO_POLICY$PERMITROOTLOGIN
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s
```

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

6. Compatibilidad con System V

Muchos scripts de arranque son genéricos, bien por los diferentes sistemas init o por los muchos sistemas Unix diferentes.

Esta compatibilidad pasa por varias posibilidades. Por ejemplo, un servicio rc.service permitirá leer y ejecutar el contenido de /etc/rcX.d. O bien, un programa llamado systemd-sysv-generator hará exactamente lo mismo.

Sin embargo, es en todo caso preferible que convierta sus scripts de arranque en unidades systemd.

7. Acciones de sistema

Al igual que init (via `halt`, `reboot` o `telinit`) permite reiniciar o detener un sistema, systemd hace lo mismo. Aquí tenemos una nueva tabla de comparación:

Comando	Systemd	Descripción
`halt`	`systemctl halt`	Detiene el sistema sin apagar el equipo.
`poweroff`, `halt -p`	`systemctl poweroff`	Detiene el sistema y apaga el equipo.
`reboot`	`systemctl reboot`	Reinicia el equipo.
`pm-suspend`	`systemctl suspend`	Suspende la ejecución del sistema (ahorro de energia).
`pm-hibernate`	`systemctl hibernate`	Hiberna el sistema.

Los comandos **shutdown** todavía están disponibles, al igual que **halt**, **reboot** o **poweroff** que son aliases a los comandos **systemd** asociados.

8. Gestión de la consola

El diagrama presentado en la sección dedicada a init System V sigue siendo válido: systemd reemplaza init. Systemd autoriza el reinicio automático de un servicio usando la entrada **Restart**. Las consolas y los terminales se convierten en meros servicios, cuyo comando será ejecutado en cada desconexión de un usuario. He aquí el ejemplo de una Fedora 31, fíjese en el valor **always**:

```
# cat /usr/lib/systemd/system/console-getty.service
...
```

```
[Service]
ExecStart=-/sbin/agetty -o '-p -- \\u' --noclear --keep-baud console
115200,38400,9600 $TERM
Type=idle
Restart=always
...
```

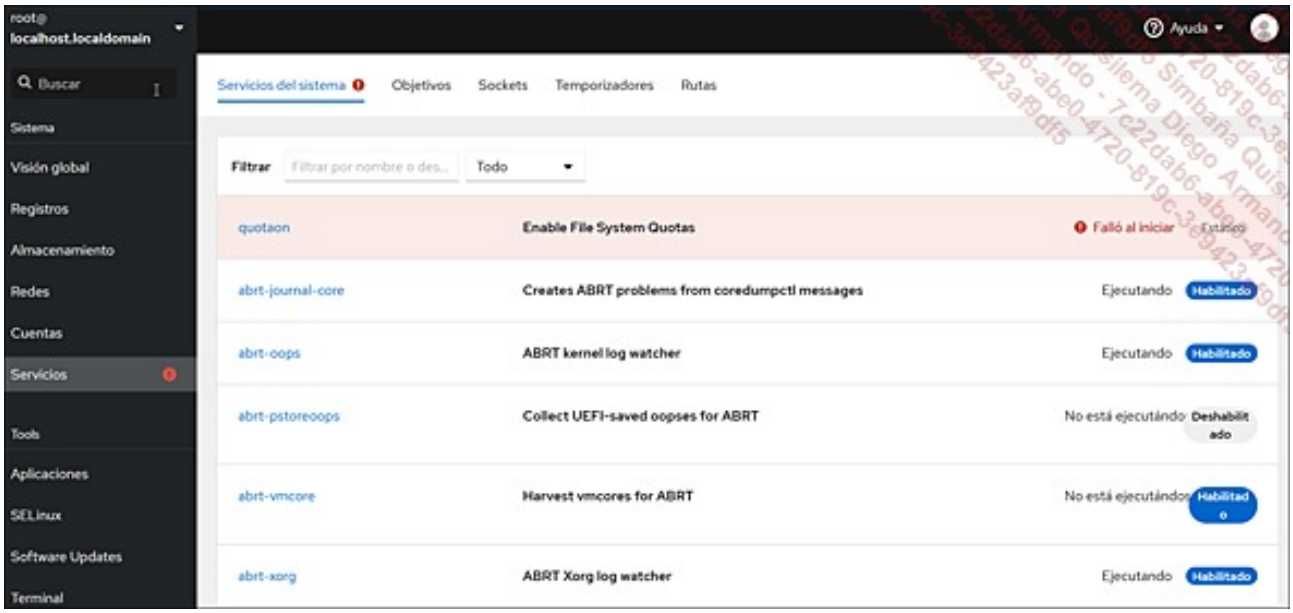
9. Interfaz gráfica

No existe una interfaz gráfica oficial de systemd. Hubo una, que todavía estaba disponible en antiguas distribuciones, llamada **systemadm**. Formaba parte del paquete **systemd-ui**Debian y Ubuntu. Esta interfaz permitía controlar el conjunto de unidades systemd y realizar todas las acciones del comando systemctl en varias unidades.

También puede usar **systemd-manager**, considerado como inestable, a través del gestor de paquetes snap, bajo su propio riesgo.

```
$ sudo snap install --beta --devmode systemd-manager
```

Puede usar la interfaz web **cockpit** para, entre otras cosas, administrar los servicios systemd. Cockpit está disponible aquí: https://cockpit-project.org/



Cockpit pilotando los servicios systemd