Find when cron.daily cron.weekly and cron.monthly run on Redhat / CentOS / Debian Linux and systemd-timers

Author : admin

The problem - Apache restart at random times

I've noticed today something that is occuring for quite some time but was out of my scope for quite long as I'm not directly involved in our Alert monitoring at my daily job as sys admin. Interestingly an Apache HTTPD webserver is triggering alarm twice a day for a short downtime that lasts for 9 seconds.

I've decided to investigate what is triggering WebServer restart in such random time and investigated on the system for any background running scripts as well as reviewed the system logs. As I couldn't find nothing there the only logical place to check was **cron jobs**.
The usual

```
crontab -u root -l
```

Had no configured cron jobbed scripts so I digged further to check whether there isn't cron jobs records

for a script that is triggering the reload of **Apache** in **/etc/crontab**
/var/spool/cron/root and **/var/spool/cron/httpd.**
Nothing was found there and hence as there was no **anacron** service running but **/usr/sbin/crond** the other expected place to look up for a trigger even was **/etc/cron***

1. Configured default cron execution times, every day, every hour every month

```
 # ls -ld /etc/cron.*
drwxr-xr-x 2 root root 4096 feb 27 10:54 /etc/cron.d/
drwxr-xr-x 2 root root 4096 dec 27 10:55 /etc/cron.daily/
drwxr-xr-x 2 root root 4096 dec  7 23:04 /etc/cron.hourly/
drwxr-xr-x 2 root root 4096 dec  7 23:04 /etc/cron.monthly/
drwxr-xr-x 2 root root 4096 dec  7 23:04 /etc/cron.weekly/
```

After a look up to each of above directories, finally I found the very expected **logrorate shell script** set to execute from **/etc/cron.daily/logrotate and inside it I've found after the log files were set to be gzipped and moved to execute WebServer restart with:**

```
systemctl reload httpd
```

My first reaction was to ponder seriously why the script is invoking **systemctl reload httpd** instead of the good oldschool

```
apachectl -k graceful
```

But it seems on Redhat and CentOS since RHEL / CentOS version 6.X onwards **systemctl reload httpd** is supposed to be identical and a substitute for **apachectl -k graceful.**
Okay the craziness of innovation continued as obviously the reload was causing a Downtime to be visible in the Zabbix HTTPD port Monitoring graph ...
Now as the problem was identified the other logical question poped up how to find out **what is the exact timing scheduled to run the script in that unusual random times each time ??**

2. Find out cron scripts timing Redhat / CentOS / Fedora / SLES

/etc/cron.{daily,monthly,weekly} placed scripts's execution method has changed over the years, causing a chaos just like many Linux standard things we know due to the inclusion of systemd and some other additional weird OS design changes. The result is the result explained above scripts are running at a strange unexpeted times ... one thing that was intruduced was **anacron -** which is also executing commands periodically with a different preset frequency. However it is considered more thrustworhty by **crond daemon**, because anacron does not assume the machine is continuosly running and if the machine is down due to a shutdown or a failure (if it is a Virtual Machine) or simply a **crond dies out,** some cronjob necessery for overall set environment or application might not run, what anacron guarantees is even though that and even if **crond** is in unworking defunct state, the preset scheduled scripts will still be served.
anacron's default file location is in **/etc/anacrontab.**

A standard /etc/anacrontab looks like so:

```
[root@centos ~]:# 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
```

START_HOURS_RANGE : The START_HOURS_RANGE variable sets the time frame, when the job could started.
The jobs will start during the 3-22 (3AM-10PM) hours only.

- **cron.daily** will run at 3:05 (After Midnight) A.M. i.e. run once a day at 3:05AM.
- **cron.weekly** will run at 3:25 AM i.e. run once a week at 3:25AM.
- **cron.monthly** will run at 3:45 AM i.e. run once a month at 3:45AM.

If the RANDOM_DELAY env var. is set, a random value between 0 and RANDOM_DELAY minutes will be added to the start up delay of anacron served jobs.
For instance RANDOM_DELAY equels 45 would therefore add, randomly, between 0 and 45 minutes to the user defined delay.

Delay will be *5 minutes + RANDOM_DELAY* for **cron.daily** for above **cron.daily, cron.weekly, cron.monthly** config records, i.e. 05:01 + 0-45 minutes

A full detailed explanation on automating system tasks on Redhat Enterprise Linux is worthy reading here.

!!! Note !!! that listed jobs will be running in queue. After one finish, then next will start.

3. SuSE Enterprise Linux cron jobs not running at desired times why?

in SuSE it is much more complicated to have a right timing for standard default cron jobs that comes preinstalled with a service

In older SLES release **/etc/crontab** looked like so:

```
  SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

  # run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

As time of writting article it looks like:

```
  SHELL=/bin/sh
PATH=/usr/bin:/usr/sbin:/sbin:/bin:/usr/lib/news/bin
MAILTO=root
#
# check scripts in cron.hourly, cron.daily, cron.weekly, and cron.monthly
#
-*/15 * * * *   root  test -x /usr/lib/cron/run-crons && /usr/lib/cron/run-crons >/dev/null 2>&1
```

This runs any scripts placed in **/etc/cron.{hourly, daily, weekly, monthly}** but it may not run them when you expect them to run.
/usr/lib/cron/run-crons compares the current time to the /var/spool/cron/lastrun/cron.{time} file to determine if those jobs need to be run.

For hourly, it checks if the current time is greater than (or exactly) 60 minutes past the timestamp of the **/var/spool/cron/lastrun/cron.hourly** file.

For weekly, it checks if the current time is greater than (or exactly) 10080 minutes past the timestamp of the **/var/spool/cron/lastrun/cron.weekly** file.

Monthly uses a caclucation to check the time difference, but is the same type of check to see if it has been one month after the last run.

Daily has a couple variations available - By default it checks if it is more than or exactly 1440 minutes since lastrun.
If DAILY_TIME is set in the /etc/sysconfig/cron file (again a suse specific innovation), then that is the time (within 15minutes) when daily will run.

For systems that are powered off at DAILY_TIME, daily tasks will run at the DAILY_TIME, unless it has been more than x days, if it is, they run at the next running of run-crons. (default 7days, can set shorter time in /etc/sysconfig/cron.)
Because of these changes, the first time you place a job in one of the /etc/cron.{time} directories, it will run the next time run-crons runs, which is at every 15mins (xx:00, xx:15, xx:30, xx:45) and that time will be the lastrun, and become the normal schedule for future runs. Note that there is the potential that your schedules will begin drift by 15minute increments.

As you see this is very complicated stuff and since God is in the simplicity it is much better to just not use /etc/cron.* for whatever scripts and manually schedule each of the system cron jobs and custom scripts with cron at specific times.

4. Debian Linux time start schedule for cron.daily / cron.monthly / cron.weekly timing

As the last many years many of the servers I've managed were running Debian GNU / Linux, my first place to check was **/etc/crontab** which is the standard cronjobs file that is setting the { **daily , monthly , weekly crons }**

```
debian:~# ls -ld /etc/cron.*
drwxr-xr-x 2 root root 4096 ??? 27 10:54 /etc/cron.d/
drwxr-xr-x 2 root root 4096 ??? 27 10:55 /etc/cron.daily/
drwxr-xr-x 2 root root 4096 ???  7 23:04 /etc/cron.hourly/
drwxr-xr-x 2 root root 4096 ???  7 23:04 /etc/cron.monthly/
drwxr-xr-x 2 root root 4096 ???  7 23:04 /etc/cron.weekly/
```

```
debian:~# cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
```

What above does is:

- Run cron.hourly once at every hour at 1:17 am
- Run cron.daily once at every day at 6:25 am.
- Run cron.weekly once at every day at 6:47 am.
- Run cron.monthly once at every day at 6:42 am.

As you can see if anacron is present on the system it is run via it otherwise it is run via **run-parts** binary command which is reading and executing one by one all scripts insude **/etc/cron.hourly**, **/etc/cron.weekly** , **/etc/cron.mothly** .

anacron - few more words

Anacron is the canonical way to run at least the jobs from /etc/cron.{daily,weekly,monthly) after startup, even when their execution was missed because the system was not running at the given time. Anacron does not handle any cron jobs from /etc/cron.d, so any package that wants its /etc/cron.d cronjob being executed by anacron needs to take special measures.

If anacron is installed, regular processing of the /etc/cron.d{daily,weekly,monthly} is omitted by code in /etc/crontab but handled by anacron via /etc/anacrontab. Anacron's execution of these job lists has changed multiple times in the past:

```
debian:~# cat /etc/anacrontab
# /etc/anacrontab: configuration file for anacron

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

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
HOME=/root
LOGNAME=root

# These replace cron's entries
1    5    cron.daily    run-parts --report /etc/cron.daily
7    10    cron.weekly    run-parts --report /etc/cron.weekly
@monthly    15    cron.monthly    run-parts --report /etc/cron.monthly
```

In wheezy and earlier, anacron is executed via init script on startup and via /etc/cron.d at 07:30. This causes the jobs to be run in order, if scheduled, beginning at 07:35. If the system is rebooted between midnight and 07:35, the jobs run after five minutes of uptime.
In stretch, anacron is executed via a systemd timer every hour, including the night hours. This causes the jobs to be run in order, if scheduled, beween midnight and 01:00, which is a significant change to the previous behavior.
In buster, anacron is executed via a systemd timer every hour with the exception of midnight to 07:00 where anacron is not invoked. This brings back a bit of the old timing, with the jobs to be run in order, if scheduled, beween 07:00 and 08:00. Since anacron is also invoked once at system startup, a reboot between midnight and 08:00 also causes the jobs to be scheduled after five minutes of uptime.
anacron also didn't have an upstream release in nearly two decades and is also currently orphaned in Debian.

As of 2019-07 (right after buster's release) it is planned to have cron and anacron replaced by **cronie.**

cronie - *Cronie was forked by Red Hat from ISC Cron 4.1 in 2007, is the default cron implementation in Fedora and Red Hat Enterprise Linux at least since Version 6. cronie seems to have an acive upstream, but is currently missing some of the things that Debian has added to vixie cron over the years. With the finishing of cron's conversion to quilt (3.0), effort can begin to add the Debian extensions to Vixie cron to cronie.*

Because cronie doesn't have all the Debian extensions yet, it is not yet suitable as a cron replacement, *so it is not in Debian.*

5. systemd-timers - The new crazy systemd stuff for script system job scheduling

Timers are systemd unit files with a suffix of .timer. systemd-timers was introduced with systemd so older Linux OS-es does not have it.
Timers are like other unit configuration files and are loaded from the same paths but include a [Timer] section which defines when and how the timer activates. Timers are defined as one of two types:

- Realtime timers (a.k.a. wallclock timers) activate on a calendar event, the same way that cronjobs do. The option OnCalendar= is used to define them.
- *Monotonic timers activate after a time span relative to a varying starting point. They stop if the computer is temporarily suspended or shut down. There are number of different monotonic timers but all have the form: OnTypeSec=. Common monotonic timers include OnBootSec and OnActiveSec.*

For each .timer file, a matching .service file exists (e.g. foo.timer and foo.service). The .timer file activates and controls the .service file. The .service does not require an [Install] section as it is the timer units that are enabled. If necessary, it is possible to control a differently-named unit using the Unit= option in the timer's [Timer] section.

systemd-timers is a complex stuff and I'll not get into much details but the idea was to give awareness of its existence for more info check its manual man systemd.timer

Its most basic use is to **list all configured systemd.timers, below is from my home Debian laptop**

```
debian:~# systemctl list-timers --all
NEXT                          LEFT       LAST                          PASSED       UNIT
ACTIVATES
Tue 2020-03-24 23:33:58 EET  18s left     Tue 2020-03-24 23:31:28 EET  2min 11s
ago laptop-mode.timer            lmt-poll.service
Tue 2020-03-24 23:39:00 EET  5min left     Tue 2020-03-24 23:09:01 EET  24min ago
 phpsessionclean.timer        phpsessionclean.service
```

Wed 2020-03-25 00:00:00 EET 26min left Tue 2020-03-24 00:00:01 EET 23h ago ***logrotate.timer logrotate.service***
Wed 2020-03-25 00:00:00 EET 26min left Tue 2020-03-24 00:00:01 EET 23h ago ***man-db.timer man-db.service***
Wed 2020-03-25 02:38:42 EET 3h 5min left Tue 2020-03-24 13:02:01 EET 10h ago ***apt-daily.timer apt-daily.service***
Wed 2020-03-25 06:13:02 EET 6h left Tue 2020-03-24 08:48:20 EET 14h ago ***apt-daily-upgrade.timer apt-daily-upgrade.service***
Wed 2020-03-25 07:31:57 EET 7h left Tue 2020-03-24 23:30:28 EET 3min 11s ago ***anacron.timer anacron.service***
Wed 2020-03-25 17:56:01 EET 18h left Tue 2020-03-24 17:56:01 EET 5h 37min ago ***systemd-tmpfiles-clean.timer systemd-tmpfiles-clean.service***

8 timers listed.

N ! B! If a timer gets out of sync, it may help to delete its **stamp-*** file in **/var/lib/systemd/timers** (or **~/.local/share/systemd/** in case of user timers). These are zero length files which mark the last time each timer was run. If deleted, they will be reconstructed on the next start of their timer.

Summary

?In this article, I've shortly explain logic behind debugging weird restart events etc. of Linux configured services such as Apache due to configured scripts set to run with a predefined scheduled job timing. I shortly explained on how to figure out why the preset default install configured cron jobs such as **logrorate - the service that is doing system logs archiving and nulling** run at a certain time. I shortly explained the mechanism behind cron.{daily, monthy, weekly} and its execution via anacron - runner program similar to crond that never misses to run a scheduled job even if a system downtime occurs due to a crashed Docker container etc. **run-parts** command's use was shortly explained. A short look at **systemd.timers** was made which is now essential part of almost every new Linux release and often used by system scripts for scheduling time based maintainance tasks.