# Datos Generales

* **Título del trabajo**: Virtualización – Fundamentos de Docker
* **Alumnos**: Rodrigo Aguirre – rodri\_wsc@hotmail.com
* **Materia**: AYSO
* **Profesor/a**:
* **Fecha de Entrega**: 05/06/2025

# Índice

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

# Introducción

El tema de Docker fue elegido debido a su creciente relevancia en el desarrollo y despliegue de aplicaciones modernas. Desde su nacimiento en 2013, Docker se ha convertido en una herramienta fundamental para desarrolladores e ingenieros.

Esta tecnología permite empaquetar aplicaciones junto con todas sus dependencias, garantizando que se ejecuten de manera consistente en cualquier entorno, ya sea local o en servidores remotos y cloud. Facilita la creación de entornos reproducibles, livianos y aislados. Su uso es transversal a lenguajes y tecnologías, y permite integrar fácilmente bases de datos, APIs, servicios web, y herramientas de análisis de datos en contenedores independientes.

Con este trabajo se busca alcanzar los siguientes objetivos:

* Comprender el concepto de virtualización mediante contenedores.
* Analizar las diferencias clave entre Docker y las máquinas virtuales tradicionales.
* Aplicar Docker en un ejemplo práctico de uso real (una app con Python y MySQL).

# Marco Teórico

La virtualización es una tecnología que permite ejecutar múltiples sistemas operativos o entornos de ejecución sobre un mismo hardware físico, de forma simultánea y aislada. Tradicionalmente, esto se logra a través de máquinas virtuales (VMs) que utilizan un software denominado hipervisor (bare-metal/nativo o software/alojado), el cual abstrae el hardware para permitir que distintos sistemas operativos operen como si fueran independientes. Esta técnica optimiza el uso de recursos y mejora la escalabilidad y la portabilidad de los sistemas.

Docker representa una evolución en el ámbito de la virtualización, aunque no reemplaza totalmente el uso de VMs, se trata de una forma de virtualización más ligera. Esta tecnología fue creada por Docker Inc en 2013 y está basada en contenedores, que permiten empaquetar aplicaciones y sus dependencias para correr en diferentes tipos de entorno, volviéndolas más portables. Su arquitectura esta compuesta por distintos elementos como:

* Docker Engine, es el corazón de Docker, que contiene el Docker Daemon que gestiona contenedores, imágenes, redes y volúmenes. El Docker CLI, la interfaz de línea de comandos para interactuar con el Daemon y la REST API que permite la interacción de otras herramientas y plataformas.
* Docker Image, es la plantilla del contenedor, define como se construye el mismo a través de instrucciones (capas) en un Dockerfile.
* Docker Containter, son instancias aisladas de una imagen.

A diferencia de las máquinas virtuales, los contenedores de Docker no requieren virtualizar un sistema operativo completo, lo cual consume gran cantidad de recursos (RAM y CPU). En su lugar, comparten el kernel del sistema host y aíslan procesos mediante tecnologías del sistema operativo Linux, como namespaces y cgroups. Esto los hace más ligeros, rápidos y eficientes en el uso de recursos.

Además, los contenedores son rápidos de crear y destruir, lo que vuelve a los sistemas fácilmente escalables dado que se pueden levantar tantas instancias de un mismo contenedor como sea requerido.

También podemos destacar que son fácilmente distribuibles a través de los registros públicos de imágenes como Docker Hub, aunque también se pueden gestionar registros privados para software propietario.

Para grandes proyectos existen tecnologías de orquestación de contenedores que permiten controlarlos de manera inteligente a gran escala, permiten desplegar contenedores de manera automática, o reiniciar contenedores que fallan, etc. Por ej: Kubernetes, Docker Swarm, entre otras.

Podemos resumir los usos típicos de Docker para:

1. Empaquetar aplicaciones con todas sus dependencias.
2. Evitar problemas de compatibilidad entre entornos (desarrollo, testing, producción).
3. Desplegar aplicaciones fácilmente en distintos servidores o nubes.
4. Aislar servicios (por ejemplo, una app web + base de datos + backend en contenedores separados).
5. Automatizar entornos de desarrollo, permitiendo que cualquier desarrollador trabaje con la misma configuración.

# Caso Práctico

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.
* Código fuente comentado.
* Capturas de pantalla si corresponde.
* Explicación de decisiones de diseño (por ejemplo: ¿por qué se eligió un método de ordenamiento sobre otro?).
* Validación del funcionamiento.

El código debe estar identado correctamente y con comentarios que expliquen los pasos importantes.

# 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 si se realizó en grupo).

# 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.
* Evaluación de rendimiento (si aplica, por ejemplo, comparar el tiempo de ejecución entre algoritmos).
* Enlace a repositorio si el trabajo está subido a GitHub u otra plataforma.

# Conclusiones

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

* Qué se aprendió al hacer el trabajo.
* Qué utilidad tiene el tema trabajado para la programación o para otros proyectos.
* Posibles mejoras o extensiones futuras.
* Dificultades que surgieron 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:

* Python Software Foundation. (2024). *Python 3 Documentation*. <https://docs.python.org/3/>
* Sweigart, A. (2019). *Automate the Boring Stuff with Python*. No Starch Press.

# Anexos

Material adicional que no va en el cuerpo principal del trabajo pero que aporta valor.

Pueden ser:

* Capturas del programa funcionando.
* Enlace al video explicativo.
* Código completo como archivo externo o adjunto.
* Cuadros comparativos.
* Documentos auxiliares (como diagramas de flujo).

# Recomendaciones para la presentación

* Formato del archivo: **.docx** o **.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