

PRÁCTICA 3: Monitorización y “Profiling”

Índice

1	Introducción	4
2	Monitores para Hardware	4
2.1	Mensajes del kernel: dmesg	4
3	Monitores para Software	4
3.1	Subsistema de archivos	4
3.2	Monitorizando un servicio o ejecución de un programa: strace	5
4	Monitores generales	5
4.1	Munin	5
4.2	Nagios	6
4.3	Ganglia	6
4.4	ZABBIX	6
4.5	Cacti	6
4.6	AWstats	7
5	Automatización	7
5.1	cron y systemd	7
5.2	Scripts	8
5.2.1	Shell y comandos del sistema: grep, find, awk y sed	8
5.2.2	Python y PHP	8
5.3	A nivel de Plataforma: Ansible	9
6	Profiling	9
6.1	Scripts	9
6.2	SQL	9

LECCIÓN1: Monitorización del RAID1, Monitores y Automatización

1	Introducción	4
2	Monitores para Hardware	4
2.1	Mensajes del kernel: dmesg	4
3	Monitores para Software	4
3.1	Subsistema de archivos	4
3.2	Monitorizando un servicio o ejecución de un programa: strace	5
4	Monitores generales	5
4.1	Munin	5
4.2	Nagios	6
4.3	Ganglia	6
4.4	ZABBIX	6
4.5	Cacti	6
4.6	AWstats	7

1) Introducción

Esta práctica consiste en la utilización de herramientas de monitorización del sistema para visualizar cómo se comporta el sistema ante ciertas actividades que los usuarios u otros servicios generan.

Las herramientas de monitorización presentan medidas del sistema permitiendo generar informes e históricos que puedan ser de utilidad para un análisis a posteriori (off-line) o para tomar decisiones sobre la marcha (on-line).

En última instancia, el objetivo de esta práctica es monitorizar varios sistemas que tienen servicios instalados siendo consciente de cómo se realiza el proceso tanto a bajo nivel como usando herramientas que nos permiten trabajar a alto nivel.

2) Monitores para Hardware

Además del estado del software del servidor, también existen programas que nos permiten ver el estado del hardware de nuestra máquina. En primer lugar, muchas BIOS (Basic Input Output System) (ya en extinción debido a la aparición de los Universal Extended Firmwares, UEFI) nos permiten acceder a cierta información sobre el estado del HW, sin embargo, para no tener que reiniciar, podemos utilizar otras herramientas. Concretamente, para Linux está: `hddtemp` para la temperatura del HD tenemos y el proyecto `lm-sensors`: <https://github.com/lm-sensors/lm-sensors> (con su correspondiente GUI: `xsensors`).

Al estar trabajando con máquinas virtuales, la ejecución y obtención de resultados de estos monitores no aporta nada ya que el hardware que monitorizan es virtual.

No obstante, debemos ser conscientes de que hay ciertos comandos que ya hemos visto que nos permiten listar el hardware disponible: `lspci`, `lsusb`, `lshw` nos muestran los dispositivos conectados a los buses e información general sobre el HW del equipo. Pruebe a ejecutarlos y ver qué muestran.

2.1. Mensajes del kernel: dmesg

El kernel de Linux permite conocer qué actividad ha ocurrido gracias a los mensajes que proporciona el kernel. Esto es especialmente útil para detectar problemas con el HW o periféricos.

3) Monitores para Software

3.1. Subsistema de archivos

En linux (UNIX) todo se manipula a través de archivos de una manera cómoda y transparente. Existe un directorio especial: `/proc` (visto en clase de teoría) y `/var` (algo se ha comentado también al respecto) que nos pueden dar información tanto del hardware como del software.

En estos directorios se encuentran archivos fundamentales en la monitorización del comportamiento del sistema, de las aplicaciones y de los usuarios. Cabe destacar el subdirectorio `/var/log` que contiene los archivos en los que se van volcando los logs de los servicios y algunas aplicaciones.

Hay que tener ciertas precauciones ya que, una mala gestión de los archivos de bitácora puede resultar en un sistema caído. Para evitar que los logs crezcan indefinidamente, se realiza una rotación de estos archivos (`logrotate`).

Gracias a estos archivos podemos identificar errores y problemas, además esta tarea puede ser automatizada. Como ejemplo retomaremos los RAID1 en Ubuntu server de la primera práctica.

Usted debe conocer cómo identificar, monitorizar y reconstruir los discos RAID que tiene configurados en sus máquinas virtuales. Puede encontrar información imprescindible en las páginas del manual para `mdadm` y el archivo `/proc/mdstat` además de en [David Greaves *et al.*, 2013](#).

3.2. Monitorizando un servicio o ejecución de un programa: `strace`

Hay un conjunto de programas que permiten hacer una traza de las llamadas al sistema realizadas por un programa (servicio) en ejecución, p.ej. `strace` (system call tracer).

Este tipo de programas pueden ser útiles de cara a detectar problemas que no se muestran en los archivos de “log”.

En <http://chadfowler.com/2014/01/26/the-magic-of-strace.html> tiene ejemplos de uso y podrá apreciar la utilidad de este tipo de programas.

4) Monitores generales

Además de los comandos integrados vistos en teoría y los que han usado en prácticas, existen otros programas muy populares que permiten monitorizar el sistema.

Hay algunos de entorno corporativo de grandes empresas como NetApp (<http://www.netapp.com/es/products/management-software/>), aunque los que se verán en las siguientes subsecciones también son usados por grandes instituciones y empresas.

4.1. Munin

Munin (significa memoria) está disponible en <http://munin-monitoring.org/>.

Para su instalación (en CentOS) puede hacerlo compilando el código fuente (como ha podido hacer con `lm_sensors`) alojado en la página o a través del paquete disponible en el repositorio EPEL (<http://fedoraproject.org/wiki/EPEL/es>). “¿Cómo puedo utilizar estos paquetes adicionales?” es el título de la subsección dentro de la página donde se explica cómo activar el repositorio que contiene los paquetes. Tan solo debe instalar un `.rpm` y podrá usar `yum` para instalar `munin`.

Para Ubuntu, está disponible sin tener que añadir ningún repositorio adicional.

4.2. Nagios

Es otro software muy usado para monitorizar sistemas. Recientemente ha pasado por una transformación de proyecto de la comunidad a proyecto empresarial. Debido a esto, ha aparecido Naemon <http://naemon.org>.

4.3. Ganglia

Es un proyecto alojado en <http://ganglia.sourceforge.net/> que monitoriza sistemas de cómputo distribuidos (normalmente para altas prestaciones) y permite una gran escalabilidad.

Como puede verse en su página, importantes instituciones y compañías lo usan (p.ej. UC Berkely, MIT, Twitter, ...).

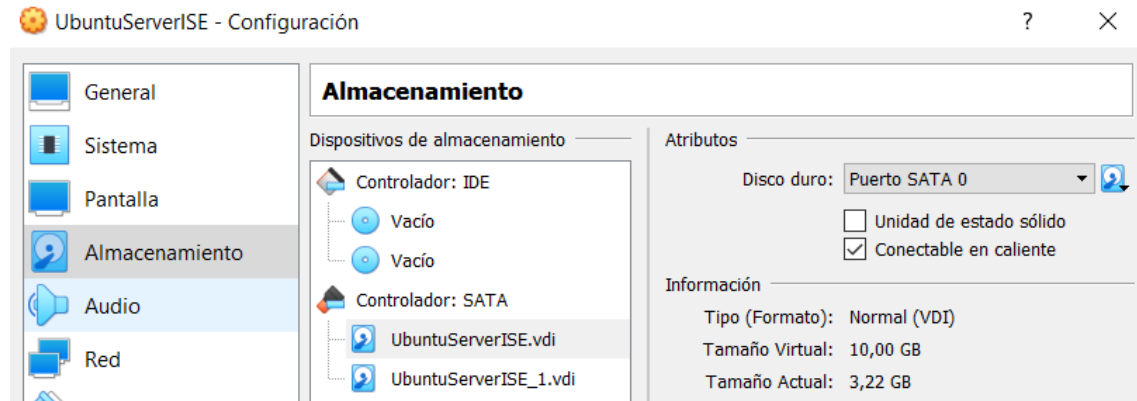
4.5. Cacti

Este monitor (<http://www.cacti.net/>) es otro front-end para RRDtool (<http://oss.oetiker.ch/rrdtool/>) que permite monitorizar muchos parámetros sin sobrecargar excesivamente el sistema.

4.6. AWstats

Es un monitor de servicios específicos de servidores web p.ej.: HTTP, FTP, correo. La página del proyecto es <http://awstats.sourceforge.net/>. Se puede compilar el código fuente aunque están disponibles los paquetes en los repositorios.

Primeramente, vamos a simular que mientras estamos trabajando se nos rompe uno de los discos duros del RAID de Ubuntu (y después veremos cómo recuperarlo). Para ello (y con la máquina apagada) debemos habilitar la opción de *Conectable en caliente* desde la configuración del almacenamiento de la máquina virtual para poder quitar el disco duro mientras la máquina está encendida.



Configuración de la maquina antes de tocar nada.

```

pero@pero:~$ lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
loop0                               7:0      0  55.3M 1 loop  /snap/core18/1885
loop1                               7:1      0  55.4M 1 loop  /snap/core18/1932
loop2                               7:2      0  70.6M 1 loop  /snap/lxd/16922
loop3                               7:3      0  67.8M 1 loop  /snap/lxd/18150
loop4                               7:4      0   31M 1 loop  /snap/snapd/9721
loop5                               7:5      0  30.3M 1 loop  /snap/snapd/9279
sda                                 8:0      0   10G  0 disk
├─sda1                             8:1      0    1M  0 part
├─sda2                             8:2      0  300M  0 part
│   └─md0                          9:0      0  299M  0 raid1
│       └─md0p1                    259:0    0  294M  0 part  /boot
├─sda3                             8:3      0   9.7G  0 part
│   └─md1                          9:1      0   9.7G  0 raid1
│       └─dm_crypt-0                253:0    0   9.7G  0 crypt
│           └─vg0-lv--0--root        253:1    0    8G  0 lvm    /
│               └─vg0-lv--0--home    253:2    0    1G  0 lvm    /home
│                   └─vg0-lv--0--swap 253:3    0  692M  0 lvm    [SWAP]
└─sdb                               8:16     0   10G  0 disk
    ├─sdb1                         8:17     0    1M  0 part
    ├─sdb2                         8:18     0  300M  0 part
    │   └─md0                      9:0      0  299M  0 raid1
    │       └─md0p1                259:0    0  294M  0 part  /boot
    ├─sdb3                         8:19     0   9.7G  0 part
    │   └─md1                      9:1      0   9.7G  0 raid1
    │       └─dm_crypt-0            253:0    0   9.7G  0 crypt
    │           └─vg0-lv--0--root    253:1    0    8G  0 lvm    /
    │               └─vg0-lv--0--home 253:2    0    1G  0 lvm    /home
    │                   └─vg0-lv--0--swap 253:3    0  692M  0 lvm    [SWAP]
└─sr0                             11:0     1 1024M  0 rom
sr1                               11:1     1 1024M  0 rom

```

Iniciamos. Y una vez que se inicie, volvemos a la configuración y quitamos el disco duro (ControladorSATA).

Al quitarlo, nos da fallo en *sda3* y *sda2* (los que están en RAID) y nos dice que el sistema continúa únicamente con un disco duro (*sdb*).

```
peroj@peroj:~$ [ 300.482329] sd 2:0:0:0: rejecting I/O to offline device
[ 300.482353] blk_update_request: I/O error, dev sda, sector 618504 op 0x1:(WRITE) flags 0x800 phys
_seg 1 prio class 0
[ 300.482861] md: super_written gets error=10
[ 300.483124] md/raid1:md1: Disk failure on sda3, disabling device.
[ 300.483124] md/raid1:md1: Operation continuing on 1 devices.
[ 305.138010] md/raid1:md0: Disk failure on sda2, disabling device.
[ 305.138010] md/raid1:md0: Operation continuing on 1 devices.

peroj@peroj:~$ lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
loop0                               7:0      0  55.3M  1 loop  /snap/core18/1885
loop1                               7:1      0  55.4M  1 loop  /snap/core18/1932
loop2                               7:2      0  70.6M  1 loop  /snap/lxd/16922
loop3                               7:3      0  67.8M  1 loop  /snap/lxd/18150
loop4                               7:4      0   31M  1 loop  /snap/snapd/9721
loop5                               7:5      0  30.3M  1 loop  /snap/snapd/9279
sdb                                 8:16      0   10G  0 disk
├─sdb1                             8:17      0    1M  0 part
├─sdb2                             8:18      0  300M  0 part
├─┬md0                             9:0      0  299M  0 raid1
├─┬md0p1                           259:0    0  294M  0 part  /boot
├─┬sdb3                             8:19      0   9.7G  0 part
├─┬md1                             9:1      0   9.7G  0 raid1
├─┬dm_crypt-0                       253:0    0   9.7G  0 crypt
├─┬vg0-lv--0--root                 253:1    0    8G  0 lvm    /
├─┬vg0-lv--0--home                 253:2    0    1G  0 lvm    /home
├─┬vg0-lv--0--swap                 253:3    0  692M  0 lvm    [SWAP]
└─sr0                             11:0     1  1024M  0 rom
sr1                               11:1     1  1024M  0 rom
```

Como estamos en RAID (y tenemos lo mismo en el *sda* que en el *sdb*), si se nos rompe el disco duro a, el sistema sigue funcionando.

Pero, ahora bien, si restauramos a la instantánea anterior. Y en vez de quitarlo en caliente, lo quitamos ahora antes de iniciar. Y iniciamos el sistema con un disco duro. En este caso, no puede procesar el grupo de volúmenes y quedará esperando (intentando recuperar) hasta conseguir iniciar con un solo disco duro (tardará como mucho 5 minutos hasta que el sistema se da cuenta de que no puede recuperar la información).

```
[ 3.958000] raid0: using avx2xz2 recovery algorithm
[ 3.958293] xor: automatically using best checksumming function  avx
[ 3.959809] async_tx: api initialized (async)
done.
Begin: Running /scripts/init-premount ... done.
Begin: Mounting root file system ... Begin: Running /scripts/local-top ...   Volume group "vg0" not
found
  Cannot process volume group vg0
  Volume group "vg0" not found
  Cannot process volume group vg0
cryptsetup: Waiting for encrypted source device
  UUID=a9a936b2-074b-4ded-b9e9-730bf54352ac...
```

En este caso, pasado el tiempo de espera, se iniciará por defecto una herramienta del sistema llamada *initramfs*, que es un sistema de ficheros que se monta en memoria RAM para que se intente arreglar el error de inicio (dentro de esta herramienta se cambia la distribución de teclado, por ejemplo, el `_` es la `/`). Contiene las herramientas justas para gestionar el sistema para arreglar el sistema cuando resulta dañado (podemos verlas pulsando dos veces el `tab`). En este caso, sabemos que falla, pero en otros casos con esta herramienta no bastará.


```

Dropping to a shell.

BusyBox v1.30.1 (Ubuntu 1:1.30.1-4ubuntu6.2) built-in shell (ash)
Enter 'help' for a list of built-in commands.

(initramfs)
[
[[
acpid          dmesg          kbd_mode       ntfs-3g        sort
arch           dhclient       kill            nuke           stat
ash            dhclient-script kmod           openvt         static-sh
awk           dmsetup        ln             pdata_tools   stty
basename       du             loadfont       pidof          switch_root
blkid          dumpe2fs       loadkeys       pivot_root     sync
blockdev       dumpkmap       loadmap        plymouth       tail
btrfs          echo           losetup        plymouthd      tee
cache_check    egrep          ls             poweroff       test
chmod          env            lvm            printf         thin_check
chroot         expr           lzop           ps             touch
chvt           fbset          mdadm          pwd            tr
clear          fgrep          mdmon          readlink       true
cmp            find           minips         reboot         tty
cp             fold           mkdir          reset          udevadm
cpio           fstrim         mke2fs         resume         umount
cryptroot-unlock gzip           modinfo        rm             uname
cryptsetup     halt           modprobe       rmdir          uniq
cut            head           more           rmmount        vgchange
date           hostname      mount          run-init       wait-for-root
dd             hwclock       mount.fuse      run-parts      wc
dealloct       ifconfig      mount.ntfs     sed            wget
deluser        ip            modprobe       seq            which
devmem         ipconfig      mv             setfont        wipefs
df             iscsistart    nfsmount       sg_inq         yes
(initramfs)

```

También podemos ver la función **dmesg** que nos da información sobre el sistema, sobre todo el sistema, a nivel de hardware también te da la información de todos los mensajes que el hardware devuelve.

Como sabemos que lo que falla el RAID, lo primero que vamos a hacer es ver el fichero **/proc/mdstat** que contiene la información del RAID. Vemos que tanto md0 como md1 están inactivos porque no se han podido arrancar.

```

(initramfs) cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : inactive sda2[0](S)
      306176 blocks super 1.2

md1 : inactive sda3[0](S)
      10166272 blocks super 1.2

unused devices: <none>
(initramfs) _

```

Como hemos visto antes, cuando hemos quitado el disco duro en caliente en Ubuntu, con un disco duro se puede arrancar el sistema y así el RAID continuar. Para forzar esta situación y que el sistema continúe arrancando con un único RAID usamos **mdadm -R /dev/md0** y **mdadm -R /dev/md1** (el - está en la ¿).

Y vemos que lo primero que nos dice es que el RAID1 md0 está activo con uno de los dos discos duros de espejo. Y volvemos a visualizar el fichero para ver qué se han hecho efectivos los cambios, ya que en este fichero es donde podemos ver cuántos discos duros están caídos en los corchetes **[2/1]**, de dos está activo uno, y en los corchetes **[_U]** donde la **_** indica que está caído el primero, es decir, **sda** y la **U** (up) indica que el segundo disco duro(**sdb**) está activo.

```
(initramfs) mdadm -R /dev/md0
[ 1148.047943] md/raid1:md0: active with 1 out of 2 mirrors
[ 1148.048286] md0: detected capacity change from 0 to 313524224
mdadm: started array /dev/md0
(initramfs) [ 1148.056764] md0: p1
(initramfs) mdadm -R /dev/md1
[ 1218.482418] md/raid1:md1: active with 1 out of 2 mirrors
[ 1218.482777] md1: detected capacity change from 0 to 10410262528
mdadm: started array /dev/md1
(initramfs) cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active (auto-read-only) raid1 sda2[0]
      306176 blocks super 1.2 [2/1] [_U]

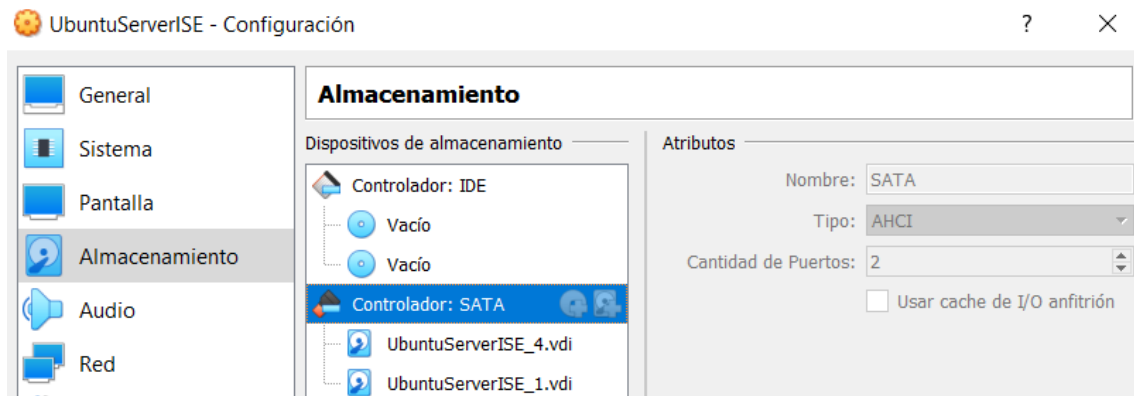
md1 : active (auto-read-only) raid1 sda3[0]
      10166272 blocks super 1.2 [2/1] [_U]

unused devices: <none>
```

Hacemos **exit** para salirnos del *initramfs*. Y vemos que nos pide la contraseña del volumen encriptado, por lo que ya ha conseguido organizarse para iniciar con un disco duro. Una vez recuperado el sistema sin perder información debemos arreglar el RAID. Comprobamos cuando entramos al sistema con **lsblk** que ahora estamos con un disco duro y que **sdb** ha pasado a ser nuestro **sda**.

```
Last login: Mon Nov  9 18:23:03 UTC 2020 on tty1
peroj@peroj:~$ lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0                               7:0      0  55.3M  1 loop /snap/core18/1885
loop1                               7:1      0  55.4M  1 loop /snap/core18/1932
loop2                               7:2      0  70.6M  1 loop /snap/lxd/16922
loop3                               7:3      0  67.8M  1 loop /snap/lxd/18150
loop4                               7:4      0  30.3M  1 loop /snap/snapd/9279
loop5                               7:5      0   31M   1 loop /snap/snapd/9721
sda                                 8:0      0   10G   0 disk
├─sda1                             8:1      0    1M   0 part
├─sda2                             8:2      0  300M   0 part
│   └─md0                          9:0      0  299M   0 raid1
│       └─md0p1                    259:0     0  294M   0 part /boot
├─sda3                             8:3      0   9.7G   0 part
│   └─md1                          9:1      0   9.7G   0 raid1
│       └─dm_crypt-0               253:0     0   9.7G   0 crypt
│           ├─vg0-lv--0--root      253:1     0    8G   0 lvm /
│           ├─vg0-lv--0--home      253:2     0    1G   0 lvm /home
│           └─vg0-lv--0--swap      253:3     0  692M   0 lvm [SWAP]
sr0                                 11:0     1 1024M   0 rom
sr1                                 11:1     1 1024M   0 rom
```


Ahora deberíamos conectar nuestro disco duro nuevo para reemplazar el estropeado. Esto lo vamos a simular insertando en caliente en el sistema un nuevo disco duro.



Apagamos a máquina por si hubiera problema por incompatibilidad en los puertos SATA de los discos duros. Debemos asegurarnos de que el disco duro que contiene la información está en el puerto SATA 0 para que el sistema lo pueda tomar de arranque (y el nuevo en el SATA 1). Iniciamos la máquina de nuevo y debería arrancar sin problemas con un único disco duro.

Con **lsblk** vemos que el disco duro nuevo lo tenemos vacío (con 10GB). Con **mdadm --add /dev/md0 /dev/sdb** podíamos añadir un RAID a un disco duro, pero el problema es que tenemos dos RAID y evidentemente no se puede hacer esto así, primero, porque el disco duro *sda* tiene 3 particiones (*sda1* 1MB para el grub del sistema, *sda2* 300MB para el RAID de *boot* y *sda3* 9'7GB para el RAID "gordo" del sistema). Por esto no podríamos usar ese comando para hacer esto porque le estaríamos diciendo que le añada a *md0* todo el disco duro *sdb*.

```

peroj@peroj:~$ lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
loop0                              7:0      0  55.3M  1 loop /snap/core18/1885
loop1                              7:1      0  55.4M  1 loop /snap/core18/1932
loop2                              7:2      0  70.6M  1 loop /snap/lxd/16922
loop3                              7:3      0  67.8M  1 loop /snap/lxd/18150
loop4                              7:4      0  30.3M  1 loop /snap/snapd/9279
loop5                              7:5      0   31M   1 loop /snap/snapd/9721
sda                                 8:0      0   10G   0 disk
├─sda1                             8:1      0    1M   0 part
├─sda2                             8:2      0  300M   0 part
│   └─md0                          9:0      0  299M   0 raid1
│       └─md0p1                    259:0     0  294M   0 part /boot
└─sda3                             8:3      0   9.7G   0 part
    └─md1                          9:1      0   9.7G   0 raid1
        └─dm_crypt-0              253:0     0   9.7G   0 crypt
            └─vg0-lv--0--root      253:1     0    8G   0 lvm    /
                └─vg0-lv--0--home  253:2     0    1G   0 lvm    /home
                    └─vg0-lv--0--swap 253:3     0   692M  0 lvm    [SWAP]
sdb                                 8:16     0   10G   0 disk
sr0                                11:0     1 1024M   0 rom
sr1                                11:1     1 1024M   0 rom

```

Entonces debemos de hacer las particiones de *sdb* a mano, hacer 3 particiones y después añadir a cada partición el RAID que les corresponde. Esto lo hacemos con la herramienta **fdisk**.

1º- Con **fdisk /dev/sdb** entramos a la consola de la herramienta para formatear *sdb*.

2º- Para añadir una nueva partición pulsamos **n**. Para que sea una partición primaria **p**. Y establecemos el número de la partición **1**.

3º- Para lo siguiente debemos usar una regla de tres para calcular los sectores (2048-20971519). Algo así como, si 20971519 total de sectores --> 10GB.... Entonces, empezando en 2048 debemos recordar que la primera partición tiene 1MB, que es más o menos 2048 sectores, entonces habría que pasar de 2048 a 4096 (ya que si empezamos en 2048 el último sector sería el 4096).

```
root@peroj:/home/peroj# fdisk /dev/sdb
Welcome to fdisk (util-linux 2.34).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0xaece785c.

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-20971519, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-20971519, default 20971519): 4096
Created a new partition 1 of type 'Linux' and of size 1 MiB.
```

4º)- Si no fuésemos a hacer más particiones deberíamos guardar la creada hasta ahora pulsando **w**. Pero como vamos a seguir, volvemos al paso 1º para crear **sdb2** y en este caso asignaremos el número de partición a **2** (que era de 300MB). Entonces como nos hemos quedado en el 4096, debemos empezar en el 4097 los sectores (eliminando así el mega que se van poniendo entre sector y sector por defecto).

```
Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (2-4, default 2): 2
First sector (4097-20971519, default 6144): 4097
Last sector, +/-sectors or +/-size{K,M,G,T,P} (4097-20971519, default 20971519): 618496
Created a new partition 2 of type 'Linux' and of size 300 MiB.
```

5º) Del mismo modo creamos la partición **3**. Dejando por defecto el apartado de los sectores, para así comprobar cuando se cree que tiene exactamente los 9'7GB que le corresponden.

6º) Pulsamos a **w** para crear las particiones a nivel físico y salir de **fdisk**.

```
Command (m for help): n
Partition type
   p   primary (2 primary, 0 extended, 2 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (3,4, default 3): 3
First sector (618497-20971519, default 620544):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (620544-20971519, default 20971519):
Created a new partition 3 of type 'Linux' and of size 9.7 GiB.
Command (m for help): w
```

```

The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.

root@peroj:/home/peroj# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
loop0                               7:0      0 55.3M 1 loop  /snap/core18/1885
loop1                               7:1      0 55.4M 1 loop  /snap/core18/1932
loop2                               7:2      0 70.6M 1 loop  /snap/lxd/16922
loop3                               7:3      0 67.8M 1 loop  /snap/lxd/18150
loop4                               7:4      0 30.3M 1 loop  /snap/snapd/9279
loop5                               7:5      0  31M 1 loop  /snap/snapd/9721
sda                                  8:0      0   10G 0 disk
├─sda1                              8:1      0    1M 0 part
├─sda2                              8:2      0  300M 0 part
│   └─md0                          9:0      0  299M 0 raid1
│       └─md0p1                    259:0     0  294M 0 part  /boot
├─sda3                              8:3      0   9.7G 0 part
│   └─md1                          9:1      0   9.7G 0 raid1
│       └─dm_crypt-0               253:0     0   9.7G 0 crypt
│           ├─vg0-lv--0--root      253:1     0    8G 0 lvm    /
│           ├─vg0-lv--0--home      253:2     0    1G 0 lvm    /home
│           └─vg0-lv--0--swap      253:3     0  692M 0 lvm    [SWAP]
└─sdb                               8:16     0   10G 0 disk
    ├─sdb1                         8:17     0    1M 0 part
    ├─sdb2                         8:18     0  300M 0 part
    └─sdb3                         8:19     0   9.7G 0 part
sr0                                 11:0     1 1024M 0 rom
sr1                                 11:1     1 1024M 0 rom
root@peroj:/home/peroj# _

```

Ahora ya podemos sincronizar los RAID ya que ahora le indicamos a **mdadm** que a **md0** le añade **sdb2** y que a **md1** le añade el **sdb3**. Entonces para sincronizar el RAID de *boot* usamos **mdadm -add /dev/md0 /dev/sdb2** y para el RAID del sistema usamos **mdadm --add /dev/md1 /dev/sdb3**.

```

root@peroj:/home/peroj# mdadm --add /dev/md0 /dev/sdb2
mdadm: added /dev/sdb2
root@peroj:/home/peroj# mdadm --add /dev/md1 /dev/sdb3
mdadm: added /dev/sdb3

```

Una vez ejecutado esto, podemos ver cada segundo en el fichero de configuración de los RAID con **watch -n 1 cat /proc/mdstat** como se van restaurando el sistema y resincronizando los espejos.

```

Every 1.0s: cat /proc/mdstat                                peroj: Sun Nov 15 18:49:40 2020

Personalities : [raid1] [linear] [multipath] [raid0] [raid6] [raid5] [raid4] [raid10]
md0 : active raid1 sdb2[2] sda2[0]
      306176 blocks super 1.2 [2/2] [UU]

md1 : active raid1 sdb3[2] sda3[0]
      10166272 blocks super 1.2 [2/1] [U_]
      [====>.....] recovery = 28.9% (2939328/10166272) finish=3.3min speed=36313K/sec

unused devices: <none>

```

En cuanto acabe, ya tendríamos el sistema completamente recuperado con nuestro disco duro nuevo como si nada hubiese pasado.

```
Every 1.0s: cat /proc/mdstat                                peroj: Sun Nov 15 18:52:30 2020

Personalities : [raid1] [linear] [multipath] [raid0] [raid6] [raid5] [raid4] [raid10]
md0 : active raid1 sdb2[2] sda2[0]
      306176 blocks super 1.2 [2/2] [UU]

md1 : active raid1 sdb3[2] sda3[0]
      10166272 blocks super 1.2 [2/2] [UU]

unused devices: <none>
```

```
root@peroj:/home/peroj# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
loop0                               7:0      0  55.3M  1 loop  /snap/core18/1885
loop1                               7:1      0  55.4M  1 loop  /snap/core18/1932
loop2                               7:2      0  70.6M  1 loop  /snap/lxd/16922
loop3                               7:3      0  67.8M  1 loop  /snap/lxd/18150
loop4                               7:4      0  30.3M  1 loop  /snap/snapd/9279
loop5                               7:5      0   31M   1 loop  /snap/snapd/9721
sda                                 8:0      0   10G   0 disk
├─sda1                             8:1      0    1M   0 part
├─sda2                             8:2      0   300M  0 part
│   └─md0                          9:0      0   299M  0 raid1
│       └─md0p1                    259:0    0   294M  0 part  /boot
├─sda3                             8:3      0   9.7G  0 part
│   └─md1                          9:1      0   9.7G  0 raid1
│       └─dm_crypt-0              253:0    0   9.7G  0 crypt
│           ├─vg0-lv--0--root      253:1    0    8G   0 lvm    /
│           ├─vg0-lv--0--home      253:2    0    1G   0 lvm    /home
│           └─vg0-lv--0--swap      253:3    0   692M  0 lvm    [SWAP]
└─sdb                               8:16     0   10G   0 disk
    ├─sdb1                         8:17     0    1M   0 part
    ├─sdb2                         8:18     0   300M  0 part
    │   └─md0                      9:0      0   299M  0 raid1
    │       └─md0p1                259:0    0   294M  0 part  /boot
    ├─sdb3                         8:19     0   9.7G  0 part
    │   └─md1                      9:1      0   9.7G  0 raid1
    │       └─dm_crypt-0          253:0    0   9.7G  0 crypt
    │           ├─vg0-lv--0--root  253:1    0    8G   0 lvm    /
    │           ├─vg0-lv--0--home  253:2    0    1G   0 lvm    /home
    │           └─vg0-lv--0--swap  253:3    0   692M  0 lvm    [SWAP]
    └─sr0                          11:0     1  1024M  0 rom
        └─sr1                      11:1     1  1024M  0 rom
```

LECCIÓN2:

5 Automatización	7
5.1 cron y systemd	7
5.2 Scripts	8
5.2.1 Shell y comandos del sistema: grep, find, awk y sed	8
5.2.2 Python y PHP	8
5.3 A nivel de Plataforma: Ansible	9
6 Profiling	9
6.1 Scripts	9
6.2 SQL	9

5) Automatización

5.1. cron y systemd

Tradicionalmente, el servicio cron ha permitido ejecutar cada cierto intervalo de tiempo una tarea concreta. Esto es muy útil de cara a recopilar información o monitorizar el sistema realizando una tarea concreta y lanzar alertas como, por ejemplo, enviar un correo electrónico cuando la carga esté por encima de un valor determinado (aunque algunos comandos de monitorización permitan hacer esa tarea directamente).

Con la introducción de systemd, cron puede seguir siendo usado pero su funcionamiento está basado en los *timers* que activan *services*. La documentación más recomendable sobre la gestión de systemd y los servicios está en los manuales de Red Hat, concretamente en https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/sect-managing_services_with_systemd-unit_files, aunque en https://people.redhat.com/pladd/systemd_NYRHUG_2016-03.pdf tiene un resumen con más enlaces a documentación.

Como ejemplo, podemos tomar este script escrito en Python que nos permite monitorizar si algún dispositivo de nuestro RAID ha fallado (puede ver más ejemplos en http://echorand.me/site/notes/articles/python_linux/article.html):

```
import re
f=open( '/proc/mdstat ' )
for line in f:
    b=re.findall( ' \[ [U]* [_]+ [U]* \] ', line )
    if (b!=[]):
        print ( "—ERROR en RAID—" )
print ( "—OK Script —" )
```

Podemos automatizar la ejecución de este script en systemd definiendo un *timer* dentro del directorio `/etc/systemd/system/` que se encarga de gestionar un servicio. Por tanto, hemos de crear dos archivos: `mon_raid.timer` y `mon_raid.service`

```
[Unit]
Description=Monitor RAID status
[Timer]
OnCalendar=minutely
[Install]
WantedBy=timers.target

[Unit]
```

```
Description=Monitor RAID status
[Service]
Type=simple
ExecStart=/usr/bin/python3 /home/alberto/mon-raid.py
```

Para que pueda funcionar hemos de activar y habilitar el *timer* (de manera análoga a lo que hicimos con los servicios tras instalarlos en la Práctica 2). Una vez hecho esto, podremos monitorizar su ejecución gracias al comando `journalctl`, p.ej. `journalctl -u mon-raid --since="yesterday"`.

5.2. Scripts

5.2.1. Shell y comandos del sistema: grep, find, awk y sed

Como ya han estudiado en Sistemas Operativos, el shell, en sus diferentes versiones (z, bourne, bash, c, ...) permite programar permitiendo hacer tareas de manera automática. Además de lo que ya saben, es importante mencionar tres comandos muy útiles de cara a automatizar tareas: `grep`, `find`, `awk` y `sed`, puede consultar algunos ejemplos en [Stuff, 2017b], [Stuff, 2017a].

Por ejemplo con `sed` puede buscar una cadena en un archivo y reemplazarla por otra, así podría dar acceso por contraseña durante unos instantes al servicio `ssh` para que los usuarios puedan copiar su llave pública.

Con `awk` puede programar la manipulación del texto así como generar salidas más completas a partir de la información en un archivo.

Con `grep` puede realizar filtrado de cadenas, útil cuando un archivo, listado, etc tiene unas dimensiones grandes, p.ej. `ps -Af | grep firefox` nos mostraría la información del proceso `firefox` (descartando el resto de información) Por último, con `find`, aunque probablemente ya lo haya usado en SOs, puede buscar archivos y, una vez encontrados, realizar acciones sobre ellos, p.ej. `find /home/alberto/docs -name '*pdf' -exec cp {} ~/PDFs \;` copiará todos los archivos cuyo nombre termine en `pdf` y los copia en la carpeta `/home/alberto/PDFs`

5.2.2. Python y PHP

Uno de los lenguajes más populares a día de hoy para programar servidores, concretamente páginas web dinámicas, es PHP. Existen numerosos y extendidos CMS (Content Management System) escritos en PHP, p.ej. Wordpress, Joomla, CakePHP, etc. Para poder programar sin tener que partir de cero existen varios frameworks como Symfony y Zend entre otros.

Python es un lenguaje de scripting con una creciente popularidad y se presenta como una gran alternativa a PHP (el framework `django` se basa en Python). También está ganando usuarios en el mundo de la investigación presentándose como una alternativa a Matlab (sin tener en cuenta Octave).

De cara a la asignatura, debido a su facilidad para manipular cadenas de texto mediante bibliotecas, nos puede permitir escribir tareas automatizadas que manipulen archivos de

configuración. Como ejemplos muy sencillos que no requieren excesivo conocimiento del lenguaje tenemos los proporcionados en [Saha, 2013]. En [(IBM), 2013] también hay otros scripts que muestran posibles casos prácticos además de todas las *recetas* disponibles en <https://github.com/ActiveState/code/tree/master/recipes/Python>.

Como último ejemplo, es oportuno citar la biblioteca para hacer uso de la API de Zabbix disponible en [Cyca, 2017].

5.3. A nivel de Plataforma: Ansible

Además de todo lo visto, también existen interfaces que permiten programar scripts y visualizar su ejecución de una manera cómoda y visual. Por falta de tiempo, no los cubriremos en la asignatura con detalle, no obstante, sí que veremos en una lección el funcionamiento básico de Ansible y se le anima a visitar las webs de los proyectos y ver algún vídeo relacionado.

P.ej. Puppetlabs <http://puppetlabs.com/> Ansible <http://www.ansible.com/home> Run-deck <http://rundeck.org/screencasts.html>

6) Profiling

6.1. Scripts

Para estudiar el comportamiento de los scripts, independientemente el lenguaje que usemos, podemos usar varios profilerers, p.ej. en Bash podemos usar la opción `set -x` y modificar la variable local PS4 para que muestre el tiempo http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_02.html. En el caso de PHP tenemos el proyecto desarrollado inicialmente por Facebook:

XHPProf: <http://pecl.php.net/package/xhprof> <https://github.com/facebook/xhprof>

Además de la documentación oficial, en la página de un prestigioso sistema de enseñanza virtual hay unos tutoriales interesantes sobre cómo realizar el profiling: http://docs.moodle.org/dev/Profiling_PHP.

En el caso de Python se puede utilizar el módulo cProfile. Para más información consulte en <https://docs.python.org/3/library/profile.html>.

6.2. SQL

El mismo MySQL (y MariaDB) que instalamos en la práctica anterior tiene un “profiler” para analizar cuánto tardan las consultas.

Puede ver la documentación en:

<http://dev.mysql.com/doc/refman/5.0/en/show-profiles.html> <http://dev.mysql.com/doc/refman/5.0/en/show-profile.html>

Además, es especialmente útil la herramienta MySQLWorkBench y otros comandos disponibles como `mysqloptimize` que optimizan la ejecución.

Pruebe a obtener un profile de una tabla cogiendo todos los atributos (`SELECT * ...`) y a escoger únicamente uno o dos (`SELECT ID,AGE ...`) y compare el tiempo que requiere consultar información extra que no necesitamos.

En esta Lección vamos a ver cómo automatizar los procesos de la lección anterior creando scripts y servicios asociados a estos en Ubuntu.

Como hemos visto, en el fichero **/proc/mdstat** está la información acerca de los RAID. Cuando uno de los discos duros del RAID está caído, aparece denotado por una **_** y si están funcionando correctamente por una **U**. Entonces vamos a ver cómo hacerle un “Profiling” (poder automatizar este tipo de cosas y perfilarlas) al proceso anterior, es decir, cómo mantener controlado en nuestro sistema el estado de los RAID (en este caso).

Primeramente, haremos un script en Python (mostrado en el apartado 5), donde importamos la librería para expresiones regulares para detectar la **_** en el contenido de **/proc/mdstat**, lo que devolverá que existe un error en el RAID. Esta opción es buena para usar puntualmente, pero si queremos automatizarlo tenemos dos opciones:

- o bien, modificando el fichero **crontab** donde podemos indicar líneas de ejecución, el periodo de ejecución y el usuario que lo ejecutará.
- o bien, crear un servicio que ejecute esto cada minuto, usando **timer**. Tenemos que definir un **timer** dentro de la carpeta **/etc/systemd/system/** y debemos crear también un servicio asociado a este **timer** para que el sistema automáticamente cuando lea este **timer** busca el servicio análogo al mismo. Y una vez definido, damos de alta el servicio como siempre lo hemos hecho con **systemctl enable** y **start timernuevo**. De esta forma, cada minuto vamos a saber cómo está el RAID y podemos monitorizar (o visualizar) su ejecución con el **journalctl**, que va mostrando todos los mensajes del sistema desde un momento determinado.

Antes de encender la máquina con UbuntuServer debemos cerciorarnos en *Configuracion/Almacenamiento* que el disco duro *sdb* sea *Conectable en caliente* (para hacer comprobaciones posteriores). También tomamos una instantánea del estado actual de la máquina.

Creamos el script con **vi mon_raid.py** como hemos visto (guardar y salir **:wq**). La expresión regular **'\[U]*[_]+[U]*\'** busca en el fichero si hay una **_** sola o detrás o delante o entre dos U, básicamente, si encuentra cualquier guion bajo.

```
import re

f=open('/proc/mdstat')

for line in f:
    b=re.findall('\[U]*[_]+[U]*', line)
    print(b)
    if(b!=[]):
        print("--ERROR_EN_RAID--")
print("--END_OF_SCRIPT")
```

Comprobamos que funciona correctamente ejecutándolo con **python3 mon_raid.py** y vemos que como no ha encontrado nada, sale todo vacío.

```
peroj@peroj:~$ python3 mon_raid.py
[]
[]
[]
[]
[]
[]
[]
[]
--END_OF_SCRIPT
```

Ahora vamos a desconectar el segundo disco duro *sdb* y cuando la máquina lo haya detectado volvemos a ejecutar el script y vemos que encuentra dos errores, lógicamente, ya que tenemos dos RAID.

```
peroj@peroj:~$
peroj@peroj:~$
peroj@peroj:~$ [ 1060.879020] sd 3:0:0:0: rejecting I/O to offline device
[ 1060.879147] blk_update_request: I/O error, dev sdb, sector 620552 op 0x1:(WRITE) flags 0x800 phys
_seg 1 prio class 0
[ 1060.879407] md: super_written gets error=10
[ 1060.879557] md/raid1:md1: Disk failure on sdb3, disabling device.
[ 1060.879557] md/raid1:md1: Operation continuing on 1 devices.
[ 1065.531413] md/raid1:md0: Disk failure on sdb2, disabling device.
[ 1065.531413] md/raid1:md0: Operation continuing on 1 devices.

peroj@peroj:~$
peroj@peroj:~$ python3 mon_raid.py
[]
[]
['[U_]']
--ERROR_EN_RAID--
[]
[]
['[U_]']
--ERROR_EN_RAID--
[]
[]
--END_OF_SCRIPT
peroj@peroj:~$
```

Una vez visto el funcionamiento, vamos a crear un **timer** que minuto a minuto ejecute el script para chequear los RAID. De esta forma, nosotros tenemos información detallada minuto a minuto de si ha pasado algo.

Ejecutamos **sudo su** y nos movemos a la carpeta **/etc/systemd/system** y crearemos dos ficheros, el **.timer** y el **.service** asociados a nuestro script.

```
peroj@peroj:~$ cd /etc/systemd/system
peroj@peroj:/etc/systemd/system$ ls
cloud-final.service.wants      mdmonitor.service.wants      snap-snapd-9279.mount
cloud-init.target.wants        multi-user.target.wants      snap-snapd-9721.mount
dbus-org.freedesktop.resolve1.service  multipath-tools.service      snap.lxd.activate.service
dbus-org.freedesktop.thermald.service  network-online.target.wants  snap.lxd.daemon.service
dbus-org.freedesktop.timesync1.service  open-vm-tools.service.requires  snap.lxd.daemon.unix.socket
default.target.wants              paths.target.wants           sockets.target.wants
emergency.target.wants            rescue.target.wants          sshd.service
final.target.wants               snap-core18-1885.mount       sysinit.target.wants
getty.target.wants               snap-core18-1932.mount       syslog.service
graphical.target.wants           snap-lxd-16922.mount         timers.target.wants
iscsi.service                   snap-lxd-18150.mount         vmtoolsd.service
peroj@peroj:/etc/systemd/system$ vi mon_raid.timer_
```

Primero creamos el **timer** con **vi mon_raid.timer** :

```
[Unit]
Description=Monitor RAID Service
[Timer]
OnCalendar=minutely
[Install]
WantedBy=timers.target
```

El **Unit** indica que vamos a crear una unidad, es decir, un servicio. Le damos la descripción. Le ponemos el **target** que es donde le decimos que se actualizará esto minuto a minuto y por último en el **Install**, le indicamos que lo instale como **Timer** y que esté pendiente minuto a minuto de esto.

Después creamos el *servicio* con **vi mon RAID.service** (con el mismo nombre que el *timer*):

```
[Unit]
Description=Monitor RAID Service
[Service]
Type=simple
ExecStart=/usr/bin/python3 /home/peroj/mon RAID.py
```

Vemos que ya lo tenemos definido con **ls** y con **systemctl enable mon RAID.timer** le decimos al sistema que habilite el *timer* (internamente el sistema buscará el *servicio*) y con **systemctl start mon RAID.timer** que inicie el servicio. Comprobamos que está *running* con **systemctl status mon RAID.timer**.

```
root@peroj:/etc/systemd/system# ls
cloud-final.service.wants      mon RAID.service              snap-snapd-9279.mount
cloud-init.target.wants        mon RAID.timer                snap-snapd-9721.mount
dbus-org.freedesktop.resolve1.service  multi-user.target.wants      snap.lxd.activate.service
dbus-org.freedesktop.thermal.service  multipath-tools.service      snap.lxd.daemon.service
dbus-org.freedesktop.timesync1.service network-online.target.wants  snap.lxd.daemon.unix.socket
default.target.wants            open-vm-tools.service.requires  sockets.target.wants
emergency.target.wants          paths.target.wants            sshd.service
final.target.wants              rescue.target.wants           sysinit.target.wants
getty.target.wants              snap-core18-1885.mount        syslog.service
graphical.target.wants          snap-core18-1932.mount        timers.target.wants
iscsi.service                   snap-lxd-16922.mount          vmtoolsd.service
mdmonitor.service.wants         snap-lxd-18150.mount
root@peroj:/etc/systemd/system# systemctl enable mon RAID.timer
Created symlink /etc/systemd/system/timers.target.wants/mon RAID.timer → /etc/systemd/system/mon RAID.timer.
root@peroj:/etc/systemd/system# systemctl start mon RAID.timer
root@peroj:/etc/systemd/system# systemctl status mon RAID.timer
● mon RAID.timer - Monitor RAID Service
   Loaded: loaded (/etc/systemd/system/mon RAID.timer; enabled; vendor preset: enabled)
   Active: active (waiting) since Wed 2020-11-18 17:32:06 UTC; 5s ago
   Trigger: Wed 2020-11-18 17:33:00 UTC; 47s left
   Triggers: ● mon RAID.service
Nov 18 17:32:06 peroj systemd[1]: Started Monitor RAID Service.
```

Para ver lo que está monitorizando el servicio usamos **journalctl** que recoge los mensajes del sistema (incluidos los mensajes lanzados por los servicios) pero deberemos de filtrarlo para ver concretamente el nuestro y por ejemplo desde ayer, con el comando siguiente **journalctl -u mon RAID --since="yesterday"**.

```
root@peroj:/etc/systemd/system# journalctl -u mon RAID --since="yesterday"
-- Logs begin at Tue 2020-09-29 13:18:31 UTC, end at Wed 2020-11-18 17:35:09 UTC. --
Nov 18 17:33:09 peroj systemd[1]: Started Monitor RAID Service.
Nov 18 17:33:09 peroj python3[1564]: []
Nov 18 17:33:09 peroj python3[1564]: []
Nov 18 17:33:09 peroj python3[1564]: ['U_']
Nov 18 17:33:09 peroj python3[1564]: --ERROR_EN_RAID--
Nov 18 17:33:09 peroj python3[1564]: []
Nov 18 17:33:09 peroj python3[1564]: []
Nov 18 17:33:09 peroj python3[1564]: ['U_']
Nov 18 17:33:09 peroj python3[1564]: --ERROR_EN_RAID--
Nov 18 17:33:09 peroj python3[1564]: []
Nov 18 17:33:09 peroj python3[1564]: []
Nov 18 17:33:09 peroj python3[1564]: --END_OF_SCRIPT
Nov 18 17:33:09 peroj systemd[1]: mon RAID.service: Succeeded.
Nov 18 17:34:09 peroj systemd[1]: Started Monitor RAID Service.
Nov 18 17:34:09 peroj python3[1584]: []
Nov 18 17:34:09 peroj python3[1584]: []
Nov 18 17:34:09 peroj python3[1584]: ['U_']
Nov 18 17:34:09 peroj python3[1584]: --ERROR_EN_RAID--
Nov 18 17:34:09 peroj python3[1584]: []
Nov 18 17:34:09 peroj python3[1584]: []
Nov 18 17:34:09 peroj python3[1584]: ['U_']
Nov 18 17:34:09 peroj python3[1584]: --ERROR_EN_RAID--
Nov 18 17:34:09 peroj python3[1584]: []
Nov 18 17:34:09 peroj python3[1584]: []
Nov 18 17:34:09 peroj python3[1584]: --END_OF_SCRIPT
Nov 18 17:34:09 peroj systemd[1]: mon RAID.service: Succeeded.
Nov 18 17:35:09 peroj systemd[1]: Started Monitor RAID Service.
Nov 18 17:35:09 peroj python3[1597]: []
```

Como vemos el servicio está reconociendo correctamente el error en los RAID cada minuto.

MEMORIA: Zabbix

4.4. ZABBIX

Es otro programa que permite monitorizar el sistema también de código abierto. Su instalación es relativamente sencilla ya que solo hay que añadir los repositorios (visto en la práctica anterior) e instalarlo con el gestor de paquetes.

Este es el sistema de monitorización en el que nos centraremos en las lecciones. En [\[Zabbix SIA, 2020\]](#) encontrará toda la información que necesitará para el seguimiento de las lecciones.

El objetivo es que usted sea capaz de instalar este sistema de monitorización y configurar éste para que se monitorice a sí mismo así como a CentOS (a través del agente) además de conocer cómo interactuar con el sistema de monitorización usando la API que proporcionar y conocer en qué consiste el protocolo SNMP.

Software de monitorización que crea un frontend en el navegador que permite ver de forma visual los envíos de los agentes (la información que envían los servidores monitorizados).

Ejercicio 1:

Realice una instalación de Zabbix 5.0 en su servidor con Ubuntu Server 20.04 y configure para que se monitorice a él mismo y para que monitorice a la máquina con CentOS. Puede configurar varios parámetros para monitorizar, uso de CPU, memoria, etc. pero debe configurar de manera obligatoria la monitorización de los servicios SSH y HTTP. Documente el proceso de instalación y configuración indicando las referencias que ha utilizado así como los problemas que ha encontrado. Para ello puede usar cualquier tipo de formato de documento (respetando claridad y corrección) y procure que en las capturas aparezca su nombre de usuario (en el prompt p.ej.). El archivo debe estar subido a SWAD (zona mis trabajos) antes del examen de esta práctica.

5.3. A nivel de Plataforma: Ansible

Además de todo lo visto, también existen interfaces que permiten programar scripts y visualizar su ejecución de una manera cómoda y visual. Por falta de tiempo, no los cubriremos en la asignatura con detalle, no obstante, sí que veremos en una lección el funcionamiento básico de Ansible y se le anima a visitar las webs de los proyectos y ver algún vídeo relacionado.

P.ej. Puppetlabs <http://puppetlabs.com/> Ansible <http://www.ansible.com/home> Rundeck <http://rundeck.org/screencasts.html>

Ejercicio 2: Usted deberá saber cómo instalar y configurar Ansible para poder hacer un ping a las máquinas virtuales de los servidores y ejecutar un comando básico (p.ej. el script de monitorización del RAID1). También debe ser consciente de la posibilidad de escribir acciones más complejas mediante playbooks escritos con YAML. Incluya capturas de pantalla del proceso con una breve descripción en el mismo documento que suba para el ejercicio de Zabbix.

Pasos:

- 1- Instalar y configurar Ansible en Ubuntu y añadimos las IPs de los servidores CentOS y Ubuntu al fichero de configuración (también podemos crear un grupo y en el momento de ejecutarlo en vez de ejecutar **ansible** con **all** ponemos el nombre del grupo).
- 2- Deberemos hacerlo con clave pública-privada y cambiar al puerto por defecto (22) de ssh. Deberemos regenerar las claves pública-privada, de Ubuntu a sí mismo y de Ubuntu a CentOS.
- 3- Cuando Ansible nos devuelva el ping correctamente a los servidores, deberemos crear en CentOS (o enviar por ssh) el fichero **mon_raid.py** y simplemente si tenemos ping y el fichero está en CentOS, debería funcionar correctamente.

El software Ansible, permite automatizar el profiling anterior (la monitorización del RAID), porque a Ansible se le añade tantas máquinas como se tengan (en nuestro caso 2 IPs, Ubuntu y CentOS) y le dice a las máquinas con el script que hemos creado que devuelvan cómo están los RAID (en este caso) en cada máquina, sin necesidad de ir consultándolo servidor por servidor, directamente con Ansible, desde el servidor “maestro” puedes ver la información del resto de servidores.

Con Ansible podemos automatizar monitorizaciones a lo largo de multitud de servidores.

Teniendo Ansible instalado tenemos que configurar que monitorice (automatice el script) tanto Ubuntu como CentOS. Realmente es simplemente añadirle los host (tanto Ubuntu como

CentOS) para que Ansible sea capaz de verlos y poder hacer este tipo de automatizaciones, Ansible, más que para monitorizar se utiliza para automatizar. Ansible en definitiva nos permite automatizar tareas en muchos servidores.

Una vez que tenemos los host añadidos, podemos comprobarlo desde Ubuntu con ***ansible all -m ping -u peroj*** . Como Ansible hace la conexión por ssh, debemos estar (dado que no tenemos acceso para root por ssh en las máquinas) y poner el mismo usuario en todos los servidores para hacerle ping, en nuestro caso tenemos el mismo tanto en CentOS como en Ubuntu (en un caso real deberemos tener un usuario para estas cosas). Comprobamos que se hace ping correctamente.

Ahora debemos crear el script ***mon_raid.py*** tanto en CentOS como en Ubuntu (y en la misma ubicación). También podemos enviarlo por ssh desde Ubuntu para CentOS.

Ahora podemos ejecutar con Ansible en todos los servidores y recibir el resultado en nuestra máquina con ***ansible all -a "python3 /home/peroj/mon_raid.py"*** (si el fichero a ejecutar no estuviera en la misma ubicación en las dos máquinas, esto deberíamos hacerlo con un script de configuración). Comprobamos y vemos como en Ubuntu nos devuelve el error en los RAID pero en CentOS no, dado que en CentOS funciona correctamente.

Ansible nos permite una automatización brutal si tenemos muchos servidores, incluso después de esto podríamos coger con otro script qué RAIDs están fallando y arreglarlo en todas las máquinas a la vez ejecutando dicho script (siempre y cuando fuese el mismo problema en todas y se pudiese subsanar así).

También debemos saber que se pueden hacer acciones más complejas mediante los ***playbooks*** (libros de juegos) que son los scripts de Ansible escritos en lenguaje YAML.

Ansible

Ansible es una solución de RedHat para la gestión de la configuración. Automatiza la configuración de servidores a gran escala empleando descriptores (Playbooks) en formato abierto de texto plano gestionables en un repositorio SCM. Ansible, es un referente de la industria, junto a otras soluciones como Puppet o Chef.

Ansible funciona de forma declarativa. Es decir, en lugar de indicar la secuencia de pasos para alcanzar una configuración deseada (lo que sería procedural), se describe el estado final esperado. Esto implica que los comandos de ansible son (o más bien, deberían ser) idempotentes. Los usuarios pueden construir nuevos comandos empleando las Tasks de los Playbooks.

En estas prácticas solo abordaremos la ejecución de comandos de Ansible, quedando excluidos los Playbooks.

La alumna deberá comprender el funcionamiento en Ansible de:

- Inventario.
- Módulos

Como ejercicio práctico final (no entregable), la alumna debe ser capaz de ejecutar un comando ansible para apagar todas las máquinas virtuales (al menos centos y ubuntu) desde su ordenador anfitrión.

Referencias:

- Tutorial de Introducción a Ansible:
https://docs.ansible.com/ansible/latest/user_guide/intro_getting_started.html
- Inventario de Ansible:
https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html
- Documentación de Módulos:
https://docs.ansible.com/ansible/latest/modules/modules_by_category.html
- Documentación General: <https://docs.ansible.com>

1º) Instalamos **ansible** en UbuntuServer con **sudo apt-get install ansible**. Comprobamos la versión con **ansible --version**.

```
UbuntuServerISE (Instantánea 5 - RAID y sistema recuperados) [Corriendo] - Oracle VM ...
Archivo Máquina Ver Entrada Dispositivos Ayuda
root@peroj:/# sudo apt-get install ansible
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ieee-data python3-argcomplete python3-crypto python3-dnspython python3-jmespath python3-kerberos
  python3-libcloud python3-lockfile python3-netaddr python3-ntlm-auth python3-requests-kerberos
  python3-requests-ntlm python3-selinux python3-winrm python3-xlrd python3-xlsxwriter
Suggested packages:
  cowsay sshpass python-lockfile-doc ipython3 python-netaddr-docs
The following NEW packages will be installed:
  ansible ieee-data python3-argcomplete python3-crypto python3-dnspython python3-jmespath
  python3-kerberos python3-libcloud python3-lockfile python3-netaddr python3-ntlm-auth
  python3-requests-kerberos python3-requests-ntlm python3-selinux python3-winrm python3-xlrd
0 upgraded, 16 newly installed, 0 to remove and 77 not upgraded.
Need to get 9643 kB of archives.
After this operation, 90.2 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://es.archive.ubuntu.com/ubuntu focal/main amd64 python3-crypto amd64 2.6.1-13ubuntu2 [237
kB]
Get:2 http://es.archive.ubuntu.com/ubuntu focal/main amd64 python3-dnspython all 1.16.0-1build1 [89.
1 kB]
Get:3 http://es.archive.ubuntu.com/ubuntu focal/main amd64 ieee-data all 20180805.1 [1589 kB]
Get:4 http://es.archive.ubuntu.com/ubuntu focal/main amd64 python3-netaddr all 0.7.19-3 [235 kB]
Get:5 http://es.archive.ubuntu.com/ubuntu focal/universe amd64 ansible all 2.9.6+dfsg-1 [5794 kB]
Get:6 http://es.archive.ubuntu.com/ubuntu focal/universe amd64 python3-argcomplete all 1.8.1-1.3ubun
tu1 [27.2 kB]
Get:7 http://es.archive.ubuntu.com/ubuntu focal/main amd64 python3-jmespath all 0.9.4-2 [21.3 kB]
Get:8 http://es.archive.ubuntu.com/ubuntu focal/universe amd64 python3-kerberos amd64 1.1.14-3.1buil
d1 [22.6 kB]
Get:9 http://es.archive.ubuntu.com/ubuntu focal/main amd64 python3-lockfile all 1:0.12.2-2ubuntu2 [1
4.6 kB]
Get:10 http://es.archive.ubuntu.com/ubuntu focal/universe amd64 python3-libcloud all 2.8.0-1 [1403 k
B]
87% [10 python3-libcloud 1064 kB/1403 kB 76%] 622 kB/s 0s_
```

Y vemos que usa la versión 3.8.5 de Python, es decir, el módulo **python3**.

```
root@peroj:/# ansible --version
ansible 2.9.6
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/mod
ules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.8.5 (default, Jul 28 2020, 12:59:40) [GCC 9.3.0]
```

2º) Ahora deberemos irnos al archivo de inventario con **vi /etc/ansible/hosts** donde indicaremos los host con las IPs de los servidores con los que queremos tratar, en nuestro caso, tanto de Ubuntu como de CentOS (podemos ver varios ejemplos de configuración comentados en el archivo).

Este archivo se suele utilizar también para configurar variables que serán válidas sólo para hosts o grupos específicos, a fin de usarse dentro de los playbooks y las plantillas. Algunas variables también pueden afectar la forma en que se ejecuta un playbook, como la variable **ansible_python_interpreter** que veremos a continuación.

```
root@peroj:/# ls /etc/ansible/
ansible.cfg hosts
root@peroj:/# vi /etc/ansible/hosts_
```

Creamos un grupo de servidores llamado *auto_raids* donde incluiremos las IPs tanto de CentOS como de Ubuntu. También creamos (si es necesario) un subgrupo de este llamado *auto_raids:vars* donde establecemos el parámetro ***ansible_python_interpreter=/usr/bin/python3*** (descomentar solo si es necesario).

```
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
# - Comments begin with the '#' character
# - Blank lines are ignored
# - Groups of hosts are delimited by [header] elements
# - You can enter hostnames or ip addresses
# - A hostname/ip can be a member of multiple groups

#Grupo para servidores Ubuntu y CentOS
[auto_raids]
ubuntu_server ansible_host=192.168.56.105
centos_server ansible_host=192.168.56.110

#Subgrupo que establece el parametro que garantiza que el servidor remoto utilice python3
#[auto_raids:vars]
#ansible_python_interpreter=/usr/bin/python3
```

Comprobamos que se han añadido los host correctamente al inventario con ***ansible-inventory --list -y***.

```
root@peroj:/# ansible-inventory --list -y
all:
  children:
    auto_raids:
      hosts:
        centos_server:
          ansible_host: 192.168.56.110
        ubuntu_server:
          ansible_host: 192.168.56.105
      ungrouped: {}
```

3º) Antes de probar conexión, debemos cambiar el puerto de las máquinas al puerto por defecto (22), con lo que esto conlleva, en Ubuntu deberemos simplemente avisar al firewall, pero en CentOS también deberemos actualizar el SELinux y ya después, volver a generar los pares de claves pública-privada, tanto de Ubuntu consigo mismo como con CentOS (ya que *ansible* se conecta por ssh).

-En Ubuntu: Abrimos **/etc/ssh/sshd_config** y cambiamos el puerto al *Port 22* y el parámetro *Password Authentication yes* para poder generar las nuevas claves (después lo cambiamos otra vez). Después de guardar el fichero, reiniciamos el servicio con **systemctl restart sshd.service** para aplicar los cambios y lo vemos con **systemctl status sshd.service**. Nos aseguramos que el puerto 22 está abierto en el firewall **ufw allow 22** (que modifica las *iptables* para que nos permita comunicación por este puerto).

```
peroj@peroj:/$ systemctl restart sshd.service
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ====
Authentication is required to restart 'sshd.service'.
Authenticating as: peroj
Password:
==== AUTHENTICATION COMPLETE ====
peroj@peroj:/$ systemctl status sshd.service
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/systemd/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2020-11-18 20:44:18 UTC; 12s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
  Process: 5666 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
 Main PID: 5681 (sshd)
    Tasks: 1 (limit: 4621)
   Memory: 1.1M
    CGroup: /system.slice/ssh.service
            └─5681 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups

Nov 18 20:44:18 peroj systemd[1]: Starting OpenBSD Secure Shell server...
Nov 18 20:44:18 peroj sshd[5681]: Server listening on 0.0.0.0 port 22.
Nov 18 20:44:18 peroj sshd[5681]: Server listening on :: port 22.
Nov 18 20:44:18 peroj systemd[1]: Started OpenBSD Secure Shell server.
peroj@peroj:/$ ssh-keygen
```

Primero generamos la clave en el cliente (Ubuntu) con **ssh-keygen**. Se la mandamos al servidor (Ubuntu) con **ssh-copy-id peroj@192.168.56.105 -p 22**, metemos la contraseña de Ubuntu y vemos que se ha añadido una clave. Finalmente, comprobamos haciendo login por **ssh peroj@192.168.56.105 -p 22**, ya no nos pide la contraseña.

```
peroj@peroj:/$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/peroj/.ssh/id_rsa):
/home/peroj/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/peroj/.ssh/id_rsa
Your public key has been saved in /home/peroj/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:Q+aRlt7bHbc2p0Za9cdAx2IurMCiCItKJwzVRypeWB8 peroj@peroj
The key's randomart image is:
+---[RSA 3072]-----+
|    .E      .    |
|  .0.0.. 0    + 0 |
| .0 0...B . + 0 |
|0. 0  .*00  0 0 . |
|+0.. . .S... ..+0 |
|0+... ..0 .00= |
|0 0      . .+.+0 |
| .          . 0.0 |
| ..          ..   |
+---[SHA256]-----+
peroj@peroj:/$ ssh-copy-id peroj@192.168.56.105 -p 22
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/peroj/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are alr
eady installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to inst
all the new keys
peroj@192.168.56.105's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh -p '22' 'peroj@192.168.56.105'"
and check to make sure that only the key(s) you wanted were added.
peroj@peroj:/$
```



```

peroj@peroj:/$ sudo ufw allow 22
Skipping adding existing rule
Skipping adding existing rule (v6)
peroj@peroj:/$ ssh peroj@192.168.56.105 -p 22
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-52-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Wed Nov 18 20:50:43 UTC 2020

System load: 0.0               Processes: 141
Usage of /home: 0.2% of 975MB   Users logged in: 1
Memory usage: 15%              IPv4 address for enp0s3: 10.0.2.15
Swap usage: 0%                 IPv4 address for enp0s8: 192.168.56.105

 * Introducing self-healing high availability clustering for MicroK8s!
   Super simple, hardened and opinionated Kubernetes for production.

   https://microk8s.io/high-availability

80 updates can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Last login: Wed Nov 18 16:53:30 2020
peroj@peroj:~$ exit
logout
Connection to 192.168.56.105 closed.
peroj@peroj:/$

```

Ya tenemos conexión por ssh de Ubuntu consigo misma a través del puerto 22.

En Ubuntu debemos recordar poner el parámetro de ***/etc/ssh/sshd_config*** ***AuthenticationPassword no.***

-En CentOS: Haremos “el mismo” proceso anterior para configurarle el ssh server y podernos conectar desde UbuntuServer. Comprobamos que esta dado de alta el servicio con ***sudo systemctl status sshd.service***. Vamos a cambiarle el puerto 22 que trae por defecto por el 22022 y el *PasswordAuthentication* a *yes* modificando el fichero ***sudo vi /etc/ssh/sshd_config***.

```
CentOS-LVM (Instantánea 3 - LAMP) [Corriendo] - Oracle VM VirtualBox
CentOS Linux 8 (Core)
Kernel 4.18.0-193.el8.x86_64 on an x86_64

Activate the web console with: systemctl enable --now cockpit.socket

localhost login: peroj
Password:
Last failed login: Mon Nov  9 13:26:28 EST 2020 from 192.168.56.1 on ssh:notty
There were 5 failed login attempts since the last successful login.
Last login: Mon Nov  9 12:29:53 on tty1
[peroj@localhost ~]$ sudo su
[sudo] password for peroj:
[root@localhost peroj]# systemctl status sshd.service
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2020-11-18 15:31:18 EST; 43min ago
     Docs: man:sshd(8)
           man:sshd_config(5)
   Main PID: 890 (sshd)
     Tasks: 1 (limit: 23960)
    Memory: 2.4M
   CGroup: /system.slice/sshd.service
           └─890 /usr/sbin/sshd -D -oCiphers=aes256-gcm@openssh.com,chacha20-poly1305@openssh.com,a

nov 18 15:31:17 localhost.localdomain systemd[1]: Starting OpenSSH server daemon...
nov 18 15:31:18 localhost.localdomain sshd[890]: Server listening on 0.0.0.0 port 22022.
nov 18 15:31:18 localhost.localdomain sshd[890]: Server listening on :: port 22022.
nov 18 15:31:18 localhost.localdomain systemd[1]: Started OpenSSH server daemon.
[root@localhost peroj]# vi /etc/ssh/sshd_config_
```

Una vez hecho esto, debemos avisar al sistema de seguridad de Linux SELinux de que vamos a cambiar el puerto con la orden ***semanage port -a -t ssh_port_t -p tcp 22*** y reiniciamos el servicio con ***sudo systemctl restart sshd.service***.

También avisamos al cortafuegos de que permita conexiones por ese puerto con ***sudo firewall-cmd --permanent --add-port=22022/tcp*** y recargamos las reglas con ***sudo firewall-cmd --reload***.

```
"/etc/ssh/sshd_config" 148L, 4422C written
[root@localhost peroj]# semanage port -a -t ssh_port_t -p tcp 22
ValueError: El puerto tcp/22 ya está definido
[root@localhost peroj]# systemctl restart sshd.service
[root@localhost peroj]# systemctl status sshd.service
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2020-11-18 16:17:36 EST; 5s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
   Main PID: 1549 (sshd)
     Tasks: 1 (limit: 23960)
    Memory: 1.1M
   CGroup: /system.slice/sshd.service
           └─1549 /usr/sbin/sshd -D -oCiphers=aes256-gcm@openssh.com,chacha20-poly1305@openssh.com,a

nov 18 16:17:36 localhost.localdomain systemd[1]: Stopped OpenSSH server daemon.
nov 18 16:17:36 localhost.localdomain systemd[1]: Starting OpenSSH server daemon...
nov 18 16:17:36 localhost.localdomain sshd[1549]: Server listening on 0.0.0.0 port 22.
nov 18 16:17:36 localhost.localdomain sshd[1549]: Server listening on :: port 22.
nov 18 16:17:36 localhost.localdomain systemd[1]: Started OpenSSH server daemon.
[root@localhost peroj]# firewall-cmd --permanent --add-port=22/tcp
usage: see firewall-cmd man page
firewall-cmd: error: unrecognized arguments: --add-port=22/tcp
[root@localhost peroj]# firewall-cmd --permanent --add-port=22/tcp
success
[root@localhost peroj]# firewall-cmd --reload
success
```

Ya si intentamos conectarnos desde Ubuntu a CentOS con **ssh peroj@192.168.56.110 -p 22** ya podemos sin problema.

Ahora debemos enviar desde Ubuntu la clave pública-privada a CentOS para que no sea necesaria la contraseña.

```
peroj@peroj:/$ ssh-copy-id peroj@192.168.56.110 -p 22
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/peroj/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install all the new keys
peroj@192.168.56.110's password:

Number of key(s) added: 1

Now try logging into the machine, with:  "ssh -p '22' 'peroj@192.168.56.110'"
and check to make sure that only the key(s) you wanted were added.

peroj@peroj:/$ ssh peroj@192.168.56.110 -p 22
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Wed Nov 18 16:30:59 2020 from 192.168.56.105
[peroj@localhost ~]$ exit
logout
Connection to 192.168.56.110 closed.
peroj@peroj:/$
```

Y finalmente, le cambiamos del fichero de configuración de CentOS **sudo vi /etc/ssh/sshd_config** el **PasswordAuthentication** a no, para que solo se pueda conectar un usuario que previamente esté vinculado con una clave pública-privada. Y reiniciamos el servicio con **sudo systemctl restart sshd.service**.

```
"/etc/ssh/sshd_config" 148L, 4422C written
[root@localhost peroj]# systemctl restart sshd.service
[root@localhost peroj]# _
```

UbuntuServer1SE (Instantánea 5 - RAID y sistema recuperados) [Corriendo] - C

Archivo Máquina Ver Entrada Dispositivos Ayuda

```
peroj@peroj:/$ ssh peroj@192.168.56.110 -p 22
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Wed Nov 18 16:31:51 2020 from 192.168.56.105
[peroj@localhost ~]$ exit
logout
Connection to 192.168.56.110 closed.
```

Archivo Máquina Ver Entrada Dispositivos Ayuda

```
peroj@peroj:/$ ssh peroj@192.168.56.105 -p 22
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-52-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Wed Nov 18 21:39:02 UTC 2020

System load: 0.0          Processes: 140
Usage of /home: 0.2% of 975MB   Users logged in: 1
Memory usage: 15%          IPv4 address for enp0s3: 10.0.2.15
Swap usage: 0%             IPv4 address for enp0s8: 192.168.56.105

 * Introducing self-healing high availability clustering for MicroK8s!
   Super simple, hardened and opinionated Kubernetes for production.

   https://microk8s.io/high-availability

80 updates can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Last login: Wed Nov 18 21:35:15 2020 from 192.168.56.105
peroj@peroj:~$ exit
logout
Connection to 192.168.56.105 closed.
peroj@peroj:/$ ssh peroj@192.168.56.110 -p 22
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Wed Nov 18 16:35:08 2020 from 192.168.56.105
[peroj@localhost ~]$ exit
logout
Connection to 192.168.56.110 closed.
```

4º) Una vez que tenemos correctamente configuradas las conexiones ssh procedemos a comprobar que Ansible nos devuelve el ping correctamente a los servidores con **ansible all -m ping -u peroj** o **ansible auto_raids -m ping -u peroj**.

Archivo Máquina Ver Entrada Dispositivos Ayuda


```
peroj@peroj:/$ ansible all -m ping -u peroj
ubuntu_server | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
centos_server | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": false,
  "ping": "pong"
}
peroj@peroj:/$ ansible auto_raids -m ping -u peroj
ubuntu_server | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
centos_server | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": false,
  "ping": "pong"
}
```

5º) Ahora debemos crear el script **mon_raid.py** en CentOS (y en la misma ubicación que en Ubuntu). También podemos enviarlo por ssh desde Ubuntu para CentOS.

Ahora podemos ejecutar con Ansible en todos los servidores y recibir el resultado en nuestra máquina con **ansible all -a "python3 /home/perorj/mon_raid.py"** (si el fichero a ejecutar no estuviera en la misma ubicación en las dos máquinas, esto deberíamos hacerlo con un script de configuración o playbook). Comprobamos y vemos como en Ubuntu nos devuelve el error en los RAID pero en CentOS no, dado que en CentOS funciona correctamente.

 CentOS-LVM (Instantánea 3 - LAMP) [Corriendo] - Oracle VM VirtualBox

```
[root@localhost peroj]# ls
mon_raid.py  UbuntuBackup
[root@localhost peroj]# _
```

 UbuntuServerISE (Instantánea 5 - RAID y sistema recuperados) [Corriendo]

Archivo Máquina Ver Entrada Dispositivos Ayuda

```
perorj@perorj:~$ ansible all -a "python3 /home/perorj/mon_raid.py"
centos_server | CHANGED | rc=0 >>
[]
[]
--END_OF_SCRIPT
ubuntu_server | CHANGED | rc=0 >>
[]
[]
['[U_] ']
--ERROR_EN_RAID--
[]
[]
['[U_] ']
--ERROR_EN_RAID--
[]
[]
--END_OF_SCRIPT
perorj@perorj:~$
```