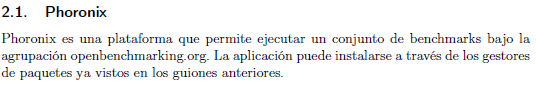


***Benchmarking***

El fin puede ser llegar a comparar, pero es más correcto definirlo como una serie de test para comprobar el rendimiento de tu sistema. Comprobar cuánta carga puede soportar y detectar posibles cuellos de botella. Se puede usar Zabbix y añadiendo un benchmark que estrese al sistema comprobar a través de la interfaz de Zabbix la respuesta del sistema.



<https://openbenchmarking.org/tests> (buscar 2 test que no ejecuten nada en disco y probarlos)

Dentro de esta organización existen benchmarkings de todo tipo (videojuegos, cpu..) y mediante Phoronix podremos lanzar todos estos benchmarks. Ahora mismo no se puede instalar la aplicacion desde los gestores de paquetes de Ubuntu dado que no contiene los repositorios. Por lo que para instalarlo, deberemos descargarnos los paquetes desde un enlace de Phoronix (.deb) y con el dpkg instalamos el paquete.

URL descarga

**EJERCICIO1** (no elegir test de disco duro porque se rompe el disco duro del ORDENADOR! hacerlo de cpu (sudokut), ram, ...).

Si hacemos una carpeta en /var/www/html/phoronix y metemos dentro los resultados de los test, como tenemos el servidor web activo, si metemos en el navegador URLubuntu o centos /phoronix/test y podremos ver los resultados graficamente.

**EJERCICIO OPCIONAL**

Ejecutar uno de los benchmarks usando una imagen del contenedor. Es decir, en vez de ejecutar el benchmark en la máquina virtual directamente, podemos ejecutarlo desplegando el contenedor con el docker y ejecutando el test. En el caso de la mv es un hardware virtualizado y el de docker es el hardware nativo del host, por lo tanto, seguramente encontremos diferencias entre el ejercicio1 y el ejercicio1 opcional.

Para instalar Phoronix en las distribuciones nuevas de Linux 2020:

*wget URL* y *dpkg -i paquete.deb*

Otro tipo de benchmarks muy utilizados en los servidores web son dos. Apache Benchmark (ab) que viene instalado con el Apache. Este, a una URL le lanza peticiones (tantas como le digamos) y nos da el tiempo de respuesta.

Apache JMeter es un benchmark desarrollado en Java. Nos da una forma mucho mas profesional para testear un servidor web ya que permite hacer varios procesos a la vez, varias hebras a la vez, simular situaciones reales a las que se puede ver sometido un servidor web en cualquier momento de su funcionamiento. Permite llevar a cabo test de carga muy reales, permitiendo crear concurrencia real en el sistema porque permite ejecutar varias hebras dentro de una cpu asi como distribuir la creación de carga en varias máquinas.

**Ejercicio2**

Vamos a usar JMeter para testear una aplicación que ya está hecha, que contiene dos contenerdores (uno para la BD de la app y otro para simular la app en si). Esto está en github.

Esta aplicación simula un sistema de notas, como si fuesen las notas de la Universidad. Hay establecidos una serie de alumnos y profesores, notas... Lo que monta esta práctica es una BD que contiene los alumnos, los profesores y las notas. Entonces la idea es usar JMeter para simular que los alumnos se están conectando para saber sus notas y los profesores se están conectando para saber los alumnos. Creamos un test que vaya logeando a alumnos y profesores y pidiendo ciertas cosas. Nosotros solo tendremos que arrancar la aplicacion, sus docker y demás.

Se usa Docker Compose, que permite abrir varios docker a la vez y conectarlos ya que sino tendríamos que estar abriendo docker a docker los contenedores. Permite abrilos y administrarlos todos a la vez. Para acceder a la aplicación usamos la dirección por el puerto 3000. SEGUIR README DEL GIT de la aplicación.

El proceso de consulta es el siguiente:

1º) Tenemos que mandar una petición al servidor con las credenciales de un alumno o administradores. El servidor nos devuelve un token (cadena de caracteres que nos permite hacer peticiones con el login al servidor, si no tienes el token no están permitido para hacer peticiones más allá del login)

2º) Una vez tenemos el token ya podemos solicitar datos.

Con JMeter lo que vamos a hacer es automatizar todo esto proporcionandole el .csv de los alumnos. Será de forma gráfica. Debemos comprobar que el diseño de la aplicación concuerda con el proporcionado en el git. En este, vemos que tenemos:

En ETSIIT Alumnos deberemos poner los datos del servidor. Luego vemos que tenemos dos hebras, una para los alumnos (le pasamos el .csv de los alumnos) y otra para los administradores (le pasamos el .csv de los administradores). En “Login Alumno” y “Login Administrador” debemos configurar el login donde se extrae el token que te trae el servidor. El “espera un poco” es una espera aleatoria que se le pone al sistema para simular una carga real. Entre alumno y alumno se le mete una espera aleatoria (una vez que tenemos el token ya podemos recuperar los datos del alumno). Igual para administrador.

Esto es lo que hay en github. Lo que nosotros tenemos que hacer son los puntos del ejercicio2.

***Breve introducción a los contenedores: docker*** (hacer todo esto antes de comenzar la práctica con la aplicación, ej2)

La MV, dada la infraestructura del sistema monta un SO para cada una de las máquinas virtuales con su hardware virtualizado. Sin embargo, docker no, docker usa la infraestructura del sistema y el SO del host para montar la interfaz de docker y aquí desplegar las aplicaciones, así las aplicaciones pueden tener conexión entre ellas de forma nativa (en MV no se podría de forma nativa, si a través de servidores web por ejemplo) y las aplicaciones comparten la misma memoria.

Gracias a docker se han popularizado mucho los microservicios que son pequeños servicios que se abren, hacen lo que tienen que hacer y se cierran, ahorrando así recursos, ya que no tiene que estar siempre activo en estado “ocioso” o esperando peticiones.

Dado al desarrollo de la tecnología web muchas veces no nos interesa tener muchos servicios a la vez, entonces, la solución sería tener los servicios principales activos y si llega una peticion que requiere otro servicio en concreto, se levanta este servicio con docker, se devuelve la petición y se vuelve a bajar el servicio (microservicios).

Instalación Docker:

Para instalar docker en Ubuntu Server seguimos el punto 2.3.2. Tener en cuenta que al añadir el repositorio es todo una linea, y si no funciona, debemos cambiar $(lsb\_release -cs) por focal.

Una vez tenemos el repositorio añadido, actualizamos los repositorios y ya podemos instalar docker.

Para poder usar docker sin tener que ser root, debemos añadirnos al grupo docker y volver a logearnos. Y comprobamos si está instalado.

Instalación Docker Compose:

Cuando tengamos docker y docker compose ya podemos instalar la aplicación.

Nos bajamos la aplicación del github y hacemos lo que se indica en 2.3.4. Una vez hecho ya tendríamos desplegado por el puerto 3000 la aplicación. Para comprobarlo arrancamos el archivo pruebaEntorno.sh y si devuelve los datos de una alumna la aplicación se ha desplegado bien.

<dockerfile>:

En este caso el docker se ha creado de un docker previo de node (FROM). Y ha hecho una carpeta dentro del docker que se llama app(RUN) y lo ha copiado al docker a la carpeta app (COPY). Lo a expuesto el docker por el puerto 3000 (EXPOSE) ya que se entra por este puerto a este docker. Como directorio de trabajo ha puesto la carpeta que ha creado (WORKDIR). Actualiza npm(RUN), pone el entorno de node como producción(ENV) y lo arranca(CMD).

Esto es un dockerfile, como se construye un docker. Normalmente heredas de una imagen que ya existe en docker y ejecutas los comandos que tu quieres. Y se te queda como una especie de máquina virtual, un CONTENEDOR con esa configuración. Cuando tu haces el docker run lo que hace es arrancar esa configuración. Esto para node.

Para mongo igual. Hereda de mongo. Copia un script sh y le cambbia los permisos y lo inicializa. En el script inicializa una base de datos de mongo.

Para juntarlos lo que se hace es el docker compose. Una vez que están abiertos le dices que puertos quieres y cómo se inicializa y tal. Descripción de la aplicación:

(FIGURA 2.3 git)

Tenemos JMeter que manda las peticiones al servidor correspondiente (que es Ubuntu server). Ubuntu Server tiene con el docker levantado la aplicacion de node y mongodb. JMeter va a hacer las peticiones al puerto 3000 e internamente ya están configurados los docker para que por el puerto 27017 node se conecta a mongo y mongo le devuelva (internamente está hecha esta conexión).

<Figura 2.6> Diseño de la aplicación

(Donde pone cliente es JMeter)

Primero mandamos al servidor una petición POST para que nos devuelva el token. El servidor internamente comprueba el usuario en la base de datos mongo y devuelve al servidor si todo ha ido bien un 200 con el token (en Consulta Info Jmeter recibe el token). Una vez que lo tenemos hacemos una petición GET usando el token y ya nos devolvería el servidor la información del usuario.

En JMeter se hacen 2 peticiones,una con el token y otra con la información y se va a devolver dos respuestas, una con el token si el login ha ido bien y otra respuesta con la información.

***Ajuste de sistema y de servicios: Performance engineering***

Para tocar parámetros hay que saber lo que se hace. Se puede hacer modificando los valores dentro de los archivos que podemos encontrar en la estructura de directorios que separa estos archivos en los distintos subsistemas, cambiando así el comportamiento del sistema sin tener que reiniciarlo. Pero esto es bastante peliagulo.

Para evitarlo, cuando tenemos que modificar parámetros del kernel (memoria, dispositivos...) lo haremos con sysctl. (no confundir con systemctl).

“El trabajo opcional meh.”

ERROR EN INSTALACION DE DOCKER Y DOCKER COMPOSE (AUNQUE FUNCIONA)

