Clase 23: Rsyslog y tareas programadas

En un sistema Linux hay muchas tareas rutinarias que deben llevarse a cabo periódicamente: la actualización de la base de datos de archivos mediante `updatedb`, copias de respaldo, limpieza de archivos temporales, por nombrar algunos ejemplos. En Linux disponemos de una utilidad llamada **cron** que nos permite realizar tareas repetitivas de manera automática y periódica, de registrar el resultado en un log, y de reportar cualquier anomalía al administrador. Si solamente necesitamos ejecutar una tarea en el futuro por única vez, **at** nos resultará de mayor utilidad. Estos serán los temas principales de este capítulo, junto con el registro de eventos del sistema utilizando el clásico **rsyslog** y **journald**, la herramienta provista por **systemd**.

Cómo usar cron

Para empezar, aclaremos que **cron** en realidad está compuesto por dos utilidades separadas:

- **crond** es el servicio que "despierta" cada minuto y chequea si hay tareas que ejecutar en la tabla de tareas agendadas.
- **crontab** es el programa que nos permite editar la tabla (la cual en realidad consiste en un archivo de texto plano) de tareas agendadas.

Salvo que el administrador lo haya configurado de otra forma, todos los usuarios pueden automatizar sus propias tareas creando una tabla personal. Como dijimos anteriormente, las tablas de tareas agendadas son archivos de texto plano. Por lo general, estos archivos se guardan en **/var/spool/cron** y en cada línea de estos aparecen el detalle de cuál es la tarea que se debe ejecutar, cuándo, y utilizando qué cuenta del sistema. Por esta razón, por lo general cada tarea agendada recibe el nombre de *crontab entry*, o *entrada de crontab*.

Crear y agendar una tarea

Para crear una tarea agendada, utilizaremos el comando

```
crontab -e
```

A continuación, se abrirá el editor de texto que tengamos configurado por defecto, con el que deberemos escribir (en la primera línea libre del archivo que se abra) utilizando el siguiente formato:

```
minuto hora día_del_mes día_de_la_semana comando
```

Donde:

- minuto (0 a 59)

- hora (0 a 23)
- día_del_mes (1 a 31)
- mes (1 a 12)
- día_de_la_semana (0: Domingo a 6: Sábado)

Por ejemplo, para ejecutar el comando **/bin/updatedb** todos los días lunes de septiembre a las 13:55, agregaríamos la siguiente línea en el **crontab** de un usuario que tenga permiso para ejecutar tal programa:

```
55 13 * 9 1 /bin/updatedb
```

---

*Cuando colocamos el comando en el archivo de **crontab**, se aconseja escribir la ruta completa al ejecutable.*

Analicemos la línea anterior:

- Los dos primeros números indican la hora (13:55)
- La estrella (o asterisco) indica que la regla se cumple sin importar qué día del mes (1 al 31) caiga lunes.
- Los siguientes dos números representan el mes en el que queremos ejecutar la tarea (9: septiembre) y el día de la semana (1: lunes).

Alternativamente, podemos especificar el momento deseado para la ejecución de una tarea empleando las siguientes indicaciones:

ALTERNATIVA	DESCRIPCIÓN
`@reboot miprograma`	Ejecutar una vez al iniciar el sistema
`@yearly miprograma`	Ejecutar una vez por año (equivalente a `0 0 1 1 * miprograma`)
`@monthly miprograma`	Una vez por mes (equivalente a `0 0 1 * * miprograma`)
`@weekly miprograma`	Una vez por semana (equivalente a `0 0 * * 0 miprograma`)
`@daily miprograma`	Una vez por día (equivalente a `0 0 * * * miprograma`)
`@hourly miprograma`	Una vez por día (equivalente a `0 * * * * miprograma`)

Finalmente, los scripts que se depositen en **/etc/cron.hourly**, **/etc/cron.daily**, **/etc/cron.weekly**, y **/etc/cron.monthly** se ejecutarán una vez por hora, por día, por semana, o por mes, respectivamente. Por lo general, ese es el caso de algunas aplicaciones que al momento de su instalación configuran su tarea agendada de mantenimiento propio como apreciamos en la imagen siguiente:

```
ls /etc/cron.{hourly,daily,weekly,monthly}
```

```
[gacanepa@server ~]$ ls /etc/cron.{hourly,daily,weekly,monthly}
/etc/cron.daily:
0yum-daily.cron  logrotate  man-db.cron  mlocate

/etc/cron.hourly:
0anacron  0yum-hourly.cron

/etc/cron.monthly:

/etc/cron.weekly:
[gacanepa@server ~]$
```

En el caso particular que observamos arriba, solamente **/etc/cron.hourly** y **/etc/cron.daily** contienen scripts que se ejecutarán una vez por hora y por día, respectivamente. Es importante aclarar que esto puede variar de acuerdo a la distribución que estemos empleando y el uso que le estemos dando al sistema.

Ejemplos de cron: uso de comodines

Recordemos que el servicio **crond** chequea cada minuto las entradas de **crontab** y las ejecuta si se cumplen las condiciones de fecha, día, y hora especificadas en cada una. Cuando deseamos que una condición se cumpla siempre, utilizamos un asterisco en el lugar correspondiente. Por ejemplo, si necesitamos ejecutar el script **/home/alumno/miscript.sh** todos los días a las 12:45 hs, agregaremos la siguiente línea a nuestra tabla:

```
45 12 * * * /home/alumno/miscript.sh
```

Mientras que los primeros dos números especifican la hora (12:45 hs), los siguientes tres asteriscos indican que la tarea debe ejecutarse 1) todos los días (1 a 31), 2) de todos los meses, y 3) todos los días de la semana (Domingo a Sábado), respectivamente.

Siguiendo ese razonamiento, ¿qué indicará la siguiente línea?

```
00,15,30,45 11 * * 1-5 /home/alumno/segundoscript.sh
```

Repasemos en detalle:

El primer (00,15,30,45) y segundo campos (11) indican que este script debe correr a las 11:00, 11:15, 11:30, y 11:45 hs. de cualquier día del mes, todos los meses (indicado por el * en el tercer y cuarto campos, respectivamente), pero solamente de Lunes a Viernes (1-5 en el quinto campo). Dicho de forma más simple, **cron** ejecutará este script a las 11:00, 11:15, 11:30, y 11:45 hs. de Lunes a Viernes.

Como vemos, las entradas de **crontab** admiten listas de elementos separados por comas y rangos para especificar las condiciones de fecha, día, y hora.

Para ver nuestra propia tabla de tareas agendadas utilizaremos `crontab -l`. El usuario root también puede ver (y por supuesto, también editar) la tabla de otros usuarios. Para eso se debe agregar la opción `-u` seguida del nombre del usuario en cuestión.

Consideraciones finales

La sección 5 del *man page* de **crontab** (la cual se invoca mediante `man 5 crontab`) especifica que en el archivo de configuración de este programa (**/etc/crontab**) y en nuestras tablas personales podemos utilizar variables de entorno. Esto significa que es posible otorgarle valores a variables que queremos que estén disponibles durante la ejecución. Algunas de ellas son las siguientes:

PATH contiene los directorios donde crontab podrá buscar por archivos binarios ejecutables. Si el directorio donde se encuentra uno en particular forma parte de esta variable, podremos utilizar una ruta relativa (en vez de una absoluta, como mencionamos en el post anterior) en el sexto campo de la entrada de **crontab**. Por ejemplo, dada el valor de la variable **PATH** que mostramos a continuación:

```
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
```

si el programa que deseamos ejecutar está ubicado en **/usr/local/bin** (ver resaltado en rojo), podemos escribir

```
10 15 * * * miprograma
```

en vez de

```
10 15 * * * /usr/local/bin/miprograma
```

SHELL especifica la shell a utiliza para ejecutar las tareas agendadas. Si se omite esta variable, **crontab** utilizará por defecto **/bin/sh**.

MAILTO permite especificar una dirección de correo a la que se deberá enviar la salida de un programa (incluyendo cualquier error). Si se desea enviar el correo electrónico a un usuario del mismo equipo, solamente es necesario especificar el nombre de este. Por otra parte, también es posible enviar el reporte a una dirección externa (siempre y cuando el equipo tenga un servicio de correo instalado). Por ejemplo,

```
MAILTO=alumno
```

hará el envío al usuario alumno del mismo equipo, mientras que

```
MAILTO=fulano@midominio.com
```

indica que se debe enviar a la dirección mencionada.

Puede suceder que no deseemos que todos los usuarios del sistema tengan acceso a **cron** (probablemente para impedir que agenden tareas o inicien procesos que consumirían recursos del sistema o que representen una falla de seguridad). Para restringir el acceso deberemos crear (si no existen ya) los siguientes archivos:

- **/etc/cron.allow**
- **/etc/cron.deny**

Estos archivos consisten en listas de usuarios, uno por línea. Si el archivo **.allow** existe, solamente los usuarios que aparezcan en el mismo podrán hacer uso de la herramienta correspondiente. En el caso de que **.allow** no exista pero sí **.deny**, solamente se permitirá el acceso a los usuarios que **no** aparezcan en el mismo. Si ninguno de los dos archivos (**.allow** y **.deny**) están presentes, todos los usuarios tendrán acceso a **cron**.

Anacron

Inclusive con todo el empeño que pongamos en mantener nuestra tabla de tareas, puede que una de ellas no se ejecute en el momento estipulado (debido a que el equipo estaba apagado o reiniciándose, por ejemplo). De esta manera, el sistema esperará hasta la próxima coincidencia de tiempo para correrla. Por ejemplo, si un script debe correr diariamente a las 11:30 am y el equipo está apagado en ese momento, **cron** recién intentará ejecutarlo nuevamente el próximo día a la misma hora. Si la situación se repite luego, es probable que la tarea nunca corra. Para vencer esa limitación podemos hacer uso de **anacron**.

Anacron puede hacer que se ejecuten tareas agendadas que no corrieron mientras el equipo estaba apagado. Cuando el sistema vuelva a iniciarse, el servicio se encarga de que corran una después de otra. En la imagen siguiente observamos su configuración en el archivo **/etc/anacrontab**:

```
[root@server ~]# cat /etc/anacrontab
# /etc/anacrontab: configuration file for anacron

# See anacron(8) and anacrontab(5) for details.

SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
# the maximal random delay added to the base delay of the jobs
RANDOM_DELAY=45
# the jobs will be started during the following hours only
START_HOURS_RANGE=3-22

#period in days   delay in minutes   job-identifier   command
1       5       cron.daily              nice run-parts /etc/cron.daily
7       25      cron.weekly             nice run-parts /etc/cron.weekly
@monthly 45     cron.monthly            nice run-parts /etc/cron.monthly
[root@server ~]#
```

En la parte superior vemos las siguientes variables generales:

- **SHELL** indica cuál es la consola que se utilizará para correr scripts (**/bin/sh**).
- **PATH** representa las rutas donde se buscarán los ejecutables si no se especifican rutas absolutas de los scripts (**/sbin:/bin:/usr/sbin:/usr/bin**).
- **MAILTO** es la dirección de correo electrónico donde se deben enviar los mensajes de error (root en este caso).
- **RANDOM_DELAY** es el número máximo de minutos que transcurrirán desde el momento en que el sistema se inicia y las tareas pendientes comienzan a ejecutarse.
- **START_HOURS_RANGE** representa el intervalo durante el cual se ejecutarán las tareas pendientes.

Más abajo podemos ver las siguientes tareas agendadas:

```
cron.daily (nice run-parts /etc/cron.daily)
cron.weekly (nice run-parts /etc/cron.weekly)
cron.monthly (nice run-parts /etc/cron.monthly)
```

El comando `run-parts` se encarga de ejecutar los archivos ejecutables que se encuentran dentro de los directorios especificados. Lo hace de a uno a la vez en orden alfabético en intervalos especificados en la columna **delay in minutes**. Según podemos ver en la imagen de arriba, esto significa que los scripts ubicados en **/etc/cron.daily** se ejecutan luego de 5 minutos entre uno y el siguiente. Finalmente, la columna **period in days** nos indica cada cuántos días se debe chequear si hay tareas pendientes en alguno de los directorios mencionados. En resumen, cron actualiza la fecha y hora en la que se ejecutó una cierta tarea para que anacron no vuelva a ejecutarla si dicha fecha y hora se encuentra dentro del período dado por **period in days**.

Ejecutar una tarea programada por única vez

Algunas veces desearemos programar una tarea para su ejecución por una única vez. Si bien en teoría podríamos utilizar una entrada de crontab para cumplir con este objetivo, en Linux disponemos de una utilidad llamada at que nos permite hacerlo de manera mucho más fácil y sin tener que borrar la entrada posteriormente. Por ese motivo, en este post mostraremos cómo usar at para correr un programa o ejecutar un comando agendado una sola vez.

Cómo usar at

Para usar at es necesario instalarlo para luego iniciar el servicio y habilitarlo para que arranque en los próximos reinicios. Para instalar at, utilizaremos los siguientes comandos:

```
yum install at # CentOS
aptitude install at # Debian y similares
systemctl start atd # Iniciar
systemctl enable atd # Habilitar
```

Una vez que el servicio atd está corriendo, podemos agendar cualquier tarea de la siguiente manera:

```
echo "escribir el comando aquí entre comillas" | at [opción para indicar el momento deseado]
```

donde el tiempo (fecha y hora) deseado de ejecución debe ser compatible con el estándar POSIX. A continuación algunos ejemplos:

Hora:

- 9pm
- 20:45
- now (ahora)
- noon (12pm)
- midnight (00:00)

Fecha:

- today (hoy)
- tomorrow (mañana)
- Sep 10 (10 de septiembre)
- Jan 5 (5 de enero)

Podemos también crear combinaciones de fecha y hora, como por ejemplo:

- **10:45pm Jan 10** representa las **22:45** hs del día **10 de enero**
- **8:23am May 11** indica las **8:23** hs del día **11 de mayo**

Incremento:

- **+minutes** (minutos)

- **+hours** (horas)
- **+days** (días)
- **+weeks** (semanas)
- **+months** (meses)
- **+years** (años)

---

*Para el incremento también se aceptan las formas singulares **minute**, **hour**, **day**, **week**, **month**, y **year**. El incremento se utiliza para "sumar" una cierta medida de tiempo al momento indicado para la ejecución de la tarea.*

Veamos ahora algunos ejemplos que nos ayudarán a ilustrar:

EJEMPLO 1 – Para crear un tarball llamado **scripts.tar.gz** (donde guardaremos todos los archivos **.sh** ubicados dentro del directorio actual) dentro de 2 minutos, haremos lo siguiente:

```
echo "tar czf scripts.tar.gz *.sh" | at now +2 minutes
```

Veamos el resultado en la imagen siguiente, donde podemos observar que la fecha de modificación del archivo corresponde con la hora anunciada por `at` para la ejecución del comando anterior:

```
[gacanepa@server ~]$ date
Sat Oct  6 16:32:44 -03 2018
[gacanepa@server ~]$ echo "tar czf scripts.tar.gz *.sh" | at now +2 minutes
job 10 at Sat Oct  6 16:34:00 2018
[gacanepa@server ~]$ ls -l scripts.tar.gz
-rw-rw-r-- 1 gacanepa gacanepa 1896 Oct  6 16:34 scripts.tar.gz
[gacanepa@server ~]$
```

EJEMPLO 2 – Si queremos ejecutar `updatedb` el 11 de septiembre a las 7:15am, usaremos el siguiente comando:

```
echo "updatedb" | at 7:15am Sep 11
```

---

*De la misma manera en que lo hacemos con **cron**, también es posible restringir el acceso a `at` utilizando los archivos **/etc/at.allow** y **/etc/at.deny***

Existen otros comandos para administrar las tareas que hayamos agendado mediante dicha herramienta:

1) ver nuestra propia lista de tareas agendadas (o las de todos los usuarios, si contamos con permisos de root): `atq`.

2) eliminarlas de manera individual: `atrm` seguido por el número de tarea que devuelva `atq`.

```
[gacanepa@server ~]$ echo "tar czf scripts.tar.gz *.sh" | at now +2 minutes
job 11 at Sat Oct  6 19:11:00 2018
[gacanepa@server ~]$ echo "updatedb" | at 7:15am Sep 11
job 12 at Wed Sep 11 07:15:00 2019
[gacanepa@server ~]$ atq
11      Sat Oct  6 19:11:00 2018 a gacanepa
12      Wed Sep 11 07:15:00 2019 a gacanepa
[gacanepa@server ~]$ atrm 12
[gacanepa@server ~]$ atq
11      Sat Oct  6 19:11:00 2018 a gacanepa
[gacanepa@server ~]$
```

3) hacer que una de ellas se ejecute recién cuando los niveles de uso del sistema estén por debajo de un umbral dado: `batch`.

Si estamos ante un equipo en el que estén corriendo procesos que ocupen el CPU de manera más o menos intensiva, podemos utilizar este comando a fin de ejecutar una tarea cuando la carga promedio (load average) caiga por debajo de 0.8. Por ejemplo,

```
echo "updatedb" | batch
```

hará que se ejecute updatedb cuando se cumpla la condición que acabamos de mencionar.

Los timers en systemd

Los *timers* (temporizadores) en **systemd** son un tipo especial de unidad que permiten agendar tareas de forma periódica de forma similar a **cron**. Sin embargo, presentan la particularidad de poseer mayor flexibilidad a la hora de especificar cuándo ha de llevarse a cabo una acción.

Los archivos de definición de los temporizadores se caracterizan por tener la extensión **.timer**, y contienen las instrucciones para controlar un servicio o evento. Para ilustrar, analicemos el ejemplo que vemos en la imagen siguiente:

```
[Unit]
Description=Daily apt download activities

[Timer]
OnCalendar=*-*-* 6,18:00
RandomizedDelaySec=12h
Persistent=true
```

En la sección **[Unit]** encontraremos una descripción del temporizador, mientras que debajo de **[Timer]** veremos la configuración propiamente dicha, donde

OnCalendar indica fecha y hora en la que se debe disparar el **timer**. En este caso particular, será todos los días de todos los meses de todos los años a las 6 y 18:00 hs.

RandomizedDelaySec=12h especifica un intervalo de demora al azar que se puede llegar a aplicar al temporizador, entre 0 y el valor indicado.

Persistent=true nos dice que si systemd no disparó el timer en una ocasión previa (por ejemplo, si el equipo estaba apagado), lo hará de forma automática apenas sea posible.

---

*En el man page de **systemd.timer** podemos encontrar todas las directivas de configuración disponibles.*

En el caso de que no se muestre la directiva **Unit** (no confundir con la sección que lleva el mismo nombre), el temporizador controlará el servicio que tenga el mismo nombre. Veamos dos ejemplos en la siguiente imagen:

```
vpsadmin@localhost:~$ ls /lib/systemd/system | grep apt-daily
apt-daily.service
apt-daily.timer
apt-daily-upgrade.service
apt-daily-upgrade.timer
vpsadmin@localhost:~$
```

En otras palabras, **apt-daily.timer** y **apt-daily-upgrade.timer** controlarán los servicios relacionados (**apt-daily.service** y **apt-daily-upgrade.service**). Por suerte, todo esto sucede esto sin necesidad de especificarlo en el archivo de configuración del *timer*.

Registros de eventos del sistema

La mayoría (si no todos) de los servicios que se ejecutan en un sistema Linux originan algún tipo de registro como resultado de su operación. Ya sea que se trate de mensajes de error, eventos de inicio de sesión, o incluso cuestiones puramente informativas, es importante que sepamos dónde se guardan. Esto nos permitirá examinarlos cuando sea necesario en búsqueda de información importante.

Un mensaje de registro suele tener información sobre la seguridad del sistema, aunque puede contener cualquier información. Junto con cada mensaje se incluye la fecha y hora del envío.

El protocolo que se utiliza para la transmisión de estos mensajes recibe el nombre de **rsyslog**, al igual que la aplicación que se encarga de tal envío. Reemplazó a **syslog** en esta función como el estándar de facto en la mayoría de las distribuciones hace más de una década. El protocolo en sí es muy sencillo. Consiste en un servidor ejecutando el servicio relacionado, también llamado **rsyslog** (demonio de rsyslog) que registra los mensajes de texto plano (menos de 1024 bytes) enviados por cada aplicación.

Aunque **rsyslog** tiene algunos problemas de seguridad, su sencillez ha hecho que muchos dispositivos lo implementen, tanto para enviar como para recibir. Eso hace posible integrar mensajes de varios tipos de sistemas en un solo repositorio central (incluso provenientes de distintos equipos).

Veamos las siguientes líneas del archivo de configuración **/etc/rsyslog.conf** donde las líneas que comienzan con # son ignoradas por el servicio.

- **kern.*** indica todos los mensajes del kernel, sean de la severidad que sean.
- ***.info;mail.none;authpriv.none;cron.none** se utiliza para registrar todos los mensajes informativos (**info**) sin importar cuál sea el recurso que los emite (de ahí el uso del comodín al principio) con la excepción de aquellos cuyo origen sea **mail**, **authpriv**, y **cron**.
- Todos los mensajes de **authpriv** (autenticación privada) deben guardarse en **/var/log/secure**, mientras que para **mail** y **cron** el destino debe ser **/var/log/maillog** y **/var/log/cron**, respectivamente. Tal como en el primer ejemplo, el uso de * luego del punto nos dice que esta regla aplica para cualquier tipo de evento sin importar su severidad.

```
#### RULES ####
# Origen.Criticidad (Prioridad)                         Archivo_de_log
# Log all kernel messages to the console.
# Logging much else clutters up the screen.
#kern.*                                                 /dev/console

# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none;cron.none                /var/log/messages

# The authpriv file has restricted access.
authpriv.*                                              /var/log/secure

# Log all the mail messages in one place.
mail.*                                                  -/var/log/maillog


# Log cron stuff
cron.*                                                  /var/log/cron
```

*El signo **-** delante de **/var/log/maillog** (y otros) se utilizaba en versiones anteriores para omitir la sincronización con el archivo inmediatamente luego de cada escritura. Hoy es una opción en desuso.*

Los ejemplos que acabamos de analizar nos servirán para explicar la estructura del mensaje en la próxima sección.

Estructura del mensaje

El mensaje enviado por **rsyslog** se compone de tres campos:

1. Prioridad
2. Cabecera
3. Texto

Entre los tres campos anteriores no deben sumar más de 1024 bytes, pero no hay longitud mínima. Veamos ahora con un poco más de detalle el significado y posibilidades de cada uno de ellos:

- La **prioridad** indica tanto el *recurso* (origen) del mensaje como la *severidad* de este. Suele seguir la convención tomada de la RFC 3164. Los recursos más comunes son **auth**, **authpriv**, **cron**, **daemon**, **kern**, **lpr**, **mail**, y **local0** a **local7** para mensajes definidos por usuarios. La severidad (en orden de importancia) puede ser **debug**, **info**, **notice**, **warn**, **err**, **crit**, **alert**, y **emerg**. Por defecto, todos los mensajes de la severidad especificada y mayores serán registrados de acuerdo con lo que se indique.

En la imagen siguiente vemos que todos los mensajes provenientes de **local7** y aquellos con severidad igual o mayor a **notice** originados por **local1** se guardarán en **/var/log/boot.log** y **/var/log/mi.log**, respectivamente. Este último, en particular, fue agregado a mano para propósitos ilustrativos.

```
# Save boot messages also to boot.log
local7.*                                                /var/log/boot.log

local1.notice                                           /var/log/mi.log
```

- La **cabecera** indica tanto la fecha y hora del mensaje como el servidor (nombre de equipo desde donde se origina) y, junto con el **texto** explicativo, constituye la información que se guarda en el archivo de registro.

Con el comando `logger` podemos enviar mensajes personalizados a **rsyslog** manualmente o mediante scripts. Por ejemplo,

```
logger -p local1.notice "Este es un aviso"
logger -p local1.warn "Esta es una advertencia"
```

hará que se guarden en **/var/log/mi.log** (destino para **local1.notice** y de severidad mayor) los mensajes *Este es un aviso* y *Esta es una advertencia*, como vemos a continuación:

```
[gacanepa@server ~]$ logger -p local1.notice "Este es un aviso"
[gacanepa@server ~]$ logger -p local1.warn "Esta es una advertencia"
[gacanepa@server ~]$ tail -n 2 /var/log/mi.log
Oct 17 13:22:51 server gacanepa: Este es un aviso
Oct 17 13:23:00 server gacanepa: Esta es una advertencia
[gacanepa@server ~]$
```

En la imagen de arriba se puede apreciar la fecha y hora en la que sucedieron los eventos, el equipo que los originó (llamado **server** en este caso) y el usuario que los emitió (**gacanepa**).

Luego de hacer cambios en el archivo de configuración será necesario que reiniciemos el servicio mediante `systemctl restart rsyslog` para que estos tomen efecto.

Alternativamente, se pueden enviar los mensajes del sistema a:

- **otra máquina**, para almacenarlos remotamente. Para esto es necesario indicar en el campo de acción el nombre o dirección de dicho sistema precedido por dos

signos **@** en vez de una ruta a un archivo de registro local. Esto es útil si tenemos una máquina segura, en la que podemos confiar (conectada a la red). De esta manera, se guardaría allí una copia de los mensajes de nuestro sistema, la cual no podría ser modificada en caso de que alguien pudiera iniciar sesión en la máquina que los está generando. Esto requiere que el sistema remoto esté habilitado para tomar mensajes provenientes de otros equipos.

- **un script** que se ejecute cada vez que se origina un mensaje de un determinado tipo. En este caso debemos colocar el signo **^** delante de la ruta completa al script.

```
# Enviar a 192.168.0.10 mensajes local2.warn y de severidad superior
local2.warn  @@192.168.0.10:514

# Enviar a /opt/miscript.sh los mensajes local0.debug y superiores:
local0.debug  ^/opt/miscript.sh
```

La imagen anterior nos indica que los mensajes **local2.warn** y superiores serán enviados a **192.168.0.10**, mientras que **local0.debug** y aquellos de severidad mayor causarán que se ejecute **/opt/miscript.sh**.

Journalctl

Bajo **systemd**, el servicio encargado de recolectar y almacenar los mensajes del kernel y otras fuentes recibe el nombre de **journald**. Una característica distintiva de journald es que no almacena la información en texto plano. Por el contrario, lo hace de manera estructurada en archivos binarios dentro de **/run/log/format**. Esto facilita el identificar información relevante de manera rápida, aunque requiere el uso de la herramienta externa journalctl para interpretar los logs.

La ventaja que presenta el uso de `journalctl` es que nos permite centralizar la administración e inspección de los varios logs del sistema (muchas veces dispersos en **/var/log** o en alguno de sus subdirectorios como vimos antes) sin importar el proceso que los origine.

Por ejemplo, para visualizar los mensajes originados por las units **ssh** y **cups** en Debian usaremos la opción `-u` (de *unit*). Como vemos a continuación, la podemos repetir cuantas veces lo deseemos:

```
journalctl -u ssh -u cups
```

```
root@debiancla:~# journalctl -u ssh -u cups
-- Logs begin at Thu 2018-10-18 10:34:09 -03, end at Thu 2018-10-1
oct 18 10:34:21 debiancla systemd[1]: Started CUPS Scheduler.
oct 18 10:34:22 debiancla systemd[1]: Starting OpenBSD Secure Shel
oct 18 10:34:23 debiancla sshd[697]: Server listening on 0.0.0.0 p
oct 18 10:34:23 debiancla sshd[697]: Server listening on :: port 2
oct 18 10:34:23 debiancla systemd[1]: Started OpenBSD Secure Shell
oct 18 10:39:24 debiancla systemd[1]: Stopping CUPS Scheduler...
oct 18 10:39:24 debiancla systemd[1]: Stopped CUPS Scheduler.
oct 18 10:39:24 debiancla systemd[1]: Started CUPS Scheduler.
oct 18 10:45:48 debiancla sshd[1235]: Connection closed by 192.168
oct 18 10:46:14 debiancla sshd[1237]: Accepted password for alumno
oct 18 10:46:14 debiancla sshd[1237]: pam_unix(sshd:session): sess
root@debiancla:~#
```

Si deseamos tener más información de contexto, utilizaremos la opción combinada `-xu`. Esta alternativa nos resultará particularmente útil cuando el inicio de un servicio con `systemctl` falle por alguna razón, dándonos información valiosa para encontrar el origen del inconveniente.

Para salir de `journalctl`, debemos presionar la tecla q.

Además, con `journalctl --boot` podemos acceder a los datos correspondientes a un booteo en particular. La lista de inicios para los que se dispone de información puede verse reemplazando `--boot` con `--list-boots`. El argumento `-0` se utiliza para indicar el booteo indicado como **0**:

```
journalctl --list-boots # Lista de inicios disponibles
journalctl --boot -0 | head -n 5 # Primeras 5 líneas
journalctl --boot -0 | tail -n 5 # Últimas 5 líneas
```

```
root@debiancla:~# journalctl --list-boots
 0 e5e5358298c449e68502b5de5c473b96 Thu 2018-10-18 10:34:09 -03—Thu 2018-1
root@debiancla:~# journalctl --boot -0 | head -n 5
-- Logs begin at Thu 2018-10-18 10:34:09 -03, end at Thu 2018-10-18 11:17:
oct 18 10:34:09 debiancla kernel: Linux version 4.9.0-4-amd64 (debian-kern
0516 (Debian 6.3.0-18) ) #1 SMP Debian 4.9.65-3+deb9u1 (2017-12-23)
oct 18 10:34:09 debiancla kernel: Command line: BOOT_IMAGE=/vmlinuz-4.9.0-
ro quiet
oct 18 10:34:09 debiancla kernel: x86/fpu: Supporting XSAVE feature 0x001:
oct 18 10:34:09 debiancla kernel: x86/fpu: Supporting XSAVE feature 0x002:
root@debiancla:~# journalctl --boot -0 | tail -n 5
oct 18 11:04:01 debiancla anacron[1315]: Normal exit (0 jobs run)
oct 18 11:04:01 debiancla systemd[1]: anacron.timer: Adding 4min 9.674685s
oct 18 11:17:02 debiancla CRON[1339]: pam_unix(cron:session): session open
oct 18 11:17:02 debiancla CRON[1340]: (root) CMD (   cd / && run-parts --r
oct 18 11:17:02 debiancla CRON[1339]: pam_unix(cron:session): session clos
root@debiancla:~#
```

Para lograr que se guarden los registros de arranque del sistema de manera persistente deberemos habilitarlo de la siguiente forma:

```
mkdir /var/log/journal
systemd-tmpfiles --create --prefix /var/log/journal
systemctl restart systemd-journald
```

Otra manera de configurar la permanencia de los logs es a través del archivo **/etc/systemd/journald.conf** indicando **Storage=persistent** debajo de la sección **[Journal]**.