

Ecosistema Docker

Administración de Sistemas

Unai Lopez Novoa
unai.lopez@ehu.eus

eman ta zabal zazu



Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

Contenido

1. Persistencia en Docker
2. Docker Compose
3. Registros Docker
4. Docker Hub



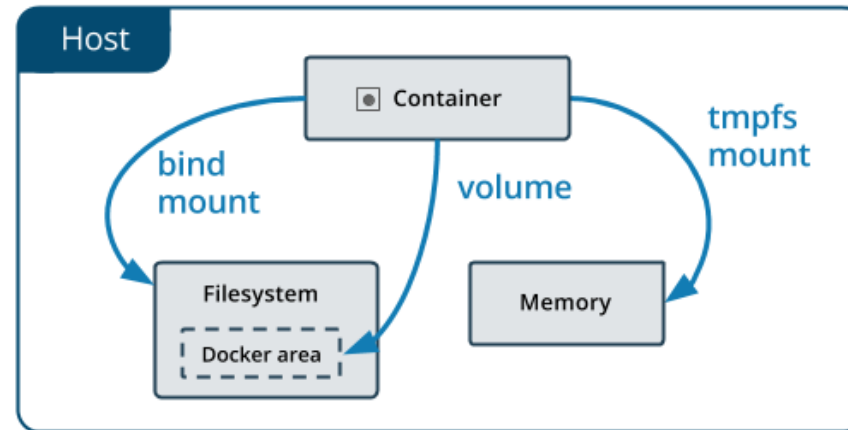
Persistencia en Docker

- En múltiples ocasiones necesitaremos recuperar datos generados/almacenados dentro de un contenedor.
 - P.e. si aloja una BBDD.
- Docker proporciona diferentes formas para intercambiar ficheros con un contenedor en marcha.



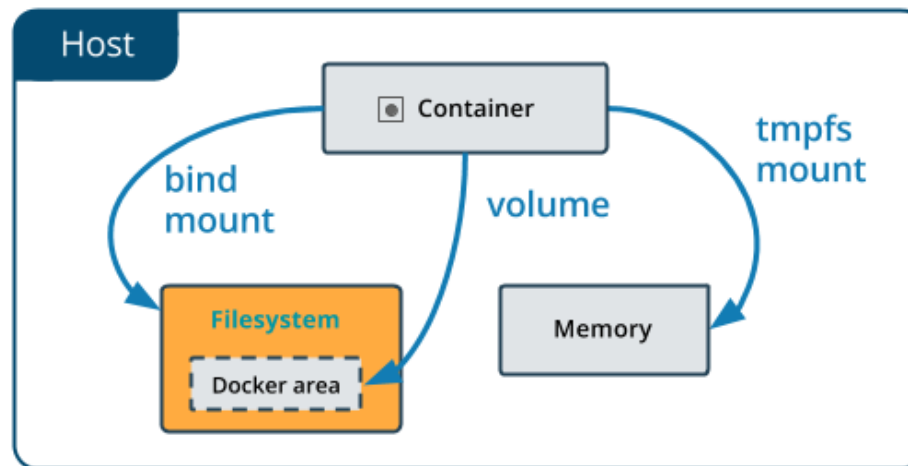
Persistencia en Docker

- Hay 3 formas de acceder al almacenamiento de la máquina anfitrión desde un contenedor:
 - *bind mounts*: Montar una carpeta del anfitrión en el contenedor
 - Volúmenes: Reservar un espacio persistente con Docker
 - *tmpfs*: Utilizar la memoria del anfitrión



bind mounts

- Montaje de una carpeta del anfitrión en el contenedor.
 - Se referencia a la carpeta existente en el anfitrión
- Técnica existente desde los comienzos de Docker
- Puede suponer un riesgo para el anfitrión
 - El contenedor podría modificar el sistema de ficheros anfitrión.



bind mounts

- La forma más simple de definir un *bind mount* es como un parámetro al ejecutar el contenedor.
- Parámetro `-v` del comando “docker run”

```
docker run -v <dir-anfitrión>:<dir-contenedor> <imagen>
```

- Ejemplo:

```
docker run -v $(pwd) :/app mi-linux
```

- Mapea el directorio actual (`$PWD`) en el anfitrión con el directorio `/app` dentro del contenedor con la imagen “milmagen”



bind mounts

- Alternativamente, se puede utilizar el comando `--mount`
 - Es más verboso que `-v`
 - Implica definir (al menos) las variables *source* y *target*.

- Uso:

```
docker run <parámetros-run> \  
  --mount type=bind,source=<directorio-anfitrión>, \  
  target=<directorio-contenedor> \  
  <imagen-docker>
```

- Ejemplo:

```
docker run -it --name devtest \  
  --mount type=bind,source="$(pwd)",target=/app \  
  busybox sh
```



bind mounts

- Única diferencia entre `-v` y `--mount`:
 - Al montar un directorio que no existe en el anfitrión:
 - `-v` crea el directorio en el anfitrión.
 - `--mount` no crea el directorio y muestra un error.
- En ambos casos, se puede inspeccionar un montaje:

```
docker inspect <nombre-contenedor>
```

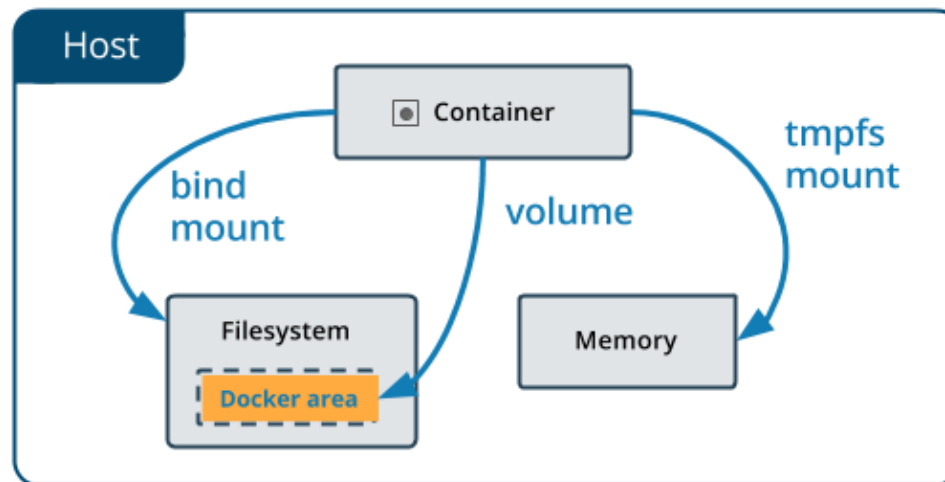
- Buscar la sección “Mounts” de la salida:

```
...  
"Mounts": [  
  { "Type": "bind",  
    "Source": "/tmp/source/target",  
    "Destination": "/app",  
    ...  
  }  
]
```



Volúmenes

- Son espacios de almacenamiento en el anfitrión gestionados por Docker.
- Pueden ser de 2 tipos:
 - Nombrados (se les establece un nombre)
 - Anónimos (sin nombre)



Volúmenes

- Se almacenan en el anfitrión.
 - En Linux, por defecto en `/var/lib/docker/volumes/`
 - El anfitrión no debería modificar los directorios de los volúmenes.
 - Sólo leerlos para, p.e., hacer backups.
- Su uso es preferible a los bind mounts:
 - Son más seguros
 - El contenedor no puede acceder al sistemas de ficheros anfitrión.
 - Toda la gestión se hace en el entorno Docker
 - Y utilizando las herramientas de Docker
 - Si el tamaño del volumen crece, no influye en el tamaño del contenedor



Volúmenes

- Gestionando volúmenes
 - Crear un volumen nombrado:

```
docker volume create <nombre-volumen>
```

- Listar volúmenes:

```
docker volume ls
```

- Inspeccionar un volumen:

```
docker volume inspect <nombre-volumen>
```

- Borrar un volumen:

```
docker volume rm <nombre-volumen>
```



Volúmenes

- Igual que los *bind mounts*, se configuran con el parámetro `-v` o `--mount`.

- Uso con `-v`:

```
docker run \  
-v <nombre-volumen>:<directorio-anfitrión> <imagen-docker>
```

- Uso con `--mount`:

```
docker run \  
--mount source=<nombre-volumen>,target=<directorio-anfitrión> \  
<imagen-docker>
```



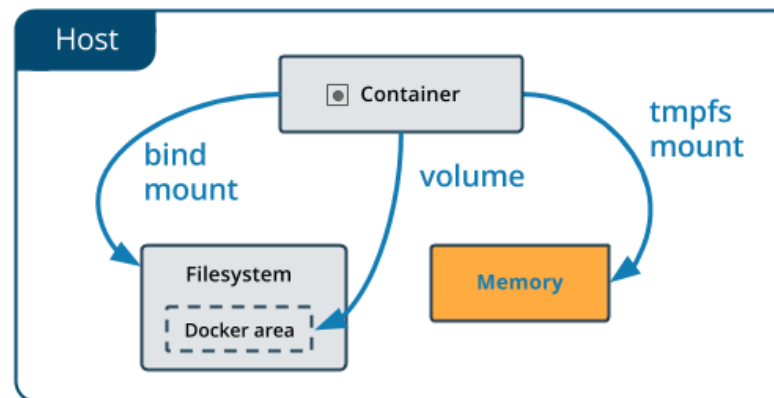
Persistencia en Docker

- ¿Cuándo utilizar COPY en el Dockerfile?
 - Dependencias
 - Ficheros de configuración / necesarios para la instalación
 - Ficheros de uso habitual
- ¿Cuándo utilizar *bind mounts*/volúmenes?
 - Versiones actualizadas de objetos / configuración
 - Respaldo de los datos capturados en el contenedor
- Puede ser útil dejar comentado en el Dockerfile los *bind mounts* / volúmenes que se planean mapear.
 - O alternatively, formas de copiar los datos con COPY.



tmpfs

- Utilizar un espacio de memoria como almacenamiento.
 - No permite compartir datos entre contenedor y anfitrión.
 - No es persistente.



- Se utiliza el parámetro `--tmpfs` o `--mount`.
 - Más información: <https://docs.docker.com/storage/tmpfs/>



Ejercicio 1

- Crear un volumen Docker llamado "datos-ej-1"
- Lanzar un contenedor de "alpine" con el comando "sh":
 - Montar el volumen "datos-ej-1" en /datos dentro del contenedor.
- Desde la Shell del contenedor "alpine":
 - Crear 2 ficheros con texto aleatorio en /datos.
 - Cerrar sesión y asegurarse de que el contenedor se ha parado.
- Lanzar un contenedor de "busybox" con el comando "sh":
 - Montar el volumen "datos-ej-1" en /datos dentro del contenedor.
- Desde la Shell del contenedor "busybox":
 - Verificar que los 2 ficheros con texto están dentro de /datos.
- Desde la Shell del anfitrión:
 - Encontrar el directorio donde se almacenan los 2 ficheros.

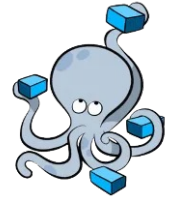


Docker Compose

- Hasta ahora hemos gestionado la ejecución de diferentes contenedores de forma manual :
 - Ejecución usando “docker run”, ...
 - Redirección de puertos con el parámetro ‘-p’.
- En entornos locales, podemos usar Compose para especificar todas las configuraciones de ejecución en un mismo fichero.
 - En lugar de configurar cada contenedor por separado.



Docker Compose



- Herramienta para facilitar la gestión de múltiples contenedores relacionados.
- Permite automatizar los parámetros necesarios para lanzar contenedores.
 - Que utilizaríamos con “docker run”.
- Se utiliza mediante línea de comandos.
 - Comprobar la versión Compose instalada:

```
docker compose version
```



Compose: Configuración

- Para desplegar contenedores con Compose:
 - 1) Describir imágenes con Dockerfiles
 - 2) Crear un archivo docker-compose.yml
 - 3) Utilizar los comandos docker-compose
- El fichero docker-compose.yml contiene:
 - Listado de los contenedores a crear
 - Por cada contenedor:
 - Imagen o Dockerfile a utilizar
 - Configuración (p.e. mapeo de puertos)



Compose: Configuración

- Ejemplo de docker-compose.yml:

Indica el listado de contenedores

```
services:
  servidor-redis:
    image: redis
  servidor-web:
    build: .
    ports:
      - 6060:1080
```

6060 en el anfitrión,
1080 en el contenedor

- Inicia:

- 1 contenedor utilizando la imagen "redis" de Docker Hub.
- 1 contenedor con el Dockerfile ubicado en la misma carpeta.
 - Redirige el puerto 1080 en el contenedor al 6060 del anfitrión.



Compose: Configuración

- Se pueden indicar rutas concretas a los directorios y Dockerfiles de los contenedores
 - Dentro de build, utilizar:
 - context: directorio con los archivos necesarios
 - dockerfile: nombre (o ruta) al fichero Dockerfile
- Ejemplo:

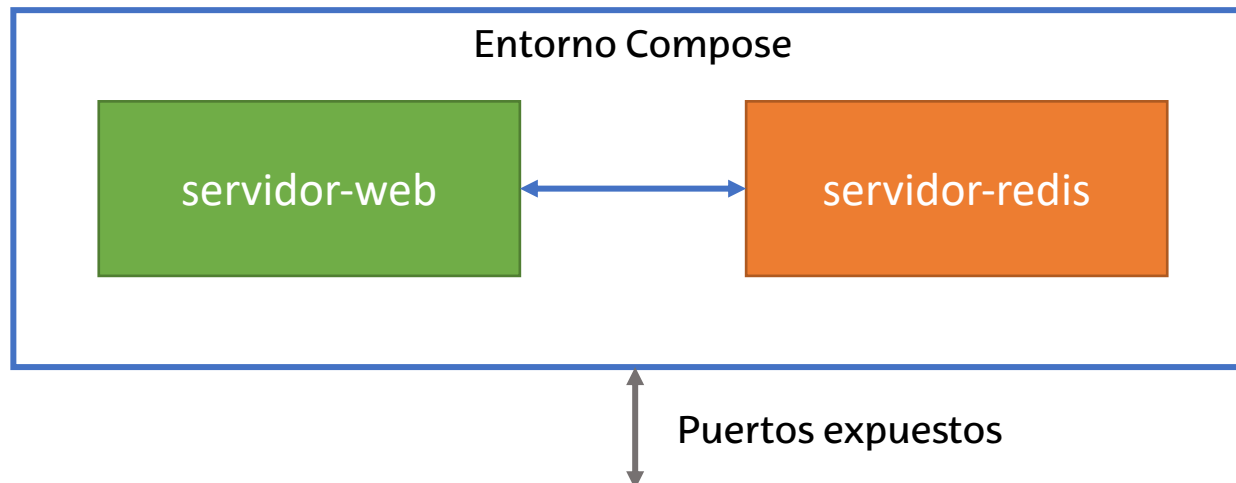
```
services:  
  servidor-web:  
    build:  
      context: .  
      dockerfile: Dockerfile.dev
```

Utilizar el directorio actual y "Dockerfile.dev" para crear el contenedor



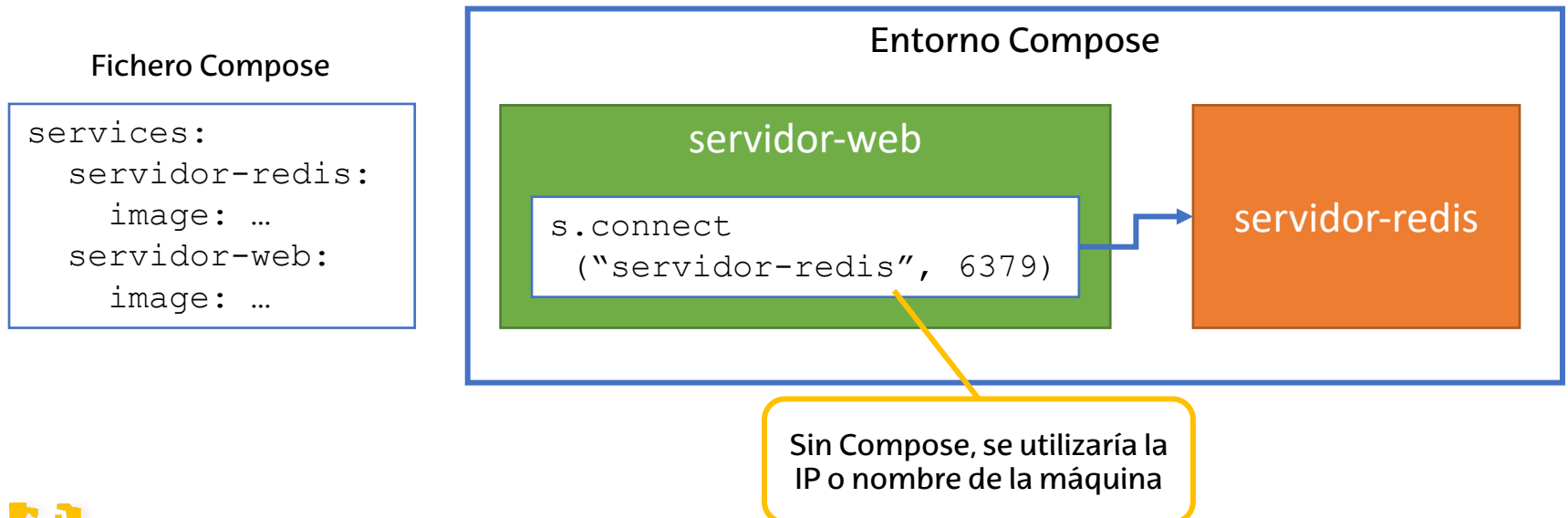
Compose: Configuración

- Configuración de red:
 - No hace falta especificar un mapeo de puertos entre contenedores
 - Sólo definir puertos a exponer fuera de los contenedores.
 - Al definir los contenedores dentro de “services”, Compose crea una red entre ellos con plena visibilidad de puertos.



Compose: Configuración

- Configuración de red:
 - Para alcanzar un contenedor desde otro se utiliza el nombre definido en el fichero .yaml.
- Ejemplo:
 - El código en "servidor-web" se conecta al puerto 6379 de "servidor-redis".



Compose: Configuración

- Es posible sobre-escribir el comando de arranque de un contenedor.
 - P.e. si tenemos más de una funcionalidad en un mismo contenedor.
- Utilizar "command" dentro del contenedor objetivo.
 - Ejemplo:

```
...
servidor-web-tests:
  build:
    context: .
    dockerfile: Dockerfile.dev
  command: ["npm", "run", "test"]
```



Compose: Configuración

- Añadiendo Volúmenes

- Añadir una línea "volumes:" dentro del contenedor objetivo
 - <nombre-volumen>:<directorio-en-el-contenedor>
- Añadir una sección "volumes" con el nombre del volumen
 - Añadir "external:true" si el volumen ya está creado en el sistema
- Ejemplo:

```
services:  
  servidor-web:  
    build: .  
    volumes:  
      - mi-vol:/data  
volumes:  
  mi-vol:  
    external: true
```



Compose: Políticas de reinicio

- Se pueden indicar qué debe hacerse con un contenedor al finalizar su ejecución.
- Se define mediante políticas de reinicio:

Política	Comportamiento
no	No reiniciar el contenedor (política por defecto).
on-failure	Reiniciar si el contenedor finaliza con un error. Se interpreta error cuando el código de finalización del proceso es superior a 0.
always	Reiniciar siempre. Si se ha parado manualmente, se reinicia cuando se reinicie el servicio Docker.
unless-stopped	Reiniciar siempre. Si se ha parado manualmente, no se reinicia cuando se reinicie el servicio Docker.



Compose: Políticas de reinicio

- Añadir la política deseada al fichero .yaml
 - P.e. configurar "servidor-web" para que se reinicie siempre:

```
...
services:
  servidor-redis:
    ...
  servidor-web:
    restart: always
    build: .
    ports:
      - "6060:1080"
```



Compose: Comandos

- Iniciar el grupo de contenedores:

```
docker compose up
```

- Equivale a "docker run <imagen>"

- Crear e iniciar grupo de contenedores

```
docker compose up --build
```

- Equivale a "docker build ." + "docker run <imagen>"

- Para ambas versiones del comando:

- Lanzar desde la carpeta con el fichero .yaml
- Los colores identifican la salida de cada contenedor.



Compose: Comandos

- Iniciar grupo de contenedores en 2º plano

```
docker compose up -d
```

- Equivale a "docker run -d"

- Parar grupo contenedores:

```
docker compose down
```

- Equivale a "docker stop"

- *Ambos comandos deben lanzarse desde la carpeta que contiene el fichero .yml.*



Compose: Comandos

- Comprobar el estado de los contenedores

```
docker compose ps
```

- Equivale a “docker ps”
 - Se tiene que lanzar desde la carpeta que contiene el fichero .yaml
-
- Más información:
 - Documentación completa de Docker Compose:
<https://docs.docker.com/compose/>
 - Sección con consejos para utilizar Compose en un entorno de producción:
<https://docs.docker.com/compose/production/>



Ejercicio 2

- Crear 1 imagen Docker:
 - Imagen base: Ubuntu
 - Dependencia: python3
 - Copiar fichero desde anfitrión: index.html
 - Código de index.html:

```
<!DOCTYPE html>
<html>
<head> <title>Hola Docker</title> </head>
<body> <h1>Hola!</h1> <p>Soy una Web en Compose.</p> </body>
</html>
```

- Comando de arranque: `python3 -m http.server 1080`
 - Se debe servir el fichero index.html
- *Continúa en la siguiente diapositiva.*



Ejercicio 2

- Crear un entorno Docker Compose con:
 - La imagen recién definida, nombre "servidor-web".
 - Mapear puerto 1080 del contenedor al puerto 80 del anfitrión.
 - La imagen "redis", nombre "servidor-bbdd".
- Verificar que ambos contenedores se ejecutan correctamente.
 - Acceder al contenedor "redis" y abrir "redis-cli".
 - Abrir la URL <http://<VUESTRA-IP/>> en un navegador.
 - Si no funciona, revisar la configuración del Firewall en Google Cloud.



Registro Docker

- Hasta ahora, hemos obtenido las imágenes Docker de terceros desde DockerHub.

Pero...

- En algunos entornos, nos puede interesar tener un sistema local de gestión de imágenes.
 - P.e., por motivos de coste o de seguridad.



Registro Docker

- Es una aplicación que permite almacenar y distribuir imágenes Docker.
- Desde el punto de vista *sysadmin*, permite controlar:
 - El almacenamiento.
 - La distribución de imágenes.
 - La integración del almacenamiento en el *pipeline* interno de desarrollo software.
- Código open source
 - Licencia Apache



Registro Docker

- El registro Docker es una instancia de la imagen *registry*
- Funciona como un contenedor Docker más
- Comandos básicos:
 - Instanciar un registro local Docker:

```
docker run -d -p 5000:5000 --name registry registry:2
```

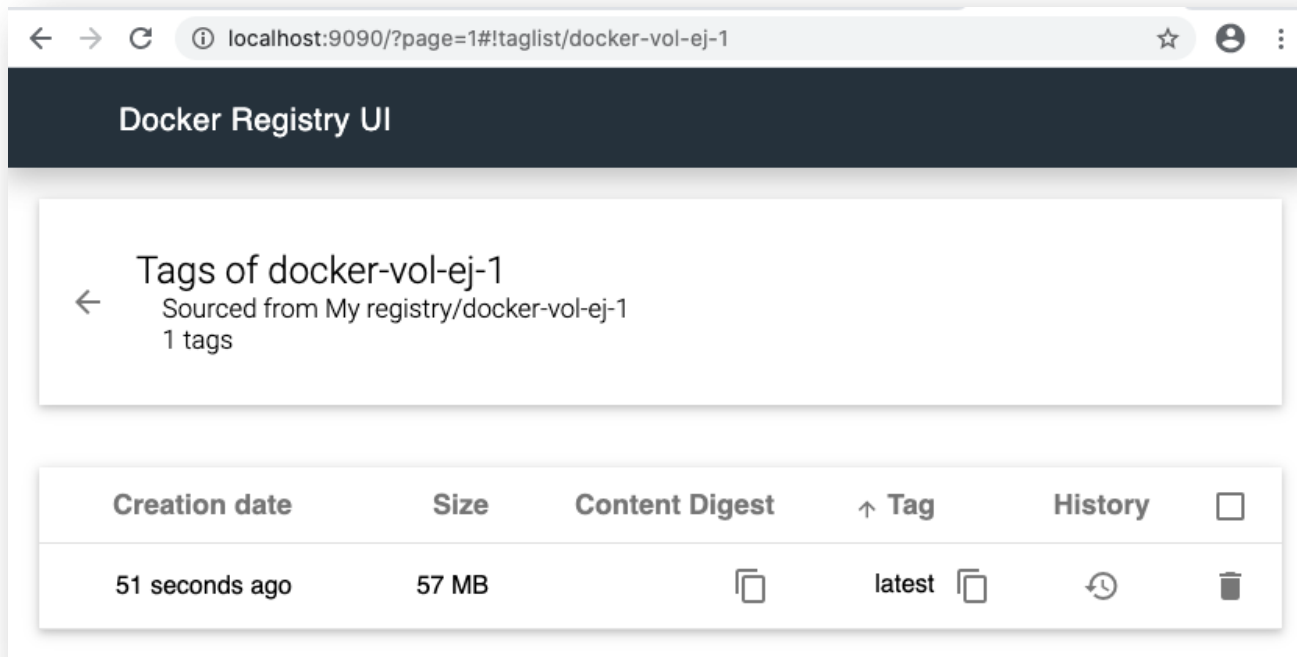
- Parar un registro local Docker:

```
docker container stop registry
```



Registro Docker

- Se pueden instalar interfaces Web para el registro
 - Ejemplo: Docker Registry UI de Joxit
 - Repositorio: <https://github.com/Joxit/docker-registry-ui>



Registro Docker

- Se pueden instalar interfaces Web para el registro
 - Ejemplo: Docker Registry UI de Joxit
 - Despliegue con Docker Compose
 - Fichero completo: <https://github.com/Joxit/docker-registry-ui#recommended-docker-registry-usage>
 - Necesario incluir lo siguiente (en negrita):

```
services:
  registry-ui:
    image: joxit/docker-registry-ui:main
    ...
  registry-server:
    image: registry:2.8.2
    restart: always
    ports:
      - 5000:5000
    ...
```



Registro Docker

- Comandos básicos:

- Elegir una imagen a subir al registro.
 - Alternativamente, descargar una de Docker Hub, p.e. "ubuntu":

```
docker pull ubuntu
```

- Etiquetar la imagen para que apunte al registro local:

```
docker image tag ubuntu localhost:5000/mi-ubuntu
```

- Enviar una imagen al registro:

```
docker push localhost:5000/mi-ubuntu
```



Registro Docker

- Comandos básicos:
 - Descargar una imagen del registro:

```
docker pull localhost:5000/mi-ubuntu
```

- Parar el registro y borrar su contenido:

```
docker container stop registry  
docker container rm -v registry
```



Registro Docker

- Detalles sobre la convención de nombres:
 - El nombre de una imagen:
 - Está formado por una serie de componentes separados por el carácter /
 - Opcionalmente, puede ir precedido por la URL del registro
 - A su vez, opcionalmente, esta URL puede incluir un número de puerto
 - Opcionalmente, puede incluir su número de versión al final, utilizando el carácter : como separador.
 - Si la URL del registro no está presente, se toma por defecto el registro público: registry-1.docker.io
 - Si la versión no se indica al final, se asume "latest" por defecto.



Registro Docker

- Detalles sobre la convención de nombres:
 - Los valores por defecto nos permiten acortar los nombres
 - Ejemplo:

```
docker pull ubuntu
```

- ... es la versión resumida de:

```
docker pull docker.io/library/ubuntu
```



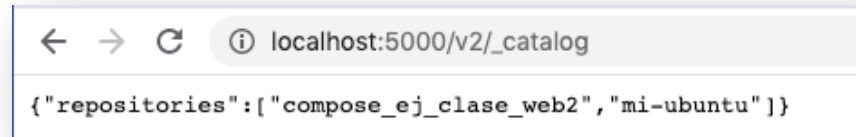
Registro Docker

- Auditando el registro
 - El registro incorpora una API REST para operaciones CRUD:
 - Se puede consultar con un cliente REST (p.e. CURL o Postman)
 - Los endpoint GET se pueden consultar con un navegador
 - Los resultados se devuelven en formato JSON.
 - La API completa en: <https://docs.docker.com/registry/spec/api/>



Registro Docker

- Auditando el registro
 - Ejemplos de las operaciones GET de la API:
 - `<URL>/v2/_catalog`: Listado de las imágenes almacenadas



A screenshot of a web browser window. The address bar shows the URL `localhost:5000/v2/_catalog`. Below the address bar, the JSON response is displayed: `{"repositories":["compose_ej_clase_web2","mi-ubuntu"]}`.

- `<URL>/v2/<nombre-imagen>/tags/list`: Tags de una imagen almacenada



A screenshot of a web browser window. The address bar shows the URL `localhost:5000/v2/mi-ubuntu/tags/list`. Below the address bar, the JSON response is displayed: `{"name":"mi-ubuntu","tags":["latest"]}`.



Ejercicio 3

- Lanzar un registro Docker local con la interfaz de Joxit
 - Verificar que está en marcha con un navegador
- Taggear la 1ª imagen creada en el ejercicio anterior y subirla al registro local.
- Verificar que la imagen se ha subido correctamente utilizando la interfaz Web.



Docker Hub

- Es el registro oficial de imágenes Docker.
- Por defecto, Docker Engine busca las imágenes en Hub.
 - Permite que los clientes Docker descarguen (*pull*) y envíen imágenes (*push*).
- Equivalente a GitHub en el contexto Docker.
 - Permite crear repositorios públicos y privados.



Docker Hub

- Modelo de negocio *freemium*
 - Precios a fecha 18 de octubre de 2023

Personal ¹ Includes the Docker essentials and is ideal for individual developers, education, open source communities, and small businesses. \$0 <ul style="list-style-type: none">• Docker Desktop ⓘ• Unlimited public repositories• Docker Engine + Kubernetes ⓘ• Limited image pulls per day• Unlimited scoped tokens ⓘ	Pro ¹ Extends the Docker capabilities and includes pro tools for individual developers who want to accelerate their productivity. \$5 /month ← Everything in Personal, plus: <ul style="list-style-type: none">• Docker Desktop ⓘ• Unlimited private repositories• 5,000 image pulls per day• 5 concurrent builds ⓘ• 300 vulnerability scans	Team ⁵⁺ Ideal for teams and includes capabilities for enhanced collaboration, productivity and security. \$9 /user/month max 100 users minimum 5 seats ← Everything in Pro, plus: <ul style="list-style-type: none">• Docker Desktop ⓘ• Unlimited teams• 15 concurrent builds ⓘ• Unlimited vulnerability scans• Add users in bulk• Audit logs ⓘ	Business ⁵⁺ Ideal for medium and large businesses who need centralized management and advanced security capabilities. \$24 /user/month only available with an annual subscription ← Everything in Team, plus: <ul style="list-style-type: none">• Hardened Docker Desktop• Enhanced Container Isolation• Settings management• Centralized management• Image Access Management• Registry Access Management• Single Sign-On (SSO)• SCIM user provisioning
---	---	--	--



Docker Hub

- Conceptos:
 - Docker ID:
 - Nombre elegido al crear una cuenta en Hub
 - Es el espacio de nombres para los servicios Docker en Hub
 - Repositorios:
 - Espacios para almacenar imágenes Docker.
 - Un repositorio puede almacenar múltiples imágenes con diferentes tags.
 - Organizaciones:
 - Grupos de equipos y repositorios que se gestionan juntos.
 - Equipos:
 - Grupos de usuarios de Docker Hub que pertenecen a una organización.



Docker Hub

- Creación de un repositorio:


Create Repository


ulopeznvoa | servidor-web

Repositorio para el servidor Web

Visibility

Using 1 of 1 private repositories. [Get more](#)


☒ **Public**  Public repositories appear in Docker Hub search results



☐ **Private**  Only you can view private repositories

Build Settings *(optional)*

Autobuild triggers a new build with every git push to your source code repository. [Learn More](#).

Please re-link a GitHub or Bitbucket account

 We've updated how Docker Hub connects to GitHub and Bitbucket. You'll need to re-link a GitHub or Bitbucket account to create new automated builds. [Learn More](#)

 Disconnected  Disconnected

[Cancel](#) [Create](#) [Create & Build](#)

Pro tip

You can push a new image to this repository using the CLI

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

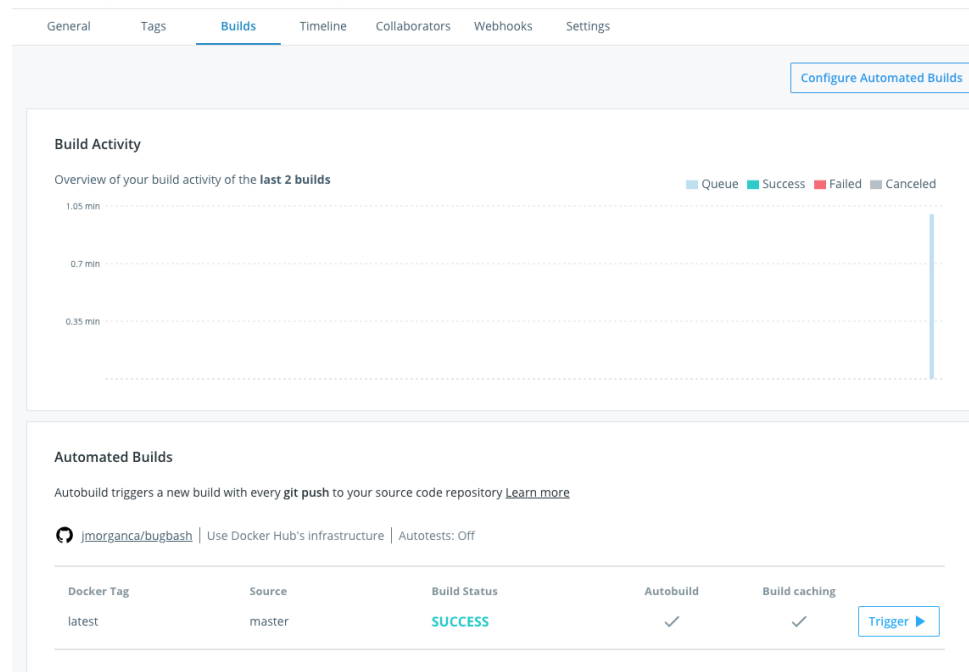
Make sure to change *tagname* with your desired image repository tag.



Docker Hub

- Integración con GitHub

- Permite construir una imagen Docker automáticamente tras un cambio en el repositorio de código fuente.
 - Actualmente, es una característica de pago



Docker Hub

- Métodos de autenticación
 - Usuario y contraseña
 - Tokens de acceso
 - Cadenas de 32 caracteres alfanuméricos
 - Útiles para mantener un registro de las operaciones en un repositorio
 - Más información: <https://docs.docker.com/docker-hub/access-tokens/>
 - Autenticación de 2 factores
 - Generación códigos de 13 caracteres alfanuméricos de un solo uso
 - Más información: <https://docs.docker.com/docker-hub/2fa/>



Bibliografía

- Stephen Grider. "Docker and Kubernetes: The complete guide", Udemy, 2020.
 - <https://www.udemy.com/course/docker-and-kubernetes-the-complete-guide>
- Jean-Philippe Gouigoux. "Docker. Primeros pasos y puesta en práctica de una arquitectura basada en micro-servicios", 2018.
 - Editorial ENI, ISBN: 2409015891.
- Adrian Mouat. "Understanding Volumes in Docker", Container-solutions, 2017.
 - <https://blog.container-solutions.com/understanding-volumes-docker>
- Consultados en octubre 2020

