Benchmarking y ajustes del sistema

# Julio A. Fresneda García – 49215154F

# Phoronix

Phoronix es una plataforma en la cual podemos encontrar una gran cantidad de tests para hacer benchmarks. Podemos obtener tanto tests como suites (conjuntos de tests). Photonix dispone de phoromatic, la cual es una interfaz gráfica.

## *Instalación y configuración de Phoronix*

Para instalar phoronix, ejecutamos el siguiente comando:

**$ sudo apt-get install phoronix-test-suite**

Vamos a necesitar algunas dependencias, como php, php-sqlite3, php-zip y firefox, las cuales debemos instalar también.

Lo primero que hay que hacer es movernos a:

***/usr/bin/phoronix-test-suite***

Y ejecutar el archivo “phoronix-test-suite” para instalarlo.

Después, tenemos que configurarlo. Para eso abrimos el archivo:

***~/.phoronix-test-suite/user-config.xml***

Lo único que hay que modificar es asignarle un puerto en RemoteAccessPort. En este caso he puesto 20000.

## Configuración de Phoromatic

Vamos a lanzar ahora phoromatic. El comando es:

***phoronix-test-suite start-phoromatic-server***

Para acceder a phoromatic, abrimos Firefox. Como Ubuntu Server no tiene interfaz gráfica, hay que abrirlo desde Ubuntu Desktop (U otro sistema con interfaz gráfica) conectado al servidor por ssh.

Para esto, usamos el comando:

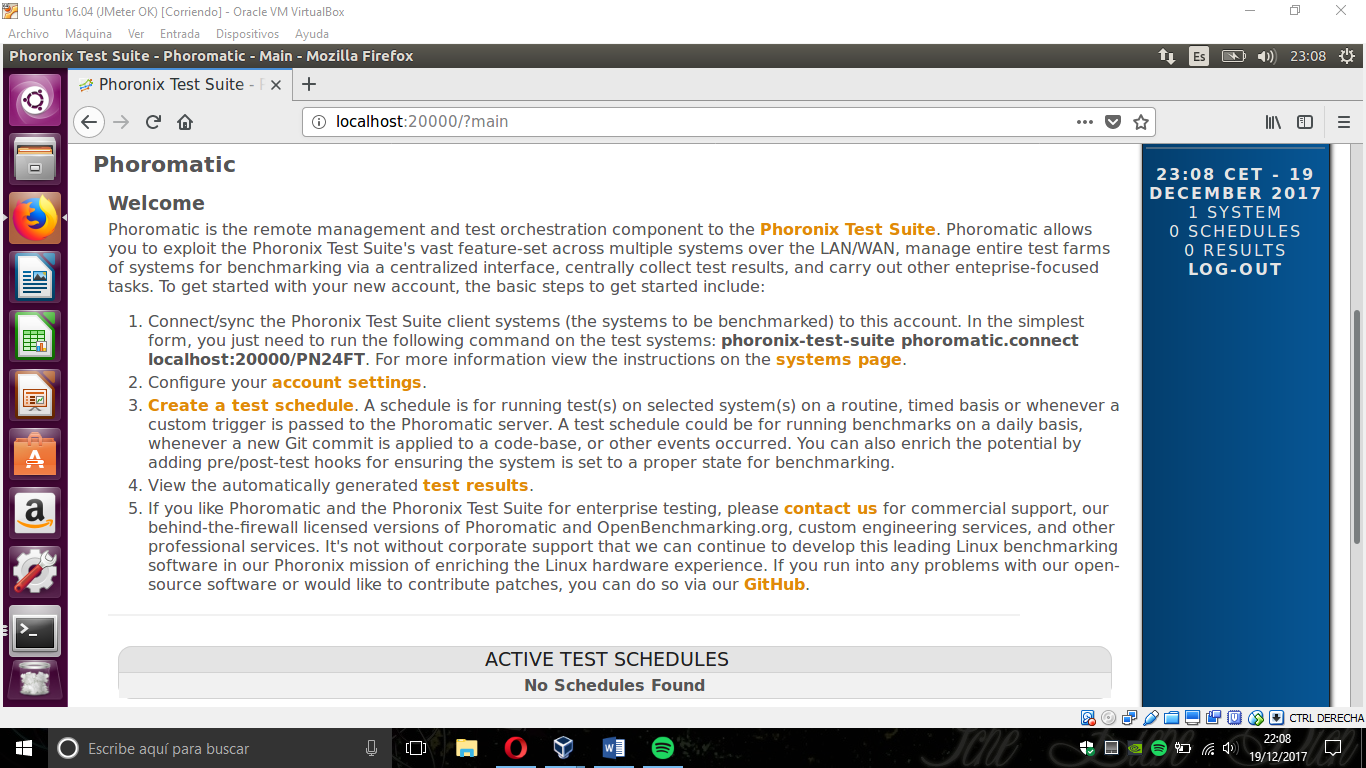
***ssh –X http://192.168.57.102***

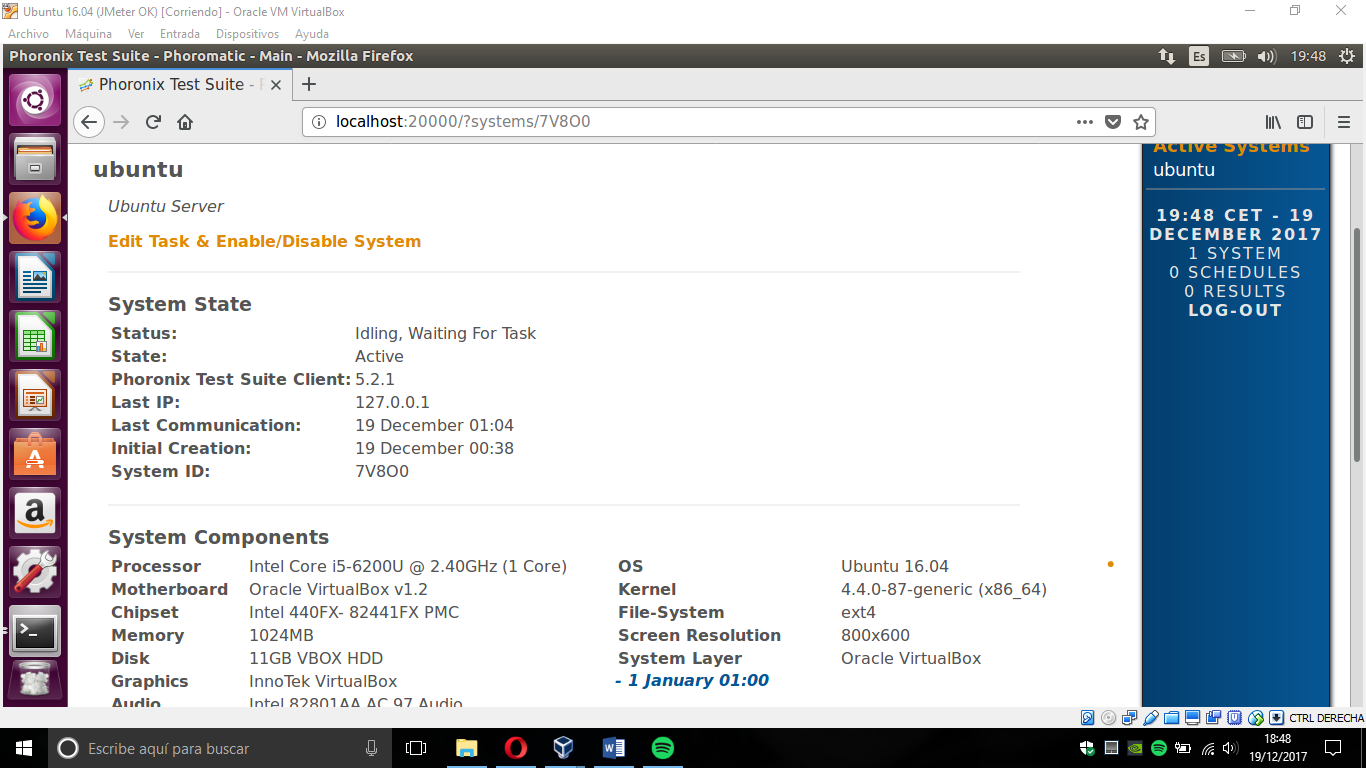
Abrimos Firefox, y nos metemos a localhost:20000.

Lo primero que hay que hacer es configurarlo:

Vamos a crear una nueva cuenta con user juliofgx17, pass practicas,ISE

Después seguimos los pasos que la propia web nos pone:



Una vez completado, ya tendríamos Phoromatic listo:

## Benchmarking

Vamos a instalar y probar distintos tests: C-Ray, Compress-7zip y AIO-Stress.

Estos tests se han probado en un PC con las siguientes características:



## C-Ray

C-Ray es un test de CPU. El comando es para lanzarlo es:

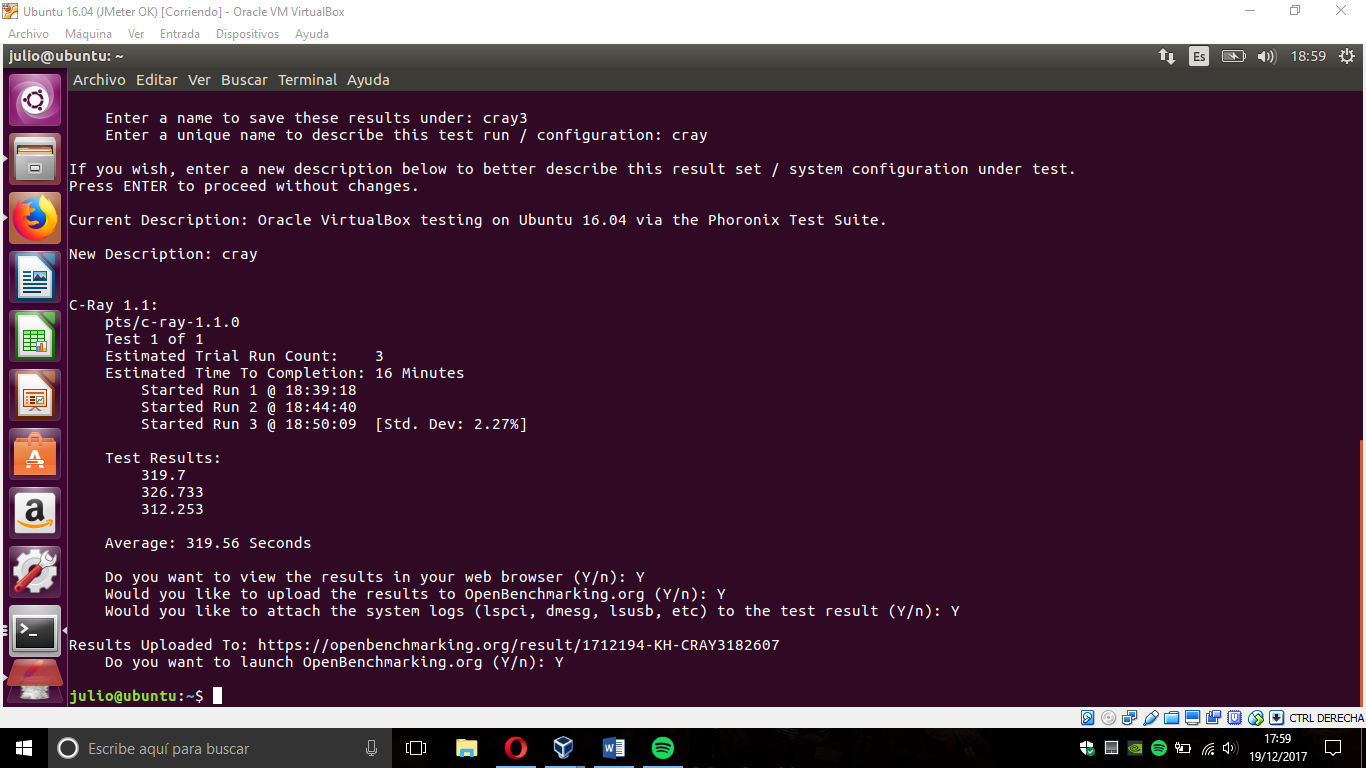
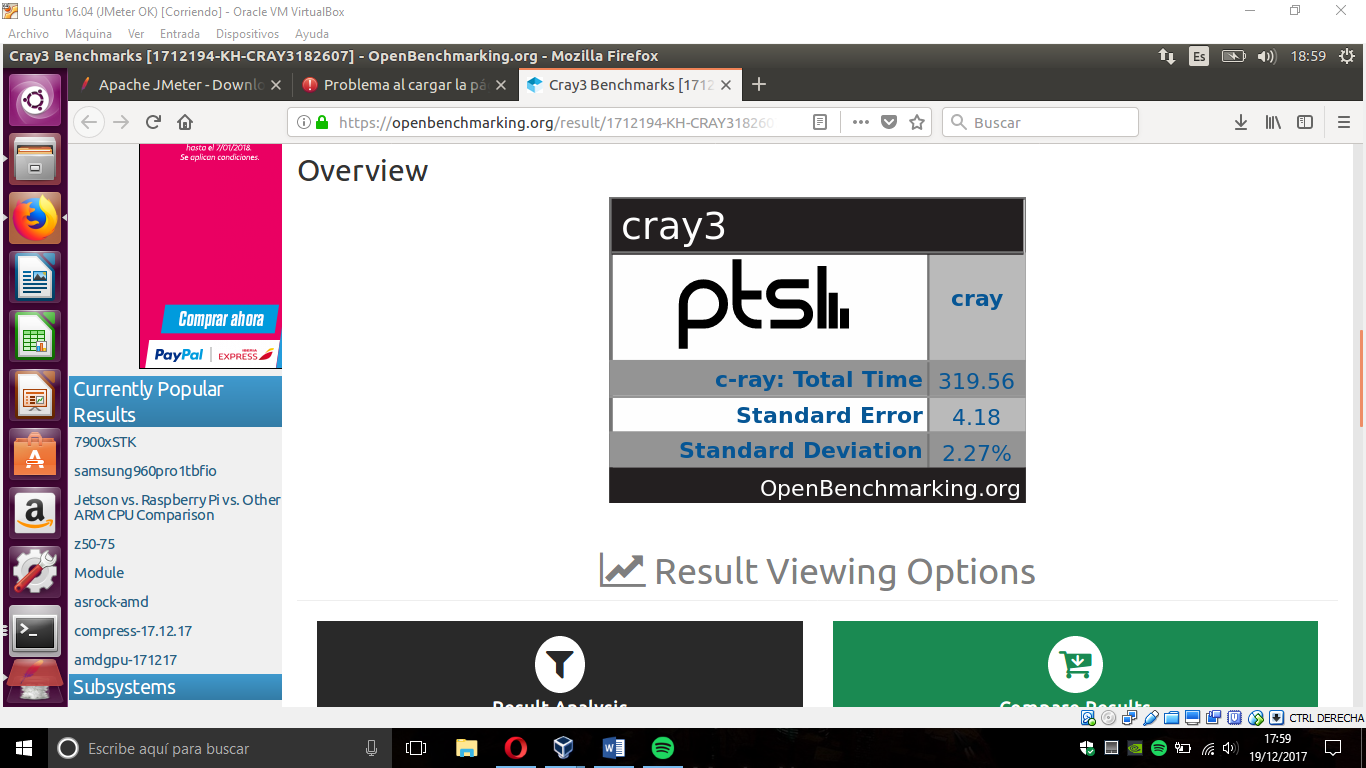
***phoronix-test-suite benchmark c-ray***

Este test consiste en un raytracer (algoritmo para síntesis de imágenes tridimensionales) simple diseñado para probar el rendimiento de CPU con operaciones de coma flotante.

Esta prueba tiene varios subprocesos (16 subprocesos por núcleo), disparará 8 rayos por píxel para suavizar, y generará una imagen de 1600 x 1200.

Como vemos en las siguientes imágenes, el test ha consistido en ejecutar el algoritmo antes descrito tres veces, y calcular la media.

En este caso, mi pc tarda en generar la imagen, de media, 319.56 segundos. Los resultados son éstos:



## Compress-7zip

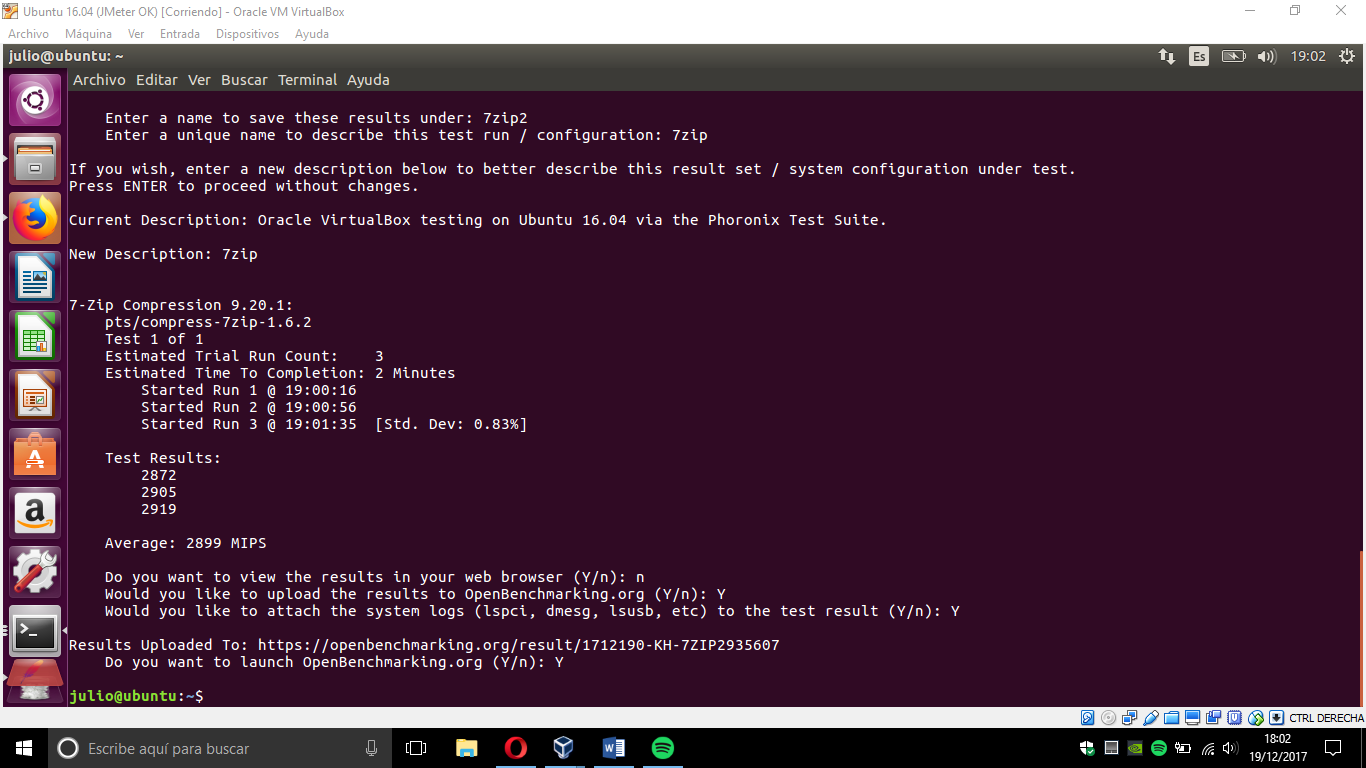
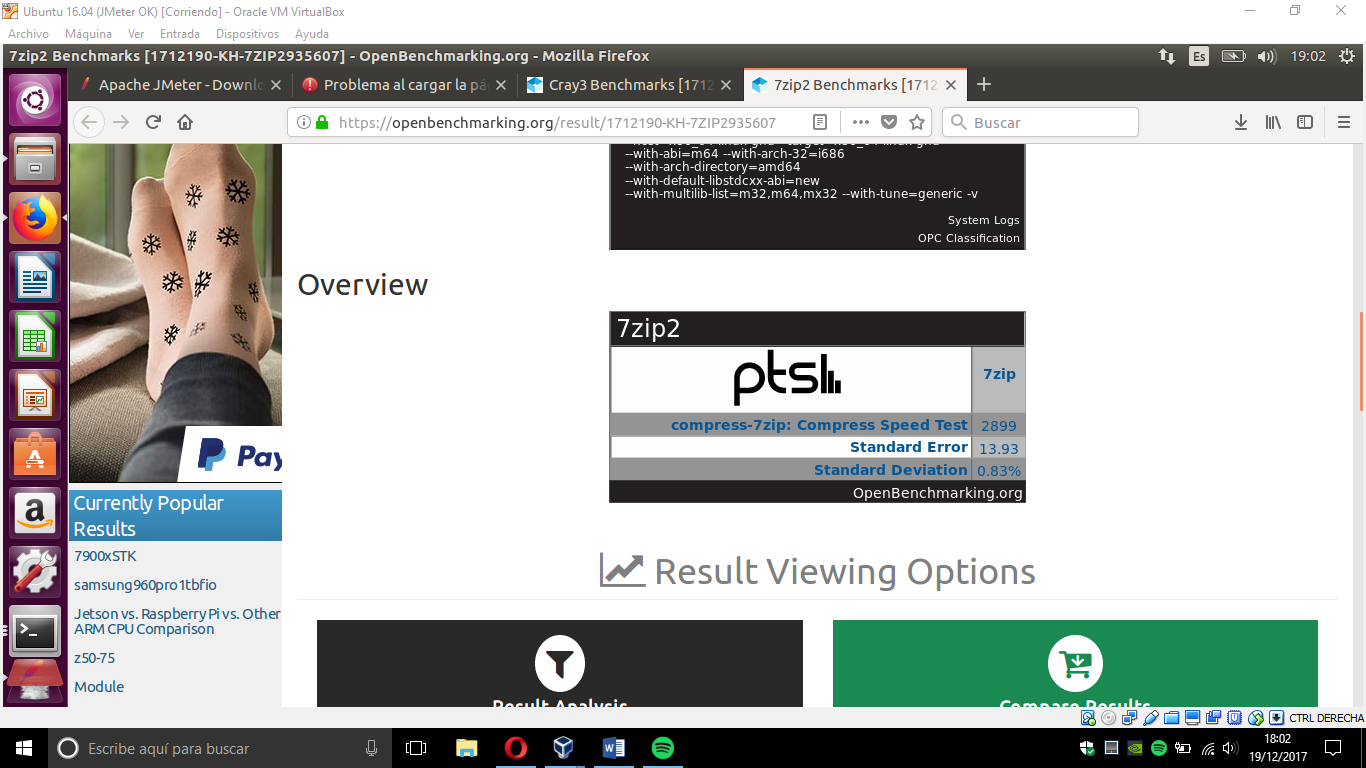
También es un test de CPU.

El comando es:

***phoronix-test-suite benchmark compress-7zip***

Compress-7zip es un test de velocidad de compresión, usando la característica de benchmarks integrada de p7zip (programa 7-Zip).

Como vemos, la media obtenida es de las 3 ejecuciones del algoritmo de compresión es de 2899 MIPS (Millones de instrucciones por segundo).

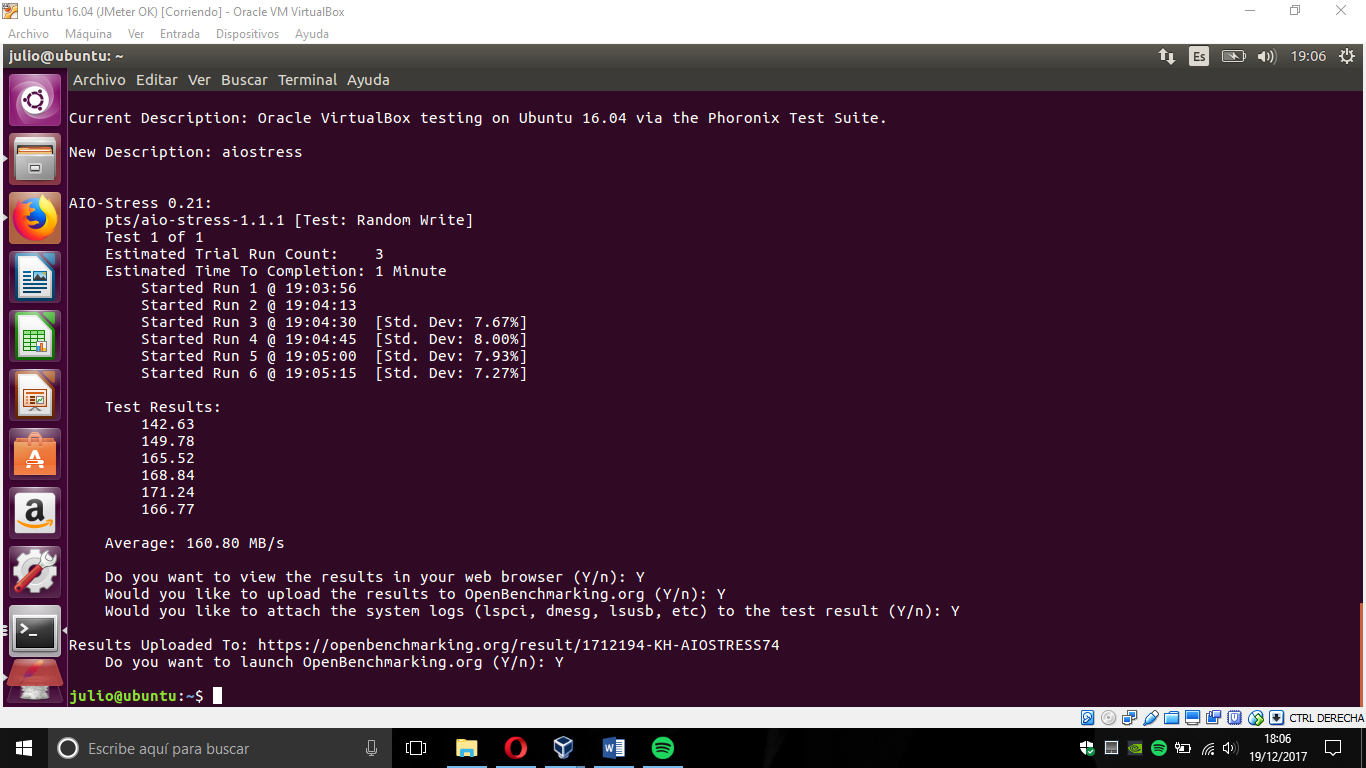
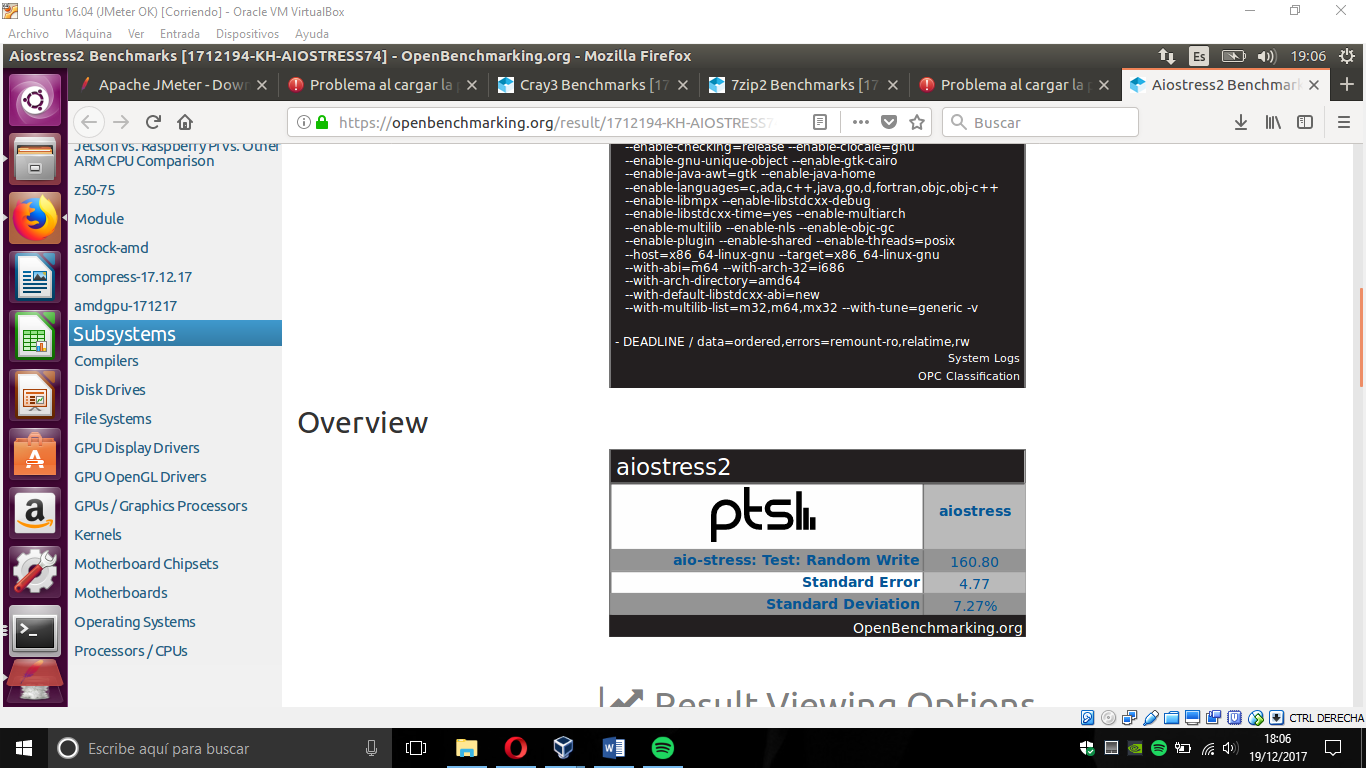


## AIO-Stress

Es un test de disco duro. El comando para lanzarlo es:

***phoronix-test-suite benchmark aio-stress***

AIO-Stress es un test asíncrono de escritura y lectura de datos creado por SuSE. Este test usa un archivo de texto de 2GB y escribe en bloques de 64kb. Aunque los loops estimados eran de 3, el test ha llevado a cabo el algoritmo 6 veces, obteniendo una media de escritura de 160 MB/s, con un error del 4.77% y una desviación del 7.27%



# Apache HTTP Server benchmark

Ya hemos hecho benchmarks a la CPU y al disco duro de nuestro servidor. Ahora vamos a testear a Apache.

Para eso, tenemos dos opciones: ab y JMeter.

## ab (Apache Benchmark)

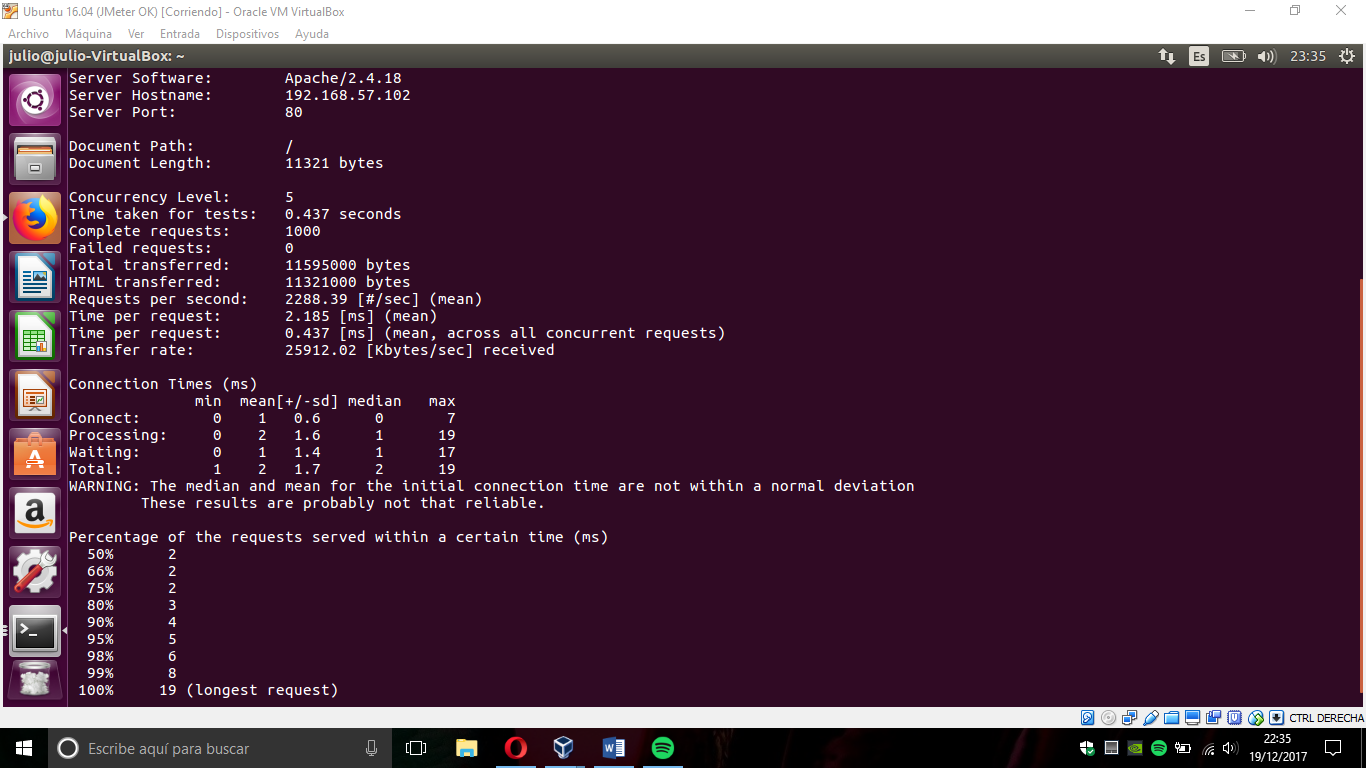
ab sirve para mostrar cuántas solicitudes por segundo puede ejecutar la instalación de Apache. Para lanzarlo, ejecutamos el siguiente comando desde una terminal:

***ab -c5 -n1000*** [***http://192.168.57.102/***](http://192.168.57.102/)

-c ajusta la concurrencia (las peticiones que se envían a la vez).

-n ajusta el número de peticiones que se envían.

Ejecutando el comando, obtenemos el siguiente resultado.



Estos resultados se interpretan de la siguiente manera:

**Server Hostname**

La IP del servidor testeado

**Server Port**

El puerto por el que se ha testeado el servidor

**Document Path**

El documento del servidor que se ha testeado. En este caso, /

**Document Length**

Longitud del documento

**Concurrency Level**

Número de peticiones simultáneas

**Time taken for tests**

Tiempo que ha tardado el test desde que el primer socket de conexión es creado hasta el momento que se obtiene la última respuesta. Es decir, tiempo que ha tardado en completarse el test

**Complete requests**

Número de peticiones completadas con éxito

**Failed requests**

Número de peticiones que han fallado

**Total transferred**

El número total de bytes recibidos desde el servidor (y enviados desde nuestro PC)

**HTML transferred**

El número total de bytes del documento html (/ en nuestro caso) recibidos

**Requests per second**

Peticiones enviadas y recibidas en un segundo

**Time per request**

La media del tiempo que toma en enviarse una petición (en ms). El primer valor es calculado con la fórmula: (nivel de concurrencia) \* (tiempo que toma cada petición) \* 1000 / tiempo total,  mientras que el segundo valor es calculado con la fórmula: (tiempo que toma cada petición) \* 1000 / tiempo total

**Transfer rate**

Bytes transferidos por segundo, se calcula con la fórmula:  (bytes totales leídos) / 1024 / (tiempo que toma)

En cuanto a la concurrencia, si lo hacemos con –c1 (sin concurrencia), el test tarda de media 1.3 segundos, lo cual tiene sentido. Sin embargo, si usamos un número grande, –c50, por ejemplo, tarda de media unos 0.4 segundos, un poco más que con –c5. Esto posiblemente sea porque el servidor no tiene tantas hebras para llevar a cabo la concurrencia y se produce cuello de botella.

## Apache JMeter

JMeter es un software que puede medir y generar carga de forma genérica para varios elementos.

Para instalar JMeter, lo descargamos desde su página web, y lo extraemos. Una vez extraído, ejecutamos

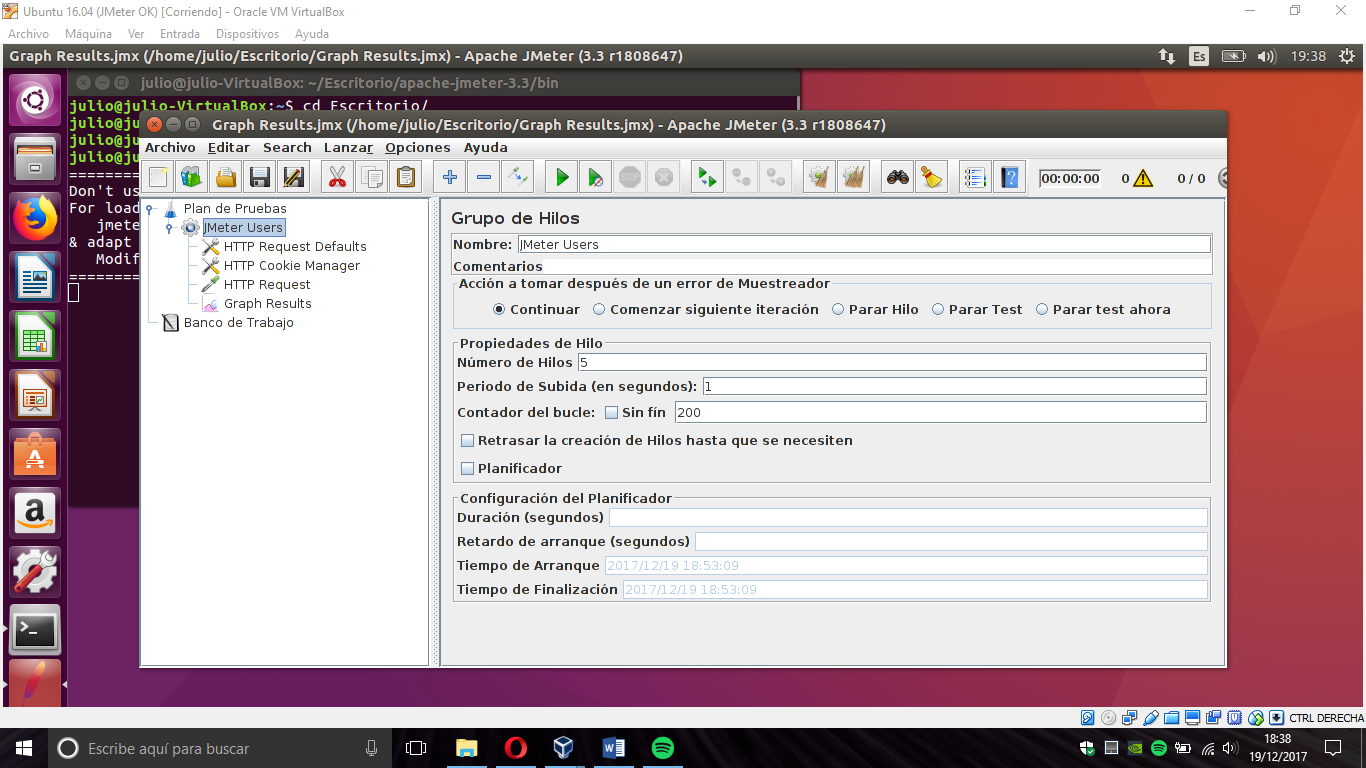
***./jmeter***

## Creando un Test

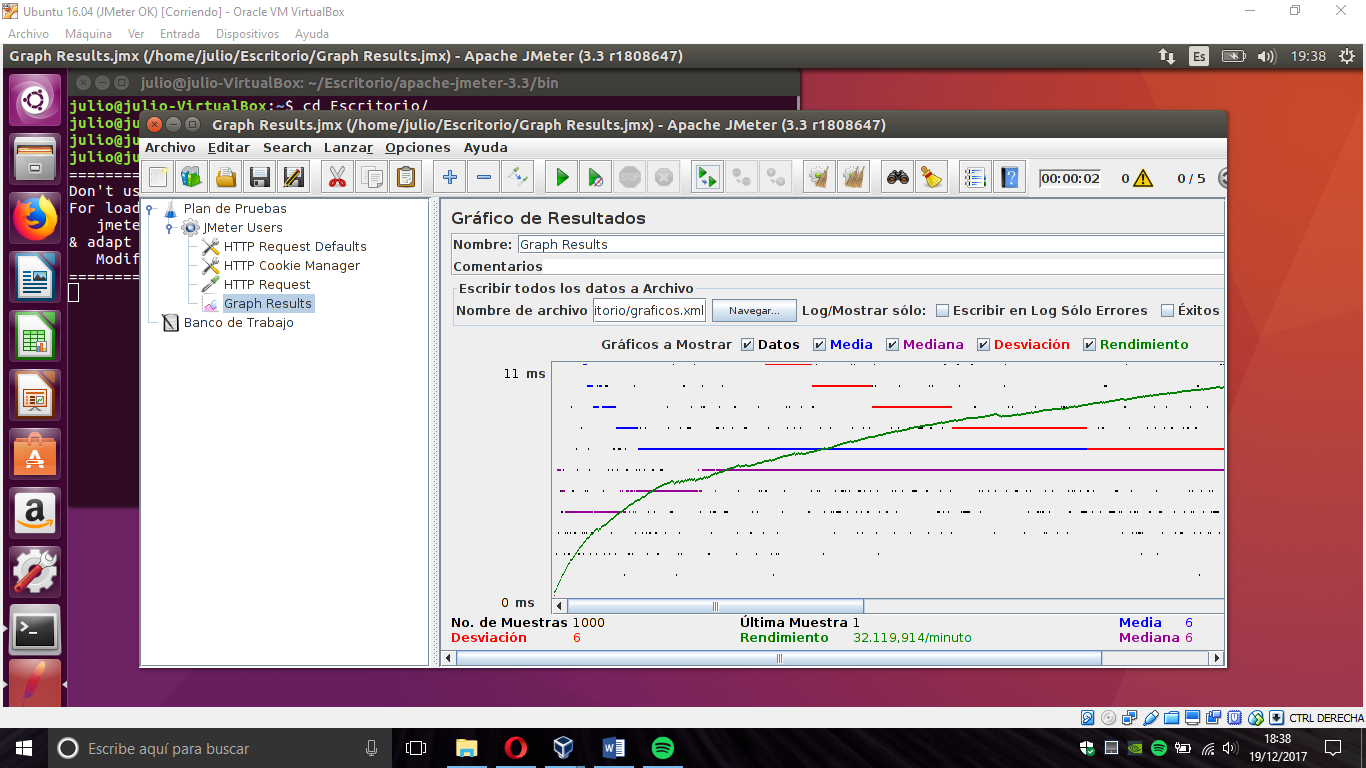
Lo primero es crear un test básico. Para eso, debemos crear lo siguientes elementos:

* Thread Group, el cual contendrá el resto de elementos, y donde configuraremos la concurrencia y el número de peticiones.
* HTTP Request Defaults, donde configuraremos cosas como la IP y el puerto
* HTTP Cookie Manager, para manejar cookies
* HTTP Request, es decir, la petición HTTP
* Graph Results, el cual es un Listener, y nos sirve para ver los resultados gráficamente.

En nuestra configuración vamos a usar 5 hilos concurrentes, y 200 bucles, lo que hacen un total de 1000 peticiones. En el elemento “HTTP Request Defaults” pondremos la IP del servidor, y en “HTTP” Request, el path, ***/*** en este caso.



Una vez ejecutado el test, podemos ver gráficamente la media, mediana, desviación típica y rendimiento.



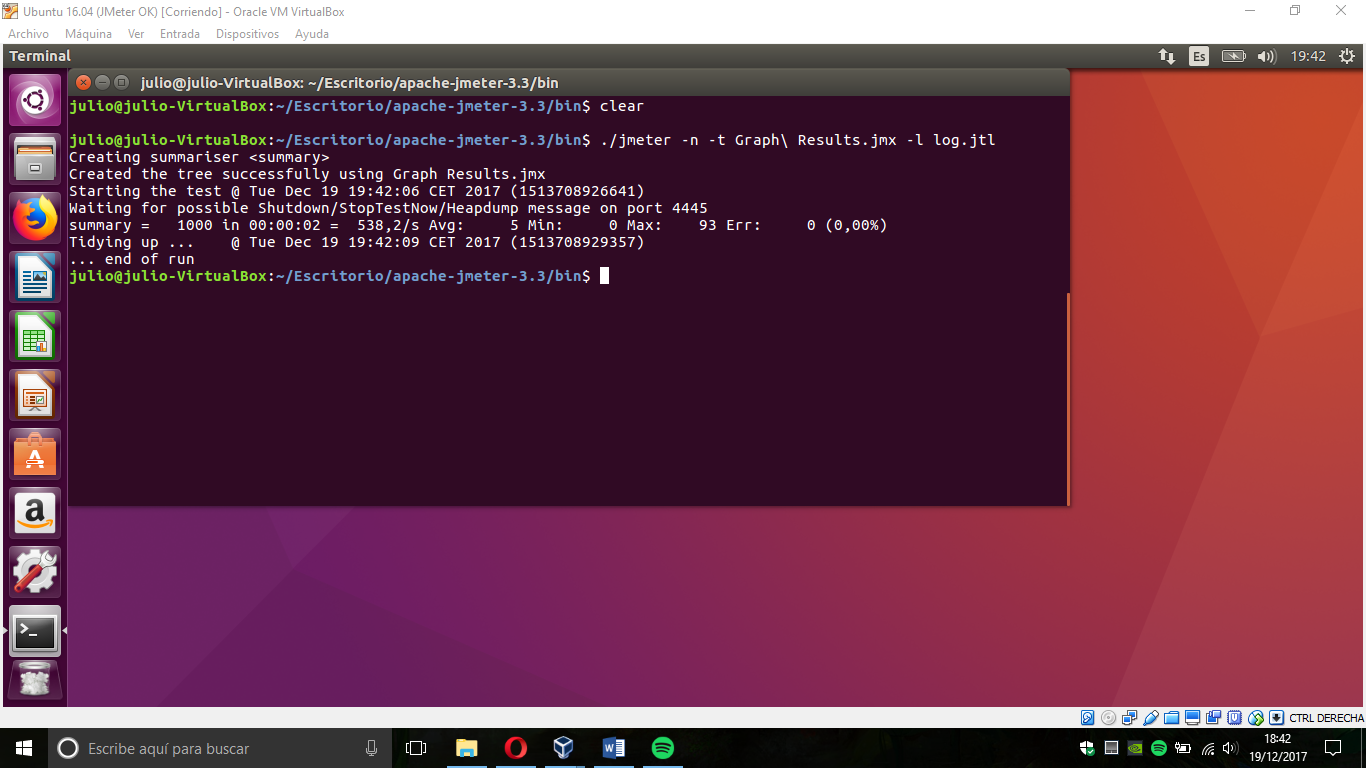
Gracias a la interfaz gráfica podemos ver que funciona correctamente y configurarlo con facilidad, pero una vez creado el test, lo ideal es ejecutarlo en modo non-gui (comandos).

## Ejecutando el test

Para ejecutar el test usamos el siguiente comando:

***./jmeter –n –t TestJMeter.jmx –l log.jtl***

Donde –n nos dice que no usamos interfaz, en -t <archivo> pondremos el archivo que contiene el test a ejecutar, y en –l el archivo donde se guardarán los logs. Podemos ver el resultado:



Como vemos, se envían y reciben 538,2 peticiones por segundo, algo más lento que usando ab.

# Cambiando valores del sistema para optimizar

Vamos a intentar optimizar nuestro servidor. Para esto modificaremos algunas variables de archivos de configuración.

## Kernel

He modificado algunos archivos de /proc/sys/kernel, como “num\_max\_threads”, o “perf\_cpu\_time\_max\_percent”. Para ello he usado el comando:

***sysctl –w kernel.<nombre>***

Sin embargo, no he notado diferencias destacables, seguramente porque el sistema no estaba demasiado ocupado. Con más carga creo que sí que habría diferencias.

## LAMP

## HostnameLookups

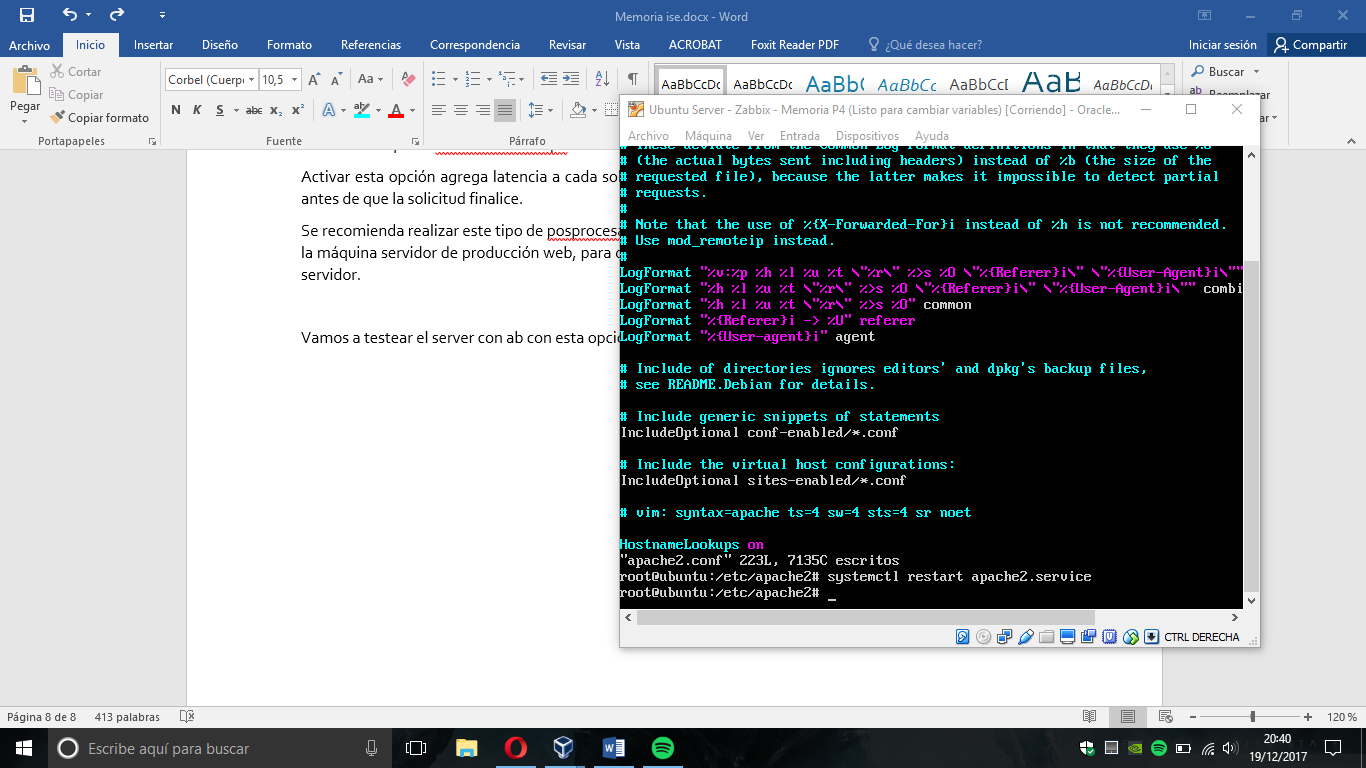
En Apache tenemos la opción HostnameLookups, que viene desactivada por defecto.

Activar esta opción agrega latencia a cada solicitud porque requiere una búsqueda de DNS para completar antes de que la solicitud finalice.

Se recomienda realizar este tipo de posprocesamiento de los archivos de registro en una máquina que no sea la máquina servidor de producción web, para que esta actividad no afecte negativamente al rendimiento del servidor.

Vamos a testear el server con ab con esta opción primero activada, y después desactivada.

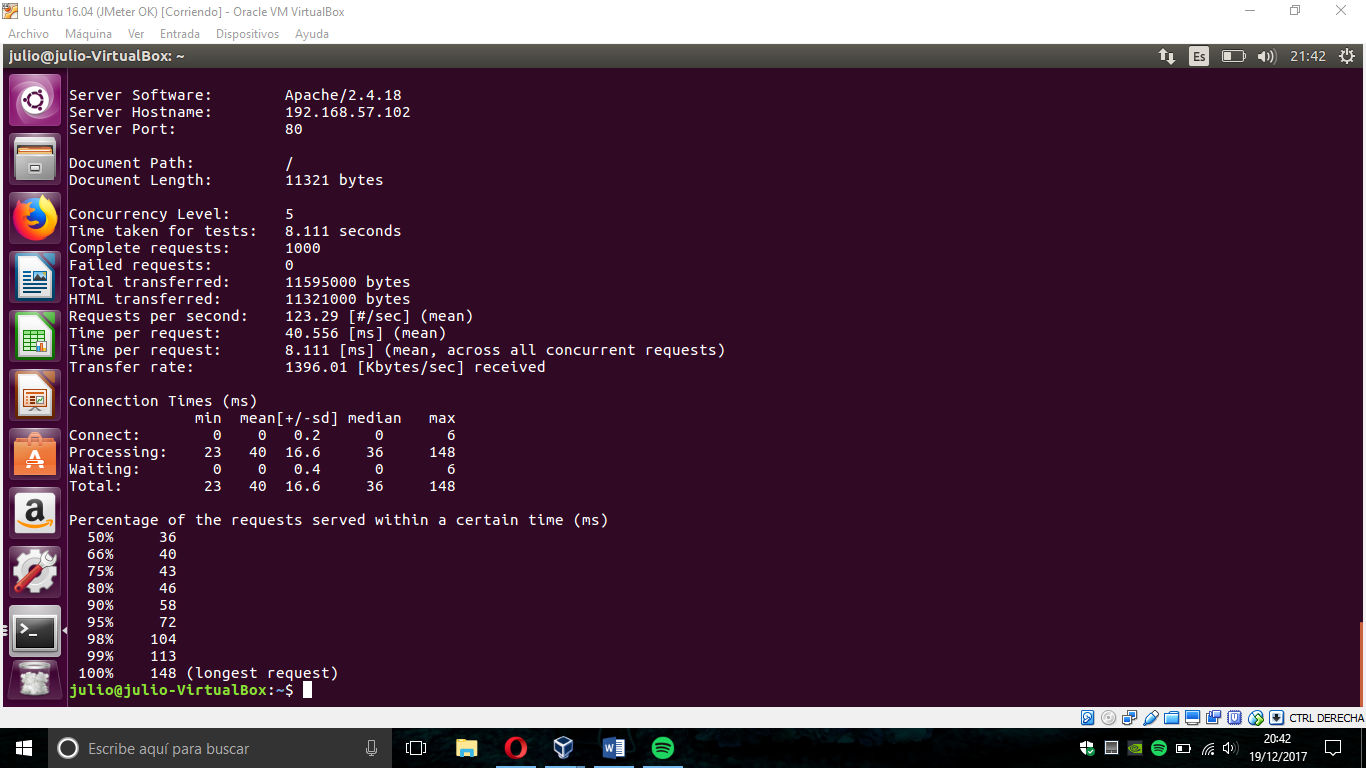
Esta variable se cambia (o añade, si no está) en el archivo /etc/apache2/apache2.conf



Usando desde otro sistema el comando

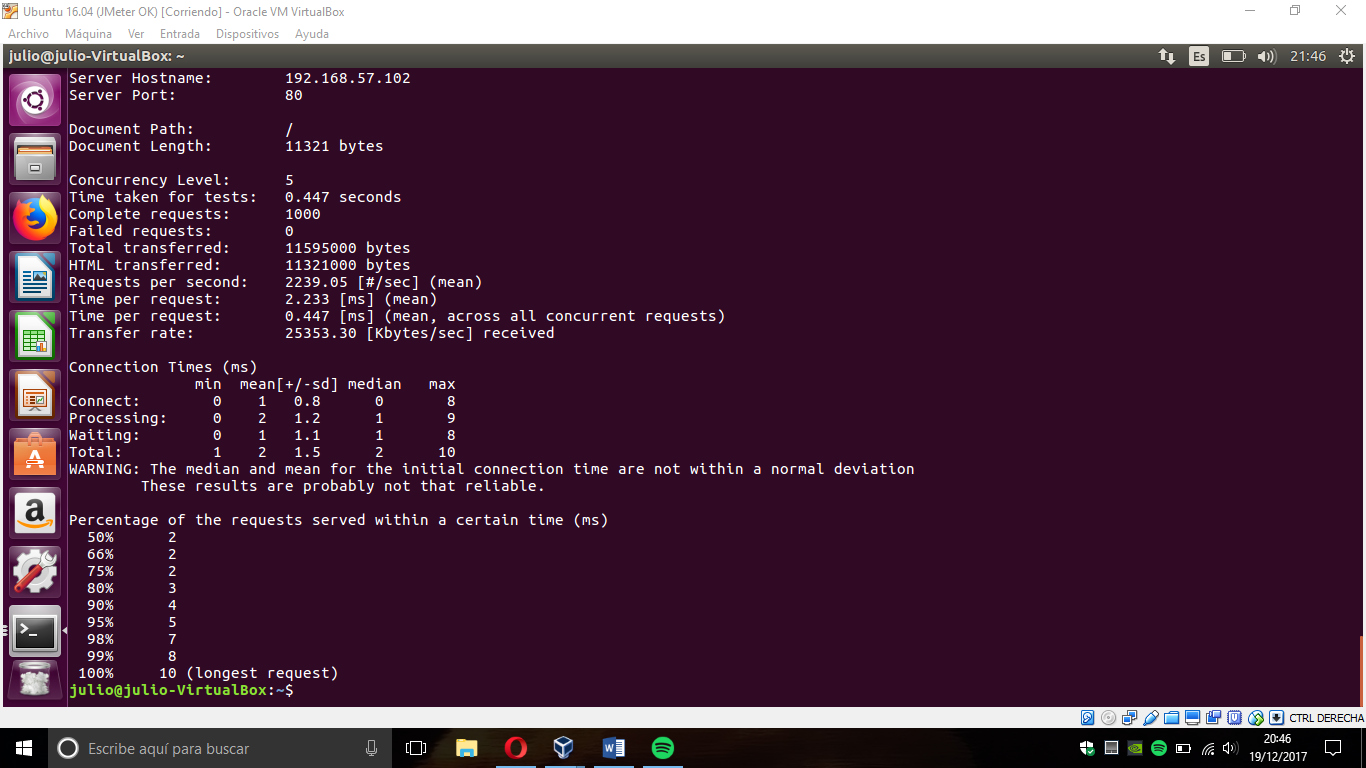
***$ ab –c5 –n1000*** [***http://192.168.57.102/***](http://192.168.57.102/)

Obtenemos los siguientes resultados.



Se han enviado 1000 peticiones, con una concurrencia de 5, y el test ha tardado 8.111 segundos.

Ahora vamos a probar a hacer el mismo test con la opción HostnameLookups desactivada:



Vemos que el test ha tardado 0.447 segundos, es decir, aproximadamente 16 veces menos que en el anterior test. Por lo tanto, nos conviene mantener la opción desactivada. Este resultado es normal, ya que la búsqueda DNS para cada solicitud consume mucho tiempo.

Como la diferencia es tan evidente, no es necesario hacer varias veces el mismo test y sacar la media, con una sola prueba basta.

## FollowSymLinks y SymLinksIfOwnerMatch

Siempre que en las opciones no esté FollowSymLinks activado, o si tenemos SymLinksIfOwnerMatch activado, Apache necesitará emitir llamadas adicionales al sistema para verificar los enlaces simbólicos. (Una llamada adicional por componente de nombre de archivo).

Por ejemplo, si en el archivo de configuración tenemos:

***<Directory />***

***Options SymLinksIfOwnerMatch***

***</ Directory>***

y se realiza una solicitud para el URI /index.html, Apache realizará lstat (2) en “/www”, “/www/htdocs” y “/www/htdocs/index.html”. Los resultados de estos lstats nunca se almacenan en caché, por lo que se producirán en cada solicitud. Para evitar esto, podemos añadir las siguientes líneas al archivo de configuración:

***<Directory />***

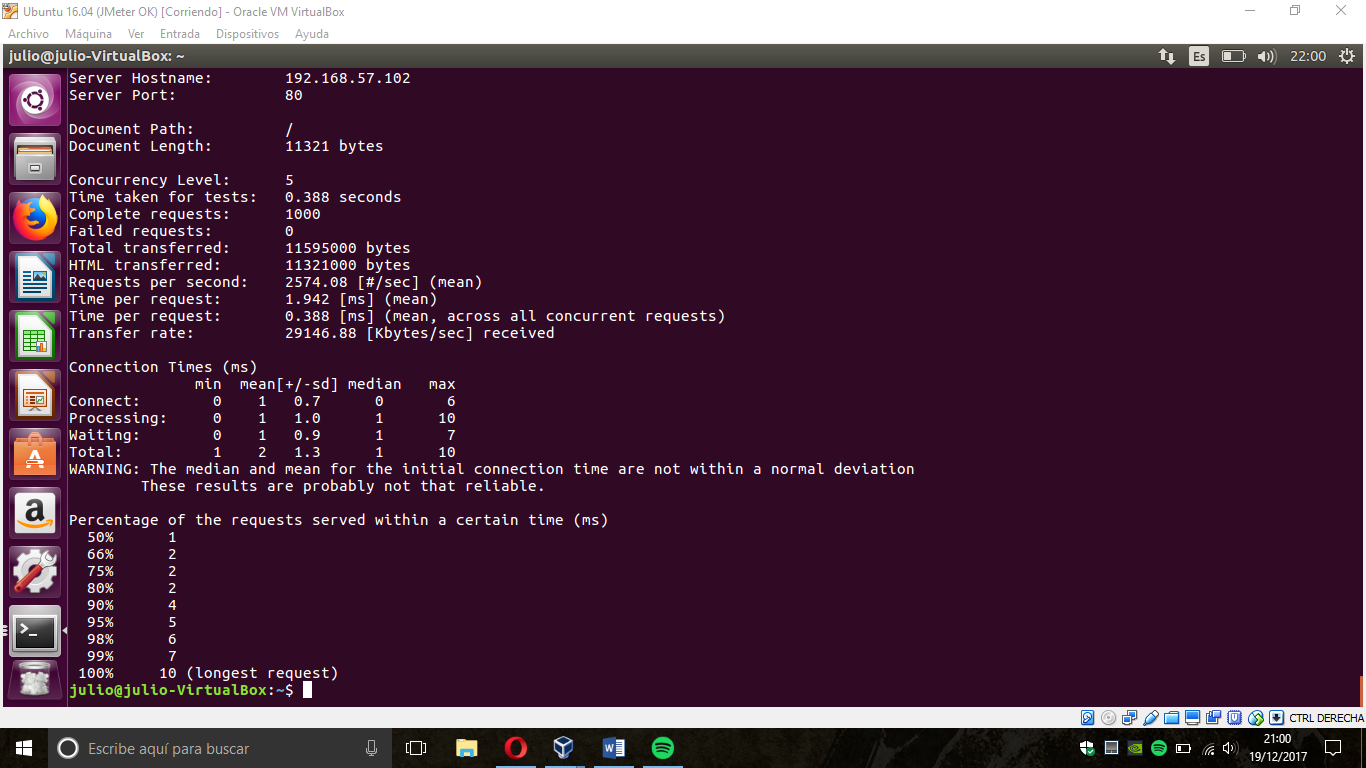
***Options -FollowSymLinks + SymLinksIfOwnerMatch***

***</ Directory>***

Esto al menos evita las comprobaciones adicionales, ahorrándonos unos milisegundos.

Vamos a probar añadiendo al archivo de configuración estas opciones. Con las opciones por defecto, he ejecutado el test ab 10 veces, y la media aritmética de lo que han tardado es de 0.456 segundos.

Con estas opciones, he vuelto a hacer 10 testeos, y la media aritmética de lo que ha tardado es de 0.402 segundos. La diferencia, como vemos en esta captura, no es demasiado notable, pero nos ahorramos esos milisegundos de las llamadas para verificar los enlaces simbólicos.



## Memory-limit

En php hay una variable llamada memory-limit. Se encuentra en:

***/etc/php/7.0/apache2/php.ini***

La variable tenía 128M, y yo le he puesto 16M, pero al cambiarla, en los tests no he notado diferencia. Seguramente sea porque ni siquiera haya necesitado los 16MB completos.

## Opciones de MySQL

Hay muchas opciones para optimizar MySql, sin embargo, en mi equipo no he notado diferencias, y puede ser por dos razones:

* Nuestro servidor es demasiado simple (sólo tiene el archivo http por defecto)
* Estas opciones están pensadas para optimizar un servidor con Moodle

Las opciones se pueden cambiar en ***/etc/mysql/my.cnf*** y son las siguientes:

* ***query\_cache\_type = 1***
* ***query\_cache\_size = 36M***
* ***query\_cache\_min\_res\_unit = 2K***
* ***table\_cache = 256***