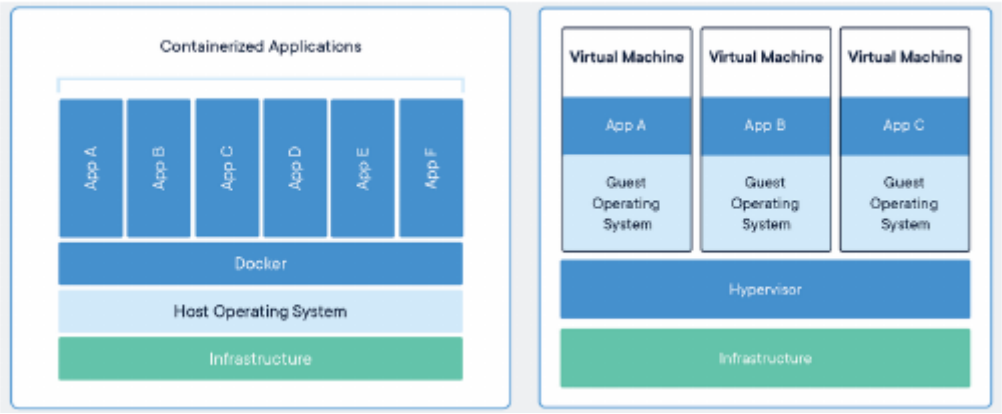
ISE 22/23 - Práctica 4: Benchmarking

1. Notas del guión de prácticas

- **Benchmark para DNS:**
  - NameBench
  - GRC's DNS Benchmark
- **Benchmark:**
  - Una aplicacion para generar una carga en un componente o el sistema completo para obtener una métrica de rendimiento.
  - Parámetros comunes de benchmarking:
    - HB: higher better
    - LB: Lower better
- **Phoronix:**
  - Phoronix es una plataforma que permite ejecutar un conjunto de benchmarks bajo la agrupación openbenchmarking.org.
  - phoromatic permite orquestar y automatizar la ejecución de benchmarks en múltiples máquinas.
  - Podemos usar la interfaz web levantando un servidor con las siguientes instrucciones (tomadas de la documentación), y accediendo a localhost:3603/?benchmark
    - A Phoromatic server can be started using `phoronix-test-suite start-phoromatic-server` (or the included `systemd phoromatic-server service` file).
    - Clients can connect to the server using the `phoronix-test-suite phoromatic.connect` command as well as a `phoromatic-client systemd service`.
    - See the Phoromatic section of the documentation for more information on setting up Phoromatic.
      - `start-phoromatic-server`: start the Phoromatic web server for controlling local Phoronix Test Suite client systems to facilitate automated and repeated test orchestration and other automated features targeted at the enterprise. Result Viewer
      - `start-result-viewer`: start the web-based result viewer.
- **ab:**
  - Herramienta para “Apache HTTP server benchmarking ”, ab. Su misión es mostrar cuantas peticiones por segundo el servidor de HTTP (no solo a apache, puede ser otro) es capaz de servir.
  - Opción que especifica la concurrencia así como el número de peticiones:
    - `ab -c` : Number of multiple requests to perform at a time. Default is one request at a time.
    - `ab -n`: Number of requests to perform for the benchmarking session. The default is to just perform a single request which usually leads to non-representative benchmarking results.

- **Jmeter:**
  - ab nos da muchas posibilidades si lo sabemos usar, pero herramientas como jmeter proveen tests más complejos.
  - Se autodefine como *"...designed to load test functional behavior and measure performance..."*, no establecen ningún tecnicismo concreto ya que pueden medir y generar carga de forma genérica para varios elementos.
  - 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.
  - Puede ejecutarse en modo de línea de comandos para aligerar la carga de la máquina que está generando las peticiones al servidor además de permitir la automatización de ciertos tests.
  - 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.
- **Locust (código abierto):**
  - 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.
- **Breve introducción a contenedores: docker**



  - 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 a MV es el ahorro en la función de Hypervisor (reemplazada por el S.O. de la máquina física) y el ahorro del S.O. invitado (reemplazado por la aplicación docker).
    - Como consecuencia, los contenedores ocupan menos espacio , requieren menos recursos y son más veloces en el arranque.
  - Tecnologías: LXC, Open-VZ, RKT, Virtuozzo, **Docker**
    - Pero Docker recientemente ha sido comprado por Mirantis, lo cual puede afectar a la versión comunidad. Así que mencionamos también **podman**, que tiene la misma sintaxis y corre sin un servicio demonio, con los riesgos de seguridad que eso conlleva. Se llega hasta a recomendar hacer un `alias docker=podman`
    - También mencionamos **buildah** como herramienta complementaria a docker-compose
- **Microservicios:**
  - La popularidad de los contenedores va ligada a la arquitectura de microservicios (microservices), donde 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.
- **Instalación en Ubuntu Server:**

  DOCKER:

  1. Añadimos llave GPG para validar el repositorio:

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

  2. Añadimos repositorio (todo escrito en una línea):

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

  3. Actualizamos lista de repositorios:

     ```
     sudo apt update
     ```

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

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

  5. Comprobar estado del servicio (ver que está iniciado y activado...) Añadimos el usuario al grupo docker 2
     - Aunque es el procedimiento descrito en la guía oficial, tiene ciertos problemas de seguridad

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

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

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

  DOCKER-COMPOSE: Ansible orquesta máquinas virtuales, para orquestar contenedores tenemos docker-compose (entre otras herramientas, vease buildah, kubernetes). Nos permite desplegar aplicaciones con varios contenedores de modo que cada uno tenga una función. Los pasos para instalarlo son:

  1. Instalamos (ya está el repo configurado)

     ```
     apt install docker-compose
     ```

  2. Probamos

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

JMETER:

3. Instalar repo:

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

4. Levantar 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.

- **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         //Indica imagen a partir de la q crear el contenedor
(node)
RUN mkdir -p /usr/src/app   //Descargarla, crear dir, meter archivos a
dir
COPY . /usr/src/app
EXPOSE 3000                 //Puerto en el q escucha el contenedor
WORKDIR /usr/src/app
RUN ["npm", "update"]
ENV NODE_ENV=production     //Conf var de entorno y run
CMD ["npm","start"]
```

    Contenido del dockerfile de la máquina que contiene la base de datos (mongodb):

```
FROM mongo              //Descargar y sacar scripts (con cambio en
permisos)
COPY ./scripts/* /tmp/
RUN chmod 755 /tmp/initializeMongoDB.sh
WORKDIR /tmp
CMD ./initializeMongoDB.sh  //Rellena la BD
```

  - La herramienta para configurar varios contenedores simultáneamente es docker-compose, así que vamos a ver lo mismo que arriba, pero con un compose:

```
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
```

- **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

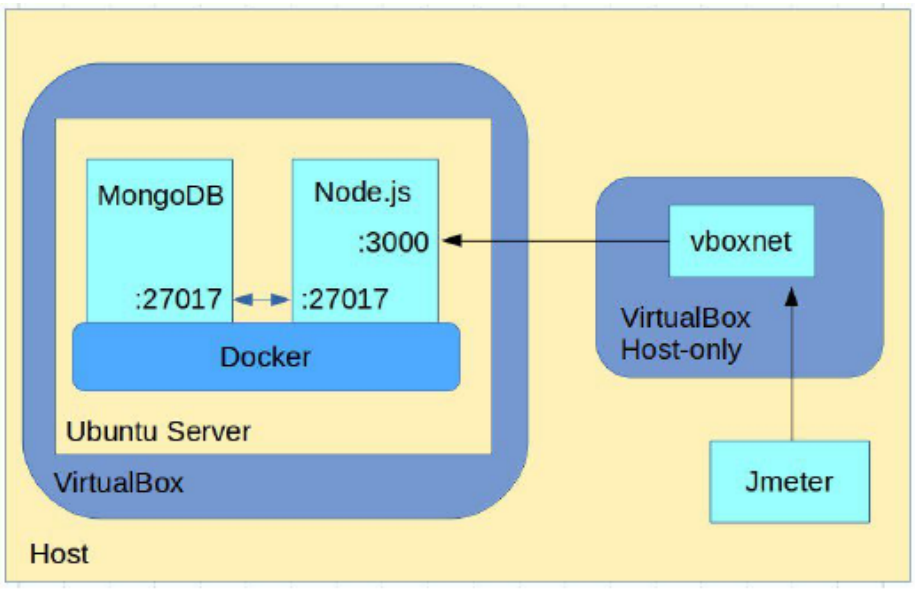


Figura 2.3: Descripción de los niveles de abstracción y los distintos elementos en ejecución.

    - Se incorpora un script en bash que comprueba el correcto funcionamiento de la aplicación. Fig 2.4.

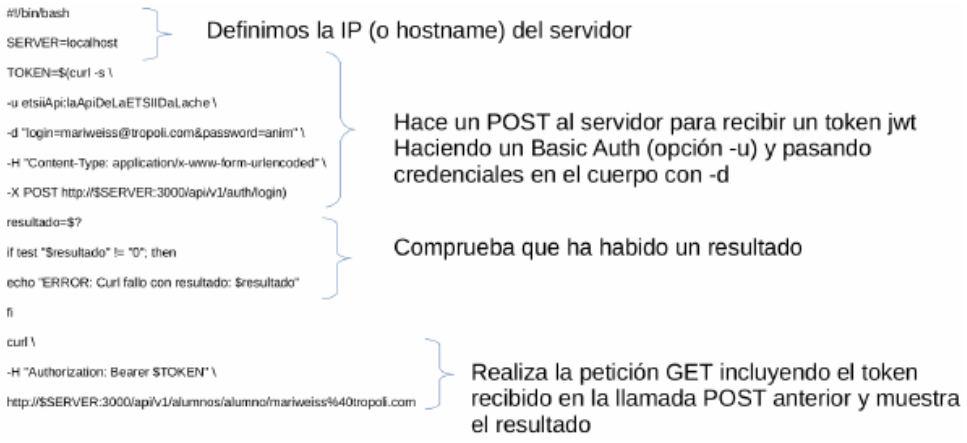


Figura 2.4: Descripción de la funcionalidad de cada bloque del script.

- Otra forma de comprobar que la aplicación está ejecutándose correctamente: 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.

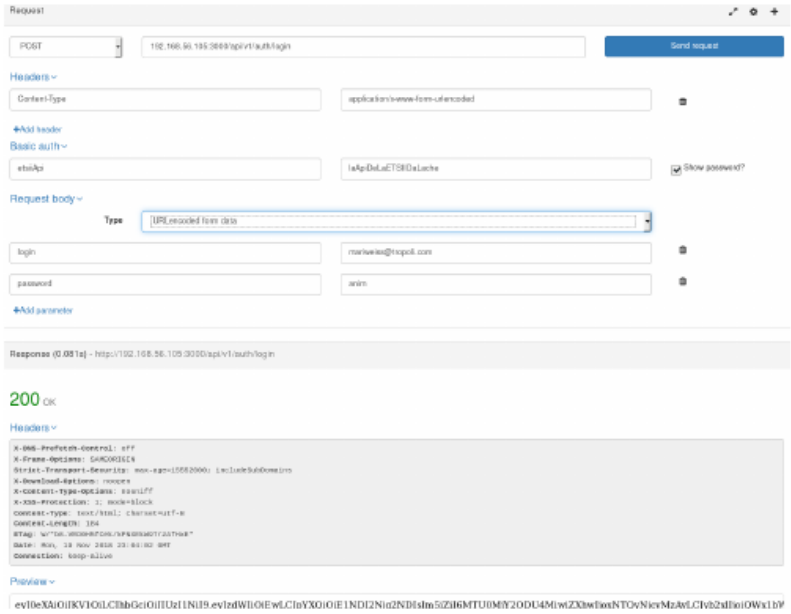


Figura 2.5: Ejemplo de petición HTTP usando el cliente Rested `https://addons.mozilla.org/en-US/firefox/addon/rested/`.

- 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. 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). Tiene varias ventajas frente a otros sistemas (claridad de JSON vs xml, tamaño, forma de cifrado) además de resolver el problema de cross-domain.

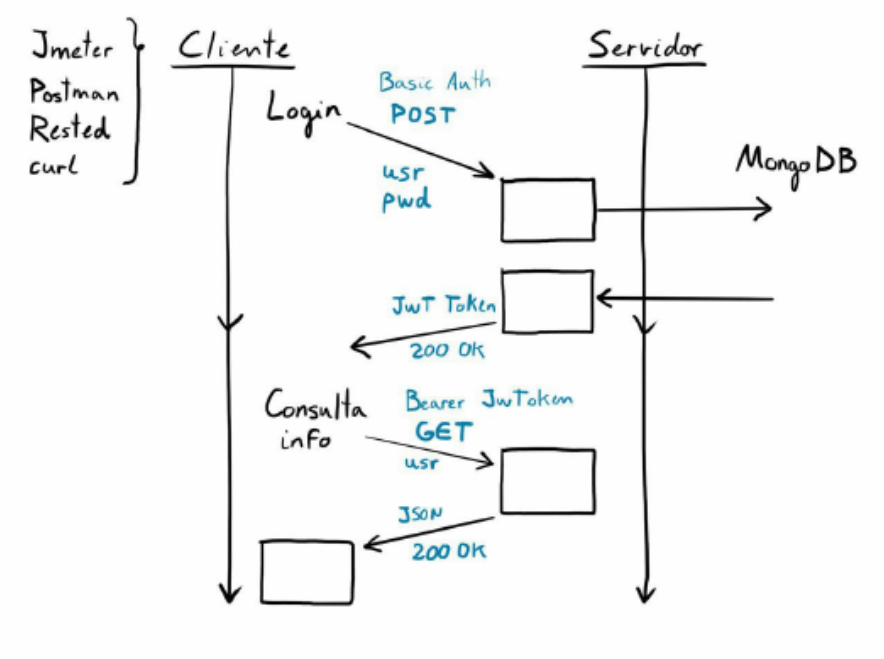


Figura 2.6: Descripción de los pasos de comunicación entre el cliente y el servidor.

- **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.
  - **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 y que incluyen las recomendaciones de la documentación de Apache.
    - Otro artículo en la misma línea es el incluido en la documentación de Wordpress, un Content Management System (CMS) de los más populares y, además, en la documentación de MariaDB tiene varios ejemplos al respecto.

2. Ejercicio 1: Phoronix

Instalar phoronix y dependencias:

```
#Ubuntu
sudo apt install wget php gcc

#Rocky
sudo dnf install wget tar php gcc
```

Ubuntu server

Instalar suite phoronix

```
git clone https://github.com/phoronix-test-suite/phoronix-test-suite.git
cd phoronix-test-suite
sudo ./install-sh
```

Listar benchmarks, hay de muchos tipos. Nos dirá qué testea cada uno (sistema, procesador, gráficos...).

```
phoronix-test-suite list-available-tests
```

Ejecutamos primer benchmark, test de prueba de cpu:

> Voy a poner aquí el q usa el de wuolah (smallpt) pero lo importante es elegir otro distinto, que si no canta. Al fin y al cabo la cosa es mirar luego las opciones de cada benchmark y seguir las instrucciones.

```
phoronix-test-suite benchmark smallpt #Instalará el benchmark de forma temporal y lo ejecutará, puede tardar
```

Para instalar y correr el benchmark (sin que sea temporal).

> Again: leer las opciones del benchmark para saber como ejecutarlo

```
phoronix-test-suite install smallpt #Instalará el benchmark localmente
cd ~/.phoronix-test-suite/installed-tests/pts #Aquí se instalan los benchmarks

#Según el creador, ejecutamos el programa con estas opciones
cd smallpt-1.2.1/
g++ -O3 -fopenmp smallpt.cpp -o smallpt #Compila smallpt, hay warnings por
narrowing, ignorar
time sudo ./smallpt 128 #128 es el numero de ejecuciones/cálculos por pixel
```

> No voy a poner el otro benchmark q usa el de wuolah pq al final la ejecución depende del benchmark específico que elijamos. Con el anterior nos hacemos una idea.

Rocky

Instalar phoronix

```
git clone https://github.com/phoronix-test-suite/phoronix-test-suite.git
cd phoronix-test-suite
sudo ./install-sh
```

El resto igual que en ubuntu: listar, ejecutar temporalmente o de forma manual.

Creo q esto es suficiente para el examen, paso de ejercicios opcionales.

3. Ejercicio 2: Jmeter

> Esto es básicamente un segundo resumen del guión, pero me parece más útil este que el mío la vd

- Test básico ab: Queremos saber el comportamiento de nuestra página cuando hay 100 peticiones con 10 usuarios que se conecten al mismo tiempo. Para realizar esta prueba, escribimos en la terminal:

```
ab -c 10 -n 100 https://www.ubunlog.com/ #Sustituir ubuntu por la página a
testear
```

-c: indica el número de usuarios virtuales

-n: indica el número de peticiones de cada usuario

- Microservicios: Arquitectura donde los distintos servicios se ejecutan de forma independiente unos de otros y se comunican mediante APIs o sistemas de mensajería. Su popularidad ha ido ligada a los dockers.
    - Ejemplos: redis o rabbitMQ
    - Ventajas principales: flexibilidad y escalabilidad (vertical y horizontal).
    - Uso principal de los microservicios: Esquema Servidor de Producción y de Desarrollo.
- Contenedores (docker): Ventajas de una vm y un contenedor: encapsular informacion
    - vm: muchas capas y espacio en disco.
    - Contenedor: Comparte bibliotecas del SO anfitrion y la informacion sigue encapsulada. Los contenedores ocupan menos espacio , requieren menos recursos y son más veloces en el arranque.

Instalación de docker

> Mirar INSTALACIÓN en apartado de notas del guión de la práctica. Aquí voy a poner notas q ha puesto el d wuolah que están bien, como opciones de los comandos y tal.

- **DOCKER:**
  - Opciones docker:
    - docker images : lista las imagenes cargadas en el sistema
    - docker rmi (--force) imagen : borra la imagen del sistema
  - Opciones docker run:
    - -d/--detached: ejecuta la imagen "in the background"
    - -p/--port: especifica el puerto donde acceder a la aplicacion del docker
    - -i/--interactive: permite interactuar con el docker. Se ejecuta junta a -t/--tty
    - -t/--tty: especifica terminal (bash). Se ejecuta junto --i -
    - it [image] bash: ejecuta el docker en modo interactivo a través del terminal bash.
- **DOCKERFILE**
  - Puedes entrar a las imágenes en modo bash para ver como se ejecutan los comandos y el estado en el que quedan:

```
cd iseP4Jmeter/monogodb
docker build -t isep4jmeter_mongodb . #construye la imagen existente con el dockerfile en .
docker run -it isep4jmeter_mongodb bash #ejecutamos la imagen y entramos en bash

#Nos mete en el directorio /tmp que es el ultimo WORKDIR
```

```
cd iseP4Jmeter/nodejs
docker build -t isep4jmeter_mongodb . #construye la imagen existente con el dockerfile en .
docker run -it isep4jmeter_nodejs bash #ejecutamos la imagen y entramos en bash

#Nos mete en el directorio /usr/share/app que es el ultimo WORKDIR
```

  - Opcion de build:
    - -t/--target: especifica el directorio que contiene el dockerfile y los archivos de la app
    - docker build prune : borra las imágenes cargadas en caché.
    - No confundir con la opción de run:
      - -t/--tty: especifica la terminal (bash)
- **DOCKER-COMPOSE:** son ficheros YAML
  - Al instalar en esta MV seguramente nos falle el espacio - extender el disco (crear un disco de 10G y meterlo):

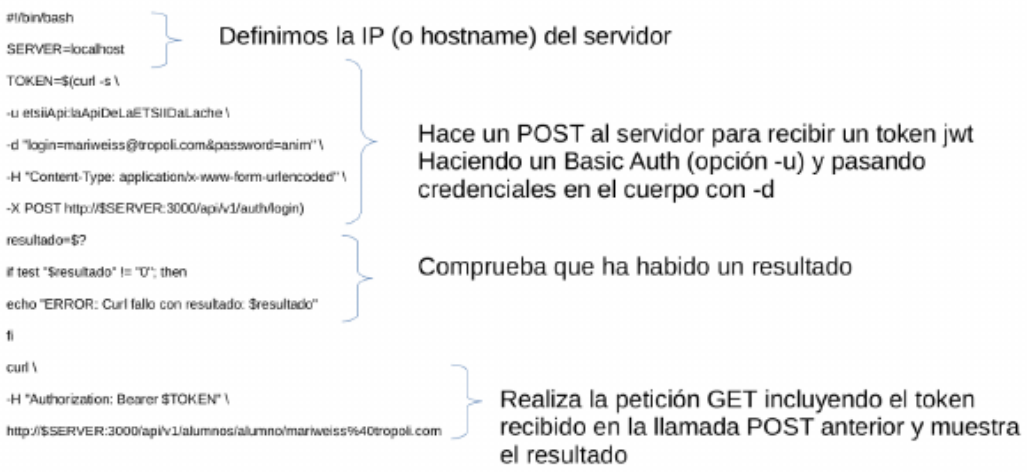
```
lsblk -f    #Para ver cuál ampliar. Buscamos el q contenga root (/)
#Importante quedarnos con el formato: xfs o ext4

sudo vgdisplay
sudo vgextend vg0 /dev/sdc #Ampliar vg

sudo lvdisplay; sudo lvextend -l +100%FREE /dev/vg0/raiz #Ampliar lv

#Recuerda si tu fs es xfs o ext4 y ejecuta un comando:
    sudo xfs_growfs /dev/vg0/raiz #Ampliar sistema de archivos si es xfs
    sudo resize2fs /dev/vg0/raiz #Ampliar sistema de archivos si es ext4
```

- **iseP4JMeter:**
  - Resumen del readme:
    - La aplicación ES una API Rest (2 sistemas que se comunican para obtener información) para obtener información sobre alumnos tras una autenticación previa.
    - Se inicia con docker-compose up y al iniciar podemos ver su descripción en el navegador con la url http://192.168.56.105:3000/:
    - La aplicación tiene 2 servicios:
      - /auth/login: Permite identificar al usuario como alumno o administrador. El acceso está protegido por HTTP BasicAuth(un usuario y contraseña). Cuando te logea, te devuelve un JWT (el token) que se usa para el siguiente servicio.
      - /alumnos/alumno: Devuelve el registro de calificaciones del alumno. Si es un administrador, puede consultar la nota de cualquier alumno. Se debe proporcionar un JWT(el del servicio login) para usarlo.
    - El proceso de consulta es:
      - Identificarse en el servicio de login proporcionando credenciales de válidas de alumno o administrador. Obteniendo un token.
      - Solicitar los datos del propio alumno identificado (alumno) o de un grupo de alumnos (administrador). Podemos hacer uso de los servicios con el script de prueba: ./pruebaEntorno.sh
  - Resumen de esta figura de la q he pasado muy fuerte cuando he hecho la parte del guión:



        - Comando curl : es una herramienta para transferir datos desde o para un servidor (HTTP/s).
            - Opciones:
                - -s/--silent: Silencia el output.
                - -u/--user: especifica el nombre de usuario y contraseña.
                - -d/--data: Envia el dato especificado (POST) al servidor HTTP (credenciales)
                - -H/--header: Cabecera para obtener una página web
                - -X/--request: especifica un metodo de solicitud (GET) al comunicarse al servidor HTTP
        - El $ token es una cadena de identificación que nos da el servicio de login.
    - //TODO PAG 17 a 32 PDF: TEST COMPLETO JMETER