Ingeniería de Servidores (2018-2019)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 3 : Monitorización y "Profiling"

6 de noviembre de 2018

Índice

OBJETIVOS MÍNIMOS

1. Conocer y saber usar las herramientas que permitan obtener datos sobre el sistema a nivel hardware y software (SO y servicios).
2. Saber interpretar los resultados proporcionados por las aplicaciones de monitorización.
3. Conocer los archivos que proporcionan información del sistema.
4. Tener conocimiento básico sobre automatización y orquestación
5. Ser capaz de utilizar y configurar un monitor de sistema

Contenido de las lecciones

1. Monitorización del RAID1 y Monitores básicos
2. Automatización y Zabbix
3. Automatización y Zabbix (II)

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 (*B*asic 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.*, 2011, 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`, pero todavía necesita madurar a nivel de documentación.

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, ...).

4.4. 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, 2017] 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.

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.

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="ye`

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 Managemente System) escritos en PHP, p.ej. Wordpress, Joomla, CakePHP, etc. Para poder programar sin tener que partir de cero existen varios frameworks como Synfony 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

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 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`

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`. Para PHP tenemos el proyecto desarrollado por Facebook:
Xdebug: `http://www.xdebug.org/index.php` XHProf: `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: Puede obtener más información en `http://docs.python.org/2/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.

Referencias

[Cyca, 2017] Cyca, L. (2017). Pyzabbix. `https://github.com/lukecyca/pyzabbix`. [Online; consultada 14-September-2017].

[David Greaves *et al.*, 2011] David Greaves *et al.* (2006–2011). Detecting, querying and testing. `https://raid.wiki.kernel.org/index.php/Detecting,_querying_and_testing`. [Online; consultada 14-September-2017].

[David Greaves *et al.*, 2013] David Greaves *et al.* (2007–2013). mdstat. `https://raid.wiki.kernel.org/index.php/Mdstat`. [Online; consultada 14-September-2017].

[(IBM), 2013] (IBM), J. K. (2013). Python for system administrators. `https://www.ibm.com/developerworks/aix/library/au-python/index.html`. [Online; consultada 14-September-2017].

[Saha, 2013] Saha, A. (2013). Linux system mining with python. `http://echorand.me/site/notes/articles/python_linux/article.html`. [Online; consultada 14-September-2017].

[Stuff, 2017a] Stuff, T. G. (2017a). Ejemplos de awk. `http://www.thegeekstuff.com/2010/01/awk-introduction-tutorial-7-awk-print-examples/`. [Online; consultada 14-September-2017].

[Stuff, 2017b] Stuff, T. G. (2017b). Ejemplos de sed. `http://www.thegeekstuff.com/2009/10/unix-sed-tutorial-advanced-sed-substitution-examples/`. [Online; consultada 14-September-2017].

[Zabbix SIA, 2017] Zabbix SIA (2017). Manual de zabbix. `https://www.zabbix.com/documentation/3.4/manual`. [Online; consultada 14-September-2017].