Docker Básico

Arturo Silvelo

Try New Roads

Comandos básicos

- Comando básico para invocar al cliente de Docker:

```
docker [OPTIONS] COMMAND
```

- **OPTIONS**: Opciones que modifican el comportamiento del comando, como configuraciones o modos específicos.
- **COMMAND**: La acción que deseas que Docker ejecute, como `run`, `pull`, `build`, entre otros.

- Ejemplos comunes:
  - `docker run -d nginx`: Ejecuta un contenedor en segundo plano con la imagen `nginx`.
  - `docker ps -a`: Lista todos los contenedores, incluyendo los que están detenidos.
- **Nota**: Para ver una lista completa de comandos y opciones, puedes ejecutar:

```
docker --help
docker COMMAND --help
```

Descargar imágenes

```
docker [OPTIONS] pull IMAGE TAG
```

En Docker, las imágenes suelen seguir el esquema de **versionado semántico (SemVer)** en su etiqueta (`<TAG>`), el cual se estructura en el formato `MAJOR.MINOR.PATCH`:

- **MAJOR**: Cambios incompatibles o grandes, como una nueva versión con cambios significativos.
- **MINOR**: Nuevas funcionalidades que mantienen compatibilidad con versiones anteriores.
- **PATCH**: Correcciones de errores o pequeñas mejoras sin cambios en la funcionalidad.

- Ejemplo: `docker pull nginx:1.18.0`
  - Aquí, `1` es la versión principal, `18` es la versión menor y `0` es una corrección menor.
  - Esto permite a los desarrolladores seleccionar versiones específicas o actualizar a nuevas sin romper compatibilidad.

Gestionar imágenes:

```
docker image COMMAND
```

Los comandos de gestión de imágenes permiten realizar varias operaciones útiles sobre las imágenes de Docker:

- **ls**: Lista todas las imágenes en el host.
- **pull <IMAGE>**: Descarga una imagen de un registro.
- **rm <IMAGE>**: Elimina una imagen del host.

- Ejemplo:
  - `docker image ls` - Muestra todas las imágenes disponibles.

Mostrar los contenedores

```
docker ps OPTIONS
```

Este comando permite visualizar los contenedores que están actualmente en ejecución, así como aquellos que han sido detenidos:

- **-a**: Muestra todos los contenedores, incluidos los detenidos.
- **-q**: Muestra solo los IDs de los contenedores.

- Ejemplo:
  - `docker ps -a` - Muestra todos los contenedores, tanto en ejecución como detenidos.

Crear un contenedor

```
docker run OPTIONS IMAGE COMMANDS
```

Este comando se utiliza para crear y ejecutar un nuevo contenedor a partir de una imagen especificada:

- **-d**: Ejecuta el contenedor en segundo plano.
- **-p <HOST_PORT>:<CONTAINER_PORT>**: Publica un puerto del contenedor en el host.
- **--name <NAME>**: Asigna un nombre al contenedor.

- Ejemplo:
  - `docker run -d -p 8080:80 nginx` - Crea y ejecuta un contenedor de Nginx en segundo plano.

Ejecutar dentro de un contenedor

```
docker exec [OPTIONS] CONTAINER COMMAND
```

Permite ejecutar un comando específico dentro de un contenedor en ejecución:

- **-it**: Permite la interacción con el contenedor.
- Ejemplo:
  - `docker exec -it my_container /bin/bash` - Inicia una sesión de shell dentro del contenedor.

Parar un contenedor

```
docker stop OPTIONS CONTAINER
```

Este comando detiene un contenedor en ejecución de forma controlada:

- **-t <SECONDS>**: Espera un número específico de segundos antes de forzar la detención.

- Ejemplo:
  - `docker stop my_container` - Detiene el contenedor especificado.

Gestionar redes

```
docker network COMMAND
```

Permite realizar operaciones sobre las redes de Docker:

- **ls**: Lista todas las redes disponibles.
- **create <NETWORK>**: Crea una nueva red.
- **rm <NETWORK>**: Elimina una red especificada.

- Ejemplo:
  - `docker network ls` - Muestra todas las redes disponibles en el host.

Gestionar volúmenes

```
docker volume COMMAND
```

Este comando permite gestionar volúmenes en Docker, que son utilizados para persistir datos:

- **ls**: Lista todos los volúmenes existentes.
- **create <VOLUME>**: Crea un nuevo volumen.
- **rm <VOLUME>**: Elimina un volumen especificado.

- Ejemplo:
  - `docker volume ls` - Muestra todos los volúmenes disponibles.

Copiar contenido entre contenedor y host

```
docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH
```

Permite transferir archivos entre un contenedor y el sistema host:

- Esta operación es útil para recuperar logs, datos, o para transferir archivos hacia el contenedor.

- Ejemplo:
  - `docker cp my_container:/data/file.txt ./file.txt` - Copia un archivo del contenedor al host.

Ver los registros de un contenedor

```
docker logs [OPTIONS] CONTAINER
```

Este comando muestra los registros generados por un contenedor:

- **-f**: Permite seguir los registros en tiempo real.
- **--tail <N>**: Muestra las últimas N líneas de los registros.

- Ejemplo:
  - `docker logs -f my_container` - Sigue los registros del contenedor en tiempo real.

Eliminar un contenedor

```
docker rm OPTIONS CONTAINER
```

Este comando se utiliza para eliminar un contenedor que ha sido detenido:

- **-f**: Fuerza la eliminación de un contenedor en ejecución.

- Ejemplo:
  - `docker rm my_container` - Elimina el contenedor especificado.

Contenedores en Segundo Plano

Los contenedores en segundo plano (detached mode) se ejecutan en *background*, lo que permite que el terminal permanezca libre para otras tareas. Esta funcionalidad es especialmente útil en entornos de servidor, donde se pueden ejecutar múltiples servicios sin necesidad de mantener el terminal ocupado.

```
docker run -d --name=nginx-test nginx
```

Ventajas del modo en segundo plano:

- Mantiene el terminal libre para ejecutar otros comandos.
- Permite gestionar múltiples contenedores simultáneamente.
- Ideal para aplicaciones de larga duración o en producción.

Modo Interactivo

Para acceder a un contenedor que se está ejecutando en segundo plano, es necesario utilizar el comando `docker exec` para lanzar una terminal interactiva.

```
docker exec -it nginx-test /bin/bash
```

Este comando nos permitirá acceder al contenedor y realizar diversas tareas, tales como:

- **Instalación de software:** Instalar aplicaciones o herramientas necesarias dentro del contenedor.
- **Pruebas y depuración:** Ejecutar comandos para comprobar el estado del contenedor y solucionar problemas.
- **Exploración del sistema de archivos:** Navegar por la estructura de directorios y archivos del contenedor.

Puertos

Los puertos son esenciales para conectar y comunicar un contenedor con el mundo exterior, permitiendo el acceso a los servicios que se ejecutan dentro de él, como servidores web, bases de datos o APIs. Al exponer un puerto, Docker facilita la interacción entre aplicaciones externas, otros contenedores y el host.

```
docker run -d -p HOST CONTAINER IMAGE
docker run -d -p 8080:80 --name nginx-port nginx
```

> Asegúrate de que el puerto en el host no esté en uso por otra aplicación.
> Puedes exponer múltiples puertos usando varias opciones `-p` en un solo comando.

Formato del comando:

- `<HOST>:<CONTAINER>` asigna puertos entre el host y el contenedor.
- En el ejemplo `8080:80`, el puerto `8080` del host se mapea al puerto `80` del contenedor, lo que permite acceder al servicio del contenedor a través del puerto `8080` en el host.

Logs

Los registros (logs) permiten monitorear y depurar el comportamiento de las aplicaciones dentro de los contenedores, facilitando la comprensión de su ejecución y la detección de problemas o excepciones.

Para ver los logs de un contenedor en ejecución, se utiliza el siguiente comando:

```
docker logs [OPTIONS] CONTAINER
```

Ejemplo: Para obtener los logs de un contenedor llamado `nginx-port`:

```
docker logs nginx-port
```

Opciones útiles:

- `-f`: Muestra los logs en tiempo real (follow).
- `--tail`: Muestra solo las últimas líneas de los logs.

Variables de Entorno

Las variables de entorno son valores configurables que se pueden definir para un contenedor, proporcionando un medio flexible para configurar aplicaciones sin modificar su código fuente. Esto permite que las aplicaciones se comporten de manera diferente según el entorno en el que se ejecutan.

> Es recomendable no incluir información sensible, como contraseñas, directamente en los comandos. Considera usar archivos de configuración o herramientas de gestión de secretos.
> Puedes acceder a las variables de entorno dentro del contenedor a través del código de la aplicación, facilitando la personalización.

Ejemplo 1: Configurar credenciales para una base de datos MongoDB:

```
docker run -P -e MONGO_INITDB_ROOT_USERNAME=root \
           -e MONGO_INITDB_ROOT_PASSWORD=toor \
           --name mongo-dev -d mongo
```

Contenedores Sin Servicios

Los contenedores sin servicios en Docker son aquellos que no ejecutan un servicio de larga duración (como un servidor web o una base de datos). En su lugar, realizan tareas puntuales y luego se detienen. Estos contenedores son ideales para ejecutar tareas únicas, scripts o comandos que se completan rápidamente.

Ejemplo 1: Ejecutar un comando en un contenedor Ubuntu:

```
docker run --rm ubuntu echo "Hola desde Docker"
```

Ejemplo 2: Ejecutar un script de Python:

```
docker run --rm python:3.8 python -c "print('Hello from Docker!')"
```

Ejemplo 3: Crear un archivo comprimido utilizando tar:

```
docker run --rm -v $(pwd):/data ubuntu tar -czvf /data/mi_archivo.tar.gz /data
```

Nota: Usar la opción `--rm` permite eliminar automáticamente el contenedor una vez que ha terminado de ejecutar el comando, lo que ayuda a mantener el entorno limpio.