**Trabajo Integrador Arquitectura y Sistemas Operativos**

**Virtualización**

**Alumnos:**

* 1

**Materia:**

**Arquitectura y Sistemas Operativos**

**Profesor:**

**Tutor:**

**Fecha de entrega:**

05 de junio de 2025

# Índice

1. Introducción
2. Marco Teórico
3. Caso Práctico
4. Metodología Utilizada
5. Resultados Obtenidos
6. Conclusiones
7. Bibliografía
8. Anexos

**Introducción**

La virtualización constituye un pilar esencial en la informática moderna, permitiendo la creación de entornos aislados que optimizan el uso de recursos y facilitan el desarrollo, despliegue y mantenimiento de aplicaciones. Este tema se seleccionó debido a su relevancia en el desarrollo y despliegue de aplicaciones modernas, ya que la virtualización posibilita la ejecución de múltiples sistemas o aplicaciones en un mismo hardware físico, mejorando la eficiencia y la escalabilidad. En el contexto de la formación como técnico en programación, comprender la virtualización es fundamental, ya que permite gestionar entornos de desarrollo y producción de manera eficiente, especialmente mediante el uso de herramientas como Docker, que agilizan la creación de contenedores ligeros y portátiles. El objetivo de este trabajo es explorar los conceptos fundamentales de máquinas virtuales y contenedores, implementar un contenedor Docker que ejecute un programa Python simple y analizar su funcionamiento para comprender las ventajas y desafíos de esta tecnología en un entorno práctico.

**Marco Teórico**

La virtualización es una tecnología que permite crear representaciones abstractas de recursos físicos o lógicos, como hardware, sistemas operativos o aplicaciones, para ejecutar múltiples entornos aislados en un solo sistema físico. Según Tanenbaum y Bos (2015), la virtualización abstrae los recursos físicos de un ordenador, permitiendo que múltiples sistemas operativos o aplicaciones se ejecuten de manera independiente en el mismo hardware.

**Máquinas Virtuales (VM):** Una máquina virtual emula un sistema completo, incluyendo hardware virtualizado (CPU, memoria, almacenamiento, red). Cada VM incluye un sistema operativo completo y es gestionada por un hipervisor, lo que garantiza un aislamiento robusto pero con un mayor consumo de recursos debido a la emulación del hardware.

**Contenedores:** Los contenedores son entornos ligeros que comparten el kernel del sistema operativo anfitrión, encapsulando únicamente las aplicaciones y sus dependencias (bibliotecas, configuraciones). Esto los hace más eficientes en términos de recursos, ya que no requieren un sistema operativo completo por contenedor. Según la documentación ofi- cial de Docker (consultada el 30/05/2025), los contenedores son ideales para aplicaciones que necesitan portabilidad y escalabilidad.

**Diferencias clave:**

* + *VMs:* Mayor aislamiento, pero mayor consumo de recursos (memoria, CPU). Cada VM ejecuta un sistema operativo completo.
  + *Contenedores:* Menor aislamiento (comparten el kernel), pero más ligeros y rápidos. Ideales para microservicios.

1. **Clasificaciones**

La virtualización se clasifica en varios tipos según los recursos abstraídos:

* + **Virtualización de hardware:** Crea entornos completos que emulan hardware físico, gestionados por hipervisores. Ejemplo: VMware ESXi.
  + **Virtualización de software:** Permite ejecutar aplicaciones en entornos aislados sin emular hardware completo. Ejemplo: entornos virtuales de Python (venv).
  + **Virtualización de contenedores:** Abstrae el entorno de ejecución de aplica- ciones, compartiendo el kernel del anfitrión. Ejemplo: Docker.

**Hipervisores:**

* + *Tipo 1 (nativos o bare-metal):* Se ejecutan directamente sobre el hardware físico, sin un sistema operativo anfitrión. Ejemplo: VMware ESXi, que ofrece alto rendimiento y es común en centros de datos debido a su eficiencia y robustez.
  + *Tipo 2 (hosteados):* Se ejecutan sobre un sistema operativo anfitrión. Ejemplo: Oracle VirtualBox, que es más sencillo de configurar pero menos eficiente, ideal para entornos de desarrollo o pruebas.

1. **Estructura de Docker**

Docker es una plataforma de virtualización basada en contenedores que utiliza compo- nentes clave para su funcionamiento:

* + **Imágenes:** Archivos inmutables que contienen todo lo necesario para ejecutar una aplicación (código, dependencias, configuraciones). Las imágenes se crean a partir de Dockerfiles y se almacenan en registros como Docker Hub.
  + **Contenedores:** Instancias ejecutables de una imagen, que se ejecutan de forma aislada pero comparten el kernel del sistema operativo anfitrión.
  + **Dockerfiles:** Archivos de texto con instrucciones para construir imágenes de Docker. Contienen comandos como FROM, RUN, COPY, y ENTRYPOINT para definir el entorno y comportamiento de la imagen.

**Arquitectura de Docker:**

* + *Docker Engine:* El núcleo de Docker, compuesto por el demonio (dockerd) que gestiona contenedores, imágenes y redes, y una API REST para interactuar con él.
  + *Docker Client:* Interfaz de línea de comandos (docker) que permite a los usuarios interactuar con el Docker Engine.
  + *Registros:* Repositorios (como Docker Hub) para almacenar y distribuir imágenes.

1. **Comparación entre VMs y Contenedores**

Table 1: Comparación entre Máquinas Virtuales y Contenedores

|  |  |  |
| --- | --- | --- |
| **Característica** | **Máquinas**  **Virtuales** | **Contenedores** |
| **Aislamiento** | Alto (hardware y SO  completos) | Moderado (comparte  kernel) |
| **Uso de recursos** | Alto (SO completo  por VM) | Bajo (sin SO  completo) |
| **Tiempo de inicio** | Minutos | Segundos |
| **Portabilidad** | Moderada (depende  del hipervisor) | Alta (estándar  Docker) |
| **E**jemplo | VMware, VirtualBox | Docker, Podman |

**Diagrama Comparativo Referencias**

* + Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems*. Pearson.
  + Docker Inc. (2025). Documentación oficial de Docker. Recuperado el 30/05/2025 de https://docs.docker.com.

**Caso Practico:**

💻 Aquí se debe presentar un problema o situación concreta que haya sido desarrollada o simulada para aplicar el contenido del trabajo.

# Incluye:

* + Breve descripción del problema a resolver.
  + Capturas de pantalla si corresponde.
  + Validación del funcionamiento.

# Metodología Utilizada

 Describe los pasos seguidos durante el desarrollo del trabajo. Se pueden incluir:

* + Investigación previa (fuentes utilizadas).
  + Etapas de diseño y prueba del código.
  + Herramientas y recursos utilizados (IDE, librerías, control de versiones, etc.).
  + Trabajo colaborativo (reparto de tareas en el grupo de trabajo).

# Resultados Obtenidos

📊 Detalla qué se logró con el caso práctico, qué aspectos funcionaron correctamente y qué dificultades se presentaron.

Se pueden incluir:

* + Casos de prueba realizados.
  + Errores corregidos.

# Conclusiones

🎯 Reflexión final del grupo de trabajo. Aquí se recomienda incluir:

* + Qué se aprendió al hacer el trabajo.
  + Posibles mejoras o extensiones futuras.
  + Dificultades que surgieron si las hubo y cómo se resolvieron.

# Bibliografía

📚 Listado de fuentes consultadas, utilizando normas básicas APA u otro formato consistente. Se sugiere incluir:

* + Libros.
  + Documentación oficial.
  + Sitios web con fecha de acceso.
  + Artículos o materiales recomendados por el docente. Ejemplo:
  + Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts. John Wiley & Sons. Capítulo 3, "Procesos".
  + Stallings, W. (2023). Operating Systems: Internals and Design Principles.

# 📥 Recomendaciones para la presentación

* + Formato del archivo: **.pdf**
  + Tipografía: Arial o Calibri, tamaño 11 o 12
  + Interlineado: 1,5
  + Márgenes estándar (2.5 cm)
  + Portada opcional con el logo de la institución (si se requiere)
  + Entrega digital mediante plataforma institucional o por correo