

## OBJETIVOS MÍNIMOS

1. **Contenedores.** Funcionamiento y relación con Virtualización.
2. **Docker.** Ejecución de contenedores y orquestación con compose.
3. **Phoronix Test Suite:** descarga, ejecución y gestión de resultados.
4. **Simulación de cargas de trabajo** con entornos Web empleando AB y JMeter.
5. **Monitorización de prestaciones** empleando los recursos del OS y herramientas básicas como top.
6. **Sistema de ficheros /proc.** Propósito y principales contenidos relacionados con la monitorización de Linux.
7. **Generación y consulta de los ficheros de logs del sistema.**
8. **Papel de logrotate en la gestión de logs.** Opciones básicas de configuración y ejecución.
9. **Programación de tareas periódicas de administración con cron.**
10. **Monitorización a escala con Prometheus + Grafana.**

# 1. Contenedores

## a. ¿Qué es un contenedor?

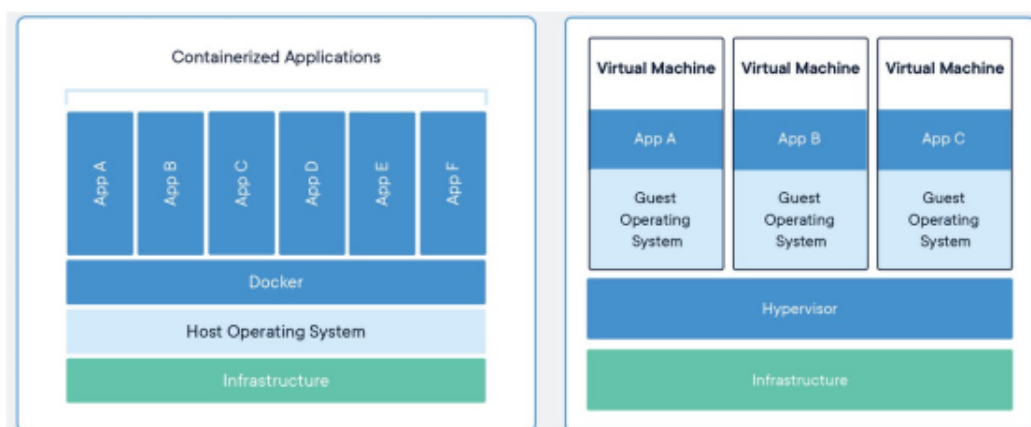
Los contenedores son paquetes software que incluyen todos los elementos necesarios para ejecutar servicios en cualquier entorno. Virtualizan el sistema operativo, son unidades ejecutables de software en las que el código de aplicación se empaqueta junto con sus bibliotecas y dependencias.

Un contenedor en marcha utiliza un sistema de archivos aislado. Este sistema de archivos aislado es proporcionado por una imagen, y la imagen debe contener todo lo necesario para ejecutar una aplicación - todas dependencias, configuraciones, scripts, binarios, etc. La imagen también contiene otras configuraciones para el contenedor, como variables de entorno, un comando predeterminado para ejecutar y otros metadatos.

## b. Contenedores vs máquinas virtuales

En una virtualización tradicional se utiliza un hipervisor para virtualizar el hardware físico, de ahí cada máquina virtual contiene un SO invitado y una copia virtual del hardware que el sistema operativo necesita para ejecutarse, junto con una aplicación y sus bibliotecas y dependencias. Consumen una gran cantidad de recursos y generan mucha sobrecarga.

De otra forma, el contenedor, en vez de virtualizar el hardware subyacente, los contenedores virtualizan el sistema operativo para que cada contenedor individual contenga solo la aplicación junto con sus bibliotecas y dependencias. Por la ausencia del SO invitado, los contenedores son tan ligeros y rápidos. Tan sólo puedes ejecutar contenedores del mismo sistema operativo, es decir, se usa el kernel del SO del host. Su ejecución está aislada del resto de procesos de la máquina host. Además, son más veloces en el arranque.



### c. Beneficios de un contenedor

- Menos sobrecarga: requieren menos recursos de sistema (no incluyen imágenes del SO)
- Mayor portabilidad: las aplicaciones que se ejecutan sobre los contenedores se pueden ejecutar en SO y plataformas hardware diferentes.
- funcionamiento más constante: las aplicaciones en contenedores van a ejecutarse igual independientemente de donde.
- Mayor eficiencia
- Mejor desarrollo de aplicaciones

## 2. Docker

### a. Ejecución de contenedores

Docker run, a no ser que usemos docker compose, en cuyo caso se usa `sudo docker compose up`.

### b. Orquestación con compose

Docker compose es una herramienta que nos deja definir y lanzar varios contenedores en una misma máquina, les asigna una propia red a cada uno. Usa archivos `.yaml`, en este se explica cómo queremos que se desplieguen los contenedores.

#### Ejemplo de docker-compose.yaml:

```
version: "3.9" (versión del formato del archivo docker compose, siempre debe ser la misma)
services: (tenemos 2 servicios definidos, web y redis)
web:
  build: . (lugar donde se encuentra el dockerfile)
  ports:
    - "5000:5000" (redirección del puerto 5000 del host al 5000 de contenedor)
redis:
  image: "redis:alpine"
```

Con el comando `docker compose up` leemos el archivo `docker-compose.yaml`, luego tomamos la imagen definida, la compilamos y ejecutamos. Pero, ¿qué ejecutamos con ese comando? `docker compose up` por así decirlo, es el encargado de lanzar los `dockerfiles` de los contenedores que se encuentran en el `.yaml` y gestionar las dependencias entre ellos (cuál ejecutar o parar antes que otros), casi segura que es así but idk...

`sudo docker compose ps` lista, y `sudo docker compose stop` finaliza la ejecución del contenedor, `docker compose down`, para la ejecución y borra el proceso (no aparece en el `ps` al listar).

### c. Parámetros interesantes del docker compose

- build: dónde está el dockerfile
- image: especifica la imagen que queremos que se use
- ports: mapeado de los puertos “puerto\_servidor:puerto\_contenedor”
- links: linkeamos los contenedores con otros servicios especificando su nombre.
- depends\_on: dependencias de los contenedores
- environment: para añadir variables de entorno

### d. Docker engine o docker daemon

Es necesario para poder ejecutar docker. Es una tecnología de contenedorización de código abierto para la construcción de las aplicaciones. Es el cliente quién utiliza la API de docker para interactuar o controlar el docker daemon a través de comandos o scripts. Es el daemon quién crea y gestiona los objetos Docker como imágenes, contenedores,...

- Dockerfile: en este documento se indican los comandos necesarios para crear imagen de docker y acciones que debe llevar a cabo sobre esta. Este tiene dos elementos muy importantes: configuración del contenedor respecto a red y almacenamiento.

FROM node:16.13.0-stretch	(imagen a partir de donde se crea el contenedor)
RUN mkdir -p /usr/src/app	(ejecutar con run, crea un directorio)
COPY . /usr/src/app	(copia archivos del repositorio en el contenedor)
EXPOSE 3000	(puerto por donde escucha el contenedor)
WORKDIR /usr/src/app	
RUN ["npm", "install"]	
ENV NODE_ENV=production	
CMD ["npm", "start"]	

- Docker Compose: herramienta para configurar varios contenedores de manera que den un único servicio. Cuando la invocamos lee el archivo docker-compose.yml.

#### 1.2.1. Ejercicio valuable. Ejecutar “Hello World”

- **Añadiendo el usuario que se desea al grupo docker:**  
*sudo usermod -aG docker nombre\_de\_usuario*  
*groups nombre\_usuario*

Para correr la imagen de hello world, se debe usar el comando “docker run hello-world”. Sabemos que la imagen está pues se encuentra en el registro de docker en docker hub. Para bajarla se debería hacer “docker pull hello-world”; sin embargo, el pull está incluido en el comando de docker run.

### 3. Benchmarks

#### a. ¿Qué es un benchmark?

Un benchmark, en el contexto de la informática y la tecnología, es una medida estándar que se utiliza para evaluar y comparar el rendimiento de hardware, software o sistemas completos. El propósito principal de un benchmark es proporcionar una forma objetiva de comparar diferentes configuraciones, componentes o productos para determinar cuál es más eficiente, rápido o adecuado para una determinada tarea o aplicación.

Al programar un benchmark se deben tener en cuenta varias cuestiones:

- objetivo
- métricas: unidades, variables, puntuaciones
- instrucciones para su uso
- ejemplo de uso analizando los resultados

#### b. ¿Y benchmarking?

Benchmarking es aplicar una carga mientras monitorizamos un sistema, para comprobar cómo podría responder nuestro sistema a cargas similares.

#### c. OpenBenchmarking

OpenBenchmarking es un repositorio de benchmarks creados bajo licencia opensource. Phoronix es la plataforma de OpenBenchmarking para ejecutar un conjunto de benchmarks.

#### d. Phoronix Test Suite

##### i. Descarga

La podemos instalar usando contenedores en Docker con: `docker pull phoronix/pts`

##### ii. Ejecución

Para ejecutar poder trabajar con phoronix en docker usamos el comando: `docker run -it phoronix/pts`

¿Qué hace -it? Es una combinación de dos flags:

- -i ó --interactive : permite actuar con el contenedor a través de la entrada estándar, stdin.
- -t ó --tty: asigna pseudo-terminal dentro del contenedor.

Para salir tan solo hacemos Ctrl+C

##### iii. Dentro de phoronix-test-suite

- para listar los benchmarks disponibles: `phoronix-test-suite list-available-tests`

- para seleccionar y descargar un benchmark: `phoronix-test-suite install nombre-del-benchmark`
- para ejecutar un benchmark: `phoronix-test-suite run nombre-del-benchmark`

#### iv. Obtención de resultados

- Para almacenar los resultados de la ejecución previamente:  
*phoronix-test-suite run nombre-del-benchmark -r /ruta/hacia/archivo-de-resultados*
- Para convertir un archivo de resultados (.xml) a texto  
*phoronix-test-suite result-file-to-text nombre-del-resultado*  
("nombre-del-resultado" con el nombre del resultado que aparece en la URL proporcionada después de la ejecución del benchmark)
- Moviendo la carpeta de resultados a /var/www/html y accediendo a ella por el navegador del host. (.phoronix-test-suite/test-results)

### **2.1.1. Ejercicio valuable. Ejecutar un benchmark**

Yo lo he hecho con pts/git:

1. phoronix-test-suite install pts/git
2. phoronix-test-suite run pts/git
3. Especificamos nombre: resultadosGit1

```

isabel@isabel-VivoBook-ASUSLaptop-X421EA-5433E: ~
+ spec_store_bypass: Mitigation of SSB disabled via pcrl
+ spectre_v1: Mitigation of usercopy/swappgs barriers and __user pointer san
itization
+ spectre_v2: Mitigation of Enhanced / Automatic IBRS IPB: conditional SSB
filling PRBSB-eIBRS: SW sequence
+ srbds: Not affected
+ tsx_async_abort: Not affected

Would you like to save these Test results (Y/n): Y
Enter a name for the result file: resultadoscotti
Enter a unique name to describe this test run / configuration:

If desired, enter a new description below to better describe this result set / system configuration u
nder test.
Press ENTER to proceed without changes.

Current Description: Docker testing on Ubuntu 20.04.4 LTS via the Phoronix Test Suite.

New Description:

Git:
pts/git-1.1.0
Tested 1 of 1
Estimated Trial Run Count: 3
Estimated Time to Completion: 2 Minutes [07:45 UTC]
  Running Pre-Test Script @ 07:44:03
    Started Run 1 @ 07:44:03
      Running Interim Test Script @ 07:44:54
        Started Run 2 @ 07:44:56
          Running Interim Test Script @ 07:45:48
            Started Run 3 @ 07:45:50
              Running Post-Test Script @ 07:46:42

Time to Complete Common Git Commands:
49.004
49.521
49.77

Average: 49.432 Seconds
Deviation: 0.79%

Comparison of 2,071 OpenBenchmarking.org samples since 11 April 2020; median result: 55.44 Second
s. Box plot of samples:
[-----]----- This Result (62nd Percentile): 49.432 ^
                                     Intel Core i9-14900K: 33.38 ^
                                     Intel Core i5-14500: 37.66 ^
                                     Intel Core i9-13900K: 42.83 ^
                                     AMD EPYC 7712: 55.19 ^

Do you want to view the text results of the testing (Y/n):

```

Al terminar la ejecución me dieron este enlace para ver los resultados:

<https://openbenchmarking.org/result/2405076-NE-PRUEBA1BE17> desde ahí podemos descargarlo como pdf

## **2.1.2. Ejercicio Opcional. Comparar benchmark en contenedor y en vm SIN HACER**

### **e. Apache Benchmark (ab)**

Su principal objetivo es mostrar cuántas peticiones por segundo el servidor de HTTP (cualquiera, Apache, Nginx,...) es capaz de servir.

#### **i. Parámetros básicos:**

- concurrencia (-c) : número de peticiones por unidad de tiempo. Por defecto es una.
- número de peticiones (-n): número de peticiones a realizar en la prueba del benchmark. Por defecto es una, lo que no suele dar resultados representativos.
- tiempo límite (-t): máximo número de segundos para ejecutar el benchmark.

<https://www.datadoghq.com/blog/apachebench/>

#### **ii. Ejecución de la prueba de carga**

##### **- Instalación de apacheBench:**

apt-get update

apt-get install -y apache2-utils

##### **- Ejecución**

ab <OPTIONS> <WEB\_SERVER\_ADDRESS>/<PATH>

Por ejemplo, ab -n 1000 -c 10 http://localhost/

#### **iii. Resultados obtenidos**

<https://www.datadoghq.com/blog/apachebench/>

Server Software:	AmazonS3
Server Hostname:	<SOME_HOST>
Server Port:	443
SSL/TLS Protocol:	TLSv1.2,ECDHE-RSA-AES128-GCM-SHA256,2048,128
TLS Server Name:	<SOME_HOST>
Document Path:	/
Document Length:	45563 bytes
Concurrency Level:	2
Time taken for tests:	3.955 seconds
Complete requests:	100
Failed requests:	0

Total transferred: 4625489 bytes  
HTML transferred: 4556300 bytes  
Requests per second: 25.29 [#/sec] (mean)  
Time per request: 79.094 [ms] (mean)  
Time per request: 39.547 [ms] (mean, across all concurrent requests)  
Transfer rate: 1142.21 [Kbytes/sec] received

**Connection Times (ms)**

	min	mean[+/-sd]	median	max
Connect:	40	53 8.1	51	99
Processing:	12	24 9.4	23	98
Waiting:	5	14 10.6	12	95
Total:	57	77 15.0	75	197

**Percentage of the requests served within a certain time (ms)**

50%	75
66%	77
75%	80
80%	81
90%	85
95%	92
98%	116
99%	197
100%	197 (longest request)

**2.1.1. Ejercicio Evaluable. Carga Http con AB**

Ejecute el benchmark sobre los servidores Http empleados en las prácticas del Bloque 1 (Apache y Nginx). Se deja a la decisión del alumno/a la definición de los parámetros del benchmark, pero se valorará que sea capaz de definir un entorno de ejecución que eleve significativamente la carga de los servidores.

No lo he hecho la vd



## 4. Simulación de carga con JMeter

Apache JMeter es una herramienta de software de código abierto diseñada para realizar pruebas de carga, pruebas de rendimiento y pruebas de estrés en aplicaciones web, servidores y protocolos. Es una de las herramientas más populares y ampliamente utilizadas en la industria para evaluar y medir el rendimiento de sistemas informáticos bajo diversas condiciones

Características interesantes:

- Puede crear concurrencia real pues puede usar varias hebras dentro de la misma CPU
- Puede distribuir el esfuerzo de carga entre varios equipos de la red local
- Se puede ejecutar en modo línea de comandos, lo que permite aligerar la carga de la máquina que está generando las peticiones al servidor y la automatización de algunos tests.
- Podemos 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

¿Qué es un proxy?.

Un servidor proxy es el intermediario entre el usuario y el internet, permitiendo así que la ip del usuario se mantenga privada asignándole una nueva.

Permite acceder a sitios bloqueados, usarse como firewall, filtrar contenido, privacidad.

¿Qué es Flood?

Flood es una plataforma de pruebas de carga en la nube que permite a los equipos de desarrollo y pruebas evaluar el rendimiento y la escalabilidad de sus aplicaciones web, APIs y servicios en la nube. La plataforma Flood proporciona una infraestructura escalable y flexible para ejecutar pruebas de carga de gran escala desde múltiples ubicaciones geográficas.

### a. Instalación de la aplicación para el test con JMeter

En nuestro caso se encuentra el en git de profesor. Lo descargamos y en la propia carpeta se encuentra el docker-compose.yml, luego al al hacer docker compose up el servicio se lanza mostrando los logs de ejecución de sus componentes.

DOCKER-COMPOSE.YML

```
version: '2.0'
services:
  #MongoDB based in the original Mongo Image
  mongodb:
    image: mongo:6
    ports:
      - "27017:27017"
```

```
# Initialize mongodb with data
```

```
mongodbinit:
```

```
  build: ./mongodb
```

```
  links:
```

```
    - mongodb
```

```
# Nodejs App
```

```
nodejs:
```

```
  build: ./nodejs
```

```
  ports:
```

```
    - "3000:3000"
```

```
  links:
```

```
    - mongodb
```

Tenemos 3 servicios, mongodbinit y nodejs dependen de mongodb.

- Node usa el motor de JavaScript V8 de Google: una VM tremendamente rápida y de gran calidad escrita por gente como Lars Bak, uno de los mejores ingenieros del mundo especializados en VMs. No olvidemos que V8 es actualizado constantemente y es uno de los intérpretes más rápidos que puedan existir en la actualidad para cualquier lenguaje dinámico. Además las capacidades de Node para I/O (Entrada/Salida) son realmente ligeras y potentes, dando al desarrollador la posibilidad de utilizar a tope la I/O del sistema. Node soporta protocolos TCP, DNS y HTTP. Una de las cosas más importantes por las que la gente se confunde cuando se explica Node.js es entender qué es exactamente. ¿Es un lenguaje diferente? ¿es solo un framework por encima o es algo más? Node.js definitivamente no es un nuevo lenguaje, y no es solo un framework de JavaScript. Puede ser considerado como un entorno de tiempo de ejecución para JavaScript construido sobre el motor V8 de Google. Entonces, nos proporciona un contexto donde podemos escribir código JavaScript en cualquier plataforma donde se puede instalar Node.js.
- **MongoDB** es una base de datos escalable, de alto rendimiento, orientada a documentos y sin esquema predefinido. MongoDB se aleja del clásico paradigma relacional. En este sentido, podemos llamarla una **base de datos “NoSQL”**, que es un término que se acuñó hace unos años para indicar que no se va a emplear SQL, ampliamente extendido en las bases de datos comerciales como Oracle, MySQL, MariaDB, SQL Server, etc. Se trata de un sistema de bases de datos de código abierto y completamente gratuito, si bien la empresa que lo creó ofrece servicios profesionales a empresas.

En definitiva, node.js usa MongoDB como base de datos.

\* Si realizamos test con muchas hebras y con carga CPU lo adecuado es usar la interfaz de CLI en JMeter \*

### **3.3. Ejercicio evaluable. Simulación de carga con Jmeter**

#### **Comando para ejecutar JMeter desde la terminal:**

```
./apache-jmeter-5.6.3/bin/jmeter.sh -n -t iseP4JMeter-ejercicio3.3.jmc -l  
iseP4JMeter-master/resultados.jtl
```

1. **dentro de la carpeta iseP4Jmeter-master: docker compose up**
2. `sudo ufw allow 3000`: permite el tráfico entrante al puerto 3000 en Ubuntu usando UFW (Uncomplicated Firewall). Sin embargo, si estás obteniendo el error "ERROR: Curl fallo con resultado: 7", significa que la conexión al puerto 3000 en localhost está siendo rechazada, lo que puede indicar que no hay ningún servicio escuchando en ese puerto o que el servicio no está configurado correctamente para aceptar conexiones entrantes. Lo habilitamos, pues es el puerto que pone en el docker-compose que se va a usar para node.js.
3. Listar puertos habilitados para el firewall: `sudo ufw status`
4. **Ver nuestra IP**: ip config, en concreto la de broadcast, multicast, en mi caso es la 172.20.10.4
5. **buscar en el navegador** <http://172.20.10.4:3000>
6. **jsonformatter.org**: para ver los resultados que nos devuelven al ejecutar pruebaEntorno.sh. <https://jsonformatter.org/>
7. **Ejecutar Jmeter**: dentro de /Escritorio/ISE/B2/apache.../bin hacer `./jmeter` o `java -jar ./iseP4JMeter-master/apache-jmeter-5.6.3/bin/ApacheJMeter.jar`

Comenzamos a hacer lo que pide la entrega...

El subdirectorio [JMeter](#) contiene los archivos necesarios para realizar la sesión de prácticas:

- alumnos.csv: Archivo con credenciales de alumnos
- administradores.csv: Archivo con credenciales de administradores
- apiAlumno.log: Log de acceso Http en formato apache.

La prueba de JMeter debe:

1. Simular el acceso concurrente de un número de alumnos no menor a 10 y un número de administradores no menor de 3.

Para ello lo que hacemos es que al crear los grupos de hilos, al grupo de los alumnos le decimos que queremos 10 hebras, y de la misma forma, al grupo de administradores le decimos que queremos 3.

2. Parametrizar el "EndPoint" del servicio mediante variables para la dirección y puerto del servidor. Emplee "User Defined Variables" del Test Plan.

Añadimos en el Plan de Pruebas en variables definidas por el usuario, la ip y el puerto, cada uno con sus respectivos valores.

3. Definir globalmente las propiedades de los accesos Http y la Autenticación Basic. Emplee HTTP Request Defaults y HTTP Authorization Manager.
4. Los accesos de alumnos y administradores se modelarán con 2 Thread Groups independientes.
5. La carga de accesos de administradores se modelará empleando el registro de accesos del archivo apiAlumno.log

## ESTRUCTURA DEL APACHE JMETER

### **PLAN DE PRUEBAS** etsi alumnos api

(Definimos las variables IP y PUERTO por donde se ejecuta, 3000)

#### **VALORES POR DEFECTO PARA LA PETICIÓN HTTP** access to etsii api

(marcamos protocolo, en este caso http, nombre del servidor o ip: \${IP}, y puerto: \${PUERTO}, \${..} vbles definidas en el plan de pruebas)

#### **GESTIÓN DE AUTORIZACIÓN HTTP** basic.auth api

(En las autorizaciones almacenadas en el gestor de autorización especificamos URL Base: [http://\\${IP}:\\${PUERTO}/api/v1/auth/login](http://${IP}:${PUERTO}/api/v1/auth/login)

Nombre de Usuario: el necesario, en este caso etsiiApi

Contraseña: la necesaria, en este caso laApiDeLaETSII DaLache

Mechanism: BASIC)

#### **GRUPO DE HILOS** Alumnos

(debemos especificar: número de hilos, periodo de subida y contador en bucle, en mi caso 30, 15, 5)

#### **CONFIGURACIÓN DEL CSV DATA SET** credenciales alumnos

(Permite cargar los datos de los alumnos desde un archivo, luego hay que especificar

Ruta del archivo csv: ./jMeter/alumnos.csv

Codificación del fichero: UTF-8

Utilizar primera línea como nombre de variable: true

Delimitador: , (abre el archivo y mira que delimitadores están marcados)

Datos entrecomillados: False

Reciclar en EOF: True

Para el hilo final del EOF: False

Modo compartido: Actual grupo de hilos )

#### **PETICIÓN HTTP** login alumnos

(Hace una solicitud HTTP para el inicio de sesión del alumno, se especifica

Petición HTTP: POST

Ruta: api/v1/auth/login

Además queremos que se envíe el usuario y la contraseña como parámetros, luego en la tabla los añadimos con el mismo nombre que sale en el .csv)

**EXTRACTOR DE EXPRESIONES REGULARES** extract JWT Token  
(cuando hacemos la petición http, esta nos devuelve una respuesta, con esto extraemos el token de dicha respuesta y así poder iniciar sesión)

**TEMPORIZADOR ALEATORIO GAUSSIANO** dejado por defecto

**PETICIÓN HTTP**

(Hace una solicitud HTTP para el inicio de sesión del alumno, se especifica

Petición HTTP: GET

Ruta: /api/v1/alumnos/alumno/\${\_\_urlencode(\${login} )

**GESTOR DE CABECERA HTTP** JWT Token

(Queremos añadir el token extraído como encabezado de autorización, luego

Nombre. Authoritation

Valor: Bearer \${token})

**GRUPO DE HILOS** administradores

**CONFIGURACIÓN DEL CSV DATA SET** credenciales administrador

**PETICIÓN HTTP** login administrador

**EXTRACTOR DE EXPRESIONES REGULARES** extract JWT Token

**MUESTREADOR DE ACCESO AL LOG** Accesos administrativos

(Permite que el administrador tenga libre acceso al archivo.log, especificamos

Protocolo: HTTP

Servidor: \${IP}

Puerto: \${PUERTO}

Archivo del log: en este caso apiAlumnos.log)

**GESTOR DE CABECERA HTTP** JWT Token

**TEMPORIZADOR ALEATORIO GAUSSIANO**

**REPORTE RESUMEN**

**VER RESULTADOS EN ÁRBOL**

**INFORME AGREGADO**

En resumen, te identificas como el home de la API y te devuelve un token, este te permite identificarte en tu path correspondiente ..../email. Está programado para que la respuesta a un post sea tu token en concreto. Al hacer el GET, se pasa el token.

- ¿Qué es un archivo .log?

Los logs son archivos de texto normales. Como su nombre indica, estos ficheros registran todos los procesos que han sido definidos como relevantes por el programador de la aplicación. Por ejemplo, en el caso de los archivos log de una base de datos se registran todos los cambios de aquellas transacciones completadas exitosamente. Así, en caso de que un fallo del sistema elimine información de la base de datos, el log será la clave para la restauración completa de la base de datos correspondiente.

En nuestro caso, el archivo log contiene las peticiones GET que tiene que hacer el administrador.

Las diferencias entre las solicitudes HTTP POST y GET son principalmente en cómo se transmiten los datos y en el propósito de cada método. Aquí tienes un resumen de las diferencias clave:

#### **Método GET:**

1. **Transmisión de datos:** Los datos se envían como parte de la URL en la línea de solicitud.
2. **Visibilidad de los datos:** Los parámetros se muestran en la URL, lo que los hace visibles en el historial del navegador y en otros lugares, como registros de servidores web.
3. **Longitud de la URL:** Limitado por la longitud máxima de la URL, lo que puede generar problemas con grandes cantidades de datos.
4. **Cacheable:** Las solicitudes GET son cacheables, lo que significa que los resultados pueden almacenarse en caché por el navegador o en servidores proxy.
5. **Seguridad:** Menos seguro para datos sensibles, ya que los parámetros se muestran en la URL.

#### **Método POST:**

1. **Transmisión de datos:** Los datos se envían en el cuerpo del mensaje HTTP.
2. **Visibilidad de los datos:** Los parámetros no son visibles en la URL, lo que proporciona una mayor privacidad y seguridad.
3. **Longitud de la URL:** No está limitado por la longitud máxima de la URL, lo que permite el envío de grandes cantidades de datos.
4. **No cacheable por defecto:** Las solicitudes POST no son cacheables por defecto, aunque esto puede ser controlado por cabeceras específicas.
5. **Seguridad:** Más adecuado para el envío de datos sensibles, ya que los parámetros no se muestran en la URL y se envían de forma más segura en el cuerpo de la solicitud.

#### **Consideraciones adicionales:**

- **Propósito:** GET se utiliza generalmente para recuperar recursos, mientras que POST se utiliza para enviar datos al servidor para su procesamiento.
- **Usos comunes:** GET se utiliza comúnmente para solicitudes de lectura (por ejemplo, obtener datos de una API), mientras que POST se utiliza para enviar datos del formulario, realizar cambios en el servidor (como crear, actualizar o eliminar recursos) y para enviar datos confidenciales, como contraseñas.

En resumen, la elección entre GET y POST depende del propósito de la solicitud y de la naturaleza de los datos que se están enviando. GET es más adecuado para solicitudes de lectura y para enviar datos no sensibles, mientras que POST es más adecuado para enviar datos sensibles y realizar cambios en el servidor.

## 5. JMETER VS APACHE BENCHMARK

- En JMeter puedes decidir el número de hilos
- En Apache Benchmark fue diseñado originalmente para probar Apache HTTP pero puede probar cualquier servidor web sobre él.
- En JMeter puedes probar diferentes servidores y protocolos.
- Apache Benchmark se puede ejecutar desde la terminal sin necesidad del uso de un archivo.
- JMeter tiene sus propias interfaces para crear y ejecutar pruebas de carga
- Apache Benchmark consume menos CPU
- JMeter requiere más recursos ya que trata con muchos datos al mismo tiempo.
- Apache Benchmark es adecuado para probar UNA API, si queremos probar más de una se complica.
- JMeter es una herramienta muy poderosa para crear cualquier tipo de escenarios de prueba de carga con la capacidad de hacer manipulaciones complicadas de datos entre solicitudes.
- AB ofrece entradas de registro y reporte cuando la prueba ha terminado, pero no se pueden ver muchas métricas y estadísticas en el informe.
- Estás limitado con la autenticación básica en AB.
- JMeter tiene soporte para diferentes tipo de autenticación.
- AB no te permite controlar tu caché DNS, JMeter si con el gerente de caché DNS.
- JMeter permite la concurrencia real

## 6. Monitoring

### a. TOP

Nos permite obtener métricas de procesos y sobre el uso de los recursos de cómputo del servidor donde se ejecuta. Htop es lo mismo pero con mayor capacidad de visualización (<https://docs.rockylinux.org/gemstones/htop/>).

Lo que hace la herramienta top, realmente, es reunir y presentar los datos de monitorización ofrecidos por el sistema operativo.

Top proporciona una visión dinámica en tiempo real de un funcionamiento del sistema. Muestra información resumida del sistema, así como una lista de procesos o hilos que están gestionando actualmente Linux kernel. Los tipos de información resumida del sistema mostrados y los tipos, orden y tamaño de la información mostrada para los procesos son todos los usuarios configurables.

top [opciones]

```
isabel@isabel-VivoBook-ASUSLaptop-X421EA-S433E: ~/Escritorio/ISE/B2/iseP4JMeter-master$ top

top - 12:34:16 up 13:58,  1 user,  load average: 1,02, 0,96, 0,89
Tareas: 349 total,  1 ejecutar,  347 hibernar,  0 detener,  1 zombie
%Cpu(s):  3,1 us,  1,5 sy,  0,0 ni, 95,2 id,  0,1 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Mem : 15665,0 total,  688,0 libre,  5707,4 usado,  9269,7 búfer/caché
MiB Intercambio:  2048,0 total,  2047,7 libre,  0,2 usado.  7334,0 dispon

  PID  USUARIO  PR  NI  VIRT  RES  SHR  S  %CPU  %MEM  HORA+  ORDEN
33392  isabel    20   0 3295664 444640 121672 S   10,0   2,8  4:37.87 Isolate+
1505   isabel    20   0 6314792 365480 180108 S    8,3   2,3 33:01.89 gnome-s+
1058   isabel    9  -11 3072940 36852  30836 S    4,3   0,2 57:28.07 pulseau+
9699   isabel    20   0 4609688 303336 163940 S    3,7   1,9 11:14.79 spotify
4421   isabel    20   0  14,7g 944236 337768 S    2,0   5,9 51:30.04 firefox
40991  isabel    20   0 3051420 366496 116024 S    1,3   2,3  0:37.10 Isolate+
39801  isabel    20   0  558652  53656  40716 S    1,0   0,3  0:01.05 gnome-t+
40384  isabel    20   0 2539676 180560  96624 S    1,0   1,1  0:35.24 Isolate+
 895   root      20   0  2168892  43976  30848 S    0,3   0,3  0:38.35 contain+
1787   isabel    20   0  318360  12276  7168 S    0,3   0,1  1:14.27 ibus-da+
33291  isabel    20   0 2604364 261104 102884 S    0,3   1,6  0:09.74 Isolate+
36187  root      0 -20    0      0      0 I    0,3   0,0  0:16.01 kworker+
36664  root      0 -20    0      0      0 I    0,3   0,0  0:14.45 kworker+
37107  root      20   0    0      0      0 I    0,3   0,0  0:01.28 kworker+
```

- top: modo de visualización, ahora está en modo general pues no puse parámetros.
- 12:34:16: hora actual
- up 13:58: tiempo que el sistema lleva en marcha, 13 horas y 54 minutos
- 1 user: usuarios conectados



- load average: carga en los últimos 5, 10 y 15 minutos. No representa el uso de la CPU sino el tiempo de espera de los procesos. Indican cuánto de congestionado está tu sistema. Si es menor que 1, significa que de media los procesos son atendidos a medida que llegan, si es mayor que uno, existen tiempos de espera.
- tareas: procesos en ejecución, procesos activos, procesos en ejecución sin hacer nada, procesos cargados pero detenidos y procesos zombie (ejecución terminada pero reside en memoria)
- %CPU: porcentajes de uso de CPU (usuario, sistema, demanda de procesos de baja prioridad, idle, tiempo dedicado a E/S, interrupciones hardware, interrupciones software, tiempo usado por hipervisor si estamos virtualizando)
- Memoria (se sobreentiende)

En la tabla, de izquierda a derecha:

- PID: id usuario
- USER: nombre
- PR: prioridad
- NI: aumenta o decrementa la prioridad
- VIRT: consumo de memoria
- RES: consumo de memoria RAM
- SHR: cantidad de memoria virtual
- Estado
- %CPU: capacidad de procesamiento que consume
- %MEM: % de mem RAM
- HORA+: tiempo total de CPU usado desde su inicio
- ORDEN: nombre del proceso o el comando usado para iniciar la tarea

Más información: <https://man7.org/linux/man-pages/man1/top.1.html>

### **b. Sistema de ficheros /proc.**

El sistema de archivos /proc principalmente proporciona acceso al estado de cada proceso activo y hebra del sistema. /proc contiene un SA's virtual, no existe físicamente en disco, si no que el kernel lo crea en memoria.

En este directorio encontramos el origen de la información sobre monitorización. Por ejemplo: loadavg, meminfo, uptime, vmstats, cpuinfo. Mientras que /proc/<processId> contiene información detallada sobre los procesos en ejecución.

¿proc y monitorización de Linux? top extrae la información a través de /proc.

### **c. Stress**

Nos permite evaluar y probar el rendimiento de los sistemas linux en diversas condiciones. Es una herramienta simple pero poderosa diseñada para imponer una cantidad configurable de CPU, memoria, E/S o estrés en disco en un sistema Linux. Al simular cargas de trabajo pesadas, permite a los administradores observar cómo responde el sistema bajo presión.

Instalación: `sudo yum install stress`

Ejemplo de uso: `stress --cpu 8 -v --timeout 300`

- `--cpu`: número de hilos de trabajo en `sqrt()`.
- `v` ó `--verbose`: modo detalle
- `--timeout` ó `-t`: tiempo que se ejecuta el stress en segundos

#### 4.1.1. Ejercicio evaluable. Stress + Top

Simule una carga de trabajo en su equipo anfitrión o en un MV empleando el programa stress. Monitoriza con Top (o HTop) la evolución de la carga y justifica razonadamente los valores obtenidos.

1. Descargamos stress: `sudo apt install stress`  
Parámetros: `stress --help`
2. Instalamos htop o top: `sudo apt install htop`
3. Ejecutamos stress para cargar el sistema, por ejemplo:  
`stress --cpu 4 --timeout 60s`
4. Abrimos otra terminal y ejecutamos top o htop para monitorear la carga del sistema en tiempo real.

```
isabel@isabel-VivoBook-ASUSLaptop-X421EA-S433E: ~$ stress --cpu 4 --timeout 60s
stress: info: [3101] dispatching hogs: 4 cpu, 0 io, 0 vm, 0 hdd
stress: info: [3101] successful run completed in 60s
isabel@isabel-VivoBook-ASUSLaptop-X421EA-S433E: ~$ top
top - 11:56:02 up 1:37, 0.93s, 1.88i, 1.37o, 9.93s, 1.88i, 1.37o, 9.93s
PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
53133 isabel    20   0 3708    256    256 R 100.0 0.0 0:10.76 stress --cpu 4 --timeout 60s
53137 isabel    20   0 3708    256    256 R 100.0 0.0 0:10.78 stress --cpu 4 --timeout 60s
53138 isabel    20   0 3708    256    256 R 100.0 0.0 0:10.79 stress --cpu 4 --timeout 60s
53139 isabel    20   0 3708    256    256 R 100.0 0.0 0:10.79 stress --cpu 4 --timeout 60s
51837 isabel    20  60 20808  2728  1168 S 3.9 1.8 0:36.27 /snap/firefox/3836/usr/lib/firefox/f
53093 isabel    20  21844 6784 3712 R 2.0 0.0 0:02.64 htop
916 isabel    20  61368 3696 1718 S 0.7 2.2 23:10.01 /usr/sbin/gnome-shell
15883 isabel    20  13.9c 11676 3608 S 0.7 7.5 32:59.23 /snap/firefox/3836/usr/lib/firefox/f
3074 isabel    20  32538 5896 1178 S 0.7 3.3 7:11.39 /snap/firefox/3836/usr/lib/firefox/f
9073 root      20  11178 45408 32256 S 0.7 0.3 1:05.14 /usr/bin/containerd
1 root      20  1648 13184 8320 S 0.0 0.1 0:20.27 /lib/systemd/systemd --system --dese
253 root     19  56616 28800 27648 S 0.0 0.2 0:01.87 /lib/systemd/systemd-journald
280 root     20  26996 6800 4624 S 0.0 0.0 0:00.89 /lib/systemd/systemd-udev
449 root     20  14864 6784 6816 S 0.0 0.0 0:22.66 /lib/systemd/systemd-cond
450 root     20  26600 14616 9344 S 0.0 0.1 0:06.76 /lib/systemd/systemd-resolved
488 root     20  2438 7724 6956 S 0.0 0.0 0:00.59 /usr/libexec/accounts-daemon
489 root     20  3816 2048 1928 S 0.0 0.0 0:02.01 /usr/sbin/acpid
491 avahi     20  7636 3840 3584 S 0.0 0.0 0:00.40 avahi-daemon: running [isabel-VivoB
493 root     20  18152 2944 2688 S 0.0 0.0 0:00.03 /usr/sbin/cron -f -P
495 messageb 20  11160 6528 3840 S 0.0 0.0 0:17.62 @dbus-daemon --system --address=system
497 root     20  3368 19736 16356 S 0.0 0.1 0:13.67 /usr/sbin/NetworkManager --no-daemon
502 root     20  81844 3840 3584 S 0.0 0.0 0:02.40 /usr/sbin/lrqlbalance --foreground
506 root     20  49684 26068 11648 S 0.0 0.1 0:00.08 /usr/bin/python3 /usr/bin/networkd-
509 root     20  2468 11492 7876 S 0.0 0.1 0:04.84 /usr/libexec/polkitd --no-debug
512 root     20  2438 7688 6848 S 0.0 0.0 0:00.02 /usr/libexec/power-profiles-daemon
517 syslog   20  2178 5888 4224 S 0.0 0.0 0:00.52 /usr/sbin/rsyslogd -n -iNONE
520 root     20  2468 11492 7876 S 0.0 0.1 0:00.00 /usr/libexec/polkitd --no-debug
524 root     20  82844 3640 3584 S 0.0 0.0 0:00.00 /usr/sbin/lrqlbalance --foreground
525 root     20  2438 7688 6848 S 0.0 0.0 0:00.00 /usr/libexec/power-profiles-daemon
530 root     20  2468 11492 7876 S 0.0 0.1 0:01.38 /usr/libexec/polkitd --no-debug
531 root     20  2438 7688 6848 S 0.0 0.0 0:00.00 /usr/libexec/power-profiles-daemon
532 root     20  2438 7724 6956 S 0.0 0.0 0:00.53 /usr/libexec/accounts-daemon
534 root     20  2438 7724 6956 S 0.0 0.0 0:00.01 /usr/libexec/accounts-daemon
538 root     20  2398 6784 6272 S 0.0 0.0 0:00.05 /usr/libexec/switchover-control
541 root     20  48228 8168 6656 S 0.0 0.1 0:01.39 /lib/systemd/systemd-logind
543 root     20  2748 10268 4472 S 0.0 0.1 0:03.61 /usr/sbin/thermald --system --dbus-
544 root     20  3838 12444 16524 S 0.0 0.1 0:00.30 /usr/libexec/udisks2/udisksd
549 root     20  18680 11776 9856 S 0.0 0.1 0:03.33 /sbin/wpa_supplicant -u -s -O /run/v
554 root     20  3368 19736 16356 S 0.0 0.1 0:00.40 /usr/sbin/NetworkManager --no-daemon
556 avahi     20  7444 1536 1280 S 0.0 0.0 0:00.00 avahi-daemon: chroot helper
557 syslog   20  2178 5888 4224 S 0.0 0.0 0:00.20 /usr/sbin/rsyslogd -n -iNONE
558 syslog   20  2178 5888 4224 S 0.0 0.0 0:00.04 /usr/sbin/rsyslogd -n -iNONE
559 syslog   20  2178 5888 4224 S 0.0 0.0 0:00.25 /usr/sbin/rsyslogd -n -iNONE
560 root     20  3368 19736 16356 S 0.0 0.1 0:05.49 /usr/sbin/NetworkManager --no-daemon
```

#### d. Programación de tareas periódicas como cron

Cron es un administrador de tareas de Linux que permite ejecutar comandos en un momento determinado, por ejemplo, cada minuto, día, semana o mes. Si queremos trabajar con cron, podemos hacerlo a través del comando crontab.

¿Qué es crontab? Es un archivo de texto donde se listan todas las tareas que deben ejecutarse y el momento en el que deben hacerlo.

A continuación podremos ver las diferentes opciones del crontab:

- crontab -e: edita el archivo crontab de un usuario. Cada línea que se configure será una tarea Cron.
- crontab -l: lista el archivo crontab del usuario, con todas sus tareas configuradas.
- crontab -r: elimina el archivo crontab de un usuario. El borrado no es recuperable.

Formato: Minuto Hora Día-del-Mes Mes Día-de-la-Semana Comando-a-Ejecutar  
(Si en lugar de un número utilizamos un asterisco, el comando indicado se ejecutará cada minuto, hora, día de mes, mes o día de la semana)

#### e. Logger y Log

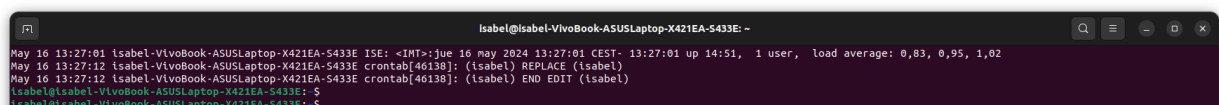
Al proceso de generación del log se le suele llamar guardar, registrar o loguear (un neologismo del inglés logging) y al **proceso o sistema que realiza la grabación en el log** se le suele llamar logger o registrador.

##### 4.2.1.- Ejercicio evaluable. Tarea periódica con cron.

**Empleando cron y la utilidad logger. Programe una tarea periódica en su espacio de usuario que genere un mensaje de log con la etiqueta "ISE" y cuyo texto tenga la forma "<SusIniciales>: <Fecha y hora actual> – <Carga actual del Sistema>".**

Hacemos crontab -e y escribimos \* \* \* \* \* logger -t "ISE" "IMT :\$(date) - \$(uptime)"

**Ver contenido:** cat /var/log/syslog



```
isabel@isabel-VivoBook-ASUSLaptop-X421EA-S433E: ~  
May 16 13:27:01 isabel-VivoBook-ASUSLaptop-X421EA-S433E ISE: <IMT>:jue 16 may 2024 13:27:01 CEST- 13:27:01 up 14:51, 1 user, load average: 0,03, 0,95, 1,02  
May 16 13:27:12 isabel-VivoBook-ASUSLaptop-X421EA-S433E crontab[46138]: (isabel) REPLACE (isabel)  
May 16 13:27:12 isabel-VivoBook-ASUSLaptop-X421EA-S433E crontab[46138]: (isabel) END EDIT (isabel)  
isabel@isabel-VivoBook-ASUSLaptop-X421EA-S433E: $  
isabel@isabel-VivoBook-ASUSLaptop-X421EA-S433E: $
```

#### **f. Logs del sistema**

Una forma básica de monitorización es analizar los logs generados por el sistema operativo o los servicios instalados en el servidor. En Linux los logs se almacenan en el directorio /var/logs (/var también contiene los servicios instalados)

Por ejemplo, cron, cron-20240222, cron- 20240225, etc. Estos archivos se corresponden con “rotaciones” del log original (cron en el ejemplo anterior). Los archivos de logs se rotan para mantener un tamaño reducido que permita su consulta.

- logrotate: programa responsable de hacer la labor de rotado. Este funciona como tarea periódica en cron. Logrotate permite, entre otras funciones, definir el rotado por tamaño o antigüedad, comprimir los archivos rotados o ejecutar tareas en el momento de la rotación. El fichero de configuración por defecto es /etc/logrotate.conf, pero las configuraciones específicas de los servicios las puede encontrar en /etc/logrotate.d/.
- Apache Httpd rotatelogs es otra herramienta específica para realizar el rotado de los archivos de logs. Esta herramienta se ha vuelto muy popular por que no precisa notificar/reiniciar el servicio original y porque es muy ligera y fácil de integrar con cualquier programa que emplee la salida estándar (stdout) para logs.
- En Linux systemd centraliza la recepción y posible almacenamiento de todos los logs. El servicio systemd-journald está activo por defecto para el registro y la consulta de la actividad o “journal” del sistema. Por defecto, la información del journal se almacena de forma volátil (en memoria y en /run), de forma que se pierde al reiniciar el servidor. Para almacenarlos de forma permanente, debe modificarse la configuración en /etc/systemd/journald.conf. En Rocky 9 la persistencia por defecto es “auto” por lo que, simplemente, creando el directorio /var/log/journal, se activa la persistencia no volátil.

#### **4.3.1.- Ejercicio Evaluable. Logs de arranque del sistema.**

**Consulte los logs del último arranque de una MV Rocky empleando journalctl. Concretamente, presente mensajes de niveles warning o más importantes.**

journalctl -p warning -b

## 7. Grafana + Prometheus

### a. Grafana

Grafana es una plataforma de análisis y visualización de datos de código abierto que se utiliza comúnmente para monitorear y analizar métricas en tiempo real. Para construir su pila tiene distintos componentes que se estructuran en: dashboards, paneles y alertas.

Grafana proporciona los servicios de visualización y alertas sobre los datos de telemetría, pero no realiza por si misma la extracción de esta información.

### b. Prometheus

La encargada de extraer y almacenar la información de monitorización de forma eficiente y fácilmente adaptable a las necesidades de cada proyecto.

- Optimización para la gestión de series temporales
- Modelos de datos abiertos y personalizables
- Métricas estandarizadas

Grafana (Visualización + Alarmas) junto con Prometheus (Recolección de Datos y Almacenamiento) son una combinación muy habitual en los Stacks Tecnológicos de muchas organizaciones IT.

### i. Node Exporter

La opción de node exporter en Prometheus se define como una herramienta a cargo de la obtención de estadísticas de las aplicaciones bajo el formato que estos sistemas implementan, para luego convertir esas estadísticas en métricas que la plataforma de Prometheus pueda implementar.

Posterior a esto, las estadísticas obtenidas y transformadas en métricas se exponen en una URL que sea compatible con Prometheus.

Su fichero de configuración es `node_exporter.service` en `/etc/systemd/system`. El nuestro del ejercicio es

```
[UNIT]
Description=Node Exporter
After=network.target

[Service]
User=node_exporter
Group=node_exporter
Type=simple
ExecStart=/bin/sh -c '/usr/local/bin/node_exporter --collector.systemd'

[Install]
WantedBy=multi-user.target
```

- UNIT: metadatos descriptivos de servicio, descripción o dependencias
- Service: detalles de cómo se ejecuta. Especificamos usuario, grupo, y tipo simple (1 proceso) y la ubicación del ejecutable
- Install: cuando se debe habilitar el servicio. En nuestro caso, cuando se inicie el nivel de ejecución multiusuario.

En el caso de que queramos más información por parte de node\_exporter, este es un ejemplo:

```
sudo tee /etc/systemd/system/node_exporter.service<<EOF
[Unit]
Description=Prometheus Node Exporter
Documentation=https://github.com/prometheus/node_exporter
Wants=network-online.target
After=network-online.target

[Service]
Type=simple
User=prometheus
Group=prometheus
ExecReload=/bin/kill -HUP $MAINPID
ExecStart=/usr/local/bin/node_exporter \
    --collector.cpu \
    --collector.diskstats \
    --collector.filesystem \
    --collector.loadavg \
    --collector.meminfo \
    --collector.filefd \
    --collector.netdev \
    --collector.stat \
    --collector.netstat \
    --collector.systemd \
    --collector.uname \
    --collector.vmstat \
    --collector.time \
    --collector.mdadm \
    --collector.zfs \
    --collector.tcpstat \
    --collector.bonding \
    --collector.hwmon \
    --collector.arp \
    --web.listen-address=0.0.0.0:9100 \
    --web.telemetry-path="/metrics"

[Install]
WantedBy=multi-user.target
EOF
```

Los siguientes pasos tras la generación de archivo son:

- sudo systemctl daemon-reload
- sudo systemctl enable node\_exporter
- sudo systemctl start node\_exporter
- sudo systemctl status node\_exporter

Para poder acceder a las métricas de node exporter, debemos tener el archivo prometheus.yml configurado, en nuestro caso es:

```
---
global:
  scrape_interval: 5s
scrape_configs:
  - job_name: "prometheus_service"
    static_configs:
      - targets: ["prometheus:9090"]
  - job_name: 'node_exporter_metrics'
    scrape_interval: 5s
    static_configs:
      - targets: ['192.168.56.101:9100']
```

No nos podemos olvidar de que tanto prometheus como grafana usan contenedores docker, luego previo a cualquier cosa, se debe ejecutar docker compose up, el archivo docker-compose.yml en nuestro caso es:

```
---
version: "3"
services:
  prometheus:
    image: prom/prometheus:v2.50.0
    ports:
      - 9090:9090
    volumes:
      - ./prometheus_data:/prometheus
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    command:
      - "--config.file=/etc/prometheus/prometheus.yml"
  grafana:
    image: grafana/grafana:9.1.0
    ports:
      - 4000:3000
    volumes:
      - ./grafana_data:/var/lib/grafana
    depends_on:
      - prometheus
```

Una vez esté ejecutándose, podemos ver las métricas en <http://localhost:9090/metrics> (puerto 9090 pues es el especificado en el docker-compose.yml)

Una vez en el navegador, en la opción graph podemos probar métricas, como por ejemplo:

- `rate(node_cpu_seconds_total{mode="system"}[1m])`  
La cantidad media de tiempo de CPU pasó en modo de sistema, por segundo, durante el último minuto (en segundos)
- `node_filesystem_avail_bytes`  
El espacio del sistema de archivos disponible para los usuarios no root (en bytes)
- `rate(node_network_receive_bytes_total[1m])`  
El tráfico medio de la red recibió, por segundo, en el último minuto (en bytes)

Ahora podemos ver las cosas más bonitas y legibles con grafana.

Para acoplar prometheus con grafana debemos acceder con <http://localhost:4000> a grafana, y una vez dentro en Configuration - data sources - add data source - type prometheus, debemos poner <http://localhost:9090>, que es donde tenemos prometheus :D

Ahí ya podemos seleccionar algún dashboard predefinido para Grafana:

<https://grafana.com/grafana/dashboards/10180-kds-linux-hosts/>

<https://grafana.com/grafana/dashboards/1860-node-exporter-full/>

y empezar a jugar.

#### **4.1.1.- Ejercicio Evaluable. Monitorización de un equipo Linux.**

Prácticamente lo denso ya está explicado.

- **El alumno/a debe extender el dashboard anterior para incorporar indicadores sobre el nivel de activación ("Activo"/"Inactivo", 1/0) de los servicios: SSHD y Apache Httpd en el equipo Linux monitorizado.**

```
node_systemd_unit_state{name="sshd.service",state="active"}
node_systemd_unit_state{name="httpd.service",state="active"}
```

- **Deberá agregar un nuevo panel sobre el nivel de uso total de la CPU en tanto por ciento (%). A este panel se le asociará una alarma para que se dispare cuando la media del uso de CPU supere el 75% de CPU durante 5 minutos. Ponga de manifiesto el funcionamiento de la alarma empleando alguna herramienta de carga de las vistas en clase (por ejemplo, stress).**






1 Set a query and alert condition


Query patterns Explain ☐ Run queries Builder **Beta** Code


Metrics browser > `100 - (avg(irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)`

> Options Legend: {{mode}} Format: Time series Step: Type: Range

▼ A (Expression)   

Operation Classic condition ▼

Conditions WHEN avg() ▼ OF B ▼ IS ABOVE ▼ 75 



+ Add query + Add expression **Run queries**

**Set alert condition**  
Select one of your queries or expressions set above that contains your alert condition.

☐ B - query

☒ A - expression

## VARIOS CONCEPTOS DE TIPOS TEST:

- ¿Qué es el ajuste de sistemas?  
Es la acción de modificar parámetros del sistema para optimizar su funcionamiento, según los resultados obtenidos mediante monitorización.
- ¿Cómo se comunican los sistemas basados en microservicios?  
Mediante APIs o sistemas de mensajes
- Al hacer el test de Jmeter ¿Dónde generamos carga?  
En el contenedor de la API, en el de la BD, en la MV que los contiene y en el cliente que lanza el test

## **COMPROBACIONES PREVIAS AL EXAMEN**

- Conectividad por SSH
- Conectividad de las VM a la red
- IPs
- Ejecución de JMeter
- Ejecución de Grafana y prometheus
- MV con prompt pedido `PS1='[\u@\h -\t \w]$ '`
- Usuarios con capacidad de root

Mucha suerte jugadores.

Isa <3