U-tad

Facultad de Ingeniería de software

MACHMON - Monitorización de sistemas

Proyecto Fin de Carrera

Autor:
Javier Lima

Junio 2020

Contents

Chapter 1

Introducción

1.1 Justificación y contexto

Todo este tiempo en la universidad hemos desarrollado código para generar valor con aplicacones o scripts, pero ¿cómo medimos el rendimiento del sistema realmente?. Imaginemos que tenemos varias aplicaciones funcionando en una máquina, ¿como podríamos saber si es posible meter una nueva aplicación dentro del sistema?. Machmon, es un proyecto para profundizar en los sistemas y así poder saber la salud de ellos, consiste en recoger métricas que nos aporten información sobre la salud de los ordenadores (como la cpu, memoria ram, disco, etc) y representarlas gráficamente para sacar conclusiones sobre sus estados.

Actualmente, la monitorización de los ordenadores, es común en todas las empresas con sistemas propios que manejen datos de forma digital, es un paso más en el control de los sistemas, al final acabas aprovechando más los recursos, prevees posibles incidencias, mejoras la experiencia del cliente y ahorras tiempo y dinero.

1.2 Planteamiento del problema

1.3 Objetivos del trabajo

Como propósito este trabajo tendrá unos objetivos a cumplir:

1.4 Aumentar mis conocientos de sistemas linux

Linux es uno de los sistemas más populares entre los desarrolladores, la consola que tiene facilita mucho la operación de esas máquinas. Es por ello, que el uso de los sistemas Linux me va a ayudar al manejo de esos sistemas.

1.5 Utilizar tecnologías nuevas

La programación es muy revolucionaria, en cualquier momento aparece una aplicación nueva capaz de dejar obsoleta a otra. El estar puesto en las nuevas tecnologías nos facilita el trabajo, nos permite ver las utilidades de cada herramienta y los beneficios que nos pueda aportar.

Este trabajo tiene un fin, obtener la salud de los sistemas monitorizados, pero en el camino nos encontremos dificultades que seguramente una herramienta nos las solucione.

1.6 Modelo abstracto

Es un objetivo un poco pretencioso, trata de ser capaz de recibir métricas de cualquier tipo de sistema, que exista una entidad capaz de controlar los mensajes recibidos y almacenarlos para su tratamiento, es decir, que si en algún momento decido introducir una nueva máquina en el sistema sea capaz de entender los mensajes también y almacenarlos.

1.7 Concurrencia

Es un objetivo fundamental, es necesario tener en cuenta posibles mensajes que lleguen a la vez. La concurrencia nos soluciona el objetivo, tendremos que ser capaces de elaborar un sistema cuncurrente.

Chapter 2

Estudio de metricas

2.1 Estándares de medida de las métricas

- Memoria: kilobyte unidad utilizada para todas las métricas.
- Tiempo: milisegundos(preguntar por mbps).
- Temperatura: grados celcius.

2.2 cpusNumber

Recogeremos el número de cpus tanto en uso como totales y así poder hacer un seguimiento al rendimiento, obtendremos los elementos con los comandos nproc y lscpu. La métrica va a estar nombrada como cpusNumber y va a estar constituida por:

Nombre de la métrica: **cpusNumber**
- cpusTotalNumber: número de cpus totales.(int)
- cpusUsageNumber: número de cpus en uso. (int)

2.3 cpu

Recogeremos los porcentajes de uso de la cpu, lo obtendremos con el comando "iostat -c" (que hay que instalarse). La métrica va a estar nombrada como cpu y va a estar constituida por:

Nombre de la métrica: **cpu**
- userPercentage: porcentaje de uso de cpu que se produjo durante la ejecución a nivel de usuario. (float)

- nicePercentage: porcentaje de uso de cpu que se produjo al ejecutar a nivel de usuario con buena prioridad. (float)

- systemPercentage: porcentaje de uso de cpu que se produjo durante la ejecución a nivel de sistema. (float)

- iowaitPercentage: porcentaje de tiempo que la CPU o las CPUs estuvieron inactivas durante las cuales el sistema tuvo una solicitud I/O de disco pendiente. (float)

- stealPercentage: porcentaje de tiempo que la CPU virtual o las CPUs pasaron en espera involuntaria mientras el hipervisor daba servicio a otro procesador virtual. (float)

- idlePercentage: porcentaje de tiempo que la CPU o las CPU estuvieron inactivas y el sistema no tenía una solicitud de I/O de disco pendiente. (float)

2.4 mem

La memoria RAM la obtenemos del comando free, nos centraremos tanto en la memoria RAM como la memoria swap. La métrica va a estar nombrada como mem y va a estar constituida por:

Nombre de la métrica: **mem**
- totalMem: memoria física total.(int)
- usedMem: memoria física en uso. (int)
- freeMem: memoria física libre. (int)
- sharedMem: memoria RAM compartida actualmente en uso. (int)
- buffersMem: memoria actual del búffer de caché. (int)
- cachedMem: memoria de la caché de disco. (int)
- swapTotalMem: memoria virtual total. (int)
- swapUsedMem: memoria virtual en uso. (int)
- swapFreeMem: memoria virtual libre. (int)
- totalRAM: memoria RAM total.(int)
- usedRAM: memoria RAM en uso. (int)
- freeRAM: memoria RAM libre. (int)

2.5 disk

La memoria del disco la obtenemos del comando df --total, nos centraremos en el cálculo total y dejamos a un lado el resto sistema de ficheros, en un futuro se podría añadir el desglose del cálculo total. La métrica va a estar

nombrada como disk y va a estar constituida por:

Nombre de la métrica: **disk**
- identificatorName: nombre de identificación del disco. (string)
- totalDisk: memoria total del disco. (int)
- usedDisk: memoria en uso del disco. (int)
- freeDisk: memoria libre en el disco. (int)
- usagePercentDisk: porcentaje de uso del disco. (int)

2.6 temperature

La temperatura del sistema está mockeada, está seteada a 27 grados más un random de 0 a 11 grados La métrica va a estar nombrada como temperature y va a estar constituida por:

Nombre de la métrica: **temperature**
- degrees: temperatura del pc. (int)

2.7 partitions

Las particiones las obtenemos con el comando df memoria del disco la obtenemos del comando df, nos centraremos en la carpeta raíz utilizando el elemento /dev/sda1 para filtrar con grep. La métrica va a estar nombrada como partitions y va a estar constituida por:

Nombre de la métrica: **partitions**
- identificatorName: nombre de identificación de la partición. (string)
- type: tipo de partición. (string)
- totalDisk: memoria total de la partición. (int)
- usedDisk: memoria en uso de la partición. (int)
- freeDisk: memoria libre de la partición. (int)
- usagePercentDisk: porcentaje de uso del disco. (int)
- mountPoint: localización del direcotorio de la partición. (string)

2.8 ioRatio

El ratio de IO del disco lo obtendremos a partir del comando iostat, de donde filtraremos la salida para quedarnos con el disco principal sda. La métrica

va a estar nombrada como ioRatio y va a estar constituida por:

Nombre de la métrica: **ioRatio**
- deviceName: nombre de la partición de memoria. (string)
- transfersPerSecond: indica el número de transferencias por segundo que se emitieron al dispositivo. Una transferencia es una solicitud I/0 al dispositivo, se pueden combinar varias solicitudes lógicas en una sola solicitud porque es de tamaño indeterminado. (float)
- kilobytesReadsPerSecond: el número de kilobytes leídos del dispositivo por segundo. (float)
- kilobytesWrittenPerSecond: el número de kilobytes escritos en el dispositivo por segundo. (float)
- kilobytesRead: El número total de kilobytes leídos. (int)
- kilobytesWritten: El número total de kilobytes escritos. (int)

2.9 logs

Los logs del sistema los leeremos de archivo /var/log/syslog en nuestro sistema ubuntu. La métrica va a estar nombrada como logs y va a estar constituida por:

Nombre de la métrica: **logs**
- date: fecha en la que se hizo el log. (datetime)
- nameHost: nombre del sistema. (string)
- process?: nombre del proceso del sitema??. (string)?
- message: mensaje del log. (string)

2.10 processesNumber

Número de procesos del sistema, lo obtendremos con el comando ps, el argumento -a nos devuelve solo los procesos activos, nos guardaremos el número de procesos activos y el número de procesos totales La métrica va a estar nombrada como processesNumber y va a estar constituida por:

Nombre de la métrica: **processesNumber**
- activeProcessesNumber: el número de procesos activos de la máquina. (int)
- totalProcessesNumber: el número total de procesos de la máquina. (int)

2.11 process

La tabla de procesos del sistema la obtendremos del comando ps, filtramos el comando para que nos devuelva las entradas que nos interesan, que son las que tienen carga de cpu, es decir aquellas que tengan distinto de 0 el porcentaje de cpu. La métrica va a estar nombrada como process y va a estar constituida por:

Nombre de la métrica: **process**

- usedPercentageCpu: porcentaje de uso de la cpu del proceso. (float)
- pid: número identificador del proceso. (int)
- usedPercentageMem: porcentaje de uso de la memoria RAM del proceso. (float)
- nice: número que define la prioridad del proceso. Esta prioridad se llama Niceness en Linux, y tiene un valor entre -20 y 19. Cuanto más bajo sea el índice de Niceness, mayor será una prioridad dada a esa tarea. El valor predeterminado de todos los procesos es 0. (int)
- group: nombre del grupo al que pertenece el proceso. (string)
- user: nombre del usuario que ejecutó el proceso. (string)
- state: estado en el que se encuentra el proceso (R en ejecución, S dormido, T detenido, X muerto). (string)
- start: hora a la que empezó el proceso. (datetime)
- cpuTime: tiempo de ejecución en cpu. (datetime)
- command: descripción de la ejecución del proceso. (string)

2.12 latency

La latencia es el tiempo que lleva enviar una señal más el tiempo que lleva recibir un acuse de recibo de esa señal. Para calcularla utilizamos el comando ping en localhost con 3 paquetes y nos quedamos con el rtt. La métrica va a estar nombrada como latency y va a estar constituida por:

Nombre de la métrica: **latency**

- minRTT: es el número mínimo que tardó una de las peticiones ECHO_REQUEST. (float)
- meanRTT: es la media de lo que tardaron las peticiones ECHO_REQUEST. (float)
- maxRTT: es el número máximo que tardó una de las peticiones ECHO_REQUEST. (float)

- mdevRTT: es la desviación estándar, esencialmente un promedio de cuán lejos está cada RTT de ping de la RTT media. Cuanto más alto es el número, más variable es el RTT. (float)

- packageTransmited: es el número de paquetes transmitidos durante la prueba de latencia. (int)

- packageReceived: es el número de paquetes recibidos durante la prueba de latencia. (int)

- packageLossPercentage: es el porcentage de paquetes perdidos durante la prueba de latencia. (float)

- timeRequest: son los milisegundos que se han tardado en hacer la prueba de latencia. (int)

- clientServer: son los milisegundos que ha tardado el servidor en recibir las métricas. (int)

2.13 networkMetrics

Las métricas de red que disponen las tarjetas de red del sistema las obtendremos con el comando ifconfig. La métrica va a estar nombrada como netWorkMetrics y va a estar constituida por:

Nombre de la métrica: **networkMetrics**

- networkCardName: nombre de la tarjeta de red. (string)

- MTU: (unidad de transmisión máxima) es el tamaño de cada paquete recibido por la tarjeta de red. El valor de MTU para todos los dispositivos Ethernet de manera predeterminada se establece en 1500. Establecer un valor más alto podría poner en peligro la fragmentación del paquete o desbordamientos del búfer. (int)

- IP: ip visible de la tarjeta de red. (string)

- netMask: la máscara de red del sistema. (string)

- broadcastAddress: la dirección que se usará para representar las transmisiones a la red. (string)

- IPv6Address: es la etiqueta numérica usada para identificar la interfaz de red en una red IPv6. (string)

- MACAddress: es la dirección MAC de la tarjeta de red. (string)

- txQueueLen: denota la longitud de la cola de transmisión del dispositivo. Por lo general, lo configura en valores más pequeños para dispositivos más lentos con una latencia alta. (int)

- connectionProtocol: protocolo de conexión utilizado. (string)

- RXPackages: número de paquetes recibidos por la interfaz de red. (int)

- RXErrors: número de paquetes con error recibidos por la interfaz de red. (int)

- TXPackages: número de paquetes transmitidos por la interfaz de red. (int)

- TXErrors: número de paquetes con error transmitidos por la interfaz de red. (int)

- collisions: el valor de este campo debería ser idealmente 0. Si tiene un valor mayor que 0, podría significar que los paquetes están colisionando mientras atraviesan la red, una señal segura de congestión de la red. (int)

2.14 systemAdditionalInfo

La información adicional del sistema la obtendremos del comando uptime. La métrica va a estar nombrada como systemAdditionalInfo y va a estar constituida por:

Nombre de la métrica: **systemAdditionalInfo**

- systemRunningTime: es el tiempo que lleva funcionando el sistema. (string)

- usersLoggedOnNumber: número de usuarios conectados. (int)

- systemLoadAverage1M: promedio de carga del sistema durante el último minuto. (float)

- systemLoadAverage5M: promedio de carga del sistema durante los últimos 5 minutos. (float)

- systemLoadAverage15M: promedio de carga del sistema durante los últimos 15 minutos. (float)

Tras la línea en blanco, tenemos otro párrafo. En él, además, escribiremos una cruz (†).

Chapter 3

Bibliografía

https://pandorafms.com/blog/es/monitorizacion-de-sistemas/