Ingeniería de Servidores (2019-2020)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 4 : Benchmarking y Ajuste del Sistema

Índice

OBJETIVOS MÍNIMOS

1. Conocer varios benchmarks para diferentes servicios.
2. Saber comparar distintas configuraciones (o implementaciones) de servicios en base a benchmarks.
3. Aplicar un test de carga a una aplicación basada en microservicios.
4. Conocer y saber cómo modificar el valor algunos parámetros que pueden mejorar las prestaciones.

Contenido de las lecciones

1. Benchmarks suites y Jmeter

1. Introducción

Como ejercicio final tras haber instalado, configurado y monitorizado servicios, vamos a aplicar una carga que nos pueda servir cómo indicador para ver cómo podría responder nuestro sistema. En base a los resultados obtenidos (y monitorizados) consideraremos también algunos parámetros que puedan mejorar las prestaciones ante cargas concretas. En primer lugar estudiaremos un conjunto de benchmarks para luego centrarnos en uno concreto para un servicio que ya conocemos y que trataremos de optimizar su comportamiento.

2. Benchmarks

Hay muchos para cada tipo de servicio (p.ej. Para DNSs: NameBench y GRC's DNS Benchmark, `http://sourceforge.net/directory/os:linux/?q=benchmark`) pero puede resultar más interesante programar uno para analizar algún parámetro concreto que deseemos. Siempre debemos tener en cuenta un mínimo de cuestiones al hacerlo:

1. Objetivo del benchmark.
2. Métricas (unidades, variables, puntuaciones, etc.).
3. Instrucciones para su uso.
4. Ejemplo de uso analizando los resultados.

2.1. Phoronix

Phoronix es una plataforma que permite ejecutar un conjunto de benchmarks bajo la agrupación openbenchmarking.org. La aplicación puede instalarse a través de los gestores de paquetes ya vistos en los guiones anteriores.

> **Enunciado**: una vez que haya indagado sobre los benchmarks disponibles, seleccione como mínimo dos de ellos y proceda a ejecutarlos en Ubuntu y CentOS. Comente las diferencias.

> **Enunciado**: tras ver la parte de la práctica relacionada con contenedores, pruebe a ejecutar un benchmark de los seleccionados arriba y analice las diferencias en los resultados. Puede lanzar un contenedor con la imagen de phoronix, consulte: `https://www.phoronix.com/scan.php?page=article&item=docker-phoronix-pts&num=1`

Otra funcionalidad interesante es la interfaz phoromatic que permite orquestar y automatizar la ejecución de benchmarks en múltiples máquinas.
Podemos hacer uso de una interfaz web mediante la instalación de un navegador, p.ej. firefox, y ejecutando el comando correspondiente indicado en la documentación [9]. La interfaz realiza una monitorización de la ejecución que podemos contrastar simultaneamente con la que se realiza mediante Zabbix 2.1.

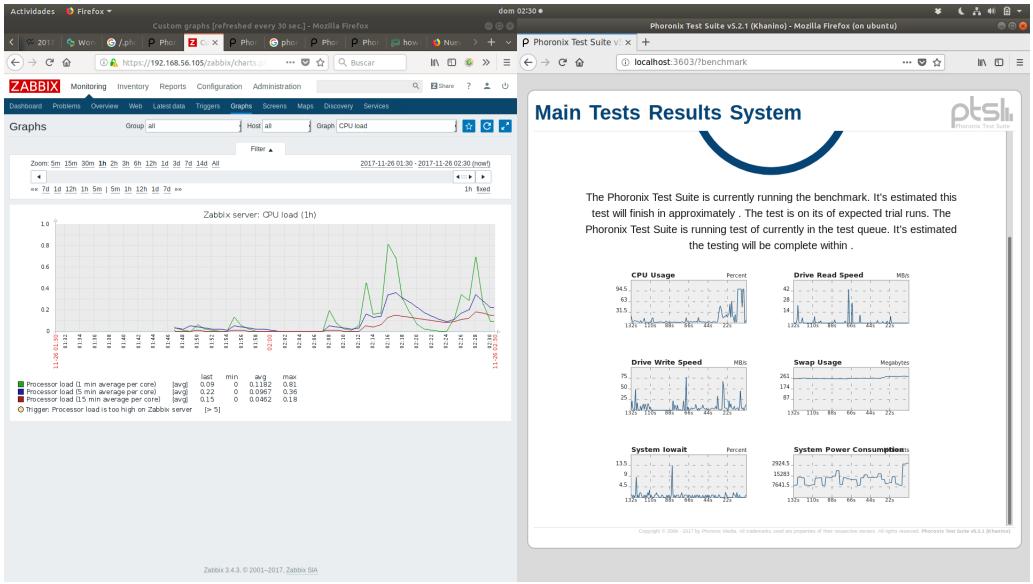

Figura 2.1: Ejemplo de monitorización con Zabbix y phoronix durante la ejecución de un benchmark

2.2. ab y Jmeter

Dentro de los benchmarks más populares para servidores web podemos encontrar la herramienta para "Apache HTTP server benchmarking "(comando ab) [4]. Concretamente, su misión es mostrar cuantas peticiones por segundo el servidor de HTTP (puede ser Apache, Nginx, IIS o cualquier otro) es capaz de servir.
Dentro de los parámetros básicos que usted debe conocer encontramos la opción que especifica la concurrencia así como el número de peticiones. Usted debe probar a ejecutar el benchmark (monitorizar su ejecución en el cliente y en el servidor) con sus máquinas virtuales (Ubuntu y CentOS) y obtener conclusiones respecto al tipo de concurrencia que es capaz de generar ab además de comparar resultados.
Aunque tenemos muchas posibilidades usando ab de manera correcta, hay otras herramientas que nos permiten hacer tests más complejos, concretamente vamos a ver Jmeter `http://jmeter.apache.org/` [6]. Este software se autodefine como una aplicación "*...designed to load test functional behavior and measure performance...*", es decir, no establecen ningún tecnicismo concreto ya que pueden medir y generar carga de forma genérica para varios elementos.
Ha sido comparado por la empresa Octoperf con Gatling (otra herramienta popular para realizar test de estrés [8]) obteniendo la conclusión de que ambos tienen un comportamiento y capacidades similares [12].
Algunas características interesantes de Jmeter es que puede crear concurrencia real debido a la posibilidad de usar varias hebras dentro de la mismas CPU así como distribuir la creación de carga en varias máquinas. La posibilidad de ejecutarse en modo de línea de comandos permite aligerar la carga de la máquina que está generando las peticiones al servidor además de permitir la automatización de ciertos tests. También es muy interesante la funcionalidad que nos permite configurar como proxy de nuestro navegador a Jmeter para que vaya registrando nuestra navegación como usuarios y luego podamos replicar esas peticiones de manera automática y paralela.
Otra herramienta que le puede ser interesante seguir es el proyecto de código abierto Locust `https://locust.io/` que permite escribir en código (con Python) los tests sin necesidad de interfaces permitiendo comprobar la escalabilidad sin límites en lo que se refiere a la creación de hebras.

Enunciado: tras probar un test básico para una web [7], utilizaremos Jmeter para hacer un test sobre una aplicación que ejecuta sobre dos contenedores (uno para la BD y otro para la aplicación en sí). El código está disponible en `https://github.com/davidPalomar-ugr/iseP4JMeter` donde se dan detalles sobre cómo ejecutar la aplicación en una de nuestras máquinas virtuales. El test de Jmeter debe incluir los siguientes elementos:

- El test debe tener parametrizados el Host y el Puerto en el Test Plan (puede hacer referencia usando $param)
- Debe hacer dos grupos de hebras distintos para simular el acceso de los alumnos y los administradores. Las credenciales de alumno y administrador se cogen de los archivos: alumnos.csv y administrador.csv respectivamente.
- Añadimos esperas aleatorias a cada grupo de hebras (Gaussian Random Timer)
- El login de alumno, su consulta de datos (recuperar datos alumno) y login del adminsitrador son peticiones HTTP.
- El muestreo para simular el acceso de los administradores lo debe coger el archivo apiAlumnos.log (usando un Acces Log Sampler)
- Use una expresión regular (Regular Expressión Extractor) para extraer el token JWT que hay que añadir a la cabecera de las peticiones (usando HTTP Header Manager)

2.3. Breve introducción a los contenedores: docker

Ya hicimos referencia a los contenedores en la práctica 1 cuando hablamos del uso de las máquinas virtuales usando la analogía de proceso y hebra. Con ese conocimiento es suficiente para realizar la práctica si bien vamos a ilustrar esa compartición la Figura 2.2 El contenedor presenta a la aplicación una nueva capa abstracción sobre el sistema operativo haciéndola homogénea, facilitando la portabilidad y, sobre todo, el encapsulamiento de información. La principal ventaja respecto al uso de máquinas virtuales es el ahorro en la función de Hypervisor (que es reemplazado por el S.O. de la máquina física) y el ahorro del S.O. invitado (que es reemplazado por la aplicación docker). Como consecuencias directas los contenedores ocupan menos espacio , requieren menos recursos y son más veloces en el arranque.
Aunque hay muchas tecnologías que implementan este nivel de abstracción (LXC, OpenVZ, RKT, Virtuozzo, etc.), Docker es una de las más populares a día de hoy. Hay dos implementaciones, una para la comunidad y otra bajo licencia con funcionalidades extras [10], además puede ver una tabla comparativa en `https://www.docker.com/products/container-runtime`. No obstante, esta popularidad puede verse perjudicada por la reciente compra (13 de Noviembre 2019) por parte de Mirantis (o afectar a la parte abierta del proyecto...).

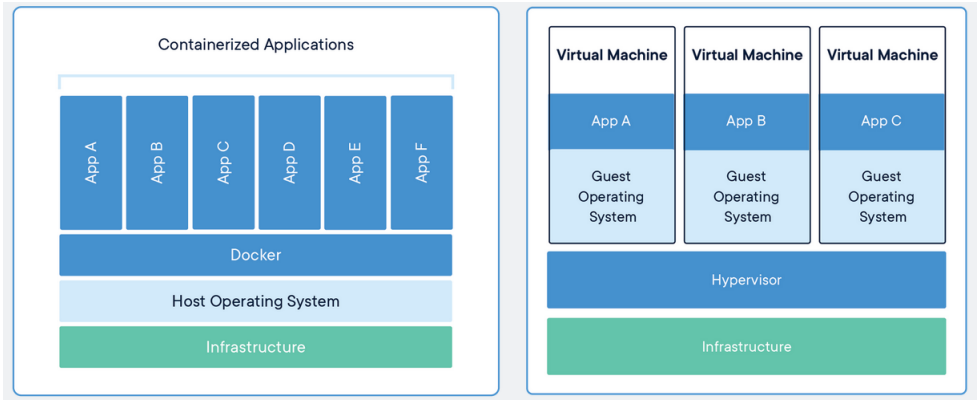

Figura 2.2: Nivel de virtualización de un contenedor. Fuente: `https://www.docker.com/resources/what-container`

Debido a este cambio, es obligado mencionar tecnologías como **podman** `https://podman.io/` que nos permiten ejecutar contenedores con la misma sintaxis que docker y sin necesidad de tener un servicio como demonio [1]. Otra herramienta complementaria a docker-compose como citaremos adelante es **buildah** `buildah.io`.
En `https://training.play-with-docker.com/` tiene cantidad de recursos para aprender los elementos básicos (y más avanzados). Concretamente, en `https://training.play-with-docker.com/ops-s1-hello/` puede ejecutar el "hola mundo"sin necesidad de instalar nada en su máquina anfitriona e ir familiarizándose con esta tecnología. Se le recomienda que realice el tutorial para entender mejor lo que haremos posteriormente.

2.3.1. Microservicios

La popularidad de los contenedores ha ido ligada al éxito de un tipo de arquitectura conocida como microservicios (*microservices*). En este tipo de arquitecturas tenemos distintos servicios que se ejecutan de forma independiente unos de otros y se comunican mediante APIs o sistemas de mensajería (RabbitMQ p.ej.).
Las ventajas de esta arquitectura son varias pero la flexibilidad y escalabilidad (vertical y horizontal) quizá sean las más destacables.

2.3.2. Instalando docker en Ubuntu Server

Añadimos llave GPG para validar el repositorio:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Añadimos repositorio (todo escrito en una línea):

```
sudo add-apt-repository
"deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
```

[1] Hasta el punto que se recomienda hacer un alias docker podman

Actualizamos lista de repositorios:

```
sudo apt update
```

Buscamos el repositorio de docker (community edition) y lo instalamos:

```
apt search docker-ce
sudo apt install docker-ce
```

Comprobar estado del servicio (ver que está iniciado y activado...)
Añadimos el usuario al grupo docker:

```
sudo usermod -aG docker su_nombre_de_usu
```

(Volver a loguearse o llamar a bash)
Podemos probar los comandos:

```
docker info ; docker run hello-world
```

2.3.3. Instalando docker-compose en Ubuntu Server

Ansible orquesta máquinas virtuales, para orquestar contenedores tenemos docker-compose (entre otras herramientas, vease buildah, kubernetes) que nos permite desplegar aplicaciones con varios contenedores de modo que cada uno tenga una función. Los pasos para instalarlo son:
Instalamos con apt (porque ya tenemos el repo configurado...)

```
apt install docker-compose
```

Y probamos...

```
docker-compose
docker-compose --version
```

2.3.4. Instalación de la aplicación para el test con Jmeter

Ahora vamos a desplegar la aplicación en nuestro Ubuntu Server, para ello lo primero que debemos hacer es clonar el repositorio de David Palomar desde nuestra máquina Ubuntu:

```
git clone https://github.com/davidPalomar-ugr/iseP4JMeter.git
```

Tras esto, tendremos un directorio nuevo: iseP4JMeter al cual podremos acceder y levantar la aplicación con docker compose:

```
cd iseP4JMeter
docker-compose up
```

En pantalla tendremos mostradas las peticiones que vayamos haciendo, podemos dejar docker como demonio o background con la opción -d.

2.3.5. Detalles sobre el dockerfile y compose

El *dockerfile* es el archivo donde indicamos todos los comandos necesarios para crear una imagen de docker y las acciones que debe llevar a cabo sobre esta (copiar archivos, instalar paquetes, modificar configuraciones). Desde un punto de vista muy abstracto podríamos decir que es el *playbook* de Ansible que aplica docker al levantar el contenedor.
En este dockerfile, dos elementos muy relevantes son la configuración del contenedor en lo que a red y almacenamiento se refiere.
En la aplicación sobre la que aplicaremos la carga tenemos dos contenedores que se configuran con los respectivos dockerfiles:
Contenido del dockerfile de la máquina que contiene la aplicación (en node.js):

```
FROM node:8
RUN mkdir -p /usr/src/app
COPY . /usr/src/app
EXPOSE 3000
WORKDIR /usr/src/app
RUN ["npm", "update"]
ENV NODE_ENV=production
CMD ["npm","start"]
```

En este archivo, indicamos la imagen a partir de la que crear el contenedor (node) con la etiqueta 8. Tras descargar la imagen (en caso de que no esté descargada ya) ejecutará un comando (con RUN) para crear un directorio y copiar los archivos del repositorio en el contendor. Posteriormente, indicamos el puerto en el que queramos que escuche el contenedor (EXPOSE), esto está relacionado con la opción -p al ejecutar el contenedor ya que podemos hacer un mapeo del puerto del anfitrión ejecutando los contenedores con los que están exponiendo éstos. Por último se indica que la aplicación comience su ejecución tras configurar una variable de entorno.
Contenido del dockerfile de la máquina que contiene la base de datos (mongodb):

```
FROM mongo
COPY ./scripts/* /tmp/
RUN chmod 755 /tmp/initializeMongoDB.sh
WORKDIR /tmp
CMD ./initializeMongoDB.sh
```

En este caso, la imagen de partida es de mongo, una vez disponible, se copian unos scripts del repositorio al contenedor y se ejecutan (tras cambiar los permisos correspondientes). Por último se lanza un script para rellenar la base de datos.
Como ya hemos comentado, la herramienta para configurar varios contenedores de manera que den un único servicio es docker-compose. Cuando la invocamos, ésta lee el archivo *compose* que especifica los siguientes elementos:

```
version: '2.0'
services:
```

```
#MongoDB based in the original Mongo Image
mongodb:
  image: mongo
  ports:
    - "27017:27017"

# Initialize mongodb with data
mongodbinit:
  build: ./mongodb
  links:
    - mongodb

# Nodejs App
nodejs:
  build: ./nodejs
  ports:
    - "3000:3000"
  links:
    - mongodb
```

Como siempre, le remitimos a la documentación oficial para ver los numerosos detalles sobre estos archivos: `https://docs.docker.com/get-started/part2/#dockerfile` `https://docs.docker.com/engine/reference/builder/` `https://docs.docker.com/compose/compose-file` `https://docs.docker.com/develop/develop-images/dockerfile_best-practices/`

2.3.6. La aplicación iseP4JMeter

La aplicación presenta una API Rest para obtener información sobre los alumnos tras autenticarse. Los detalles están descritos en el correspondiente README.md aunque, para más claridad, se ilustra el funcionamiento con el diagrama de la Figura 2.3
Además, se incorpora un script en bash que comprueba el correcto funcionamiento de la aplicación. En la Figura 2.4 está una descripción de la función de cada bloque.
Existe otra forma de comprobar que la aplicación está ejecutándose correctamente: podemos instalar un plugin a nuestro navegador (POSTMAN, RESTED, etc.) y generar las dos peticiones recuperando el token tras la autenticación y luego haciendo la petición tal y como muestra la Figura 2.5.
Respecto al mecanismo de autenticación, la API incorpora un tipo básico del protocolo HTTP que se usa introduciendo credenciales en la cabecera de la petición [1]. El fin de este mecanismo es reducir el ruido de Internet. Además de esta autenticación, la aplicación incorpora otro mecanismo basado en JSON Web Tokens (JWT) [2, 3]. Tiene varias ventajas frente a otros sistemas (claridad de JSON vs xml, tamaño, forma de cifrado) además de resolver el problema de *cross-domain*.
Para ilustrar el funcionamiento de la aplicación se muestra el diagrama de comunicaciones entre el cliente y el servidor en la Figura 2.6.

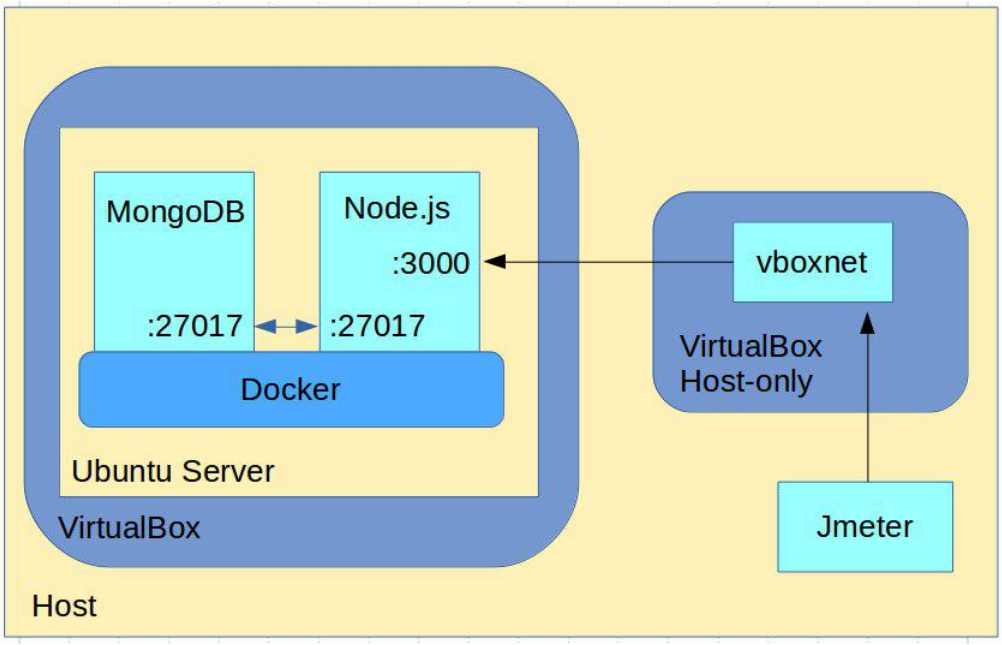

Figura 2.3: Descripción de los niveles de abstracción y los distintos elementos en ejecución.

```
#!/bin/bash
SERVER=localhost
TOKEN=$(curl -s \
-u etsiiApi:laApiDeLaETSIIDaLache \
-d "login=mariweiss@tropoli.com&password=anim" \
-H "Content-Type: application/x-www-form-urlencoded" \
-X POST http://$SERVER:3000/api/v1/auth/login)
resultado=$?
if test "$resultado" != "0"; then
echo "ERROR: Curl fallo con resultado: $resultado"
fi
curl \
-H "Authorization: Bearer $TOKEN" \
http://$SERVER:3000/api/v1/alumnos/alumno/mariweiss%40tropoli.com
```

Definimos la IP (o hostname) del servidor

Hace un POST al servidor para recibir un token jwt
Haciendo un Basic Auth (opción -u) y pasando
credenciales en el cuerpo con -d

Comprueba que ha habido un resultado

Realiza la petición GET incluyendo el token
recibido en la llamada POST anterior y muestra
el resultado

Figura 2.4: Descripción de la funcionalidad de cada bloque del script.

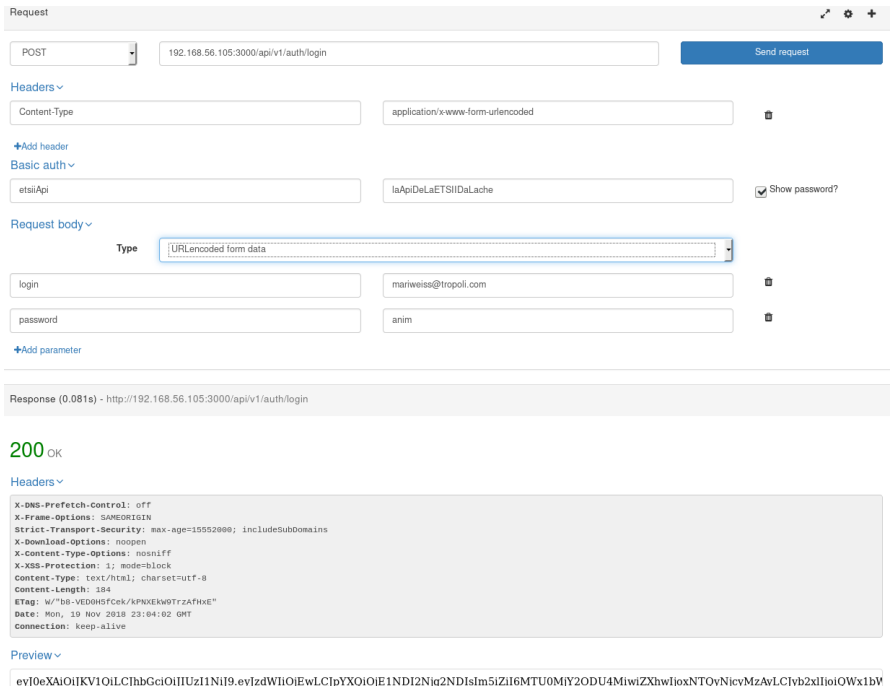

Figura 2.5: Descripción de la funcionalidad de cada bloque del script.

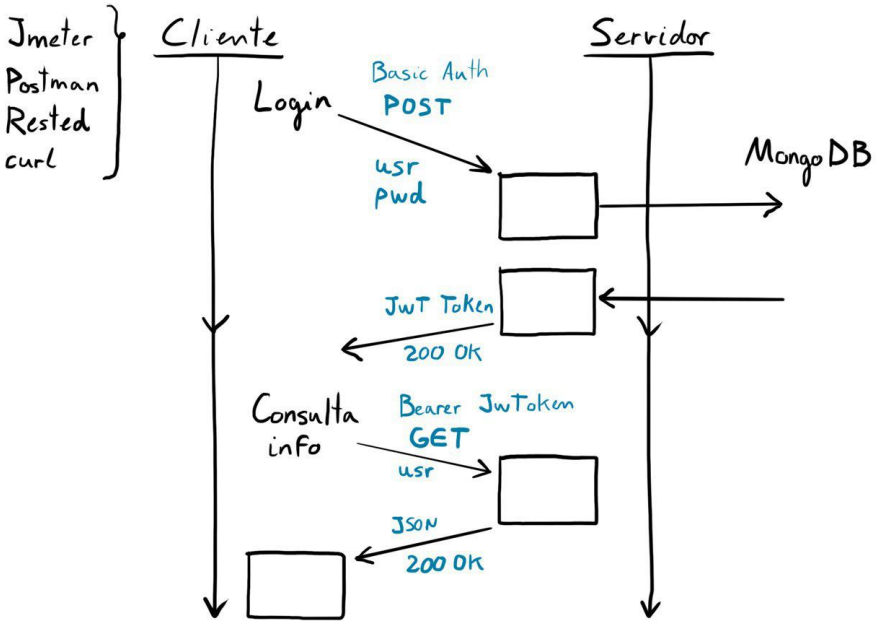

Figura 2.6: Descripción de los pasos de comunicación entre el cliente y el servidor.

3. Ajuste del sistema y de servicios: *Performance engineering*

Independientemente de la infraestructura que tengamos por debajo (HW, contenedores, máquinas virtuales, etc.) debemos ser conscientes de que cada capa tiene sus propios parámetros y ajustes y que, éstos, afectarán al rendimiento de la aplicación.
La modificación de los parámetros del sistema es una tarea compleja pues existen una gran cantidad de éstos para otra gran cantidad de subsistemas que, además, están relacionados. Tan solo en lo referente a los parámetros del kernel (el del anfitrión o el del invitado si usamos MV), tenemos un gran abanico de variables cuyos valores pueden ser números naturales, siendo muy difícil establecer o estimar una aproximación al óptimo (teniendo en cuenta que éste puede variar dependiendo de la carga). Algunos de estos elementos ajustables del kernel pueden ser accedidos y modificados a través del sistema de archivos (/proc/sys). Modificando los valores dentro de los archivos, es posible cambiar el comportamiento del sistema sin tener que reiniciarlo. Dentro de /proc/sys encontramos una estructura de directorios que separa los archivos de los distintos subsistemas (fs: sistema de archivos, kernel: kernel, vm: memoria virtual, dev: dispositivos, etc.)
Para evitar errores o valores inválidos, es recomendable usar el comando sysctl para modificar los parámetros del kernel.

3.1. Optimizando un servidor web

Como ya estamos familiarizados con la pila LAMP, vamos a considerar algunos parámetros y configuraciones interesantes que nos permitan obtener un mayor rendimiento.
No solo hay que tener en cuenta los elementos del sistema operativo y de los servicios en ejecución sino que cada código tiene unas características que nos obligan a monitorizar durante un tiempo la actividad e ir ajustando los valores en base a los resultados. Por ejemplo, si queremos tener una plataforma para enseñanza como puede ser moodle, ellos mismos ya tienen algunas sugerencias que, a posteriori, podremos modificar, pero son un buen punto de partida [11] y que incluyen las recomendaciones de la documentaciónd de Apache [5]. Otro artículo en la misma línea es el incluido en la documentación de Wordpress, un Content Management System (CMS) de los más populares [13].

> **Trabajo opcional**: con esta información usted podría modificar los parámetros de configuración de Apache, PHP o MariaDB para observar un cambio en el comportamiento del servidor (CentOS o Ubuntu server) mediante la aplicación de un benchmark y analizando el cambio en las prestaciones o mediante el análisis de datos de monitorización ante una carga aplicada.

Referencias

[1] . `https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication`. [Online; consultada 14-Nov-2019].

[2] . https://jwt.io/introduction. [Online; consultada 14-Nov-2019].

[3] . https://auth0.com/learn/json-web-tokens/. [Online; consultada 14-Nov-2019].

[4] The Apache Software Foundation. ab - apache http server benchmarking tool. https://httpd.apache.org/docs/2.4/programs/ab.html, 2017. [Online; consultada 14-September-2017].

[5] The Apache Software Foundation. Apache performance tuning. http://httpd.apache.org/docs/current/misc/perf-tuning.html, 2017. [Online; consultada 14-September-2017].

[6] The Apache Software Foundation. Best practices. http://jmeter.apache.org/usermanual/best-practices.html, 2017. [Online; consultada 14-September-2017].

[7] The Apache Software Foundation. Building a web test plan. http://jmeter.apache.org/usermanual/build-web-test-plan.html, 2017. [Online; consultada 14-September-2017].

[8] Gatling. Gatling. http://gatling-tool.org/, 2017. [Online; consultada 14-September-2017].

[9] Phoronix Media. Phoronix test suite. https://www.phoronix-test-suite.com/documentation/phoronix-test-suite.html, 2017. [Online; consultada 14-September-2017].

[10] Michael Friis (Docker Inc.). Announcing docker enterprise edition. https://www.docker.com/blog/docker-enterprise-edition/, 2017. [Online; consultada 14-Nov-2019].

[11] Moodle. Performance recommendations. https://docs.moodle.org/33/en/Performance_recommendations, 2017. [Online; consultada 14-September-2017].

[12] Jérôme Loisel CTO Octoperf. Jmeter vs gatling tool. https://octoperf.com/blog/2015/06/08/jmeter-vs-gatling/#gist23106684, 2015. [Online; consultada 14-September-2017].

[13] WordPress. Wordpress optimization. https://codex.wordpress.org/WordPress_Optimization, 2017. [Online; consultada 14-September-2017].