

Prácticas Ingeniería de Servidores

Ismael Sallami Moreno

`ism350zsallami@correo.ugr.es`

`https://ismael-sallami.github.io/`

`https://elblogdeismael.github.io/`

Universidad de Granada

Licencia

Este trabajo está licenciado bajo una [Licencia Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional](https://creativecommons.org/licenses/by-nc-nd/4.0/). <https://creativecommons.org/licenses/by-nc-nd/4.0/>

Usted es libre de:

- **Compartir** — copiar y redistribuir el material en cualquier medio o formato.

Bajo los siguientes términos:

- **Reconocimiento** — Debe otorgar el crédito adecuado, proporcionar un enlace a la licencia e indicar si se han realizado cambios. Puede hacerlo de cualquier manera razonable, pero no de una manera que sugiera que tiene el apoyo del licenciante o lo recibe por el uso que hace.
- **NoComercial** — No puede utilizar el material para fines comerciales.
- **SinObraDerivada** — Si remezcla, transforma o crea a partir del material, no puede distribuir el material modificado.



Índice general

I	Apuntes de Clase	7
1.	Bloque 1	9
1.1.	Ping y SSH	9
1.2.	LVM y RAID	10
1.2.1.	LVM (Logical Volume Manager)	10
1.2.2.	Directorios base en Linux	10
1.2.3.	RAID (Redundant Array of Independent Disks)	10
1.2.4.	Ventajas	10
1.2.5.	Niveles de RAID	10
1.2.6.	Tipos de RAID	11
1.2.7.	Ejercicio Opcional: Configuración de RAID1 para /var	11
1.2.8.	Objetivo	11
1.2.9.	Pasos a seguir	11
1.2.10.	LVM (Logical Volume Manager)	12
1.2.11.	Conceptos Clave	12
1.2.12.	Visualización de LVM	12
1.2.13.	Consideraciones sobre Volumen Groups	12
1.2.14.	Creación del LVM sobre el RAID	13
1.2.15.	Movimiento de /var al RAID	13
1.2.16.	Resolución de Problemas	13
1.2.17.	Problema 1: Configurar el sistema de archivos en rvar	13
1.2.18.	Problema 2: Montar y trasladar /var al nuevo volumen	13
1.2.19.	Interpretación de lsblk	15
1.3.	Firewall + SSHD	16
1.3.1.	Ejercicio Opcional	16
1.3.2.	SSH	16
1.3.3.	Ejercicio Opcional	19
1.4.	Automatización con Ansible	20
1.4.1.	Ejercicio Obligatorio	22
2.	Bloque 2	23
2.1.	Breve introducción a Docker y contenedores	23
2.2.	Monitoring	30
2.3.	Cron	33
2.4.	Grafana + Prometheus	33
2.4.1.	Ejercicio Obligatorio. Monitorización con Grafana + Prometheus.	36

II	Prácticas	37
3.	Bloque 1	39
3.1.	Ejercicio 1 Opcional	39
3.1.1.	Solución	39
3.2.	Servidor con LVM + RAID	43
3.2.1.	Aspectos clave de LVM	43
3.2.2.	Niveles de RAID: 0, 1 y 5	44
3.2.3.	Aspectos clave para la administración de servidores Linux	46
3.2.4.	Ejercicio Opcional	47
3.3.	Acceso seguro al servidor: Firewall + SSHD	50
3.3.1.	Iptables	50
3.3.2.	firewalld y firewall-cmd en Rocky Linux	51
3.3.3.	Ejercicio Opcional	53
3.4.	SSH	55
3.4.1.	Administración remota con SSH y criptografía asimétrica	55
3.4.2.	Ejercicio Opcional	56
3.5.	Automatización de la configuración con Ansible	59
3.5.1.	Introducción a Ansible	59
3.5.2.	Ejercicio Obligatorio	60
4.	Bloque 2	63
4.1.	Prerequisitos	63
4.1.1.	¿Qué es un contenedor?	63
4.1.2.	Ejemplos	65
4.1.3.	Instalación de Docker y Docker Compose	66
4.1.4.	Guía rápida de prueba	67
4.1.5.	Instalación de Docker Engine en Rocky Linux 9 (o CentOS Stream 9)	68
4.2.	Microservicios	69
4.3.	Ejercicio Evaluable. Ejecutar “Hello World”	69
4.4.	BenchMarks	70
4.4.1.	¿Qué es un Benchmark?	70
4.4.2.	¿Por qué crear un Benchmark propio?	70
4.4.3.	OpenBenchmarking y Phoronix Test Suite (PTS)	71
4.4.4.	Ejercicio Opcional. Ejecutar un Benchmark	72
4.4.5.	Resumen: Phoronix Test Suite en Máquina Local vs Docker	74
4.4.6.	Ejercicio Opcional. Comparar benchmark en contenedor y en VM	77
4.4.7.	Apache Benchmarck	78
4.4.8.	Ejercicio Opcional. Carga Http con AB	80
4.5.	Simulación de Carga con Jmeter	83
4.5.1.	Comparativa de herramientas de pruebas de carga y rendimiento	83
4.5.2.	Instalación de Jmeter	85
4.5.3.	Detalles sobre el Dockerfile y Docker-Compose	85
4.5.4.	Ejercicio Obligatorio. Simulación de Carga Http con JMeter	87
4.6.	Monitoring	90
4.6.1.	Top	90
4.6.2.	4.1.1.- Ejercicio Opcional. Stress + Top	91

4.7.	Programación de tareas periódicas con cron	92
4.7.1.	Ejercicio Opcional. Tarea periódica con cron	93
4.8.	Logs del sistema	93
4.8.1.	Ejercicio Opcional. Logs de arranque del sistema	94
4.9.	Grafana + Prometheus	96
4.9.1.	Ejercicio Obligatorio. Monitorización con Grafana + Prometheus	97

Parte I
Apuntes de Clase

Capítulo 1

Bloque 1

1.1. Ping y SSH

Sobre esta parte no decidí tomar apuntes en clase debido a que son conceptos vistos en la Asignatura de Fundamentos de Redes, por lo que consideré que no era necesario. De todas formas en la parte de Prácticas (Resolución) se comenta paso a paso lo que se hizo en esta parte.

1.2. LVM y RAID

1.2.1. LVM (Logical Volume Manager)

- **Discos y particiones:**
 - **sda:** Primer disco del sistema.
 - Luego pueden existir **sdb**, **sdc**, etc.
- **Consideraciones importantes:**
 - Si el directorio **/boot** se llena, el sistema podría fallar al arrancar debido a la falta de espacio disponible.
 - Se recomienda evitar el uso de **swap** en sistemas virtualizados, ya que reduce el rendimiento.

1.2.2. Directorios base en Linux

- **/boot:** Contiene los archivos de arranque del sistema.
- **/etc:** Almacena los archivos de configuración del sistema operativo.
- **/dev:** Contiene archivos especiales que representan dispositivos del sistema.
- **/mnt:** Punto de montaje temporal para sistemas de archivos.
- **/var:** Contiene datos variables del sistema, como logs y archivos temporales. Puede crecer mucho, por lo que se recomienda monitorearlo.

1.2.3. RAID (Redundant Array of Independent Disks)

RAID es una tecnología que permite combinar múltiples discos para mejorar la redundancia y/o el rendimiento del sistema de almacenamiento.

1.2.4. Ventajas

- Permite unir volúmenes de almacenamiento.
- Mejora el rendimiento si los discos están en buses distintos, ya que permite acceso en paralelo.

1.2.5. Niveles de RAID

- **RAID 0 (Striping):**
 - Divide los datos en bloques y los distribuye entre varios discos.
 - **Problema:** Si un disco falla, se pierde toda la información.
 - Se usa poco en la práctica debido a su baja robustez.
- **RAID 1 (Mirroring):**
 - Duplica los datos en dos o más discos.

- Si un disco falla, el otro sigue funcionando con los mismos datos.
 - Aporta robustez al sistema.
 - **Problema:** Se paga el doble en almacenamiento.
 - Se usa frecuentemente en `/boot` para garantizar que el sistema pueda arrancar en caso de fallo de un disco.
- **RAID 5 (Paridad distribuida):**
 - Se distribuyen los datos y la paridad entre todos los discos.
 - Si un disco falla, se pueden recuperar los datos utilizando la paridad.
 - **Problema:** Puede reducir el rendimiento debido al tiempo necesario para calcular la paridad.
 - **Ventaja:** Equilibra costos, robustez y capacidad de recuperación.
 - Se puede usar un disco de repuesto que entra en acción si uno falla.

1.2.6. Tipos de RAID

- **RAID por Hardware:**
 - Utiliza un controlador RAID físico.
 - Es más eficiente y transparente para el sistema operativo.
- **RAID por Software:**
 - Administrado por el sistema operativo.
 - Puede ser modificado por el administrador, lo que representa un riesgo.
 - Requiere más recursos del sistema.

1.2.7. Ejercicio Opcional: Configuración de RAID1 para `/var`

1.2.8. Objetivo

El objetivo es proporcionar a `/var` un respaldo frente a fallos mediante la creación de un RAID1. Para ello, debemos montar un RAID1 y mover `/var` dentro de este.

1.2.9. Pasos a seguir

1. **Creación de discos virtuales en la máquina virtual (MV)**
 - Se crean dos discos: `raid1` y `raid2`.
 - Usamos `lsblk` para verificar la existencia de `sdb` y `sdc`.
2. **Instalación de `mdadm`**
 - `sudo dnf provides mdadm` (para verificar qué paquete lo proporciona).
 - `sudo dnf install mdadm` (para instalarlo).
3. **Creación del RAID1**

- `sudo mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sdb /dev/sdc`
- Aparecerá un *warning* sobre la creación de metadatos, confirmamos con "sí".
- Para monitorear la sincronización del RAID: `watch -n 1 more /proc/mdstat`
- Luego, verificamos con `lsblk`.

4. Prueba de fallo de disco

- Podemos simular la falla de un disco con: `echo 1 > /dev/sdb`
- Se observa que md0 sigue funcionando al ser un *mirror*.

1.2.10. LVM (Logical Volume Manager)

1.2.11. Conceptos Clave

- **LV (Logical Volume)**
- **VG (Volume Group)**
- **PV (Physical Volume)**

La estructura básica de LVM es la siguiente:

```

  LV          LV
  ,           ,
  Volumen Group
  ,
  ,   ,   ,
  pv   pv   pv

```

Un **Logical Volume (LV)** puede expandirse tomando espacio libre de otros discos o estructuras.

1.2.12. Visualización de LVM

- Para ver los discos que son de tipo LVM, usamos `lsblk` en la columna de *type*.
- Para ver opciones de `pv`, `vg` y `lv`, usamos `tab-completion`.

1.2.13. Consideraciones sobre Volumen Groups

Si un **Volume Group** contiene un disco magnético, un SSD y un RAID, el acceso a los datos puede no ser homogéneo. Por ello, el volumen lógico debe montarse sobre el RAID.

1.2.14. Creación del LVM sobre el RAID

1. Convertir el RAID en un Physical Volume: `pvcreate /dev/md0`
2. Crear un Volume Group llamado `raid1`: `vgcreate raid1 /dev/md0`
3. Crear un Logical Volume de 10GB dentro del Volume Group: `lvcreate -L 10G -n rvar raid1`

1.2.15. Movimiento de `/var` al RAID

Después de crear el volumen lógico, debemos mover `/var` dentro del RAID:

- El nombre del volumen dentro del RAID es `/dev/raid1/rvar`.
- También puede aparecer como `/dev/mapper/raid1-rvar`, ya que son sinónimos creados mediante enlaces simbólicos.

1.2.16. Resolución de Problemas

1.2.17. Problema 1: Configurar el sistema de archivos en `rvar`

Actualmente, el volumen lógico `rvar` está vacío y sin un sistema de archivos. Para solucionarlo, seguimos estos pasos:

1. **Seleccionar un sistema de archivos:** Los sistemas de archivos recomendados son `ext4` y `xfs`, ya que son transaccionales y previenen la corrupción de datos en caso de fallos.
2. **Verificar los sistemas de archivos soportados:** Ejecutamos el siguiente comando para listar los sistemas disponibles en el kernel:

```
ls /lib/modules/$(uname -r)/kernel/fs
```

3. **Formatear el volumen lógico con `ext4`:**

```
mkfs.ext4 /dev/mapper/raid1-rvar
```

1.2.18. Problema 2: Montar y trasladar `/var` al nuevo volumen

El volumen lógico `rvar` debe ser montado en el sistema y debemos trasladar `/var` sin perder datos. Para ello:

1. **Montar el volumen lógico:** Primero, montamos `rvar` en un directorio temporal:

```
mount /dev/mapper/raid1-rvar /mnt/
```

Podemos verificar con:

```
mount
```

Al revisar `/mnt/`, aparecerá el directorio `lost+found`, indicando que el sistema de archivos está activo.

2. **Cambiar a modo mantenimiento:** Para evitar la pérdida de datos al copiar `/var`, debemos entrar en el `runlevel1`, también conocido como **modo mantenimiento**. Esto se hace con:

```
systemctl isolate runlevel1.target
```

Podemos confirmar el estado con:

```
systemctl status
```

3. **Copiar los datos de `/var` a `/mnt/`:** Copiamos todo el contenido de `/var` manteniendo atributos con:

```
cp -a /var/* /mnt/
```

4. **Crear un respaldo de `/var`:** Antes de reemplazar `/var`, hacemos una copia de seguridad por si algo falla:

```
mv /var /var_old
```

5. **Desmontar `/mnt` y montar el nuevo `/var`:**

```
umount /mnt  
mkdir /var  
mount /dev/mapper/raid1-rvar /var
```

Podemos verificar con:

```
df -h
```

6. **Hacer el montaje permanente en `/etc/fstab`:** Si reiniciamos ahora, la configuración se perdería. Para evitarlo, editamos el archivo `/etc/fstab` y agregamos la siguiente línea:

```
/dev/mapper/raid1-rvar /var ext4 defaults 0 0
```

7. Probar la configuración antes de reiniciar:

```
mount -a
systemctl daemon-reload
mount
```

Si todo está correcto, reiniciamos el sistema.

1.2.19. Interpretación de lsblk

El comando `lsblk` nos permite visualizar la estructura de almacenamiento del sistema. Es importante identificar:

- **Discos físicos y particiones:** Aparecen como `sda`, `sdb`, `sdc`, etc.
- **Volúmenes lógicos:** Se muestran bajo `/dev/mapper/`.
- **SR0:** Indica la unidad de CD-ROM.

1.3. Firewall + SSHD

Ya sabemos de otras Asignaturas que un **firewall** es un sistema que controla el tráfico de red, permitiendo o bloqueando ciertas conexiones. En Linux, el firewall más común es **iptables**.

El comando que tenemos que aprender a gestionar es `firewall-cmd`, tiene diversas opciones como `status`, `state`. Otra forma para ver si está arrancado es `systemctl status firewall`.

Otras opciones: `firewall-cmd --list-all`, en este comando debemos de ver los filtros que están definidos, en `services` podemos ver los servicios que están habilitados.

Con `sudo firewall-cmd --add-service=http`, cuando lo listo otra vez, se añade el puerto `http` en la opción `services`. Cuando se hace con `firewall-cmd`, se hace dinámica, por lo que si reiniciamos esto se pierde. Para que se quede debemos de ejecutar el comando `sudo fire firewall-cmd --runtime-to-permanent`. De forma alternativa, podemos ejecutar primero la opción `--permanent` y luego `--add-port=443/http`, pero si lo listamos no lo lleva a memoria, no hay ninguna opción que lo deje permanente y que lo lleve a memoria. Tenemos dos opciones o trabajar con servicios o llevarlo a memoria.

Si queremos conocer los servicios, podemos ejecutar `firewall-cmd --get-services`

Los firewalls en Linux son muy eficientes debido a que en este SO se trabaja con servicios, por lo que se puede controlar el tráfico de forma muy precisa. Al implementarse a nivel de Kernel ayuda a que sea muy eficiente. *No se va a preguntar nada sobre iptables.*

Debemos de usar el programa `nmap` ya que nos dice que puertos están abiertos en un servidor. Para instalarlo, usamos `sudo dnf install nmap`. En el guión debemos de mirar la referencia número 31.

1.3.1. Ejercicio Opcional

Debemos de instalar un servicio HTTP, se recomienda Apache. Podemos escanear los 100 puertos más usados con el comando `nmap -F localhost`. Para instalar Apache, usamos `sudo dnf install httpd`. Para arrancar el servicio, usamos `sudo systemctl start httpd`. Para comprobar que está arrancado, usamos `sudo systemctl status httpd`. Para que arranque en el inicio, usamos `sudo systemctl enable httpd`. Para comprobar que está escuchando en el puerto 80, usamos `sudo ss -tulnp | grep 80`.

Debemos de poder acceder desde el navegador de la máquina anfitrión.

1.3.2. SSH

Es un programa de terminal remoto, ya lo hemos usado en Asignaturas anteriormente. Antes se usaba Telnet, en este se especifica la dirección IP y el puerto, pero no era seguro, por lo que cualquier *man in the middle* podía interceptarlo y suplantar la identidad. Además, todas las respuestas que se producían eran en abierto, los `cat`,... En cambio, SSH es seguro, ya que se cifra la información. Este hace lo mismo que Telnet, pero de forma segura. Se recomienda hacerlo `ssh usuario@ip`. O bien sustituir la IP por el nombre del dominio. Para cerrar la conexión, usamos `exit`.

Ya hemos estudiado en otras Asignaturas el tema de criptografía en cuanto al cifrado simétrico y asimétrico. En especial, en Fundamentos de Redes. Por si acaso, vamos a hacer una pequeña introducción. Tenemos dos tipos de cifrado:

- **Cifrado simétrico:** Se usa una clave para cifrar y descifrar. El problema es que si alguien intercepta la clave, puede descifrar todo. Por ello, se usa poco.
- **Cifrado asimétrico:** Se usa una clave pública para cifrar y una privada para descifrar. La clave pública se puede compartir, pero la privada no. Se usa para firmar documentos, ya que si se cifra con la clave privada, solo se puede descifrar con la clave pública.

Debemos de destacar los conceptos de **autenticación** y **autorización**. La autenticación es el proceso de verificar la identidad de un usuario, mientras que la autorización es el proceso de verificar si un usuario tiene permiso para acceder a un recurso. Además, del uso de certificados digitales, que son un tipo de credencial que se usa para autenticar la identidad de un usuario o un servidor. Este se consigue a través de una entidad certificadora, usando una clave pública y privada (Big Brother).

Algoritmos :

- Llave simétrica: DES (tenemos dos llaves, una que es privada y otra que es pública, lo que se cifra con la privada solo se puede descifrar con la pública, por dentro se gestiona en base a números primos muy grandes).
- Llave asimétrica: RSA (Se guarda la privada, mientras que la pública se comparte, estando incluso compartida en la propia web, de manera que la persona que quiera compartir contigo usa la pública para codificar el texto y te lo manda codificado, de esta manera si hay un man in the middle lo ve cifrado, solo la persona con la llave privada puede ver su contenido).

Otro uso de la llave privada es la *firma*, de esta manera se garantiza la confidencialidad y la autenticación del mensaje debido a que es confidencial debido a que solo se puede descifrar con la llave pública, y autenticidad debido a que solo se puede cifrar con la llave privada, la cual solo conoce esa entidad.

Firma Digital: Usando Hash SHA256, se cifra con la llave privada y se envía el mensaje cifrado y el mensaje en abierto. La persona que recibe el mensaje cifrado con la llave privada, lo descifra y lo compara con el mensaje en abierto, si son iguales, se garantiza la autenticidad del mensaje. Ejemplo de ello puede ser el minado de monedas como es el Bitcoin. Además, otro ejemplo de ello es cuando descargamos algo y podemos verificar que es el del propio distribuidor usando el Hash que me proporciona. *Por ende, podemos considerar que un HASH + llave privada es el certificado digital.* Para asegurarnos de que la llave pública es de quien dice ser debemos de ver el certificado digital, como hemos mencionado anteriormente. Este proceso para cuando encontramos un certificado digital que es de confianza, ya que si no lo es, no podemos asegurar que la llave pública sea de quien dice ser. Para ver los certificados, vemos que en Firefox, accedemos a settings y buscamos *certificados*, podemos ver las entidades que nuestro browser reconoce.

Advertencia

El tema de certificados y demás al ser materia de Fundamentos de Redes, no se va a preguntar en el examen.

Nos conectamos a un ordenador remoto mediante ssh, de esta manera le decimos que nos queremos conectar, el te envía la llave pública, es decir, se va a su configuración, recupera la llave y te la envía. A continuación, le enviamos nuestras credenciales cifradas con la llave pública, y cuando le llega, las descifra con la llave privada y comprueba si son correctas. Si lo son, nos deja entrar. Si no lo son, nos dice que no podemos entrar. Además, de esta manera se previene que un *man-in-the-middle* no pueda interceptar las credenciales.

Podemos meterle una entrada en el fichero de host, de esta manera cuando nos conectamos a un servidor, nos conectamos a una IP, pero si le metemos una entrada en el fichero de host, le decimos que esa IP es un nombre, de esta manera cuando nos conectamos a ese nombre, nos conectamos a esa IP. Para ello, debemos de editar el fichero `/etc/hosts` y añadir la IP y el nombre del servidor, así es más cómodo, para ello en Linux es `ssh usuario@nombre`.

Cuando ejecutamos el comando para conectarnos, nos imprime un mensaje con el HASH, y nos pregunta que si nos lo creemos, si en vez de una llave pública esta reconocida en nuestras llaves y ve que tenemos el certificado no nos preguntaría. En todos los equipos se crea el directorio ssh en home, dentro de este tenemos un archivo que se llama `known_hosts`, el cual contiene las llaves públicas de los ordenadores que ya has confiado, de manera que no nos va a preguntar cuando nos conectemos de nuevo, si queremos que en la primera vez tampoco pregunta la copiamos y pegamos, entre otras opciones. *Nota: Por defecto se crea ese directorio cuando se conecta por ssh.*

Este proceso es muy costoso en términos de cómputo, por lo que se usa lo más mínimo posible. Por esto se usa un proceso de handshaking, de esta manera el equipo le manda una propuesta de llave simétrica y de esta manera ya se usa en adelante, ya que el uso de llave privada y pública es muy costoso.

¿Cómo puedo acceder sin contraseña? Para ello nos autenticamos con llave pública y llave privada. En el caso de que decidamos hacerlo así cuando le mandamos que soy 'nombre', recupera la llave de el directorio `.ssh` y le manda un mensaje de firmame esto con la llave privada de dicho usuario, y de esta manera se asegura de que efectivamente es el usuario que dice ser. Además, en el directorio home solo puede escribir el usuario y el root.

En la MV:

- `ssh-keygen`: crea la llave privada dentro de home, luego nos pregunta si queremos protegerla con contraseña, si no queremos, le damos a enter, *pero siempre debe de hacerse*. Luego si nos vamos al directorio vemos dos documentos la llave privada y la pública. Si queremos que no nos pregunte siempre la contraseña, usamos el comando `ssh-agent`, de esta manera el la escribe por ti, hasta que usamos el comando `exit`, en este caso se olvida.

Para usar la pública:

- Accedemos al directorio `/home/.ssh`

- usamos el comando `ssh-copy-id usuario@ip/hostname` (copia mi identidad, se puede hacer con `ctrl+c/v`), y luego nos hace una prueba para que le demos que de verdad somos el usuario al que nos queremos conectar, para ello le mandamos la contraseña.
- Luego comprobamos que podemos entrar sin contraseña.
- En la máquina donde nos hemos conectado, vemos un fichero llamado `id_rsa.pub`, y si lo listamos debe de ser el mismo que la clave pública de la máquina inicial, desde la que nos conectamos a esta.

Nota: El directorio `.ssh` se crea en el home cuando es el cliente, mientras que en el servidor se crea en el `/etc/ssh`, también podemos crearlo nosotros a priori.

Ahora podemos ejecutar un comando remoto, por ejemplo: `ssh usuario@ip comando`, de esta manera se ejecuta el comando en la máquina remota. En este punto comienza la automatización de configuración remota. *Ansible son scripts pero ejecutados de manera remota.* Además, podemos copiar archivos usando el comando `scp` (secure copy), de esta manera se copia el archivo de una máquina a otra. *Nota: `scp` es un comando de `ssh`.* El comando es `scp origen destino`, por ejemplo: `scp /home/usuario/archivo usuario@ip:/home/usuario/`, de esta manera se copia el archivo de la máquina local a la remota. Si queremos copiar un directorio, usamos el comando `scp -r origen destino`. Además podemos conectarnos mediante `sftp` (secure file transfer protocol), de esta manera se conecta a la máquina remota y podemos copiar archivos de manera interactiva. *Nota: `sftp` es un comando de `ssh`.* Para conectarnos, usamos el comando `sftp usuario@ip`, y luego podemos usar los comandos `put` y `get` para copiar archivos de una máquina a otra.

1.3.3. Ejercicio Opcional

En el ejercicio debemos de dar acceso mediante `ssh` usando la llave pública y privada, debemos de modificar el puerto para conectarnos en vez del 22 para el `ssh` usar otro que sea mayor que el 1024, para ello podemos asegurarnos de que no estamos colisionando con un puerto conocido buscando en `Innernet`.

1. Accedemos al contenido del demonio del `ssh` que se encuentra en `/etc/ssh`, una vez dentro nos pone el comando que debemos de ejecutar `semanage...` (aparece en el fichero de `ssh` de la máquina Rocky), para que se apliquen los cambios debemos de reiniciar el servicio, para ello usamos el comando `sudo systemctl restart sshd` o bien reiniciar la máquina.
2. Además debemos de modificar el firewall para permitir ese puerto, para ello usamos el comando `sudo firewall-cmd --add-port=puerto/tcp --permanent`, y luego reiniciamos el servicio con `sudo systemctl restart firewallld`.
3. Luego debemos de copiar la llave pública a la máquina remota, para ello usamos el comando `ssh-copy-id usuario@ip`, y luego nos pide la contraseña, una vez introducida, ya podemos conectarnos sin contraseña.
4. Para comprobar que todo está correcto, usamos el comando `ssh usuario@ip -p puerto`, de esta manera nos conectamos a la máquina remota.

Con el parámetro `-p` le especificamos el puerto al que nos queremos conectar, de esta manera nos conectamos al puerto que hemos especificado, ya que de manera predeterminada se conecta al puerto 22.

1.4. Automatización con Ansible

Por defecto Ansible ya viene instalado en la mayoría de las distribuciones de Linux. Además, por defecto al instalar Ansible se instala Python y SSH. Para instalarlo debemos de usar el comando `sudo dnf install ansible`. Para comprobar que está instalado, usamos el comando `ansible --version`, o bien mediante el comando `sudo dnf list installed | grep -i ansible`. Además, podemos ver la documentación de Ansible en la página oficial de Ansible. Su configuración se encuentra en `/etc/ansible`, *este solo se instala en el controlador ya que es desde donde se maneja*.

En la configuración de `/etc/ansible`, podemos cambiar varios parámetros, como es el caso del número de hebras que usa o número de procesos que se ejecutan de manera paralela, en este caso suele estar por defecto en 5 hebras.

En nuestro directorio base¹, tenemos el archivo `.ansible.cfg`, el cual podemos editar siendo los cambios aplicados de la misma manera que si lo hiciéramos en el archivo de configuración de `/etc/ansible`. Por defecto viene vacío, pero tiene un comando para generar la configuración por defecto, el cual es `ansible-config dump`. Además, podemos ver la configuración de Ansible con el comando `ansible-config list`.

En cuanto al formato que se usa en Ansible, cabe destacar que se establecen etiquetas, como puede ser `web server`, de manera que cuando ejecutemos Ansible, podemos elegir esta etiqueta para que lo lance con lo que esté definido debajo de dicha etiqueta en el archivo de configuración.

Entramos en ansible con `cd .ansible`, y debemos de definir etiquetas para entrar a los hosts que se definen en este archivo de configuración. En este archivo se definen los hosts.

Una de las referencias importante es la guía sobre AD-HOC y la lista de todos los módulos que podemos ejecutar². Los módulos que vienen por defecto son los *ansible.builtin*. El ping de Ansible, actúa de manera distinta, lo que hace es que comprueba si se tiene acceso al servidor al que se hace el ping y comprueba si este tiene python instalado. Solo se le pasa un parámetro.

Para ejecutar un comando AD-HOC³ es con `ansible ansible_control -m ping`, este comando debe darnos correcto si efectivamente puede ejecutar el comando de manera correcta. Cabe destacar que `ansible_control` debe de estar en la configuración de ansible. Otro de los comandos a destacar es: `ansible ansible_control -m ping -a 'data = "Hola Mundo"'`, de nuevo **cabe destacar que el comando `ansi_control` esta definido en la configuración de ansible**.

En clase estamos ejecutando comando con lo anteriormente mencionada, podemos destacar que si intentamos ejecutar comandos de sudo da error, para ello debemos de añadir `--become`, pero nos pide la contraseña, esto no nos interesa ya que no queremos que nos pida la contraseña todas las máquinas.

¹Accedemos con el comando `cd`.

²Referencias 42 y 43 del guión.

³Solo disponible en la controladora y no en las manejadas.

Para ello nos vamos al fichero `/etc/` y ejecutamos `more sudo`. en `.ansible` las frases con `%` delante es un grupo, vemos una línea que nos lo permite para todos (todos los wheels), pero esto es *una mala práctica*.

Así que en vez de eso, añadimos la línea `david ALL=(ALL)NOPASSWD:ALL`, debo de salir y volver a entrar, porque puede recordar sesiones anteriores de `sudo`. En este punto si nos deja entrar usando `--become` sin poner la contraseña.

Con el comando `ansible ansible-inventory --graph` vemos las máquinas que tenemos bien podemos ejecutar `ansible ansible-inventory -i <ruta fichero> --graph`.

Para más información sobre Ansible, podemos visitar la documentación oficial en <https://docs.ansible.com>. Aquí encontraremos guías, referencias y ejemplos que nos ayudarán a entender y utilizar Ansible de manera efectiva.

O bien acceder a esta url: <https://docs.ansible.com/ansible/latest/index.html> y sobretodo a esta https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html.

Para ejecutar un archivo `.yaml` es `ansible -playbook -i <ruta fichero> <archivos extra>`.

El fichero `basics.yaml` que se usa en clase como ejemplo es:

```
1 servers:
2   hosts:
3     ansi_control:
4       ansible_host: 192.168.56.101
5       ansible_user: david
6       ansible_port: 22232
7     ansi01:
8       ansible_host: 192.168.56.111
9       ansible_user: admin
10    ansi02:
11      ansible_host: 192.168.56.112
12      ansible_user: admin
13  vars:
14    ansible_port: 22
15
16 unlagged:
17   hosts:
18     ansi01:
19     ansi02:
```

Debemos de consultar las claves especiales en la documentación. La forma de depurar es comentar todo e ir ejecutando líneas y descomentando.

Cabe la posibilidad de hacer playbooks parametrizables.

Uno parte de los playbooks que se usan en clase es:

```
1 - name: Esta el usuario alguien ahí?
2   ansible.builtin.shell:
3     cmd: who
4   register: who_results (guarda el resultado del comando who en la
5     variable who_results)
6
7 - name: Print Connected
8   ansible.builtin.debug:
9     msg: "Connected list: {{ who_results.stdout }}"
```

Tenemos que entregar un directorio con los ficheros⁴:

- basics.yaml
- ejecutaPlaybook.sh
- hosts.yaml
- vars.yaml: variables.

1.4.1. Ejercicio Obligatorio

En cuanto al ejercicio que debemos de **entregar**, el cambio que debemos de hacer es en `/etc/ssh/.conf`, ahí cambiamos la línea de `AccessRoot` y lo ponemos con el atributo `yes`.

Los pasos son⁵:

1. Crear un nuevo usuario llamado “admin” que pueda ejecutar comandos privilegiados sin contraseña. *Añadir usuario en las tasks de ansible y lo añadimos a sudoers.*
2. Dar acceso por SSH al usuario “admin” con llave pública. *Debemos de enviar la llave pública, buscando un módulo de ansible que lo hace.*
3. Crear el grupo “wheel” (si no existe) y permitir a sus miembros ejecutar sudo. *Existe, porque usamos Rocky, asumimos que no sabemos que máquina es.* Si ejecutamos el mismo comando varias veces con los mismos parámetros el resultado final siempre es el mismo.
4. Añadir una lista variable de usuarios (se proporcionará un ejemplo con al menos dos), añadiéndolos al grupo “wheel” y concediéndoles acceso por SSH con llave pública. La sacamos en el directorio `.ssh` y en el fichero `id_pub`.
5. Deshabilitar el acceso por contraseña sobre SSH para el usuario root.

Estos quedan con un usuario admin que puede ejecutar comandos privilegiados sin contraseña.

Con esto podemos pasar el 2º `playbook`.

Debemos de realizar el ejercicio correspondiente a la creación de la web e instalación del servidor web

Un ejercicio interesante para comenzar es con `ansible ansi_control -a "poweroff" --become`, ya que solo se apaga si esta todo correcto.

Para depurarlo si lo rechaza, debemos de:

1. Me conecto con ssh.
2. Probamos haciendo `poweroff` con ssh, entre otras opciones.

⁴Básicamente todos los que usemos.

⁵Todo es dentro del `playbook`.

Capítulo 2

Bloque 2

2.1. Breve introducción a Docker y contenedores

Lo que proporcionan los contenedores es un entorno seguro donde nuestra máquina puede correr programas.

Podemos lanzar *namespace* y este nos lanza un entorno en el que nos manda a nuestra raíz, de manera que tenemos un filesystem para nosotros. De esta forma tenemos un filesystem aislado. Si ejecutamos `ps -ax` vemos que no vemos los procesos de la máquina anfitriona.

La idea principal de las redes en los contenedores es que se puedan comunicar entre sí.

En resumen, los contenedores nos dan un espacio aislado y virtualizado. Hay un poco de pérdida de las prestaciones cuando se ejecuta el servicio de red virtualizado, pero es muy poco. Lo más importante es que el acceso a memoria, CPU, ... son las mismas.

La principal utilización de los contenedores es porque te ofrece un espacio de *safe box*, y un espacio importante que es el *filesystem privado*.

Cuando se ejecuta en otros espacios se arrastran todas las dependencias.

Se puede extraer tanto la arquitectura como el sistema operativo, dependencias y demás.

Si necesito un analizador de red, puedo instalarlo sobre un docker y de esta manera no me ensucia mi espacio de trabajo y cuando necesito espacio puedo borrarlo.

En el caso de linux, todos los dockers son sistemas más pequeños sobre linux, con lo básico para que arranque.

Las aplicaciones que ejecutamos en el docker deben de ser ajustadas para que se ejecuten, debido a que la tecnología de docker es diferente a la de la máquina anfitriona.

Debemos de adaptarlo para que pregunte al contenedor por el límite de aplicaciones que pueden ejecutar, para que todo se ejecute de la manera correcta.

Instalación de Docker

Están desarrolladas para Linux, por lo que no se puede correr en tecnologías de Windows, si se puede con las extensiones de Linux para Windows.

Usamos la guía de CentOS para instalarlo en Rocky, debemos de ejercicio Obligatorio. Monitorización con Grafana + Prometheus.tener cuidado con la guía ya que

en el último paso, aparecen pasos opcionales para que el docker sea ejecutado sin ser root (privilegios de superusuario).

Podemos instalar el DockerDesktop para un uso más sencillo, pero no es recomendable para un uso profesional.

Se puede ejecutar sobre MacOS pero usando una máquina virtual, lo cual no tiene mucho sentido.

Entramos en `docker-hub.com`, y buscamos el contenedor de `hello-world`, ya que es el inicial para ver que todo funciona correctamente. Para ello debemos de ejecutar el comando `docker pull hello-world`. Si no indicamos el número de versión, se instala la latest(última versión).

Lo ejecutamos con el comando `docker run hello-world`. Una vez que ejecutamos el comando, se descarga la imagen y se ejecuta el contenedor, mostrando un mensaje para ver que todo ha funcionado correctamente.

Todos los ejercicios son opcionales.

A continuación debemos de centrarnos en el punto de OpenBenchmarking, que es un benchmarking de docker.

El de `blender` es muy utilizado por marcas más comerciales para la renderización de imágenes y vídeos.

Debemos de tener en cuenta siempre las dependencias que necesitamos para ejecutar el contenedor.

Hay una aplicación que se llama `Phoronix`, que es un benchmarking de docker, pero para el Departamento de ISE, están pensando en que esta en desuso.

Así que vamos a trabajar con `Phoronix` en un contenedor para que sea más sencillo de instalar y de ejecutar.

En la guía si buscamos `Phoronix`, bajamos el de `pts`. Para ello ejecutamos el comando `docker pull phoronix/pts`.

Con el comando `docker images` vemos que se ha descargado la imagen.

SI ejecutamos el comando `docker run -it phoronix/pts` (usamos el comando `-it` para que se ejecute en el primer plano y os dé una shell interactiva). Con el comando `system-info`, podemos ver las información de los cores, incluso de los que están el un contenedor.

Con el comando `system-sensors`, podemos ver los sensores que el ordenador tiene de manera nativa.

El comando `phoronix-test-suite`(no es que sea un comando es lo que se muestra en la shell cuando ejecutamos el comando anterior de `run`), y si de seguido añadimos `list-available-tests`, podemos ver los test que podemos ejecutar. Para más info podemos acceder a la web de docker y en el apartado de documentación.

Apache Benchmark

Sirve para servidores http. Usando el comando `ab` nos sale la información sobre el mismo. Algunos de los parámetros pueden ser:

- `-n`: Número de peticiones.
- `-c`: Número de conexiones concurrentes.
- ...

Si lo hacemos con la `ugr`, podemos ver que en las características que el documento html de la web aparece con muy pocos bytes. Probamos a ejecutar `curl -v http://www.ugr.es`. En específico, `curl` se usa para hacer peticiones http y depurar las mismas. *Debemos de aprender a usarla.*

Vemos que la web de la `ugr` esta sobre https, por lo que `apache-benchmark` no se ha dado cuenta, por lo que debemos de usar el comando `ab -n 20 -c 4 https://www.ugr.es/` (debemos de incluir la barra del final) y vemos que en este caso los bytes del html es mucho mayor y es más realista (previamente hemos ejecutado el comando `curl -v http://www.ugr.es/`).

Simulación de Carga con Jmeter

Se menciona como una tecnología a conocer en las entrevistas de trabajo, por lo que es importante conocerla.

Usa Java, por lo que podemos usarlo con la última versión de Java.

La lanzamos en 2^o plano para que no consuma la consola. Para ello ejecutamos el comando `jmeter &`.

1. Añadimos un elemento de thread, que corresponde con las hebras que se van a lanzar de manera concurrente. Para ello pulsamos add y seleccionamos thread group. Añadimos un nombre, hay parámetros que son importantes. *Se recomienda poner el Jmeter en Inglés para que no haya problemas con los nombres de los elementos y para que las guías de ayuda no tengan errores.*
2. Dentro del grupo de hebras creamos un *sample*. Una de las ventajas de Apache es la gran inmensidad de plugins que tiene. Dentro de sample seleccionamos la *carga de http*. Gracias a la opción de *follow redirects* hace que se ejecute redirecciones de manera automática.
3. Configuramos los parámetros, como es el protocolo que vamos a usar, puerto, ...
4. Añadimos un *listener* para poder ver los resultados (View Results Tree). Para desactivar un elemento debemos de hacer clic derecho y seleccionar la opción "disable". Hay plugins para poder ver los resultados de manera más visual.

Si queremos simular que se diriga a una página secundaria (`ugr.es/centros`), debemos de añadir el nombre de la página en el campo de *path*. Podemos configurar el elemento común de *HTTP defaults*.

Podemos usar variables de usuario. Podemos cargar el archivo que nos da bien con la extensión `.jtl` y usamos `jmeter` o con un excel (ya que es un archivo en formato csv).

En cuanto a la aplicación de `jmeter`, debemos de tener en cuenta que en el orden que aparezcan los elementos es el orden en el que se van a ejecutar.

Dentro de `/var/logs`, tenemos el archivo que muestra los logs y tiene una línea por cada usuario.

El usar el *not sampler* es la forma más habitual de hacerlo, ya que es la forma más sencilla de hacerlo.

Instalación de la aplicación para el test con JMeter

La aplicación que utilizaremos para el ejercicio de prueba de carga se encuentra en el repositorio de GitHub: <https://github.com/davidPalomar-ugr/iseP4JMeter.git>.

Puede clonar el repositorio utilizando GIT o descargarlo como un archivo comprimido. Una vez descargado, obtendrá un nuevo directorio llamado `iseP4JMeter`, al cual podrá acceder y levantar la aplicación con los siguientes comandos:

```
cd iseP4JMeter
docker compose up
```

El servicio se lanza en primer plano mostrando los logs de ejecución de sus componentes (incluidas las llamadas HTTP), lo que puede ser muy útil para depurar incidencias. Si desea que el servicio continúe en segundo plano, puede ejecutar el comando con la opción `-d`:

```
docker compose up -d
```

Detalles sobre el Dockerfile y Docker Compose

El `Dockerfile` es el archivo donde se especifican todos los comandos necesarios para crear una imagen de Docker y las acciones que debe realizar sobre esta (como copiar archivos, instalar paquetes o modificar configuraciones). Desde un punto de vista abstracto, podríamos decir que es el *playbook* que Docker aplica al levantar el contenedor.

En este caso, la aplicación sobre la que aplicaremos la carga consta de dos contenedores configurados con sus respectivos `Dockerfiles`:

Dockerfile de la aplicación (Node.js):

```
FROM node:16.13.0-stretch
RUN mkdir -p /usr/src/app
COPY . /usr/src/app
EXPOSE 3000
WORKDIR /usr/src/app
RUN ["npm", "install"]
ENV NODE_ENV=production
CMD ["npm", "start"]
```

En este archivo se especifica:

- La imagen base (`node:16.13.0-stretch`), que incluye Node.js versión 16.13.0 y Debian Stretch como sistema operativo base.
- La creación de un directorio y la copia de los archivos del repositorio en el contenedor.
- La exposición del puerto 3000 para la aplicación.
- La instalación de las dependencias con `npm install`.
- La ejecución de la aplicación con `npm start`.

Dockerfile de la base de datos (MongoDB):

```
FROM mongo:6
COPY ./scripts/* /tmp/
RUN chmod 755 /tmp/initializeMongoDB.sh
WORKDIR /tmp
CMD ./initializeMongoDB.sh
```

En este archivo se especifica:

- La imagen base (`mongo:6`).
- La copia de scripts al contenedor y la asignación de permisos de ejecución.
- La ejecución de un script para inicializar la base de datos.

Archivo `docker-compose.yml`:

El archivo `docker-compose.yml` permite configurar varios contenedores para que trabajen juntos como un único servicio. Su contenido es el siguiente:

```
version: '2.0'
services:
  mongodb:
    image: mongo:6
    ports:
      - "27017:27017"
  mongodbinit:
    build: ./mongodb
    links:
      - mongodb
  nodejs:
    build: ./nodejs
    ports:
      - "3000:3000"
    links:
      - mongodb
```

Este archivo define:

- El servicio de MongoDB, exponiendo el puerto 27017.
- La inicialización de MongoDB con datos mediante un contenedor adicional.
- El servicio de la aplicación Node.js, exponiendo el puerto 3000 y enlazándolo con MongoDB.

Para más detalles, consulte la documentación oficial de Docker Compose.

Ejercicio Obligatorio: Simulación de Carga HTTP con JMeter

El repositorio <https://github.com/davidPalomar-ugr/iseP4JMeter.git> contiene la aplicación objeto de la prueba de carga y una descripción de los requerimientos sobre la carga a simular. Siga las instrucciones del repositorio y elabore la prueba de carga con JMeter.

El ejercicio debe realizarse en un directorio que contenga todos los artefactos necesarios para la ejecución de la prueba de carga. Los *paths* a los archivos deben definirse de forma relativa para garantizar que la ejecución sea independiente de la ubicación del directorio.

Como validación final, debe ser capaz de ejecutar la prueba de carga desde la línea de comandos (sin interfaz gráfica) desde cualquier directorio de su equipo.

Dentro de la web donde se encuentra la información del ejercicio se nos va indicando lo que debemos de ir haciendo.

Cuando usamos la API, debemos de tener cuidado con los espacios para evitar errores, además cada alumno solo puede consultar su propio expediente.

Nos vamos a donde hemos clonado el repositorio y ejecutamos el comando `./pruebaEntorno.sh`.

Para los caracteres extraños que no codifica adecuadamente las webs, podemos usar webs como `urlencoder` para ver como sería codificado, por ejemplo “_” es “%40”

Cuando ejecutmos el comando anterior de `./pruebaEntorno.sh`, se lista varios elementos como es el token que están separados mediante “.”.

Usando webs como JWT Decoder, si copiamos y pegamos el token aquí, podemos ver la información que contiene. Si mantenemos el ratón en el tiempo de expiración, podemos ver la fecha y hora de expiración.

Como no se proporciona seguridad, en términos de que cualquiera puede interceptar el token, se nos da un número que representa al usuario en vez del correo (el token si está firmado con la llave), es decir, no esta cifrado pero sí firmado, por ende, podemos ver si lo ha firmado una entidad de confianza.

En cuanto al uso de la herramienta aleatoria, lo relacionado con esto no será materia evaluable.

1. Debemos de crear en Jmeter:

- Thread Group.
- Alumnos.
 - Dentro de este debemos de crear un HTTP Request y añadirle login y passwd (será de tipo “POST”), en cuanto al valor que debemos de añadir es \$login y \$passwd.
 - Creamos un *CSV Data Set Config* para poder cargar los datos de los alumnos.
 - Ahora añadimos un *Post Prodesador*, y este solamente afecta al login.
 - Luego, añadimos un archivo “Extraer JWT”. Cuando editamos el archivo, debemos de añadir la variable correspondiente y el campo de valor pasarle el token: “\$TOKEN”.

2. Creamos las variables que se indican con TestPlan.

Básicamente debemos de aprender la documentación para poder realizar la práctica.

Los documentos que debemos de tener según las clases de teoría son:

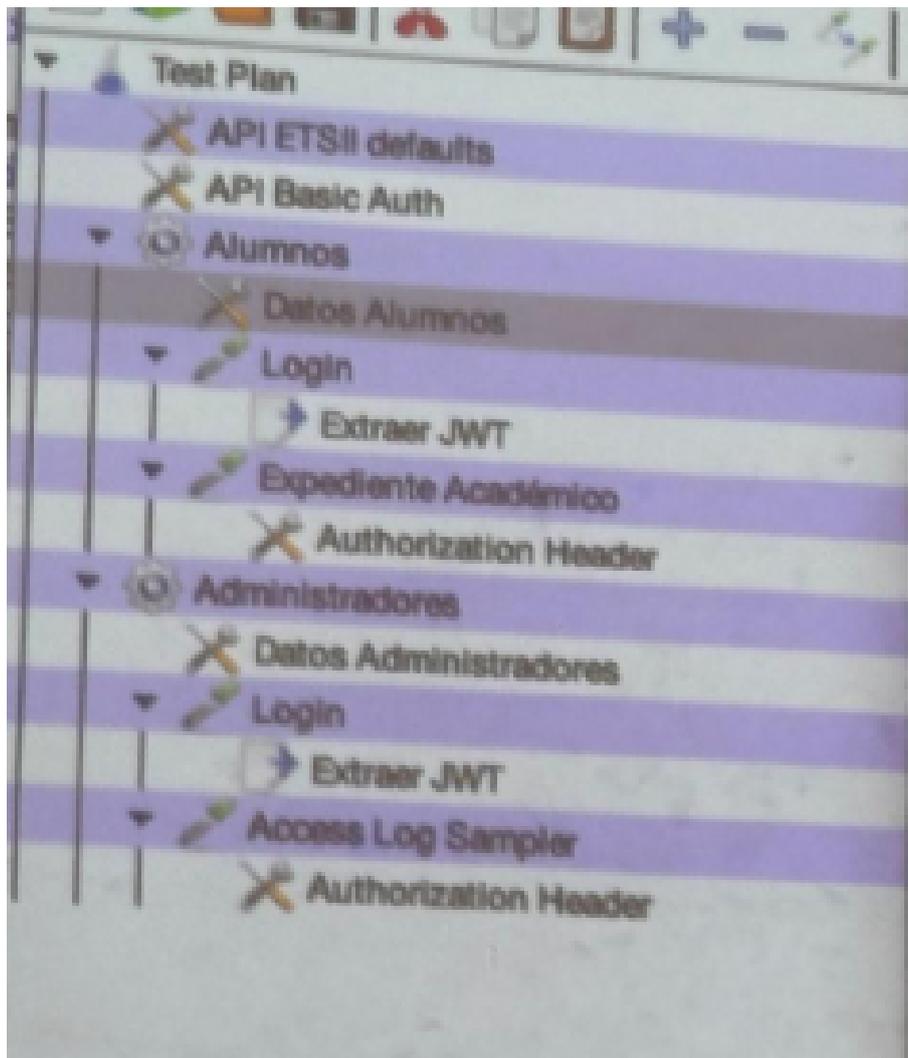


Figura 2.1: Documentos necesarios para la práctica.

Debemos de entregar en Prado los archivos de carga ya que lo demás lo tiene él.

Docker Compose

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones Docker de múltiples contenedores. Con Compose, puedes usar un archivo YAML para configurar los servicios de tu aplicación, y luego con un solo comando puedes crear e iniciar todos los servicios desde esa configuración, este lee nuestro archivo `docker-compose.yml` y levanta los contenedores según la configuración especificada.

Entramos en hub.docker.com y buscamos `mongo:6` la imagen es un cuadrado¹. Ya solo queda entrar en el docker y ejecutar ese contenedor de manera independiente haciendo uso del comando `docker run -it mongo:6`.

En el Docker Compose no necesitamos el mapeado de puertos.

Si entramos en el directorio `mongodb` y abrimos el archivo `dockerfile`² y lo eje-

¹El link es <https://hub.docker.com/r/litmuschaos/mongo>.

²Es similar a un `makefile`.

cutamos, se nos crea un contenedor de mongo, vemos que ejecuta el archivo `init.db`, inicializando la base de datos con los datos que esten ahí definidos. Una imagen se basa en otra y así hasta que al final se llega a la imagen de linux.

2.2. Monitoring

Top

Para ejecutarlo en nuestra máquina debemos de ejecutar el comando `top`. Este comando nos muestra los procesos que se están ejecutando en la máquina, junto con información sobre el uso de CPU, memoria y otros recursos.

Si ejecutamos el comando `more loadavg`, nos ofrece en cada momentos el valor de los programas, tiempo de ejecución, etc.

El comando `wath -n <tiempo> <"comando">`, de esta manera nos ejecuta el comando que se expone cada `<tiempo>` segundos.

Si vemos en una máquina con un solo core vemos un 1, quiere decir, que siempre había un proceso esperado para ejecutarse, es decir, que la CPU esta saturada, el uso de la CPU es del 200 %.

Si tenemos un 1 y 2 cores, significa que la CPU está al 50 %, es decir, que no está saturada.

Si tenemos 4 cores y tenemos un 4, significa que la CPU está al 100 %.

Información de Top en Linux

El comando `top` en Linux muestra información en tiempo real sobre los procesos en ejecución y el estado del sistema. A continuación, se explica el significado de cada elemento que aparece en su salida:

Encabezado del sistema (primeras líneas)

Este bloque muestra información general del sistema.

Primera línea: Tiempos de actividad

```
top - 16:35:28 up 2:15,  2 users,  load average: 1.23, 0.85, 0.67
```

- **16:35:28**: Hora actual del sistema.
- **up 2:15**: Tiempo que el sistema lleva encendido (en este caso, 2 horas y 15 minutos).
- **2 users**: Número de usuarios conectados.
- **load average: 1.23, 0.85, 0.67**: Promedio de carga del sistema en los últimos 1, 5 y 15 minutos. Valores cercanos o superiores al número de núcleos indican alta carga.

Segunda línea: Tareas en ejecución

Tasks: 180 total, 2 running, 177 sleeping, 1 stopped, 0 zombie

- **180 total**: Número total de procesos.
- **2 running**: Procesos que están ejecutándose activamente.
- **177 sleeping**: Procesos en espera de eventos.
- **1 stopped**: Procesos detenidos (pausados).
- **0 zombie**: Procesos "zombie"(terminados pero aún en la tabla de procesos).

Tercera línea: Uso de CPU

%Cpu(s): 3.5 us, 1.2 sy, 0.0 ni, 95.0 id, 0.2 wa, 0.0 hi, 0.1 si, 0.0 st

- **us (user)**: % de CPU utilizado por procesos de usuario.
- **sy (system)**: % de CPU utilizado por procesos del sistema (kernel).
- **ni (nice)**: % de CPU consumido por procesos con prioridad ajustada (**nice**).
- **id (idle)**: % de CPU inactiva.
- **wa (I/O wait)**: % de CPU esperando operaciones de entrada/salida (disco, red).
- **hi (hardware interrupts)**: % de CPU manejando interrupciones de hardware.
- **si (software interrupts)**: % de CPU manejando interrupciones de software.
- **st (steal time)**: % de CPU robado"por una máquina virtual (si se ejecuta en un entorno virtualizado).

Cuarta y quinta líneas: Uso de memoria RAM y Swap

MiB Mem : 7854.2 total, 2364.5 free, 3184.8 used, 1304.9 buff/cache
MiB Swap: 2048.0 total, 1024.0 free, 1024.0 used. 2345.6 avail Mem

- **Mem total**: Memoria RAM total del sistema.
- **free**: Memoria libre disponible.
- **used**: Memoria en uso.
- **buff/cache**: Memoria usada como caché y buffers.
- **Swap total**: Tamaño total del área de intercambio (swap).
- **Swap free**: Espacio de swap no utilizado.
- **Swap used**: Espacio de swap en uso.
- **avail Mem**: Memoria disponible para nuevas aplicaciones.

Lista de procesos

Esta sección muestra información sobre los procesos en ejecución. Cada columna tiene un significado:

```
PID  USER      PR NI  VIRT  RES  SHR S  %CPU %MEM  TIME+ COMMAND
```

Columnas principales

- **PID**: Identificador único del proceso.
- **USER**: Usuario que ejecuta el proceso.
- **PR (priority)**: Prioridad del proceso.
- **NI (nice)**: Valor nice, afecta la prioridad del proceso (-20 = más prioridad, 19 = menos prioridad).
- **VIRT (virtual memory)**: Memoria virtual utilizada (RAM + swap).
- **RES (resident memory)**: Memoria RAM utilizada sin contar swap.
- **SHR (shared memory)**: Memoria compartida con otros procesos.
- **S (state)**: Estado del proceso:
 - R: Running (ejecutándose).
 - S: Sleeping (durmiendo).
 - D: Uninterruptible sleep (espera de I/O).
 - T: Stopped (detenido).
 - Z: Zombie (proceso finalizado, pero aún listado).
- **%CPU**: Porcentaje de uso de CPU del proceso.
- **%MEM**: Porcentaje de uso de RAM del proceso.
- **TIME+**: Tiempo total de CPU consumido por el proceso.
- **COMMAND**: Comando o nombre del proceso.

Atajos útiles en top

- q: Salir de top.
- h: Mostrar ayuda.
- k: Matar un proceso (requiere PID).
- M: Ordenar por uso de memoria.
- P: Ordenar por uso de CPU.
- T: Ordenar por tiempo de ejecución.

Si entramos en el directorio `/proc/sys/net/ipv4`, podemos ver los parámetros de configuración del kernel.

Además del comando `uptime`, podemos usar el comando `free`, que nos muestra la memoria libre y ocupada. Si usamos el comando `free -h`, nos muestra la información en un formato más legible (human readable). Una de las ventajas de estos comandos es que no es interactivo, es decir, que ocupan la terminal, como si lo es el comando `top`. Además, podemos usar el comando `vmstat`.

El comando `stress` se usa para generar pruebas artificiales sobre el uso de memoria y demás.

Si usamos `htop`, es una versión mejorada de `top` y nos muestra la información de una manera más visual. Para instalarlo, usamos el comando `apt install htop`. Si lo ejecutamos, vemos que se nos muestra la información de una manera más visual.

2.3. Cron

Este se define como un demonio que se encarga de ejecutar tareas programadas en el sistema. Se utiliza para automatizar tareas repetitivas, como copias de seguridad, actualizaciones de software y mantenimiento del sistema. Podemos usar herramientas externas como <https://freeformatter.com>, para que nos de la configuración del crontab traduciendo para ello la expresión cron. Una alternativa a cron es `anacron`. Este se ejecuta por defecto una vez cada hora.

Debemos de aprender a manejar el comando `journalctl`. Si usamos `journalctl -f`, nos muestra los logs en tiempo real. Si usamos `journalctl -u cron`, nos muestra los logs del servicio de cron. Si usamos `journalctl -u cron --since "2023-10-01 12:00:00"`, nos muestra los logs desde esa fecha y hora, tenemos una gran infinidad de opciones para filtrar los logs³. Cuando reiniciamos el equipo, perdemos los mensajes de logs de ayer, por lo que debemos de tener cuidado con esto. Para habilitarlo debemos de entrar en `/etc/systemd/journald.conf` y cambiar el parámetro de Storage de auto a yes, descomentando la línea.

El programa `logrotate` se encarga de gestionar los logs, y lo podemos encontrar en `/etc/logrotate.conf`. Este programa se encarga de rotar los logs, es decir, de crear un nuevo log y eliminar el antiguo. Si no lo tenemos configurado, los logs pueden ocupar mucho espacio en disco. Dentro de `logrotate.d` podemos añadir archivos de configuración para cada uno de los logs.

2.4. Grafana + Prometheus

Prometheus es el encargado de preguntarle a cada nodo por sus métricas, este proceso se llama **scraping**. Lo que almacena son series temporales mediante muestreos que realiza cada cierto tiempo. Puede presentar una interfaz de usuario, pero esta es muy pobre.

Una alternativa de usar la BD de Prometheus es *InfluxDB* que es una opción de BD más rápida.

Vamos a usar la que tiene Grafana mediante en lenguaje de *PromQL*, en el cual se dibujan diagramas, alertas y demás.

³Para ello podemos entrar en la documentación oficial.

Tenemos lo que se conoce como *exporters* que son las interfaces web que va a llamar Prometheus.

Para acceder a la web de Prometheus debemos de acceder al enlace: <https://prometheus.io/>.

Debemos de conocer Counter, Gauge e Histogram.

Para empezar a hacer el ejercicio dentro del guión tenemos dos archivos, copiamos el contenido de los dos archivos de compose y de Prometheus.

```
1  docker-compose.yml:
2  ---
3  version: "3"
4  services: # usamos dos servicios
5  prometheus:
6  image: prom/prometheus:v2.50.0
7  ports:
8  - 9090:9090 # es el puerto de Prometheus
9  volumes:
10 - ./prometheus_data:/prometheus # aquí almacena todos sus datos y
    lo mapeamos en un directorio externo, de manera que los cambios
    que realicemos aquí se guardan de manera permanente.
11 - ./prometheus.yml:/etc/prometheus/prometheus.yml # lo almacena
    dentro de el directorio /etc correspondiente a la configuración
    de Prometheus, estamos montando un volumen.
12 command:
13 - "--config.file=/etc/prometheus/prometheus.yml"
14 grafana:
15 image: grafana/grafana:9.1.0
16 ports:
17 - 4000:3000 # el puerto estandar es el 3000, pero es el mismo de
    node, para que no haya colisión, debemos de mapearlo sobre el
    4000
18 volumes:
19 - ./grafana_data:/var/lib/grafana
20 depends_on:
21 - prometheus
```

Lo arrancamos como siempre con `docker compose up`, una vez que sepamos que funciona lo arrancamos en 2º plano para saltarnos todos los logs que aparecen. Cuando lo lanzamos se nos crean dos subdirectorios, la de Grafana y la de Prometheus, si queremos resetarlos borramos estos dos directorios, ya que estos son los que almacenan el estado.

```
1  prometheus.yml:
2  ---
3  global:
4  scrape_interval: 5s # el tiempo en el que le pregunta por sus
    metricas
5  scrape_configs:
6  - job_name: "prometheus_service" # el nombre del servicio, es decir
    , Prometheus se esta monitorizando a sí mismo.
7  static_configs: # de esta manera le estamos diciendo donde se
    encuentra el nombre de Prometheus.
8  - targets: ["prometheus:9090"]
```

Dentro de la web de Prometheus debemos de entrar en targets para depurar el código, cada uno de los exporters que aparecen si esta en verde es que esta **correcto**.

Para las consultas, vamos a la parte gráfica (Status/Targets). Si accedemos a la

dirección de Prometheus nos da error porque solo tiene sentido dentro del docker compose, por ende, debemos de cambiar a localhost y lo que sigue.

Una de las grandes ventajas de los exporters es la sencillez al usar texto plano.

No podemos usar safari debido a que no esta soportado correctamente con grafana y Prometheus.

En el ejercicio que nos pide, vamos a usar poco los labels, se van a usar más cuando hagamos la API el Jmeter, pero en general, se usan pocos, se copian las métricas, se deben de copiar completamente.

Hay una función muy importante que vamos a usar mucho dentro de Prometheus, **rate**.

Este comando `ob=api-server"rate(http_requests_totalj[5m])`.

Significado paso a paso:

- `http_requests_total{job=.api-server}`:
 - Selecciona la métrica `http_requests_total`, que suele ser un contador del número total de peticiones HTTP.
 - El filtro `{job=.api-server}` indica que solo se quieren las métricas donde la etiqueta `job` tenga el valor `.api-server`. Esto restringe los resultados a esa parte del sistema.
- `[5m]`:
 - Es un selector de rango que indica: "mira los últimos 5 minutos de esa serie temporal".
- `rate(...)`:
 - La función `rate()` calcula la tasa de cambio promedio por segundo de una métrica de tipo *counter* dentro del rango de tiempo especificado.
 - En este caso: "¿a qué velocidad están aumentando las peticiones HTTP por segundo en los últimos 5 minutos?".

¿Qué te devuelve?

Te da la tasa promedio de solicitudes HTTP por segundo para el `api-server`, calculada a partir de los datos recogidos en los últimos 5 minutos.

Uso típico:

Esta consulta se usa comúnmente en dashboards de Grafana o alertas de Prometheus, para ver el ritmo de carga que tiene un servicio web.

Entramos en grafana: `https://grafana.com/`

Entramos en Grafana y en Settings/ Data Sources y usamos Prometheus, le damos el puerto y `http://prometheus:9090`.

Ahora debemos de hacer un *Dashboards*, New, New Dashboard, add new panel, seleccionamos labels, y metrics, se recomienda hacer con el builder ya que va sugiriendo. En el campo vacío añadimos el comando `rate` de antes.

Si nos damos cuenta el resultado de rate es un tanto por 1, por ende, si lo multiplicamos $\times 100$ nos da un porcentaje, para ello buscamos la opciónn `multiply`. Dentro de la definición del espacio de trabajo podemos ajustar la curvatura de las líneas, el estilo, ejes, ...

También podemos añadir reglas/alertas para que, por ejemplo, cuando llegue al 70% de CPU que me avise.

2.4.1. Ejercicio Obligatorio. Monitorización con Grafana + Prometheus.

Dentro de la máquina Rocky debemos de instalar le exporter de Prometheus. Este se puede lanzar por línea de comandos. Debemos de crear un servicio para que cuando se inicie el equipo estén los exporters, y estos publican los puertos HTTP donde se puede hacer scraping.

Este enlace nos ofrece una guía de como hacerlo paso a paso, es decir, un tutorial: <https://devopscube.com/monitor-linux-servers-prometheus-node-exporter/>. Si accedemos desde fuera no nos va a dejar, por ende, debemos de habilitar ese puerto en el firewall. Este año debemos de modificar el security extension, ya que si se cae no nos dice nada, en este caso podemos deshabilitarlas (no recomienda), o ejecutar el comando `restorecon -v <donde lo hayamos instalado>`.

Ahora debemos de modificar el siguiente fichero: `prometheus.yml`: y añadimos:

```
1 - job_name: "node" # se recomienda poner node
2 static_configs:
3 - targets: ["IP DE NUESTRA MÁQUINA:9100"]
```

Dentro de grafana debemos de copiar el ID, entramos en los dashboard y lo importamos, y copiamos el ID.

Nota: Debemos de ponerlo node, ya que por defecto es mejor y más sencillo.

Se recomienda coger el dashboard, vamos al panel y lo fusionamos, esto no es fácil, debemos de preguntarle al profesor.

Debemos de entregar un pdf en el que se explique todo al detalle con capturas de pantalla y demás. De la parte de monitores debemos de parar aunque sea 1. Lo más importante es que se vea la gráfica.

Dentro de la API web podemos añadir `/metrics` y vemos las métricas.

Una de las alternativas más rentables es coger todos los contenedores del JS y añadirlo dentro del Docker Compose. La mejor sería usar la tarjeta que nos ofrece el propio `docker compose`. No podemos usar la de la IP ya que cada vez que arrancamos la máquina esta cambia al usar una IP dinámica.

Parte II
Prácticas

Capítulo 3

Bloque 1

3.1. Ejercicio 1 Opcional

Cuando se hace referencia a la imagen X, se refiere a la imagen 2.X(hasta el punto de Firewall)

El alumno/a debe ser capaz de presentar un MV con la configuración descrita en este apartado. La configuración debe ser permanente, es decir, en todo caso, tras reiniciar el equipo, la configuración será la esperada. Para validar la configuración de red, el alumno/a debe ser capaz de:

- Hacer ping desde el equipo anfitrión a la MV y viceversa.
- Hacer ping desde la MV a cualquier equipo accesible públicamente en Internet por FQHN o IP.
- Conectar por ssh desde el equipo anfitrión a la MV .

3.1.1. Solución

Una vez que hayamos instalado el SO que se nos pide correctamente. Debemos de realizar una serie de ajustes previos:

- Añadir nuestro usuario, para ello debemos de ejecutar lo siguientes comandos (iniciando como usuario root):
 - `sudo useradd nombre_de_usuario`
 - `sudo passwd nombre_de_usuario`
 - `sudo usermod -aG wheel nombre_de_usuario` para que pueda usar el comando sudo.
- Configurar la red NAT y una de tipo Host-Only, para ello en Herramientas en la VM debemos seleccionar la opción de Red y añadir una nueva interfaz de red de tipo Host-Only, y paso seguido configurar la red NAT (ver Figura 1 y 2)

- Comprobar que el servicio SSH está instalado, por defecto se suele instalar, para asegurarnos debemos de ejecutar el comando `sudo systemctl status ssh`. En el caso de que no venga instalado debemos de ejecutar el comando `sudo dnf install -y openssh-server openssh-clients`¹ (Ver Figura 5).
- Cambiar la variable PS1 como se nos pedía, para ello debemos de editar el fichero de `bashrc` y exportar la variable PS1 con el valor que se nos pedía:
 - PS1='`\u@\h:\t:\w\$`' (Ver Figura 5), donde:
 - `\u`: Nombre del usuario actual.
 - `\h`: Nombre del hostname (nombre del sistema).
 - `\t`: Hora actual en formato de 24 horas (HH:MM:SS).
 - `\w`: Directorio de trabajo actual.
 - `$`: Símbolo del prompt, que será `$` para un usuario normal y `#` para root.

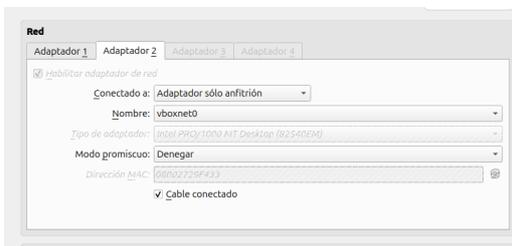


Figura 3.1: Configuración de la red NAT y Host-Only

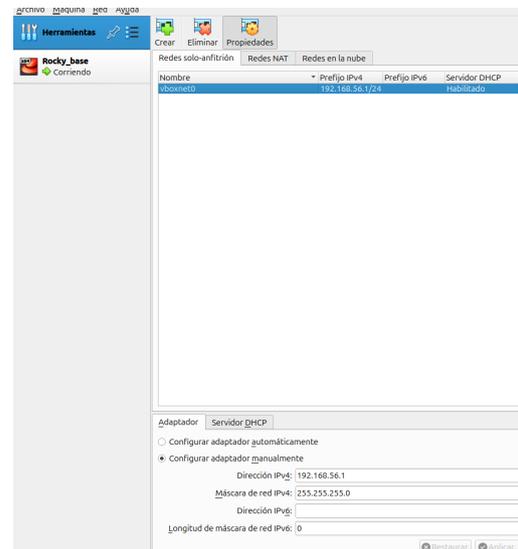


Figura 3.2: Configuración de Virtual-Box para la red de Host-Only

Además se nos pide que la Ip de Host Only sea estática, para ello vamos a asegurarnos usando la herramienta `nmtui`, en la que vamos a ver si es estática o no la ip. Como podemos ver en la siguiente imagen esta configurada como ip automática, que viene siendo lo mismo que dinámica por lo que debemos de cambiarlo a manual para poder configurar la ip estática. (Ver Figura 3 y 4). Para ver que efectivamente la ip cambió, podemos verlo en la Figura 5.

Una vez hayamos cambiado la ip estática, debemos de verificar que efectivamente se ha cambiado y para ello usamos el comando `ip a` y vemos que efectivamente se ha cambiado la ip a la que hemos asignado. (Ver Figura 7).

Llegado a este punto vamos a realizar un ping a la máquina anfitriona y viceversa, para ello usamos el comando `ping -c <número de pings>ip_de_la_maquina` y vemos que efectivamente hay conexión entre ambas máquinas. (Ver Figura 8, 9 y

¹Incluimos clients para añadir el servicio de cliente.



Figura 3.3: Con nmtui vemos que es dinámica

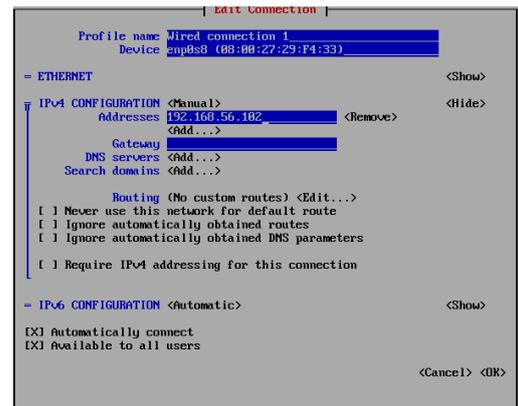


Figura 3.4: Cambiamos a manual y asignamos una ip estática válida

```

ism@vbox:~$ sudo systemctl status sshd
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2025-03-05 12:55:40 CET; 1min 17s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
   Main PID: 1894 (sshd)
     Tasks: 1 (limit: 11109)
    Memory: 1.8M
       CPU: 6ms
   CGroup: /system.slice/ssh.service
           └─1894 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

Mar 05 12:55:40 vbox systemd[1]: Starting OpenSSH server daemon...
Mar 05 12:55:40 vbox sshd[1894]: Server listening on 0.0.0.0 port 22.
Mar 05 12:55:40 vbox sshd[1894]: Server listening on :: port 22.
Mar 05 12:55:40 vbox systemd[1]: Started OpenSSH server daemon.
ism@vbox:~$ export PS1='\u@h:\t:\w\$ '
ism@vbox:~$ source ~/.bashrc
ism@vbox:~$

```

Figura 3.5: Sshd y variable PS1

```

ism@vbox:~$ source ~/.bashrc
ism@vbox:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: emp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
   link/ether 08:00:27:c8:87:e3 brd ff:ff:ff:ff:ff:ff
   inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute emp0s3
       valid_lft 85979sec preferred_lft 85979sec
   inet6 fd00::a00:27ff:fe08:87e3/64 scope global dynamic noprefixroute
       valid_lft 85982sec preferred_lft 13982sec
   inet6 fe80::a00:27ff:fe08:87e3/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: emp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
   link/ether 08:00:27:29:f4:33 brd ff:ff:ff:ff:ff:ff
   inet 192.168.56.101/24 brd 192.168.56.255 scope global dynamic noprefixroute emp0s8
       valid_lft 479sec preferred_lft 479sec
   inet6 fe80::ed72:4176:68fa:f368/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
ism@vbox:~$

```

Figura 3.6: Resultado de ip a

```

ismMV01@vbox ~]$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:c8:87:e3 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
        valid_lft 86142sec preferred_lft 86142sec
    inet6 fd00::a00:27ff:fec8:87e3/64 scope global dynamic noprefixroute
        valid_lft 86143sec preferred_lft 14143sec
    inet6 fe80::a00:27ff:fec8:87e3/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:29:f4:33 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.102/24 brd 192.168.56.255 scope global noprefixroute enp0s8
        valid_lft forever preferred_lft forever
    inet6 fe80::ed72:4176:68fa:f368/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
ismMV01@vbox ~]$

```

Figura 3.7: Resultado de `ip a` para verificar el cambio de ip

10). Además, vemos que gracias al *NAT* podemos hacer ping a cualquier máquina accesible en internet². (Ver Figura 11).

```

> ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: eno1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000
    link/ether 08:9c:25:bf:41:3e brd ff:ff:ff:ff:ff:ff
    altname enp2s0
3: vlp3s1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 1c:ca:51:46:df:78 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.138/24 brd 192.168.1.255 scope global dynamic noprefixroute vlp3s1
        valid_lft 84512sec preferred_lft 84512sec
    inet6 2a0c:5a82:250b:5f00:f4ae:9c2:6471:95b/64 scope global temporary dynamic
        valid_lft 602913sec preferred_lft 84257sec
    inet6 2a0c:5a82:250b:5f00:f4ae:9c2:6471:95b/64 scope global mngtmtaddr noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::c2d2:77d0:65d9:9c71/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
4: vboxnet8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 0a:00:27:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.1/24 brd 192.168.56.255 scope global vboxnet8
        valid_lft forever preferred_lft forever
    inet6 fe80::800:27ff:fe00:0/64 scope link
        valid_lft forever preferred_lft forever

```

Figura 3.8: Resultado del comando de `ip a` en la máquina anfitrión para ver la ip

```

> ping -c 3 192.168.56.102
PING 192.168.56.102 (192.168.56.102) 56(84) bytes of data.
64 bytes from 192.168.56.102: icmp_seq=1 ttl=64 time=0.505 ms
64 bytes from 192.168.56.102: icmp_seq=2 ttl=64 time=0.146 ms
64 bytes from 192.168.56.102: icmp_seq=3 ttl=64 time=0.336 ms

--- 192.168.56.102 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2071ms
rtt min/avg/max/mdev = 0.146/0.329/0.505/0.146 ms

```

Figura 3.10: Ping de la máquina anfitrión a la máquina virtual

```

ismMV01@vbox-22:29:11 ~]$ ping -c 3 192.168.1.138
PING 192.168.1.138 (192.168.1.138) 56(84) bytes of data.
64 bytes from 192.168.1.138: icmp_seq=1 ttl=255 time=0.356 ms
64 bytes from 192.168.1.138: icmp_seq=2 ttl=255 time=0.362 ms
64 bytes from 192.168.1.138: icmp_seq=3 ttl=255 time=0.466 ms

--- 192.168.1.138 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2042ms
rtt min/avg/max/mdev = 0.356/0.394/0.466/0.050 ms
ismMV01@vbox-22:29:24 ~]$

```

Figura 3.9: Ping a la máquina anfitrión

```

ismMV01@vbox-22:30:01 ~]$ ping -c 3 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=255 time=77.9 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=255 time=101 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=255 time=122 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 77.912/100.350/122.369/18.151 ms
ismMV01@vbox-22:30:58 ~]$

```

Figura 3.11: Ping a un servidor público (Google)

En cuanto al servicio `ssh`, debemos de ver el estado del servicio `sshd` con el comando `sudo systemctl status sshd` y vemos que esta corriendo. En este punto desde el anfitrión podemos introducir la línea de comando `ssh ismMV01@192.168.56.102` y vemos que efectivamente todo funciona correctamente. (Ver Figura 12 y 13).

²Cabe destacar que durante el desarrollo de la actividad, surgían algunas problemas con `NetworkManager`, `polkit` y `DBus`, pero se solucionaban al reiniciarlos o bien reinstalarlos

```

> ssh ismV01@192.168.56.102
The authenticity of host '192.168.56.102 (192.168.56.102)' can't be established.
ED25519 key fingerprint is SHA256:WkKcPCXSCNaGe/95NN14fkq/WYFcvacXnh9pLQC3Jo.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.56.102' (ED25519) to the list of known hosts.
ismV01@192.168.56.102's password:
Last login: Wed Mar  5 22:50:26 2025
[ismV01@vbox-22:54:09 ~]$ ls
[ismV01@vbox-22:54:12 ~]$ echo "el anfitrión ha entrado" >> anfitrión.txt
[ismV01@vbox-22:54:44 ~]$ ls
anfitrión.txt
[ismV01@vbox-22:54:46 ~]$ exit
logout
Connection to 192.168.56.102 closed.

```

Figura 3.12: Ssh en la máquina anfitrión y creación de un archivo en la MV

```

[ismV01@vbox-22:51:47 ~]# ls
anfitrión.txt
[ismV01@vbox-22:55:10 ~]# cat anfitrión.txt
el anfitrión ha entrado
[ismV01@vbox-22:55:12 ~]#

```

Figura 3.13: Ver el contenido del archivo creado en la MV desde el anfitrión

3.2. Servidor con LVM + RAID

3.2.1. Aspectos clave de LVM

Para gestionar eficazmente el *Logical Volume Manager* (LVM), es fundamental comprender los siguientes componentes y conceptos:

Componentes de la arquitectura de almacenamiento

- **Physical Volume (PV):** Representa los dispositivos de almacenamiento físico, como discos duros o particiones, que se incorporan al sistema LVM.
- **Volume Group (VG):** Es una agrupación de uno o más PVs que forman un pool de almacenamiento, del cual se pueden asignar espacios para crear volúmenes lógicos.
- **Logical Volume (LV):** Son volúmenes virtuales creados dentro de un VG. Los LVs se utilizan como si fueran particiones de disco tradicionales, permitiendo la creación de sistemas de archivos o la asignación directa a aplicaciones.

Gestión de almacenamiento con diferentes características físicas

LVM ofrece flexibilidad para manejar dispositivos de almacenamiento con diversas características físicas:

- **HDD y SSD:** Se pueden combinar en un mismo VG, permitiendo equilibrar rendimiento y capacidad según las necesidades.
- **RAID:** LVM puede trabajar sobre dispositivos RAID, proporcionando una capa adicional de gestión y flexibilidad sobre la configuración RAID existente.

Etiquetado y correspondencia con los archivos de dispositivo

Cada componente en LVM tiene una nomenclatura específica y se asocia a archivos de dispositivo en el sistema:

- **Physical Volumes:** Corresponden a dispositivos físicos, como `/dev/sda1`, `/dev/sdb1`, etc.
- **Volume Groups:** Se nombran según la convención establecida por el administrador, por ejemplo, `vg_datos`.

- **Logical Volumes:** Se nombran dentro de su VG correspondiente, como `/dev/vg_datos/lv_backup`, donde `lv_backup` es el nombre del LV.

Comandos de LVM para la gestión de componentes

LVM proporciona una serie de comandos para administrar sus componentes:

- `pvcreate`: Inicializa un dispositivo físico como PV.
- `vgcreate`: Crea un VG a partir de uno o más PVs.
- `lvcreate`: Crea un LV dentro de un VG.
- `pvs`, `vgs`, `lvs`: Muestran información sobre PVs, VGs y LVs respectivamente.
- `pvremove`, `vgremove`, `lvremove`: Eliminan PVs, VGs y LVs respectivamente.

Estos comandos permiten una gestión eficiente y flexible del almacenamiento en sistemas que utilizan LVM.

3.2.2. Niveles de RAID: 0, 1 y 5

Redundant Array of Independent Disks (RAID) es una tecnología que permite combinar múltiples dispositivos de almacenamiento en una unidad lógica para mejorar el rendimiento, la redundancia o ambos. A continuación, se detallan los niveles de RAID 0, 1 y 5, sus ventajas, desventajas y su administración en sistemas Linux utilizando la herramienta de línea de comandos `mdadm`.

RAID 0

RAID 0, conocido como *striping*, distribuye los datos de manera equitativa entre dos o más discos sin información de paridad ni redundancia.

- **Ventajas:**
 - Mayor rendimiento en lectura y escritura debido a la distribución de datos entre los discos.
 - Uso completo de la capacidad de almacenamiento total, ya que no se reserva espacio para paridad o duplicación.
- **Desventajas:**
 - Ausencia de redundancia; la falla de un solo disco resulta en la pérdida total de los datos.

RAID 1

RAID 1, o *mirroring*, duplica los datos en dos o más discos, creando copias idénticas en cada uno.

■ Ventajas:

- Alta redundancia; los datos permanecen intactos incluso si uno de los discos falla.
- Mejora en la velocidad de lectura, ya que los datos pueden leerse desde cualquiera de los discos.

■ Desventajas:

- Capacidad de almacenamiento efectiva reducida al 50 % del total, ya que los datos se duplican.

RAID 5

RAID 5 combina rendimiento y redundancia distribuyendo los datos y la paridad entre tres o más discos.

■ Ventajas:

- Proporciona tolerancia a fallos; si un disco falla, los datos pueden recuperarse con la información de paridad.
- Mejor aprovechamiento del almacenamiento comparado con RAID 1, ya que solo se utiliza una fracción del espacio para la paridad.

■ Desventajas:

- Rendimiento de escritura inferior al de RAID 0 debido al cálculo de la paridad.
- En caso de falla de un disco, la reconstrucción puede ser lenta y afectar el rendimiento.

Administración de RAID en Linux con mdadm

La herramienta `mdadm` permite gestionar arreglos RAID en Linux. A continuación, se presentan comandos esenciales:

■ Crear un RAID 0 con dos discos:

```
1 mdadm --create --verbose /dev/md0 --level=0 --raid-devices  
=2 /dev/sdX /dev/sdY  
2
```

■ Crear un RAID 1 con dos discos:

```
1 mdadm --create --verbose /dev/md0 --level=1 --raid-devices  
=2 /dev/sdX /dev/sdY  
2
```

- Crear un RAID 5 con tres discos:

```
1 mdadm --create --verbose /dev/md0 --level=5 --raid-devices
  =3 /dev/sdX /dev/sdY /dev/sdZ
2
```

- Verificar el estado del RAID:

```
1 cat /proc/mdstat
2
```

- Detener un RAID:

```
1 mdadm --stop /dev/md0
2
```

3.2.3. Aspectos clave para la administración de servidores Linux

Para implementar eficazmente soluciones en servidores Linux, es esencial comprender y manejar los siguientes aspectos:

Modos de ejecución y modo de mantenimiento en un servidor Linux

Los sistemas Linux operan en diferentes niveles de ejecución o *runlevels*, que determinan los servicios y procesos que se ejecutan. Con la adopción de *systemd*, estos niveles se denominan *targets*. El modo de mantenimiento, conocido como *rescue.target* o *emergency.target*, es crucial para tareas de recuperación y administración del sistema. Para cambiar al modo de mantenimiento, se puede utilizar el siguiente comando:

```
1 sudo systemctl isolate rescue.target
```

Para volver al modo multiusuario estándar:

```
1 sudo systemctl isolate multi-user.target
```

Estructura estándar del sistema de archivos en Linux

La estructura de directorios en Linux sigue el estándar de jerarquía de sistemas de archivos (*Filesystem Hierarchy Standard - FHS*). Algunos directorios principales incluyen:

- **/**: Directorio raíz que contiene todos los demás directorios.
- **/bin**: Ejecutables esenciales para todos los usuarios.
- **/etc**: Archivos de configuración del sistema.
- **/home**: Directorios personales de los usuarios.
- **/var**: Datos variables como registros y colas de impresión.

Sistemas de archivos comunes en Linux

Linux soporta diversos sistemas de archivos. Algunos de los más comunes son:

- **ext4**: Sistema de archivos por defecto en muchas distribuciones, conocido por su estabilidad y rendimiento.
- **XFS**: Adecuado para manejar grandes volúmenes de datos y archivos de gran tamaño.
- **Btrfs**: Ofrece características avanzadas como instantáneas (*snapshots*) y compresión.

Montaje y desmontaje de volúmenes

El montaje de sistemas de archivos permite acceder a dispositivos de almacenamiento. Para montar un dispositivo:

```
1 sudo mount /dev/sdX1 /mnt/punto_de_montaje
```

Para desmontarlo:

```
1 sudo umount /mnt/punto_de_montaje
```

Las configuraciones de montaje persistentes se definen en el archivo `/etc/fstab`.

Comandos básicos para la gestión de archivos

La administración de archivos en Linux se realiza mediante comandos de línea.

Estos se deben de haber estudiado en asignaturas anteriores como Sistemas Operativos.

3.2.4. Ejercicio Opcional

Partiendo de un servidor básico configurado de acuerdo al apartado 2, el alumno/a deberá afrontar el caso práctico descrito a continuación:

Se desea instalar un servicio de gestión documental en el servidor. Se espera que este servicio precise de una cantidad espacio de almacenamiento creciente con el tiempo, pudiendo llegar a ser considerable.

Por otro lado, el contenido será crítico, por lo que se desea proporcionar algún mecanismo de respaldo ante fallos en el dispositivo de almacenamiento.

El alumno/a debe diseñar los cambios en el sistema de almacenamiento e implementarlo empleando prácticas adecuadas de administración que garanticen la conservación de la información en el sistema y procuren la máxima disponibilidad del servicio.

Solución

Para la resolución de este ejercicio vamos a seguir lo realizado en clase. (Ver apuntes de clase correspondientes a la sección de LVM+RAID).

En primer lugar creamos los discos, para ello entramos en la configuración de la máquina virtual y añadimos dos discos duros de 1GB cada uno. Una vez añadidos los discos duros, iniciamos la máquina virtual y ejecutamos el comando `lsblk` para ver los discos duros que tenemos disponibles. (Ver Figura 14).

```

lsml001@vbox-23:44:20 ~]$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda         8:0    0   20G  0 disk
├─sda1      8:1    0    1G  0 part /boot
├─sda2      8:2    0   19G  0 part
└─r1_vbox-root 253:0  0   17G  0 lvm /
   └─r1_vbox-swap 253:1  0    2G  0 lvm [SWAP]
sdb         8:16   0    1G  0 disk
sdc         8:32   0    1G  0 disk
sr0         11:0   1 1024M  0 rom
lsml001@vbox-23:44:26 ~]$
    
```

Figura 3.14: Resultado del comando lsblk

```

lsml001@vbox-23:43:46 ~]$ sudo mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sdb /dev/sdc
mdadm: Note: this array has metadata at the start and
may not be suitable as a boot device.  If you plan to
store "/boot" on this device please ensure that
your boot-loader understands md/v1.x metadata, or use
--metadata=9.99
Continue creating array [p#0] y
mdadm: Defaulting to version 1.2 metadata
[ 465.188531] md/raid1:md0: md: clean -- starting background reconstruction
[ 465.188724] md/raid1:md0: active with 2 out of 2 mirrors
[ 465.188749] md0: detected capacity change from 0 to 2893056
mdadm: array /dev/md0 started.
lsml001@vbox-23:58:33 ~]$ [ 465.194978] md: rescue of RAID array md0
[ 470.678044] md: md0: rescue done.
    
```

Figura 3.15: Comando mdadm

A continuación, instalamos *mdadm* con el comando `sudo dnf install -y mdadm` y creamos un RAID 1 con los dos discos duros que hemos añadido. Para ello ejecutamos el comando `sudo mdadm -create -verbose /dev/md0 -level=1 -raid-devices=2 /dev/sdb /dev/sdc` y vemos que se ha creado correctamente. (Ver Figura 15).

Acto seguido debemos de crear un PV, VG y LV. Para ello ejecutamos los siguientes comandos (Ver Figura 16):

- `sudo pvcreate /dev/md0`
- `sudo vgcreate vg_datos /dev/md0`
- `sudo lvcreate -L 900M -n lv_datos vg_datos`

```

lsml001@vbox-23:53:45 ~]$ sudo pvcreate /dev/md0
Physical volume "/dev/md0" successfully created.
lsml001@vbox-23:53:59 ~]$ sudo vgcreate raid1 /dev/mapper/mcolog_md0 mem mqueuz/
lsml001@vbox-23:53:59 ~]$ sudo vgcreate raid1 /dev/md0
Volume group "raid1" successfully created.
lsml001@vbox-23:54:18 ~]$ sudo lvcreate -L 10G -n rvar raid1
Volume group "raid1" has insufficient free space (255 extents): 2560 required.
lsml001@vbox-23:54:42 ~]$ sudo lvcreate -L 1G -n rvar raid1
Volume group "raid1" has insufficient free space (255 extents): 256 required.
lsml001@vbox-23:55:04 ~]$ sudo lvcreate -L 900M -n rvar raid1
Logical volume "rvar" created.
lsml001@vbox-23:55:13 ~]$
    
```

Figura 3.16: Resultado de crear los volúmenes

```

lsml001@vbox-23:57:29 ~]$ sudo mkfs.ext4 /dev/mapper/raid1-rvar
mke2fs 1.46.5 (30-Dec-2021)
Creating filesystem with 230400 4k blocks and 57600 inodes
Filesystem UUID: 22749b96-b1ce-44b3-acef-5444bb69e307
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376
Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
lsml001@vbox-23:57:38 ~]$
    
```

Figura 3.17: Comando mkfs

Ahora debemos de formatear el volumen lógico en formato ext4. (Ver Figura 17). A continuación, montamos (Ver Figura 18) y trasladamos la carpeta *var* al nuevo volumen lógico. Para ello ejecutamos los comando que figuran en los apuntes de clase.

```

lsml001@vbox-00:01:28 ~]$ sudo mount /dev/mapper/raid1-rvar /mnt/
[ 119.180431] EXT4-fs (dm-2): mounted filesystem with ordered data mode. Quota mode: none.
lsml001@vbox-00:01:27 ~]$
    
```

Figura 3.18: Resultado del comando mount

```

lsml001@vbox-00:03:55 ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        869M   0  869M   0% /dev
tmpfs           888M   0  888M   0% /dev/shm
tmpfs           356M  5.0M  351M   2% /run
tmpfs           17G   1.1G   16G   7% /
/dev/mapper/r1_vbox-root 1814M 196M 1618M 12% /
/dev/sda1       1814M 196M 1618M 12% /boot
/dev/mapper/raid1-rvar 868M  24K 807M  1% /mnt
tmpfs          178M   0  178M   0% /run/user/1000
lsml001@vbox-00:03:59 ~]$
    
```

Figura 3.19: Comando df -h

Ejecutamos el comando `df -h` para ver que efectivamente se ha montado correctamente el volumen lógico. (Ver Figura 19).

Ejecutamos el `isolate` del `systemctl` para entrar en modo de mantenimiento y poder trasladar la carpeta *var* al nuevo volumen lógico. (Ver Figura 20).

Ahora debemos de trabajar con la serie de comandos que se especifican para poder mover la carpeta y que se haga de forma correcta. (Ver Figura 21).

Una vez trasladada la carpeta, debemos de hacer permanentes los cambios, para ello debemos de editar el fichero `/etc/fstab` y añadir la siguiente línea (Ver Figura 22):

```

[ 1380.655560] audit: type=1131 audit(1741647077.136:317): pid=1 uid=0 auid=4294967295 ses=42949672
5 sub_jsystem_u:system_r:init t:0 msg="unit=firewalld com=systemd" exe="/usr/lib/systemd/systemd"
hostname? addr? terminal? res=success
[ 1380.663641] audit: type=1131 audit(1741647077.143:318): pid=1 uid=0 auid=4294967295 ses=42949672
5 sub_jsystem_u:system_r:init t:0 msg="unit=polkit com=systemd" exe="/usr/lib/systemd/systemd" h
ostname? addr? terminal? res=success
[ 1380.718221] audit: type=1131 audit(1741647077.198:319): pid=1 uid=0 auid=4294967295 ses=42949672
5 sub_jsystem_u:system_r:init t:0 msg="unit=NetworkManager com=systemd" exe="/usr/lib/systemd/sy
stemd" hostname? addr? terminal? res=success
[ 1380.818811] audit: type=1131 audit(1741647077.298:320): pid=1 uid=0 auid=4294967295 ses=42949672
5 sub_jsystem_u:system_r:init t:0 msg="unit=rsyslog com=systemd" exe="/usr/lib/systemd/systemd"
hostname? addr? terminal? res=success
[ 1380.865572] audit: type=1131 audit(1741647077.307:321): pid=1 uid=0 auid=4294967295 ses=42949672
5 sub_jsystem_u:system_r:init t:0 msg="unit=dbus-broker com=systemd" exe="/usr/lib/systemd/system
d" hostname? addr? terminal? res=success
[ 1380.836348] audit: type=1131 audit(1741647077.316:322): prog-id=68 op=1000
You are in rescue mode. After logging in, type 'journalctl -xb' to view
system logs, 'systemctl reboot' to reboot, 'systemctl default' or 'exit'
to boot into default mode.
Give root password for maintenance
(or press Control-D to continue): [ 1310.861307] kernel: printk: sbk: 4 callbacks suppressed
[ 1310.861307] audit: type=1131 audit(1741647087.341:327): pid=1 uid=0 auid=4294967295 ses=42949672
5 sub_jsystem_u:system_r:init t:0 msg="unit=NetworkManager-dispatcher com=systemd" exe="/usr/lib
/systemd/systemd" hostname? addr? terminal? res=success
    
```

Figura 3.20: Resultado del comando `systemctl isolate`

```

root@vbox:~# cp -r /var/* /mnt/
root@vbox:~# mv /usr/* /usr.old
root@vbox:~# umount /mnt/
root@vbox:~# mkdir /var
root@vbox:~# mount /dev/mapper/raid1-rvar /var
[ 1532.091919] EXT4-fs (da-2): mounted filesystem with ordered data mode. Quota mode: none.
root@vbox:~# df -h
Filesystem              Size  Used Avail Use% Mounted on
/dev/mapper/rl_vbox-root 176G  1.1G  166G   1% /
/dev/sda1                1814M  196M  1618M  12% /boot
tmpfs                   1792M   0 1792M   0% /run/user/1000
/dev/mapper/raid1-rvar  868M   75M  793M  10% /var
root@vbox:~#
    
```

Figura 3.21: Serie de operaciones para trasladar la carpeta `var`

```
1 /dev/mapper/raid1-rvar /var ext4 defaults 0 0
```

Recargamos la configuraciones con el daemon de `systemd` y vemos que aparece la parte que estamos montando (Ver Figura 23).

```

/etc/fstab
Created by anaconda on Wed Mar  5 11:25:01 2025

# Accessible filesystems, by reference, are maintained under '/dev/disk/'.
# See man pages (fstab(8), findfs(8), mount(8) and/or blkid(8) for more info.
# After editing this file, run 'systemctl daemon-reload' to update systemd
# units generated from this file.
/dev/mapper/rl_vbox-root /      xfs     defaults        0 0
[ 1310.668278] s1s-disk-af84-mbdf0108324 /boot xfs     defaults        0 0
/dev/mapper/rl_vbox-swap none  swap    defaults        1 0
/dev/mapper/raid1-rvar  /var  ext4    defaults        0 0
    
```

Figura 3.22: Resultado de editar `/etc/fstab` como se indica

```

debugfs on /sys/kernel/debug type debugfs (rw,nosuid,nodev,noexec,relat
tracefs on /sys/kernel/tracing type tracefs (rw,nosuid,nodev,noexec,relat
fusectl on /sys/fs/fuse/connections type fusectl (rw,nosuid,nodev,noexec,relat
configfs on /sys/kernel/config type configfs (rw,nosuid,nodev,noexec,relat
/dev/sda1 on /boot type xfs (rw,relatime,seclabel,attr2,inode64,logt
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,seclabel,
nfs=790,uid=1000,gid=1000,inode64)
/dev/mapper/raid1-rvar on /var type ext4 (rw,relatime,seclabel)
    
```

Figura 3.23: Resultado del comando `systemctl daemon-reload` y `mount`

Y como podemos comprobar en la Figura 24, hemos conseguido trasladar la carpeta `var` al nuevo volumen lógico y todo funciona correctamente.

```

Rocky Linux 9.0 (Blue Onyx)
Kernel 5.14.0-70.13.1.el9_0.x86_64 on an x86_64

vbox login: ismM001
Password:
Last login: Tue Mar 11 00:03:55 on tty1
[ismM001@vbox-00:26:41 ~]$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINTS
sda          8:0    0   20G  0 disk
├─sda1       8:1    0    1G  0 part  /boot
├─sda2       8:2    0   19G  0 part
├─r1_vbox-root 253:0    0   17G  0 lvm   /
├─r1_vbox-swap 253:1    0    2G  0 lvm   [SWAP]
sdb          8:16   0    1G  0 disk
├─md127     9:127  0  1022M  0 raid1
├─raid1-rvar 253:2    0   900M  0 lvm   /var
sdc          8:32   0    1G  0 disk
├─md127     9:127  0  1022M  0 raid1
├─raid1-rvar 253:2    0   900M  0 lvm   /var
sr0         11:0    1  1024M  0 rom
[ismM001@vbox-00:26:44 ~]$
    
```

Figura 3.24: Resultado final

3.3. Acceso seguro al servidor: Firewall + SSHD

3.3.1. Iptables

`iptables` es una herramienta de línea de comandos utilizada para configurar el cortafuegos en el núcleo de Linux, implementado dentro del proyecto Netfilter. Aunque `iptables` es un marco heredado, `nftables` se presenta como su reemplazo moderno, ofreciendo una capa de compatibilidad.

Instalación

El núcleo estándar de Arch Linux viene compilado con soporte para `iptables`. Para instalar las utilidades de usuario, se debe instalar el paquete `iptables`, que es una dependencia indirecta del metapaquete `base`, por lo que debería estar instalado por defecto en el sistema.

Conceptos Básicos

- **Tablas:** Colecciones de reglas con propósitos específicos.
- **Cadenas:** Listas de reglas dentro de una tabla que se recorren en orden.
- **Reglas:** Definiciones que consisten en un criterio de coincidencia y una acción a ejecutar si se cumple dicho criterio.

Configuración y Uso

`iptables` se puede configurar directamente desde la línea de comandos o mediante diversas interfaces frontales, tanto de consola como gráficas. Para mostrar las reglas actuales, se utiliza:

```
iptables -L
```

Para restablecer las reglas:

```
iptables -F
```

Registro de Actividad

`iptables` permite registrar paquetes para monitorear actividad o depurar reglas. Es posible limitar la tasa de registro para evitar la saturación de los registros del sistema.

Alternativas y Herramientas Relacionadas

- `nftables`: Proyecto que busca reemplazar a `iptables`, proporcionando un nuevo marco de filtrado de paquetes y una utilidad de espacio de usuario llamada `nft`.
- `ipset`: Aplicación complementaria que permite crear conjuntos de direcciones IP para ser utilizados en reglas de `iptables`, facilitando el bloqueo o la aceptación de múltiples direcciones de manera eficiente.

- **ufw (Uncomplicated Firewall):** Programa para gestionar el cortafuegos de manera sencilla, proporcionando una interfaz de línea de comandos fácil de usar.

Recursos Adicionales

Para configuraciones más avanzadas, como la creación de un cortafuegos con estado o compartir la conexión a Internet, se pueden consultar las siguientes guías:

- **Cortafuegos con Estado Sencillo:** Explica cómo configurar un cortafuegos con estado utilizando `iptables`, detallando las reglas necesarias y su propósito.
- **Compartir Conexión a Internet:** Describe cómo compartir la conexión a Internet desde una máquina a otras, incluyendo los requisitos y pasos necesarios.

3.3.2. firewalld y firewall-cmd en Rocky Linux

`firewalld` es un servicio de cortafuegos dinámico que permite gestionar reglas de firewall sin necesidad de reiniciar el servicio. Su herramienta de línea de comandos, `firewall-cmd`, proporciona una interfaz para la administración del firewall en sistemas como Rocky Linux.

Gestión del Firewall con firewall-cmd

`firewall-cmd` permite configurar reglas de firewall de manera flexible y en tiempo real. Algunas de las operaciones más comunes incluyen:

- **Verificar el estado del firewall:**

```
firewall-cmd --state
```

- **Listar las reglas activas:**

```
firewall-cmd --list-all
```

- **Abrir un puerto específico (ejemplo: puerto 80 TCP):**

```
firewall-cmd --add-port=80/tcp --permanent
```

- **Cerrar un puerto específico:**

```
firewall-cmd --remove-port=80/tcp --permanent
```

- **Recargar la configuración del firewall para aplicar cambios:**

```
firewall-cmd --reload
```

Administración del Servicio firewalld con systemctl

`systemctl` es una herramienta utilizada para gestionar servicios en sistemas basados en `systemd`, como Rocky Linux. Se puede utilizar para controlar el servicio `firewalld` de la siguiente manera:

- **Verificar si el servicio está activo:**

```
systemctl status firewalld
```

- **Iniciar el servicio si está detenido:**

```
systemctl start firewalld
```

- **Detener el servicio si se necesita desactivarlo temporalmente:**

```
systemctl stop firewalld
```

- **Habilitar el servicio para que se inicie automáticamente al arrancar el sistema:**

```
systemctl enable firewalld
```

- **Deshabilitar el servicio para evitar su inicio automático:**

```
systemctl disable firewalld
```

Verificación de Configuración con nmap

`nmap` es una herramienta de escaneo de redes que permite verificar los puertos abiertos y configuraciones de firewall. Se puede usar para comprobar la efectividad de las reglas de firewall aplicadas. Algunos ejemplos de uso incluyen:

- **Escanear los puertos abiertos de una dirección IP específica:**

```
nmap <direccion_ip>
```

- **Escanear puertos específicos (ejemplo: puertos 22 y 80):**

```
nmap -p 22,80 <direccion_ip>
```

- **Detectar el sistema operativo del host analizado:**

```
nmap -O <direccion_ip>
```

Estas herramientas permiten una administración avanzada del firewall en Rocky Linux, garantizando la seguridad de la red y el control del tráfico de red en el sistema.

3.3.3. Ejercicio Opcional

Como caso práctico, partiendo de una MV con la configuración base descrita en el apartado 2, el alumno/a deberá ser capaz de instalar un servidor de HTTP, Apache o Nginx, y habilitar/deshabilitar su acceso por Firewall.

Para ello, instalará el servidor web de su elección y modificará la home page para mostrar un mensaje: 'Bienvenidos a la web de <Nombre y Apellidos del alumno/a> en Prácticas ISE'.

El servicio web debe estar accesible en la servidor (MV) en el puerto por defecto (80) usando un navegador web convencional corriendo en el anfitrión (Host).

Un escaneo de puertos sobre el servidor solo debe mostrar como accesibles los puerto web y ssh.

Solución

Para la resolución de este ejercicio vamos a seguir los siguientes pasos:

1. Instalamos el servidor web Apache con el comando `sudo dnf install -y httpd`. También tenemos la opción de instalar Nginx con el comando `sudo dnf install nginx- y`. *Tenemos libre elección de servidor web, en mi caso será Apache.*
2. Debemos de activar este servicio con el comando:
 - Apache: `sudo systemctl enable httpd --now`
 - Nginx: `sudo systemctl enable nginx --now`
 - Debemos de verificar que el servicio esta activo con los comandos:
 - `sudo systemctl status httpd # Para Apache`
 - `sudo systemctl status nginx # Para Nginx`
3. Modificación de la página principal.
Debemos de personalizar la página de inicio de nuestra web para que muestre el mensaje que se nos pide en el enunciado.
 - Apache: `echo "Bienvenidos a la web de <Nombre y Apellidos> en Prácticas ISE" | sudo tee /var/www/html/index.html`
 - Nginx: `echo "Bienvenidos a la web de <Nombre y Apellidos> en Prácticas ISE" | sudo tee /usr/share/nginx/html/index.html`
4. Configuración del Firewall.
Para permitir el acceso a la web, es necesario abrir el puerto 80 en el firewall. Para ello, ejecutamos los siguientes comandos:
 - `sudo firewall-cmd --add-service=http --permanent`
 - `sudo firewall-cmd --reload`
 - Para verificar que el puerto esta abierto: `sudo firewall-cmd --list-all`

5. Acceder desde el Navegador.

Si todo está configurado correctamente, se visualizará el mensaje personalizado³(Ver Figura 3.25).



Figura 3.25: Acceso a la web desde el navegador

6. Restricción de puertos con Firewall.

Para asegurar que solo los puertos web (80) y SSH (22) estén accesibles, se deben eliminar otras reglas de firewall:

- `sudo firewall-cmd --list-services --permanent4`
- `sudo firewall-cmd --remove-service=<nombre> --permanent`
- `sudo firewall-cmd --reload`
- Extra.

Como extra podemos ver los puertos que están abiertos con el comando `sudo firewall-cmd --list-ports --permanent` y cerramos el puerto con `sudo firewall-cmd --remove-port=xxxx/tcp --permanent`, donde xxxx es el puerto que queremos cerrar. Otra forma de solo tener dos puertos abiertos es con los comandos:

- `sudo firewall-cmd --add-service=http --permanent`
- `sudo firewall-cmd --add-service=ssh --permanent`
- Y de nuevo recargamos y listamos todos los puertos.

³Para saber a que ip acceder, podemos usar el comando `ip a` y coger la correspondiente a `enp0s8`.

⁴Con este comando podemos ver los servicios que están activos en el firewall, desactivamos todos, a excepción de http y ssh.

3.4. SSH

3.4.1. Administración remota con SSH y criptografía asimétrica

La administración remota es una tarea fundamental en la gestión de servidores. Una de las herramientas más utilizadas para este propósito es **SSH** (Secure Shell), que permite acceder y administrar un sistema de manera segura a través de una red. Es importante destacar que **SSH** puede referirse tanto a un cliente como a un servicio.

Instalación y configuración de SSH

Para instalar y habilitar el servicio SSH en un sistema basado en Rocky Linux, se utilizan los siguientes comandos:

```
1 sudo dnf install -y openssh-server
2 sudo systemctl enable --now sshd
```

El servicio `sshd` es el demonio que permite las conexiones SSH entrantes. Su configuración se encuentra en el archivo:

```
1 /etc/ssh/sshd_config
```

Al editar este archivo, es posible modificar diversas opciones de seguridad, como:

- **Deshabilitar el acceso como root por contraseña:** Para mejorar la seguridad, se recomienda restringir el acceso directo de root:

```
1     PermitRootLogin no
2
```

- **Cambio de puerto predeterminado:** SSH por defecto usa el puerto 22, pero puede cambiarse para reducir ataques automatizados:

```
1     Port 2222
2
```

Después de modificar la configuración, es necesario reiniciar el servicio:

```
1 sudo systemctl restart sshd
```

Configuración del firewall para SSH

Si se cambia el puerto de SSH, es necesario actualizar la configuración del firewall:

```
1 sudo firewall-cmd --add-port=2222/tcp --permanent
2 sudo firewall-cmd --reload
```

Criptografía simétrica y asimétrica en SSH

Criptografía simétrica es un método en el que la misma clave se usa tanto para cifrar como para descifrar los datos. Es rápida pero menos segura para la comunicación remota, ya que ambas partes deben compartir la clave de forma segura.

Criptografía asimétrica, utilizada en SSH, emplea un par de claves: una clave pública y una clave privada. El cliente genera un par de claves y comparte la clave

pública con el servidor, lo que permite autenticarse sin necesidad de enviar una contraseña.

Para generar un par de claves en SSH, se usa:

```
1 ssh-keygen -t rsa -b 4096
```

Y para copiar la clave pública al servidor:

```
1 ssh-copy-id usuario@servidor
```

Esto permite autenticarse sin necesidad de contraseña, mejorando la seguridad y facilitando la automatización de tareas en servidores remotos.

3.4.2. Ejercicio Opcional

Partiendo de un servidor base configurado siguiendo las indicaciones del apartado 2, el alumno/a modificará servicio SSHD para que, en lugar del puerto por defecto (22), se ejecute en un puerto alternativo de un valor mayor a 1024. Se recomienda que consulte la lista de puertos reconocidos por el sistema en `/etc/ports` para evitar emplear un puerto que ya tenga una aplicación predefinida.

Se concederá acceso remoto por llave pública a un usuario de su elección. El ejercicio se validará ejecutando un comando de forma remota sobre el servidor SSH con la nueva configuración. El comando presentará el contenido completo (incluido ficheros y directorios ocultos) con del directorio home del usuario remoto empleado en la conexión. Para ello, desde el ordenador anfitrión (o una MV distinta a la que se va a acceder) se empleará `ssh` sin terminal remoto y sin contraseña, pasando como único como parámetro el comando a ejecutar.

Solución

Modificación del servicio SSHD y acceso remoto por llave pública

En este ejercicio, se modificará la configuración del servicio SSH para que utilice un puerto alternativo y se habilitará el acceso remoto mediante autenticación con clave pública.

Paso 1: Verificar puertos disponibles

Antes de modificar la configuración de SSH, es recomendable consultar la lista de puertos reconocidos por el sistema para evitar conflictos con otros servicios. Para ello, se puede inspeccionar el archivo:

```
1 cat /etc/services | less
```

Se debe elegir un puerto mayor a 1024 que no esté en uso.

Paso 2: Modificar la configuración de SSH

Editar el archivo de configuración de SSH con un editor de texto como `'vim'` o `'nano'`:

```
1 sudo nano /etc/ssh/sshd_config
```

Buscar la línea que especifica el puerto (`'#Port 22'`), descomentarla y cambiarla por el número de puerto elegido, por ejemplo:

1 Port 2222

Guardar los cambios y salir del editor.

Solución al error al cambiar el puerto de SSH

Si al cambiar el puerto en la configuración de SSH y reiniciar el servicio se obtiene un error, se deben seguir los siguientes pasos para solucionar el problema:

1. **Verificar errores en la configuración de SSH** Ejecutar el siguiente comando para comprobar si hay errores en el archivo de configuración:

```
1 sudo sshd -t
2
```

Si se detectan errores de sintaxis o configuración en el archivo `/etc/ssh/sshd_config`, se deben corregir antes de continuar.

2. **Confirmar que el puerto está permitido en SELinux (si aplica)** Si el sistema usa SELinux, es necesario permitir el nuevo puerto:

```
1 sudo semanage port -a -t ssh_port_t -p tcp 2222
2
```

Si el comando `semanage` no está disponible, se puede instalar con:

```
1 sudo dnf install polycoreutils-python-utils
2
```

Luego, verificar que el puerto se agregó correctamente con:

```
1 sudo semanage port -l | grep ssh
2
```

3. **Permitir el nuevo puerto en firewalld** Si el sistema usa `firewalld`, se debe agregar el puerto y recargar la configuración:

```
1 sudo firewall-cmd --permanent --add-port=2222/tcp
2 sudo firewall-cmd --reload
3
```

4. **Reiniciar el servicio SSH y comprobar su estado** Una vez realizados los cambios, reiniciar el servicio SSH:

```
1 sudo systemctl restart sshd
2
```

Si el servicio sigue fallando, revisar los logs para obtener más detalles sobre el error:

```
1 sudo journalctl -xeu sshd
2
```

Siguiendo estos pasos, se podrá cambiar el puerto de SSH sin problemas y reiniciar el servicio correctamente.

Paso 3: Ajustar firewalld para permitir conexiones en el nuevo puerto

Si 'firewalld' está en uso, es necesario agregar el nuevo puerto al firewall y eliminar el acceso al puerto 22 si no se desea que permanezca abierto:

```
1 sudo firewall-cmd --permanent --add-port=2222/tcp
2 sudo firewall-cmd --permanent --remove-service=ssh
3 sudo firewall-cmd --reload
```

Paso 4: Reiniciar el servicio SSH

Aplicar los cambios reiniciando el servicio SSH:

```
1 sudo systemctl restart sshd
```

Paso 5: Configurar autenticación por clave pública

Desde el cliente, generar un par de claves SSH si no se tienen:

```
1 ssh-keygen -t rsa -b 4096
```

Copiar la clave pública al servidor(Ver Figura 3.26):

```
1 ssh-copy-id -p 2222 usuario@IP_del_Servidor
```

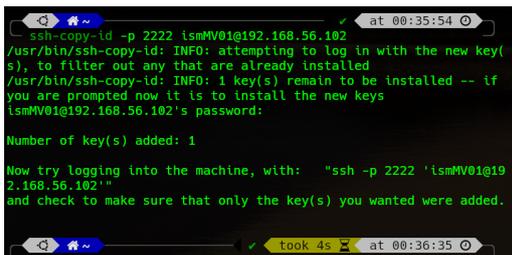


Figura 3.26: SSH en mi máquina

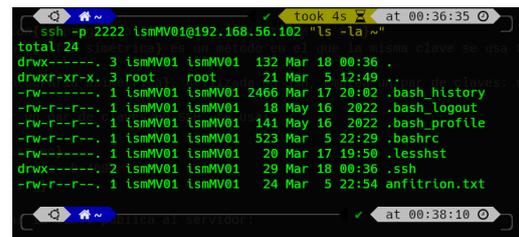


Figura 3.27: SSH final

Paso 6: Validar la conexión sin contraseña

Para comprobar que el acceso es correcto, se ejecutará un comando remoto en la máquina virtual desde otro equipo o una máquina anfitriona. Se listará el contenido completo del directorio home del usuario remoto, incluidos los archivos ocultos (Ver Figura 3.27):

```
1 ssh -p 2222 usuario@IP_del_Servidor "ls -la ~"
```

Si todo está configurado correctamente, el comando mostrará los archivos del directorio home sin requerir contraseña.

3.5. Automatización de la configuración con Ansible

3.5.1. Introducción a Ansible

Ansible es una herramienta de automatización que permite gestionar configuraciones y despliegues en múltiples sistemas de manera sencilla y eficiente. Su funcionamiento se basa en la ejecución de comandos remotos a través de SSH y utiliza Python como lenguaje de scripting, lo que facilita su integración en la mayoría de las distribuciones Linux.

Comandos Ad-Hoc en Ansible

Los comandos ad-hoc en Ansible permiten ejecutar tareas únicas y rápidas en uno o varios nodos gestionados sin necesidad de crear un playbook. Se utilizan a través de la herramienta de línea de comandos `ansible` y son ideales para operaciones puntuales, como reiniciar servicios, copiar archivos o instalar paquetes. Aunque son fáciles de usar, no son reutilizables ni idempotentes, por lo que para tareas repetitivas o complejas se recomienda el uso de playbooks. [2]

Módulos de Ansible

Ansible cuenta con una amplia variedad de módulos que encapsulan tareas específicas, como gestionar usuarios, instalar paquetes o manejar servicios. Estos módulos permiten abstraer la complejidad de las operaciones subyacentes y proporcionan una interfaz consistente para interactuar con diferentes sistemas y servicios. La documentación oficial de Ansible ofrece un índice completo de todos los módulos disponibles, organizados por categorías, facilitando la búsqueda y selección del módulo adecuado para cada tarea. [3]

Playbooks en Ansible

Los playbooks son el núcleo de Ansible y permiten definir configuraciones, despliegues y orquestaciones de manera declarativa y reproducible. Están escritos en formato YAML y constan de una o más 'plays', cada una de las cuales especifica un conjunto de tareas a ejecutar en un grupo de hosts. Los playbooks pueden:

- Declarar configuraciones de sistemas.
- Orquestar procesos manuales en múltiples máquinas en un orden definido.
- Ejecutar tareas de forma síncrona o asíncrona.

Al escribir playbooks, es esencial comprender su sintaxis y estructura para garantizar su correcta ejecución y mantenimiento. [4]

Buenas Prácticas de Seguridad

Es una práctica recomendada en Ansible evitar el uso del usuario 'root' para acceder a los nodos gestionados. En su lugar, se aconseja crear un usuario con privilegios adecuados que se conecte mediante SSH utilizando autenticación con clave pública y que pueda ejecutar comandos privilegiados sin necesidad de ingresar una

contraseña adicional. Para ello, es necesario configurar correctamente el comando `sudo` en el servidor, permitiendo que el usuario tenga los permisos necesarios para las operaciones requeridas.

Administración del Inventario

El inventario en Ansible es un archivo que define los hosts y grupos de hosts sobre los cuales se ejecutarán las tareas. Es fundamental para organizar y gestionar los nodos de manera eficiente. Ansible permite utilizar inventarios estáticos, definidos en archivos de texto, o dinámicos, generados por scripts que consultan fuentes externas. Una correcta administración del inventario facilita la segmentación de los hosts y la aplicación de configuraciones específicas a diferentes grupos de servidores.

Ejecución de Comandos Ad-Hoc por CLI

Además de los playbooks, Ansible permite la ejecución de comandos ad-hoc directamente desde la línea de comandos. Esta funcionalidad es útil para tareas rápidas y únicas, como verificar el estado de un servicio o copiar un archivo a múltiples servidores. Aunque los comandos ad-hoc son prácticos, para tareas más complejas o repetitivas se recomienda encapsular las operaciones en playbooks para asegurar la reproducibilidad y el mantenimiento del código.

Configuración de Servidores con Playbooks

Los playbooks permiten automatizar la configuración de servidores de manera declarativa. Al definir el estado deseado de los sistemas, Ansible se encarga de aplicar las tareas necesarias para alcanzar ese estado. Esto incluye la instalación y configuración de paquetes, la gestión de archivos y servicios, y la implementación de políticas de seguridad. La utilización de playbooks facilita la gestión coherente y reproducible de las configuraciones en múltiples servidores, reduciendo errores y mejorando la eficiencia operativa.

3.5.2. Ejercicio Obligatorio

El ejercicio versa sobre la configuración de servidores empleando Ansible playbooks. Se valorará la estructuración y claridad del código, el uso de parámetros para facilitar la reutilización de los playbooks, el uso de comentarios, el uso de variables, la organización de los artefactos, el uso de convenciones de nombrado de Ansible y de Yaml y, aunque escapa al objetivo de este ejercicio, el posible uso de recursos para reutilización de artefactos, como los Ansible Roles.

Partiendo de dos servidores, configurados de acuerdo a los requerimientos del apartado 2, debe modificarlos para que sea posible el acceso remoto del usuario root empleando contraseña (el acceso con contraseña está desactivado por defecto en la instalación de Rocky).

A continuación, realizará la siguiente configuración en los dos servidores empleando un playbook:

1. Crear un nuevo usuario llamado “admin” que pueda ejecutar comandos privilegiados sin contraseña.

2. Dar acceso por SSH al usuario “admin” con llave pública.
3. Crear el grupo “wheel” (si no existe) y permitir a sus miembros ejecutar sudo.
4. Añadir una lista variable de usuarios (se proporcionará un ejemplo con al menos dos), añadiéndolos al grupo “wheel” y concediéndoles acceso por SSH con llave pública.
5. Deshabilitar el acceso por contraseña sobre SSH para el usuario root.

Los servidores anteriormente configurados son ahora administrables mediante Ansible empleando el usuario “admin”. Ponga a prueba esta configuración con los siguientes cambios/playbooks:

1. Modifique convenientemente el inventario para el uso del nuevo usuario “admin”.
2. Uno de los servidores se empleará para correr Apache Httpd y el otro Nginx. Modifique el inventario de forma conveniente para realizar correctamente su administración.
3. Desarrolle un playbook para implementar los requerimientos del ejercicio 4.1.1 en los dos servidores, instalando en cada caso Apache Httpd o Nginx según la configuración del inventario.

Todos los archivos necesarios para la ejecución de los Playbooks (playbooks, inventario, variables, scripts, ...) deben estar localizados en un directorio (con posibles subdirectorios). Junto a los archivos propios de Ansible, debe proporcionar dos scripts (por ejemplo, `iniciarNodosManejados.sh` y `configurarWebServers.sh`) para la ejecución de los playbooks con todos los parámetros necesarios.

```
project_directory/  
|-- inventory.ini  
|-- group_vars/  
|   |-- all.yml  
|   |-- apache.yml  
|   '-- nginx.yml  
|-- playbooks/  
|   |-- initial_setup.yml  
|   '-- webserver_setup.yml  
|-- scripts/  
|   |-- iniciarNodosManejados.sh  
|   '-- configurarWebServers.sh  
'-- files/  
    |-- admin_ssh_key.pub  
    '-- users_ssh_keys/  
        |-- user1.pub  
        '-- user2.pub
```

Listing 3.1: Estructura del directorio para los playbooks y scripts

Para ver la resolución detallada de este ejercicio y del resto, se aconseja revisar la parte del blog destinada a los ejercicios de Ansible, donde se explican los pasos y se proporciona el código necesario para completar cada tarea.

Para una información más detallada se debe de consultar el directorio de FicheroEjercicio y los correspondientes al Bloque1.

Capítulo 4

Bloque 2

4.1. Prerequisitos

4.1.1. ¿Qué es un contenedor?

Un contenedor es una unidad estandarizada de software que empaqueta código y todas sus dependencias para que una aplicación se ejecute de forma rápida, confiable y consistente en diferentes entornos. Es una forma ligera, portátil y aislada de ejecutar procesos o aplicaciones.

Componentes clave de un contenedor

- **Aplicación principal:** El binario o script que se quiere ejecutar.
- **Dependencias:** Librerías, módulos, herramientas del sistema necesarias.
- **Sistema de archivos aislado:** Un entorno controlado, consistente y separado del sistema operativo anfitrión.
- **Red y procesos aislados:** Gracias a tecnologías como *cgroups* y *namespaces* del kernel de Linux.
- **Capacidad de ser portado:** Se comporta igual en desarrollo, pruebas o producción.

Creación y gestión de contenedores

Para crear y gestionar contenedores, se utilizan herramientas como:

Docker:

- Permite construir imágenes (plantillas de contenedor) y lanzar contenedores a partir de ellas.
- Una *imagen Docker* es como una instantánea de un sistema preconfigurado.
- Un *contenedor Docker* es una instancia en ejecución de esa imagen.

Podman:

- Alternativa moderna a Docker, compatible con sus comandos, pero no necesita un *daemon* (demonio) en segundo plano.
- Cada contenedor se ejecuta como un proceso del usuario, lo cual es más seguro en entornos multiusuario.
- Soporta *rootless containers* (contenedores sin privilegios de administrador).

¿Qué es Docker Compose?

Docker Compose es una herramienta que permite definir y ejecutar múltiples contenedores Docker mediante un archivo YAML (`docker-compose.yml`). Es ideal para sistemas distribuidos o aplicaciones que requieren múltiples servicios, como una aplicación web con base de datos y servidor de caché.

Ejemplo básico:

```

1 version: '3'
2 services:
3   web:
4     image: nginx
5     ports:
6       - "80:80"
7   db:
8     image: postgres
9     environment:
10      POSTGRES_PASSWORD: example

```

Este archivo lanza dos contenedores: uno con Nginx y otro con PostgreSQL, conectados en una red compartida automáticamente gestionada por Docker Compose.

Diferencias entre Docker, Docker Compose y Podman

Característica	Docker	Docker Compose	Podman
Motor de contenedores	Sí	Usa Docker	Sí
Ejecuta múltiples contenedores	No directamente	Sí, orquestación básica	Sí, con <code>podman-compose</code>
Requiere <code>daemon</code>	Sí	Sí (usa Docker)	No
Rootless	Limitado	No	Sí, por diseño
Compatible con Dockerfiles	Sí	Sí	Sí

Resumen conceptual

Un contenedor no es una máquina virtual. No tiene su propio kernel ni simula hardware. Comparte el kernel del sistema anfitrión, lo que lo hace mucho más ligero y rápido, ideal para microservicios, DevOps y CI/CD.

Docker Compose y Podman son formas de gestionar contenedores: la primera, muy útil para múltiples servicios coordinados; la segunda, más flexible, más segura y sin necesidad de *daemon*, orientada a usuarios avanzados y producción segura.

4.1.2. Ejemplos

- Crear un contenedor con Docker: `docker run -d -p 80:80 nginx`
- Crear un contenedor con Podman: `podman run -d -p 80:80 nginx`
- Usar Docker Compose para lanzar múltiples servicios: `docker-compose up`

Comparativa con una MV

Contenedor vs Máquina Virtual

Tanto los contenedores como las máquinas virtuales aíslan aplicaciones del sistema anfitrión, pero lo hacen de formas muy diferentes.

Similitudes: Ambos permiten ejecutar aplicaciones en entornos separados, evitando que interfieran con el sistema principal.

Característica	Máquina Virtual	Contenedor
Sistema Operativo	Incluye su propio sistema operativo	Comparte el kernel del anfitrión
Peso	Pesado: varios GBs	Ligero: pocos MBs
Arranque	Lento (minutos)	Rápido (segundos o menos)
Consumo de recursos	Alto: CPU, RAM, disco	Bajo y eficiente
Virtualización	A nivel de hardware (hipervisor)	A nivel de sistema operativo (namespaces)
Portabilidad	Limitada (por sistema operativo)	Muy alta (misma imagen funciona en múltiples sistemas)
Ideal para	Emular sistemas completos, testing de OS	Desarrollar y desplegar aplicaciones

Diferencias principales:

Analogía: Imagina un edificio:

- Una máquina virtual es como un departamento completo: tiene su propio baño, cocina y sistema eléctrico. Es independiente, pero consume más recursos.
- Un contenedor es como una habitación en un mismo piso: tiene sus propios muebles y decoración (aplicación y dependencias), pero comparte el sistema de agua, luz y estructura (el kernel del sistema operativo).

Conclusión: Un contenedor es similar a una máquina virtual, pero más ligero y rápido porque no incluye un sistema operativo completo. Esto lo hace ideal para desarrollo, pruebas, despliegue continuo y microservicios.

4.1.3. Instalación de Docker y Docker Compose

A continuación, se presenta una guía detallada para instalar Docker y Docker Compose en Ubuntu 24.04, sin utilizar Docker Desktop, lo que resulta en una instalación más ligera y estable.

1. Preparar el sistema

Abra una terminal y actualice el sistema:

```
1 sudo apt update && sudo apt upgrade -y
```

Instale las herramientas necesarias:

```
1 sudo apt install ca-certificates curl gnupg lsb-release -y
```

2. Agregar la clave GPG oficial de Docker

Cree el directorio para almacenar la clave:

```
1 sudo mkdir -p /etc/apt/keyrings
```

Descargue y almacene la clave GPG oficial de Docker:

```
1 curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo  
   gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

3. Agregar el repositorio oficial de Docker

Agregue el repositorio de Docker a la lista de fuentes de APT:

```
1 echo \  
2     "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/  
   keyrings/docker.gpg] \  
3  
4     https://download.docker.com/linux/ubuntu \  
5     $(lsb_release -cs) stable" | \  
6     sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

4. Instalar Docker Engine y Docker Compose

Nota: Es para nuestra máquina (ubuntu).

Actualice los paquetes e instale Docker y Docker Compose:

```
1 sudo apt update  
2 sudo apt install docker-ce docker-ce-cli containerd.io docker-  
   buildx-plugin docker-compose-plugin -y
```

5. Verificar que Docker está funcionando

Ejecute los siguientes comandos para verificar la instalación:

```
1 sudo docker version
2 sudo docker info
```

Si se muestra información sobre Docker, la instalación fue exitosa.

6. (Opcional) Ejecutar Docker sin sudo

Para permitir que Docker se ejecute sin necesidad de usar `sudo`, ejecute:

```
1 sudo usermod -aG docker $USER
```

Acto seguido, cierre la sesión y vuelva a iniciarla para que los cambios surtan efecto.

```
1 sudo systemctl restart docker
```

Nota: Cierre la sesión y vuelva a iniciarla (o reinicie el sistema) para que los cambios surtan efecto.

4.1.4. Guía rápida de prueba

1. Probar Docker

Ejecute el siguiente comando para probar Docker:

```
1 docker run hello-world
```

Debería aparecer un mensaje indicando que Docker está instalado correctamente.

2. Probar Docker Compose

Cree una carpeta de prueba y acceda a ella:

```
1 mkdir docker-prueba && cd docker-prueba
```

Cree un archivo `docker-compose.yml`:

```
1 nano docker-compose.yml
```

Pegue el siguiente contenido en el archivo:

```
1 version: "3.8"
2 services:
3   web:
4     image: nginx
5     ports:
6     - "8080:80"
```

Guarde y cierre el archivo. Luego, ejecute el servicio:

```
1 docker compose up -d
```

Abra un navegador y acceda a `http://localhost:8080`. Debería visualizar la página de bienvenida de NGINX.

Para detener el servicio, ejecute:

```
1 docker compose down
```

Con estos pasos, Docker y Docker Compose estarán instalados y funcionando correctamente.

4.1.5. Instalación de Docker Engine en Rocky Linux 9 (o CentOS Stream 9)

Esta guía es ideal para máquinas virtuales (MV) sin entorno gráfico.

Paso 1: Preparar el sistema

Actualice el sistema con los siguientes comandos:

```
1 sudo dnf update -y
2 sudo dnf upgrade -y
```

Instale los paquetes necesarios:

```
1 sudo dnf install -y dnf-utils device-mapper-persistent-data
   lvm2
```

Paso 2: Agregar el repositorio oficial de Docker

Agregue el repositorio oficial de Docker con el siguiente comando:

```
1 sudo dnf config-manager --add-repo https://download.docker.com/
   linux/centos/docker-ce.repo
```

Nota: Docker no ofrece repositorios específicos para Rocky Linux, pero el de CentOS funciona perfectamente.

Paso 3: Instalar Docker Engine

Instale Docker Engine y sus componentes:

```
1 sudo dnf install -y docker-ce docker-ce-cli containerd.io
   docker-buildx-plugin docker-compose-plugin
```

Paso 4: Habilitar y arrancar el servicio Docker

Habilite y arranque el servicio Docker con los siguientes comandos:

```
1 sudo systemctl enable docker
2 sudo systemctl start docker
```

Paso 5: Verificar la instalación

Verifique que Docker se instaló correctamente ejecutando:

```
1 docker --version
```

Debería mostrar una salida similar a:
Docker version 24.x.x, build xxxxxxxx

Paso 6: Permitir el uso de Docker a usuarios no root

Para permitir que Docker se ejecute sin necesidad de usar `sudo`, siga estos pasos:

1. Cree el grupo `docker` (si no existe):

```
1 sudo groupadd docker
2
```

2. Añada su usuario al grupo:

```
1 sudo usermod -aG docker $USER
2
```

3. Cierre la sesión y vuelva a iniciarla para aplicar los cambios.

Pruebe que Docker se ejecuta sin `sudo` ejecutando:

```
1 docker run hello-world
```

4.2. Microservicios

La arquitectura de microservicios es un enfoque de desarrollo de software en el que una aplicación se descompone en un conjunto de pequeños servicios independientes que se comunican entre sí mediante APIs o sistemas de mensajería como RabbitMQ, Kafka o Qpid. Cada microservicio está diseñado para encargarse de una funcionalidad específica del sistema y puede desarrollarse, desplegarse y escalarse de manera autónoma.

Este modelo ha ganado popularidad con el auge de los contenedores, ya que facilita el despliegue y la gestión de estos servicios de forma independiente. Entre sus principales ventajas destacan:

- **Flexibilidad:** Cada servicio puede utilizar su propio stack tecnológico, adaptándose mejor a sus necesidades específicas.
- **Escalabilidad:** Permite escalar verticalmente (mejorando los recursos de un servicio) u horizontalmente (añadiendo instancias del servicio según la demanda).

Para más información sobre esta arquitectura, se recomienda consultar el artículo de Martin Fowler: [Microservices](#).

4.3. Ejercicio Evaluable. Ejecutar “Hello World”

Instale Docker en su ordenador anfitrión o en una MV y ejecute el contenedor “Hello World” disponible en: .

Para ello debemos de ejecutar los siguientes comandos:

- 1: `sudo docker version`: para ver si docker está instalado correctamente.
- 2: `sudo docker run hello-world`: para ejecutar el contenedor. (Ver figura 4.1)

A terminal window showing the execution of the command 'sudo docker run hello-world'. The output displays a 'Hello from Docker!' message, a confirmation that the installation is working, and a list of four steps taken by Docker to generate the message. It also provides instructions on how to run an Ubuntu container and links to Docker's documentation and image hub.

```
sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Figura 4.1: Contenedor Hello World

4.4. BenchMarks

4.4.1. ¿Qué es un Benchmark?

Un *benchmark* es una prueba diseñada para medir el rendimiento de un sistema, componente o servicio informático. Estas pruebas pueden evaluar diversos aspectos, como velocidad, eficiencia, capacidad de respuesta o consumo de recursos. Existen múltiples tipos de *benchmarks*, cada uno enfocado en un área específica del sistema.

Por ejemplo, para evaluar el rendimiento de un servidor DNS (que traduce nombres de dominio a direcciones IP), se pueden utilizar herramientas como **NameBench** o **GRC's DNS Benchmark**, las cuales están diseñadas con pruebas estándar para este tipo de servicio.

4.4.2. ¿Por qué crear un Benchmark propio?

Aunque existen herramientas predefinidas para realizar *benchmarks*, en ocasiones es necesario programar uno propio para analizar parámetros específicos que no estén cubiertos por las herramientas existentes. Para ello, es importante considerar los siguientes elementos:

- **Objetivo del Benchmark:** Definir claramente qué se desea medir, como latencia, *throughput*, uso de CPU, etc.
- **Métricas:** Establecer las medidas que se utilizarán para evaluar el rendimiento. Esto incluye:
 - Las unidades de medida (por ejemplo, milisegundos, MB/s).
 - Las variables a observar (por ejemplo, tiempo de respuesta, carga del sistema).
 - El método para calcular los resultados finales.
- **Instrucciones de uso:** Especificar cómo ejecutar el *benchmark*, el entorno requerido y los parámetros necesarios.
- **Ejemplo de uso y análisis de resultados:** Incluir una prueba real, interpretar los datos obtenidos y extraer conclusiones relevantes.

4.4.3. OpenBenchmarking y Phoronix Test Suite (PTS)

OpenBenchmarking

OpenBenchmarking es un repositorio de *benchmarks* de código abierto que permite a los usuarios utilizar, modificar o crear nuevas pruebas basadas en las existentes. Es una excelente fuente de inspiración para desarrollar *benchmarks* personalizados.

Phoronix Test Suite (PTS)

Phoronix Test Suite (PTS) es una plataforma asociada a *OpenBenchmarking* que facilita la ejecución de *benchmarks*. Es una herramienta versátil y popular para realizar pruebas de rendimiento en sistemas Linux, Windows o incluso en máquinas virtuales (MVs). Entre sus características destacan:

- Amplia variedad de pruebas disponibles que se pueden ejecutar directamente desde su entorno.
- Integración con *OpenBenchmarking*, lo que permite comparar resultados con otras máquinas.
- Compatibilidad con múltiples entornos, incluyendo contenedores Docker.

Instalación de Phoronix Test Suite (PTS)

La instalación de *PTS* varía según el entorno utilizado:

- En sistemas Debian/Ubuntu, se pueden usar paquetes precompilados.
- En máquinas virtuales con Rocky Linux, se recomienda el instalador universal para Linux.
- En Windows, también se ofrece soporte de instalación.
- Para las prácticas, la opción más recomendada es utilizar contenedores Docker.

Enlaces útiles

A continuación, se presentan enlaces relevantes para explorar más sobre *benchmarks* y herramientas asociadas:

- Software relacionado con *benchmarks* en Linux: sourceforge.net/directory/linux/?q=benchmark
- Repositorio de OpenBenchmarking: openbenchmarking.org
- Sitio web de Phoronix Test Suite: phoronix-test-suite.com
- Página de descargas de PTS: phoronix-test-suite.com/?k=downloads
- Imagen Docker recomendada para las prácticas: hub.docker.com/r/phoronix/pts/

4.4.4. Ejercicio Opcional. Ejecutar un Benchmark

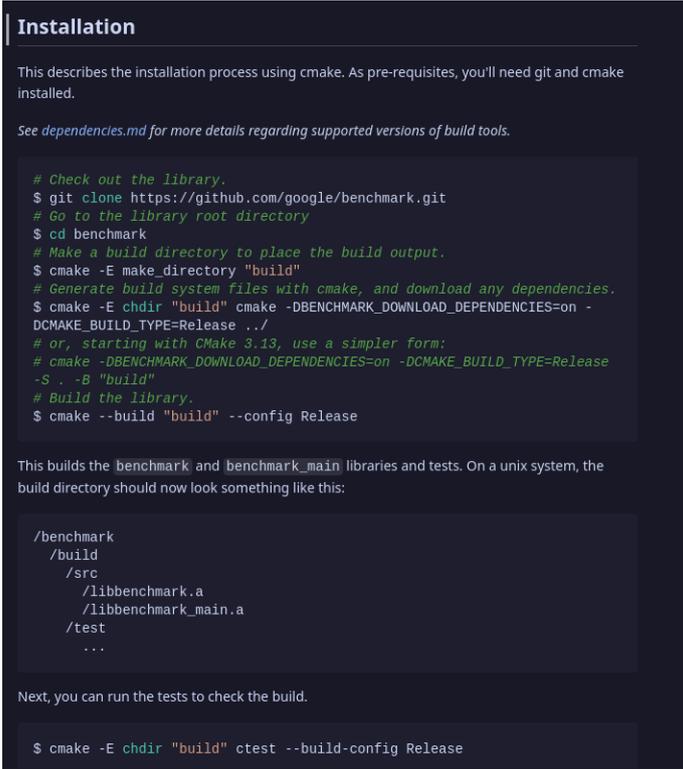
El alumno/a debe ser capaz de utilizar *Phoronix Test Suite* para:

- Descargar, instalar y ejecutar un *benchmark* de su elección.
- Almacenar y recuperar los resultados de múltiples ejecuciones del *benchmark*.
- Explicar el objetivo del *benchmark* y de los resultados obtenidos.

Solución con un benchmark distinto

En mi caso en el propio enlace que nos ofrece la guía¹ he optado por instalar el que benchmark que tiene como nombre benchmark² y estoy siguiendo los pasos que aparecen en el fichero *Readme.md*³.

Los pasos que he seguido se puede ver en la figura 4.2.



```
Installation

This describes the installation process using cmake. As pre-requisites, you'll need git and cmake installed.

See dependencies.md for more details regarding supported versions of build tools.

# Check out the library.
$ git clone https://github.com/google/benchmark.git
# Go to the library root directory
$ cd benchmark
# Make a build directory to place the build output.
$ cmake -E make_directory "build"
# Generate build system files with cmake, and download any dependencies.
$ cmake -E chdir "build" cmake -DBENCHMARK_DOWNLOAD_DEPENDENCIES=on -
DCMAKE_BUILD_TYPE=Release ./
# or, starting with CMake 3.13, use a simpler form:
$ cmake -DBENCHMARK_DOWNLOAD_DEPENDENCIES=on -DCMAKE_BUILD_TYPE=Release
-S . -B "build"
# Build the library.
$ cmake --build "build" --config Release

This builds the benchmark and benchmark_main libraries and tests. On a unix system, the
build directory should now look something like this:

/benchmark
/build
  /src
  /libbenchmark.a
  /libbenchmark_main.a
  /test
  ...

Next, you can run the tests to check the build.

$ cmake -E chdir "build" ctest --build-config Release
```

Figura 4.2: Instalación de benchmark

Y el resultado del comando de ejecución de los tests lo puede ver en la figura 4.3.

¹<https://sourceforge.net/directory/linux/?q=benchmark>

²<https://sourceforge.net/projects/benchmark.mirror/>

³Lo estoy instalando en mi máquina anfitrión.

```

Passed    0.00 sec
Start 80: profiler_manager_gtest
80/82 Test #80: profiler_manager_gtest .....
Passed    0.01 sec
Start 81: benchmark_setup_teardown_cb_types_gtest
81/82 Test #81: benchmark_setup_teardown_cb_types_gtest ....
Passed    0.01 sec
Start 82: memory_results_gtest
82/82 Test #82: memory_results_gtest .....
Passed    0.01 sec

100% tests passed, 0 tests failed out of 82

Total Test time (real) = 11.14 sec

```

Figura 4.3: Resultado de la ejecución del benchmark

A continuación, instalamos las librerías globales y probamos con el fichero básico que nos proporcionan, compilando y demás, nos da la salida que podemos ver en la figura 4.4.

```

./mybenchmark
2025-04-15T09:58:38+02:00
Running ./mybenchmark
Run on (12 X 4603 MHz CPU s)
CPU Caches:
  L1 Data 32 KiB (x6)
  L1 Instruction 32 KiB (x6)
  L2 Unified 512 KiB (x6)
  L3 Unified 16384 KiB (x1)
Load Average: 0.70, 1.18, 1.21
***WARNING*** CPU scaling is enabled, the benchmark real time
measurements may be noisy and will incur extra overhead.
-----
Benchmark                Time          CPU    Iterations
-----
BM_StringCreation        9.70 ns       9.70 ns    70462712
BM_StringCopy            15.8 ns      15.8 ns    42622026

```

Figura 4.4: Resultado de la ejecución del benchmark

Solución usando el benchmark que se especifica

Para ello me he descargado el zip de la web y como estoy en Linux he ejecutado el comando `sudo ./install.sh` y me ha instalado el programa en la ruta `/usr/bin/phoronix-test-suite`.

```

sudo ./install.sh
-e
Phoronix Test Suite Installation Completed

Executable File: /usr/bin/phoronix-test-suite
Documentation: /usr/share/doc/phoronix-test-suite/
Phoronix Test Suite Files: /usr/share/phoronix-test-suite/

```

Figura 4.5: Instalación de Phoronix Test Suite

Debemos de leer el Readme para una prueba.

En base a “*phoronix-test-suite benchmark smallpt to run a simple CPU test profile*” introducimos el comando para un primer testeo.

Donde el resultado que nos da:

```
resultsFirstAttempt
results
```

```
results:
```

```
Processor: AMD Ryzen 5 7535HS @ 4.60GHz (6 Cores
/ 12 Threads), Motherboard: ASUS TUF Gaming
A15 FA506NC_FA506NC FA506NC v1.0 (FA506NC.308
BIOS), Chipset: AMD 17h-19h PCIe Root Complex,
Memory: 2 x 8GB DDR5-4800MT/s Samsung
M425R1GB4PB0-CWMOL, Disk: 512GB SAMSUNG
MZVL8512HELU-00BTW + 1000GB KINGSTON
SNV3S1000G, Graphics: ASUS NVIDIA GeForce RTX
3050 Mobile, Audio: NVIDIA Device 2291,
Network: Realtek RTL8111/8168/8211/8411 +
Realtek RTL8852BE PCIe 802.11ax
```

```
OS: Ubuntu 24.04, Kernel: 6.11.0-21-generic (
x86_64), Display Server: X Server, Compiler:
GCC 13.3.0, File-System: ext4, Screen
Resolution: 1920x1080
```

```
Smallpt 1.0
```

```
Global Illumination Renderer; 128 Samples
```

```
Seconds < Lower Is Better
```

```
results . 15.48
```

```
|=====
```

4.4.5. Resumen: Phoronix Test Suite en Máquina Local vs Docker

El *Phoronix Test Suite* puede presentar problemas de rendimiento o fallos en una máquina local debido a dependencias faltantes, configuraciones incorrectas o conflictos en el entorno del sistema operativo. En contraste, al ejecutarlo en un contenedor Docker, se utiliza un entorno limpio y preconfigurado, optimizado por los desarrolladores para funcionar de manera eficiente.

Comparativa entre Máquina Local y Docker

Caso Específico: Benchmark smallpt

El *smallpt* es un *benchmark* dependiente de la CPU. Problemas como *throttling*, gestión térmica deficiente o extensiones PHP faltantes pueden ralentizar su ejecución

Aspecto	Máquina Local	Contenedor Docker
PHP y extensiones	Puede faltar GD, bzip2, sqlite3, etc.	Todo preinstalado y configurado
Permisos y acceso a GPU	Problemas con <code>/dev/dri</code> u otros permisos	Aislado, puede no usar GPU directamente
Entorno gráfico	Falta de X11 puede romper tests gráficos	Usa <i>fallback</i> o modo <i>headless</i>
Dependencias del sistema	Conflictos o paquetes rotos	Imagen oficial limpia y probada
Compatibilidad de librerías	Incompatibilidades posibles	Versiones probadas y compatibles

Cuadro 4.1: Comparativa entre ejecución local y en Docker

en una máquina local. En Docker, estos problemas se mitigan gracias al entorno controlado.

Ventajas de Docker

- Entorno aislado y preconfigurado.
- Independencia de configuraciones locales.
- Fácil reinicio y limpieza del entorno.
- Uso de versiones probadas por los desarrolladores.

En resumen, ejecutar *Phoronix Test Suite* en Docker garantiza mayor estabilidad y rendimiento, eliminando problemas derivados del entorno local.

Realizando diversas pruebas vemos que el comando tarda demasiado y con `htop` vemos que no se queda colgado pero tarda demasiado tiempo, por ende, he decidido ejecutarlo en contenedores que es como se recomienda en prácticas.

Para ello ejecutamos los siguientes comandos:

1. `docker pull phoronix/pts`: para descargar la imagen de Phoronix Test Suite⁴.
2. `docker run -it --rm phoronix/pts`: para ejecutar el contenedor interactivo.
3. `phoronix-test-suite benchmark smallpt`: para ejecutar el benchmark.

De manera que una vez que estamos dentro de la shell interactiva del docker podemos ejecutar los comandos del docker.

Los pasos a seguir para los tests son (dentro del docker de phoronix con la shell interactiva):

1. Listamos los disponibles usando el comando `list-available-test`
2. `install <test>`

⁴El comando es de la <https://www.phoronix-test-suite.com/?k=downloads>.

3. run <test>

En mi caso con he instalado el test php. El resultado podemos verlo en la terminal (Ver figura 4.6). Además tenemos la opción de subirlo y poder consultarlo de manera online, en mi caso es <https://openbenchmarking.org/result/2504152-NE-RESULTSPH66>.

```

Deviation: 29.71%
Samples: 12

Comparison of 1,269 OpenBenchmarking.org samples since 23 November 2018 to 15 March; median result: 0.53 Seconds. Box plot of samples:
[-----*-----#####!#* ]
  This Result (30th Percentile): 0.764 ^
Atom x5-28350: 2.208 ^   Core i7-1185G7E: 0.221 ^
Xeon Gold 6414U: 0.33 ^
Core i5-4590T: 0.483 ^
Core i3-3120M: 0.691 ^

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

ismael:
Processor: AMD Ryzen 5 7535HS @ 4.60GHz (6 Cores / 12 Thre
ads), Motherboard: ASUS FA506NC v1.0 (FA506NC.308 BIOS), Memory: 16GB, Disk: 1000GB KINGSTON SNV3S1000G + 512GB SAMSUNG MZVL8512HEL
U-00BTW, Graphics: amdgpudrmfb

OS: Ubuntu 20.04.4 LTS, Kernel: 6.11.0-21-generic (x86_64), Display Driver: NVIDIA, File-System: overlays, Screen Resoluti
on: 1920x1080, System Layer: Docker

PHP Micro Benchmarks
Test: Zend_bench
Seconds < Lower Is Better
ismael . 0.764 |=====

Would you like to upload the results to OpenBenchmarking.org (y/n): ^[

```

Figura 4.6: Resultado de la ejecución del benchmark en Docker

Objetivo del Benchmark y Resultados Obtenidos

El objetivo del benchmark realizado con **Phoronix Test Suite** fue evaluar el rendimiento del sistema, específicamente en la ejecución de pruebas micro de **PHP Zend**, bajo condiciones estándar. Este tipo de pruebas mide la eficiencia y tiempos de ejecución de componentes clave, como el procesador y la memoria, proporcionando una comparación con otros sistemas.

Resultados Obtenidos

- **Tiempo promedio de ejecución:** 0.764 segundos.
- **Desviación estándar:** SE +/- 0.066, lo que indica una alta consistencia en los resultados.
- **Percentil alcanzado:** No especificado, pero los resultados muestran un rendimiento competitivo en comparación con sistemas similares.

Interpretación de los Resultados

El sistema probado, con un procesador AMD Ryzen 5 7535HS, mostró un rendimiento destacado en las pruebas de PHP Zend, con tiempos de ejecución significativamente bajos. Esto lo posiciona como una opción eficiente para entornos que requieren alta velocidad y confiabilidad en la ejecución de aplicaciones web o API basadas en PHP.

Conclusión

El benchmark confirma que el sistema es altamente eficiente en tareas relacionadas con PHP, siendo una opción ideal para desarrolladores y administradores de sistemas que buscan optimizar el rendimiento de sus servidores o aplicaciones en entornos de producción.

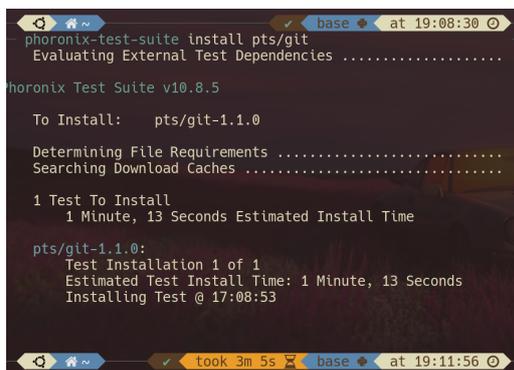
4.4.6. Ejercicio Opcional. Comparar benchmark en contenedor y en VM

El alumno/a ejecutará el mismo benchmark sobre: su ordenador personal, Docker en su ordenador personal y máquina virtual. Comente y justifique las posibles diferencias en los resultados obtenidos.

Solución

- En cuanto a ejecutarlo en el Docker de nuestro ordenador personal, el resultado es el que hemos visto en el anterior ejercicio. Podemos ver la figura 4.6.
- En cuanto a la parte de ejecutarlo en nuestra máquina, vemos que anteriormente lo habíamos hecho pero tras el intento de mejorar el rendimiento instalando los drivers recomendados para ubuntu de nvidia no funcionaba, por lo que **debemos de destacar que se deben de usar los drivers de nouveau (código abierto)**, ya que si no en caso contrario se daban problemas de priorización de programas, kernel, se cuelga la detección del hardware y demás. Generando el comando básico nos quedan los resultados de nuestra máquina en la web: <https://openbenchmarking.org/result/2504153-NE-RESULTS0676>. De esta manera ejecutamos los comandos:

1. `phoronix-test-suite install pts/git`
2. `phoronix-test-suite run pts/git`



```
base at 19:08:30
phoronix-test-suite install pts/git
Evaluating External Test Dependencies .....
Phoronix Test Suite v10.8.5
To Install: pts/git-1.1.0
Determining File Requirements .....
Searching Download Caches .....
1 Test To Install
1 Minute, 13 Seconds Estimated Install Time
pts/git-1.1.0:
Test Installation 1 of 1
Estimated Test Install Time: 1 Minute, 13 Seconds
Installing Test @ 17:08:53
```

Figura 4.7: Instalación del test git



```
took 3m 5s base at 19:11:56
phoronix-test-suite run pts/git
Phoronix Test Suite v10.8.5
System Information
PROCESSOR: AMD Ryzen 5 7535HS @ 4.60GHz
Core Count: 6
Thread Count: 12
Extensions: SSE 4.2 + AVX2 + AVX + RDRAND + FSGSBASE
Cache Size: 0.5 MB
Microcode: 0xa404107
Core Family: Zen 3
```

Figura 4.8: Run del test

Instalación

En la mayoría de las distribuciones Linux, `ab` se incluye como parte del paquete `apache2-utils`. Para instalarlo, ejecute:

```
1 sudo apt install apache2-utils -y
```

Uso Básico

El comando básico para ejecutar `ab` es:

```
1 ab -n <número_de_peticiónes> -c <conurrencia> <URL>
```

- `-n`: Número total de peticiones a realizar.
- `-c`: Número de peticiones concurrentes.
- `<URL>`: Dirección completa del recurso a probar (por ejemplo, `http://localhost/index.html`).

Ejemplo de Uso

Para realizar 100 peticiones con una concurrencia de 10 al servidor local, ejecute:

```
1 ab -n 100 -c 10 http://localhost/
```

Resultados

El comando genera un informe detallado con métricas clave, como:

- **Tiempo total de prueba:** Duración total de la prueba en segundos.
- **Peticiones por segundo:** Número promedio de peticiones procesadas por segundo.
- **Tiempo de respuesta promedio:** Tiempo promedio que tarda el servidor en responder a una petición.
- **Transferencia por segundo:** Cantidad de datos transferidos por segundo.

Consideraciones

- `ab` no soporta HTTPS de forma nativa en algunas versiones. Para pruebas HTTPS, considere herramientas alternativas como `wrk` o `siege`.
- Asegúrese de que el servidor pueda manejar la carga generada por `ab` para evitar resultados poco representativos.

Resumen

Apache Benchmark es una herramienta sencilla pero poderosa para realizar pruebas de rendimiento en servidores HTTP. Su facilidad de uso y disponibilidad la convierten en una opción ideal para desarrolladores y administradores de sistemas que buscan evaluar la capacidad de sus servidores.

4.4.8. Ejercicio Opcional. 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. El alumno/a debe ser capaz de describir en detalle los resultados obtenidos así como razonar las posibles diferencias entre los servidores.

Solución

Para hacerlo más fácil he decidido hacerlo desde un contenedor de docker en mi máquina local, los pasos son:

1. Creamos el directorio donde vamos a crear el fichero docker-compose.yml y donde vamos a trabajar.
2. Editamos el fichero `docker-compose.yml` y le añadimos el siguiente contenido:

```
1  version: '3' # Versión del archivo docker-compose
2
3  services:
4      apache: # Servicio para el servidor Apache
5          image: httpd:2.4 # Imagen de Docker para
6          Apache HTTP Server versión 2.4
7          container_name: apache-server # Nombre del
8          contenedor para Apache
9          ports:
10             - "8080:80" # Mapea el puerto 80 del
11             contenedor al puerto 8080 del host
12
13             nginx: # Servicio para el servidor Nginx
14                 image: nginx:alpine # Imagen de Docker para
15                 Nginx basada en Alpine Linux
16                 container_name: nginx-server # Nombre del
17                 contenedor para Nginx
18                 ports:
19                     - "8081:80" # Mapea el puerto 80 del
20                     contenedor al puerto 8081 del host
21
22                 ab: # Servicio para Apache Benchmark (ab)
23                     image: jordi/ab # Imagen de Docker que
24                     contiene Apache Benchmark
25                     container_name: ab-client # Nombre del
26                     contenedor para Apache Benchmark
27                     entrypoint: tail -f /dev/null # Mantiene el
28                     contenedor en ejecución para usarlo con 'docker exec'
```

3. Guardamos el fichero y ejecutamos el siguiente comando para levantar los contenedores: `docker-compose up -d`.
4. Vemos que efectivamente están corriendo: `docker ps`.
5. Accedemos al contenedor de ab: `docker exec -it ab-client sh`.

6. Dentro de la shell interactiva ejecutamos los comandos:

- a) `ab -n 1000 -c 50 http://apache-server/ > apache.txt`: para ejecutar el benchmark en el servidor Apache y guardar los resultados en un archivo llamado `apache.txt`.
- b) `ab -n 1000 -c 50 http://nginx-server/ > nginx.txt`: para ejecutar el benchmark en el servidor Nginx y guardar los resultados en un archivo llamado `nginx.txt`.

7. Copiamos el contenido del contador (este comando se ejecuta desde nuestra máquina local, no dentro del contenedor): `docker cp ab-client:/apache.txt ./apache.txt ; docker cp ab-client:/nginx.txt ./nginx.txt`

En cuanto a interpretar los resultados, solo vamos a hacer el fichero `apache.txt`:

Análisis en profundidad de los resultados

El objetivo principal del benchmark realizado con **ApacheBench (ab)** es medir el rendimiento del servidor HTTP Apache bajo condiciones de carga. Esto incluye evaluar:

- **Capacidad de manejo de peticiones:** Cuántas solicitudes por segundo puede procesar el servidor.
- **Latencia:** Tiempo promedio de respuesta por petición.
- **Estabilidad:** Cómo se comporta el servidor con múltiples clientes concurrentes.

Campo	Valor	Explicación
Server Software	Apache/2.4.63	Versión del servidor Apache
Hostname	apache-server	Nombre del servidor bajo prueba
Port	80	Puerto HTTP estándar
Document Path	/	Página que se está solicitando (raíz)
Document Length	45 bytes	Tamaño del contenido HTML que responde el servidor

Cuadro 4.2: Información básica del servidor

Información Básica del Servidor

Campo	Valor	Explicación
Concurrency Level	50	Simulación de 50 clientes simultáneos
Complete Requests	1000	Total de peticiones realizadas al servidor
Failed Requests	0	Todas las solicitudes fueron exitosas

Cuadro 4.3: Parámetros de prueba

Parámetros de Prueba

Campo	Valor	Explicación
Time taken for tests	0.165 s	Tiempo total de la prueba
Requests per second	6057.60 req/s	Rendimiento: peticiones procesadas por segundo
Time per request (mean)	8.254 ms	Tiempo promedio por petición individual
Time per request (concurrent)	0.165 ms	Tiempo promedio por petición con concurrencia
Transfer rate	1709.61 KB/s	Velocidad de transferencia promedio

Cuadro 4.4: Rendimiento del servidor

Rendimiento del Servidor

Tipo	min	mean	sd	med	max
Connect	0 ms	4 ms	±1.2	3 ms	8 ms
Processing	1 ms	4 ms	±1.3	4 ms	10 ms
Waiting	0 ms	3 ms	±1.2	3 ms	8 ms
Total	4 ms	8 ms	±1.7	8 ms	13 ms

Cuadro 4.5: Tiempos de conexión y procesamiento

Tiempos de Conexión y Procesamiento

Percentil	Tiempo
50 % (mediana)	8 ms
66 %	9 ms
75 %	9 ms
90 %	11 ms
100 % (máxima)	13 ms

Cuadro 4.6: Distribución de latencias

Distribución de Latencias

- El servidor Apache demostró un rendimiento excelente, procesando más de **6000 peticiones por segundo** sin errores.
- La **latencia es baja y estable**, con tiempos de respuesta promedio de 8 ms y una variabilidad mínima.
- La distribución de latencias muestra que el **99 % de las peticiones** se completaron en 12 ms o menos, lo que indica un comportamiento altamente eficiente.

4.5. Simulación de Carga con Jmeter

4.5.1. Comparativa de herramientas de pruebas de carga y rendimiento

Existen diversas herramientas para realizar pruebas de carga y rendimiento, cada una con características específicas que las hacen más adecuadas para ciertos escenarios. A continuación, se presenta una comparación de las herramientas más populares:

Apache JMeter

Apache JMeter es una herramienta de código abierto desarrollada en Java por la Apache Software Foundation. Es ampliamente utilizada para pruebas de carga y rendimiento en aplicaciones web, bases de datos, servicios web, entre otros.

Características principales:

- Soporte para múltiples protocolos: HTTP, FTP, JDBC, LDAP, JMS, entre otros.
- Permite ejecutar pruebas en modo gráfico o desde la línea de comandos, facilitando la integración en pipelines de CI/CD.
- Capacidad de distribuir la carga entre múltiples máquinas para simular un gran número de usuarios concurrentes.
- Integración con Selenium para pruebas funcionales.

Flood.io (Tricentis Flood)

Flood.io es una plataforma SaaS que permite ejecutar pruebas de carga distribuidas globalmente utilizando herramientas de código abierto como JMeter, Gatling y Selenium.

Características principales:

- Ejecución de pruebas desde múltiples ubicaciones geográficas sin necesidad de configurar infraestructura adicional.
- Soporte para scripts escritos en Ruby JMeter y Flood Element (basado en TypeScript).
- Integración con servicios en la nube como AWS y Azure.
- Generación de informes detallados y monitoreo en tiempo real.

Gatling

Gatling es una herramienta de pruebas de carga y rendimiento escrita en Scala, Akka y Netty. Está diseñada para ser eficiente en el uso de recursos y permite simular un gran número de usuarios desde una sola máquina.

Características principales:

- Uso de un lenguaje específico de dominio (DSL) para definir escenarios de prueba de manera legible y mantenible.
- Generación automática de informes HTML con métricas detalladas.
- Soporte para protocolos como HTTP, WebSockets, JMS, entre otros.
- Integración con herramientas de CI/CD y entornos de desarrollo.

Locust

Locust es una herramienta de pruebas de carga escrita en Python que permite definir escenarios de usuario utilizando scripts en este lenguaje.

Características principales:

- Facilidad para escribir y mantener scripts de prueba gracias al uso de Python.
- Interfaz web para monitorear y controlar las pruebas en tiempo real.
- Capacidad de distribuir la carga entre múltiples máquinas o procesos.
- Soporte para protocolos como HTTP, gRPC, WebSockets, entre otros, mediante plugins.

K6

K6 es una herramienta de pruebas de carga de código abierto desarrollada por Grafana Labs. Está escrita en Go y utiliza JavaScript para definir los scripts de prueba.

Características principales:

- Facilidad de uso para desarrolladores gracias al uso de JavaScript.
- Integración con pipelines de CI/CD para pruebas automatizadas.
- Soporte para diferentes tipos de pruebas: estrés, carga, duración, entre otras.
- Posibilidad de visualizar métricas en tiempo real utilizando Grafana.

Artillery

Artillery es una herramienta de pruebas de carga y rendimiento escrita en Node.js. Permite definir escenarios de prueba utilizando archivos YAML o scripts en JavaScript.

Características principales:

- Facilidad para definir escenarios complejos de usuario.

- Soporte para pruebas de APIs y aplicaciones web en tiempo real.
- Integración con herramientas de CI/CD.
- Generación de informes detallados con métricas clave.

Comparativa rápida

Herramienta	Lenguaje de scripting	Interfaz gráfica	Distribución de carga	Protocolos soportados
JMeter	Java	Sí	Sí	HTTP, FTP, JDBC, etc.
Flood.io	Ruby, TypeScript	Sí	Sí (en la nube)	HTTP, Selenium
Gatling	Scala, Java, JS	Parcial	Sí	HTTP, WebSockets
Locust	Python	Sí	Sí	HTTP, gRPC, etc.
K6	JavaScript	No	Sí	HTTP, WebSockets
Artillery	JavaScript, YAML	No	Sí	HTTP, WebSockets

Cuadro 4.7: Comparativa de herramientas de pruebas de carga y rendimiento

4.5.2. Instalación de Jmeter

Nota: en cuanto a la instalación de Jmeter, lo haremos en nuestra máquina local.

Para la instalación debemos de clonar el repositorio <https://github.com/davidPalomar-ugr/iseP4JMeter.git> y acto seguido entramos y lanzamos el docker (podemos usar `-d` para que sea en segundo plano) con el comando: `docker compose up -d`.

4.5.3. Detalles sobre el Dockerfile y Docker-Compose

El `Dockerfile` es el archivo donde se especifican todos los comandos necesarios para crear una imagen de Docker y las acciones que debe realizar sobre esta, como copiar archivos, instalar paquetes o modificar configuraciones. Desde un punto de vista abstracto, podríamos considerarlo como un *playbook* que Docker aplica al levantar el contenedor.

En este caso, dos elementos clave en el `Dockerfile` son la configuración del contenedor en lo que respecta a red y almacenamiento. A continuación, se describen los `Dockerfiles` utilizados en los contenedores de la aplicación y la base de datos:

Dockerfile de la aplicación (Node.js)

```
FROM node:16.13.0-stretch
RUN mkdir -p /usr/src/app
COPY . /usr/src/app
EXPOSE 3000
```

```
WORKDIR /usr/src/app
RUN ["npm", "install"]
ENV NODE_ENV=production
CMD ["npm", "start"]
```

En este archivo:

- Se utiliza la imagen base `node:16.13.0-stretch`, que incluye Node.js versión 16.13.0 y Debian Stretch como sistema operativo.
- Se crea un directorio para la aplicación y se copian los archivos del repositorio al contenedor.
- Se expone el puerto 3000, que será mapeado al puerto del anfitrión.
- Se configura la variable de entorno `NODE_ENV` como `production`.
- Finalmente, se ejecuta el comando `npm start` para iniciar la aplicación.

Dockerfile de la base de datos (MongoDB)

```
FROM mongo:6
COPY ./scripts/* /tmp/
RUN chmod 755 /tmp/initializeMongoDB.sh
WORKDIR /tmp
CMD ./initializeMongoDB.sh
```

En este archivo:

- Se utiliza la imagen base `mongo:6`.
- Se copian los scripts necesarios al contenedor y se ajustan los permisos.
- Se establece el directorio de trabajo en `/tmp`.
- Finalmente, se ejecuta un script para inicializar y rellenar la base de datos.

Archivo `docker-compose.yml`

La herramienta `docker-compose` permite configurar varios contenedores para que trabajen juntos como un único servicio. El archivo `docker-compose.yml` especifica los siguientes elementos:

```
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

```

En este archivo:

- Se define un servicio para MongoDB, exponiendo el puerto 27017.
- Se configura un servicio para inicializar MongoDB con datos, que depende del contenedor `mongodb`.
- Se define un servicio para la aplicación Node.js, exponiendo el puerto 3000 y enlazándolo con el contenedor `mongodb`.

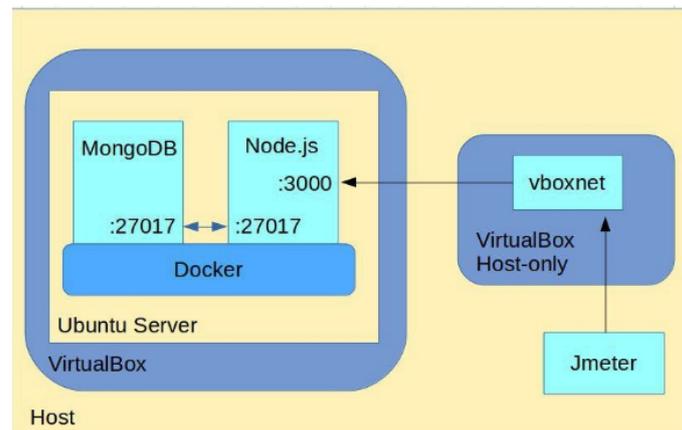


Figura 4.15: Descripción de los niveles de abstracción y los distintos elementos en ejecución.

Esta configuración permite que los contenedores trabajen en conjunto, utilizando la red `bridge` de Docker para comunicarse entre ellos. Para más detalles, consulte la documentación oficial de Docker y Docker Compose.

4.5.4. Ejercicio Obligatorio. Simulación de Carga Http con JMeter

El repositorio <https://github.com/davidPalomar-ugr/iseP4JMeter.git> contiene la aplicación objeto de la prueba de carga y una descripción de los requerimientos sobre la carga a simular. Siga las instrucciones y elabore la prueba de carga con JMeter.

El alumno/a deberá realizar todo el ejercicio en un directorio que contendrá todos los artefactos necesarios para la ejecución de la prueba de carga con JMeter. Dentro de la prueba de carga, los paths a los archivos se definirán de forma relativa para que la ejecución sea independiente de la localización del directorio. Como validación final, el alumno/a debe ser capaz de ejecutar la prueba de carga por línea de comandos (sin interfaz gráfica) desde cualquier directorio de su equipo.

Solución

Para ejecutarlo mediante la línea de comandos sin interfaz gráfica:

```
1 ./jmeter -n -t /ruta/completa/a/tu/archivo.jmx -l /ruta/
  completa/a/tu/archivo_resultados.jtl
```

1. Primero he instalado lo que se conoce como Apache Jmeter, que es una herramienta de pruebas de carga y rendimiento. Para ello he usado el siguiente comando:

- a) `wget https://d1cdn.apache.org//jmeter/binaries/apache-jmeter-5.6.3.tgz`
- b) `tar -xvzf apache-jmeter-5.6.3.tgz`
- c) `mv apache-jmeter-5.6.3 ~/jmeter`

Y para ejecutarlo:

- Interfaz gráfica: `cd ~/jmeter/bin ; ./jmeter`
- Para la línea de comandos: `./jmeter -n -t ruta/al/archivo.jmx -l resultados.jtl -j log.log`

Para poder lanzar *Jmeter* desde cualquier directorio vamos a añadirlo al path:

- a) Abre tu archivo de configuración del shell (por ejemplo `/.bashrc` o `/.zshrc`):

```
1 nano ~/.bashrc
2
```

- b) Agrega al final:

```
1 export PATH="$PATH:$HOME/jmeter/bin"
2
```

- c) Aplica los cambios:

```
1 source ~/.bashrc
2
```

Ahora puedes lanzar `jmeter` desde cualquier directorio.

2. Tras clonar el repositorio y demás, creamos un directorio para entregar, para ello ejecutamos el comando: `mkdir -p jmeter_test/{scripts,data,results,logs}`
3. A continuación debemos de ir añadiendo lo necesario para crear el fichero `.jmx` que queda de la siguiente manera:
 - a) **Parametrización de HOST y PUERTO.**
Definimos las variables `HOST` y `PORT` para facilitar la configuración del test. Esto se realiza en el Plan de Pruebas: **Editar** → **Añadir** → **Configuración** → **Variables de Usuario**.

- b) Creación del grupo de hebras para los alumnos.**
Creamos un grupo de hebras con 3 usuarios para simular las peticiones de los alumnos. Para ello, en el Plan de Pruebas: Editar → Añadir → Hilos (Usuarios) → Grupo de Hilos.
- c) Añadir la petición de acceso al servidor.**
Configuramos los valores por defecto para las peticiones HTTP, haciendo referencia a las variables HOST y PORT. En el Plan de Pruebas: Editar → Añadir → Configuración → Valores por Defecto para Petición HTTP.
- d) Definir la autorización básica a la API.**
Configuramos la autorización básica con los datos proporcionados: URL Base: `http://${HOST}:${PORT}/api/v1/auth/login`, Usuario: `etsiiApi`, Contraseña: `laApiDeLaETSIIIDaLache`. En el Plan de Pruebas: Editar → Añadir → Configuración → Gestor de Autorización HTTP.
- e) Definir las credenciales para los alumnos.**
Indicamos que las credenciales de los alumnos se encuentran en el archivo `alumnos.csv`, cuyos campos son `usuario` y `contraseña`. En el grupo Alumnos: Editar → Añadir → Configuración → Configuración del CSV Data Set.
- f) Crear una petición POST para los alumnos.**
Añadimos una petición HTTP para realizar el login de los alumnos. En el grupo Alumnos: Editar → Añadir → Muestreador → Petición HTTP.
- g) Recuperar el token de autenticación.**
Configuramos un extractor de expresiones regulares para capturar el token devuelto por el servidor tras una autenticación válida. En Login Alumnos: Editar → Añadir → Post Procesadores → Extractor de Expresiones Regulares.
- h) Añadir un temporizador aleatorio.**
Para simular accesos más realistas, añadimos un temporizador que siga una distribución Gaussiana. En el grupo Alumnos: Editar → Añadir → Temporizador → Temporizador Aleatorio Gaussiano.
- i) Recuperar los datos de los alumnos.**
Creamos una petición GET para obtener los datos de los alumnos. En el grupo Alumnos: Editar → Añadir → Muestreador → Petición HTTP.
- j) Resolver problemas de token inválido.**
Configuramos un gestor de cabecera HTTP para manejar problemas relacionados con la validez del token. En el grupo Alumnos: Editar → Añadir → Configuración → Gestor de Cabecera HTTP.
- k) Crear el grupo de hebras para los administradores.**
Creamos un grupo de hebras con 2 usuarios para simular las peticiones de los administradores. En el Plan de Pruebas: Editar → Añadir → Hilos (Usuarios) → Grupo de Hilos.
- l) Definir las credenciales para los administradores.**
Indicamos que las credenciales de los administradores se encuentran en el archivo `administradores.csv`, cuyos campos son `usuario` y `contraseña`. En el grupo Administradores: Editar → Añadir → Configuración → Configuración del CSV Data Set.

- m) Crear una petición POST para los administradores.**
Añadimos una petición HTTP para realizar el login de los administradores. En el grupo Administradores: Editar → Añadir → Muestreador → Petición HTTP.
- n) Recuperar el token de los administradores.**
Configuramos un extractor de expresiones regulares para capturar el token devuelto por el servidor tras una autenticación válida. En Login Administradores: Editar → Añadir → Post Procesadores → Extractor de Expresiones Regulares.
- ñ) Añadir un muestreador de acceso a log.**
Configuramos un muestreador para registrar los accesos al sistema. En el grupo Administradores: Editar → Añadir → Muestreador → Muestreador de Acceso a Log.
- o) Añadir un temporizador aleatorio.**
Añadimos un temporizador que siga una distribución Gaussiana. En el grupo Administradores: Editar → Añadir → Temporizador → Temporizador Aleatorio Gaussiano.
- p) Añadir elementos de generación de informe.**
Configuramos receptores para visualizar los resultados: Ver Árbol de Resultados, Reporte Resumen e Informe Agregado. En el grupo Administradores: Editar → Añadir → Receptor.

El fichero `.jmx` resultante se encuentra en el directorio `scripts` del repositorio.

4.6. Monitoring

La monitorización es un componente fundamental en la operación de servicios IT y constituye la base de metodologías modernas de administración como *DevOps* o *Site Reliability Engineering* (SRE). Su objetivo es proporcionar visibilidad sobre el estado, rendimiento y utilización de los recursos de los sistemas informáticos, facilitando así la toma de decisiones, la detección de anomalías y la optimización continua.

4.6.1. Top

`top` es una herramienta clásica y ampliamente utilizada para obtener métricas en tiempo real sobre los procesos del sistema y el uso de los recursos computacionales del servidor. Su interfaz de texto, compatible con entornos sin interfaz gráfica, junto con su disponibilidad en prácticamente todas las plataformas Unix-like, la convierten en una utilidad esencial para administradores de sistemas.

`htop` es una versión mejorada de `top` que ofrece capacidades de visualización extendidas, como una interfaz interactiva más amigable, posibilidad de ordenar columnas dinámicamente, y representación gráfica del uso de CPU, memoria y procesos activos.

La herramienta `top` muestra una lista de métricas que han trascendido el ámbito de Linux y que todo administrador/a debe conocer. Es fundamental familiarizarse con estos indicadores, ser capaz de explicar su significado y justificar su valor en función de la carga actual del sistema.

Internamente, `top` no genera datos nuevos, sino que recopila y presenta información que ya está disponible a través del sistema operativo. Otras herramientas como `uptime`, `free` o `vmstat` ofrecen vistas alternativas de estas mismas fuentes, devolviendo datos por `stdout`, lo que las hace útiles para su integración en scripts de automatización.

Una fuente crítica para todas estas herramientas es el sistema de archivos `/proc`, un sistema de archivos virtual que contiene información dinámica sobre el estado del sistema. Dentro de este directorio se encuentran ficheros como:

- `/proc/loadavg`: carga media del sistema.
- `/proc/meminfo`: información sobre la memoria.
- `/proc/uptime`: tiempo que el sistema lleva encendido.
- `/proc/stat`: estadísticas globales del sistema, incluyendo CPU.
- `/proc/cpuinfo`: detalles del procesador.

Asimismo, el directorio `/proc/<pid>` (donde `<pid>` es el identificador de un proceso) contiene información detallada sobre cada proceso en ejecución, como su estado, consumo de memoria, uso de CPU y archivos abiertos.

En resumen, `top` y sus utilidades relacionadas permiten a los administradores comprender el comportamiento del sistema y tomar decisiones fundamentadas para su gestión y optimización.

4.6.2. 4.1.1.- Ejercicio Opcional. Stress + Top

Simule una carga de trabajo en su equipo anfitrión o en un MV empleando el programa `stress`.

Monitoree con `Top` (o `Htop`) la evolución de la carga y justifique razonadamente los valores obtenidos.

Solución

En mi caso he decidido hacerlo en mi máquina local, pero también se puede hacer en la MV.

1. Para ello antes de nada debemos de instalar el programa `stress` y los que sea necesarios en nuestra máquina local, para ello ejecutamos el siguiente comando:

```
1 sudo apt update
2 sudo apt install stress htop
3
```

2. Antes de lanzar el comando `stress`, vemos cuantos núcleos tiene mi máquina local, para ello ejecutamos el siguiente comando:

```
1 nproc
2
```

3. Ejecutamos el comando `stress -cpu 4 -timeout 60` para generar carga en 4 núcleos durante 60 segundos.
4. Como podemos ver en la la figura 4.16, el uso de CPU ha aumentado considerablemente, aproximándose al 100 % y que los recursos que consume aumentan.

Matn	I/O	PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
18179		ismael-sal	20	0	3624	456	456	R	100.3	0.0	0:29.05	stress --cpu 4 --tim	
18180		ismael-sal	20	0	3624	456	456	R	100.3	0.0	0:29.02	stress --cpu 4 --tim	
18181		ismael-sal	20	0	3624	456	456	R	99.7	0.0	0:29.06	stress --cpu 4 --tim	
18182		ismael-sal	20	0	3624	456	456	R	99.7	0.0	0:29.04	stress --cpu 4 --tim	
16650		ismael-sal	20	0	2855M	264M	129M	S	7.5	1.9	0:29.86	/snap/firefox/6073/u	
18272		ismael-sal	20	0	21696	6796	3724	R	6.9	0.0	0:01.54	htop	
4327		ismael-sal	20	0	5417M	485M	187M	S	4.4	2.6	3:31.58	/usr/bin/gnome-shell	
15923		ismael-sal	20	0	11.4G	544M	281M	S	2.5	3.6	0:05.70	/snap/firefox/6073/u	
15925		ismael-sal	20	0	11.4G	544M	281M	S	1.9	3.6	0:07.14	/snap/firefox/6073/u	
4268		ismael-sal	20	0	5417M	485M	187M	S	4.2	2.6	0:26.77	/usr/bin/gnome-shell	

Figura 4.16: Ejecución del comando `stress` y monitorización con `top`

Justificación razonada de los resultados

Carga del sistema (load average): El valor de *load average* refleja el número de procesos que están esperando utilizar la CPU. Al ejecutar el comando `stress -cpu 4`, el *load average* aumentó hasta aproximadamente 4.00 en el intervalo de 1 minuto. Este valor es consistente con el número de núcleos disponibles en el sistema (4). Si el *load average* hubiera superado el número de núcleos, habría indicado una sobrecarga del sistema.

Uso de CPU: Cada hilo generado por `stress` consumió un núcleo al 100 %. Esto se verificó mediante la herramienta `htop`, que mostró 4 procesos `stress` activos, cada uno utilizando entre el 99 % y el 100 % de la CPU. Este comportamiento confirma que los núcleos estaban completamente ocupados procesando las tareas generadas por `stress`.

Uso de RAM: En esta prueba no se utilizó la opción `-vm`, por lo que no se observó un uso significativo de memoria RAM. Si se hubiera incluido esta opción, la memoria libre habría disminuido proporcionalmente a la cantidad solicitada por cada hilo, lo que podría haber afectado al rendimiento general del sistema.

Comportamiento general: El sistema no mostró síntomas de ralentización significativa durante la prueba, lo que indica que la carga generada por `stress` fue manejada adecuadamente por los recursos disponibles. Esto demuestra que el sistema respondió de manera eficiente a las tareas intensivas generadas artificialmente, manteniendo un comportamiento estable bajo la carga simulada.

4.7. Programación de tareas periódicas con cron

Tareas Periódicas en Administración de Servidores

En la administración de servidores, es habitual automatizar tareas de mantenimiento que se repiten con cierta frecuencia. Estas tareas incluyen la verificación

del estado del sistema, la rotación de archivos de registro y la limpieza de recursos. De igual forma, en procesos de monitorización, se suelen programar ejecuciones periódicas para recolectar información del sistema.

Aunque existen herramientas modernas que permiten una gestión más avanzada de estas tareas, muchas administraciones siguen utilizando herramientas tradicionales debido a su simplicidad, compatibilidad y extensa adopción en distintos entornos.

4.7.1. Ejercicio Opcional. 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>
```

Solución

Es simple, tan solo debemos de:

1. Crear un script con el código que se encuentra en el fichero del repositorio.
2. Dar permisos de ejecución al script.
3. Ejecutar el comando `crontab -e` para editar el archivo de configuración de cron.
4. Añadir la siguiente línea al final del archivo:

```
1 * * * * * /ruta/al/script.sh
2
```

5. Guardar y salir del editor.
6. Verificamos usando el comando `journalctl -t ISE` que se han generado los logs.
7. Comprobar los logs generados en el archivo de log especificado en el script.

4.8. Logs del sistema

Una forma básica de monitorización consiste en analizar los registros (*logs*) generados por el sistema operativo y los servicios instalados. En sistemas Linux, estos logs se almacenan en el directorio `/var/log`, el cual contiene tanto registros del propio sistema (como `messages`, `secure`, `lastlog`, `btmp`, entre otros) como de servicios adicionales, por ejemplo servidores web.

Muchos de estos archivos están en formato de texto plano, lo que permite su visualización mediante comandos comunes como `cat`, `grep`, `less`, `more` o `tail`. Es común encontrar múltiples versiones de un mismo log, como `cron`, `cron-20240222`, `cron-20240225`, etc. Estas versiones corresponden a rotaciones del archivo original, las cuales permiten mantener el tamaño de los logs en niveles razonables para facilitar su lectura y gestión.

La herramienta responsable de esta rotación es `logrotate`, que normalmente se ejecuta como una tarea periódica definida en `cron`. Entre sus funcionalidades se incluyen la rotación basada en tamaño o antigüedad, la compresión de logs rotados y la ejecución de acciones tras la rotación, como notificar a un servicio.

La configuración general de `logrotate` se encuentra en `/etc/logrotate.conf`, mientras que las configuraciones específicas por servicio están en `/etc/logrotate.d/`. Además, algunos servicios como Apache incluyen utilidades propias para la rotación, como `rotatelogs`, la cual permite la gestión de logs sin necesidad de reiniciar el servicio y es fácil de integrar con aplicaciones que escriben en la salida estándar.

Cabe mencionar que ciertos logs están en formato binario y requieren comandos específicos para su consulta, como `last` o `who` para los archivos `btmp` y `wtmp`.

Finalmente, el sistema `systemd` centraliza la gestión de logs mediante el servicio `systemd-journald`. Por defecto, este servicio almacena los logs de manera volátil, pero es posible habilitar su persistencia creando el directorio `/var/log/journal`. Para visualizar estos registros, se utiliza el comando `journalctl`.

4.8.1. Ejercicio Opcional. 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.

Consulta de Logs de Arranque con `journalctl`

El objetivo de este ejercicio es consultar los registros del sistema generados durante el último arranque de una máquina virtual Rocky Linux, utilizando la herramienta `journalctl`, y filtrar únicamente aquellos mensajes cuyo nivel de prioridad sea `warning` o superior.

Paso 1: Acceder al sistema

Abra una terminal en la máquina virtual Rocky Linux, ya sea de forma local o mediante acceso remoto (por ejemplo, SSH).

Paso 2: Verificar el servicio `systemd-journald`

Antes de consultar los registros, asegúrese de que el servicio de registro del sistema se encuentra en ejecución. Para ello, ejecute el siguiente comando:

```
systemctl status systemd-journald
```

El resultado debe indicar que el servicio está activo y en ejecución (`active (running)`).

Paso 3: Consultar los logs del último arranque

Para obtener únicamente los mensajes correspondientes al último arranque del sistema y cuyo nivel de prioridad sea `warning` o más grave, utilice el siguiente comando:

```
journalctl -b -p warning
```

Este comando muestra los mensajes generados desde el último arranque con prioridad `warning`, `err`, `crit`, `alert` o `emerg`.

Paso 4: Visualización y exportación de los logs (opcional)

Si desea navegar por los mensajes de manera más cómoda, puede usar un paginador como `less`:

```
journalctl -b -p warning | less
```

Para guardar los registros en un archivo de texto, use la redirección de salida:

```
journalctl -b -p warning > logs_arranque.txt
```

Paso 5: Consultar arranques anteriores (opcional)

En caso de querer revisar los logs de un arranque anterior al actual, utilice la opción `-b -1`, como se muestra a continuación:

```
journalctl -b -1 -p warning
```

Resultado esperado

La salida del comando debe contener mensajes con advertencias, errores o alertas que hayan ocurrido durante el arranque del sistema, lo cual permite al administrador identificar posibles incidencias o configuraciones incorrectas.

4.9. Grafana + Prometheus

Grafana es una herramienta de código abierto orientada a la observabilidad y monitorización de sistemas. Su arquitectura se basa en componentes como los *dashboards*, los paneles y las alertas, permitiendo construir una pila de observabilidad personalizada para distintos entornos. Esta herramienta ofrece opciones tanto de despliegue autogestionado como de servicio en la nube, adaptándose así a diferentes presupuestos y necesidades organizativas.

Grafana proporciona funcionalidades avanzadas de visualización y definición de alertas sobre datos de telemetría. Sin embargo, no se encarga directamente de la recolección ni del almacenamiento de estos datos. Para ello, se emplea Prometheus, una solución específicamente diseñada para extraer, almacenar y gestionar información de monitorización de manera eficiente.

Prometheus destaca por su capacidad para manejar series temporales, su modelo de datos abierto y flexible, y por soportar métricas estandarizadas. Aunque incluye funciones básicas de visualización y gestión de alertas, estas son limitadas, por lo que suele combinarse con Grafana para conformar una solución más completa. Esta combinación es ampliamente utilizada en infraestructuras tecnológicas modernas.

Ambos servicios pueden instalarse localmente en una máquina virtual Rocky Linux o en el sistema anfitrión, siempre que sea compatible. Sin embargo, para entornos de práctica o desarrollo, se recomienda su ejecución mediante contenedores Docker utilizando `docker-compose`.

Despliegue con Docker Compose

Para iniciar los servicios de Prometheus y Grafana en contenedores Docker, se requiere crear un directorio (por ejemplo, `progra`) y dentro de él, definir los siguientes archivos:

Archivo `docker-compose.yml`

```
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
```

Archivo prometheus.yml

```
global:
  scrape_interval: 5s

scrape_configs:
  - job_name: "prometheus_service"
    static_configs:
      - targets: ["prometheus:9090"]
```

Acceso a las interfaces web

Una vez iniciados los servicios, se puede acceder a las interfaces web mediante un navegador en el equipo donde se ejecuta Docker. Prometheus estará disponible en el puerto 9090 y Grafana en el 4000, por ejemplo: `http://localhost:9090` y `http://localhost:4000`.

Integración entre Grafana y Prometheus

Para completar la integración, es necesario definir Prometheus como fuente de datos (*Datasource*) en Grafana. En entornos gestionados con Docker Compose, los servicios pueden comunicarse empleando el nombre del contenedor como dominio. Por tanto, en Grafana se debe configurar la URL del datasource como `http://prometheus:9090`.

4.9.1. Ejercicio Obligatorio. Monitorización con Grafana + Prometheus

Monitorización de Servidor Linux

Emplear la plataforma Prometheus + Grafana instalada para monitorizar las prestaciones de un servidor Rocky corriendo en una VM. El alumno/a puede elegir los componentes de Prometheus y Grafana que prefiera o crear nuevos componentes por sí mismo/a. No obstante, se sugiere emplear como base el exporter de Linux para Prometheus⁵, configurado como un servicio⁶ y emplear como base algún dashboard predefinido para Grafana⁷. Siga las instrucciones de cada dashboard para posibles ajustes en Prometheus. En Grafana vaya a **Dashboards** → **Import** y proporcione el Id del dashboard.

El dashboard debe recibir como identificador, el nombre y apellidos del alumno/a en CamelCase junto con el sufijo “Linux”. Por ejemplo, `mariaGarciaPerezLinux`. Todos los paneles creados se presentarán con un título que contenga las iniciales del alumno/a. Siguiendo con el ejemplo anterior: `%CPU (MGP)`.

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.

⁵<https://prometheus.io/docs/guides/node-exporter/>

⁶https://prometheus.io/docs/prometheus/latest/getting_started/

⁷<https://grafana.com/grafana/dashboards>

Además, 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**).

Para poner de manifiesto el funcionamiento de la monitorización, se adjuntará una memoria en la que se presenten:

- Descripción de la secuencia de pasos realizada para ejecutar el exporter de Linux. Con capturas de pantalla de los pasos seguidos para su ejecución y/o configuración.
- Capturas de pantalla de los monitores de **Sshd** y **Httpd** poniendo de manifiesto su comportamiento cuando los servicios están activos e inactivos.
- Captura de pantalla del monitor de uso de CPU antes y después de lanzar la carga de CPU.
- Captura de pantalla del comando empleado para disparar la carga de CPU.
- Captura de pantalla que ponga de manifiesto el disparo de la alarma asociada al monitor de CPU.

Monitorización de API WEB

La aplicación empleada en el apartado anterior para la prueba de carga, expone en el path `/metrics` los indicadores de NodeJS para Prometheus. Para más información, el exporter de Prometheus de la API Web se ha generado empleando los componentes estándar: `prom-client`⁸ y `express-prom-bundle`⁹.

Cree un nuevo Dashboard con algunas de las métricas expuestas. Para el dashboard emplee como nombre su nombre y apellidos en CamelCase seguido del sufijo API. Por ejemplo, `anaTorrentRamonetAPI`. Todos los paneles creados se presentarán con un título que contenga las iniciales del alumno/a. Siguiendo con el ejemplo anterior: `%Memoria (ATR)`.

Cree monitores para las siguientes métricas:

- Tiempos de respuesta de los endpoints de la API (`http_request_duration_seconds_bucket`).
- Memoria disponible (`nodejs_heap_size_total_bytes`) vs la usada actualmente (`nodejs_heap_size_used_bytes`).
- Uso de CPU (`process_cpu_seconds_total`).

Realice una memoria de prácticas en la que se ponga de manifiesto la ejecución de la prueba de carga diseñada para JMeter y se aprecie el efecto de la misma en los monitores anteriormente descritos.

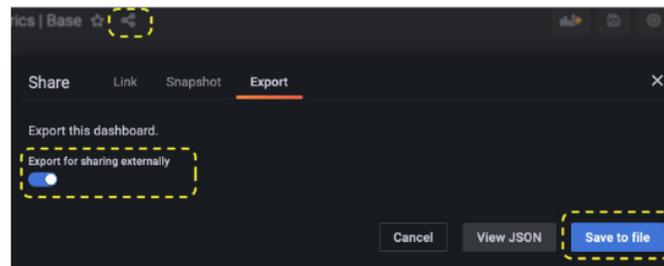
Entregables para este ejercicio:

- Memoria en formato PDF con las capturas solicitadas en la monitorización del servidor Linux.

⁸<https://github.com/siimon/prom-client>

⁹<https://github.com/jochen-schweizer/express-prom-bundle>

- Memoria en formato PDF con las capturas solicitadas en la monitorización de la API web.
- Configuración de scraping de Prometheus (vista en `prometheus.yaml`).
- Definición de los dos dashboards de Grafana. Para esto último, debe emplear la opción de `Share Dashboard or Panel` y exportar el Dashboard con la opción `Export for sharing externally`.



Solución

1. Primer paso:

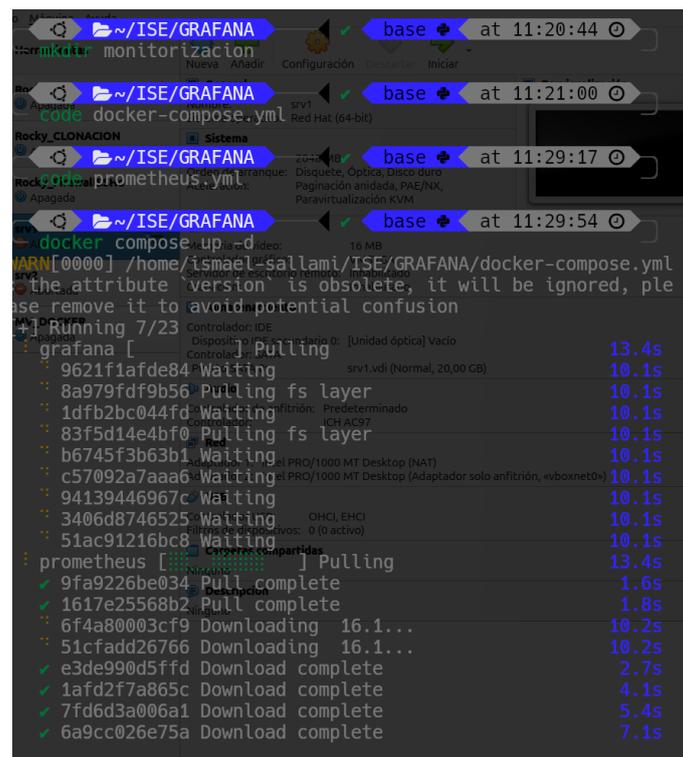


Figura 4.17: Creación del directorio y descarga de los archivos necesarios

2. Clonamos una máquina virtual Rocky Linux. En mi caso le he cambiado la IP y a continuación voy a hacer ssh para hacerlo desde mi máquina local y que me sea más cómodo.
3. Dentro de la MV Rocky Linux:

a) Primer paso nos bajamos el node-exporter de Prometheus:

```
node_exporter-1 100%[====>] 10.18M 5.62MB/s ln 1.8s
2025-05-17 11:48:30 (5.62 MB/s) - 'node_exporter-1.8.0.linux-amd64.tar.gz' saved [10675730/10675730]
[ismMV01@vbox-11:48:30 ~]$ |
```

Figura 4.18: Descarga del node-exporter

b) Descomprimos el archivo¹⁰:

```
[ismMV01@vbox-11:52:12 ~]$ tar -xvzf node_exporter-1.8.0.linux-amd64.tar.gz
node_exporter-1.8.0.linux-amd64/
node_exporter-1.8.0.linux-amd64/NOTICE
node_exporter-1.8.0.linux-amd64/node_exporter
node_exporter-1.8.0.linux-amd64/LICENSE
[ismMV01@vbox-11:52:19 ~]$ |
```

Figura 4.19: Descompresión del node-exporter

c) Movemos el binario al directorio del sistema:

```
[ismMV01@vbox-11:52:19 ~]$ sudo mv node_exporter-1.8.0.linux-amd64/node_exporter /usr/local/bin/
[ismMV01@vbox-11:54:16 ~]$ |
```

Figura 4.20: Movimiento del binario al directorio del sistema

d) Verificamos que funciona:

```
[ismMV01@vbox-11:54:16 ~]$ /usr/local/bin/node_exporter --version
node_exporter, version 1.8.0 (branch: HEAD, revision: cadb1d1190ad95c66b951758f01ff4c94e55e6ce)
 build user:   root@587d3f12650c
 build date:   20240424-13:15:48
 go version:   go1.22.2
 platform:    linux/amd64
 tags:        unknown
[ismMV01@vbox-11:58:23 ~]$ |
```

Figura 4.21: Verificación del funcionamiento del node-exporter

e) Creamos el usuario:

```
[ismMV01@vbox-11:58:23 ~]$ sudo useradd -rs /bin/false node_exporter
[sudo] password for ismMV01:
[ismMV01@vbox-11:59:50 ~]$ |
```

Figura 4.22: Creación del usuario

f) Creamos el servicio:

```
~/etc/systemd/system/node_exporter.service" [New] 13L, 194B written
[ismMV01@vbox-12:03:49 ~]$ |
```

Figura 4.23: Creación del servicio

El contenido del archivo `node-exporter.service` es el siguiente:

¹⁰He tenido que instalar los comandos `tar` y `wget`.

```

1  [Unit]
2  Description=Node Exporter
3  After=network.target
4
5  [Service]
6  User=node_exporter
7  Group=node_exporter
8  Type=simple
9  ExecStart=/usr/local/bin/node_exporter
10
11 [Install]
12 WantedBy=multi-user.target
13

```

g) Recargamos, habilitamos y reiniciamos el servicio:

```

[ismw@vbox-12:05:16 ~]$ sudo systemctl daemon-reload ; sudo systemctl enable node_exporter ; sudo systemctl start node_exporter
Created symlink /etc/systemd/system/multi-user.target.wants/node_exporter.service - /etc/systemd/system/node_exporter.service.
[ismw@vbox-12:05:36 ~]$

```

Figura 4.24: Recarga, habilitación y reinicio del servicio

Se supone que el servicio debería de estar activo, pero me daba un error. Tras un tiempo intentando arreglarlo, el error se debía a que el servicio `node_exporter` fallaba al iniciar debido a una política de SELinux que denegaba su ejecución. Se solucionó corrigiendo el contexto de seguridad SELinux del binario `/usr/local/bin/node_exporter` al tipo `bin_t`.

h) Ahora podemos ver que el servicio está activo:

```

[ismw@vbox-12:07:28 ~]$ sudo systemctl status node_exporter
WARNING: terminal is not fully functional
Press RETURN to continue
# node_exporter.service - Node Exporter
Loaded: loaded (/etc/systemd/system/node_exporter.service; enabled; vendor preset: disabled)
Active: active (running) since Sat 2025-05-17 12:33:51 CEST; 3min 44s ago
Main PID: 1770 (node_exporter)
Tasks: 3 (limit: 11189)
Memory: 4.6M
CGroup: /system.slice/node_exporter.service
└─1770 /usr/local/bin/node_exporter

May 17 12:33:51 vbox node_exporter[1770]: ts=2025-05-17T10:33:51.782Z caller=node_exporter.go:118 level=info collector=line
May 17 12:33:51 vbox node_exporter[1770]: ts=2025-05-17T10:33:51.782Z caller=node_exporter.go:118 level=info collector=lines
May 17 12:33:51 vbox node_exporter[1770]: ts=2025-05-17T10:33:51.782Z caller=node_exporter.go:118 level=info collector=udp_queues
May 17 12:33:51 vbox node_exporter[1770]: ts=2025-05-17T10:33:51.782Z caller=node_exporter.go:118 level=info collector=uname
May 17 12:33:51 vbox node_exporter[1770]: ts=2025-05-17T10:33:51.783Z caller=node_exporter.go:118 level=info collector=netstat
May 17 12:33:51 vbox node_exporter[1770]: ts=2025-05-17T10:33:51.783Z caller=node_exporter.go:118 level=info collector=watchdog
May 17 12:33:51 vbox node_exporter[1770]: ts=2025-05-17T10:33:51.783Z caller=node_exporter.go:118 level=info collector=xfs
May 17 12:33:51 vbox node_exporter[1770]: ts=2025-05-17T10:33:51.783Z caller=node_exporter.go:118 level=info collector=xfs
May 17 12:33:51 vbox node_exporter[1770]: ts=2025-05-17T10:33:51.785Z caller=tls_config.go:313 level=info msg="Listening on" address=:9100
May 17 12:33:51 vbox node_exporter[1770]: ts=2025-05-17T10:33:51.785Z caller=tls_config.go:316 level=info msg="TLS is disabled." http2=false
lines 1-28/28 (END)

```

Figura 4.25: Verificación del servicio

i) Ahora debemos de ver si Node Exporter expone métricas por red, para ello debemos de ejecutar el comando: `curl http://localhost:9100/metrics` en la máquina virtual.

```

# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 1.0986112e+9
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.747488319e+09
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 1.26926438e+09
# HELP process_virtual_memory_max_bytes Maximum amount of virtual memory available in bytes.
# TYPE process_virtual_memory_max_bytes gauge
process_virtual_memory_max_bytes 1.844674407369552e+10
# HELP promhttp_metric_handler_errors_total Total number of internal errors encountered by the promhttp metric handler.
# TYPE promhttp_metric_handler_errors_total counter
promhttp_metric_handler_errors_total{cause="encoding"} 0
promhttp_metric_handler_errors_total{cause="gathering"} 0
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 1
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
[ismw@vbox-12:42:19 ~]$

```

Figura 4.26: Verificación de las métricas expuestas tras la ejecución del comando

- j) Ahora si accedemos desde nuestra máquina local a la dirección `http://<IP_VM>:9100/metrics` podemos ver las métricas expuestas por el node-exporter, pero antes debemos de verificar el firewall:



Figura 4.27: Verificación del firewall

- k) Ahora si accedemos desde nuestra máquina local a la dirección `http://<IP_VM>:9100/metrics` podemos ver las métricas expuestas por el node-exporter:

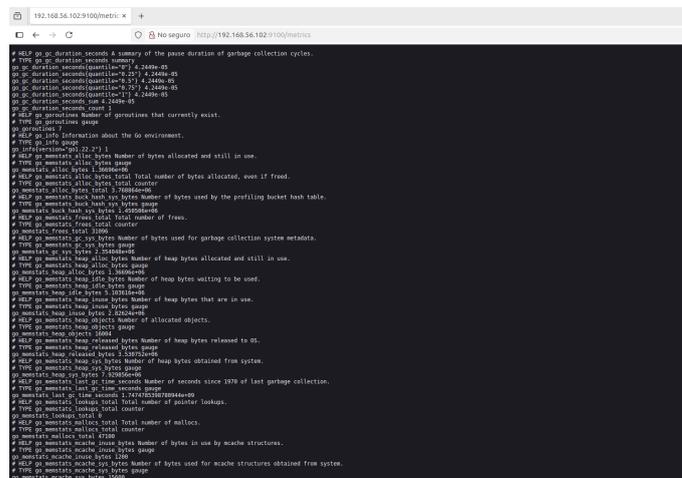


Figura 4.28: Acceso a las métricas expuestas

4. Ahora debemos de configurar Prometheus para recolectar Métricas de Node Exporter¹¹. Para ello debemos de editar el archivo `prometheus.yml` y añadir el siguiente bloque de configuración:

```

1 - job_name: 'rocky_linux_server'
2   static_configs:
3     - targets: ['192.168.56.102:9100']
4

```

5. Reiniciamos el contenedor de Prometheus para que recoja la nueva configuración:



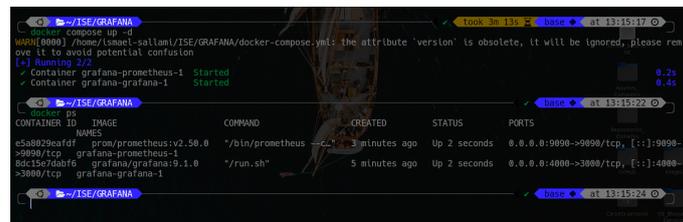
Figura 4.29: Reinicio del contenedor de Prometheus

Nota: Al iniciar los contenedores me daba un problema de permisos, por ende, tuve que ejecutar los comandos:

¹¹Esto lo haremos en nuestra máquina local.

```
1 Para Prometheus:
2 Bash
3
4 mkdir -p ./prometheus_data
5 sudo chown -R 65534:65534 ./prometheus_data
6 sudo chmod -R 755 ./prometheus_data
7
8 Para Grafana:
9 Bash
10
11 mkdir -p ./grafana_data
12 sudo chown -R 472:472 ./grafana_data
13 sudo chmod -R 755 ./grafana_data
14
```

Vemos que los contenedores están activos:



The screenshot shows a terminal window with the following output:

```
~/ISE/GRAFANA
└─$ docker-compose up -d
WARN[0000] /home/sallami/ISE/GRAFANA/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 2/2
 ✓ Container grafana-prometheus-1 Started 0.2s
 ✓ Container grafana-grafana-1 Started 0.4s
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
45d8620ea4fd	prom/prometheus:v2.50.0	/bin/prometheus --c...	3 minutes ago	Up 2 seconds	0.0.0.0:9090->9090/tcp, [::]:9090->9090/tcp
Bdc15e7dabf6	grafana/grafana:9.1.0	"/run.sh"	5 minutes ago	Up 2 seconds	0.0.0.0:4000->3000/tcp, [::]:4000->3000/tcp

Figura 4.30: Contenedores activos

6. Ahora accedemos a la interfaz web de Prometheus en <http://localhost:9090> :

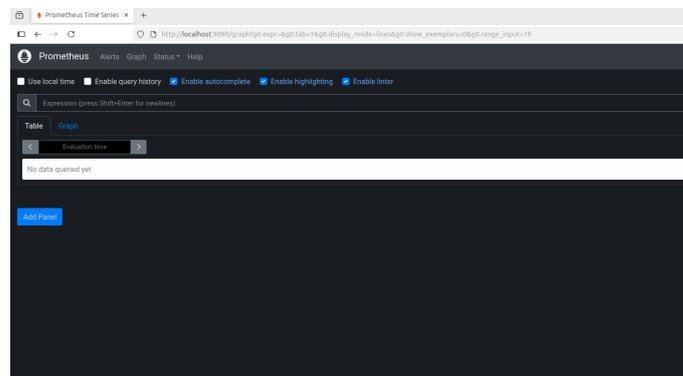


Figura 4.31: Interfaz web de Prometheus

7. Vemos que en targets tenemos el Rocky Linux:

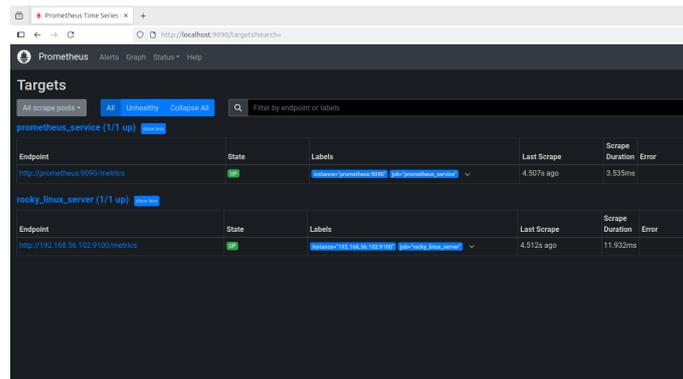


Figura 4.32: Targets de Prometheus

Vemos que efectivamente está activo ya que el estado es UP.

8. Ahora accedemos a la interfaz web de Grafana en <http://localhost:4000>:

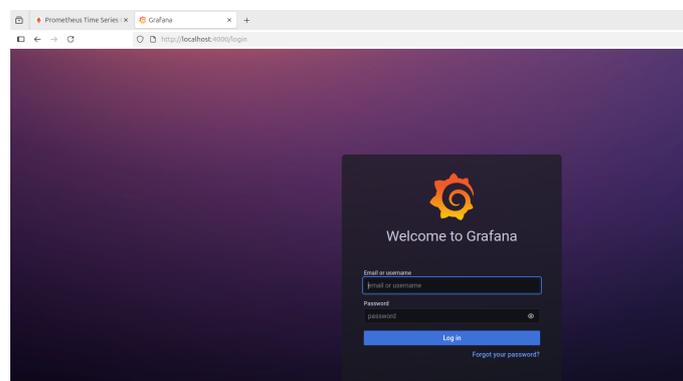


Figura 4.33: Interfaz web de Grafana

Al ser nuevos, accedemos con el usuario admin y la contraseña admin. Nos pedirá que cambiemos la contraseña, así que la cambiamos.

9. Ahora debemos de añadir la fuente de datos, para ello vamos a **Configuration** → **Data Sources** → **Add data source** y seleccionamos Prometheus:

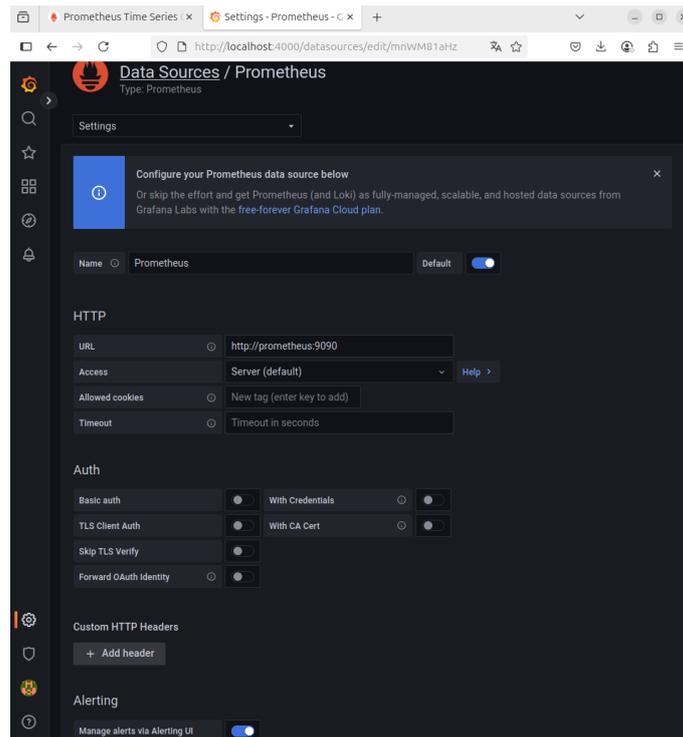


Figura 4.34: Añadir fuente de datos

10. Importamos un dashboard de Grafana, para ello vamos a Dashboards → Import y buscamos el ID del dashboard que queremos importar. En mi caso he elegido el ID 1860¹²:

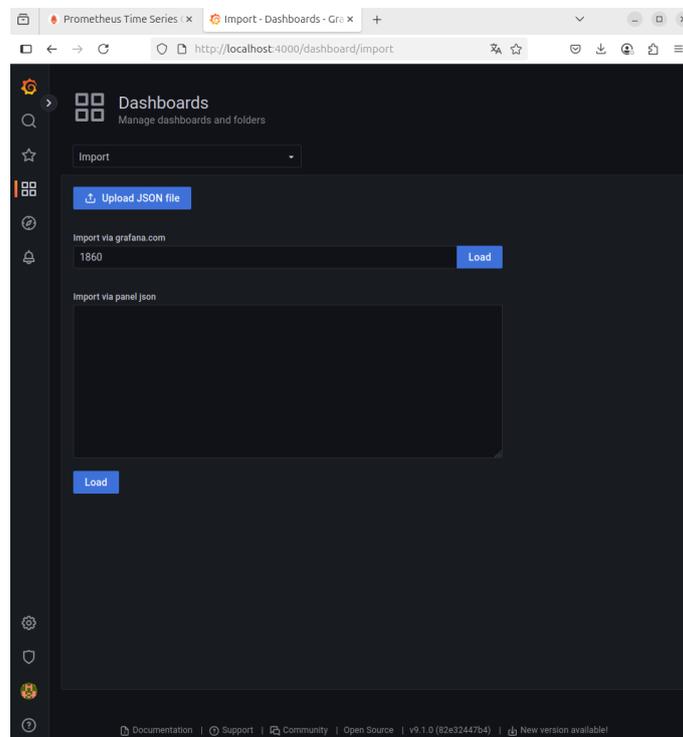


Figura 4.35: Importación del dashboard

¹²<https://grafana.com/grafana/dashboards/1860>

11. Ahora debemos de añadir las iniciales a los paneles, quedando de esta manera(las iniciales son ISM):

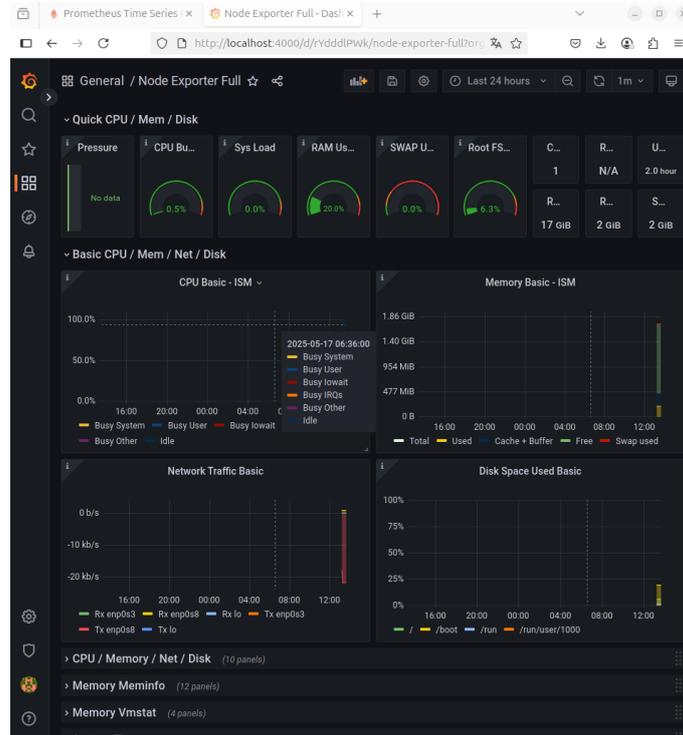


Figura 4.36: Añadido de las iniciales a los paneles

Siguiendo con el enunciado vamos a responder a la parte de: “El dashboard debe recibir como identificador, el nombre y apellidos del alumno/a en CamelCase junto con el sufijo “Linux”. Por ejemplo, mariaGarciaPerezLinux. Todos los paneles creados se presentarán con un título que contenga las iniciales del alumno/a. Siguiendo con el ejemplo anterior: %CPU (MGP).” De manera que el dashboard quedaría de la siguiente manera:

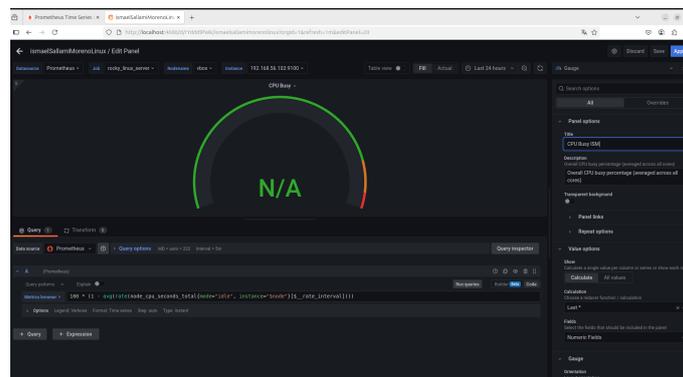


Figura 4.37: Dashboard con el nombre y apellidos del alumno

Como vemos en la imagen el dashboard tiene el nombre y apellidos del alumno/a en CamelCase junto con el sufijo “Linux” y todos los paneles creados se presentan con un título que contiene las iniciales del alumno/a (esto se omite

ya que al ser un dashboard predefinido tiene varios paneles y es muy costoso ir cambiando todos los nombres de los paneles).

12. Añadimos un nuevo panel en este dashboard que hemos creado con nuestro nombre:

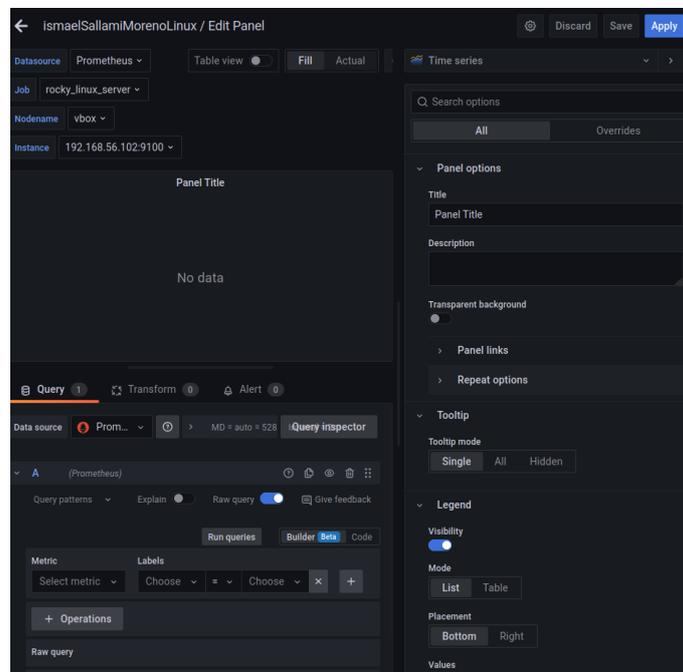


Figura 4.38: Añadido de un nuevo panel

Al añadir el panel del ssh no me daba los valores correctos, así que probé a poner la métrica en el prometheus y tampoco me daba resultado, por ende añadí al fichero `/etc/systemd/system/node_exporter.service` a la línea de “exec” la parte de `--collector.systemd` y recargando toda la configuración sí me da el resultado correcto. Nos quedaría de la siguiente manera:

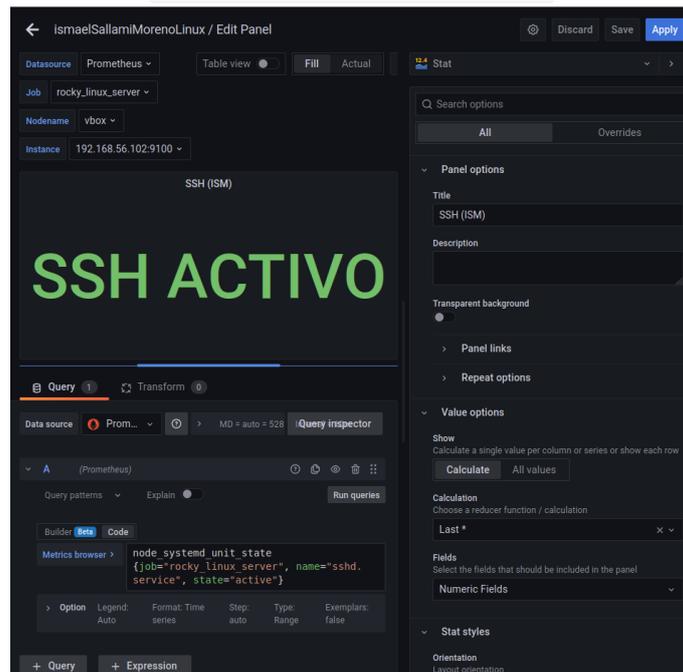


Figura 4.39: Añadido del panel del ssh

13. Ahora debemos de hacer lo mismo para httpd:

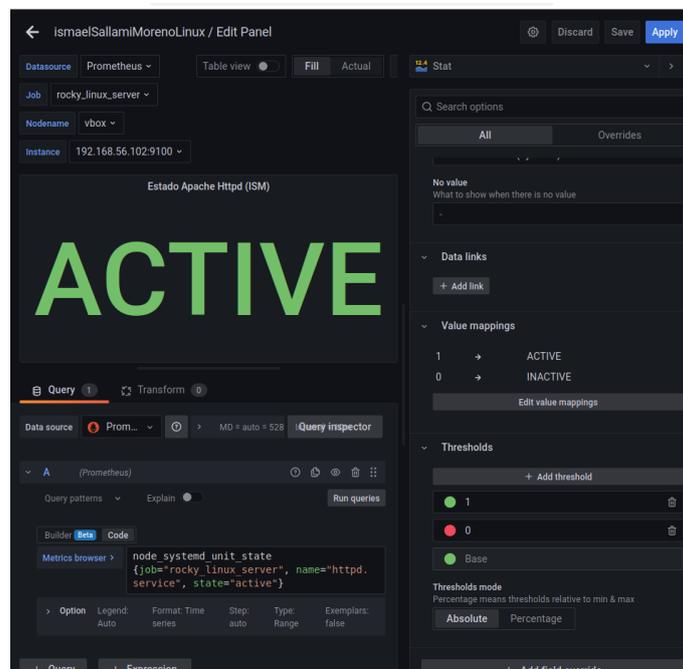


Figura 4.40: Añadido del panel del httpd

De manera conjunta nos quedaría de la siguiente manera:

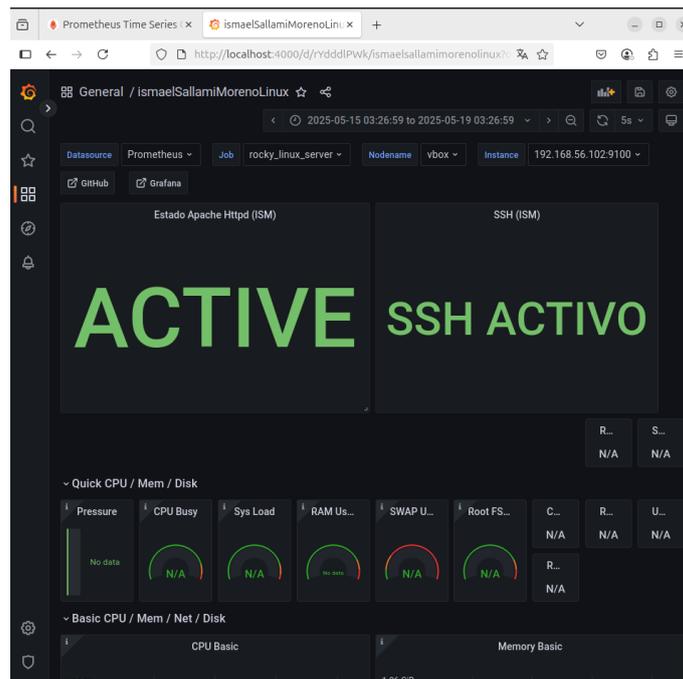


Figura 4.41: Añadido de los paneles del ssh y httpd

Nota: Las expresiones regulares que he utilizado para el ssh y httpd aparecen en las imágenes de arriba.

14. Ahora añadimos el panel de la CPU:

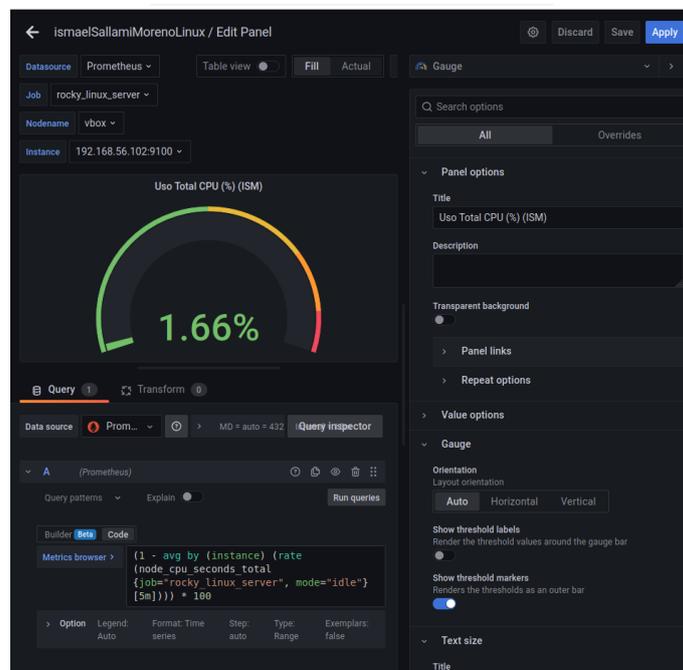


Figura 4.42: Añadido del panel de la CPU

15. Ahora añadimos la alarma al panel de la CPU:

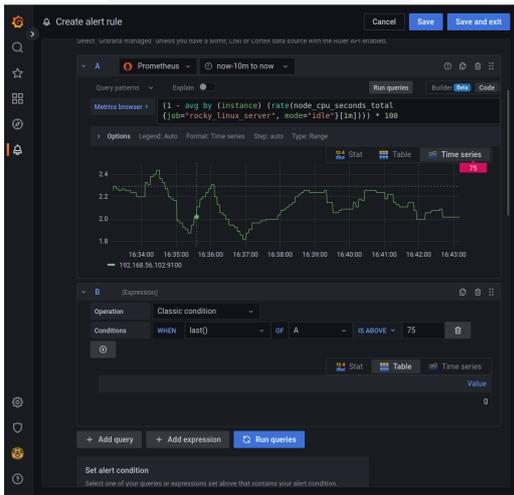


Figura 4.43: Primera Parte de la alarma

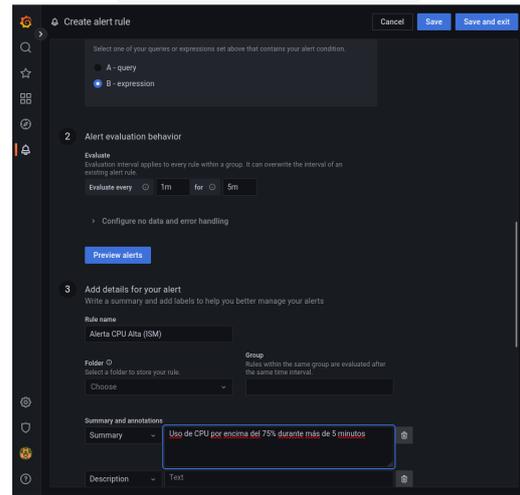


Figura 4.44: Segunda Parte de la alarma

16. De manera que la alarma nos quedaría de la siguiente manera:

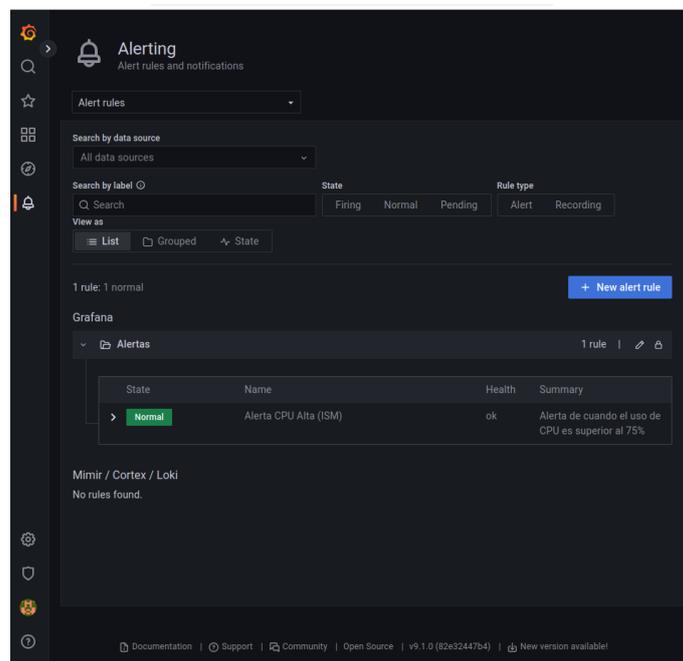


Figura 4.45: Alarma de la CPU

17. Ahora vamos a ver el estado inicial y el estado final de la CPU.



Figura 4.46: Estado inicial de la CPU

Para instalar el stress en la máquina virtual de Rocky Linux, ejecutamos el siguiente comando:

```

1  sudo dnf install epel-release -y
2  sudo dnf makecache
3  sudo dnf install stress -y
4

```

Vemos que si ejecutamos el comando `stress --cpu 1 --timeout 360s` la carga en grafana va a subir poco a poco y cuando supere el 75% de CPU durante 5 minutos se disparará la alarma:



Figura 4.47: Alto uso de CPU gracias al stress

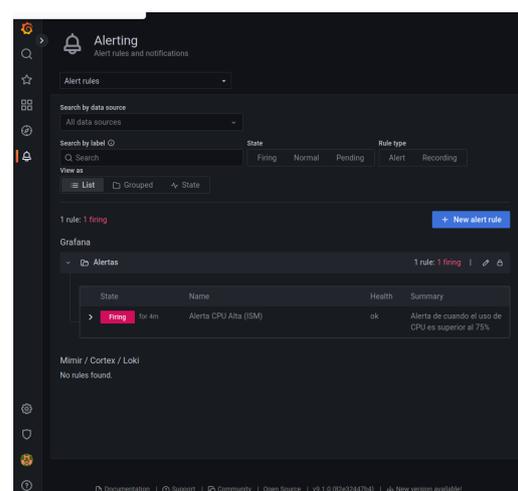


Figura 4.48: Disparo de la alarma

Luego al cancelar el stress, la carga de CPU baja y la alarma se apaga:

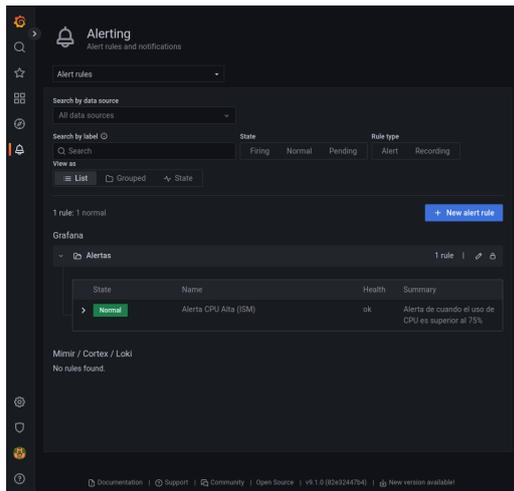


Figura 4.49: Vuelta al uso normal de CPU

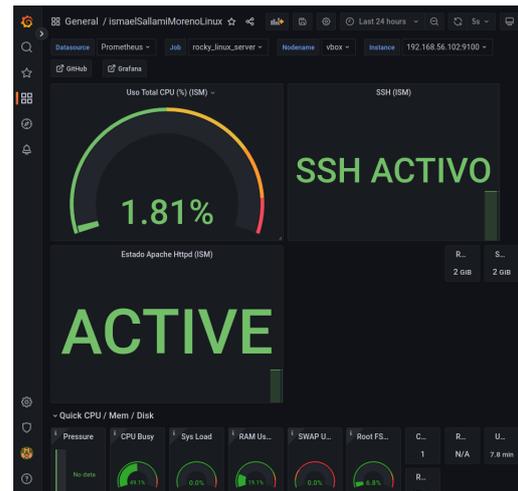


Figura 4.50: Estado normal de la alarma

El comando ha sido:

```
[ismMV01@vbox-17:28:48 ~]$ stress --cpu 1 --timeout 360s
```

Figura 4.51: Comando de stress

Ahora pasamos a la parte de la resolución de la API WEB.

Para la realización de esta parte debemos de tener en cuenta el ejercicio de prueba de carga con JMETTER, para ello lanzamos el contenedor de la API Web y entramos en la url <http://localhost:3000/metrics>.

Una vez añadido los paneles y demás ¹³, vamos a ver como queda el antes y después de los paneles tras la carga de JMeter.

¹³A continuación se adjuntan las expresiones regulares que he utilizado para cada uno de los paneles.

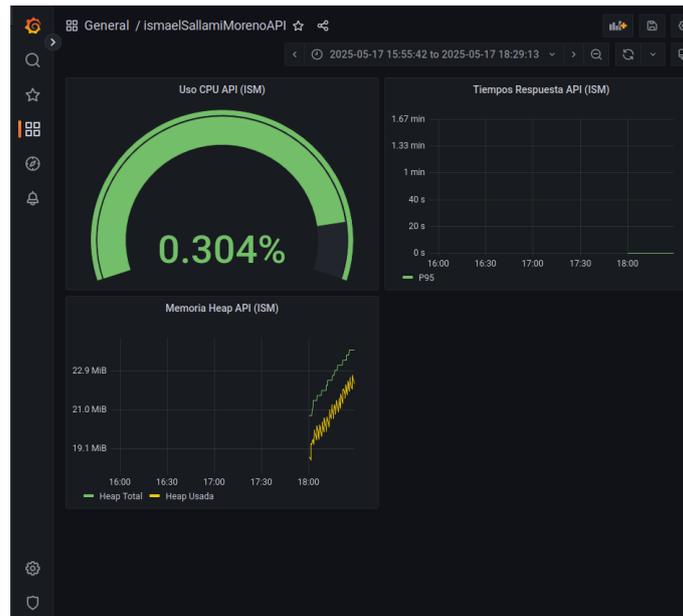


Figura 4.52: Paneles de la API Web Antes de la carga

Por último, tras la carga de JMeter, los paneles quedan de la siguiente manera:

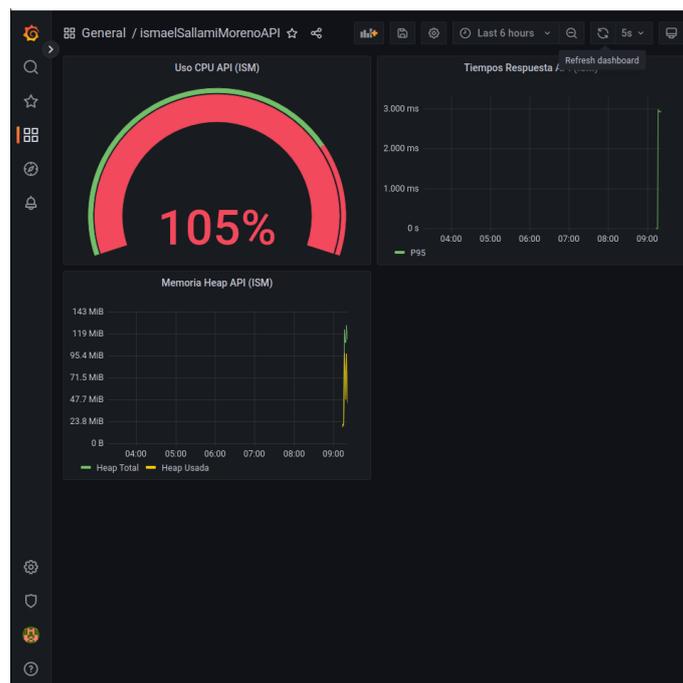


Figura 4.53: Paneles de la API Web Después de la carga

Bibliografía

- [1] Ismael Sallami Moreno, **Estudiante del Doble Grado en Ingeniería Informática + ADE**, Universidad de Granada, 2025.
- [2] Introduction to ad hoc commands - Ansible Documentation. https://docs.ansible.com/ansible/latest/command_guide/intro_adhoc.html
- [3] Index of all Modules — Ansible Community Documentation. https://docs.ansible.com/ansible/latest/collections/index_module.html
- [4] Ansible playbooks — Ansible Community Documentation. https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html