

```
In [1]: #  
\n

Aquí está cómo funciona la analogía:

**Estandarización:** Al igual que los contenedores de envío vienen en tamaños estandarizados que se ajustan a varios tipos de barcos, camiones y trenes, los contenedores Docker utilizan un formato estandarizado que puede ejecutarse en cualquier sistema que admita Docker. Esto asegura consistencia y portabilidad en diferentes entornos.

**Aislamiento:** Los contenedores de envío aíslan las mercancías de factores externos como el clima, plagas y robos. De manera similar, los contenedores Docker aíslan las aplicaciones y sus dependencias del sistema host y otros contenedores, previniendo conflictos y garantizando un comportamiento consistente.

**Empaquetado:** Los contenedores de envío permiten que varios tipos de mercancías se empaqueten y envíen juntos. Los contenedores Docker empaquetan una aplicación junto con su tiempo de ejecución, bibliotecas y dependencias, creando una unidad autosuficiente que puede enviarse como una entidad única.

**Facilidad de Manejo:** Los contenedores de envío están diseñados para una carga, descarga y transporte fáciles. Los contenedores Docker son fáciles de crear, mover e implementar en diferentes entornos, lo que hace que el proceso de desarrollo y despliegue sea más eficiente.

**Eficiencia:** Los contenedores de envío optimizan el espacio, permitiendo que las mercancías se apilen y transporten de manera eficiente. Los contenedores Docker comparten el núcleo del sistema operativo del sistema host, lo que optimiza la utilización de recursos y permite que varios contenedores se ejecuten en el mismo host.

**Modularidad:** Los contenedores de envío permiten que las mercancías se compartimenten y se carguen de manera independiente. Los contenedores Docker promueven un enfoque modular para el desarrollo de aplicaciones, donde cada componente o servicio puede encapsularse en su propio contenedor.

**Intercambiabilidad:** Los contenedores de envío se pueden transferir fácilmente entre diferentes modos de transporte (barcos, camiones, trenes). Los contenedores Docker se pueden mover entre diferentes entornos (desarrollo, pruebas, producción) e incluso entre proveedores de la nube con cambios mínimos.

**Consistencia:** Los contenedores de envío aseguran que las mercancías estén protegidas y se manejen de manera consistente durante todo su recorrido. Los contenedores Docker aseguran que las aplicaciones se ejecuten de manera consistente en diferentes etapas del ciclo de vida de desarrollo.

**Implementación Sencilla:** Los contenedores de envío se pueden cargar en varios vehículos sin necesidad de volver a empaquetar las mercancías. Los contenedores Docker se pueden implementar en diferentes sistemas sin necesidad de modificar la aplicación, garantizando un comportamiento consistente.

**Flexibilidad:** Al igual que los contenedores de envío pueden contener una amplia variedad de mercancías, los contenedores Docker pueden empaquetar aplicaciones de diferentes tipos y tecnologías. Esto los hace adecuados para una variedad diversa de aplicaciones.

La analogía ayuda a las personas a comprender los conceptos fundamentales de los contenedores Docker al compararlos con un concepto familiar y tangible: los contenedores de envío. Simplifica las complejidades de la contenerización, el aislamiento y la portabilidad, lo que facilita que las personas comprendan los beneficios y casos de uso de los contenedores Docker en el desarrollo y despliegue de software moderno.

## 1.2 ¿Cuál es la motivación para usar Docker?

---

La motivación para usar Docker surge de varios beneficios clave y desafíos enfrentados en el desarrollo y despliegue de software. Docker aborda estos desafíos y proporciona una manera más eficiente, consistente y simplificada de gestionar aplicaciones y sus dependencias. Aquí están algunas de las principales motivaciones para usar Docker:

**Consistencia en Diferentes Entornos:** Docker asegura que las aplicaciones se ejecuten de manera consistente en diferentes entornos, como desarrollo, pruebas y producción. Esto elimina el problema de "funciona en mi máquina" y minimiza las discrepancias entre los entornos de desarrollo e implementación.

**Aislamiento y Gestión de Dependencias:** Los contenedores aíslan las aplicaciones y sus dependencias del sistema host y entre sí. Esto previene conflictos entre diferentes aplicaciones y permite la gestión eficiente de dependencias sin interferir con otras partes del sistema.

**Portabilidad:** Los contenedores Docker pueden ejecutarse en cualquier sistema que admita Docker, independientemente de la infraestructura subyacente. Esta portabilidad facilita mover aplicaciones entre entornos de desarrollo, pruebas y producción, así como entre entornos locales y en la nube.

**Eficiencia y Utilización de Recursos:** Los contenedores comparten el núcleo del sistema operativo del sistema host, lo que los hace livianos y rápidos para iniciar y detener. Esta eficiente utilización de recursos permite que más contenedores se ejecuten en el mismo host en comparación con las máquinas virtuales tradicionales.

**Despliegue Rápido:** Docker permite el despliegue rápido de aplicaciones, ya que los contenedores pueden crearse, iniciarse y detenerse rápidamente. Esto es especialmente valioso para las tuberías de integración y despliegue continuo (CI/CD).

**Escalabilidad:** La arquitectura basada en contenedores de Docker permite escalar horizontalmente las aplicaciones al ejecutar varios contenedores de la misma imagen. Esta escalabilidad es esencial para manejar cargas de trabajo incrementadas.

**Control de Versiones y Reproducibilidad:** Las imágenes de Docker pueden tener versiones, lo que permite a los desarrolladores mantener configuraciones consistentes y volver fácilmente a versiones anteriores si es necesario. Esto garantiza que las diferentes etapas de desarrollo e implementación utilicen la misma imagen.

**Simplificación de DevOps:** Docker simplifica la colaboración entre los equipos de desarrollo y operaciones. Simplifica el proceso de empaquetar, distribuir e implementar aplicaciones, reduciendo conflictos y fricciones entre equipos.

**Arquitectura de Microservicios:** Docker se usa comúnmente en la arquitectura de microservicios, donde las aplicaciones se desglosan en servicios más pequeños y acoplados de manera flexible. Cada servicio se ejecuta en su propio contenedor, lo que permite el desarrollo, implementación y escalado independientes de diferentes componentes.

**Ecosistema y Comunidad:** Docker cuenta con un próspero ecosistema con una amplia gama de imágenes preconstruidas disponibles en Docker Hub. La comunidad de Docker comparte mejores prácticas, tutoriales y herramientas que facilitan la adopción de la contenerización.

**Resumen:** La motivación de Docker radica en su capacidad para simplificar el proceso de desarrollo y despliegue de software al proporcionar una forma estandarizada de empaquetar, distribuir y ejecutar aplicaciones, al mismo tiempo que aborda desafíos relacionados con la consistencia, la gestión de dependencias, la portabilidad, la eficiencia y la escalabilidad.

## 2. Contenerización y Aislamiento:

---

Contenerización y Aislamiento son conceptos fundamentales en Docker que desempeñan un papel crucial en cómo se implementan, gestionan y ejecutan las aplicaciones. Vamos a explorar estos conceptos en detalle:

- **Contenerización:** Es el proceso de empaquetar una aplicación y sus dependencias en una única unidad llamada contenedor. Los contenedores proporcionan un entorno consistente y aislado para ejecutar aplicaciones, asegurando que se comporten de la misma manera en diferentes entornos, desde el desarrollo hasta la producción.
- **Aislamiento:** El aislamiento en Docker se refiere a la separación de recursos, procesos y entornos entre contenedores y entre contenedores y el sistema anfitrión. El aislamiento garantiza que los contenedores no interfieran entre sí y operen como si estuvieran ejecutándose en sus propios entornos independientes.



## 3. Imagen de Docker:

---

- Una imagen de Docker es un paquete ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar un fragmento de software, incluido el código, el tiempo de ejecución, las herramientas del sistema, las bibliotecas y las configuraciones.
- Las imágenes de Docker sirven como plantillas para crear contenedores. Se construyen a partir de un conjunto de instrucciones definidas en un archivo especial llamado Dockerfile.



Aquí están los componentes clave y los detalles que definen lo que es una imagen de Docker:

1. **Capas:** Las imágenes de Docker están compuestas por múltiples capas. Cada capa representa un conjunto de cambios en el sistema de archivos. Las capas se apilan una encima de la otra, y cada capa representa un cambio o modificación en el sistema de archivos. Este enfoque en capas permite un uso eficiente del espacio en disco y un intercambio de imágenes más rápido.

2. **Imagen Base:** Las imágenes suelen basarse en una imagen base, que es el punto de partida para construir tu imagen personalizada. La imagen base puede ser una imagen oficial proporcionada por Docker o una imagen personalizada creada por ti u otros en la comunidad.
3. **Dockerfile:** Las imágenes se definen mediante un Dockerfile, que es un archivo de configuración de texto plano que especifica las instrucciones para construir la imagen. El Dockerfile incluye detalles como la imagen base, las instalaciones de software adicionales, las variables de entorno, el directorio de trabajo y más.
4. **Inmutable:** Las imágenes de Docker son inmutables, lo que significa que una vez creadas, su contenido no puede cambiarse. Si se requieren modificaciones, se debe crear una nueva imagen basada en la existente.
5. **Versionado:** Las imágenes pueden tener versiones mediante etiquetas. Las etiquetas son etiquetas que indican una versión específica de la imagen. Por ejemplo, una imagen puede tener etiquetas como "latest", "v1.0" o "dev".
6. **Reproducibilidad:** Las imágenes de Docker ofrecen una forma de asegurarse de que se utilice el mismo entorno en diferentes etapas del ciclo de vida de desarrollo de software, desde el desarrollo y las pruebas hasta la producción. Esta reproducibilidad reduce las inconsistencias y los problemas de implementación.
7. **Docker Hub y Registros:** Las imágenes de Docker se pueden compartir y distribuir a través de Docker Hub, un registro basado en la nube donde puedes encontrar y compartir imágenes públicas. También se pueden configurar registros privados para organizaciones que necesitan gestionar su propia distribución de imágenes.
8. **Eficiencia:** Las imágenes están diseñadas para ser ligeras. Docker utiliza un Sistema de Archivos de Unión para compartir capas entre imágenes, lo que optimiza el almacenamiento y reduce la duplicación de archivos.
9. **Aislamiento de Aplicaciones:** Las imágenes de Docker proporcionan una forma de aislar aplicaciones y sus dependencias del sistema anfitrión y de otras aplicaciones. Este aislamiento asegura que las aplicaciones se ejecuten de manera consistente sin importar el entorno del host.
10. **Facilidad de Implementación:** Las imágenes de Docker facilitan el empaquetado de aplicaciones y sus dependencias en una unidad única. Esta unidad se puede implementar y ejecutar fácilmente en cualquier entorno compatible con Docker.

**Resumen:** una imagen de Docker es una instantánea de un sistema de archivos que contiene todo lo necesario para ejecutar una aplicación. Sirve como una unidad portátil, consistente y aislada que permite la implementación y ejecución de aplicaciones en diversos entornos. Las imágenes se definen mediante Dockerfiles y se pueden versionar, compartir y distribuir a través

## 4. Dockerfile:

---

Un Dockerfile es un archivo de configuración de texto plano utilizado para definir las instrucciones para construir una imagen de Docker. Esboza los pasos y comandos necesarios para crear un paquete de software personalizado, autocontenido y ejecutable que se puede ejecutar como un contenedor de Docker. Los Dockerfiles son un componente fundamental del ecosistema de Docker y son cruciales para crear entornos consistentes, reproducibles y portátiles para las aplicaciones.



## 5. Docker Container:

Un contenedor de Docker es un paquete de software liviano, independiente y ejecutable que contiene todo lo necesario para ejecutar un fragmento de software, incluido el código de la aplicación, el tiempo de ejecución, las bibliotecas, las variables de entorno y las herramientas del sistema. Los contenedores se crean a partir de imágenes de Docker y proporcionan un entorno consistente y aislado para ejecutar aplicaciones. Encapsulan aplicaciones y sus dependencias, lo que les permite ejecutarse de manera confiable en diferentes entornos sin causar conflictos ni interferir con otras aplicaciones.



Aquí están los componentes clave y los detalles que definen lo que es un contenedor de Docker:

1. **Aislamiento:** Los contenedores aíslan las aplicaciones y sus dependencias del sistema host y de otros contenedores. Este aislamiento asegura que los cambios en un contenedor no afecten a otros, proporcionando un entorno limpio y controlado para ejecutar aplicaciones.
2. **Estandarización:** Los contenedores de Docker son unidades estandarizadas que empaquetan aplicaciones y sus dependencias juntas. Esta estandarización simplifica los procesos de implementación, pruebas y mantenimiento.
3. **Portabilidad:** Los contenedores pueden ejecutarse en cualquier sistema que admita Docker, independientemente de la infraestructura subyacente. Esta portabilidad facilita mover aplicaciones entre entornos de desarrollo, pruebas y producción, así como entre entornos locales y en la nube.
4. **Sistema de Archivos y Bibliotecas:** Los contenedores tienen su propio sistema de archivos aislado que contiene el código de la aplicación, el tiempo de ejecución, las bibliotecas y otros archivos necesarios. Este sistema de archivos aislado asegura que el contenedor tenga el entorno requerido para ejecutar la aplicación.
5. **Eficiencia de Recursos:** Los contenedores comparten el núcleo del sistema operativo del sistema host, lo que los hace livianos y eficientes en el uso de recursos. Consumen menos recursos en comparación con las máquinas virtuales tradicionales.
6. **Gestión de Dependencias:** Los contenedores encapsulan las dependencias de la aplicación, asegurando que sean consistentes y estén aisladas del sistema host y otros contenedores. Esto elimina conflictos entre diferentes requisitos de aplicaciones.
7. **Inmutable:** Los contenedores son inmutables, lo que significa que su contenido no puede cambiarse una vez creados. Si necesitas hacer cambios, creas un nuevo contenedor a partir de una imagen actualizada.
8. **Redes:** Los contenedores pueden conectarse a diferentes redes, lo que permite la comunicación entre contenedores o con recursos externos. Esta capacidad de redes permite la creación de arquitecturas de microservicios.

9. **Versionado:** Los contenedores pueden tener versiones mediante etiquetas de imagen. Diferentes versiones de una aplicación se pueden mantener y desplegar mediante etiquetas de imagen específicas.
10. **Gestión del Ciclo de Vida:** Los contenedores tienen un ciclo de vida bien definido, desde la creación hasta la ejecución y la terminación. Docker proporciona comandos para gestionar la creación, inicio, detención, pausa y eliminación de contenedores.
11. **Seguridad y Aislamiento:** Los contenedores proporcionan una capa adicional de seguridad al aislar las aplicaciones entre sí y del sistema host. Esto ayuda a prevenir que las vulnerabilidades en una aplicación afecten a otras.
12. **Implementación y Escalado de Aplicaciones:** Los contenedores simplifican la implementación y el escalado de aplicaciones. Pueden ser fácilmente orquestados y gestionados mediante herramientas como Docker Compose, Kubernetes y Docker Swarm.

**Resumen:** un contenedor de Docker es una unidad autocontenido que empaqueta una aplicación y sus dependencias, proporcionando un entorno consistente, aislado y portátil para ejecutar software en diferentes sistemas y entornos. Los contenedores son una tecnología

## 5. Docker Hub and Registries:

---

Docker Hub y los registros de contenedores son plataformas diseñadas para almacenar, gestionar, distribuir y compartir imágenes de Docker. Tienen un papel fundamental en el ecosistema de Docker al proporcionar un lugar centralizado para que los desarrolladores y las organizaciones carguen, accedan y colaboren en imágenes de contenedores. Aquí tienes una explicación del flujo de trabajo de Docker Hub y los registros de contenedores:



Docker Hub y los registros de contenedores son plataformas diseñadas para almacenar, gestionar, distribuir y compartir imágenes de Docker. Tienen un papel crítico en el ecosistema de Docker al proporcionar un lugar centralizado para que los desarrolladores y las organizaciones carguen, accedan y colaboren en imágenes de contenedores. Aquí tienes una explicación detallada de Docker Hub y los registros de contenedores:

### Docker Hub:

Docker Hub es el registro de contenedores en la nube predeterminado y más ampliamente utilizado. Sirve como un repositorio público para imágenes de Docker, permitiendo a los usuarios encontrar, compartir y colaborar en imágenes dentro de la comunidad de Docker. Algunos aspectos clave de Docker Hub incluyen:

1. **Hospedaje de Imágenes:** Docker Hub aloja una amplia colección de imágenes de Docker, incluidas las imágenes oficiales proporcionadas y mantenidas por Docker, así como las imágenes contribuidas por la comunidad.
2. **Repositorios Públicos y Privados:** Docker Hub admite tanto repositorios públicos como privados. Los repositorios públicos son accesibles para cualquier persona, mientras que los repositorios privados requieren autenticación para acceder. Los repositorios privados son útiles para almacenar imágenes propietarias o sensibles.

3. **Colaboración:** Docker Hub permite la colaboración al permitir a los usuarios publicar y compartir sus imágenes de Docker. Los desarrolladores y las organizaciones pueden contribuir con imágenes a la comunidad, poniéndolas a disposición de otros para su uso.
4. **Imágenes Oficiales:** Docker Hub aloja una colección de imágenes oficiales, que son mantenidas y proporcionadas por Docker. Estas imágenes sirven como imágenes base confiables para diversas tecnologías y marcos de trabajo.
5. **Versionado:** Las imágenes alojadas en Docker Hub pueden tener versiones mediante etiquetas. Esto permite a los usuarios acceder a versiones específicas de una imagen y ayuda a mantener la coherencia en las implementaciones.
6. **Compilaciones Automatizadas:** Docker Hub ofrece la opción de configurar compilaciones automatizadas para GitHub, Bitbucket y otros repositorios de control de versiones. Esto permite que las imágenes se construyan y actualicen automáticamente cada vez que se realizan cambios en el código.
7. **Integración:** Docker Hub se integra con diversas herramientas y plataformas, incluidos Docker Desktop, Docker Cloud (ahora integrado en Docker Hub), tuberías de integración y despliegue continuo (CI/CD) y herramientas de orquestación como Kubernetes.

### **Registros de Contenedores:**

Además de Docker Hub, existen otras soluciones de registro de contenedores que las organizaciones pueden implementar para su uso interno. Estos registros privados proporcionan un entorno controlado para almacenar y gestionar imágenes dentro de una organización. Algunos ejemplos incluyen:

1. **Registros Privados:** Las organizaciones pueden establecer registros privados de contenedores para almacenar sus propias imágenes de Docker. Esto es particularmente importante para aplicaciones propietarias o cuando se requiere un control más estricto sobre la distribución y el acceso de imágenes.
2. **Seguridad y Cumplimiento:** Los registros privados permiten a las organizaciones garantizar el cumplimiento de las políticas de seguridad y las regulaciones de la industria al gestionar imágenes dentro de su propia infraestructura.
3. **Eficiencia de Red:** Hospedar imágenes internamente puede ser más eficiente en términos de uso de la red, especialmente al tratar con imágenes grandes o en escenarios con conectividad a Internet limitada.
4. **Personalización:** Los registros privados se pueden personalizar para satisfacer los requisitos de la organización, incluida la autenticación, el control de acceso y la integración con sistemas existentes.

Algunas soluciones populares de registros privados de contenedores incluyen:

- [Amazon Elastic Container Registry \(ECR\) \(<https://aws.amazon.com/es/ecr/>\)](https://aws.amazon.com/es/ecr/): Un registro de contenedores de Docker completamente administrado proporcionado por Amazon Web Services (AWS).
- [Artifact Registry \(<https://cloud.google.com/artifact-registry?hl=es-419>\)](https://cloud.google.com/artifact-registry?hl=es-419): El servicio de registro de contenedores administrado de Google Cloud.
- [Azure Container Registry \(<https://azure.microsoft.com/en-us/products/container-registry>\)](https://azure.microsoft.com/en-us/products/container-registry): El servicio de registro de contenedores de Microsoft Azure.
- [Harbor \(<https://goharbor.io/>\)](https://goharbor.io/): Un registro de contenedores de código abierto que se puede implementar en tu propia infraestructura.

**Resumen:** Docker Hub y los registros de contenedores proporcionan plataformas para almacenar, compartir y distribuir imágenes de Docker. Docker Hub sirve como un registro público para la comunidad de Docker, mientras que las organizaciones pueden configurar registros privados para administrar sus propias imágenes de manera segura y eficiente.

## 6. Casos de uso en ciencia para Docker

---

- Docker se utiliza ampliamente para diversos propósitos, incluyendo desarrollo web, arquitectura de microservicios, integración y despliegue continuo, pruebas y entornos de desarrollo/pruebas.

Docker ha encontrado numerosos casos de uso en el campo de la ciencia debido a su capacidad para crear entornos aislados y reproducibles para ejecutar aplicaciones. Aquí hay algunos casos de uso donde Docker se aplica comúnmente en la investigación científica y la experimentación:

1. **Investigación Reproducible:** Los contenedores de Docker permiten a los científicos encapsular todo el entorno computacional, incluidas las dependencias y configuraciones del software. Esto asegura que los resultados de la investigación sean reproducibles, ya que otros pueden ejecutar el mismo entorno contenedorizado para validar los hallazgos.
2. **Análisis de Datos:** Docker se puede usar para crear entornos contenedorizados para tareas de análisis de datos, lo que permite a los investigadores empaquetar sus flujos de trabajo de procesamiento de datos, bibliotecas y scripts. Esto facilita compartir y colaborar en flujos de trabajo de análisis.
3. **Aprendizaje Automático e IA:** Docker se utiliza con frecuencia en la investigación de aprendizaje automático e inteligencia artificial. Los investigadores pueden empaquetar sus modelos, bibliotecas y dependencias en contenedores para garantizar resultados consistentes en diferentes entornos.
4. **Bioinformática:** Docker se utiliza en bioinformática para empaquetar flujos de trabajo de análisis complejos y herramientas para el procesamiento de datos biológicos. Esto permite a los investigadores compartir y reproducir análisis en un entorno controlado.
5. **Computación de Alto Rendimiento:** Los contenedores de Docker se pueden usar en clústeres de computación de alto rendimiento para crear entornos consistentes para simulaciones científicas y cálculos. Los contenedores son livianos y se pueden escalar fácilmente en nodos del clúster.
6. **Simulación Científica:** Los contenedores de Docker se utilizan para crear entornos aislados para ejecutar simulaciones, ya sea en física, química, ingeniería u otras disciplinas científicas. El entorno aislado asegura la consistencia en los resultados de la simulación.
7. **Colaboración:** Docker simplifica la colaboración entre los investigadores al garantizar que todos utilicen el mismo entorno. Los contenedores eliminan problemas de compatibilidad y la necesidad de configurar entornos desde cero.
8. **Laboratorios Virtuales:** Los investigadores pueden usar Docker para crear laboratorios virtuales para estudiantes y colegas, donde pueden acceder a entornos preconfigurados para experimentación y aprendizaje.
9. **Computación en la Nube y DevOps:** Los contenedores de Docker son adecuados para implementar aplicaciones científicas en la nube. Los contenedores se pueden mover fácilmente entre entornos locales y en la nube, lo que hace que la investigación basada en

la nube sea más eficiente.

10. **Estudios de Campo:** Los contenedores de Docker se pueden utilizar para configurar entornos consistentes y controlados para estudios de campo y experimentos remotos, asegurando que la recopilación y el análisis de datos sean estandarizados.
11. **Software Personalizado:** Los investigadores pueden empaquetar herramientas de software personalizadas, algoritmos y aplicaciones en contenedores de Docker, facilitando la distribución y el uso de sus herramientas por otros en la comunidad.
12. **Preservación de Software Heredado:** Docker se puede utilizar para encapsular y mantener software antiguo que puede ser necesario para replicar experimentos históricos o analizar conjuntos de datos antiguos.

En resumen, la capacidad de Docker para proporcionar entornos consistentes y aislados lo convierte en una herramienta valiosa en la investigación científica. Mejora la reproducibilidad, facilita la colaboración y simplifica la implementación de aplicaciones científicas en diversos entornos.

## Verificación de la Instalación de Docker:

- Abre una terminal o símbolo del sistema.
- Escribe **docker --version** para verificar si Docker está instalado correctamente.
- Escribe **docker info** para obtener información detallada sobre tu instalación de Docker.

In [23]: `!docker --version`

```
Docker version 24.0.2, build cb74dfcd85
```

## 3. Ejecutar tu Primer Contenedor Docker:

### Básico:

- Abre una terminal o símbolo del sistema.
- Escribe **docker run hello-world**.
- Docker descargará la imagen "hello-world" desde Docker Hub y la ejecutará en un contenedor.
- Si todo está configurado correctamente, deberías ver un mensaje que confirma que Docker está funcionando.

```
In [24]: !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.  
(arm64v8)
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/> (<https://hub.docker.com/>)

For more examples and ideas, visit:

<https://docs.docker.com/get-started/> (<https://docs.docker.com/get-started/>)

## Intermedio: Descargar la Imagen de Nginx

**Paso 1:** Descargar la Imagen de Nginx

- Escribe el siguiente comando para descargar la imagen de Nginx desde Docker Hub:

```
In [2]: !docker pull nginx
```

```
Using default tag: latest
latest: Pulling from library/nginx
```

```
Digest: sha256:104c7c5c54f2685f0f46f3be607ce60da7085da3eaa5ad22d3d9f01594295e
9c
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

**Paso 2:** Crear un Directorio y un Archivo HTML

Crea un directorio en tu sistema donde colocarás tu archivo index.html personalizado. Para este ejemplo, llamémoslo "my\_website":

```
In [11]: ls
```

```
1. Fundamentals.ipynb images/ index.html
```

```
In []: #!mkdir my_website
```

- Dentro del directorio "my\_website", crea un archivo index.html y añade algún contenido.  
Por ejemplo:

```
<html>
 <head>
 <title>Welcome to My Website</title>
 </head>
 <body>
 <h1>Hello, Docker!</h1>
 <p>This is my first Docker container.</p>
 </body>
</html>
```

```
In []: !cat index.html
```

```
<!DOCTYPE html>
<html>
 <head>
 <title>Welcome to My Website</title>
 </head>
 <body>
 <h1>Hello, Docker!</h1>
 <p>This is my first Docker container.</p>
 </body>
</html>
```

### Paso 3: Ejecutar el Contenedor de Nginx

- Ejecuta el siguiente comando para iniciar un contenedor de Nginx:

```
In [8]: !docker run -d -p 8080:80 -v .:/usr/share/nginx/html --name my_nginx_container
```

```
26f702e94b199ffcbbe3d49e7ed41d28d646020851cff91123dca5351bba04d33
```

Vamos a desglosar este comando:

- **docker run**: Este es el comando para crear y iniciar un contenedor a partir de una imagen.
- **-d**: Esta bandera ejecuta el contenedor en modo desacoplado, lo que significa que se ejecuta en segundo plano.
- **-p 80:80**: Esta bandera mapea el puerto 80 del host al puerto 80 dentro del contenedor, lo que te permite acceder al servidor web Nginx en el puerto 80 de tu localhost.
- **-v \$(pwd):/usr/share/nginx/html**: Esta bandera monta el directorio actual (my\_website) en tu máquina anfitriona al directorio **/usr/share/nginx/html** dentro del contenedor. De

esta manera, tu archivo personalizado ***index.html*** será servido por Nginx.

- **--name my\_nginx\_container:** Esta bandera asigna un nombre personalizado al contenedor, en este caso, "my\_nginx\_container".
- **nginx:** Este es el nombre de la imagen que deseas utilizar para crear el contenedor.

- **Paso 5:** Acceder al Servidor Web

Ahora que tu contenedor está en funcionamiento, puedes acceder al servidor web abriendo un navegador web e ingresando <http://localhost> (<http://localhost>) o <http://127.0.0.1> (<http://127.0.0.1>). Deberías ver el contenido de tu archivo index.html personalizado mostrado en el navegador.

- **Paso 6:** Detener y Eliminar el Contenedor (Opcional)

Para detener el contenedor en ejecución, utiliza el siguiente comando:

```
In [9]: !docker stop my_nginx_container
```

```
my_nginx_container
```

Para eliminar el contenedor, utiliza el siguiente comando:

```
In [10]: !docker rm my_nginx_container
```

```
my_nginx_container
```

**Note:** If you don't plan to use the container anymore, you can remove it to free up resources. However, keep in mind that removing a container also removes its data, so if you want to keep the data, you should consider using volumes to persist it.

¡Eso es todo! Has ejecutado exitosamente tu primer contenedor Docker.

Ahora puedes explorar más características de Docker, como crear imágenes personalizadas usando Dockerfiles, usar Docker Compose para gestionar aplicaciones de múltiples contenedores y mucho más. ¡Feliz contenerización!

## 4. Explorar Imágenes de Docker:

Explorar imágenes de Docker es una parte esencial de trabajar con Docker. Las imágenes de Docker son los componentes básicos de los contenedores y contienen todas las dependencias y configuraciones necesarias para ejecutar tu aplicación. Las imágenes de Docker son plantillas para los contenedores. Puedes usar imágenes existentes o crear las tuyas propias. A continuación, se detalla el proceso para explorar imágenes de Docker.

- Listar Imágenes de Docker Disponibles:

Para ver una lista de imágenes de Docker disponibles en tu sistema, abre una terminal o

In [ ]: !docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
jupyter_server	latest	bd735acb6f7b	4 days ago	1.35GB
<none>	<none>	e804ba8004c9	4 days ago	1.35GB
<none>	<none>	341a2135d531	4 days ago	1.35GB
<none>	<none>	073ce6b0f986	4 days ago	1.35GB
<none>	<none>	ade86720168c	4 days ago	1.35GB
jupyter/base-notebook	latest	eb7eb8b805af	5 days ago	1.06GB
daskdev/dask	latest	f0a7c6addbe7	7 days ago	1.32GB
my_custom_nginx	latest	89da1fb6dc9	2 weeks ago	187MB
nginx	latest	89da1fb6dc9	2 weeks ago	187MB
python	3.9	60d390c1959b	2 months ago	997MB
hfarias	latest	42eb93eb9464	3 months ago	179MB
miapp	latest	42eb93eb9464	3 months ago	179MB
test	latest	42eb93eb9464	3 months ago	179MB
docker/disk-usage-extension	0.2.7	94a994303197	3 months ago	2.81MB
ghcr.io/dask/dask-notebook	latest	59ffc0ebaa2b	3 months ago	1.58GB
ghcr.io/dask/dask	latest	c9a1114a74e0	3 months ago	1.3GB
docker/welcome-to-docker	latest	924b9d1abd68	5 months ago	13.1MB
hello-world	latest	feb5d9fea6a5	22 months ago	13.3kB

Este comando mostrará una lista de todas las imágenes de Docker que has descargado y guardado localmente en tu máquina.

- Descargar Imágenes de Docker desde Docker Hub:

Si deseas explorar nuevas imágenes de Docker, puedes descargarlas desde el repositorio oficial de Docker Hub u otros registros públicos o privados. Para descargar una imagen desde Docker Hub, utiliza el comando **docker pull** seguido del nombre de la imagen y opcionalmente la etiqueta (versión). Por ejemplo:

In [ ]: !docker pull nginx

```
Using default tag: latest
Error response from daemon: Get "https://registry-1.docker.io/v2/": dialing registry-1.docker.io:443 with direct connection: resolving host registry-1.docker.io: lookup registry-1.docker.io: no such host
```

Este comando descargará la última versión de la imagen de Nginx desde Docker Hub.

- Inspeccionar Imágenes de Docker:

Para obtener información más detallada sobre una imagen de Docker específica, puedes usar el comando **docker inspect** seguido del nombre de la imagen o el ID de la imagen. Por ejemplo:

```
In [12]: !docker inspect nginx
```

```
[
 {
 "Id": "39c39f9c75cb82586f2e4db2d79d2cbc73f93cc197e21f2f48f33b4d4a5
e423d",
 "Created": "2023-07-09T15:26:17.183542376Z",
 "Path": "/docker-entrypoint.sh",
 "Args": [
 "nginx",
 "-g",
 "daemon off;"
],
 "State": {
 "Status": "exited",
 "Running": false,
 "Paused": false,
 "Restarting": false,
 "OOMKilled": false,
 "Dead": false,
 "Pid": 0,
 "ExitCode": 0
 }
 }
]
```

Este comando proporcionará una salida en formato JSON con varios detalles sobre la imagen, como su configuración, capas, variables de entorno y más.

- Etiquetar Imágenes de Docker:

Puedes asignar una etiqueta personalizada a una imagen de Docker para facilitar su gestión o identificación posterior. Para etiquetar una imagen, utiliza el comando docker tag seguido de la imagen de origen y la nueva etiqueta. Por ejemplo:

```
In [14]: !docker tag nginx my_custom_nginx:latest
```

In [15]: !docker images

REPOSITORY		TAG	IMAGE ID	CRE
ATED	SIZE			
python		3.10-slim-buster	017c9ac972ef	5 d
ays ago	439MB			
influxdb		2.7.1	77e2b43c3ae1	5 d
ays ago	298MB			
dortiz/ainternet-c2		latest	0cdbdc7463af	5 d
ays ago	164MB			
miturno-ui-display-display		latest	1754a75257d1	7 d
ays ago	183MB			
redis		6.0.20	9d0c643353eb	7 d
ays ago	146MB			
influxdb		<none>	ca7eb2fe1267	7 d
ays ago	252MB			
my_custom_nginx		latest	ab73c7fd6723	7 d
ays ago	192MB			
nginx		latest	ab73c7fd6723	7 d
ays ago	192MB			
ashpai-controller-celery_beat		1.0	b1f55d938e65	2 w
eeks ago	480MB			
ashpai-controller		1.0	16df7c0a5a7f	2 w
eeks ago	480MB			
ashpai-controller-worker		1.0	f980e9195a64	2 w
eeks ago	480MB			
ashpai-controller-celery_flower		1.0	80e951bb5025	2 w
eeks ago	480MB			
nginx		1.25.1	ff78c7a65ec2	3 w
eeks ago	192MB			
herbarium-presentation		1.0	0babab7d9a272	4 w
eeks ago	876MB			
herbarium_postprocessing		1.0	764fc1eeb4c5	4 w
eeks ago	1.51GB			
herbarium_digitalizacion		1.0	2b249d1b53b8	4 w
eeks ago	963MB			
postgis/postgis		latest	0254c764b670	4 w
eeks ago	588MB			
dspace/dspace-postgres-pgcrypto		latest	e39415990147	6 w
eeks ago	428MB			
dspace/dspace-angular		dspace-7_x	f7c23b6e6505	6 w
eeks ago	2.79GB			
dspace/dspace-solr		dspace-7_x	b181d5ba5f78	6 w
eeks ago	505MB			
dspace/dspace		dspace-7_x-test	d62d994b5397	6 w
eeks ago	964MB			
miturno-analytics		1.0	62246cd129af	6 w
eeks ago	545MB			
web-nginx		latest	5e2d40b55900	6 w
eeks ago	22MB			
miturno-analitics-django-nginx		latest	924085a1281c	6 w
eeks ago	22MB			
miturno-cron		1.0	3e5c90e65eec	7 w
eeks ago	57.8MB			
telegraf		1.27.1	cf41418b9da8	7 w
eeks ago	383MB			
iluminaconciencia-website		1.0	7f24f84515c8	7 w
eeks ago	712MB			
eclipse-mosquitto		2.0.15	dbfb7e93690a	2 m

onths ago	14.5MB			
eclipse-mosquitto		latest	dbfb7e93690a	2 m
onths ago	14.5MB			
redis		7-alpine	edc5c8801dc6	2 m
onths ago	30.6MB			
postgres		14-alpine	cb57a91a2ecc	2 m
onths ago	241MB			
python		<none>	0e02089a8cc9	2 m
onths ago	111MB			
iegomez/mosquitto-go-auth		2.1.0-mosquitto_2.0.15	f054da1aeef5	3 m
onths ago	151MB			
hello-world		latest	b038788ddb22	3 m
onths ago	9.14kB			
postgres		13.3	0904b518ad08	2 y
ears ago	300MB			

Esto creará una nueva imagen con la etiqueta my\_custom\_nginx:latest basada en la imagen original de nginx.

- Eliminar Imágenes de Docker:

Si tienes imágenes que ya no necesitas, puedes eliminarlas para liberar espacio en el disco. Para eliminar una imagen de Docker, utiliza el comando docker rmi seguido del nombre de la imagen o el ID de la imagen. Por ejemplo:

In [ ]: !docker rmi nginx

**Nota** que no puedes eliminar una imagen si está siendo utilizada por un contenedor en ejecución. Necesitas detener y eliminar todos los contenedores que se basan en la imagen antes de poder eliminarla.

- Historial de Imágenes de Docker:

Puedes ver el historial de una imagen de Docker para conocer cómo se construyó y de qué capas está compuesta. Para ver el historial de la imagen, utiliza el comando docker history seguido del nombre de la imagen o el ID de la imagen. Por ejemplo:

```
In [19]: !docker history nginx
```

IMAGE SIZE	CREATED COMMENT	CREATED BY	S
ab73c7fd6723 B <missing> B <missing> B <missing> B <missing> 4.62kB <missing> 3.02kB <missing> 98B <missing> 2.12kB <missing> 1.62kB <missing> 4.9MB <missing> B <missing> B <missing> B <missing> B <missing> B <missing> B <missing> 7.1MB	7 days ago 7 days ago 8 days ago 8 days ago	/bin/sh -c #(nop) CMD ["nginx" "-g" "daemon... /bin/sh -c #(nop) STOPSIGAL SIGQUIT /bin/sh -c #(nop) EXPOSE 80 /bin/sh -c #(nop) ENTRYPOINT ["/docker-entr... /bin/sh -c #(nop) COPY file:9e3b2b63db9f8fc7... /bin/sh -c #(nop) COPY file:57846632accc8975... /bin/sh -c #(nop) COPY file:3b1b9915b7dd898a... /bin/sh -c #(nop) COPY file:caec368f5a54f70a... /bin/sh -c #(nop) COPY file:01e75c6dd0ce317d... /bin/sh -c set -x && groupadd --system -... /bin/sh -c #(nop) ENV PKG_RELEASE=1~bookworm /bin/sh -c #(nop) ENV NJS_VERSION=0.8.0 /bin/sh -c #(nop) ENV NGINX_VERSION=1.25.2 /bin/sh -c #(nop) LABEL maintainer=NGINX Do... /bin/sh -c #(nop) CMD ["bash"] /bin/sh -c #(nop) ADD file:bc58956fa3d1aff2e...	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Este comando mostrará una lista de capas y comandos utilizados para construir la imagen con la etiqueta <tag>.

**Resumen:** Explorar imágenes de Docker es crucial para comprender su contenido y gestionar de manera efectiva tus entornos de desarrollo y producción. Con este conocimiento, puedes descargar, inspeccionar, etiquetar y eliminar imágenes de Docker según tus necesidades específicas.

## 5. Gestionar Contenedores:

Gestionar contenedores de Docker implica tareas como iniciar, detener, inspeccionar y eliminar contenedores. A continuación, se detalla el proceso para gestionar contenedores de Docker:

- 5.1 Listar Contenedores en Ejecución:

Para ver una lista de todos los contenedores en ejecución en tu sistema, abre una terminal o símbolo del sistema y utiliza el siguiente comando:

In [ ]: !docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

Este comando mostrará una lista de contenedores en ejecución junto con sus detalles, como el ID del contenedor, la imagen, el comando, el estado, los puertos y los nombres.

- **5.2 Listar Todos los Contenedores (Ejecutándose y Detenidos):**

Para ver una lista de todos los contenedores, incluyendo los que están en ejecución y los detenidos, utiliza el siguiente comando:

In [21]: !docker ps -a

CONTAINER ID	IMAGE	COMMAND
CREATED	STATUS	PORTS
NAMES		
6684b2159dc2	hello-world	"/hello"
5 minutes ago	Exited (0) 5 minutes ago	
lucid_keldysh		
91bd264987a5	python:3.10-slim-buster	"/entrypoint
t.sh"	5 days ago	0.0.0.0:8000->8
000/tcp	django	
dcda6d2ced3d	telegraf:1.27.1	"/entrypoint
t.sh tele..."	5 days ago	Restarting (1) 23 seconds ago
ashpai-cloud-telegraf-1		
530037ca7422	iegomez/mosquitto-go-auth:2.1.0-mosquitto_2.0.15	"/usr/bin/t
ini -- /u..."	5 days ago	0.0.0.0:1883->1
883/tcp, 1884/tcp	ashpai-cloud-mosquitto-1	
48643eb3e258	nginx:1.25.1	"/docker-en
trypoint...."	5 days ago	0.0.0.0:80->80/
tcp	ashpai-cloud-nginx-1	
bf4201427431	redis:6.0.20	"docker-ent
rypoint.s..."	5 days ago	0.0.0.0:6379->6
379/tcp	ashpai-cloud-redis-1	
370591f4bee8	influxdb:2.7.1	"/docker-en
trypoint-..."	5 days ago	0.0.0.0:8086->8
086/tcp	ashpai-cloud-influxdb-1	
4b635927762d	postgres:13.3	"docker-ent
rypoint.s..."	5 days ago	0.0.0.0:5432->5
432/tcp	ashpai-cloud-postgresql-1	
e605d5b267c2	miturno-ui-display-display	"docker-ent
rypoint.s..."	7 days ago	Exited (0) 6 days ago
display		
913a587b94e9	web-nginx	"/docker-en
trypoint...."	7 days ago	Exited (0) 7 days ago
iluminaconciencia-nginx		
2f1de9ec2f4b	iluminaconciencia-website:1.0	"python3 ma
nage.py r..."	7 days ago	Exited (0) 7 days ago
iluminaconciencia-website		
fe69643648ad	ashpai-controller:1.0	"/entrypoint
t /start"	2 weeks ago	0.0.0.0:8001->8
000/tcp	ashpai-controller-web-1	
412be2691c4f	ashpai-controller-celery_beat:1.0	"/entrypoint
t /start-..."	2 weeks ago	Exited (0) 5 days ago
ashpai-controller-celery_beat-1		
12d61c8de868	ashpai-controller-celery_flower:1.0	"/entrypoint
t /start-..."	2 weeks ago	0.0.0.0:5557->5
555/tcp	ashpai-controller-flower-1	
d7b9a0bdb45a	ashpai-controller-worker:1.0	"/entrypoint
t /start-..."	2 weeks ago	Exited (0) 5 days ago
ashpai-controller-celery_worker-1		
881e3920a4ff	postgres:14-alpine	"docker-ent
rypoint.s..."	2 weeks ago	0.0.0.0:5431->5
432/tcp	ashpai-controller-db-1	
cd049a66ab0e	eclipse-mosquitto	"/docker-en
trypoint...."	2 weeks ago	0.0.0.0:9001->9
001/tcp, 0.0.0.0:1882->1883/tcp	mosquitto	
6796cc2955d7	redis:7-alpine	"docker-ent
rypoint.s..."	2 weeks ago	6379/tcp
ashpai-controller-redis-1		

```
83fe16603dbd herbarium-presentation:1.0 "python3 ma
nage.py r..." 4 weeks ago Exited (2) 4 weeks ago
herbarium-presentation
10063f655222 herbarium_digitalizacion:1.0 "python3 ma
nage.py r..." 4 weeks ago Exited (0) 4 weeks ago
herbarium_digitalization
7e96cac51f12 postgis/postgis:latest "docker-ent
rypoint.s..." 4 weeks ago Exited (0) 5 days ago
postgres-herbarium
a615af71181f herbarium_postprocessing:1.0 "python ./v
ouchers_p..." 4 weeks ago Created
herbarium_postprocessing
4ff3a573cb86 dspace/dspace-solr:dspace-7_x "/bin/bash
-c 'init-..." 6 weeks ago Exited (143) 6 weeks ago
dspaceSolr
9ab3083edcd9 dspace/dspace:dspace-7_x-test "/bin/bash
-c 'while..." 6 weeks ago Exited (143) 6 weeks ago
dspace
113f045daaa6 dspace/dspace-postgres-pgcrypto "docker-ent
rypoint.s..." 6 weeks ago Exited (0) 6 weeks ago
dspaceDb
399a37e1b56f dspace/dspace-angular:dspace-7_x "docker-ent
rypoint.s..." 6 weeks ago Exited (1) 6 weeks ago
dspace-angular
39c39f9c75cb miturno-analitycs-django-nginx "/docker-en
trypoint...." 6 weeks ago Exited (0) 12 days ago
nginx
068d79bd7444 miturno-analytics:1.0 "gunicorn -
-bind 0.0..." 6 weeks ago Exited (0) 12 days ago
miturno-analytics
c1725228a314 miturno-cron:1.0 "crond -f"
6 weeks ago Exited (137) 12 days ago
miturno-cron
```

Esto mostrará todos los contenedores junto con sus detalles.

- **5.3 Iniciar un Contenedor Detenido:**

Para iniciar un contenedor detenido, necesitas conocer su ID de contenedor o su nombre. Utiliza el siguiente comando para iniciar el contenedor:

```
In []: !docker start <container_id_or_name>
```

- **5.4 Detener un Contenedor en Ejecución:**

Para detener un contenedor en ejecución, también necesitas conocer su ID de contenedor o su nombre. Utiliza el siguiente comando para detener el contenedor de manera segura:

```
In []: !docker stop <container_id_or_name>
```

Si deseas detener un contenedor de manera forzada, puedes utilizar el comando **docker kill** en su lugar.

- **5.5 Inspeccionar Detalles del Contenedor:**

Para obtener información más detallada acerca de un contenedor específico, utiliza el comando docker inspect seguido del ID o el nombre del contenedor. Por ejemplo:

In [22]: !docker inspect miturno-cron:1.0

```
[
 {
 "Id": "sha256:3e5c90e65eec099201440a9f64f844b83821b96eeefda839e9e6
38228d289589",
 "RepoTags": [
 "miturno-cron:1.0"
],
 "RepoDigests": [],
 "Parent": "",
 "Comment": "buildkit.dockerfile.v0",
 "Created": "2023-07-05T21:58:37.053612007Z",
 "Container": "",
 "ContainerConfig": {
 "Hostname": "",
 "Domainname": "",
 "User": "",
 "AttachStdin": false,
 "AttachStdout": false,
 "AttachStderr": false,
 "Env": []
 }
 }
]
```

- **5.6 Inspeccionar Detalles del Contenedor:**

Para obtener información más detallada acerca de un contenedor específico, utiliza el comando docker inspect seguido del ID o el nombre del contenedor. Por ejemplo:

In [ ]: !docker inspect my\_container

Esto proporcionará una salida en formato JSON con varios detalles sobre el contenedor, como su configuración, ajustes de red, variables de entorno y más.

- **5.7 Eliminar un Contenedor:**

Para eliminar un contenedor, primero debe estar detenido. Si aún está en ejecución, deberás detenerlo primero utilizando el comando docker stop (como se menciona en el paso 5.4). Una vez que el contenedor esté detenido, utiliza el siguiente comando para eliminarlo:

In [ ]: !docker rm <container\_id\_or\_name>

**Nota:** El contenedor se eliminará **permanentemente** y sus recursos se liberarán.

- **5.8 Eliminar Todos los Contenedores Detenidos:**

Si deseas eliminar todos los contenedores detenidos de una vez, puedes utilizar el siguiente

```
In []: !docker container prune
```

**Nota:** Esto eliminará todos los contenedores detenidos, liberando espacio en disco.

- **5.9 Gestionar los Registros del Contenedor:**

Para ver los registros de un contenedor específico, utiliza el siguiente comando:

```
In []: !docker logs <container_id_or_name>
```

**Resumen:** Estos son algunos de los comandos básicos para gestionar contenedores. Docker ofrece muchas más opciones y funcionalidades para administrar contenedores, incluyendo pausar, cambiar nombres y adjuntarse a contenedores, entre otros. Para explorar características de gestión de contenedores más avanzadas, puedes consultar la documentación oficial de Docker (<https://docs.docker.com/> (<https://docs.docker.com/>)).

```
In []:
```