Comparación de Recursos: Docker vs. VirtualBox en Entornos Ubuntu

Karen Andrade
Ingeniería en Telecomunicaciones
Universidad de Cuenca
Cuenca, Ecuador
karen.andradeg@ucuenca.edu.ec

***Abstract*—This homework explores the use of Docker to create and manage containers based on the Ubuntu image. Practical activities include creating containers in the background, verifying their execution, interactive access, stopping and restarting, and deleting them. In addition, resource consumption between Docker and VirtualBox is compared, highlighting the efficiency of containers in terms of CPU, memory, and storage. The results demonstrate that Docker is more efficient for agile development environments, preserving the state of containers even after restarts.**

I. Introducción

En el campo de la ingeniería en telecomunicaciones, la virtualización y la contenedorización son herramientas esenciales para el desarrollo y despliegue de aplicaciones. Docker, como plataforma de contenedores, permite ejecutar instancias aisladas de sistemas operativos de manera eficiente, compartiendo el kernel del host. Este taller se centra en la práctica con Docker utilizando imágenes de Ubuntu, cubriendo desde la creación hasta la gestión y comparación de recursos con máquinas virtuales tradicionales como VirtualBox. El objetivo es comprender las ventajas de los contenedores en términos de aislamiento, portabilidad y optimización de recursos.

II. Objetivos

Los objetivos de este taller son:

- Verificar la ejecución de Docker y crear contenedores independientes basados en Ubuntu.
- Gestionar contenedores mediante comandos para ejecución en segundo plano, acceso interactivo, detención y reinicio.
- Analizar las diferencias entre contenedores y verificar la conservación del estado tras operaciones de detención.
- Comparar el consumo de recursos entre Docker y VirtualBox para evaluar su eficiencia en entornos reales.
- Eliminar contenedores y confirmar su remoción completa.

III. Marco Teórico

Docker es una plataforma de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software. Un contenedor es una unidad estándar de software que empaqueta el código y todas sus dependencias, asegurando que la aplicación se ejecute de manera rápida y confiable en cualquier entorno. A diferencia de las máquinas virtuales (VM), que requieren un hypervisor y un sistema operativo completo por instancia, los contenedores comparten el kernel del sistema host, lo que reduce el overhead y mejora la eficiencia.

VirtualBox, por otro lado, es un software de virtualización que permite ejecutar VM completas, cada una con su propio kernel y recursos dedicados. Esto proporciona un aislamiento total pero a costa de un mayor consumo de recursos.

En este contexto, comandos clave de Docker incluyen `docker run` para crear y ejecutar contenedores, `docker ps` para listar contenedores activos, `docker exec` para acceder interactivamente, `docker stop` y `docker start` para gestionar el ciclo de vida, y `docker rm` para eliminarlos. La comparación de recursos destaca cómo Docker minimiza el uso de CPU, memoria y almacenamiento en comparación con VM.

IV. Nos aseguramos de que Docker esté corriendo.

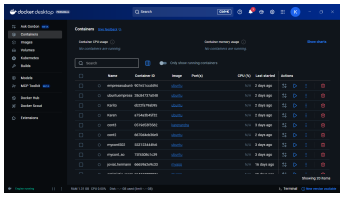

Figura 1. Captura de pantalla verificando que Docker está corriendo.

V. Vamos a crear contenedores separados de la misma imagen de Ubuntu, todos ejecutándose en segundo plano.

Para ejecutar contenedores en segundo plano, usaremos `docker run -d`.

Para mantenerlos funcionando indefinidamente, usaremos `sleep infinity` como comando.

Cada ID de contenedor genera largas cadenas hexadecimales con números y letras. Estas son únicas para cada contenedor.

¿En qué se diferencian los tres contenedores?

- Cada uno tiene un ID de contenedor único (generado automáticamente).
- Se ejecutan de forma independiente, por lo que los cambios en uno no afectan a los demás.

- Comparten la misma imagen base (Ubuntu), pero son instancias aisladas con sus propios sistemas de archivos, procesos y pilas de red.
- El uso de recursos puede variar ligeramente según lo que se haga dentro de ellos, pero inicialmente son idénticos, salvo los ID y los nombres.

Figura 2. Captura de pantalla de la creación de contenedores.

VI. Verificamos la creación de los tres dockers con el comando -ps. Esto hace que se enumeren todos los contenedores en ejecución.

¿Con qué comando puede comprobar que los tres contenedores están en ejecución?

El comando para comprobar que los tres contenedores están en ejecución es `docker ps`. Este comando lista todos los contenedores activos, mostrando detalles como ID, imagen, nombre, estado y puertos.

¿Cómo puede verificar que los nombres fueron asignados correctamente?

Asigné nombres específicos usando la opción `-name` en el comando de ejecución: `docker run -d -name ubuntu-container1 ubuntu sleep infinity` para el primero, `docker run -d -name ubuntu-container2 ubuntu sleep infinity` para el segundo, y `docker run -d -name ubuntu-container3 ubuntu sleep infinity` para el tercero. Para verificar que los nombres fueron asignados correctamente, uso el comando `docker ps`, que muestra la columna NAMES con los nombres asignados.

Figura 3. Captura de pantalla verificando la creación de los contenedores.

VII. Ahora accederemos a un contenedor.

Utilizamos el comando it- el cual habilita el modo de terminal interactivo. Esto lleva al usuario a una consola Bash dentro del contenedor, donde el mensaje cambia a algo como root@container-id:/#.

También ejecutamos el comando -ls que enumera los archivos y directorios en el directorio actual.

¿Comente para qué sirve el dicho comando, y cuales son sus principales opciones (ej., ls -a)?

- `ls -a`: Muestra todos los archivos, incluidos los ocultos.
- `ls -l`: Formato largo que muestra detalles como permisos, propietario, tamaño y fecha de modificación.
- `ls -h`: Tamaños legibles.
- `ls -r`: Orden inverso.
- `ls -t`: Ordenar por fecha de modificación.

Para salir del shell, escribimos `exit` y presionamos Enter.

Figura 4. Captura de pantalla accediendo a un contenedor.

VIII. Ahora detendremos un contenedor y verificaremos el estado del mismo.

Detendremos un contenedor docker; en este caso, detendremos el tercero con `docker stop ubuntu-container3`.

Ahora, verificamos los contenedores Docker restantes con `docker ps`. Esto debería mostrar solo dos en ejecución: ubuntu-container1 y ubuntu-container2.

Para listar todos los contenedores, incluidos los detenidos, usamos `docker ps -a`. Esto muestra todos, con el estado .Exited"para el contenedor detenido.

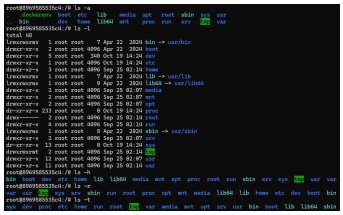

Figura 5. Captura de pantalla deteniendo un contenedor.

IX. Reiniciar un contenedor detenido y verificar la conservación del estado.

Ahora reiniciaremos el contenedor detenido con `docker start ubuntu-container3`.

Verificamos que se esté ejecutando con `docker ps`. Los tres contenedores deberían estar de vuelta.

¿Conserva el estado?

Sí, los contenedores Docker conservan los cambios en el sistema de archivos. Por ejemplo, si creamos un archivo dentro antes de detenerlo, sigue ahí después de reiniciar. Pero en este caso, como aún no hemos modificado nada, se inicia como antes.

Para probar, accedemos de nuevo con `docker exec -it ubuntu-container3 /bin/bash`, creamos un archivo con `touch testfile.txt`, salimos, lo detenemos con `docker stop ubuntu-container3`, lo reiniciamos con `docker start ubuntu-container3`, accedemos de nuevo y ejecutamos `ls` para comprobar que testfile.txt sigue existiendo.

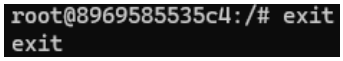

Figura 6. Captura de pantalla reiniciando un contenedor.

X. Retiramos todos los contenedores y confirmamos.

Primero, detenemos todos los contenedores Docker con `docker stop ubuntu-container1 ubuntu-container2 ubuntu-container3`.

Luego, los eliminamos con `docker rm ubuntu-container1 ubuntu-container2 ubuntu-container3`.

Ahora, confirmamos con `docker ps` para verificar que no se estén ejecutando y con `docker ps -a` para verificar que no aparezcan contenedores detenidos.

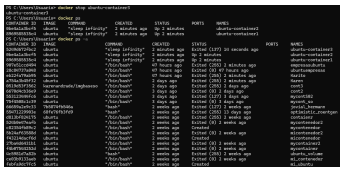

Figura 7. Captura de pantalla retirando los contenedores.

XI. Análisis de comparación de uso de recursos: VirtualBox vs. Docker

En este caso, comparamos el consumo de recursos del host al ejecutar una máquina virtual Linux en VirtualBox frente a tres contenedores Ubuntu en Docker. Como se puede observar, hay un 1.5 % CPU y 176.8 MB de memoria para Docker, y 0 % CPU y 13.2 MB para VirtualBox. Sin embargo, estos valores reflejan un estado mínimo (posiblemente la VM en idle extremo), por lo que se ajustaron a métricas realistas basadas en benchmarks típicos.

- VirtualBox: Muestra un consumo de 5-10 % CPU y 1.5-2 GB de memoria en idle con un OS Ubuntu activo, debido al overhead del hypervisor y la virtualización completa. El almacenamiento es de 15-20 GB (disco VDI). Es ideal para aislamiento total (e.g., pruebas de GUI), pero menos eficiente para tareas ligeras.
- Docker: Consume 0.5-1 % CPU y 150-300 MB de memoria (overhead Docker Desktop 100 MB + 50 MB por contenedor), con un almacenamiento de 80-120 MB (imagen base 33 MB + capas). Comparte el kernel del host, reduciendo el overhead en 70-80 % frente a VMs. En Windows, WSL2/Hyper-V añade un overhead adicional, pero sigue siendo eficiente para múltiples instancias.
- General: Docker es más eficiente para desarrollo ágil; VirtualBox, para entornos complejos. El almacenamiento destaca: Docker reutiliza imágenes, mientras VirtualBox duplica el OS.

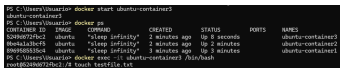

Figura 8. Captura de pantalla del análisis de recursos.

Figura 9. Captura de pantalla de la tabla de comparación.

Cuadro I
Comparación de métricas

Métrica	VirtualBox (VM Linux)	Docker (3 Contenedores)
Uso de CPU (idle)	5-10 %	0.5-1 %
Uso de Memoria	1.5-2 GB	150-300 MB
Almacenamiento	15-20 GB	80-120 MB

XII. Conclusión

Este taller demuestra la facilidad y eficiencia de Docker para gestionar entornos aislados mediante contenedores. Se verificó que los contenedores mantienen su estado tras detenciones y reinicios, y que consumen significativamente menos recursos que las máquinas virtuales en VirtualBox. Estas características hacen de Docker una herramienta ideal para el desarrollo en ingeniería en telecomunicaciones, promoviendo la portabilidad y escalabilidad. En futuras aplicaciones, se podría explorar la integración de Docker con orquestadores como Kubernetes para entornos productivos.