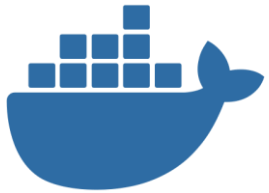
MASTER EXPERTO EN DATA SCIENCE Y BIG DATA

Edwin Delgado Paredes



Docker

Objetivo

En este curso aprenderás lo que son los contenedores, crearlos y administrarlos.

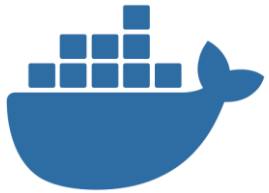
Con estos conceptos tendrás las bases para continuar con el curso de Kubernetes.

Como prerrequisito es necesario tener instalado docker-desktop

https://docs.docker.com/desktop/install/windows-install/

https://docs.docker.com/desktop/install/mac-install/

https://docs.docker.com/desktop/install/linux-install/



Docker

Contenido

Tecnología de contenedores

Daremos un paseo en el mundo de los contenedores, beneficios y principales retos.

Arquitectura Docker

Revisaremos los principales componentes de Docker y cómo se interrelacionan entre sí.

Cliente Docker

Estudiaremos y practicaremos los comandos más comunes que nos da Docker para trabajar con los contenedores.

Administración de contenedores

Echaremos un vistazo a herramientas de administración de contenedores y nos centraremos especialmente en Docker Desktop.

Registros de contenedores

Revisaremos un conjunto de librerías de imágenes de contenedores y en especial nos centraremos en Docker Hub.

Docker Compose

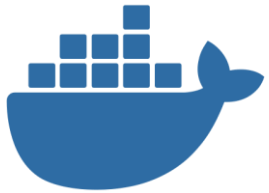
Estudiaremos como podemos definir y ejecutar varias aplicaciones en contenedores de forma fácil y rápida.

Herramientas de contenedores

Hay una amplia variedad de herramientas de terceros que pueden ser integradas e incrementar la productividad con contenedores Docker.

Orquestación de contenedores

Analizaremos las herramientas de orquestación de contenedores y veremos en detalle Docker Swarm.



Docker

Contenido

Tecnología de contenedores

Daremos un paseo en el mundo de los contenedores, beneficios y principales retos.

Arquitectura Docker

Revisaremos los principales componentes de Docker y cómo se interrelacionan entre sí.

Cliente Docker

Estudiaremos y practicaremos los comandos más comunes que nos da Docker para trabajar con los contenedores.

Administración de contenedores

Echaremos un vistazo a herramientas de administración de contenedores y nos centraremos especialmente en Docker Desktop.

Registros de contenedores

Revisaremos un conjunto de librerías de imágenes de contenedores y en especial nos centraremos en eDocker Hub.

Docker Compose

Estudiaremos como podemos definir y ejecutar varias aplicaciones en contenedores de forma fácil y rápida.

Herramientas de contenedores

Hay una amplia variedad de herramientas de terceros que pueden ser integradas e incrementar la productividad con contenedores Docker.

Orquestación de contenedores

Analizaremos las herramientas de orquestación de contenedores y veremos en detalle Docker Swarm.

Tecnología de contenedores

Entendiendo los contenedores

- Base de la informática moderna

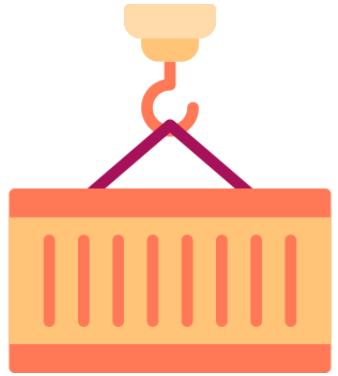
  El desarrollo y despliegue de aplicaciones mediante la tecnología de contenedores ha cambiado por completo la forma de desarrollar y gestionar el software. En esencia, la tecnología de contenedores consiste en combinar aplicaciones y todas las dependencias que requieren en un único contenedor.

- Método estandarizado

  A diferencia del despliegue tradicional de aplicaciones, en el que las diferencias en los entornos informáticos y las dependencias pueden causar problemas. Gracias a la adopción de los contenedores, las aplicaciones ya no se ven limitadas por las características del sistema anfitrión. Al llevar todo su entorno dentro de un contenedor, son mucho más adaptables y portátiles.

- Arte de la simplificación

  Los desarrolladores y los equipos de operaciones pueden utilizar la contenerización para tomar la idea detrás de la tecnología de contenedores y convertirla en una estrategia viable para una distribución de software más eficaz y fiable.



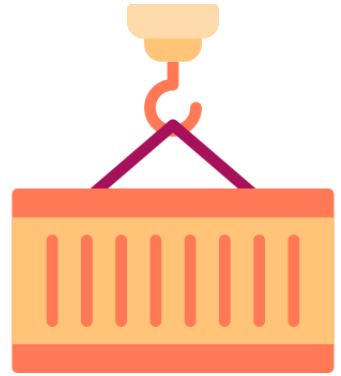
Así como en el sector del transporte se usan contenedores para llevar de forma aislada una carga que contiene una serie de productos, en el mundo del desarrollo del software se utiliza cada vez más este método para empaquetar una aplicación, distribuirla y ejecutarla.

Tecnología de contenedores

Entendiendo los contenedores

- Aplicaciones como paquetes contenerizados

  Las aplicaciones se empaquetan en contenedores a través de la tecnología de contenerización, y cada uno tiene todo lo necesario para que una aplicación funcione sin problemas. Esto incluye las configuraciones, el runtime, las bibliotecas y el código del programa. El futuro de la contenerización se transforma en una unidad autocontenida que es fácil de desplazar, lo que proporciona un rendimiento fiable y reduce el despliegue.



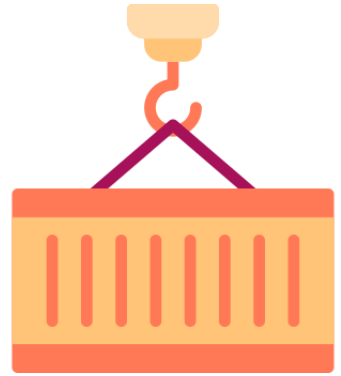
Así como en el sector del transporte se usan contenedores para llevar de forma aislada una carga que contiene una serie de productos, en el mundo del desarrollo del software se utiliza cada vez más este método para empaquetar una aplicación, distribuirla y ejecutarla.

Tecnología de contenedores

Importancia

- Ofrecen portabilidad, escalabilidad y eficiencia de recursos
  - Las aplicaciones y sus dependencias están contenidas en los contenedores, lo que garantiza su funcionamiento fiable en diversos entornos. Esta coherencia simplifica el proceso de despliegue, reduce su complejidad y resuelve el frustrante problema de "funciona en mi máquina".
  - Los contenedores proporcionan una escalabilidad sencilla, lo que permite a las aplicaciones responder fácilmente a las cargas de trabajo cambiantes.
  - Son más eficientes en el uso de recursos que las técnicas de virtualización tradicionales porque comparten el núcleo del sistema operativo anfitrión, lo que optimiza la utilización de recursos.
- Aislamiento, seguridad y fiabilidad
  - La tecnología de contenedores de aplicaciones permite un aislamiento sólido, lo que mejora la seguridad y la fiabilidad.
  - El ciclo de vida del desarrollo de software se acelera por su fácil integración en las prácticas DevOps, mientras que la modularidad y la flexibilidad se potencian por su conformidad con la arquitectura de microservicios.

La contenerización ayuda a desplegar software más rápidamente y con mayor fiabilidad y eficiencia.



Así como en el sector del transporte se usan contenedores para llevar de forma aislada una carga que contiene una serie de productos, en el mundo del desarrollo del software se utiliza cada vez más este método para empaquetar una aplicación, distribuirla y ejecutarla.

Tecnología de contenedores

Definiciones

- Contenedor

  Es un empaquetado de software que agrupa todos los archivos de una aplicación, con sus dependencias y archivos de configuración necesarios donde se ejecuta de forma aislada.

  Este empaquetado se guarda como una imagen.

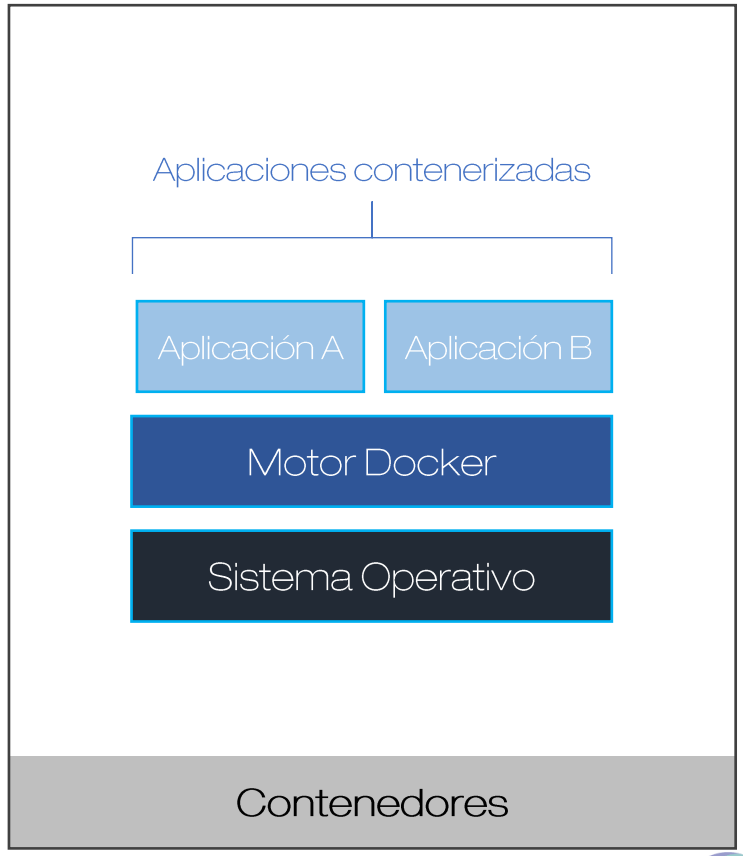
  Se puede desplegar una imagen descargándola y ejecutándola de forma independiente en el sistema operativo huésped de cualquier entorno.

  Cada ejecución de una imagen se considera genera un contenedor o instancia de la aplicación y se puede ejecutar varias veces de forma simultánea.

  Se pueden ejecutar en un hardware físico, en la nube, en una máquina virtual.

- Motor de contenedores

  Es una pieza de software que acepta peticiones del usuario, incluyendo opciones de línea de comandos, extrae imágenes y, desde la perspectiva del usuario final, ejecuta el contenedor.



Tecnología de contenedores

Definiciones

- Plataforma de ejecución de contenedores (runtime)

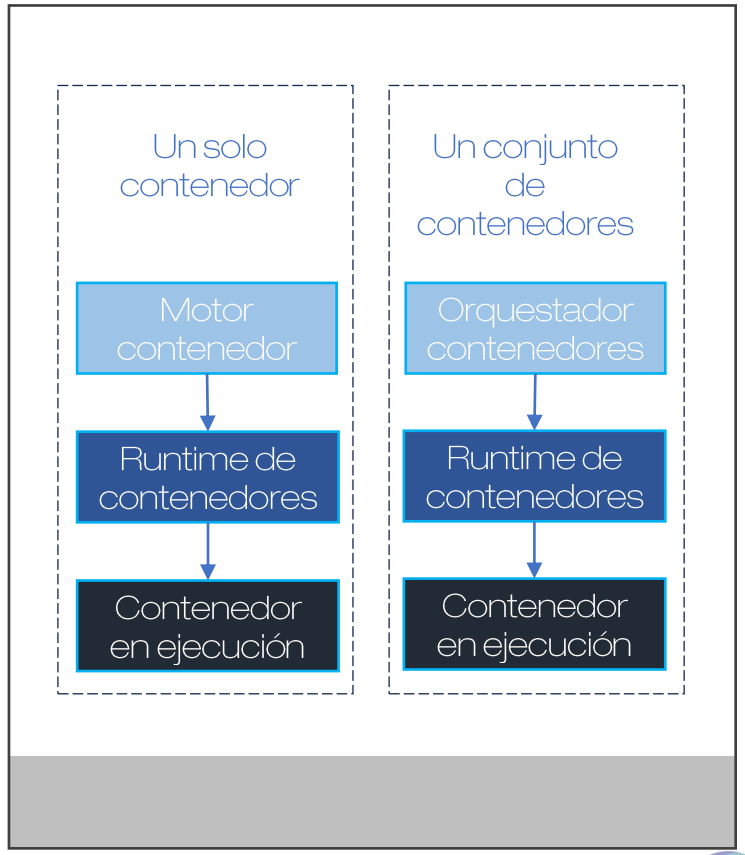
  Se encarga de **gestionar el ciclo de vida del contenedor**: configurar su entorno, ejecutarlo, detenerlo, etcétera. Los tiempos de ejecución pueden subdividirse en dos tipos: de alto nivel y de bajo nivel:

  - **Alto nivel**, estos runtimes gestinan imágenes y la integración con tiempos de ejecución de bajo nivel para ejecutar el proceso de contenedor.
  - **Bajo nivel**, Estos tipos de runtimes son responsables de generar un contenedor mediante llamadas al sistema del kernel de Linux

  También hay que tener en cuenta que algunos motores pueden comportarse como runtimes y, por lo tanto, pueden utilizarse desde dentro de otros motores, u orquestadores.

- Orquestador de contenedores

  Es un software encargado de **gestionar un conjunto de contenedores** a través de diferentes recursos informáticos, manejando las **configuraciones de red** y **almacenamiento** que se delegan en los runtime de los contenedores.



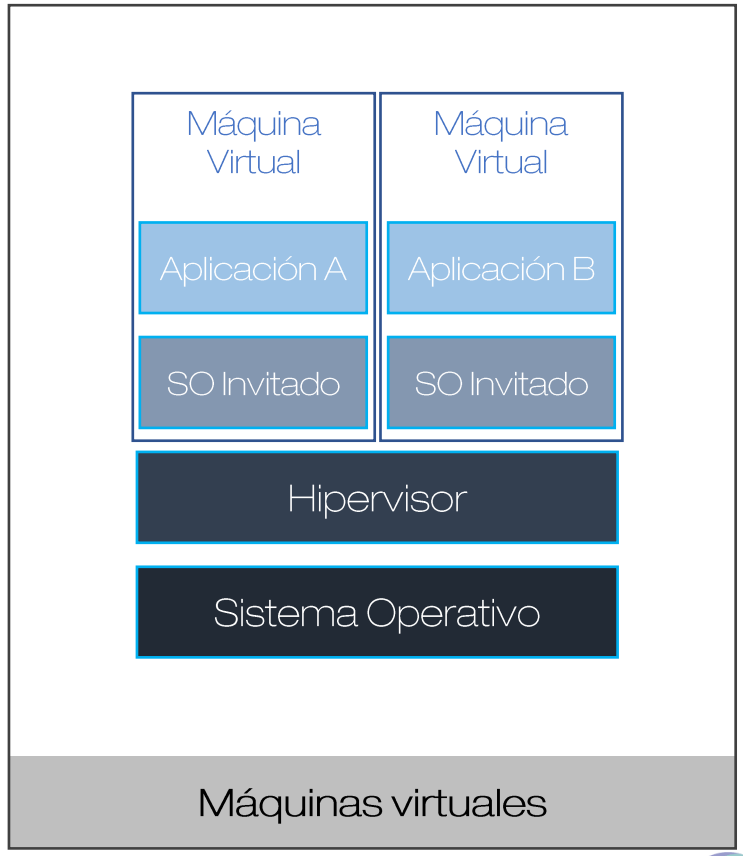
Tecnología de contenedores

Definiciones

- Máquinas virtuales

  Permiten:

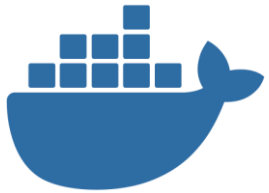
  - Virtualizan el hardware subyacente para que se puedan ejecutar múltiples sistemas operativos dentro del sistema operativo huésped.
  - Aprovisionan de servidores.
  - Una imagen de una máquina virtual contiene el sistema operativo, librerías y aplicaciones, por lo que suele ocupar bastante espacio.



Tecnología de contenedores

Diferencias entre contenedores y máquinas virtuales

Aspecto	Contenedores	Máquinas virtuales
Nivel de aislamiento	Los contenedores ofrecen aislamiento a nivel de proceso. En el sistema operativo anfitrión, cada contenedor se ejecuta como un proceso independiente. Esto proporciona un buen aislamiento, pero no es tan fuerte como las máquinas virtuales.	Las máquinas virtuales, proporcionan un mayor aislamiento. En cada VM se ejecuta un sistema operativo invitado completo, lo que aumenta el nivel de separación entre las instancias de VM.
Eficiencia de recursos	Dado que comparten el núcleo del sistema operativo anfitrión, los contenedores son muy eficientes en el uso de recursos. Como resultado, tienen poca sobrecarga, lo que les permite maximizar el uso de recursos.	Dado que cada máquina virtual viene con un sistema operativo completo, las máquinas virtuales tienen una mayor sobrecarga de recursos. El uso de varias máquinas virtuales en un host puede sobrecargar sus recursos.
Tiempo de arranque	No tienen que arrancar un SO completo, por lo que los contenedores se inician muy rápidamente, casi al instante.	Sí tienen que arrancar un SO completo, por lo que generalmente se inician más lentamente en comparación con los contenedores



Tecnología de contenedores

Diferencias entre contenedores y máquinas virtuales

Aspecto	Contenedores	Máquinas virtuales
Portabilidad	Al incluir tanto el programa como sus dependencias, los contenedores son muy portables. Esto garantiza la coherencia en múltiples entornos.	Debido a su mayor tamaño y requisitos de recursos, las máquinas virtuales son menos portables.
Densidad	Debido a su menor sobrecarga de recursos, los contenedores permiten una mayor densidad de instancias de aplicación en un único host.	Debido a sus mayores necesidades de recursos, las máquinas virtuales suelen tener una menor densidad en un host.
Casos de uso	Los contenedores son ideales en situaciones en las que se requiere un escalado rápido, flexibilidad y optimización de recursos. Las técnicas DevOps, las aplicaciones nativas de la nube y las arquitecturas de microservicios las emplean con frecuencia.	Las máquinas virtuales son adecuadas para cargas de trabajo que requieren un aislamiento sólido, compatibilidad con programas heredados o la capacidad de ejecutar varios sistemas operativos en el mismo hardware.

Tecnología de contenedores

Definiciones

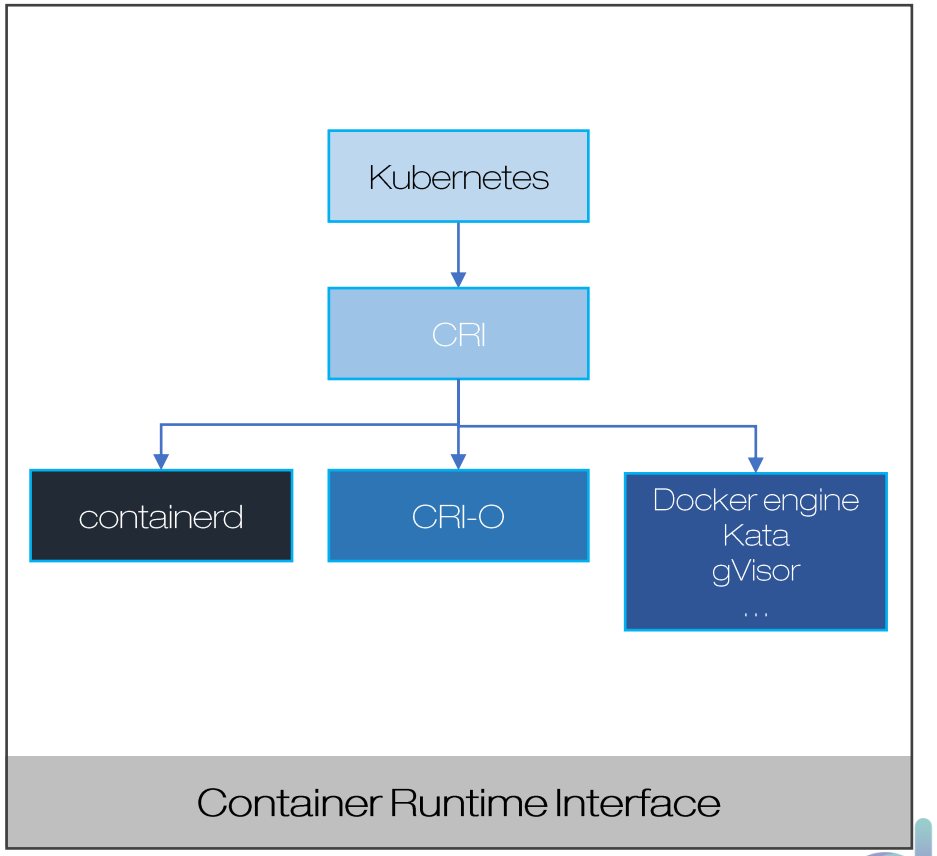
- Open Container Initiative (OCI)

  Docker y otros actores importantes del sector de los contenedores crearon la Open Container Initiative (OCI) en 2015. El objetivo de la OCI es crear estándares para formatos y runtimes de contenedores. Actualmente, la OCI cuenta con tres especificaciones principales:

  - image-spec - la especificación de imagen que describe cómo crear una imagen compatible con la OCI.
  - runtime-spec - la especificación del tiempo de ejecución para desempaquetar el paquete del sistema de archivos.
  - distribution-spec - define un protocolo API para facilitar y normalizar la distribución de contenidos.

- Container Runtime Interface (CRI)

  Al principio, Docker Engine era el único runtime disponible en la plataforma. Pero la popularidad de la contenerización dio lugar a soluciones competidoras y a la necesidad de que Kubernetes fuera compatible con todas ellas. Con el complemento Container Runtime Interface, Kubernetes puede comunicarse con los principales runtimes.

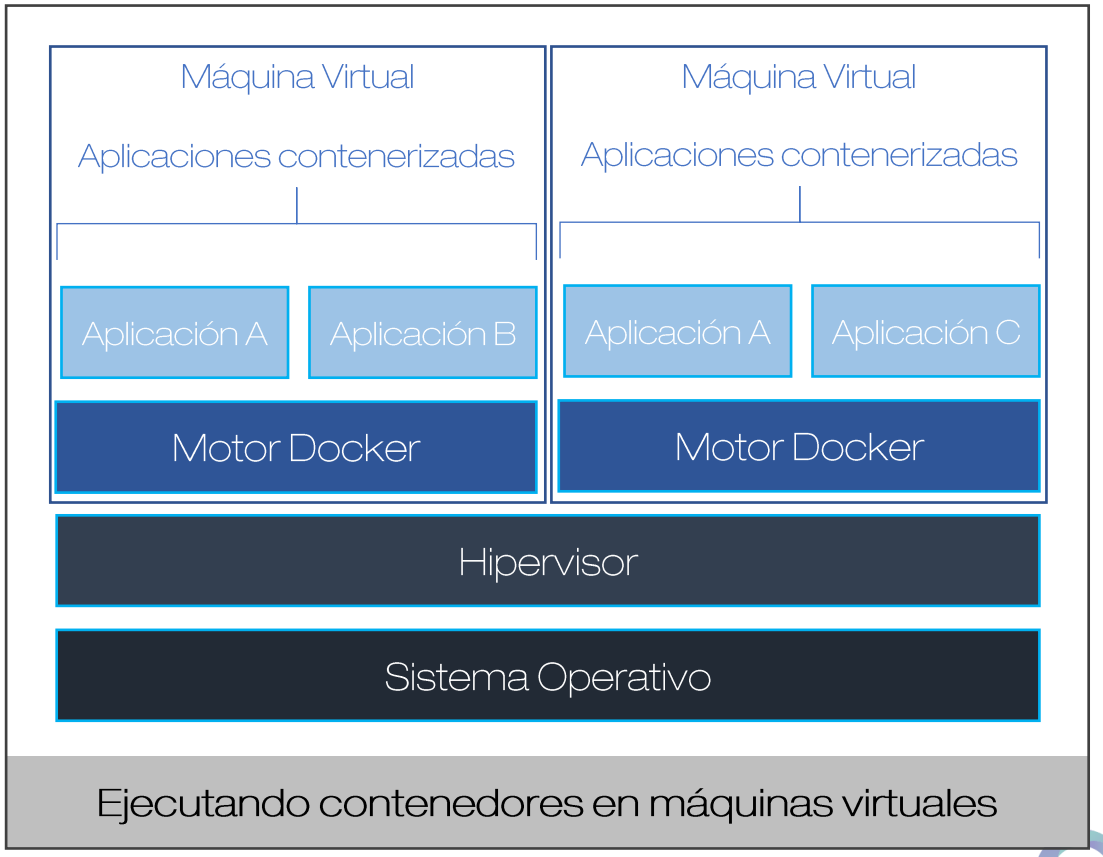


Contenedores

Contenedores en máquinas virtuales

Los contenedores en máquinas virtuales:

- Poner una capa de virtualización en el sistema operativo que maximiza la optimización de hardware.
- La capa de virtualización no necesita compartir el mismo sistema operativo base, lo que facilita su portabilidad entre distintas ubicaciones.
- Esta virtud de la capa de virtualización es útil para los contenedores que son dependientes de un sistema operativo y porque les provee de un entorno consistente para ejecutarse, incluso si el sistema operativo base es distinto.
- Esto tiene una penalización de rendimiento al agregar una capa de abstracción sobre el hardware.



Tecnología de contenedores

Plataformas más populares de contenerización

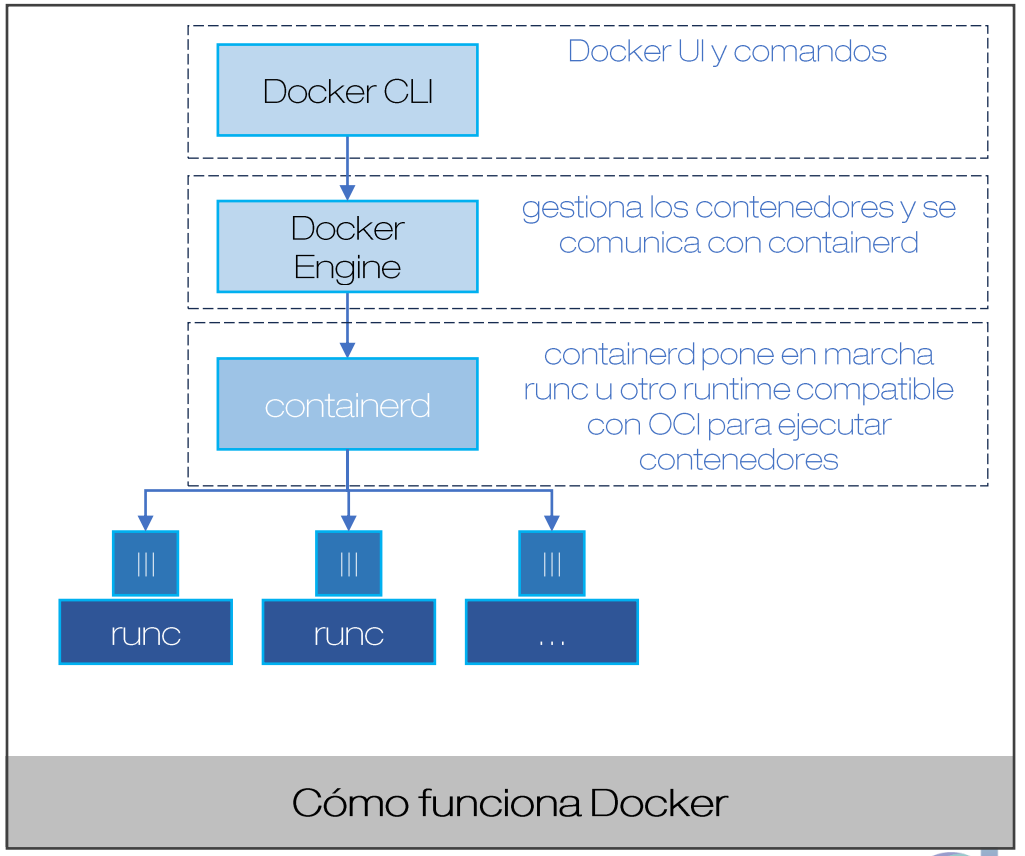
- Docker

En muchos sentidos, Docker y contenerización se han convertido en términos intercambiables. Cuando la gente habla de contenedores, Docker es con frecuencia el primer nombre que viene a la mente.

Muchos desarrolladores y organizaciones consideran ahora que Docker es la solución de contenerización preferida debido a su amplio atractivo e influencia en las tecnologías de contenerización.

Ofrece una plataforma completa para crear, empaquetar y distribuir contenedores. La interfaz fácil de usar de Docker y su gran ecosistema de imágenes y plugins lo han elevado a lo más alto de la lista entre desarrolladores y empresas de todo el mundo.

Además, el gran apoyo prestado por la comunidad lo complementa y lo convierte en la mejor opción para quienes buscan una solución tecnológica de contenerización robusta.



Tecnología de contenedores

Plataformas más populares de contenerización

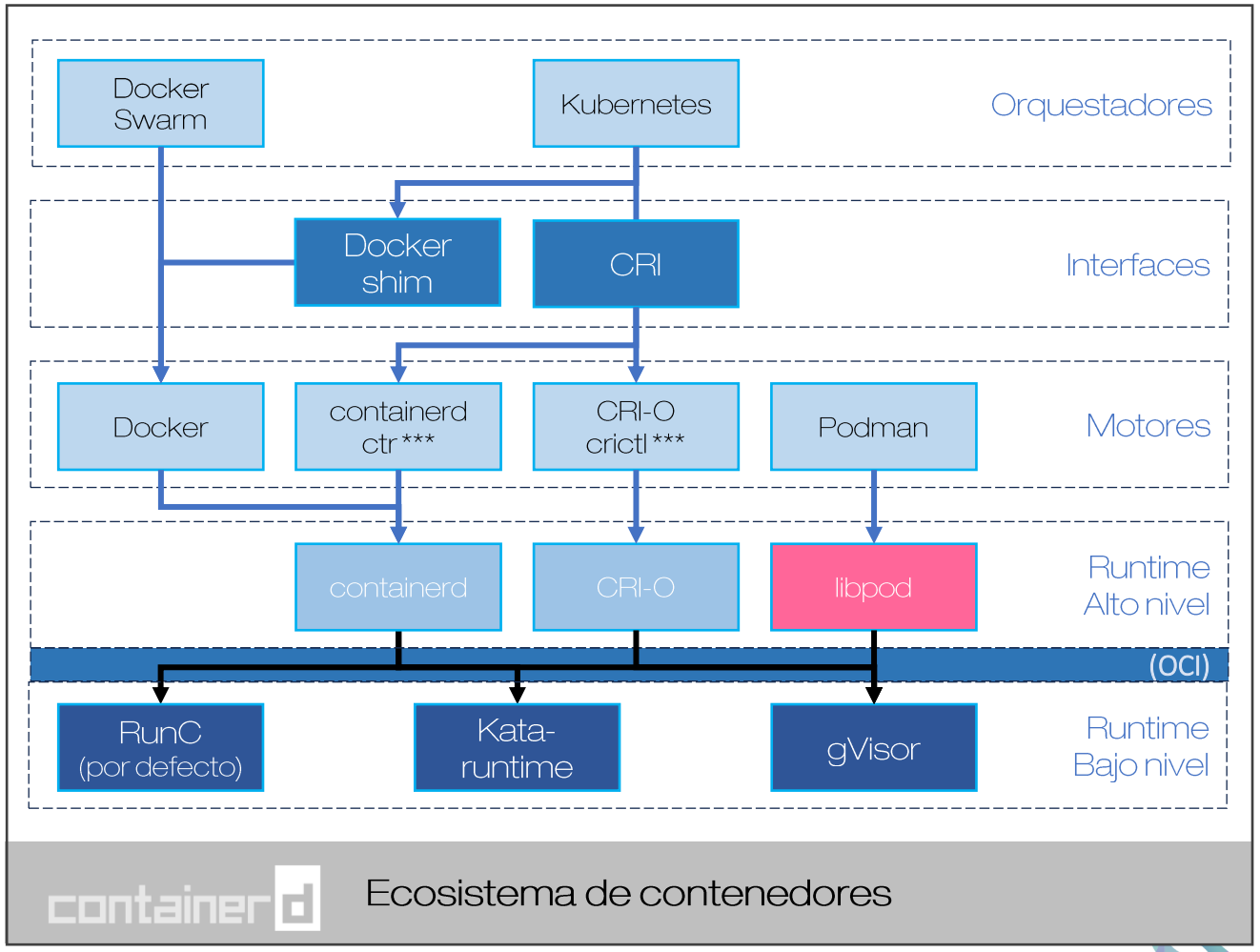
- Containerd

  Creado originalmente por Docker y cedido posteriormente a la Cloud Native Computing Foundation (CNCF)

  Containerd es un runtime de contenedores ampliamente utilizado. Proporciona un entorno de ejecución compacto y eficaz para los contenedores y actúa como un bloque de construcción fundamental para las plataformas de contenedores y orquestadores.

  containerd está disponible como demonio para Linux y Windows. Gestiona el ciclo de vida completo de contenedores de su sistema anfitrión, desde la transferencia y el almacenamiento de imágenes hasta la ejecución y supervisión del contenedor, pasando por el almacenamiento de bajo nivel, los anexos de red y mucho más. Algunas características:

  - Especificación de imágenes OCI.
  - Especificación de runtime OCI (runC)
  - Ciclo de vida y runtime de contenedores y más…



Tecnología de contenedores

Plataformas más populares de contenerización

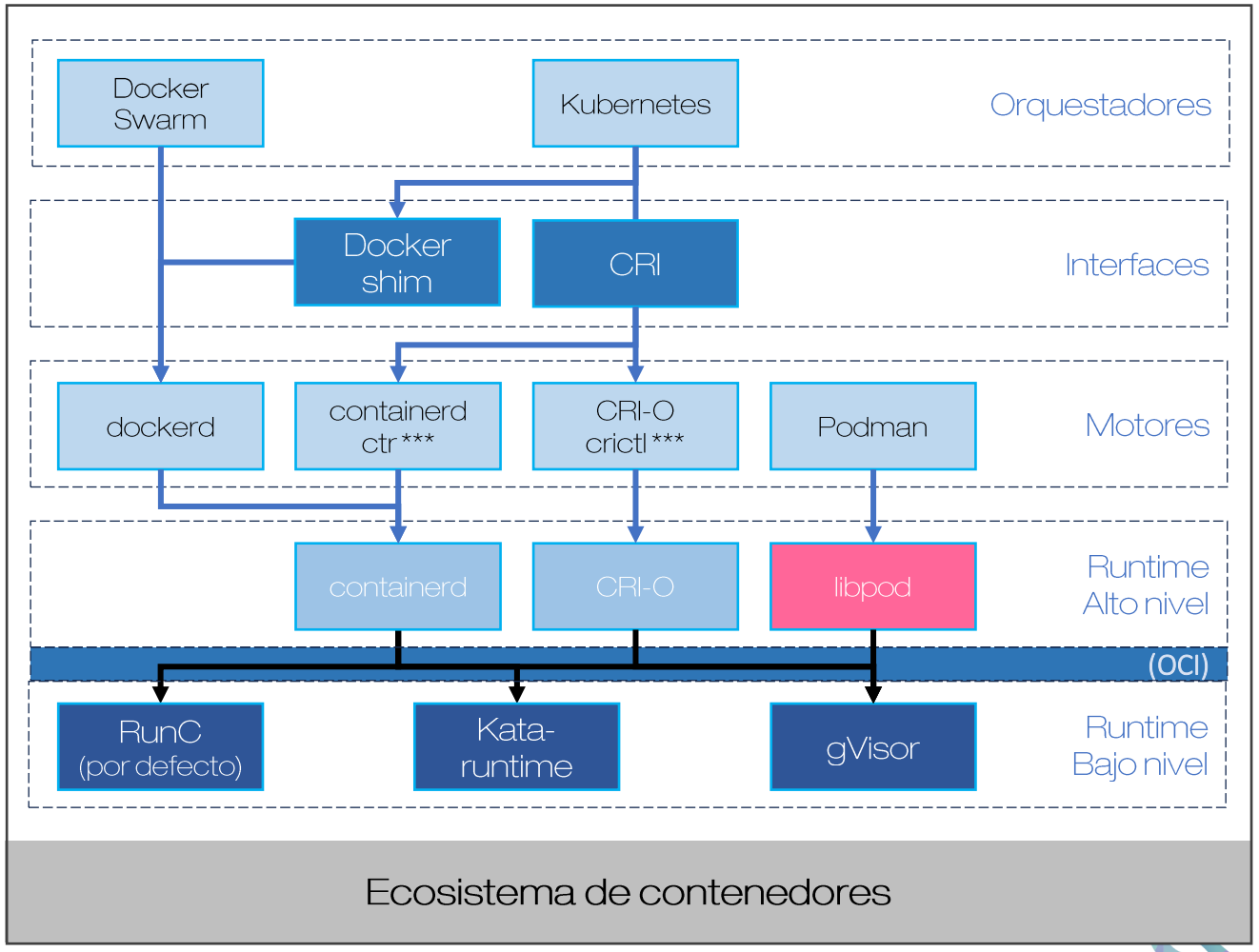
- CRI-O

  CRI-O sirve como implementación de Kubernetes CRI (Container Runtime Interface), facilita la utilización de runtimes compatibles con OCI (Open Container Initiative).

  CRI-O es totalmente compatible con imágenes de contenedores OCI y posee la capacidad de extraer de varios registros de contenedores, proporcionando un sustituto para Docker, Moby o Rkt.

  Está construido de varios componentes:

  - Runtime compatible con OCI
  - Redes (CNI) que da soporte a varios plugins como Flannel, Weave y Openshift.
  - Containers/images para descarga de imágenes.
  - Conmon para monitorización de contenedores.
  - Políticas de seguridad, incluidas las especificadas en OCI.



Tecnología de contenedores

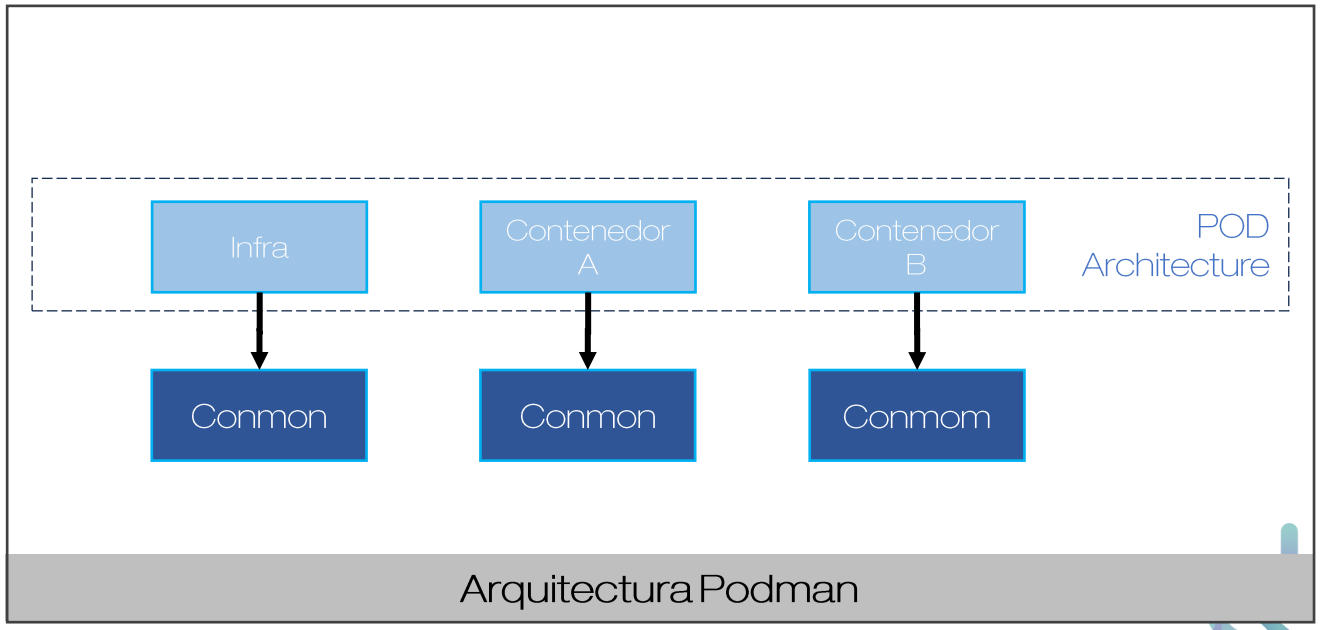
Plataformas más populares de contenerización

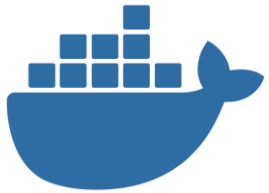
- Podman (Pod manager)

  Es una popular tecnología de contenerización que está madurando rápidamente para competir con Docker. A diferencia de Docker que utiliza un demonio para la gestión de contenedores, Podman se aproxima a los contenedores con una tecnología sin demonio llamada Conmon que se encarga de las tareas de creación de contenedores, almacenamiento del estado y extracción de imágenes de contenedores, etc.

  Esta capacidad de gestionar múltiples contenedores "out-of-the-box" usando comandos a nivel de pod es lo que hace especial a Podman. En comparación con la tecnología Docker, Conmon ocupa menos memoria. Para crear un pod, es necesario crear un archivo de manifiesto utilizando el formato declarativo y el lenguaje de serialización de datos YAML. Kubernetes consume estos manifiestos para su marco de orquestación de contenedores.

Podman heredó la CLI de Docker, lo que simplifica la migración de Docker a Podman. Algunas distribuciones de Linux incluso incluyen un paquete podman-docker, que crea un alias de docker a podman.





Tecnología de contenedores

Retos

- Seleccionar la correcta tecnología de contenedores.

  Reto: El ecosistema de los contenedores es enorme y existen muchas plataformas y herramientas diferentes. Puede resultar difícil elegir las que mejor se adaptan a sus necesidades particulares.

  Consideraciones: En función de las demandas de su aplicación, los requisitos de escalabilidad y la experiencia disponible, evalúe los runtimes de contenedores, los orquestadores y los registros.

- Vendor Lock-in

  Reto: Puede resultar difícil cambiar a otras soluciones si algunas plataformas y servicios de contenedores te encierran en su ecosistema.

  Consideraciones: Para evitar la dependencia de un proveedor, dé prioridad a los estándares abiertos y a la compatibilidad. Seleccione soluciones compatibles con contenedores portátiles

- Seguridad

  Reto: Si no se configuran y supervisan correctamente, los contenedores pueden plantear vulnerabilidades de seguridad.

  Consideración: Recuerde emplear herramientas de seguridad en tiempo de ejecución, imágenes de contenedores seguras y parchear y actualizar las aplicaciones en contenedores con regularidad.

Tecnología de contenedores

Retos

- Administración de recursos

  Reto: Puede resultar difícil distribuir eficazmente y hacer un seguimiento de los recursos de almacenamiento, CPU y memoria entre los contenedores.

  Consideración: Automatizar la gestión de recursos y el escalado en función de la demanda utilizando soluciones de orquestación de contenedores como Kubernetes.

- Complejidad en la orquestación

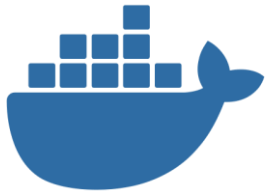
  Reto: La gestión de contenedores a escala requiere herramientas de orquestación, lo que puede añadir complejidad.

  Consideraciones: Para utilizar plataformas de orquestación como Kubernetes de forma eficiente, invertir en formación y conocimientos, y para simplificar, pensar en servicios gestionados de Kubernetes.

- Soporte limitado para aplicaciones antiguas/monolito.

  Reto: Debido a dependencias o limitaciones de diseño, algunos programas antiguos/monolitos podrían ser difíciles de contenerizar.

  Consideración: Tener en cuenta técnicas como la contenerización de componentes, la utilización de máquinas virtuales junto con contenedores o la conversión de programas tradicionales en microservicios.



Tecnología de contenedores

Retos

- Administración de datos

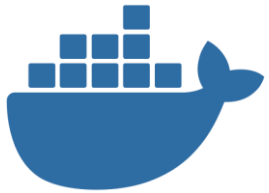
  Reto: Implantar planes de copia de seguridad y recuperación y garantizar la persistencia de los datos puede resultar difícil en un entorno de contenedores.

  Consideración: Para manejar los datos en contenedores de manera efectiva, utilice opciones de almacenamiento externo, bases de datos nativas de la nube o conjuntos con estado en Kubernetes.

- Adaptar los procesos para dar soporte a contenedores

  Reto: La adopción de contenedores suele requerir la modificación de los métodos utilizados para el desarrollo, las pruebas y la implantación.

  Consideración: Para automatizar las pruebas, agilizar los procesos y permitir la integración continua y el despliegue continuo (CI/CD), es importante promover una cultura DevOps..



Docker

Contenido

Tecnología de contenedores

Daremos un paseo en el mundo de los contenedores, beneficios y principales retos.

Arquitectura Docker

Revisaremos los principales componentes de Docker y cómo se interrelacionan entre sí.

Cliente Docker

Estudiaremos y practicaremos los comandos más comunes que nos da Docker para trabajar con los contenedores.

Administración de contenedores

Echaremos un vistazo a herramientas de administración de contenedores y nos centraremos especialmente en Docker Desktop.

Registros de contenedores

Revisaremos un conjunto de librerías de imágenes de contenedores y en especial nos centraremos en eDocker Hub.

Docker Compose

Estudiaremos como podemos definir y ejecutar varias aplicaciones en contenedores de forma fácil y rápida.

Herramientas de contenedores

Hay una amplia variedad de herramientas de terceros que pueden ser integradas e incrementar la productividad con contenedores Docker.

Orquestación de contenedores

Analizaremos las herramientas de orquestación de contenedores y veremos en detalle Docker Swarm.

Arquitectura Docker

Definición

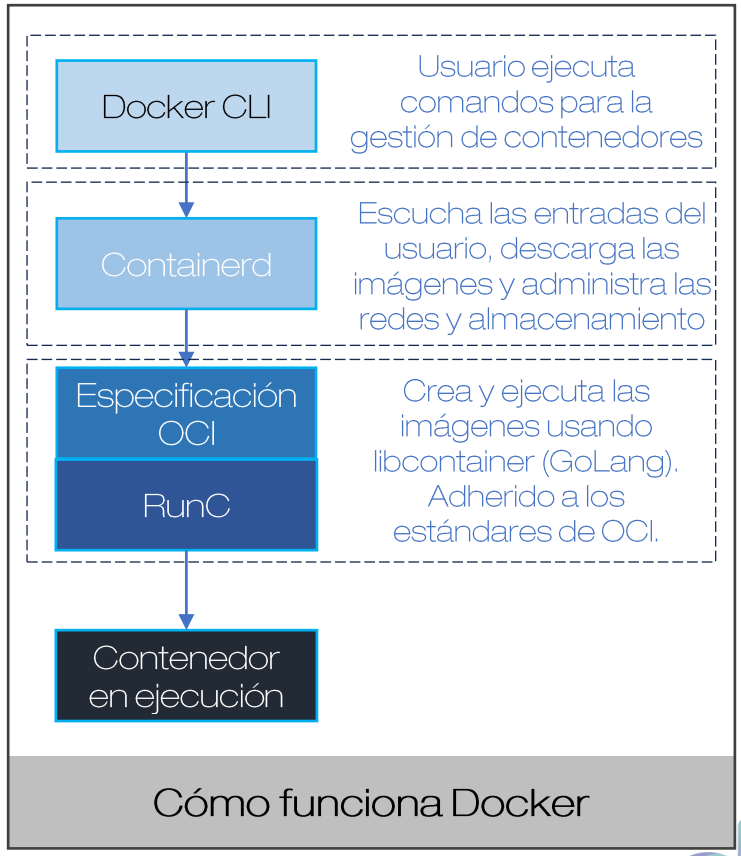
- ¿Qué es Docker?

Es una plataforma de creación de contenedores que se usa para desarrollar, distribuir y ejecutar contenedores dirigido por la empresa llamada Docker. Estos proyectos trabajan juntos para proporcionar una plataforma completa para el despliegue de contenedores.

Los proyectos más importantes son:

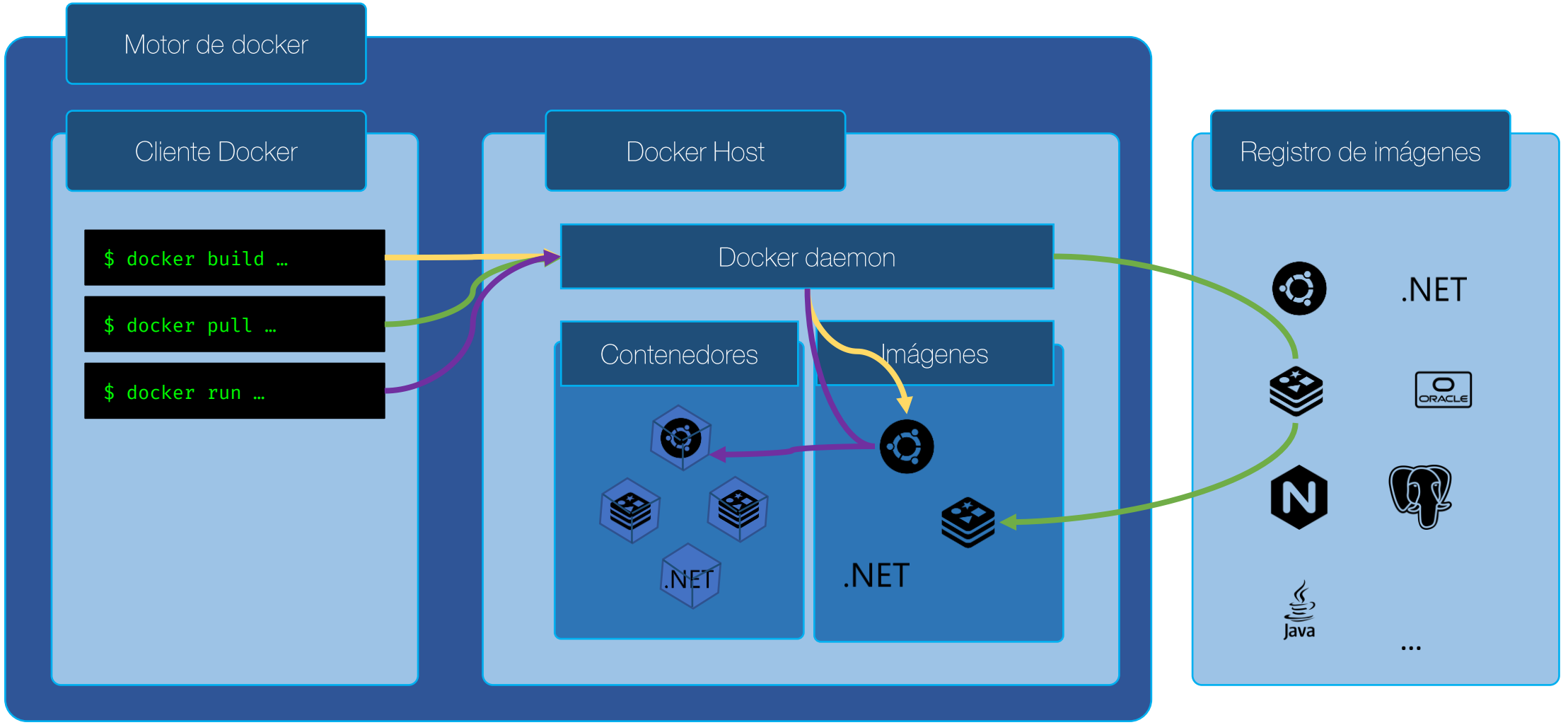
- **docker CLI** - Un programa de interfaz de **línea de comandos**. Los usuarios crean y gestionan contenedores Docker ejecutando comandos docker CLI que interactúa con **dockerd**(demonio docker).
- containerd - Un demonio que escucha los comandos del usuario. Extrae y almacena las imágenes solicitadas y controla el ciclo de vida del contenedor.
- runC - Un runtime de contenedores ligero y portátil. runC es un componente de bajo nivel que integra los componentes necesarios para que Docker interactúe con el sistema local. Los contenedores que crea esta herramienta son compatibles con OCI.

Docker tiene versiones para entornos de servidor, incluidos muchas variantes de Linux, Microsoft Windows Server 2016 y versiones posteriores.



Arquitectura Docker

La plataforma



Arquitectura Docker

La plataforma

- Motor de Docker

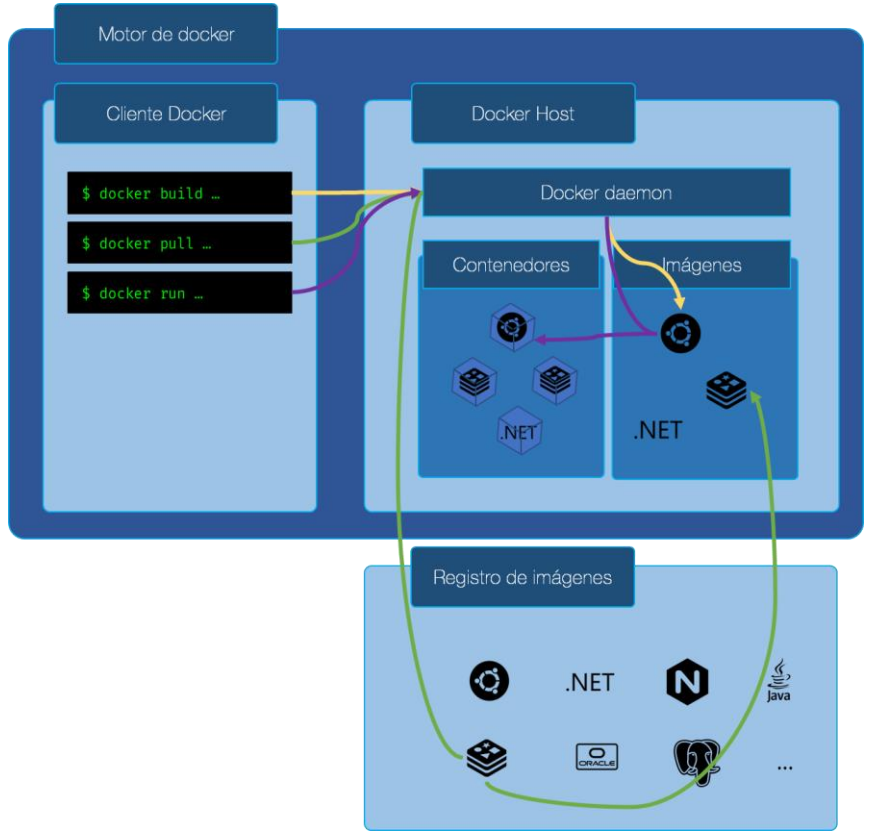
  El motor Docker utiliza containerd para gestionar el ciclo de vida del contenedor, que incluye la creación, arranque y parada de contenedores. Por defecto, containerd utiliza runc como tiempo de ejecución del contenedor.

- Demonio de Docker

  El demonio de Docker (dockerd) escucha las peticiones de la API de Docker y gestiona objetos de Docker como imágenes, contenedores, redes y volúmenes. Un demonio también puede comunicarse con otros demonios para gestionar los servicios de Docker.

- Cliente Docker

  El cliente Docker (docker) es la forma principal en que muchos usuarios de Docker interactúan con Docker. Cuando se utilizan comandos como docker run, el cliente envía estos comandos a dockerd, que los ejecuta. El comando docker utiliza la API de Docker. El cliente Docker puede comunicarse con más de un demonio.



Arquitectura Docker

La plataforma

- Docker Desktop

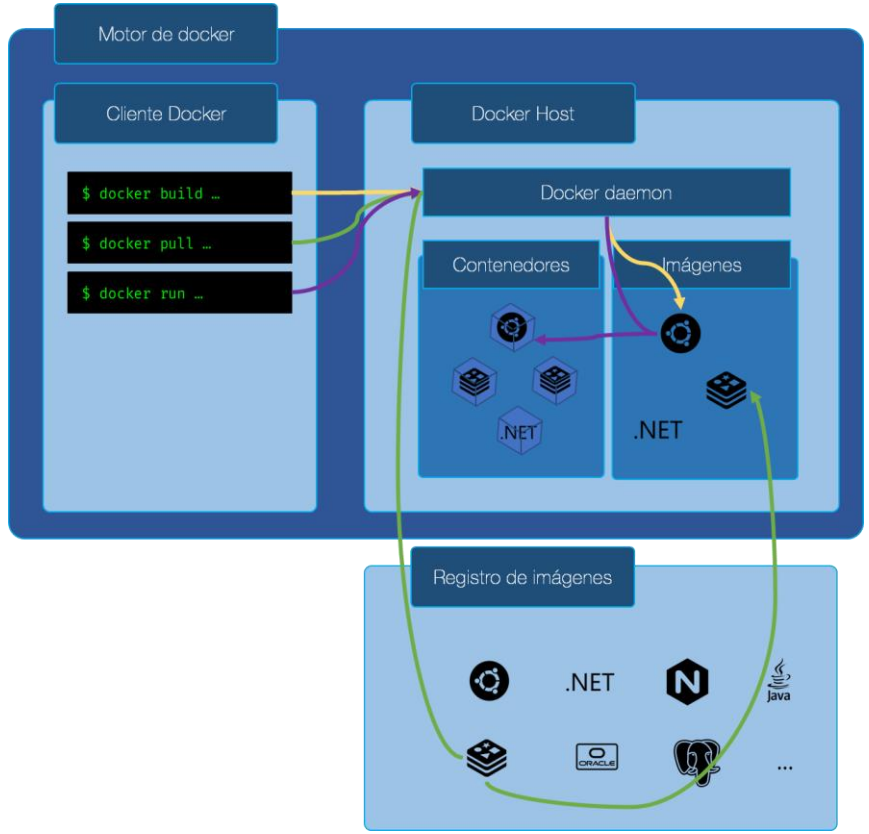
  Docker Desktop es una aplicación fácil de instalar para su entorno Mac, Windows o Linux que le permite crear y compartir aplicaciones y microservicios en contenedores. Docker Desktop incluye el demonio Docker (dockerd), el cliente Docker (docker cli), Docker Compose, Docker Content Trust, Kubernetes y Credential Helper.

- Registros Docker

  Un registro Docker almacena imágenes Docker. Docker Hub es un registro público que cualquiera puede usar, y Docker busca imágenes en Docker Hub por defecto. Incluso puedes ejecutar tu propio registro privado

- Objetos Docker

  Cuando utilizas Docker, estás creando y utilizando imágenes, contenedores, redes, volúmenes, plugins y otros objetos.



Arquitectura Docker

Instalar Docker Desktop

Install and run Docker Desktop on Mac

Install interactively | Install from the command line

1. Download the installer using the download buttons at the top of the page, or from the release notes.
2. Double-click `Docker.dmg` to open the installer, then drag the Docker icon to the **Applications** folder.
3. Double-click `Docker.app` in the **Applications** folder to start Docker.
4. The Docker menu displays the Docker Subscription Service Agreement.

   Here's a summary of the key points:

   - Docker Desktop is free for small businesses (fewer than 250 employees AND less than $10 million in annual revenue), personal use, education, and non-commercial open source projects.
   - Otherwise, it requires a paid subscription for professional use.
   - Paid subscriptions are also required for government entities.
   - Docker Pro, Team, and Business subscriptions include commercial use of Docker Desktop.
5. Select **Accept** to continue.

   Note that Docker Desktop won't run if you do not agree to the terms. You can choose to accept the terms at a later date by opening Docker Desktop.

   For more information, see Docker Desktop Subscription Service Agreement. We recommend that you also read the FAQs.
6. From the installation window, select either:
   - **Use recommended settings (Requires password)**. This let's Docker Desktop automatically set the necessary configuration settings.
   - **Use advanced settings**. You can then set the location of the Docker CLI tools either in the system or user directory, enable the default Docker socket, and enable privileged port mapping. See Settings, for more information and how to set the location of the Docker CLI tools.
7. Select **Finish**. If you have applied any of the above configurations that require a password in step 6, enter your password to confirm your choice.

Install Docker Desktop on Windows

Install interactively | Install from the command line

1. Download the installer using the download button at the top of the page, or from the release notes.
2. Double-click `Docker Desktop Installer.exe` to run the installer.
3. When prompted, ensure the **Use WSL 2 instead of Hyper-V** option on the Configuration page is selected or not depending on your choice of backend.
   If your system only supports one of the two options, you will not be able to select which backend to use.
4. Follow the instructions on the installation wizard to authorize the installer and proceed with the install.
5. When the installation is successful, select **Close** to complete the installation process.

If your admin account is different to your user account, you must add the user to the **docker-users** group:

1. Run **Computer Management** as an **administrator**.
2. Navigate to **Local Users and Groups** > **Groups** > **docker-users**.
3. Right-click to add the user to the group.
4. Sign out and sign back in for the changes to take effect.

Arquitectura Docker

Ciclo de vida de los contenedores

- Estado creado

  En el estado creado, un contenedor Docker es creado desde una imagen.

  docker create --name <name-of-container> <docker-image-name>

- Estado en ejecución

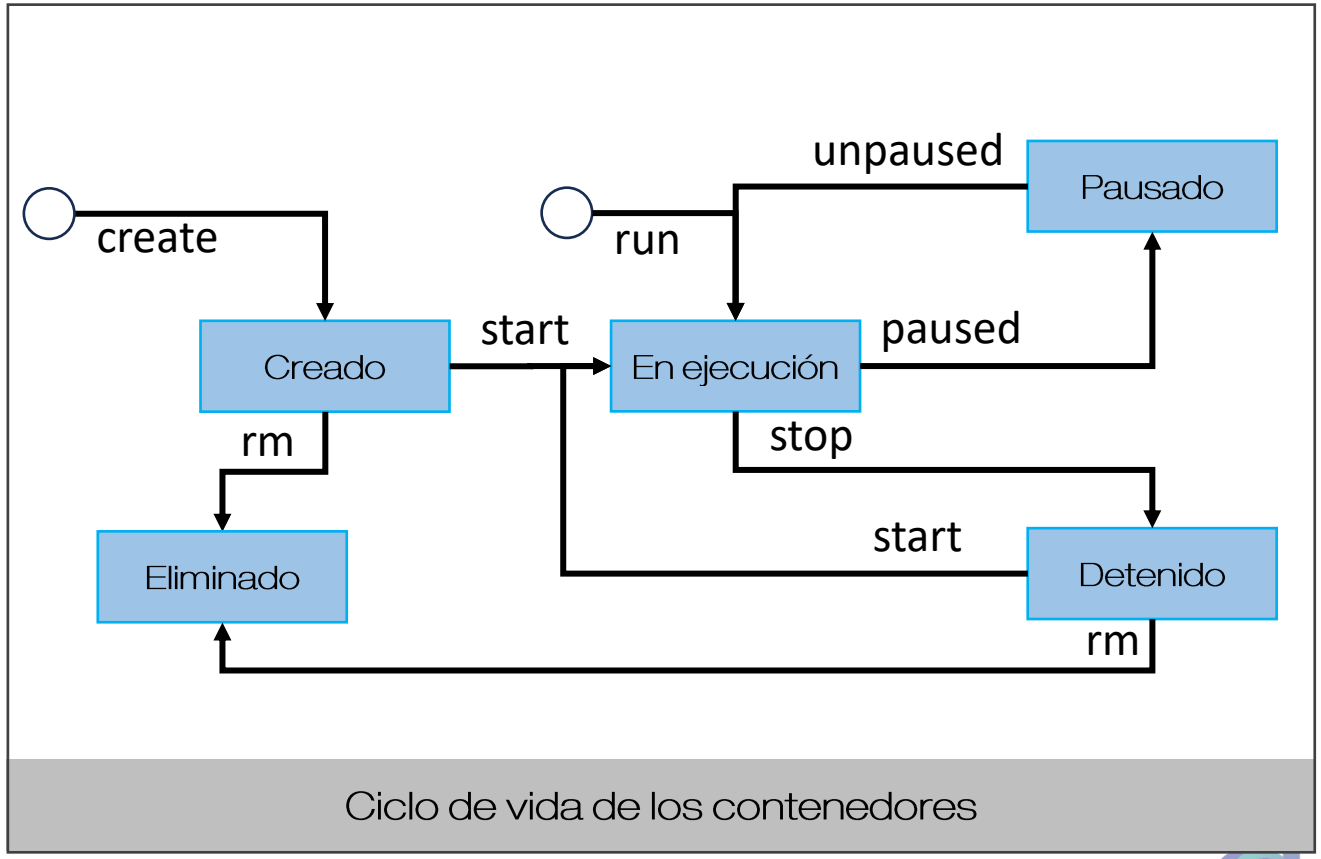
  En el estado en ejecución, el contenedor Docker inicar ejecutando los comandos mencionados en la imagen. Para ejecutar un contenedor Docker, usa el comando docker run.

  docker run <container-id>

  or

  docker run <container-name>

  Este comando crea el contenedor si este no está presente.



Ciclo de vida de los contenedores

Arquitectura Docker

Ciclo de vida de los contenedores

- Estado pausado

  En el estado pausado, el contenedor en ejecución se detiene.

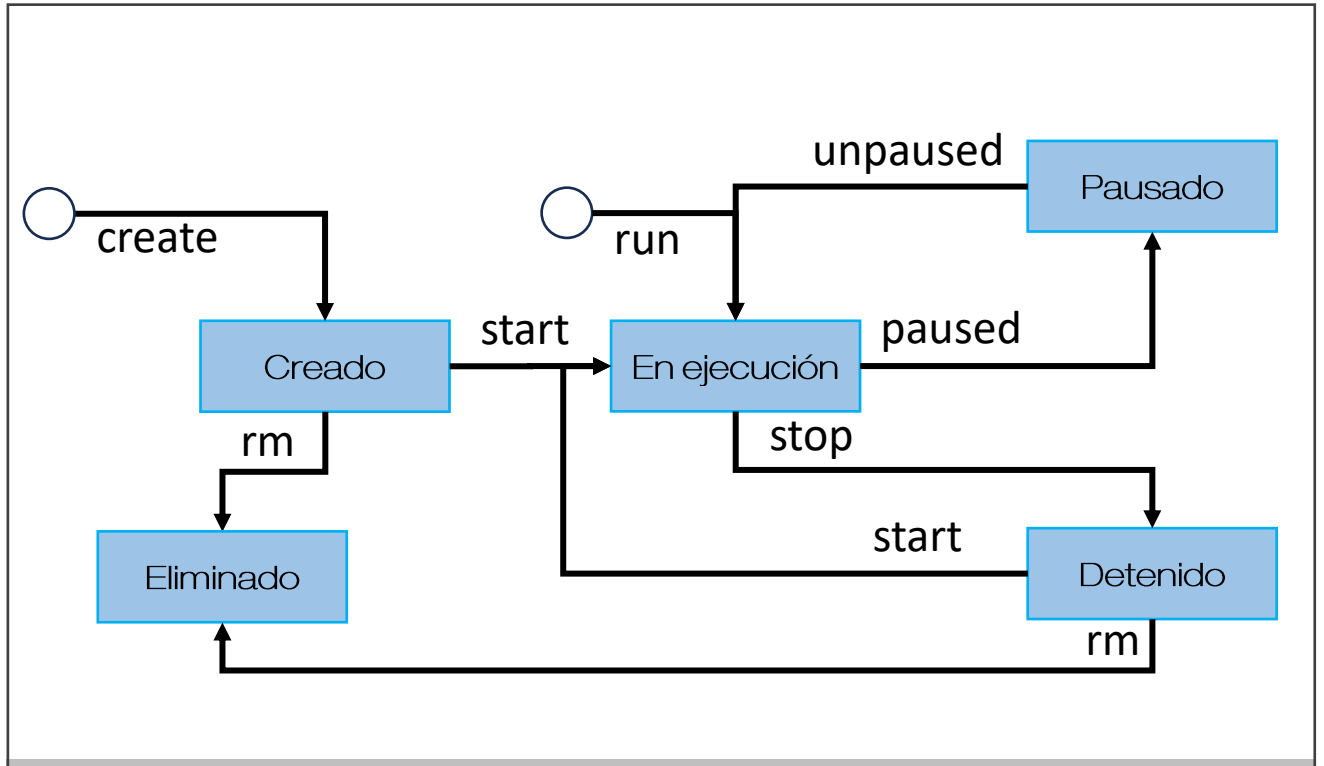
  docker pause container <container-id or container-name>

- Estado salir de pausa

  En este estado, el contenedor continúa con la ejecución.

  docker unpause <container-id or container-name>

  Ejecutar este comando sólo cuando el contenedor esté pausado.



Ciclo de vida de los contenedores

Arquitectura Docker

Ciclo de vida de los contenedores

- Estado detenido

  En el estado detenido, el proceso principal del contenedor es cerrado correctamente.

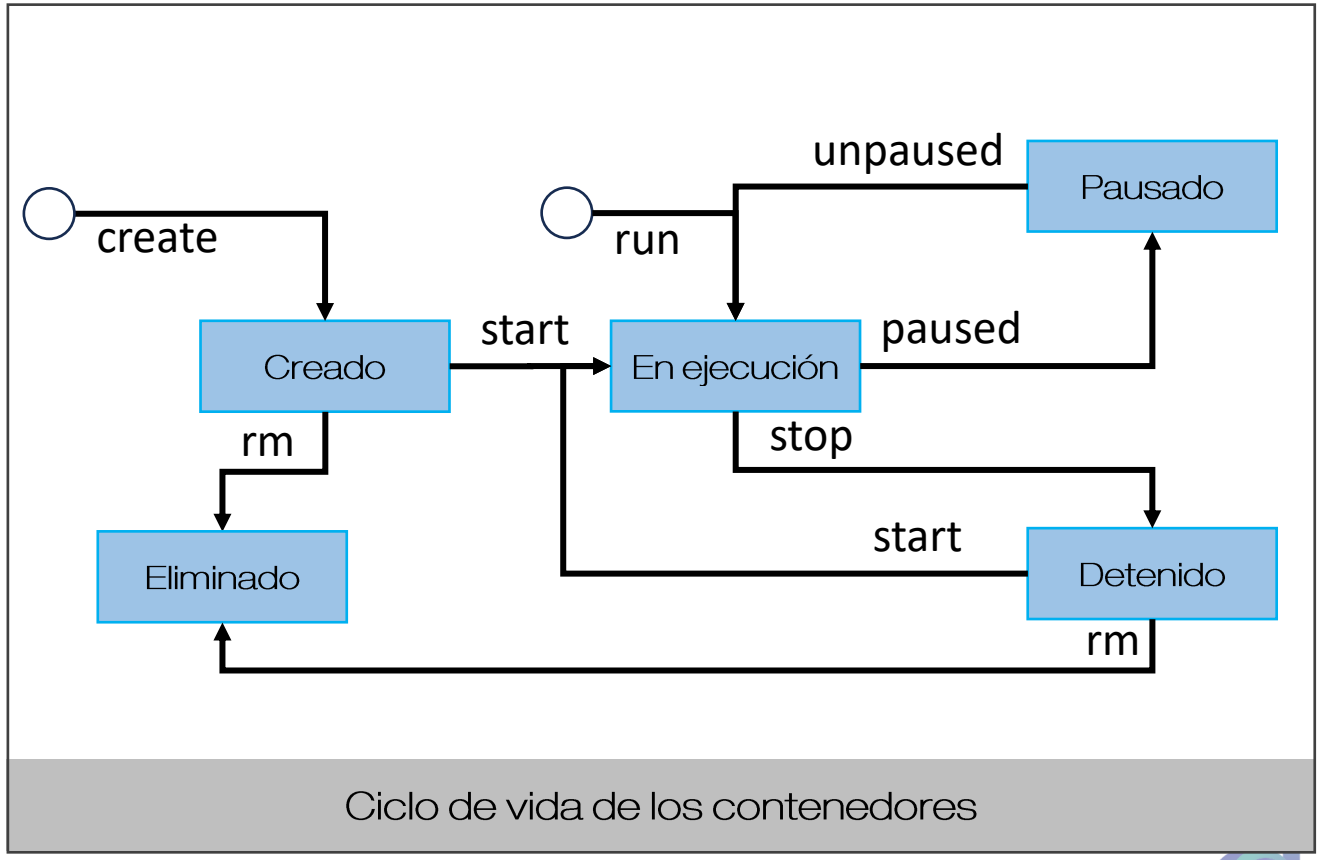
  docker stop <container-id or container-name>

  Reiniciar un contenedor se traducirá en detener el contenedor y luego ejecutarlo otra vez. (stop y run)

- Estado eliminar

  En este estado, el contenedor es detenido abruptamente.

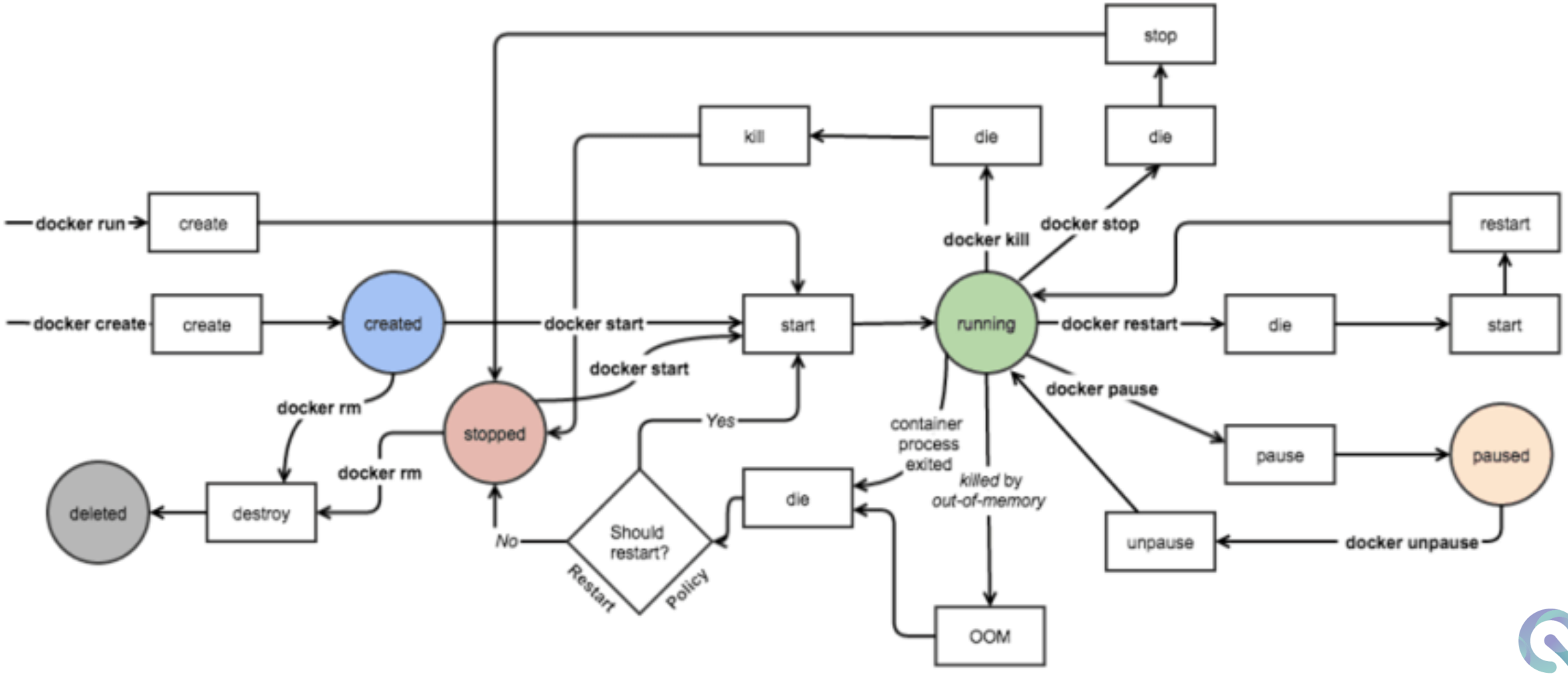
  docker kill <container-id or container-name>



Ciclo de vida de los contenedores

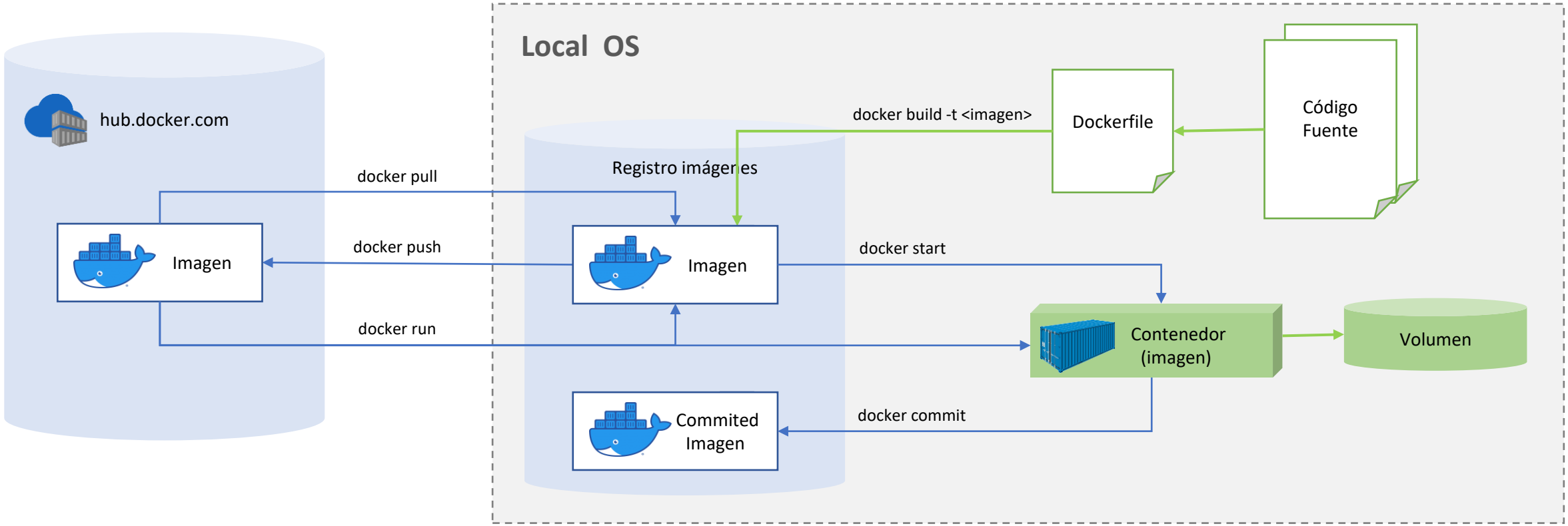
Arquitectura Docker

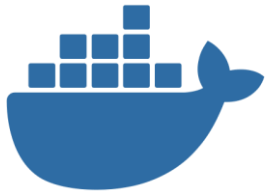
Ciclo de vida completo de los contenedores



Docker

Esquema de comandos principales de Docker





Docker

Contenido

Tecnología de contenedores

Daremos un paseo en el mundo de los contenedores, beneficios y principales retos.

Cliente Docker

Estudiaremos y practicaremos los comandos más comunes que nos da Docker para trabajar con los contenedores.

Registros de contenedores

Revisaremos un conjunto de librerías de imágenes de contenedores y en especial nos centraremos en eDocker Hub.

Herramientas de contenedores

Hay una amplia variedad de herramientas de terceros que pueden ser integradas e incrementar la productividad con contenedores Docker.

Arquitectura Docker

Revisaremos los principales componentes de Docker y cómo se interrelacionan entre sí.

Administración de contenedores

Echaremos un vistazo a herramientas de administración de contenedores y nos centraremos especialmente en Docker Desktop.

Docker Compose

Estudiaremos como podemos definir y ejecutar varias aplicaciones en contenedores de forma fácil y rápida.

Orquestación de contenedores

Analizaremos las herramientas de orquestación de contenedores y veremos en detalle Docker Swarm.

Cliente Docker

Ejemplo: ¡Hola mundo!

La explicación del ejemplo hello-world:

- El comando *docker run* es el responsable de ejecutar contenedores.
- El parámetro *hello-world* es el nombre del repositorio de la imagen, que se encuentra en Docker Hub.
- El comando busca primero la imagen de forma local y si no la ubica la busca en Dockerhub.
- Una vez que se descarga, Docker convierte la imagen en un contenedor en ejecución.
- Con el comando docker images listamos las imágenes locales. El tamaño de la imagen *hello-world* es de apenas13.3kb.

```
C:\tutorials\docker>docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:10d7d58d5ebd2a652f4d93fdd86da8f265f5318c6a73cc5b6a9798ff6d2b2e67
Status: Downloaded newer image for hello-world:latest

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.
    (amd64)
 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
```

```
C:\tutorials\docker>docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    feb5d9fea6a5   7 months ago   13.3kB
```

Cliente Docker

Nombre de la imagen y repositorio

Nombre de una imagen de docker:

- La imagen de docker no tiene un nombre de imagen per se. Tiene un *ID*, un ***repositorio*** y una ***etiqueta*** . Por lo tanto, cuando nos referimos al nombre de una imagen, por lo general nos referimos al Repositorio de la imagen y su Etiqueta.

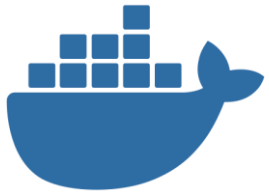
Repositorio de imagen docker[:etiqueta]

Ejemplo

alpine:3.19.0

Repositorio de docker

- Cuando se construye o crea una imagen, se crea un repositorio para esa imagen y la imagen en sí misma.
- Cuando se le agrega la imagen actual al repositorio (sino se específica una etiqueta, se le asigna el valor por defecto de *:latest)*.
- Los Tags permiten tener varias versiones de la misma imagen, por lo que podemos referirnos como myimage:latest, myimage:v1, myimage:v2 al mismo identificador de imagen.



Cliente Docker

Principales comandos

- docker pull — Descarga una imagen desde un registro.
- docker push — Carga una imagen a un registro.
- docker build — Construye una imagen desde un Dockerfile.
- docker images — Lista todas las imágenes de Docker del ordenador.
- docker run — Ejecuta una imagen creando una instancia.
- docker ps — Lista todas las instancias iniciadas y detenidos.
- docker start — Inicia una instancia.
- docker stop — Detiene una instancia.
- docker rm — Elimina una instancia.
- docker rmi — Elimina una imagen.
- docker tag — Crea una etiqueta a una imagen.

Cliente Docker

Comando docker tag

- A efectos de este tutorial el nombre de una imagen está compuesto por

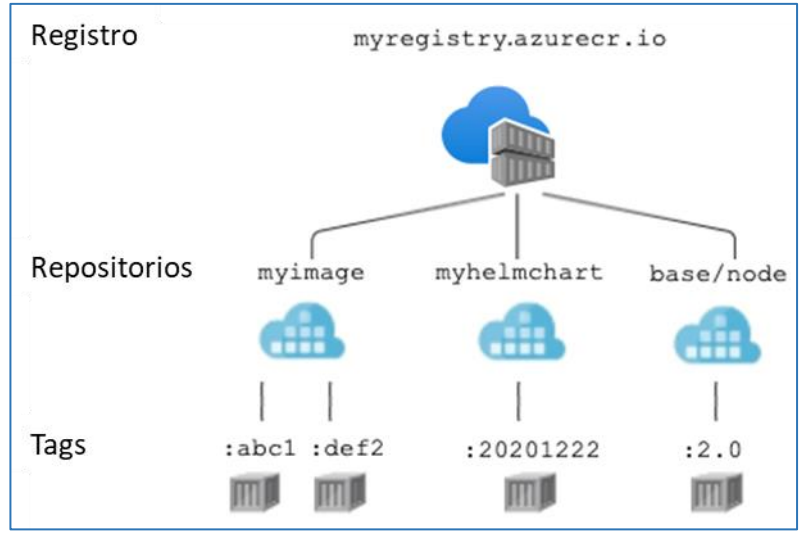
  image_name = [registry_host:[port]/][registry_name/]repository_name[:tag]

- Para etiquetar una imagen usamos el comando *docker tag*

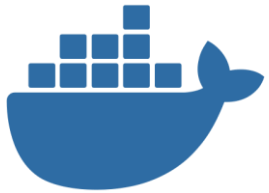
  $ docker tag <source_image_name> <target_image_name>

  Example

  $ docker tag namespace1/docker101tutorial new_namespace/docker101tutorial



- El nombre de una etiqueta debe contener caracteres ASCII válidos, pueden ser minúsculas o mayúsculas, dígitos y separadores. Un separador es un punto, una o dos barras bajas, uno o más guiones. Un tag no puede empezar por un separador y su longitud máximo puede ser de 128 caracteres.



Cliente Docker

Comando docker pull

- Descarga una imagen o un repositorio desde un registro

```
$ docker pull <image_name>
```

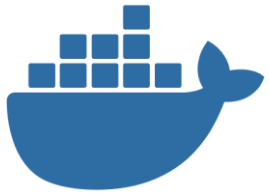
Ejemplo

```
docker pull alpine:3.19.0
```

- Para descargar todas las versiones de una imagen se utiliza la opción *--all-tags* o *-a*

```
$ docker pull --all-tags <image_name>
```

- Con ***Ctrl+C*** Para cancelar el proceso de descarga iniciado por ***docker pull***.



Cliente Docker

Comando docker images

- Para listar todas las imágenes (incluyendo las intermedias) usamos el parámetro *–a/--all*

  $ docker images -a

- Listar imágenes por nombre y tag

  $ docker images <repository_name>:<image_tag>

  $ docker images ls alpine:3.1*

- Listar las imágenes con filtro usamos el parámetro *--filter*

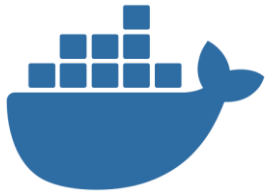
  $ docker images ls --filter=reference='alpin*'

  $ docker images ls --filter=reference='alpin*' --filter=reference='busibox'

- Revisar las opciones del comando docker images

  $ docker images –help

  docker images --format "{{.ID}}: {{.Repository}}"



Cliente Docker

Comando docker push

- Cargar una imagen o un repositorio a un registro

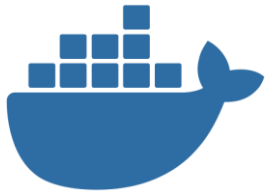
```
$ docker push <repository_name>:<image_tag>
```

Example

```
$docker push new_namespace/docker101tutorial
```

- Carga todas las imágenes de un repositorio con la opción *--all-tags* o *-a*

```
$ docker push --all-tags <repository_name>
```



Cliente Docker

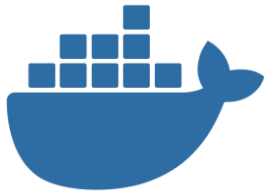
Comando docker run

- Crea una nueva capa de escritura en la imagen de un contenedor y lo inicia utilizando el comando específicado.

```
$ docker run [options] <image_name> [command] [args]
```

- Entre las opciones más comunes encontramos.
  - *detach/d* : Ejecuta el contenedor en segundo plano.
  - *env/e* : Establece variables de entorno en el contenedor.
  - *expose* : Expone un puerto o un rango de puertos del contenedor
  - *name* : Asigna un nombre al contenedor
  - *rm* : Elimina el contenedor apenas termina.
  - *tty/t* : Habilita un pseudo-TTY (Tele TYpewriter)
  - *Workdir/w* : Establece el directorio de trabajo dentro del contenedor

```
$ docker run --name test -it alpine:3.7
```



Cliente Docker

Comandos docker start/stop – pause/unpause

- start: Inicia uno o más contenedores.

```
$ docker start [options] <container> [<container>…]
```

- stop: Detiene uno o más contenedores [SIGTERM – SIGKILL]

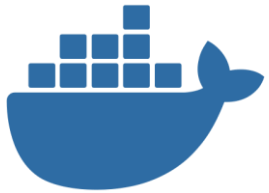
```
$ docker stop [options] <container> [<container>…]
```

- pause: Pausa uno o más contenedores [SIGSTOP]

```
$ docker pause <container> [<container>…]
```

- unpause: Reanuda uno o más contenedores [SIGCONT]

```
$ docker unpause <container> [<container>…]
```



Cliente Docker

Comando docker commit

- Crea una nueva imagen a partir de los cambios de un contenedor.

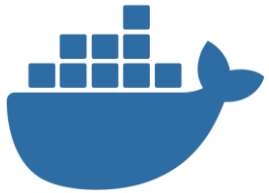
```
$ docker commit [options] <container> [repository:[tag]]
```

- Ejemplo 1

```
$ docker commit container-name new-image-name:example
```

- Ejemplo 2

```
$ docker inspect -f "{{ .Config.Env }}" <container-id>
$ docker commit --change "ENV DEBUG=true" <container-id> new-image-name:example
$ docker inspect -f "{{ .Config.Env }}" <new container-id>
```



Cliente Docker

Comando docker inspect

- Provee información detallada de un objecto construido por Docker.

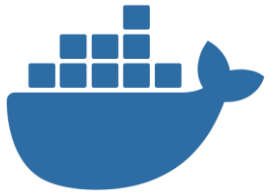
```
$ docker inspect [options] <container> [container…]
```

- Con la opción type se puede especificar el tipo de objeto.

```
--type container|image|node|network|secret|service|volume|task|plugin
```

- Ejemplo

```
$ docker inspect --size <container>
$ docker inspect --size database -f '{{ .SizeRootFs }}'
$ docker inspect --size database -f '{{ .SizeRw }}’
```



Cliente Docker

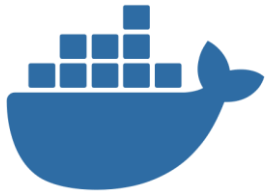
Comando docker rmi

- Elimina una o más imágenes usando su ID (corto o largo), su etiqueta o su digest. Si una imagen tiene más de etiqueta que la referencia, debe de eliminarlas todas antes que la imagen sea eliminada.

```
$ docker rmi <repository_name>:<image_tag>|<image_id>|<digest>
```

- Esto no elimina imágenes de un registro.
- No puede eliminar una imagen de un contenedor en ejecución a menos que utilice la opción -f
- Con la opción *-f* y el ID (corto o largo), se remueve todas las etiquetas de la imagen y se elimina.

```
$ docker rmi -f <image_id>
```



Cliente Docker

Comando docker rmi

$ docker images

```
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
test1                     latest              fd484f19954f        23 seconds ago      7 B (virtual 4.964 MB)
test                      latest              fd484f19954f        23 seconds ago      7 B (virtual 4.964 MB)
test2                     latest              fd484f19954f        23 seconds ago      7 B (virtual 4.964 MB)
```

$ docker rmi fd484f19954f

```
Error: Conflict, cannot delete image fd484f19954f because it is tagged in multiple repositories, use -f to force
2013/12/11 05:47:16 Error: failed to remove one or more images
```

$ docker rmi test1:latest

```
Untagged: test1:latest
```

$ docker rmi test2:latest

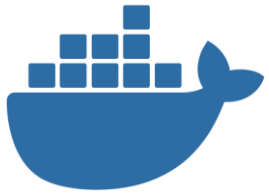
```
Untagged: test2:latest
```

$ docker images

```
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
test                      latest              fd484f19954f        23 seconds ago      7 B (virtual 4.964 MB)
```

$ docker rmi test:latest

```
Untagged: test:latest
Deleted: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8
```



Cliente Docker

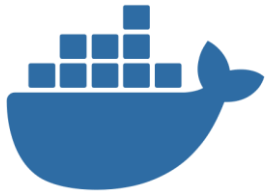
Comando docker rm

- Elimina una o más contenedores usando su ID (corto o largo) o su nombre

```
$ docker rm <container_id>|<container_name>
```

- Con la opción *-f* y el ID (corto o largo), se remueve todos los contenedores forzando a terminar aquellos que se encuentran en ejecución.

```
$ docker rm -f <container_id>|<container_name>
```



Cliente Docker

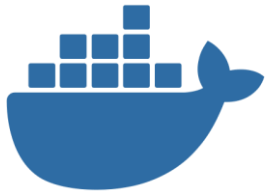
Comando docker ps

- Lista los contenedores, por defecto solo los que se encuentran en ejecución. Con la opción *-a* o *--all*, lista todos los contenedores.

```
$ docker ps --all
$ docker container ps --all
```

- Con la opción *--filter* lista los contenedores de acuerdo a las condiciones dadas. Las condiciones más comunes son:
  - *status*: con un valor de *created, restarting,running, removing, paused, exited or dead*.
  - *health*: con valor de *starting, healthy, unhealthy* o *none*.

```
$ docker ps --filter status=running
$ docker ps container ls --filter “name=local*”
```



Cliente Docker

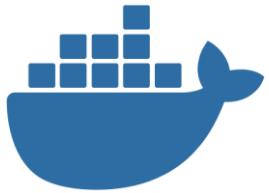
Guardar contenedores o imágenes

- Exporta el contenido del sistema de archivos de un contenedor.
  La opción *--output* o *-o* se utiliza para dar nombre al archivo de salida.

```
$ docker export --output=<filename.tar> <container_id|name>
```

- Guarda una o más imágenes en un archivo *tar*. La opción *--output* o *-o* se utiliza para dar nombre al archivo de salida.

```
$ docker save -o <filename.tar> <image_id|name>
```



Cliente Docker

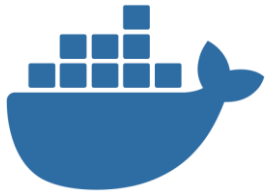
Restaurar contenedores o imágenes

- Importa el contenido del sistema de archivos guardado en un archivo *tar* de un contenedor a una imagen.

```
$ docker import <from_file|url> <image_name>
```

- Carga las imágenes en un archivo *tar*. Se utiliza la opción *--input* o *-i* para indicar el nombre del archivo *tar* desde el cual se va cargar las imágenes.

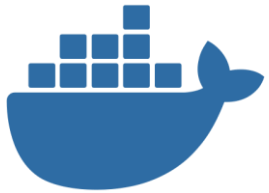
```
$ docker load -i <filename.tar>
```



Cliente Docker

Práctica 1

- Ejecutar la imagen de nginx.
- Exportar el contenedor de nginx con el nombre de nginx.tar
- Importar la imagen del archivo nginx.tar como mynginx
- Comprobar la correcta importación de la imagen
- Eliminar la imagen mynginx
- Eliminar el archivo nginx.tar
- Detener el contenedor nginx
- Eliminar el contenedor nginx
- Eliminar la imagen nginx



Cliente Docker

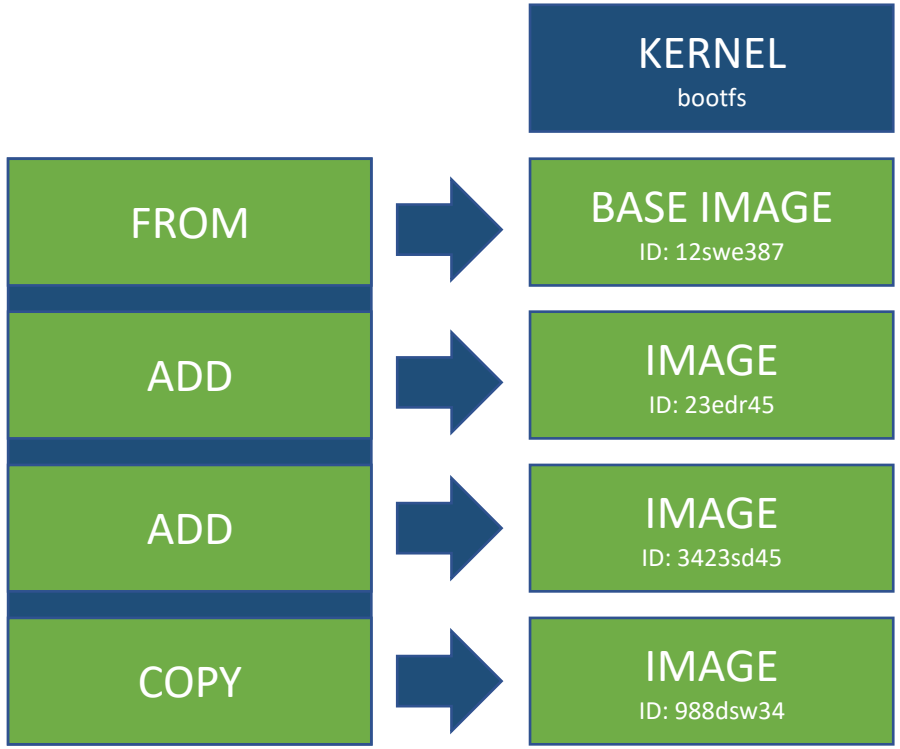
Práctica 2

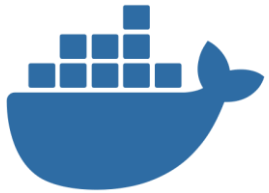
- Descargar la imagen de nginx.
- Guardar la imagen de nginx con el nombre de nginx.tar
- Cargar la imagen del archivo nginx.tar como mynginx
- Comprobar la correcta importación de la imagen
- Eliminar la imagen mynginx
- Eliminar el archivo nginx.tar
- Eliminar la imagen descargada de nginx

Dockerfile

¿Qué es?

- Es un archivo de texto que describe los pasos para crear una imagen Docker a partir de una imagen base sobre la que se realiza cambios.
- Con el comando **docker build**, se ejecuta las instrucciones del archivo **Dockerfile**. Durante la **ejecución de cada paso del archivo se agrega una capa a la imagen base**.
- El archivo .**dockerignore**, sirve para excluir archivos y directorios del contexto de ejecución del comando Docker build.

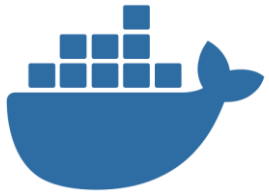




Dockerfile

Comandos

- FROM — Específica la imagen padre/base.
- WORKDIR — Establece el directorio de trabajo para los siguientes comandos.
- RUN — Instala una aplicación o paquetes necesarios para el contenedor.
- COPY — Copia archivos o directorios desde una ubicación específica.
- ADD — Similar a COPY, pero capaz de procesar urls remotos y descomprimir paquetes.
- ENTRYPOINT — Comando que se ejecuta al iniciar el contenedor. Por defecto /bin/sh -c
- CMD — Argumentos pasados a ENTRYPOINT. Si no hay ENTRYPOINT, se ejecutarán los comandos por el contenedor.
- EXPOSE — Define los puertos a través de los cuales se acceden al contenedor.
- LABEL — Agrega metadatos al contenedor.
- # — Marca la línea como comentarios.



Dockerfile

Ejemplo

- Un archivo mínimo de Dockerfile

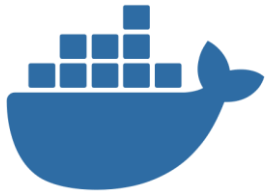
```
FROM alpine
CMD [“echo”, “Hello world!”]
```

- Construirlo y ejecutarlo

```
$ docker build –t hello .
$ docker run --rm hello
```

- La salida es

```
Hello world!
```



Docker

Comando build

- Construye una imagen desde un archivo Dockerfile y un contexto. El contexto del comando build es un conjunto de archivos ubicados en la ruta o url.

  $ docker build [options] <ruta_dockerfile>|<url>

- Cuando la url apunta a un repositorio Git, el repositorio actúa como el contexto del comando build. En el siguiente ejemplo, el fragmento indica la rama container y la carpeta docker como contexto.

  $ docker build https://github.com/docker/rootfs.git#container:docker

- Para construir una imagen con una o más etiquetas se utiliza la opción *-t* o *--tag*.

  $ docker build –t repo1/micro1:v1 –t repo2/micro1:v1 .

- Para asignar valores a los argumentos se utiliza la opción --build-arg.

  $ docker build –-build-arg DEST_FOLDER=my_folder .

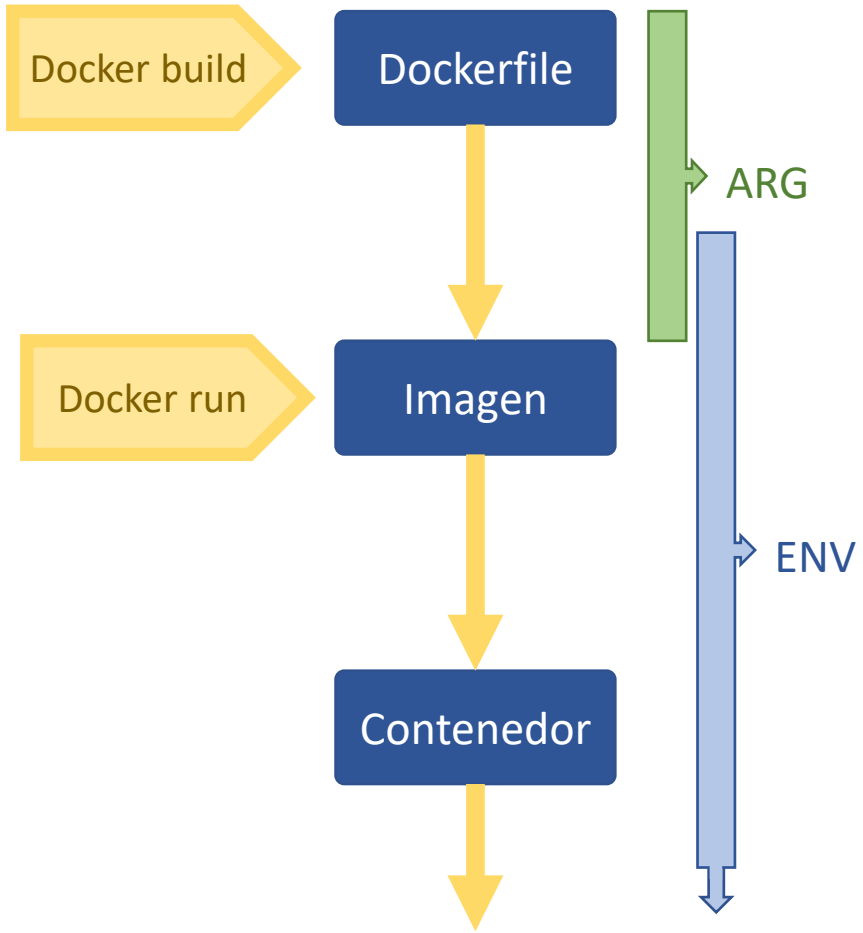
Dockerfile

Variables de entorno

- ARG Define argumentos dentro del dockerfile que están disponibles durante la construcción de la imagen. Sin embargo, el contenedor en ejecución no puede acceder a estos valores.

  Si se definen varios ARG (sin un valor por defecto) pero no se especifican en el comando build, saltará un mensaje de error.

- ENV Define variables de entorno que están disponibles durante la construcción de la imagen desde el momento que son definidas.
  A diferencia de ARG, el contenedor en ejecución sí puede tener acceso a estos valores.

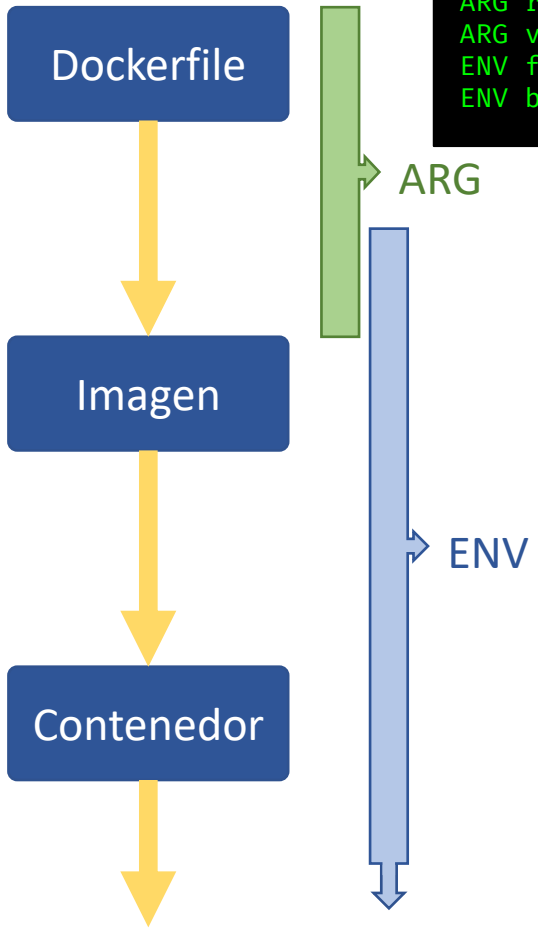


Dockerfile

Variables de entorno – Ejemplo

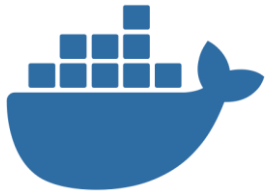
```
$ docker build --build-arg required_var=value
```

```
$ docker run –e “foo=rewrite_value”
$ docker run –e foo  # pasa el valor del host.
$ docker run --env-file=file_name # pasa desde un archivo
```




```
ARG required_var       # Se espera este valor
ARG var_name=def_value # ARG con valor por defecto
ENV foo=other_value    # ENV con valor por defecto
ENV bar=${var_name}    # ENV con valor del ARG var_name
```

Cuando se ejecutan los comandos ENTRYPOINT, CMD y RUN en formato EXEC, los valores de ARG y ENV no se reemplazan.



Dockerfile

Algunos ejemplos: Git

- Crea una imagen con Git

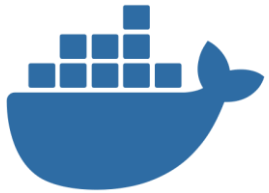
```
FROM alpine:3.19.0
RUN apk update
RUN apk add git
```

- Crea una imagen con un archivo en su estructura

```
FROM alpine:3.19.0
RUN apk update
ADD http://www.vlsitechnology.org/pharosc_8.4.tar.gz .
```

- Construir con la etiqueta *app_git:v1*, ejecutar, entrar y verificar la existencia del archivo.

```
$ docker run --rm app_git:v1 ls
```



Dockerfile

Algunos ejemplos: Copy

- Crear un archivo index.html con el contenido

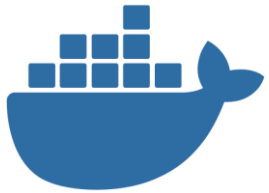
```
"Bienvenidos al laboratorio de prueba"
```

- Crear un archivo dockerfile con el contenido

```
FROM nginx:alpine
COPY index.html /usr/share/nginx/html/
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

- Construir con la etiqueta *app_copy:v1*, ejecutar y navegar a la página html.

```
$ docker run --rm -d -p 80:80  app_copy:v1
```



Dockerfile

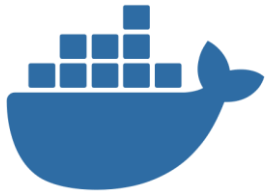
Algunos ejemplos: Entrypoint

- Crear un archivo dockerfile con el contenido

```
FROM alpine:3.19.0
LABEL maintainer="datahack"
ENTRYPOINT ["/bin/echo", "Hola, tu ENTRYPOINT en formato EXEC !"]
```

- Construir y ejecutar con la etiqueta *app_entrypoint:v1* y ver el resultado.
- Ejecutar el siguiente comando

```
$ docker run --entrypoint "/bin/echo" app_entrypoint:v1 "Hola, desde consola!"
```



Dockerfile

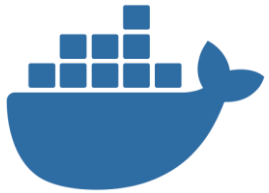
Algunos ejemplos: Workdir

- Crear un archivo dockerfile con el contenido

```
FROM alpine:3.17.1
WORKDIR /opt
RUN echo "Welcome to Docker Labs" > opt.txt
WORKDIR folder1
RUN echo "Welcome to Docker Labs" > folder1.txt
WORKDIR folder2
RUN echo "Welcome to Docker Labs" > folder2.txt
```

- Construir la imagen con la etiqueta *app_workdir:v1*.
- Ejecutar con el comando *pwd*

```
$ docker run --rm app_workdir:v1 pwd
```



Dockerfile

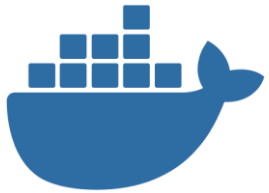
Algunos ejemplos: Python

- Crear un archivo hello.py con el siguiente contenido

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello():
        return "welcome to Dockerlabs!! successfully done !!"
if __name__ == "__main__":
        app.run()
```

- Y el archivo dockerfile con el contenido

```
FROM python:3.5
RUN apt-get update
RUN pip install Flask
ADD . /opt/webapp/
WORKDIR /opt/webapp
ENV FLASK_APP=hello.py
EXPOSE 5000
CMD ["flask", "run", "--host=0.0.0.0"]
```



Dockerfile

Algunos ejemplos: Python (cont.)

- Construir la imagen con la etiqueta *app_python:v1*

```
$ docker build -t app_python:v1 .
```

- Ejecutar la imagen especificando el puerto

```
$ docker run --rm -p 5000:5000 -d app_python:v1
```

- Navegar a la url *http://localhost:5000/*

Dockerfile

Algunos ejemplos: Args

- Crear un archivo Dockerfile con el siguiente contenido

```
FROM alpine:3.17.1
ARG A_USER=amigo
RUN echo “Bienvenido $A_USER al mundo de Docker!" > message.txt
CMD cat message.txt
```

- Construir la imagen con la etiqueta *app_args:v1 y ejecutar*.
- Construir la imagen con la etiqueta *app_args:v2* pasando como valor *Ed* en el argumento *A_USER.*

```
$ docker build -t app_args:v2 --build-arg A_USER=Ed .
```

- Ejecutar la imagen.

Dockerfile

Algunos ejemplos: Entrypoint y cmd

- Crear un archivo Dockerfile con el siguiente contenido, construir la imagen con el nombre *app_entry:v1*, ejecutar y ver la salida.

```
FROM alpine:3.19.0
ENTRYPOINT [“echo”, “Hola“]
CMD [“Command Exec”]
```

- Eliminar el contenedor creado en la línea anterior y ejecutar la siguiente línea

```
$ docker run --rm app_entry:v1 Mundo
```

- Agrega la siguiente línea al dockerfile, vuelve a construir la imagen, ejecutar y ver la salida.

```
CMD “Command shell”
```

Dockerfile

Algunos ejemplos: Entrypoint, cmd, push y pull

- Crear un archivo Dockerfile con el siguiente contenido, construir la imagen, ejecutar y ver la salida.

```
FROM alpine:3.19.0
ENTRYPOINT [“echo”, “Hola, soy Juan, te doy la bienvenida”]
CMD [ “amigo” ]
```

- Subir la imagen a un repositorio en una cuenta de Docker Hub.
- Descargar la imagen de un compañero desde Docker Hub y ejecutar la imagen pasando su nombre como parámetro.
- En el Dockerfile anterior, modificar la instrucción ENTRYPOINT por CMD pasando su nombre como valor de la variable de entorno E_USER, generar nuevamente la imagen y comprobar la salida.

Dockerfile

Algunos ejemplos

- Crear un archivo dockerfile con el contenido

```
FROM alpine:3.17.1
ENV DIRPATH /myfolder
WORKDIR $DIRPATH
```

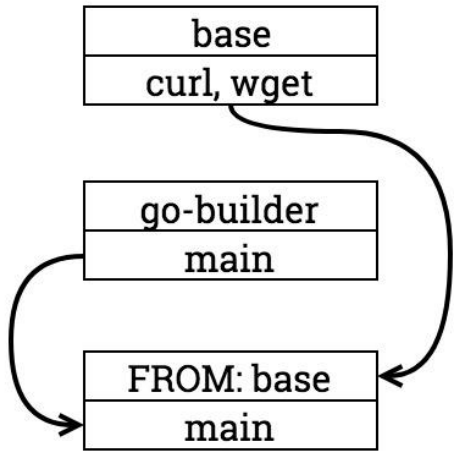
- Construir, con la etiqueta *workdir:v2*.
- Ejecutar con el comando *pwd.*
- Construir, con la etiqueta *workdir:v3* adaptando el archivo Dockerfile para que el valor de la variable *DIRPATH* también pueda obtener el valor desde un argumento.
- Ejecutar con el comando *pwd* para visualizar el directorio actual.

Docker

Dockerfile – multi-stages

- Uno de los aspectos más importantes al generar una imagen es mantener el tamaño de la misma lo más pequeño posible. Cada instrucción RUN, COPY y ADD, agrega una nueva capa a la imagen.
- Hasta hace poco, se utilizaba dos archivos dockerfile, uno para el uso de desarrollo y otro más liviano para producción. Este patrón de generación de imágenes no era óptimo.
- Los archivos dockerfile multi-stage, simplifican bastante esta tarea con el uso de múltiples sentencias FROM en el mismo archivo.
- Cada sentencia FROM inicia una nueva fase o etapa de construcción de construcción (stage). Se puede copiar selectivamente partes de un stage a otro, dejando atrás todo lo que no se quiere en la imagen final.
- Los stages por defecto no tiene un nombre y se puede hacer referencia a ellos por el número entero que le corresponde, empezando en cero. Sin embargo, se puede dar un nombre a cada stage agregando AS <name> a la sentencia FROM.

```
1   FROM alpine AS base
2   RUN apk add --no-cache curl wget
3
4   FROM golang:1.9.2 AS go-builder
5   WORKDIR /go
6   COPY *.go /go/
7   RUN CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build -o main .
8
9   FROM base
10  COPY --from=go-builder /go/main /main
11  CMD ["/main"]
```



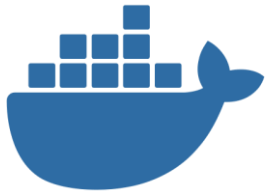
Docker

Dockerfile – multi-stages: Ejemplo

Este ejemplo combina varios stages para la compilación del código fuente hasta los binarios finales que se desplegarán en producción.

```
FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base

WORKDIR /app

EXPOSE 5080

EXPOSE 5443


ENV ASPNETCORE_URLS=http://+:5080


RUN adduser -u 5678 --disabled-password --gecos "" appuser && chown -R appuser /app

USER appuser
```

```
FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build

WORKDIR /src

COPY ["TodoApi.csproj", "./"]

RUN dotnet restore "TodoApi.csproj"

COPY . .

WORKDIR "/src/."

RUN dotnet build "TodoApi.csproj" -c Release -o /app/build


FROM build AS publish

RUN dotnet publish "TodoApi.csproj" -c Release -o /app/publish /p:UseAppHost=false


FROM base AS final

WORKDIR /app

COPY --from=publish /app/publish .

ENTRYPOINT ["dotnet", "TodoApi.dll"]
```



Docker

Configurar límites de los recursos

- Para restringir la cantidad memoria (límite duro) usamos el parámetro *-m* o *--memory*

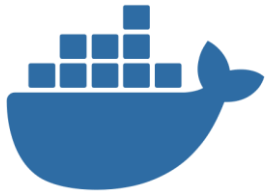
```
$ docker run -m 512m nginx
```

- Para hacer una reserva de memoria (límite suave) *--memory-reservation*

```
$ docker run -m 512m --memory-reservation=256m nginx
```

- Podemos verificar el uso de recursos con el comando *docker stats*

```
$ docker stats
```



Docker

Configurar límites de los recursos

- Para restringir la cantidad de cpu (límite duro) usamos el parámetro *--cpus*

```
$ docker run --cpus=“.5” nginx
```

- Para hacer una priorización de cpu (límite suave) *--cpu-shares*

```
$ docker run --cpus=2 --cpu-shares=2000 nginx
```

Docker

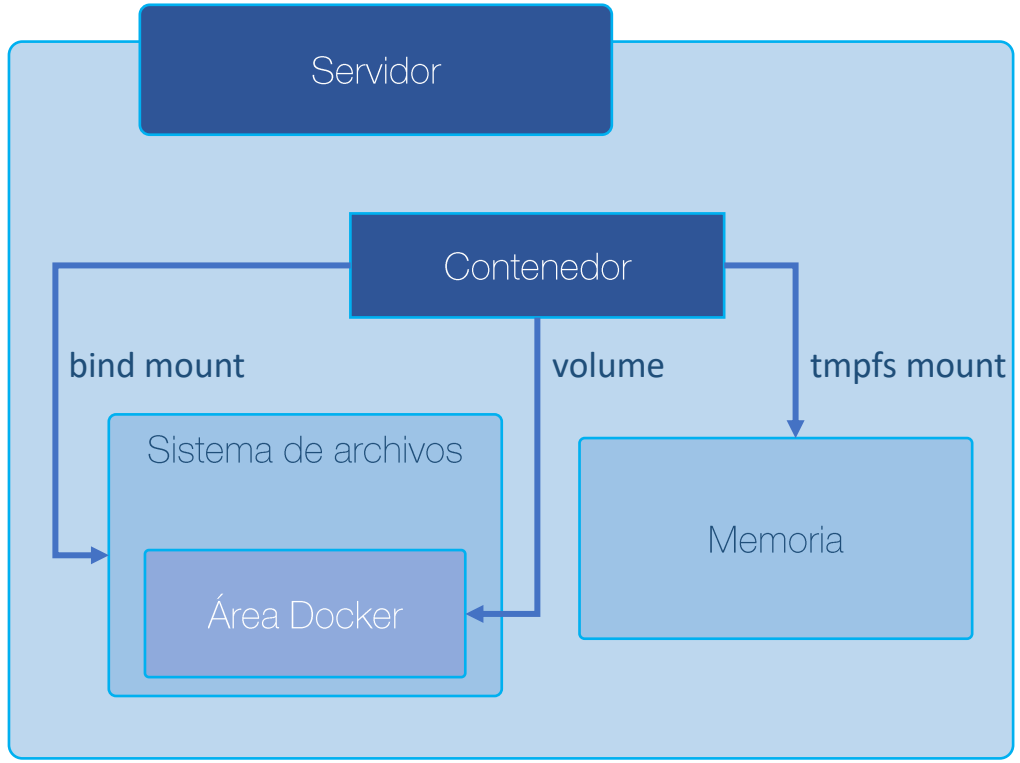
Volúmenes - Datos en docker

Por defecto, todos los archivos creados en un contenedor **se escriben en una capa de escritura**. Esto significa que los datos **no persisten** cuando se termina la ejecución del contenedor.

Docker presenta dos opciones para persistir los archivos, incluso cuando se detiene:

- Volúmenes
- Montar archivos con “bind”

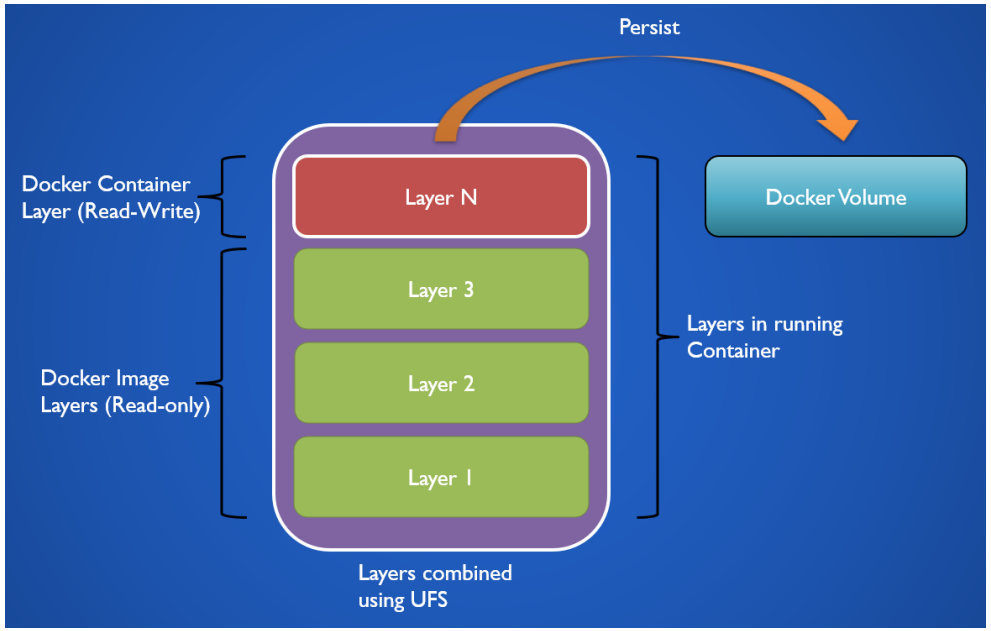
Docker también permite guardar archivos en memoria. Estos archivos no persisten.

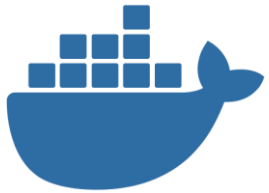


Docker

Volúmenes - En detalle

- Los volúmenes son el mecanismo preferido para la persistencia de datos generado y usado por los contenedores de Docker.
- Son fáciles de sacar copias de seguridad o de migrar.
- Se pueden administrar con los comandos de Docker CLI.
- Se pueden usar tanto en Windows como en Linux.
- Se pueden compartir de forma segura entre varios contenedores.
- Hay drivers que permiten almacenar en servidores remotos, en proveedores en la nube, encriptar el contenido de los contenedores o agregar otra funcionalidad.
- Los volúmenes nuevos pueden estar "pre" cargados con archivos.
- Los volúmenes no incrementan el tamaño de las imágenes del contenedor.





Docker

Volúmenes - Administración

- Crear un volumen

```
$ docker volume create my-volume
```

- Para listar los volúmenes

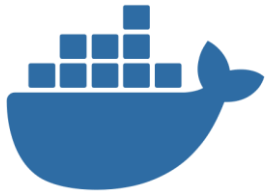
```
$ docker volumen ls
```

- Para ver los detalles de un volumen

```
$ docker volumen inspect my-volume
```

- Para ver eliminar un volumen

```
$ docker volumen rm my-volume
```



Docker

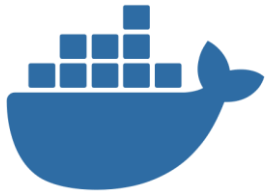
Volúmenes

- Crear un volumen

```
$ docker run -d --name test --mount source=my-volume,target=/app nginx:latest
```

- Algunas opciones de *--mount*

Opción	Req.	Descripción
type=volume		Monta un volumen administrado dentro del contenedor.
src / source		Especifica el nombre de un volumen. Si no existe, lo crea. Si no se especifica este parámetro, se asigna un volumen con un nombre aleatorio que tiene el mismo ciclo de vida que su contenedor y es destruido cuando el contenedor se destruye.
dst /destination / target	Sí	La ruta dentro del contenedor. Por ejemplo: /mydirectories/work/. Si el directorio no existe, Docker lo crea automáticamente.
readonly / ro		Indica si monta como sólo lectura o como lectura-escritura (por defecto)



Docker

Volúmenes - Describir las siguientes instrucciones

```
$ docker volume create my-volume
$ docker volume inspect my-volume
$ docker run -it -v my-volume:/data --name my-container selaworkshops/busybox:latest
$ cd /data
$ echo “prueba de persistencia” > prueba.txt
$ ls
(en otro terminal)
$ docker run -it -v my-volume:/data --name my-container-2 selaworkshops/busybox:latest
$ cd /data
$ ls
$ exit  (de ambos contenedores)
$ docker rm -f my-container my-container-2
$ docker ps -a
$ docker run -it -v my-volume:/data --name new-container selaworkshops/busybox:latest
$ cd data
$ ls
$ exit
$ docker rm -f new-container
```

Dockerfile

Práctica

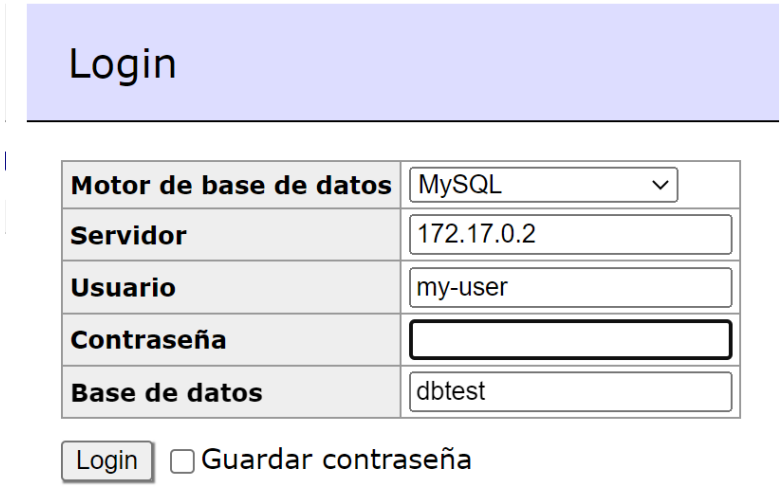
- Descargar la imagen de MySQL desde Docker Hub. (de preferencia no copiar este texto porque algunos caracteres no corresponden a los que se visualizan como los guiones o espacios)

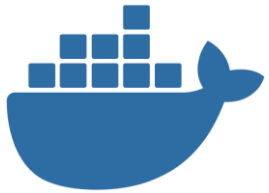
```
$ docker run --name mysql -e MYSQL_ROOT_PASSWORD=my-root-pwd -e MYSQL_USER=my-user -e MYSQL_PASSWORD=my-sqlpwd -e MYSQL_DATABASE=dbtest -d -p 3306:3306 mysql
```

- Descargar la imagen de adminer Docker Hub.

```
$ docker run --name adminer -d -p 8080:8080 adminer
```

- Conectar a mysql desde adminer *(http://localhost:8080 ó http://<local-ip:8080>).* Probar crear una tabla e insertar registros.
- Eliminar los contenedores, volver a ejecutar las instrucciones anteriores y verificar si existe la tabla y los registros insertados.
- Volver a crear los contenedores, pero esta vez crear volúmenes para el archivo de configuración de MySQL (/etc/mysql/conf) y para el directorio de base de MySQL (/var/lib/mysql).





Dockerfile

Práctica

- Con el siguiente comando los archivos en el directorio source/target estarán también disponibles en el directorio app del contenedor

```
$ docker run -d -it --name devtest --mount type=bind, source="$(pwd)"/target, target=/app nginx:latest
```

En Windows

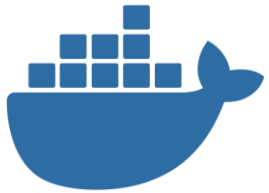
```
$ docker run --rm -d -it --name devtest --mount "type=bind,source=$pwd/target,target=/app" nginx:latest
```

- Ejecutar el siguiente comando y revisar la sección “Mount”

```
$ docker inspect devtest
```

- Detener el cometedor y eliminarlo después

```
$ docker stop devtest
$ docker rm devtest
```



Dockerfile

Práctica

- En este caso ejecutaremos el mismo comando que en la diapositiva anterior con la diferencia de que esta vez el volumen será sólo de lectura

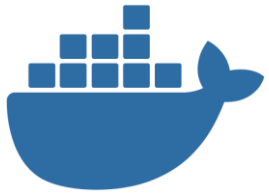
```
$ docker run -d -it --name devtest --mount type=bind, source="$(pwd)"/target, target=/app, readonly nginx:latest
```

- Ejecutar el siguiente comando y revisar la sección “Mount”

```
$ docker inspect devtest
```

- Detener el cometedor y eliminarlo después

```
$ docker stop devtest
$ docker rm devtest
```



Dockerfile

Práctica

- El siguiente ejemplo monta un volumen temporal en memoria y lo monta en la carpeta app del contenedor nginx

```
$ docker run --rm -d -it --name tmptest --mount type=tmpfs,destination=/app nginx
```

- Alternativamente puede ejecutar el siguiente comando

```
$ docker run –d –it --name tmptest –tmpfs /app nginx:latest
```

- Ejecutar el siguiente comando

```
$ docker inspect tmptest --format '{{ json .Mounts }}’
```

- Detener el cometedor y eliminarlo después

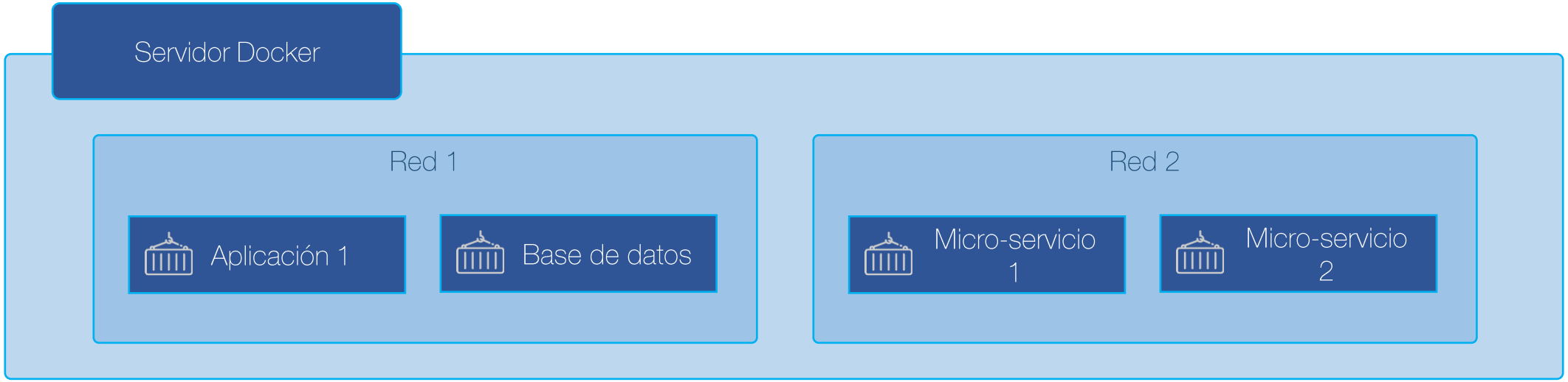
```
$ docker stop tmptest
$ docker rm tmptest
```

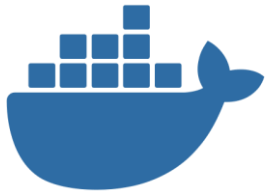
Docker

Redes

Con Docker no es necesario configurar una red porque Docker **siempre crea una red por defecto** donde todos los contenedores que se crean se registran en ella.

Para escenarios más complejos donde se quiera aislar unos contenedores de otros, se necesitaría configurar redes.





Docker

Redes

Las redes de docker permiten a un usuario enlazar a un contenedor a una red que requiera y de esta forma proveer un aislamiento a los contenedores de docker.

Algunos de los beneficios de las redes de docker son.

- Comparten el mismo sistema operativo y mantienen a los contenedores en un entorno aislado.
- Requiere pocas instancias de un SO para ejecutar la carga de trabajo.
- Ayuda al rápido despliegue del software.
- Ayuda a la portabilidad de las aplicaciones.

Docker

Redes – Modelo de redes de contenedores

Network sandbox.

- Es un sandbox (entorno aislado) que mantiene la configuración de red de los contenedores.
- Se crea cuando el usuario requiere generar un *endpoint* en la red.

Endpoints.

- Representa la configuración de red de un contenedor (IP-address, MAC-address, etc.)
- Establece la conectividad para los servicios del contenedor (dentro de la red(con otros servicios.
- Ayuda a proveer la conectividad entre los *endpoints* que pertenecen a la misma red y los aísla del resto.

Docker engine.

- Es el mismo motor instalado en tu ordenador para construir y ejecutar los contenedores usando los componentes y servicios de Docker.
- Su función es administrar la red con múltiples drivers.
- Proveer un punto de entrada a *libnetwork* para mantener las redes. *libnetwork* soporta múltiples controladores virtuales en modelo de red de contenedores robusto que provee los niveles de abstracción de red.

Docker

Redes – Controladores

Bridge.

- Es la red privada por defecto que se crea en el servidor.
- Los contenedores enlazados a esta red tienen una IP interna a través de la cual se pueden comunicar unos con otros fácilmente.
- Este controlador es usado ampliamente para ejecutar contenedores standalone.

Host.

- Es una red pública.
- Utiliza la IP del servidor y su espacio de puertos TPC para mostrar los servicios dentro del contenedor.
- Deshabilita el aislamiento de red entre el contenedor y el servidor, lo cual significa que con este controlador de red, el usuario no podrá ejecutar múltiples contenedores en el mismo servidor.

None.

- Con este controlador, los contenedores no tendrán acceso a ningún acceso a redes externas ni serán capaces de comunicarse con otros contenedores.

Docker

Redes – Controladores

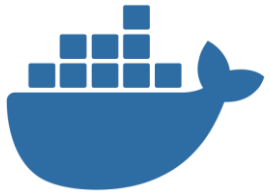
Overlay.

- Este se utiliza para crear redes privadas internas dentro de los nodos de Docker en un clúster *Docker Swarm*.

  Docker Swarm es un servicio que facilita a los equipos de desarrollo construir y administrar un clúster de nodos *Swarn* dentro de la plataforma Docker.

Macvlan.

- Simplifica los procesos de comunicación entre los contenedores.
- Asigna una dirección MAC al contenedor de docker. Con esta dirección el servidor Docker enruta el tráfico de red a un *router*.
- Es la opción sugerida cuando un usuario desea conectar directamente al contenedor con una red física en lugar de conectarlo al servidor Docker donde se hospeda.



Docker

Redes – Comandos básicos

- Listar todas las redes disponibles en el servidor Docker.

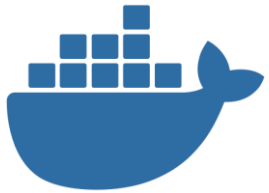
```
$ docker network ls
```

- Conectar un contenedor en ejecución a una red.

```
$ docker network connect multi-host-network container
```

- Especificar una dirección IP que desea asignar a un contenedor.

```
$ docker network connect --ip 10.10.36.122 multi-host-network container
```



Docker

Redes – Comandos básicos

- Desconectar un contenedor de una red.

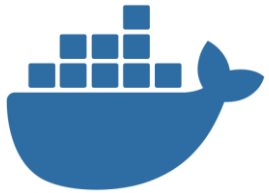
```
$ docker network disconnect multi-host-network container
```

- Eliminar una red.

```
$ docker network rm network_name
```

- Eliminar múltiples redes.

```
$ docker network rm 7478d6eeac29 network_name
```



Docker

Redes – Práctica

- Crear una red con un controlador bridge.

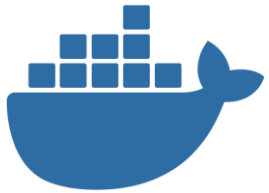
```
$ docker network create -d bridge my-bridge-network
```

- Ejecutar un contenedor enlazado a la red creada.

```
$ docker run -d -p 8081:8081 -e "port=8081" --name app --network=my-bridge-network edelpa/python-app:1.0
```

- Encontrar la IP interna del contenedor.

```
$ docker inspect app
```



Docker

Redes – Práctica

- En otro terminal, ejecutar un contenedor de alpine en forma interactiva.

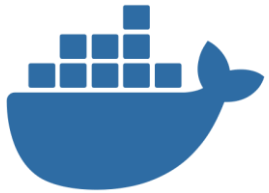
```
$ docker run -it --name client alpine:latest
```

- Instalar el cliente *curl* en el contenedor.

```
$ apk --no-cache add curl
```

- Desde el contenedor, llamar a la aplicación del contenedor anterior (*Cambiar la IP con la que le corresponda*). ¿Qué ha sucedido?

```
$ curl 172.21.0.2:8081 --connect-timeout 5
```



Docker

Redes – Práctica

- Retornar al terminal anterior y enlazar el contenedor *client* a la red creada.

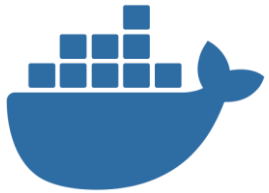
```
$ docker network connect my-bridge-network client
```

- Retornar al terminal del contenedor *client* y volver a realizar la llamada. (*Cambiar la IP con la que le corresponda*) ¿Cuál ha sido el resultado?

```
$ curl 172.21.0.2:8081 --connect-timeout 5
```

- Retornar al terminar anterior e inspeccionar la red creada.

```
$  docker inspect my-bridge-network
```



Docker

Redes – Práctica

- Desconectar ambos contenedores de la red creada

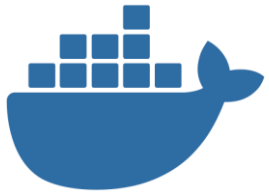
```
$ docker network disconnect my-bridge-network app
$ docker network disconnect my-bridge-network client
```

- Eliminar la red creada

```
$ docker network rm my-bridge-network
```

- Asegurarse que la red esté eliminada y salir de los terminales.

```
$ docker network ls
```



Docker

Labels

- herramienta que permite que el usuario añada metadatos en formato de clave-valor.
- La sintaxis básica de las instrucciones LABEL es:

```
LABEL <key-string>=<value-string> <key-string>=<value-string>...
```

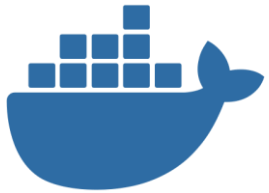
- Ejemplo de etiquetas en una imagen, en el dockerfile:

```
LABEL multi.label1="value1" multi.label2="value2" other="value3“
```

- Para ver los label de una imagen:

```
$ docker image inspect --format='{{json .Config.Labels}}' myimage
```

- Las etiquetas incluidas en las imágenes base o padre (imágenes en la línea FROM) son heredadas por su imagen.
- Si una etiqueta ya existe, pero con un valor diferente, el valor aplicado más recientemente anula cualquier valor establecido previamente.



Docker

Labels

- Para etiquetar un contenedor con dos labels en tiempo de ejecución (-l, --label, --label-file):

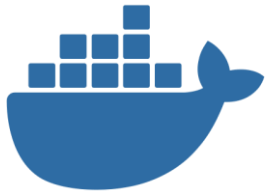
```
$ docker run -l my-label --label com.example.foo=bar ubuntu bash
```

- Utilice el indicador --label-file para cargar varias labels desde un archivo

```
$  docker run --label-file ./labels ubuntu bash
```

- El formato del archivo de labels es similar al formato para cargar variables de entorno. (A diferencia de las variables de entorno, los labels no son visibles para los procesos que se ejecutan dentro de un contenedor). El siguiente ejemplo muestra un formato de archivo de labels:

```
com.example.label1="a label"
# this is a comment
com.example.label2=another\ label
com.example.label3
```



Docker

Contenido

Tecnología de contenedores

Daremos un paseo en el mundo de los contenedores, beneficios y principales retos.

Arquitectura Docker

Revisaremos los principales componentes de Docker y cómo se interrelacionan entre sí.

Cliente Docker

Estudiaremos y practicaremos los comandos más comunes que nos da Docker para trabajar con los contenedores.

Administración de contenedores

Echaremos un vistazo a herramientas de administración de contenedores y nos centraremos especialmente en Docker Desktop.

Registros de contenedores

Revisaremos un conjunto de librerías de imágenes de contenedores y en especial nos centraremos en eDocker Hub.

Docker Compose

Estudiaremos como podemos definir y ejecutar varias aplicaciones en contenedores de forma fácil y rápida.

Herramientas de contenedores

Hay una amplia variedad de herramientas de terceros que pueden ser integradas e incrementar la productividad con contenedores Docker.

Orquestación de contenedores

Analizaremos las herramientas de orquestación de contenedores y veremos en detalle Docker Swarm.

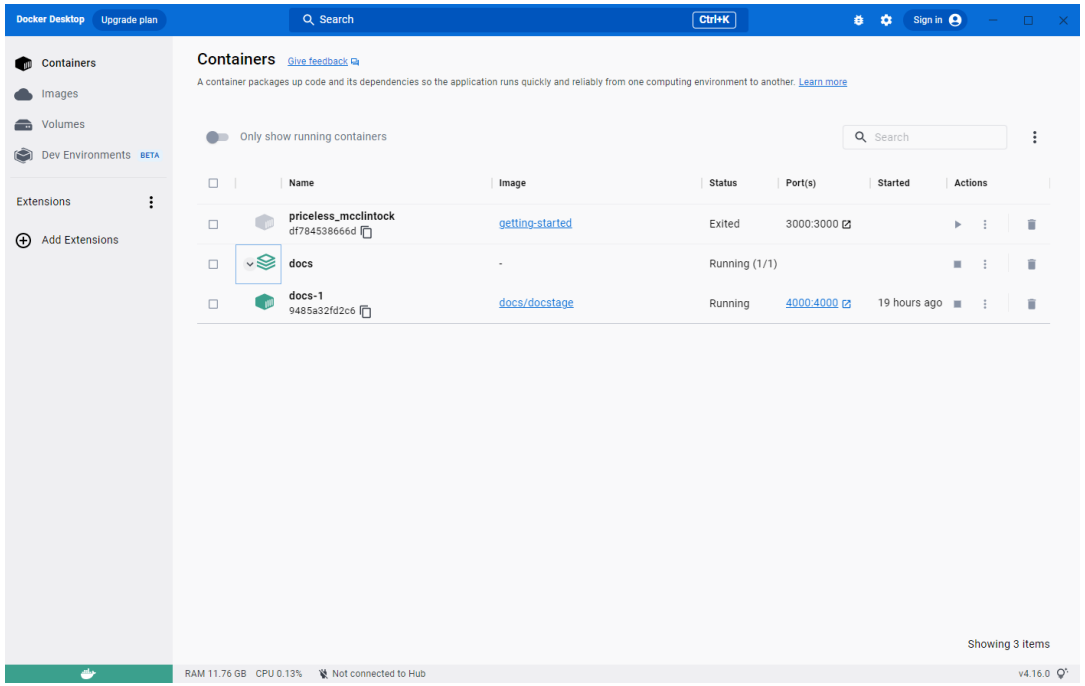
Docker Desktop

¿Qué es?

- Docker Desktop es una aplicación que se instala en un entorno Mac, Linux o Windows y que permite crear, compartir y ejecutar aplicaciones y microservicios en contenedores.

- Proporciona una GUI (Interfaz Gráfica de Usuario) sencilla que le permite gestionar contenedores, aplicaciones e imágenes directamente desde su ordenador. Docker Desktop puede utilizarse por sí solo o como herramienta complementaria de la CLI.

- Se encarga de las asignaciones de puertos, los problemas del sistema de archivos y otras configuraciones predeterminadas, y se actualiza periódicamente con correcciones de errores y actualizaciones de seguridad.



Docker Desktop

¿Qué está incluido?

- Docker Engine

  Tecnología de contenerización de código abierto para crear y contenerizar aplicaciones. Docker Engine actúa como una aplicación cliente-servidor.

- Docker CLI client

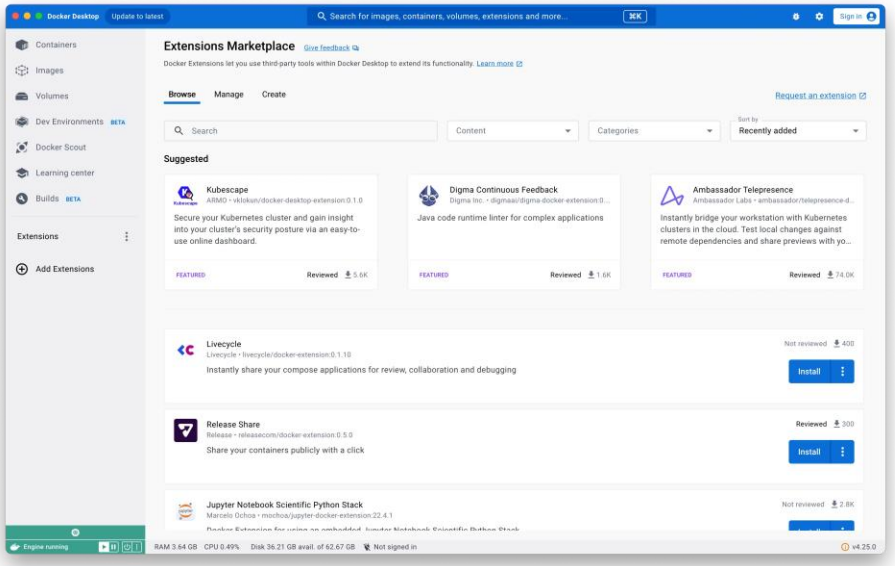
  Utiliza las APIs de Docker para controlar o interactuar con e demonio Docker a través de scripts o comandos directos de la CLI. Muchas otras aplicaciones de Docker utilizan la API subyacente y la CLI. El demonio crea y gestiona objetos Docker, como imágenes, contenedores, redes y volúmenes.

- Docker Scout

  Puede ayudar de forma proactiva a encontrar y corregir estas vulnerabilidades, ayudando a crear una cadena de suministro de software más segura. Para ello, analiza las imágenes y crea un inventario completo de los paquetes y las capas, denominado lista de materiales de software (SBOM).

- Extensiones Docker

  Permite utilizar herramientas de terceros en Docker Desktop para ampliar su funcionalidad.



Docker Desktop

¿Qué está incluido?

- Docker build

  Es una de las características más utilizadas de Docker Engine. Siempre que se está creando una imagen se está usando Docker Build. Build es una parte clave de su ciclo de vida de desarrollo de software que permite empaquetar y agrupar el código y enviarlo a cualquier lugar.

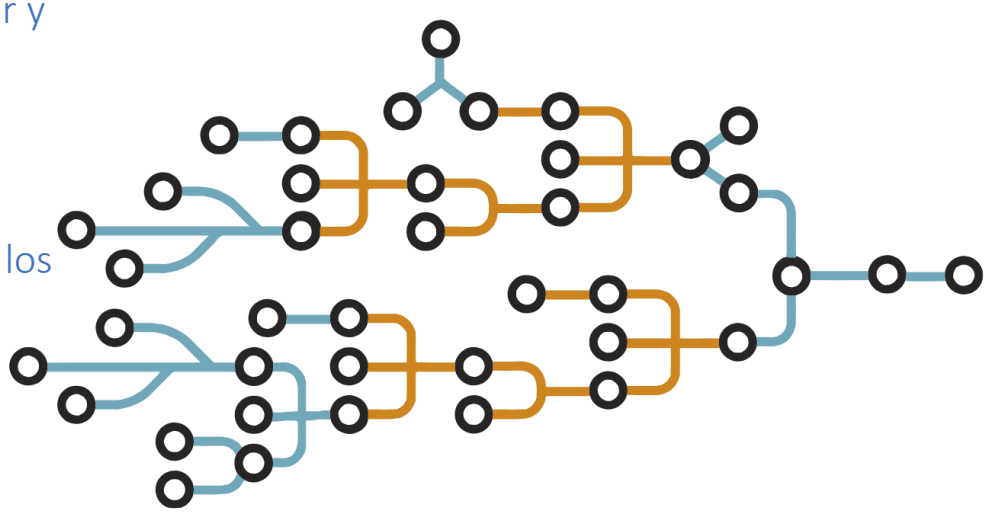
- Docker Compose:

  Es una herramienta para definir y ejecutar aplicaciones Docker multicontenedor. Con Compose, se utiliza un archivo YAML para configurar los servicios de tu aplicación. Después, con un solo comando, se crea e inicia todos los servicios a partir de la configuración.

- Docker Content Trust

  Ofrece la posibilidad de utilizar firmas digitales para los datos enviados y recibidos de registros remotos de Docker. Estas firmas permiten la verificación en el lado del cliente o en tiempo de ejecución de la integridad y el editor de etiquetas de imagen específicas.

- Kubernetes

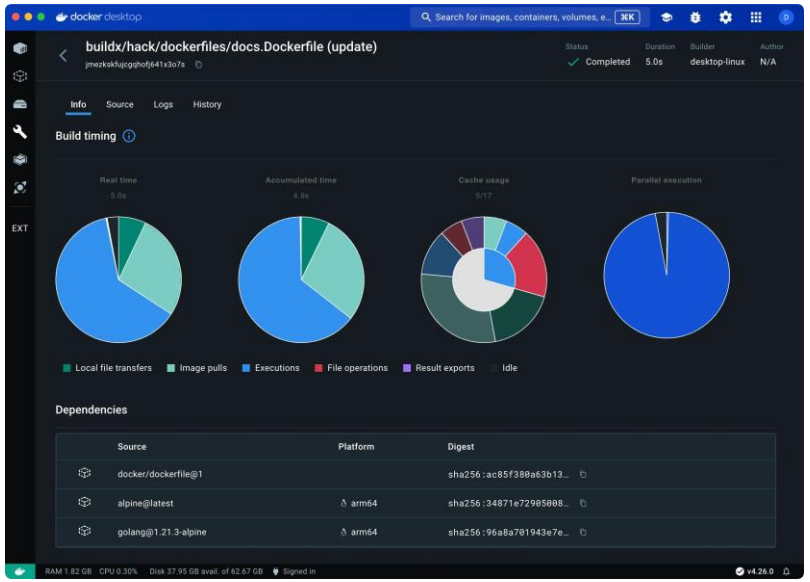


Docker Desktop

Explorando

- Contenedores

  La vista de Contenedores proporciona una vista en tiempo de ejecución de todos sus contenedores y aplicaciones. Le permite interactuar con contenedores y aplicaciones, y gestionar el ciclo de vida de sus aplicaciones directamente desde su máquina. Esta vista también proporciona una interfaz intuitiva para realizar acciones comunes para inspeccionar, interactuar y gestionar sus objetos Docker incluyendo contenedores y aplicaciones basadas en Docker Compose.

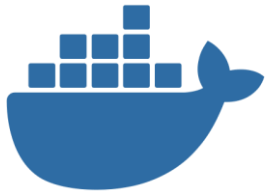


- Imágenes

  Muestra una lista de sus imágenes Docker y permite ejecutar una imagen como contenedor, extraer la última versión de una imagen de Docker Hub e inspeccionar imágenes. También muestra un resumen de las vulnerabilidades de las imágenes. Además, la vista Imágenes contiene opciones de limpieza para eliminar imágenes no deseadas del disco y recuperar espacio. Si ha iniciado sesión, también puede ver las imágenes su organización han compartido en Docker Hub.

- Builds

  Es una interfaz sencilla que le permite inspeccionar su historial de construcciones y gestionar las construcciones utilizando Docker Desktop. Por defecto, muestra una lista de todas las compilaciones en curso y completadas



Docker

Contenido

Tecnología de contenedores

Daremos un paseo en el mundo de los contenedores, beneficios y principales retos.

Arquitectura Docker

Revisaremos los principales componentes de Docker y cómo se interrelacionan entre sí.

Cliente Docker

Estudiaremos y practicaremos los comandos más comunes que nos da Docker para trabajar con los contenedores.

Administración de contenedores

Echaremos un vistazo a herramientas de administración de contenedores y nos centraremos especialmente en Docker Desktop.

Registros de contenedores

Revisaremos un conjunto de librerías de imágenes de contenedores y en especial nos centraremos en eDocker Hub.

Docker Compose

Estudiaremos como podemos definir y ejecutar varias aplicaciones en contenedores de forma fácil y rápida.

Herramientas de contenedores

Hay una amplia variedad de herramientas de terceros que pueden ser integradas e incrementar la productividad con contenedores Docker.

Orquestación de contenedores

Analizaremos las herramientas de orquestación de contenedores y veremos en detalle Docker Swarm.

Registro de contenedores

Conceptos

- ¿Qué es un registro de contenedores?

  Es un repositorio (o colección de repositorios) que se utiliza para almacenar y acceder a imágenes de contenedores.

  Los registros de contenedores pueden admitir el desarrollo de aplicaciones basadas en contenedores, a menudo como parte de los procesos DevOps.

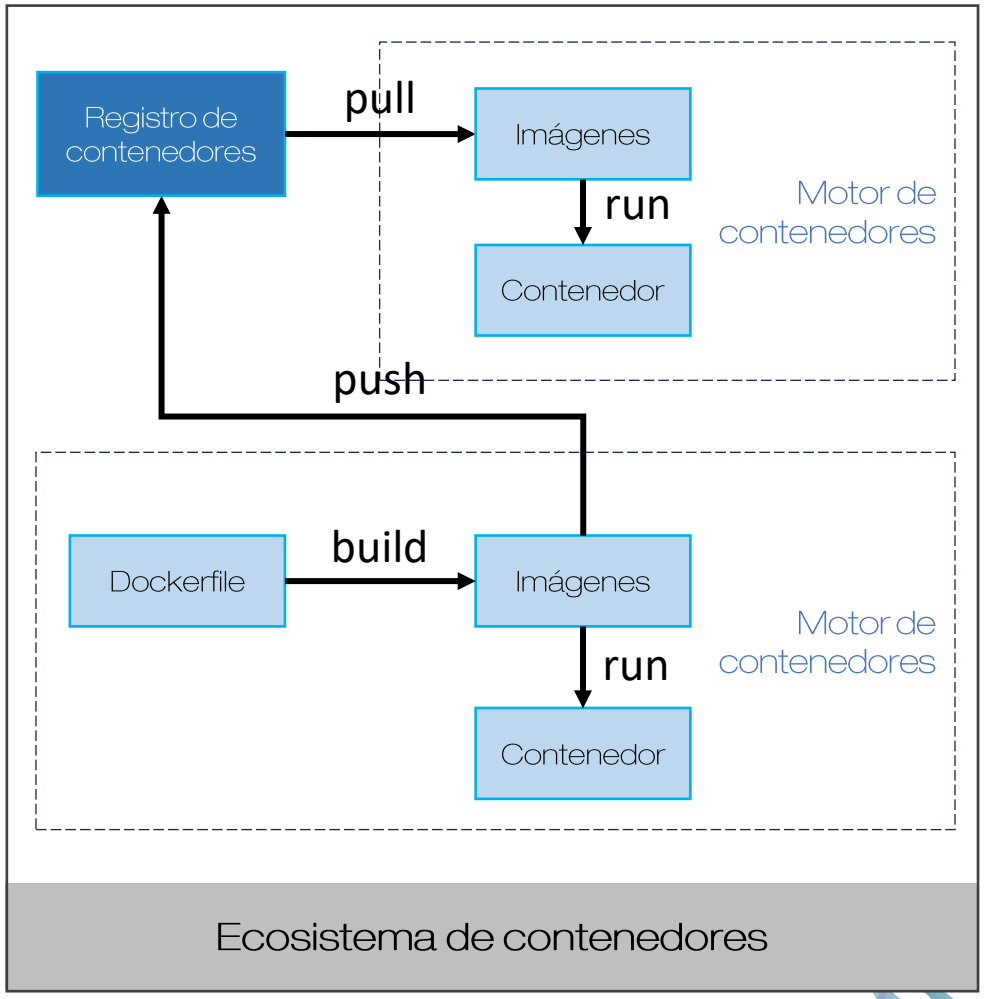
  Los registros de contenedores pueden conectarse directamente a plataformas de contenedores como Docker y Kubernetes.

- Registros públicos

  Los registros públicos suelen ser utilizados por personas o equipos pequeños que desean comenzar a usar su registro lo más rápido posible. Sin embargo, a medida que sus empresas crecen, esto puede generar problemas de seguridad más complejos, como la aplicación de parches, la privacidad y el control de acceso.

- Registros privados

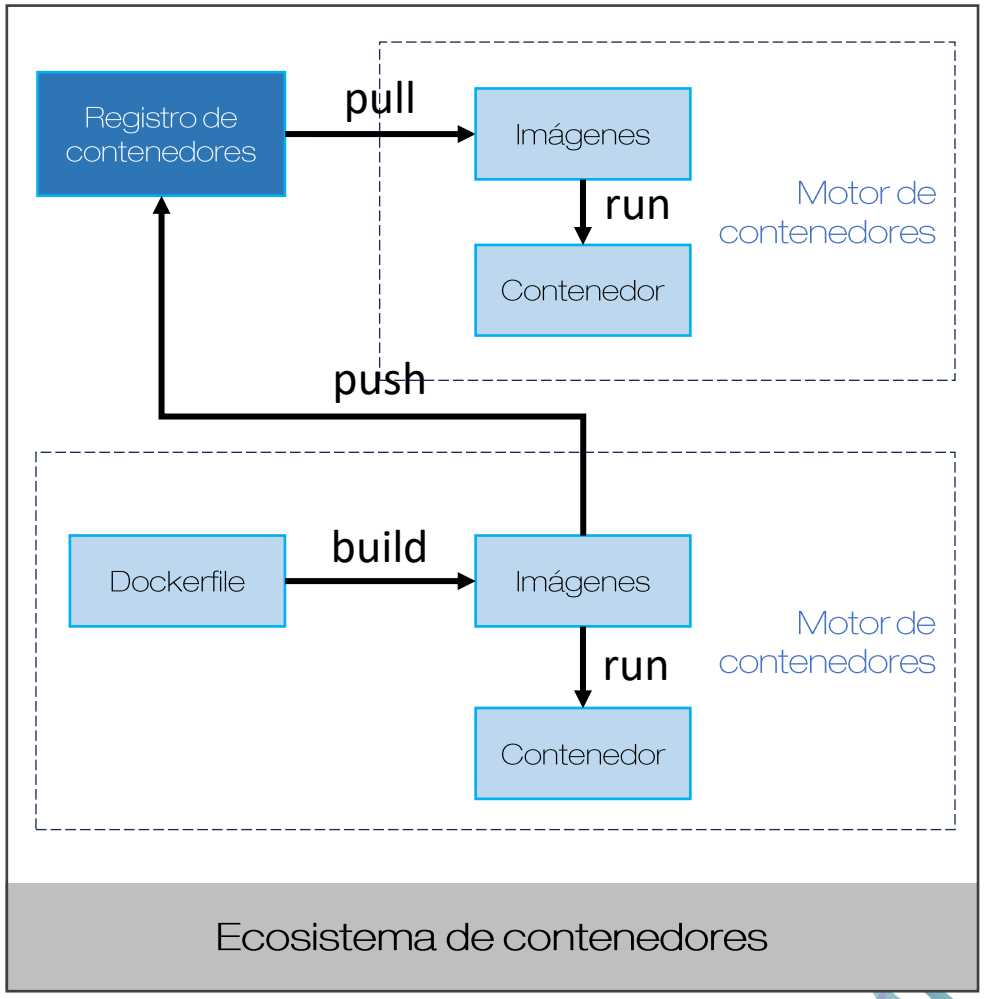
  Los registros privados permiten incorporar la seguridad y la privacidad al almacenamiento de imágenes de contenedores empresariales, ya sea que se encuentren alojados de forma remota o local. A menudo, los registros privados ofrecen soporte técnico y funciones de seguridad avanzados.



Registro de contenedores

Conceptos

- Aspectos de un registro privado
  - Compatibilidad con varios sistemas de autenticación
  - Gestión de control de acceso basado en funciones (RBAC) para imágenes locales
  - Capacidad de análisis de puntos vulnerables para mejorar la seguridad y la configuración
  - Capacidad para registrar el uso en medios verificables, de modo que se pueda rastrear la actividad hasta atribuírsele a un solo usuario
  - Mejoras para la automatización: En un entorno de DevOps, el uso de contenedores, e imágenes o registros de contenedores, permite a los desarrolladores implementar cada servicio de aplicación de forma independiente. De esta manera, no es necesario combinar cambios de código, se mejoran las pruebas y se colabora con el aislamiento de errores tanto en pruebas como en producción.



Registro de contenedores

Algunos proveedores de registro de contenedores

- Registro de contenedores Docker Hub

  Es el registro de contenedores más popular, ya que es el repositorio Docker por defecto. Funciona como un marketplace para imágenes de contenedores públicos, lo que lo convierte en la mejor opción si decides distribuir públicamente una imagen.

  En cuanto a los precios, el sistema de niveles permite desbloquear algunas funciones específicas con un plan de pago.

- Registro de contenedores Azure (ACR)

  Azure Container Registry de Microsoft se basa en Docker Registry 2.0, donde la autenticación se gestiona mediante Azure RBAC. El registro de contenedores de Azure viene con características que la mayoría de los competidores aún no están ofreciendo, tales como;

  - Purga automática de imágenes antiguas
  - Política de retención para manifiestos no etiquetados
  - Confianza en el contenido

  Es importante señalar que la confianza en el contenido se basa en un concepto creado por Docker que permite firmar las imágenes que se envían al registro de contenedores de Azure.

Registro de contenedores

Algunos proveedores de registro de contenedores

- Registro de contenedores Elastic de Amazon (ECR)

  El ECR de Amazon puede configurarse para admitir registros de Docker privados y públicos. Estos registros pueden utilizarse con AWS IAM para controlar los niveles de acceso de los usuarios, los servicios y las aplicaciones. Básicamente, puede definir qué usuarios tienen acceso a las imágenes de contenedor protegidas.

  AWS ECR también viene equipado con escaneo de vulnerabilidades en imágenes, lo que lo convierte en una característica esencial para DevSecOps. Esto se debe a que utiliza la base de datos Common Vulnerabilities and Exposures (CVEs) de Clair para evaluar la gravedad de los problemas encontrados.

  Otra gran característica de AWS ECR son las etiquetas de imagen inmutable. Cuando está activada, esta característica garantiza que nadie pueda anular una imagen una vez que se ha enviado al registro de contenedores..

Registro de contenedores

Algunos proveedores de registro de contenedores

- Registro de paquetes de GitHub

  Un aspecto notable del registro de contenedores de GitHub es el hecho de que ofrece una experiencia sin fisuras, especialmente para los desarrolladores. Básicamente, la autenticación se gestiona con un token de acceso personal, y eso es todo lo que necesitas.

  Otra opción es utilizar un repositorio público, aunque en este caso, necesitas que tus usuarios se autentiquen con una cuenta de usuario de GitHub.

- Registro de contenedores GitLab

  GitLab tiene su propio registro de contenedores, de uso gratuito y compatible con imágenes de contenedores Docker, así como con Helm Chart Ptiene una versión autoalojada y una basada en la nube a través de GitLab.com. Una de las grandes características de GitLab Container Registry es su política de limpieza que elimina las etiquetas que coinciden con un determinado patrón regex.

  Alternativamente, puedes probar su Package Registry que también es gratuito y soporta Composer, Conan, Generic, Maven, npm, NuGet, PyPI, y RubyGem. Sin duda, una gran opción a tener en cuenta si ya utilizas GitLab para el repositorio de tus proyectos.

Registro de contenedores

Algunos proveedores de registro de contenedores

- Registro de artefactos Google (GAR)

  Google Artifact Registry es su nueva forma de gestionar imágenes de contenedores y artefactos que no son contenedores, como paquetes de Maven, npm, Python, Apt o incluso Yum.

  En primer lugar, GAR puede integrarse fácilmente con procesos CI/CD para agilizar la creación y el despliegue de contenedores. Además, también proporciona un escáner de vulnerabilidades en las imágenes que se puede activar manualmente..

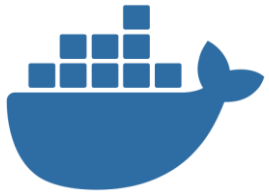
- Registro de contenedores Harbor

  Necesita ser instalado, configurado y gestionado por el usuario. Se puede usar con cualquier distribución Linux que soporte Docker. Tenga en cuenta que también puede desplegar Harbor en su clúster Kubernetes.

  Harbor soporta características como:

  - Escaneo de vulnerabilidades
  - Recolección de basura
  - Replicación entre regiones
  - Confianza de contenido

  En general, es una opción sólida a considerar si planea alojar su registro de contenedores..



Registro de contenedores

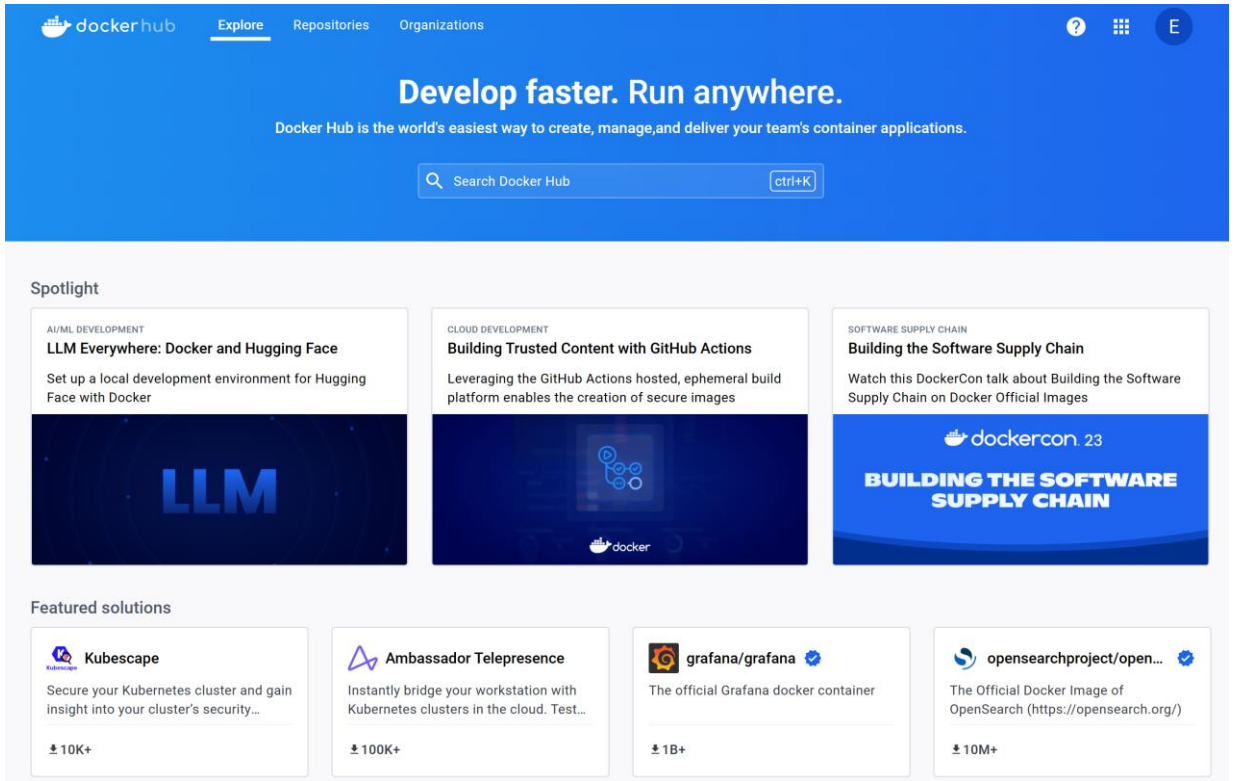
Algunos proveedores de registro de contenedores (referencial)

	Amazon ECR	Azure CR	Docker Hub	Google AR	GitHub Packages	GitLab CR	Harbor
Precio	Almacenamiento: libre (por 1 año con AWS Free Usage Tier) hasta 0.5GB, luego $0.10 GB/mes Ingreso de transferencia de datos: $0.09 per GB/mo	Almacenamiento: $0.167 /día por 10GB con el plan básico. $0.667 /día por 100GB con el plan estándar $1.667 /día por 500GB con el plan Premium.	El precio no se basa en el almacenamiento.	Almacenamiento: Libre hasta 0.5GB, luego $0.10 GB/mes Ingreso de transferencia de datos: Potencialmente libre, ver precios	Almacenamiento: 500MB para el plan libre. 2GB para el plan PRO 2GB para el plan Team 50GB para el plan Enterpris $0.25/GB por almacenamiento adicional Ingreso de transferencia de datos: Libre dentro de GitHub Actions 1GB/mes (Free) 10GB/mes (Pro) 10GB/mes (Team) 100GB/mes (Enterprise) $0.5/GB por transferencia adicional	Libre	Libre (necesita ser auto-hospedado)
Soporte paquetes (npm, Maven, yum, etc.)	❌ (AWS CodeArtifact will help with that)	❌ (pero soporta artefactos OCI)	❌	✅	✅	✅	❌ (pero soporta artefactos OCI)
Autenticación	AWS IAM	Azure RBAC	Password o Access Token	GCP IAM	Access token	Personal Access Token o Deploy Token	AD, LDAP, RBAC, and OIDC
Replicación en regiones	✅	✅ (only available with Premium tier)	❌	✅	❌	❌ (disponible en auto-hospedado)	✅
MFA Push/Pull	✅	❌	✅ (beta)	❌	❌	❌	✅
SLA	99.9%	99.9%	n/a	99.9%	n/a	n/a	Auto-hospedado
Colector de basura	✅	✅	❌	❌	❌	✅	✅
Escaneo de imágenes	✅ Libre	✅ Libre	✅ Libre, pero ver plan	✅ ($0.26/imágen)	❌	✅ (sólo con el plan Ultimate)	✅
Límites:	- 120 000/min	- up to 10 000/min.	- up to 1 440/min.	- 60 000/minute	n/a	n/a	Auto-hospedasdo
- Pull	- 120 000/min			- 18 000/minute[5]			
- Push		- up to 2000/minute.	- Desconocido				

Registro de contenedores

Docker Hub

- Es el repositorio de imágenes para contenedores que aloja la mayor cantidad de imágenes oficiales de grandes proveedores de software.
  - Imágenes oficiales de Docker.
  - Publicadores verificados.
  - OOS auspiciados por Docker.
- Es el repositorio predeterminado de la plataforma de Docker Engine por las funcionalidades de este sistema.
  - Repositorios donde subir y descargar imágenes.
  - Construir automáticamente imágenes desde GitHub y Bitbucket y subirlas a Docker Hub.
  - Disparar acciones después de subir una imagen para integrar Docker Hub con otros servicios.
  - Contiene además extensiones para Docker Desktop y Plugins para Docker Engine.



Registro de contenedores

Docker Hub

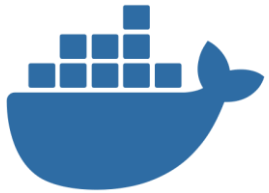
- Crear un repositorio
  - En el portal de Docker Hub, seleccionar la opción de repositorios, cerca de la esquina superior derecha, seleccionar *Create repository.*
  - El nombre del repositorio debe de ser único, debe tener entre 2 y 255 caracteres.
  - Solo puede contener letras en minúsculas, números, guiones y guiones bajos.
  - La descripción puede tener hasta 100 caracteres y son usados en el resultado de las búsquedas.
  - El repositorio puede ser público o privado.
  - Hay más opciones si se trata de una organización. Estas opciones requieren una suscripción de pago.
  - Crear una imagen
    - Construyendo: *docker build –t <hub-user>/<repo-name>[:tag]*
    - Volviendo a etiquetar la imagen con *docker tag <existing-image> <hub-user>/<repo-name>[:<tag>]*
    - Usando *Docker commit <existing-container> <hub-user>/<repo-name>[:<tag>]* para guardar los cambios.
  - Subir una imagen
    - Docker push <hub-user>/<repo-name>:<tag>

Registro de contenedores

Docker Hub

- Accesos repositorios
  - Dentro de un repositorio, se puede dar accesos a otros para cargar y descargar de tu repositorio. También se puede ver las etiquetas de las imágenes asociadas.
- Administrar repositorios
  - Cambiar un repositorio de público a privado.
    - Ir al repositorio y seleccionar la opción de ***Settings***, luego seleccionar ***Make private*** y entrar el nombre del repositorio para confirmar.
  - Mover imágenes entre repositorios.
    - De un repositorio personal a personal.
      - Cuando se consolidan los repositorios personales, se puede descargar imágenes privadas de un repositorio inicial y cargarlas en otro repositorio. Para evitar la pérdida de imágenes privadas, realizar los siguientes pasos:
        - Usar *docker login* desde el cliente, con el ID original de Docker y descargar las imágenes privadas.
        - Etiquetar las imágenes privadas con el ID destino de Docker, por ejemplo:

          docker tag namespace1/dockerimage new_namespace/dockerimage
        - Usando *docker login*, desde el cliente, con el ID destino de Docker, y cargar la imagen con la nueva etiqueta al nuevo espacio de nombre.
  - Eliminar un repositorio.
    - Seleccionar la opción de ***Settings***, luego seleccionar el ***Delete repository***, entrar el nombre del repositorio para confirmar.



Registro de contenedores

Docker Hub

- Explorando imágenes.
  - Usuarios anónimos: 100 descargas por 6 horas por dirección IP.
  - Usuarios autenticados: 200 descargas por 6 horas.
  - Usuarios con una suscripción Docker: Hasta 5000 descargas por día.

- ¿Cómo se autentica las descargas?
  - Docker Desktop: Puedes iniciar sesión en Docker Hub desde el menú de Docker Desktop
  - Docker Engine: Con el comando docker login desde un terminal para autenticarte con Docker Hub
  - Docker Swarm: debe utilizar el parámetro -- with-registry-auth para autenticarse con Docker Hub
  - GitHub Actions: Para construir y subir imágenes a Docker Hub, usar acción login. Si está utilizando otra acción, debe añadir su nombre de usuario y token de acceso de forma similar para la autenticación.
  - Kubernetes: Se verá en el curso de Kubernetes.
  - Plataformas de terceros: Artifactory, AWS Codebuild, AWS ECS/Gargate, Azure Pipelines, Chipper, CircleCI, etc.

- Escanear imágenes.

Registro de contenedores

Docker Hub

- Docker Hub impone unos límites de descargas para proteger el sistema y la infraestructura.

  En Docker, podemos buscar imágenes utilizando el comando docker search, que ayuda y obtiene las imágenes de búsqueda del repositorio público docker. En el comando docker search, tenemos la palabra clave TERMINO para localizar y encontrar las imágenes que coinciden con el TERMINO especificado.

  docker search <término>

  Ejemplos

  - Buscar imágenes de busybox

    docker seach busybox
  - Buscar las imágenes oficiales de Ubuntu

    docker search --filter=is-official=true ubuntu
  - Para obtener ayuda del comando search

    docker search --help

```
PS C:\Users\edwin> docker search --filter=is-official=true ubuntu
NAME                DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
ubuntu              Ubuntu is a Debian-based Linux operating sys…   16773     [OK]
websphere-liberty   WebSphere Liberty multi-architecture images …   296       [OK]
open-liberty        Open Liberty multi-architecture images based…   62        [OK]
neurodebian         NeuroDebian provides neuroscience research s…   105       [OK]
ubuntu-debootstrap  DEPRECATED; use "ubuntu" instead                52        [OK]
ubuntu-upstart      DEPRECATED, as is Upstart (find other proces…   115       [OK]
```

Registro de contenedores

Docker Hub

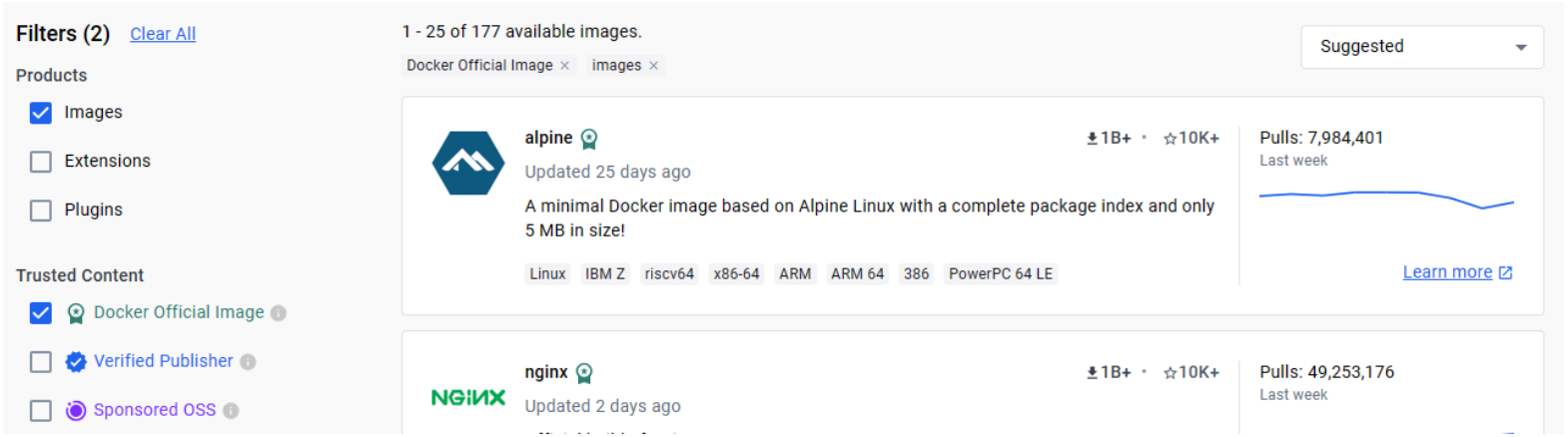
- Imágenes oficiales.

  Estas imágenes proporcionan repositorios base esenciales que sirven como punto de partida para la mayoría de los usuarios.

  Incluyen sistemas operativos como Ubuntu y Alpine, lenguajes de programación como Python y Node, y otras herramientas esenciales como Redis y MySQL.

  Las imágenes son algunas de las más seguras de Docker Hub. Esto es particularmente importante ya que las Imágenes Oficiales Docker son algunas de las más populares en Docker Hub. Normalmente, las imágenes oficiales de Docker tienen pocas o ninguna vulnerabilidad.

  Las imágenes ejemplifican las mejores prácticas de Dockerfile y proporcionan documentación clara para servir de referencia a otros autores de Dockerfile.



Registro de contenedores

Docker Hub

- Programa de editores verificados de Docker

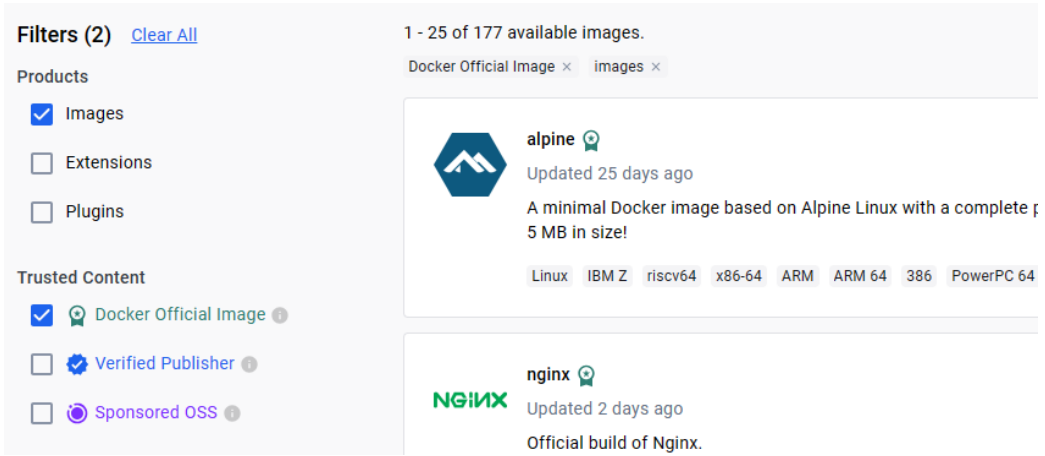
  El programa Docker Verified Publisher ofrece imágenes de alta calidad de editores comerciales verificados por Docker.

  Estas imágenes ayudan a los equipos de desarrollo a construir cadenas de suministro de software seguras, minimizando la exposición a contenido malicioso al principio del proceso para ahorrar tiempo y dinero más adelante.

  Las imágenes que forman parte de este programa tienen una insignia especial en Docker Hub que facilita a los usuarios la identificación de los proyectos que Docker ha verificado como editores comerciales de alta calidad.

  El programa Docker Verified Publisher Program (DVP) ofrece varias características y ventajas a los editores de Docker Hub. El programa concede las siguientes ventajas en función del nivel de participación:

  - Logotipo del repositorio
  - Insignia de editor verificado
  - Clasificación de búsqueda prioritaria en Docker Hub
  - Información y análisis
  - Análisis de vulnerabilidades
  - Asientos adicionales de Docker Business
  - Eliminación de la limitación de tarifas para desarrolladores
  - Oportunidades de marketing conjunto



Registro de contenedores

Docker Hub

- Programa de código abierto patrocinado por Docker

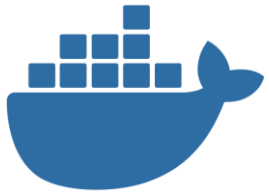
  Las imágenes de código abierto patrocinadas por Docker son publicadas y mantenidas por proyectos de código abierto patrocinados por Docker a través del programa.

  Las imágenes que forman parte de este programa tienen un distintivo especial en Docker Hub que facilita a los usuarios la identificación de los proyectos que Docker ha verificado como proyectos de código abierto de confianza, seguros y activos.

  El programa Docker-Sponsored Open Source (DSOS) ofrece varias características y ventajas a los desarrolladores de código abierto no comerciales.El programa concede las siguientes ventajas a los proyectos elegibles:

  - Logotipo del repositorio
  - Insignia de código abierto patrocinado por Docker verificado
  - Información y análisis
  - Acceso a Docker Scout para la gestión de la cadena de suministro de software
  - Eliminación de la limitación de tarifas para desarrolladores
  - Mejor visibilidad en Docker Hub

  Estas ventajas son válidas durante un año y los editores pueden renovarlas anualmente si el proyecto sigue cumpliendo los requisitos del programa. Los miembros del programa y todos los usuarios que extraigan imágenes públicas del espacio de nombres del proyecto obtienen acceso a extracciones y salidas ilimitadas.



Docker

Contenido

Tecnología de contenedores

Daremos un paseo en el mundo de los contenedores, beneficios y principales retos.

Arquitectura Docker

Revisaremos los principales componentes de Docker y cómo se interrelacionan entre sí.

Cliente Docker

Estudiaremos y practicaremos los comandos más comunes que nos da Docker para trabajar con los contenedores.

Administración de contenedores

Echaremos un vistazo a herramientas de administración de contenedores y nos centraremos especialmente en Docker Desktop.

Registros de contenedores

Revisaremos un conjunto de librerías de imágenes de contenedores y en especial nos centraremos en eDocker Hub.

Docker Compose

Estudiaremos como podemos definir y ejecutar varias aplicaciones en contenedores de forma fácil y rápida.

Herramientas de contenedores

Hay una amplia variedad de herramientas de terceros que pueden ser integradas e incrementar la productividad con contenedores Docker.

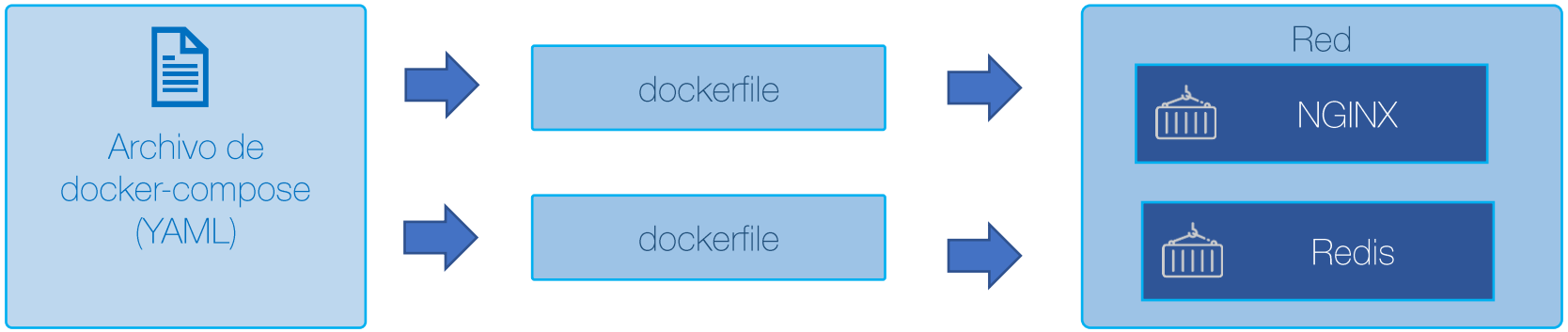
Orquestación de contenedores

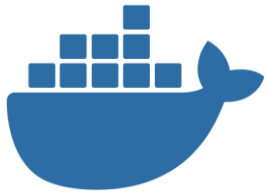
Analizaremos las herramientas de orquestación de contenedores y veremos en detalle Docker Swarm.

Docker compose

¿qué es?

- Es una herramienta para definir y ejecutar aplicaciones de varios contenedores como si fuera un solo servicio. Así se puede crear un archivo para definir los distintos contenedores y poner todo en marcha con un solo comando.
- Los archivos de docker-compose se escriben utilizando el formato YAML (Yet Another Markup Language)
- Por ejemplo, si se tiene una aplicación que requiere un servidor NGINX y una base de datos Redis, puedes crear un archivo de docker-compose y ejecutar ambos como un servicio sin necesidad de iniciar cada uno por separado.





Docker compose

Comandos sobre el archivo docker-compose.yaml

- up — Inicia todos los servicios (contenedores).
- down — Detiene todos los servicios.
- build — Valida que las imágenes estén listas, construyendo las imágenes que falta.
- images — Lista las imágenes que están listas en el archivo *docker-compose.yaml*.
- stop — Detiene los contenedores de los servicios contenidos en *docker-compose.yaml*.
- run — Creará los contenedores de los servicios especificados en *docker-compose.yaml*.
- ps — Lista todos los contenedores especificados en *docker-compose.yaml*.
- pause — Pausa todos los contenedores especificados en *docker-compose.yaml*.
- unpause — Quita la pausa de los contenedores previamente pausados.
- logs — Presenta los logs del salida del servicio indicado.

Docker compose

Estructura del archivo

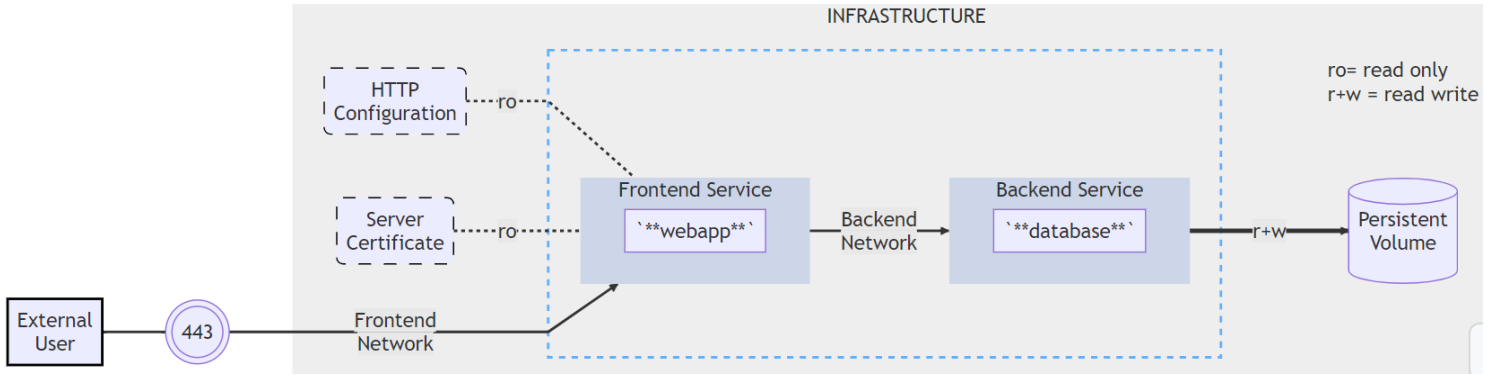
Los archivos de docker-compose trabajan aplicando múltiples comandos que son declarados dentro de un archivo *docker-compose.yaml*. La estructura básica de este archivo es:

```
version: “X”  #opcional desde la versión 1.27.0
services: ...
volumes: ...
networks: ...
configs: ...
secrets: ...
```

- Services: Se refiere a la configuración de los contenedores.
- **Volumes:** Las áreas físicas de disco que son compartidas entre el disco y el servidor.
- **Networks:** Las reglas de comunicación ente los contenedores y el servidor.
- **Configs:** Almacenes información que no es sensitiva, como archivos de configuración fuera de las imágenes..
- **Secrets:** Almacenes de información para almacenar información sensitiva como certificados, claves, etc.

Docker compose

Ejemplo ilustrativo

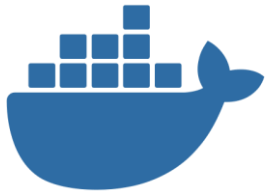


El ejemplo está compuesto por las siguientes partes:

- 2 servicios, respaldados por imágenes Docker: webapp y base de datos
- 1 secreto (certificado HTTPS), inyectado en el frontend
- 1 configuración (HTTP), inyectada en el frontend
- 1 volumen persistente, conectado al backend
- 2 redes

```
services:
  frontend:
    image: example/webapp
    ports:
      - "443:8043"
    networks:
      - front-tier
      - back-tier
    configs:
      - httpd-config
    secrets:
      - server-certificate

  backend:
    image: example/database
    volumes:
      - db-data:/etc/data
    networks:
      - back-tier
```

```
volumes:
  db-data:
    driver: flocker
    driver_opts:
      size: "10GiB"

configs:
  httpd-config:
    external: true

secrets:
  server-certificate:
    external: true

networks:
  front-tier: {}
  back-tier: {}
```



Docker compose

Sección servicios: directivas más comunes

- **image** Configura la imagen y la versión que será utilizada para construir el contenedor. Usando esta directiva se asume que la imagen ya se encuentra en el servidor o en Docker Hub.
- **build** Esta directiva puede ser utilizada en lugar de la imagen. Específica la ubicación del Dockerfile que será utilizado para construir el contenedor.
- **container_name** Define un nombre para el contenedor, el cuál corresponde al nombre del servicio.
- **volumes** Monta una ruta enlazada en la máquina servidor que puede ser usado por el contenedor.
- **environment** Define las variables de entorno para ser pasadas al contenedor.
- **depends_on** Configura un servicio como una dependencia, por lo que algunos servicios consiguen ser cargados antes que otros,
- **ports** Mapea un puerto de un contenedor al servidor utilizando una estructura *servidor_ip*:*contenedor_ip*
- **links** Enlaza este servicio a cualquier otro servicio en el archivo de docker-compose especificando sus nombres aquí.

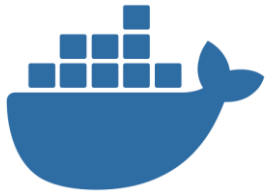
docker-compose trabaja en una estructura de directorios. Se puede tener varios grupos de contenedores en un servidor – sólo crea un directorio por contenedor y un *docker-compose.yaml* por cada directorio.

Docker compose

Práctica – docker-compose.yaml

```
versión: “3.9”
services:
  nginx:
    image: nginx:1.23.3
    container_name: test_nginx
    ports:
        - “80:80”
        - “443:443”
    volumes:
        - ./repo:/var/www
        - ./nginx/nginx.conf:/etc/nginx/nginx.conf
        - ./nginx/ssl:/etc/nginx/ssl/
    working_dir: /var/www
    links:
        - php
```

```
 php:
    image: php:8.1.0-fpm
    container_name: test_php
    volumes:
        - ./repo:/var/www
    working_dir: /var/www
    links:
        - mysql
mysql:
    image: mysql:5.7.36
    container_name: test_mysql
    ports:
      - “3306:3306”
    volumes:
        - ./mysql:/var/mysql
    environment:
        - MYSQL_ROOT_PASSWORD=root_password
```



Docker compose

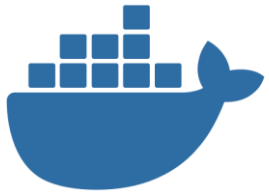
Comandos

Este comando crea e inicia los recursos descritos en el archivo docker-compose.yaml:

```
$ docker-compose up
```

Algunas opciones:

- detach/d — Ejecuta los contenedores en segundo plano.
- build — Siempre construye las imágenes aun cuando estas existan.
- force-recreate — Fuerza la recreación de los contenedores aún si no se han cambiado las especificaciones.
- v/renew-anon-volumes — Recrea los volúmenes anónimos en lugar de mantener los datos de los anteriores.
- no-deps — Evita la creación de los contenedores enlazados (dependientes)
- scale — Escala un servicio a un número de instancias. (***replicas*** en docker-compose.yaml)



Docker compose

Comandos

Ejemplos:

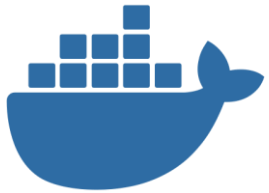
```
$ docker-compose up –d          # Inicia los contenedores en segundo plano.
$ docker-compose up nginx       # Inicia los contenedores nginx y mysql.
$ docker-compose up mysql       # Inicia solo el contenedor mysql.
```

Inicia los contenedores creados.

```
$ docker-compose start [SERVICE…]
```

Para mostrar los contenedores (incluyendo los detenidos) ejecutamos el siguiente comando.

```
$ docker-compose ps -a
```



Docker compose

Comandos

Para ingresar a un contenedor, por ejemplo, el contenedor de php, ejecutamos la siguiente instrucción

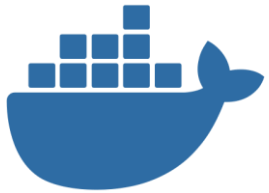
```
$ docker exec –ti test_php bash
```

Para detener los servicios activos preservando los contenedores, volúmenes y redes, junto a todas las modificaciones realizadas en ellas, debemos ejecutar el siguiente comando.

```
$ docker-compose stop
```

En el caso que se quiera descartar los cambios y destruir todo, ejecutar.

```
$ docker-compose down
```



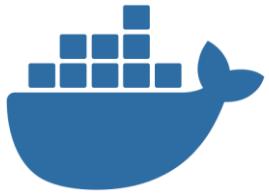
Docker compose

Práctica

El objetivo de la siguiente práctica es que los alumnos por sí mismos construyan dos soluciones utilizando *docker-compose.yaml* a partir del ejemplo anterior.

- La primera solución es replicar la práctica de mysql y adminer de la sección anterior, esta vez utilizando servicios y volúmenes de docker-compose.
- La segunda solución es similar a la anterior, pero esta vez utilizando la imagen de la base de datos de postgre y la imagen de su cliente de administración dpage/pgadmin

En ambos casos se deberá crear una tabla con al menos dos campos y un registro, luego eliminar el despliegue y volver a realizarlo para validar que las tablas persisten.



Docker compose

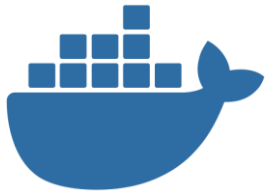
Comandos

Iniciar docker-compose desde el mismo directorio del archivo: Este comando despliega los servicios indicados en el archivo docker-compose.yaml

```
$ docker-compose build
```

Algunas opciones:

- *detach/d* Ejecuta el contenedor en segundo plano.
- force-rm Elimina los contenedores intermedios mientras se despliega.
- no-cache Construye las imágenes sin usar alguna capa previa que se encuentre en el cache.
- pull Siempre descarga la última versión de la imagen.
- build-arg: key=val Pasa cualquier argumento desde la línea de comandos.

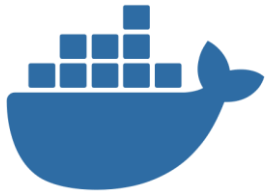


Docker compose

Build

Opciones de configuración que son aplicados al momento de ejecutar el comando build.

```
versión: “3.9”
services:
  web:
    build:
      context: ./dir
    dockerfile: Dockerfile-v2
    args:
      - buildno=1
      - gitcommit=cdc3b19
    image: webapp:tag
    labels:
      com.example.description: “Example webapp”
      com.example.Department: “Datahack”
    network: host
```



Docker compose

Práctica

El objetivo de la siguiente práctica es que los alumnos por sí mismos construyan una solución utilizando *docker-compose.yaml* a partir de la práctica de dockerfile-multistage.

- Generar el archivo docker-compose.yaml utilizando la extensión de docker.
  - (Crtl+Shift+P) y buscamos Docker: Add Docker Files to Workspace…
  - De la lista de opciones para Plataforma de Aplicaciones, seleccionamos .NET: ASP.NET Core
  - De la lista de opciones de Sistema Operativo, seleccionamos Linux.
  - En el valor de puerto, ingresamos el valor del puerto donde deseamos que se despliegue
  - Respondemos que sí cuando pregunta incluir los archivos de Docker Compose

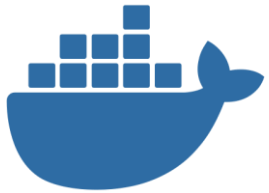
- Revisar el archivo de docker-compose.yaml y ejecutarlo.

Docker compose

Reserva y límites de recursos

Opciones para limitar los recursos de un contenedor

```
versión: “3.9”
services:
  nginx:
    image: nginx
    deploy:
      resources:
        limits:
          cpus: “2”
          memory: 512M
        reservations:
          cpus: “1”
          memory: 256M
```



Docker compose

Scale

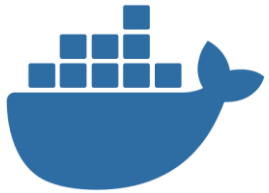
Archivo:docker-compose.yaml

```
---------------------------
version: "3"
services:
  db:
    image: postgres:latest
    environment:
      POSTGRES_USER: pspdfkit
      POSTGRES_PASSWORD: password
      POSTGRES_DB: pspdfkit
  pspdfkit:
    image: "pspdfkit/pspdfkit:latest"
    #container_name: scale_pr_pspdfkit -> No permite escalar
    environment:
      PGUSER: pspdfkit
      PGPASSWORD: password
      PGDATABASE: pspdfkit
      PGHOST: db
      PGPORT: 5432
      DASHBOARD_USERNAME: dashboard
      DASHBOARD_PASSWORD: secret
      SECRET_KEY_BASE: secret-key-base
      JWT_PUBLIC_KEY: |
        -----BEGIN PUBLIC KEY-----
        MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBALd41vG5rMzG26hhVxE65kzWC+bYQ94t
        OxsSxIQZMOc1GY8ubuqu2iku5/5isaFfG44e+VAe+YIdVeQY7cUkaaUCAwEAAQ==
        -----END PUBLIC KEY-----
      JWT_ALGORITHM: RS256
      API_AUTH_TOKEN: secret
      ASSET_STORAGE_BACKEND: built-in
    depends_on:
       - db
    expose:
      - "5000"
```

```
 nginx:
    image: nginx:latest
    container_name: scale_pr_nginx
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    depends_on:
      - pspdfkit
    ports:
      - "4000:4000“
```

Archivo: nginx.conf

```
user  nginx;

events {
    worker_connections   1000;
}
http {
        server {
              listen 4000;
              location / {
                proxy_pass http://pspdfkit:5000;
              }
        }
}
```



Docker compose

Scale

- Crear los archivos de la diapositiva anterior dentro de una carpeta *pspdkit-ningx*
- Ejecutar el siguiente comando para crear el proyecto

```
$ docker compose up
```

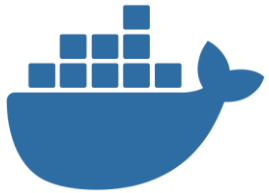
- Listar los contenedores desplegados

```
$ docker compose ps
```

- Navegar a la url *htttp://localhost:4000/dashboard*
- Escalar el servicio pspdfkit a dos contenedores

```
$ docker compose up --scale pspdfkit=2
```

- Listar los contenedores desplegados y volver a navegar al dashboard



Docker

Contenido

Tecnología de contenedores

Daremos un paseo en el mundo de los contenedores, beneficios y principales retos.

Arquitectura Docker

Revisaremos los principales componentes de Docker y cómo se interrelacionan entre sí.

Cliente Docker

Estudiaremos y practicaremos los comandos más comunes que nos da Docker para trabajar con los contenedores.

Administración de contenedores

Echaremos un vistazo a herramientas de administración de contenedores y nos centraremos especialmente en Docker Desktop.

Registros de contenedores

Revisaremos un conjunto de librerías de imágenes de contenedores y en especial nos centraremos en eDocker Hub.

Docker Compose

Estudiaremos como podemos definir y ejecutar varias aplicaciones en contenedores de forma fácil y rápida.

Herramientas de contenedores

Hay una amplia variedad de herramientas de terceros que pueden ser integradas e incrementar la productividad con contenedores Docker.

Orquestación de contenedores

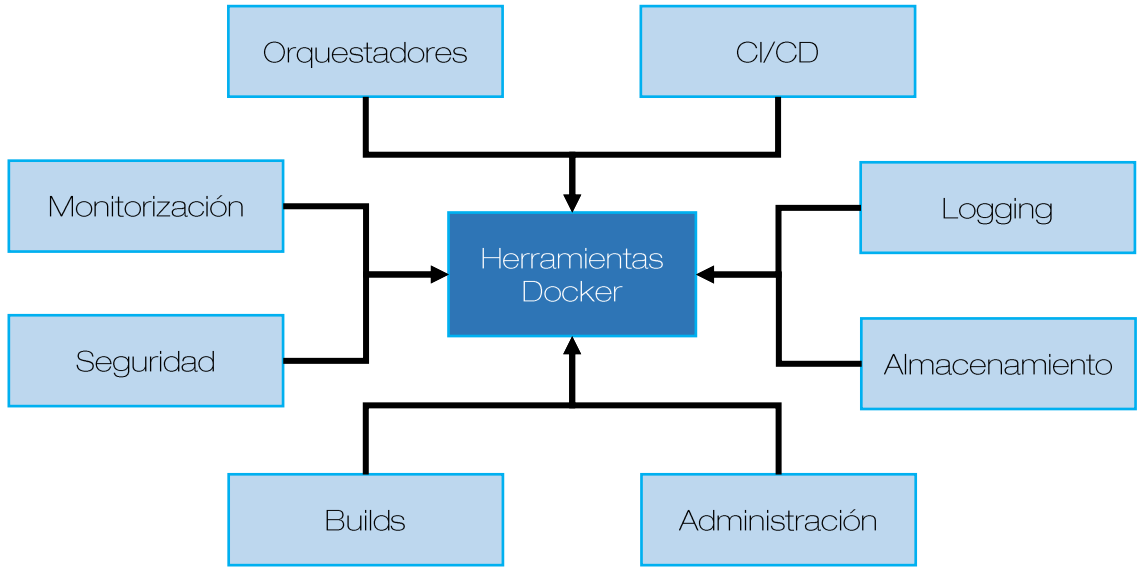
Analizaremos las herramientas de orquestación de contenedores y veremos en detalle Docker Swarm.

Herramientas de contenedores

Productividad

- Herramientas de productividad

  Docker tiene una gran cantidad de capacidades atractivas, pero quizás la mejor sea su compatibilidad con herramientas de terceros. Tanto si eres principiante como experto, las que te presentamos a continuación mejorarán la velocidad y la eficiencia.



Herramientas de contenedores

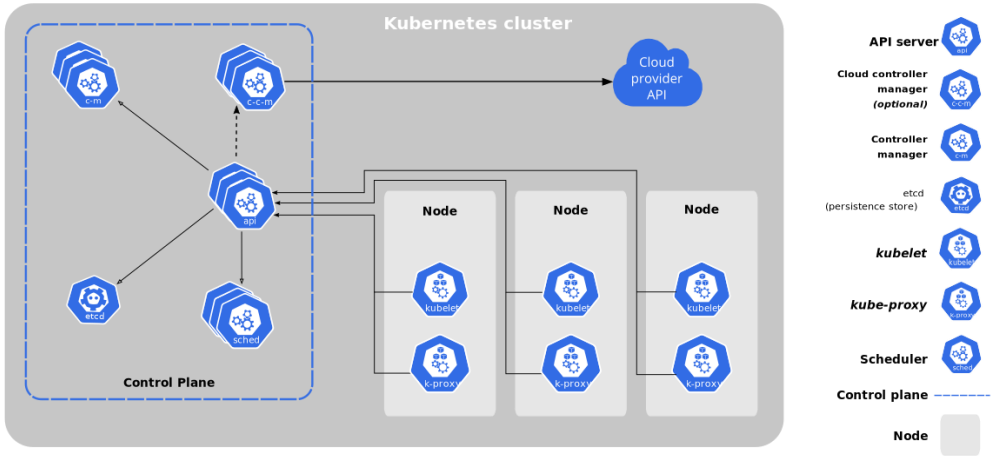
Productividad - Orquestadores

- Kubernetes

  Kubernetes sigue siendo una opción popular entre los desarrolladores al ser una plataforma de código abierto de amplias herramientas que ofrece flexibilidad y facilidad de uso mejorando los flujos de trabajo y maximizando la productividad. La plataforma también ofrece una gran biblioteca de funcionalidades desarrolladas por comunidades de todo el mundo, lo que le confiere unas capacidades de gestión de microservicios inigualables.

  El Kubernetes consta de cuatro componentes principales:

  - Nodo: Es responsable de ejecutar cargas de trabajo en contenedores y puede ser físico o virtual. Estas máquinas sirven como hosts para los tiempos de ejecución de los contenedores, y también facilitan la comunicación entre los contenedores y el servicio Kubernetes.
  - Clúster: Es un conjunto de nodos que comparten recursos y ejecutan aplicaciones en contenedores.
  - Controladores de replicación: Agentes inteligentes responsables de la programación y asignación de recursos entre contenedores.
  - Etiquetas (Labels): Son etiquetas que el servicio Kubernetes utiliza para identificar los contenedores que son miembros de un pod.



Herramientas de contenedores

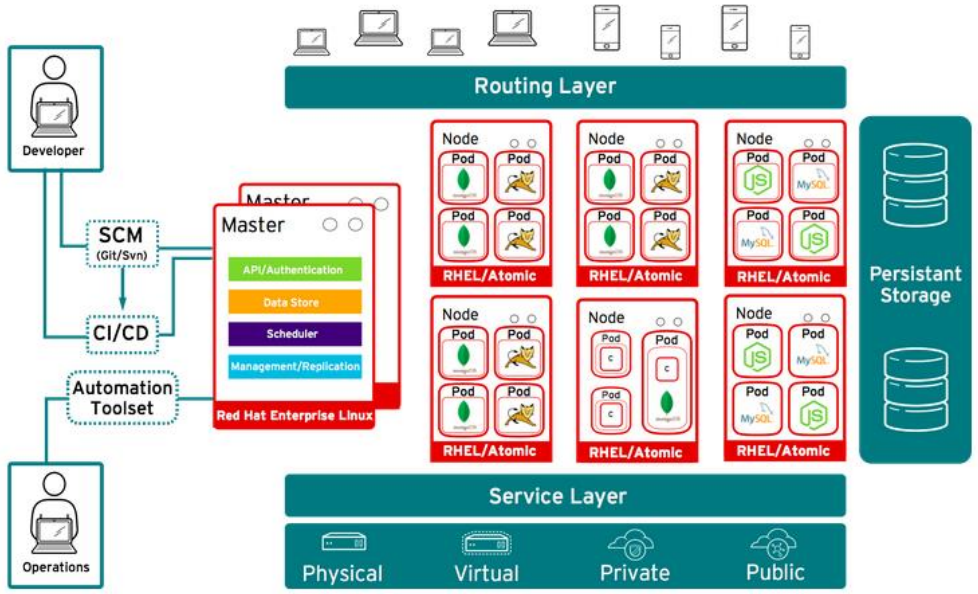
Productividad - Orquestadores

- Red Hat Openshift

  OpenShift ofrece los modelos de computación en nube Plataforma como servicio (PaaS) y Contenedor como servicio (CaaS). Esto le permite esencialmente definir el código fuente de su aplicación en un Dockerfile o convertir su código fuente en un contenedor utilizando un constructor Source-to-Image.

  Las principales características de Red Hat OpenShift incluyen:

  - Los pipelines Jenkins incorporados agilizan los flujos de trabajo, permitiendo una producción más rápida.
  - Viene con un runtime de contenedores integrado (CoreOS), pero también se integra bien con los runtime estándar CRI-O y Docker
  - Integra varias herramientas de desarrollo y operaciones para ofrecer una orquestación de contenedores de autoservicio.
  - Su Embedded Operator Hub garantiza a los administradores un fácil acceso a servicios como operadores Kubernetes, soluciones de terceros y acceso directo a proveedores de servicios en la nube, como AWS
  - OpenShift es una plataforma de código abierto, agnóstica en cuanto a proveedores, sin compromiso de dependencia de un proveedor.



Herramientas de contenedores

Productividad - Orquestadores

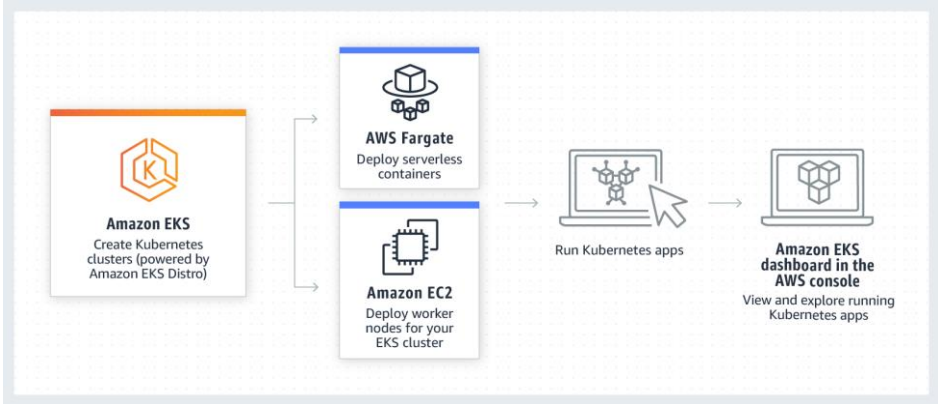
- Servicio Amazon Elastic Kubernetes (EKS)

  Amazon EKS ayuda a los desarrolladores a crear, implementar y escalar aplicaciones Kubernetes on-premise o en la nube de AWS.

  EKS automatiza tareas como la aplicación de parches, las actualizaciones y el aprovisionamiento de nodos, lo que ayuda a las organizaciones a enviar clústeres fiables, seguros y altamente escalables.

  Al mismo tiempo, EKS elimina todo el tedio y las tareas de configuración manual para administrar clústeres Kubernetes, ayudando a reducir los esfuerzos de realizar tareas repetitivas para ejecutar sus aplicaciones.

  - EKS proporciona un plano de control de Kubernetes flexible disponible en todas las regiones. Esto hace que las aplicaciones de Kubernetes alojadas en EKS estén altamente disponibles y sean escalables.
  - Puede administrar directamente sus aplicaciones desde Kubernetes utilizando AWS Controllers para Kubernetes.
  - Ampliar la funcionalidad de su clúster Kubernetes es sencillo gracias a los complementos de EKS (add-ons)
  - Escale, cree, actualice y finalice nodos de su clúster EKS fácilmente mediante un único comando.



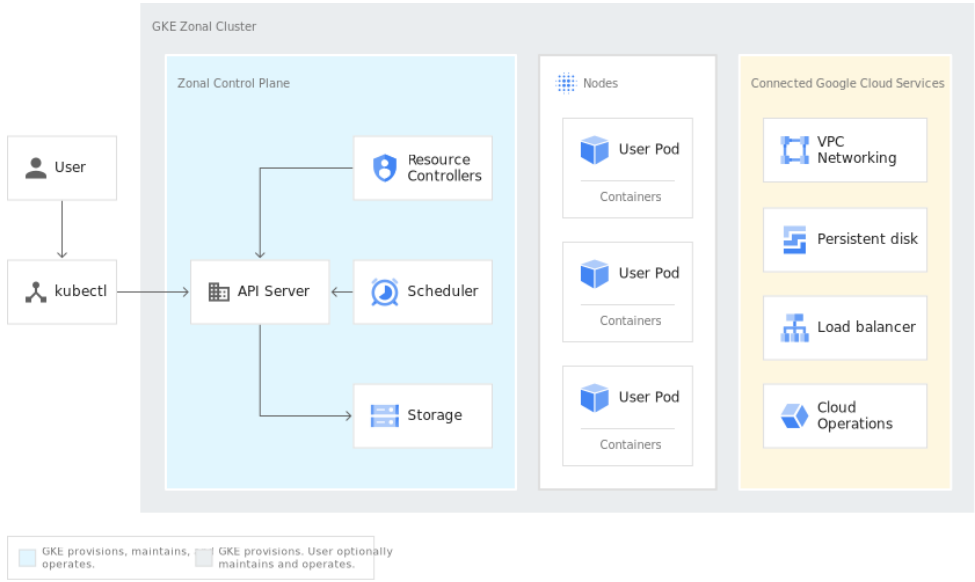
Herramientas de contenedores

Productividad - Orquestadores

- Google Kubernetes Engine (GKS)

  GKE es un servicio de orquestación gestionado que proporciona un entorno fácil de usar para desplegar, gestionar y escalar contenedores Docker en Google Cloud Platform.

  Al mismo tiempo, el motor de servicios permite crear aplicaciones ágiles y sin servidor sin comprometer la seguridad. Con múltiples canales de lanzamiento que ofrecen diferentes cadencias de actualización de nodos, GKE facilita la racionalización de las operaciones en función de las necesidades de la aplicación.



- La plataforma establece la funcionalidad básica y automatiza la gestión de clústeres para facilitar su uso
- Integra herramientas nativas de Kubernetes para que las organizaciones puedan desarrollar aplicaciones más rápidamente sin comprometer la seguridad
- Google Site Reliability Engineers ofrece asistencia en la gestión de la infraestructura
- Google mejora constantemente la plataforma GKE con nuevas funciones y mejoras, lo que la hace robusta y fiable

Herramientas de contenedores

Productividad - Orquestadores

- Azure Kubernetes Service (AKS)

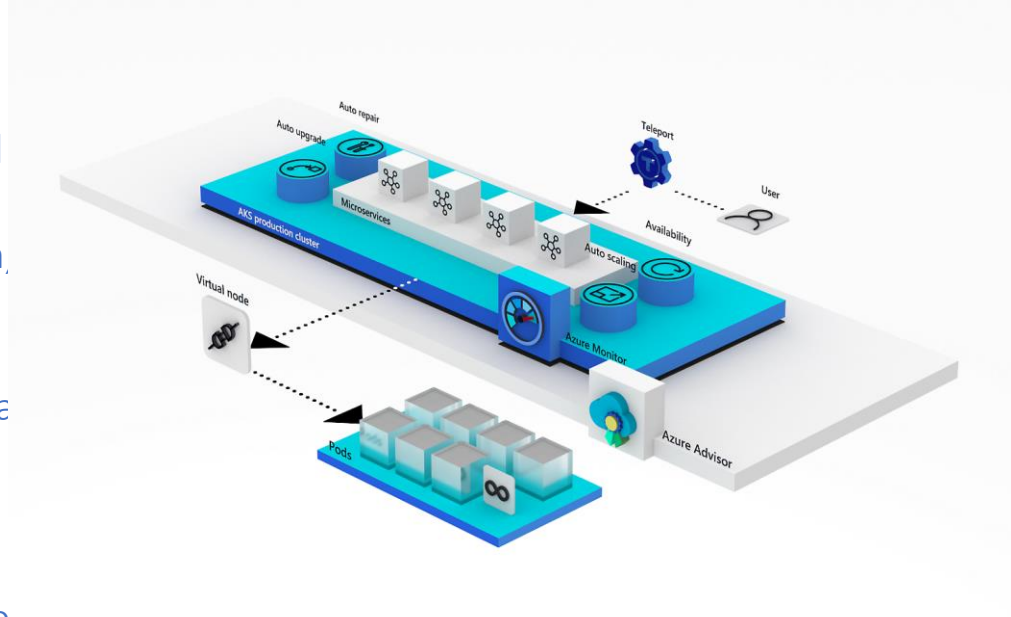
  Azure ofrece una administración y escalabilidad automatizadas de clústeres de Kubernetes para la orquestación de contenedores de nivel empresarial.

  Productividad de desarrolladores de un extremo a otro con depuración, CI/CD, registro, y mantenimiento automatizado de nodos.

  Administración avanzada de identidades y acceso para supervisar y mantener la seguridad de los contenedores para la gobernanza a escala

  Compatibilidad con recursos de Linux, Windows Server y de IoT con la implementación de AKS en la infraestructura que prefiera mediante Azure Arc.

  Aplicar controles de cumplimiento normativo mediante Azure Policy, co... límites de protección integrados y pruebas comparativas de seguridad de Internet. Obtenga un control pormenorizado del acceso y la identidad con Azure Active Directory. Conceda acceso con privilegios con acceso al clúster Just-In-Time. Use de Microsoft Defender para contenedores para mejorar, supervisar y mantener la seguridad.



Herramientas de contenedores

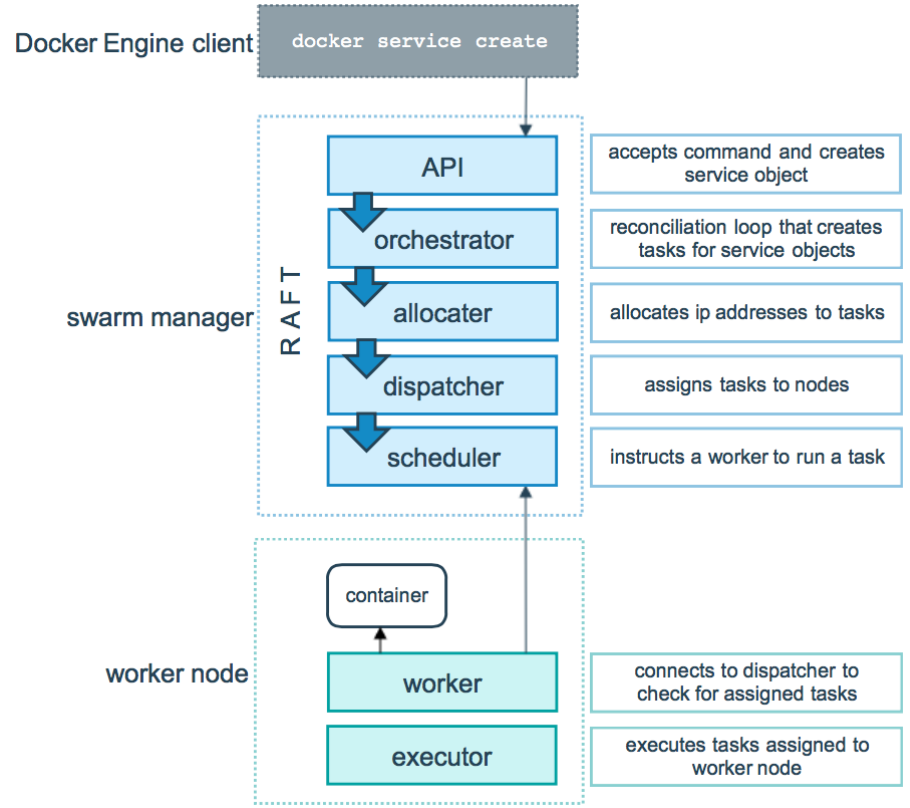
Productividad - Orquestadores

- Docker swarm

  Dado que Docker sigue siendo uno de los tiempos de ejecución de contenedores más utilizados, Docker Swarm demuestra ser una eficaz herramienta de orquestación de contenedores. Swarm facilita el escalado, la actualización de aplicaciones y el equilibrio de cargas de trabajo. Esto lo hace perfecto para el despliegue y la gestión de aplicaciones, incluso cuando se trata de clústeres extensos.

  Las principales características de Docker Swarm son:

  - Los nodos gestores ayudan con el equilibrio de carga asignando tareas a los hosts más apropiados.
  - Docker Swarm utiliza la redundancia para permitir una alta disponibilidad del servicio
  - Los contenedores Swarm son ligeros y portátiles
  - Estrechamente integrado en el ecosistema Docker, lo que permite una gestión más sencilla de los contenedores
  - No requiere plugins adicionales para su configuración
  - Garantiza una alta escalabilidad equilibrando las cargas y subiendo nodos trabajadores cuando aumenta la carga de trabajo



Herramientas de contenedores

Productividad – CI/CD

- Jenkins

  Es una herramienta de automatización de código abierto que permite a los desarrolladores de Docker de todo el mundo crear, probar y desplegar su software de forma fiable. Es una de las mejores herramientas de integración continua (CI) disponibles.

  Características principales:

  - Tiene una variedad de herramientas que se pueden integrar aún más con la pila.
  - No requiere mucho esfuerzo para configurarlo.
  - Coste: Gratis

- Circle CI

  Tiene una velocidad impecable y se encuentra entre las mejores herramientas de CI. Tiene miles de integraciones pre-construidas y cuenta con un entorno flexible para todas sus necesidades de integración y despliegue.

  Características principales:

  - Sólo se ejecuta “build" cuando se coloca una solicitud de pull. Este proceso ayuda a eliminar la construcción innecesaria cada vez que se crea una nueva rama.
  - Reduce el tiempo total de compilación ejecutando pruebas en varios contenedores al mismo tiempo.Coste:
  - Gratis hasta 6000 minutos de compilación al mes

Herramientas de contenedores

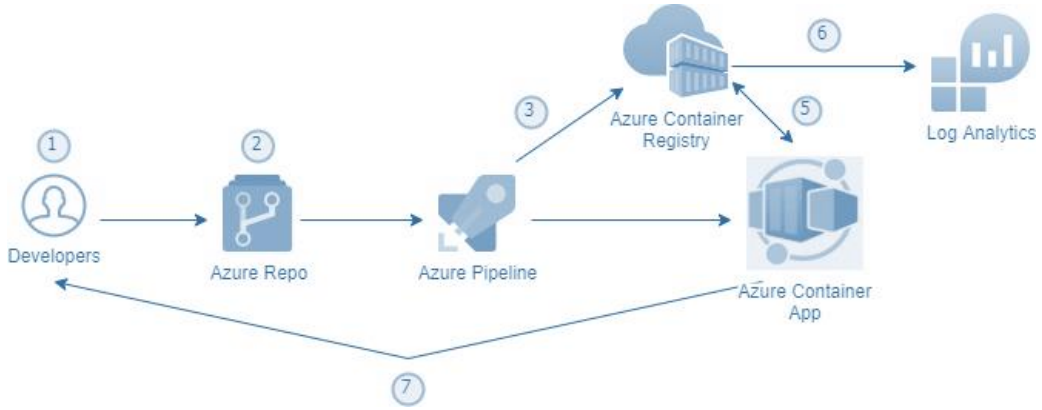
Productividad – CI/CD

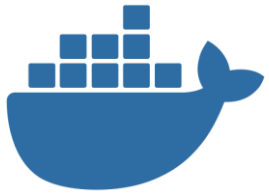
- Azure Pipelines

  Parte de Microsoft Azure, simplifica la creación y prueba de proyectos de código antes de hacerlos libremente accesibles. Puede utilizarse con prácticamente cualquier lenguaje de programación o tipo de proyecto. Azure Pipelines te permite crear y probar tu código de forma concurrente mientras lo distribuyes a cualquier destino.

  Características principales:

  - Centrado en el desarrollador: Con facilidad, los usuarios pueden crear y enviar imágenes a registros de contenedores como Docker Hub y Azure Container Registry. Los contenedores pueden desplegarse en hosts individuales o clústeres Kubernetes.
  - Soporte de colaboración para DevOps: Los usuarios pueden beneficiarse de procesos DevOps sólidos y vinculados con soporte nativo de contenedores.
  - Flexibilidad de integración: Pipeline admite un conjunto diverso de apps de construcción, prueba y despliegue desarrolladas por la comunidad, así como cientos de extensiones que van desde Slack hasta SonarCloud.





Herramientas de contenedores

Productividad – Seguridad

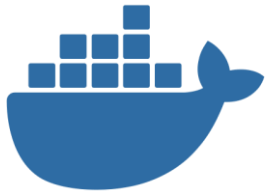
- Docker Bench

  Docker Bench es una herramienta popular para comprobar la configuración de seguridad de las instalaciones Docker. Evalúa automáticamente hosts Docker contra las mejores prácticas comunes de seguridad, proporcionando información valiosa para mejorar la seguridad de su entorno Docker.

  Características principales:

  - Comprueba si el demonio Docker está configurado de forma segura y si las funciones de seguridad en tiempo de ejecución del contenedor están habilitadas.
  - Identifica posibles vulnerabilidades en el host de Docker y proporciona una evaluación completa de la seguridad.
  - Genera un informe detallado de la auditoría de seguridad.
  - A cada prueba se le asigna un resultado INFO, NOTE, PASS o WARN.
  - Garantiza que su entorno Docker se adhiere a las mejores prácticas reconocidas por la industria.
  - Puede integrarse fácilmente en sus flujos de trabajo de seguridad existentes.

  Precio: Gratis



Herramientas de contenedores

Productividad – Seguridad

- Clair

  Clair es una herramienta de código abierto que puede realizar análisis estáticos de vulnerabilidades en contenedores Docker. Los desarrolladores la utilizan ampliamente para indexar sus imágenes de contenedores y compararlas con vulnerabilidades conocidas.

  Principales características:

  - Permite a los usuarios actualizar los datos de vulnerabilidad de varias fuentes definidas por el usuario.
  - Proporciona una API que permite a los clientes consultar la base de datos de vulnerabilidades para una imagen de contenedor determinada.
  - Realiza un análisis capa por capa de las imágenes de contenedores, inspeccionando cada capa en busca de fallos de seguridad conocidos.
  - Indexa imágenes de contenedores creando una lista de características presentes en cada imagen.
  - Se integra perfectamente con el ecosistema Docker.
  - Ofrece una herramienta de línea de comandos llamada Clair-scanner que simplifica el proceso de análisis..

  Precio: Gratis

Herramientas de contenedores

Productividad – Seguridad

- JFrog

  JFrog ofrece un potente escáner de vulnerabilidades Docker que cubre todo el ciclo de vida de sus imágenes Docker. Puede utilizar JFrog para gestionar el desarrollo, el análisis de vulnerabilidades, el control del flujo de artefactos y la distribución.

  Características principales:

  - JFrog Docker Desktop Extension escanea imágenes Docker locales para detectar vulnerabilidades de seguridad.
  - JFrog Xray realiza un profundo escaneo recursivo de las imágenes Docker.
  - Muestra todas las imágenes Docker que contienen el artefacto infectado.

  Lo mejor para: Empresas que ya utilizan productos JFrog o que buscan cubrir el ciclo de vida completo de las imágenes Docker.

  Precio: JFrog ofrece varias opciones de precios, incluyendo Pro, Enterprise X y Enterprise+. También puede obtener un plan personalizado basado en sus necesidades.

Herramientas de contenedores

Productividad – Seguridad

- Trivy

  Es otro escáner de vulnerabilidades para contenedores Docker y clústeres Kubernetes. Puede utilizarlo para detectar vulnerabilidades en varios sistemas operativos y lenguajes de programación, incluidos Oracle Linux y Red Hat Enterprise Linux.

  Características principales:

  - Cubre tanto paquetes de sistemas operativos como dependencias de lenguajes de programación.
  - Se integra perfectamente con Docker Desktop, lo que permite a los desarrolladores escanear fácilmente sus imágenes de contenedores en busca de vulnerabilidades directamente desde el Docker Dashboard.
  - Ofrece un escaneado rápido y sin estado, lo que facilita su integración en las rutinas diarias, los scripts y las canalizaciones de integración continua (CI).
  - Permite a los desarrolladores analizar y escanear un número ilimitado de imágenes de contenedores.
  - Admite varios lenguajes de programación, paquetes de sistemas operativos y dependencias de aplicaciones.
  - Sigue el principio de seguridad shift-left al permitir el escaneado temprano de artefactos y dependencias en el ciclo de vida de desarrollo de software.

  Lo mejor para: Equipos que buscan una solución de escaneado de código abierto todo en uno.

  Precio: Gratis

Herramientas de contenedores

Productividad – Seguridad

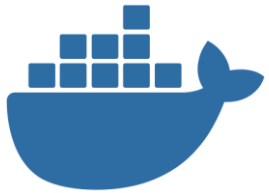
- Docker scout

  Ayudar proactivamente a encontrar y corregir estas vulnerabilidades, ayudando a crear una cadena de suministro de software más segura. Para ello, analiza sus imágenes y crea un inventario completo de los paquetes y las capas denominado lista de materiales de software (SBOM). A continuación, correlaciona este inventario con una base de datos de vulnerabilidades continuamente actualizada para identificar vulnerabilidades en sus imágenes..

  Características principales:

  - Análisis de las imágenes.
  - Base de datos de asesoramiento: Conozca las fuentes de información que utiliza Docker Scout.
  - Integraciones: Conecte Docker Scout con su CI, registros y otros servicios de terceros.
  - Panel de control: La interfaz web para Docker Scout.
  - Políticas: Asegúrese de que sus artefactos se alinean con las mejores prácticas de la cadena de suministro.

  Precio: Plan gratuito incluye hasta 3 repositorios.



Docker

Contenido

Tecnología de contenedores

Daremos un paseo en el mundo de los contenedores, beneficios y principales retos.

Arquitectura Docker

Revisaremos los principales componentes de Docker y cómo se interrelacionan entre sí.

Cliente Docker

Estudiaremos y practicaremos los comandos más comunes que nos da Docker para trabajar con los contenedores.

Administración de contenedores

Echaremos un vistazo a herramientas de administración de contenedores y nos centraremos especialmente en Docker Desktop.

Registros de contenedores

Revisaremos un conjunto de librerías de imágenes de contenedores y en especial nos centraremos en eDocker Hub.

Docker Compose

Estudiaremos como podemos definir y ejecutar varias aplicaciones en contenedores de forma fácil y rápida.

Herramientas de contenedores

Hay una amplia variedad de herramientas de terceros que pueden ser integradas e incrementar la productividad con contenedores Docker.

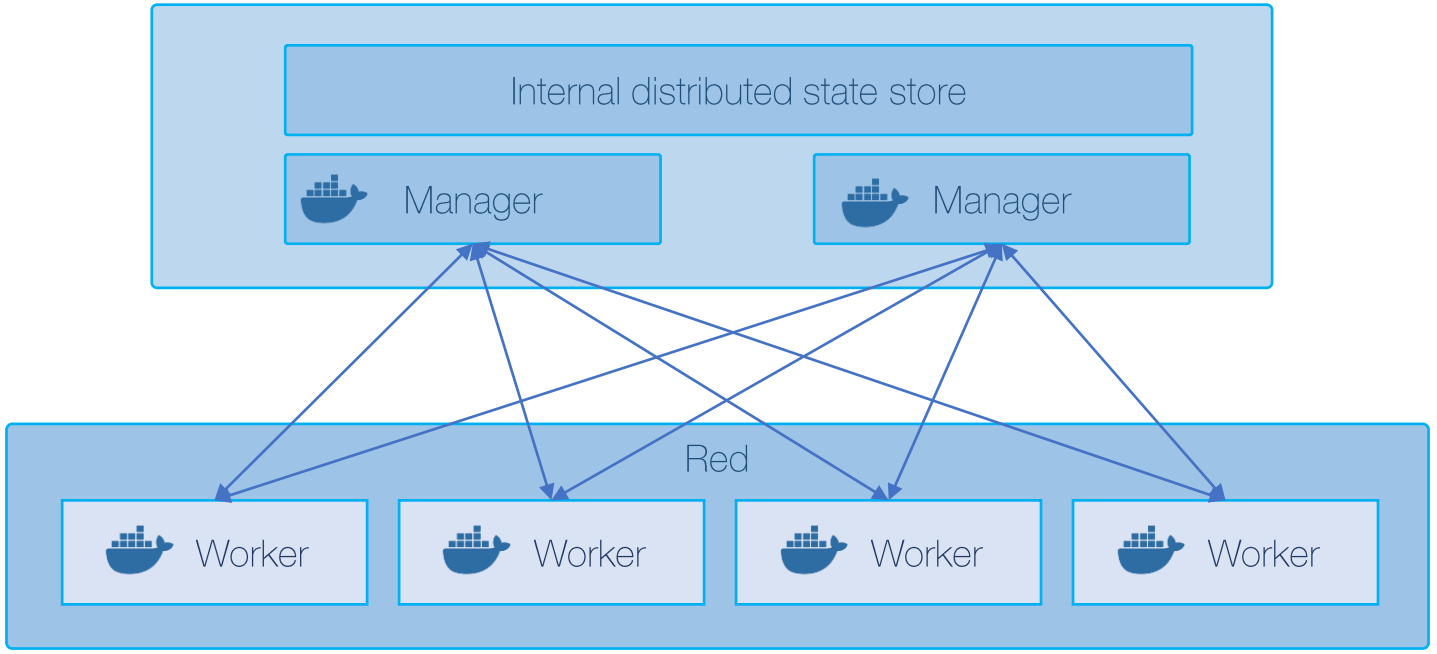
Orquestación de contenedores

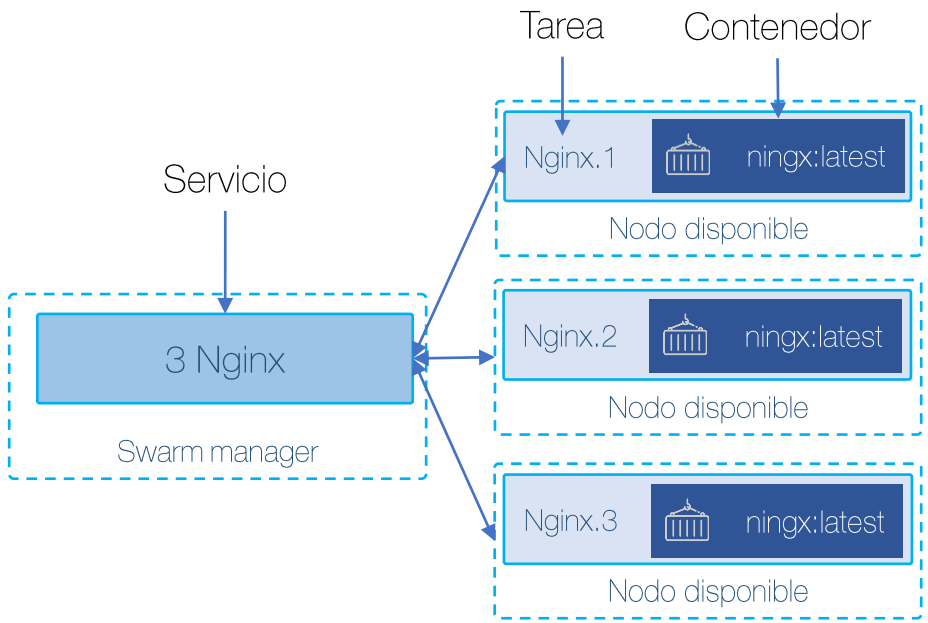
Analizaremos las herramientas de orquestación de contenedores y veremos en detalle Docker Swarm.

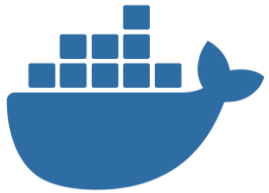
Docker swarm

¿Qué es?

Es una herramienta de orquestación de contenedores ofrecida por Docker que proporciona funciones de gestión de clústeres y orquestación integradas en el motor Docker.



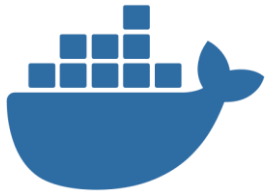




Docker swarm

Comandos

- init — Inicializa el modo swarm en docker.
- leave — Deja el modo swarm en docker.
- node ls — Lista todos los nodos.
- join-token — Lista las imágenes que están listas en el archivo *docker-compose.yaml*.
- service ls — Lista todos los servicios.
- node inspect — Elimina un servicio.
- service inspect — Describe un servicio.
- node rm — Elimina un nodo.
- service rm — Elimina un servicio.
- node promote — Promueve un nodo *worker* a *manager*.
- node demote — Degrada un nodo *manager* a un nodo *worker.*



Docker swarm

Práctica 1

Inicializar el modo swarm

```
$  docker swarm init --advertise-addr <MANAGER-IP>
```

Revisar el estado de swarm.

```
$ docker info
```

Ver el listado de los nodos

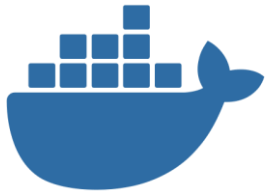
```
$ docker node ls
```

Recuperar el token para agregar un nuevo nodo

```
$  docker swarm join-token worker
```

Agregar un nuevo nodo.

```
$  docker swarm join --token SWMTKN-1-49nj1c14ie34wxv-8v…k743oj  192.168.99.100:2377
```



Docker swarm

Práctica 1

Desplegar un servicio

```
$ docker service create --replicas 1 --name helloworld alpine ping docker.com
```

Listar los servicios desplegados.

```
$ docker service ls
```

Ver los detalles del servicio desplegado

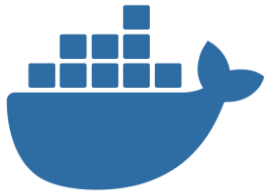
```
$ docker service inspect –pretty helloworld
```

Escalar el servicio

```
$  docker service scale helloworld=5
```

Listar la lista de tareas actualizada

```
$ docker service ps helloworld
```



Docker swarm

Práctica

Eliminar el servicio

```
docker service rm helloworld
```

Desplegar redis como servicio

```
$ docker service create --replicas 3 --name redis --update-delay 10s redis:3.0.6
```

Ver los detalles del servicio desplegado

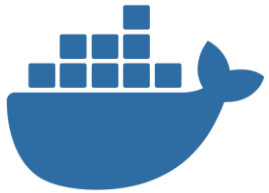
```
$ docker service inspect --pretty redis
```

Actualizar el servicio

```
$ docker service update --image redis:3.0.7 redis
```

Visualizar las tareas actualizadas

```
$ docker service ps redis
```



Docker swarm

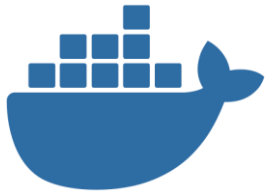
Docker Stack

Es un conjunto de comandos integrados en el motor de Docker desde la versión 1.12 y que permite desplegar soluciones utilizando ficheros yaml de docker compose (versión 3.0) en Docker Swarm.

Algunas diferencias con docker compose son que no se puede generar nuevas imágenes con los comandos de docker stack y que varias partes de los archivos de docker compose son ignorados.

Comandos de docker stack:

- docker stack deploy — Despliega un nuevo stack o actualiza uno existente.
- docker stack ls — Lista los stacks.
- docker stack ps — Lista las tareas en el stack.
- docker stack rm — Elimina uno o más stacks
- docker stack services — Lista los servicios en el stack.

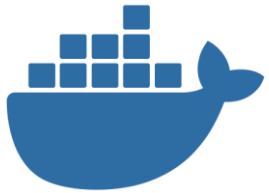


Docker swarm

Docker Stack – Práctica

Copiar en un archivo docker-compose.yaml el siguiente código:

```
version: "3.8"
services:
  mysql:
    image: mysql:5.7
    volumes:
      - mysql-data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_DATABASE: exampledb
      MYSQL_USER: exampleuser
      MYSQL_PASSWORD: examplepass
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
  wordpress:
    depends_on:
      - mysql
    image: wordpress
    restart: always
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: mysql:3306
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD: examplepass
      WORDPRESS_DB_NAME: exampledb
    volumes:
      - wordpress:/var/www/html
volumes:
  mysql-data:
  wordpress:
```



Docker swarm

Docker stack - Práctica 1 (cont.)

Desplegar la solución del archivo docker-compose.yaml, abrir el navegador y visitar la dirección http://localhost:8080

```
$ docker stack deploy -c docker-compose.yaml wordpress
```

Listar los stacks

```
$ docker stack ls
```

Listar los servicios

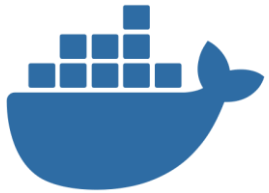
```
$ docker stack services wordpress
```

Lista las tareas

```
$ docker stack ps wordpress
```

Eliminar el stack

```
$ docker stack rm wordpress_mysql
```

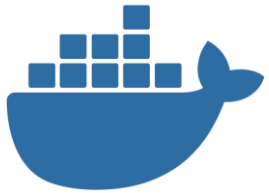


Docker swarm

Docker Secrets

Los secretos en docker:

- Se montan en el sistema de archivos en memoria dentro del contenedor.
- En los contenedores Linux se montan en /run/secrets/<secret_name>
- En los contenedores Windows se montan en c:\ProgramData\Docker\secrets.
- Solo el servicio al que se le otorga los accesos puede acceder al secreto.
- Cuando un contenedor se detiene, los secretos son desmontados de la memoria de este contenedor y eliminados de la memoria del nodo. En swarm, solo los nodos manager tienen acceso a los secretos encriptados.
- En swarm, si un nodo pierde conectividad mientras ejecuta un contenedor, éste sigue teniendo acceso a sus secretos, pero no recibirá las actualizaciones.
- Para actualizar los secretos, considere utilizar número de versión o la fecha al nombre del secreto.



Docker swarm

Docker Secrets - Comandos

Para crear un secreto:

```
$ docker secret create my_secret /path/to/secret/file
```

Para listar los secretos:

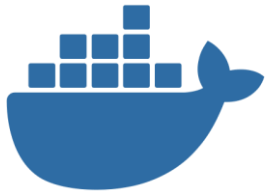
```
$ docker secret ls
```

Para ver el contenido de un secreto:

```
$ docker secret inspect my_secret
```

Para eliminar un secreto:

```
$ docker secret rm my_secret
```



Docker swarm

Docker Secrets - Práctica

- Crear un secreto desde un archivo que tenga la frase “Este es mi súper secreto”

```
$ docker secret create my_secret file.txt
```

- Verificar que el secreto ha sido creado correctamente:

```
$ docker secret ls
```

- Crear un servicio de redis que utilice el secreto

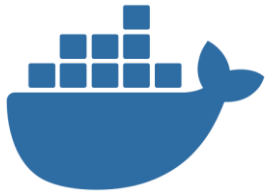
```
$ docker service create --name redis --secret my_secret redis:alpine
```

- Verificar que el servicio se esté ejecutando. Tomar nota del *task_id* y del *task_name*

```
$ docker service ps redis
```

- Obtener el id del contenedor de la tarea del servicio

```
$  docker ps --filter name=<task_name>.<task_id> -q
```



Docker swarm

Docker Secrets - Práctica

- Verificar que el servicio tiene acceso al secreto. El *container_id* es el valor obtenido en el paso anterior.

```
$ docker container exec <container_id> cat /run/secrets/my_secret
```

- Crear una nueva imagen a partir del contenedor en ejecución

```
$ docker commit <container_id> committed_redis
```

- Verificar que el secreto no existe en un contenedor a partir de la imagen anterior, el resultado puede dar un error de que no encuentra el archivo o no devolver el valor.

```
$ docker run --rm -it committed_redis cat /run/secrets/my_secret
```

- Quitar al servicio el acceso al secreto, validar que exista el secreto (obtener el nuevo *container_id*), luego eliminar el servicio y el secreto.

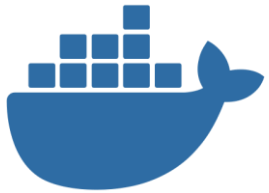
```
$ docker service update --secret-rm my_secret redis
$ docker container exec <container_id> cat /run/secrets/my_secret
$ docker service rm redis
$ docker secret rm my_secret
```

Docker swarm

Docker Secrets

Copiar en un archivo docker-compose.yaml el siguiente código:

```
version: "3.8"
services:
  db:
    image: mysql:5.7
    volumes:
      - mysql-data:/var/lib/mysql
    environment:
      MYSQL_DATABASE: exampledb
      MYSQL_USER: exampleuser
      MYSQL_PASSWORD_FILE: /run/secrets/db_password
      MYSQL_ROOT_PASSWORD_FILE:
/run/secrets/db_root_password
    secrets:
      - db_password
      - db_root_password
 wordpress:
    depends_on:
      - db
    image: wordpress
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD_FILE: /run/secrets/db_password
      WORDPRESS_DB_NAME: exampledb
    volumes:
      - wordpress:/var/www/html
    secrets:
      - db_password
secrets:
  db_password:
    file: ./db_password.txt
  db_root_password:
    file: ./db_root_password.txt
volumes:
  mysql-data:
  wordpress:
```



Docker swarm

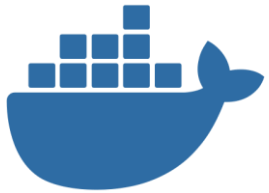
Docker Secrets - Práctica

- Crear los archivos de texto con los nombres *db_password.txt* y *db_root_password.txt*
- Crear una nueva imagen a partir del contenedor en ejecución

```
$ docker stack deploy –c docker-compose.yaml wordpress
```

- Verificar que wordpress esté funcionando en *http://localhost:8080/wp-admin/install.php*
- Eliminar el stock.

```
$ docker stack rm wordpress
```

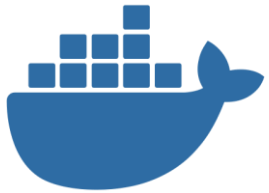


Docker swarm

Docker Configs

Las configuraciones en docker:

- Funcionan en la misma forma que los secretos, la diferencia está que no están encriptados cuando se transmite entre nodos (*pero sí cuando se almacenan en los nodos manager*) y que se montan directamente en el sistema de archivos del contenedor.
- Las configuraciones pueden ser compartido entre varios servicios y se puede usar junto con las variables de entorno.
- Para actualizar o dar marcha atrás en las configuraciones más fácilmente, considerar agregar un número de versión o fecha al nombre de la configuración.
- Tener en cuenta que los archivos de configuración son inmutables una vez han empezado a utilizarlos. Por lo que, si se desea actualizarlos, debe de crear uno distinto.



Docker swarm

Docker Config - Comandos

Para crear una configuración:

```
$ docker config create my_config /path/to/secret/file
```

Para listar configuraciones:

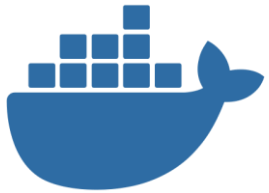
```
$ docker config ls
```

Para ver el contenido de una configuración:

```
$ docker config inspect my_config
```

Para eliminar una configuración:

```
$ docker config rm my_config
```



Docker swarm

Docker Configs - Práctica

- Crear una configuración desde un archivo que tenga la frase “Esta es una configuración”

```
$ docker config create my_config config_file.txt
```

- Verificar que la configuración ha sido creada correctamente

```
$ docker config ls
```

- Crear un servicio de redis que utilice la configuración. El destino se puede cambiar con la opción *target*

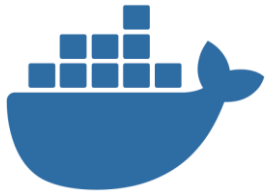
```
$ docker service create --name redis --config my_config redis:alpine
```

- Verificar que el servicio se esté ejecutando. Tomar nota del *task_id* y del *task_name*

```
$ docker service ps redis
```

- Obtener el id del contenedor de la tarea del servicio

```
$  docker ps --filter name=<task_name>.<task_id> -q
```



Docker swarm

Docker Configs - Práctica

- Verificar que el servicio tiene acceso a la configuración.

```
$ docker container exec <container_id> cat /my_config
```

- Intentar eliminar la configuración.

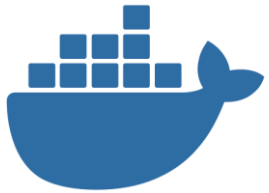
```
$ docker config rm my-config
```

- Actualizar el acceso a la configuración desde el servicio en ejecución actualizando la referencia.

```
$ docker service update --config-rm my_config redis
```

- Volver a verificar si aún existe la configuración, posteriormente eliminar el servicio y la configuración

```
$ docker container exec <container_id> cat /my_config
$ docker service rm redis
$ docker config rm my_config
```



Docker swarm

Docker Configs - Práctica

- Crear un archivo de configuración *ningx.conf*.

```
user www-data;
worker_processes 4;
pid /run/nginx.pid;
events {
   worker_connections 768;
}
http {
  upstream api {
    server api;
  }
  server {
    listen *:8000;
    location = /api {
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-Proto $scheme;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      rewrite /api/(.*) /$1 break;
      proxy_pass http://api;
    }
  }
}
```

- Crear una configuración proxy con el archivo

  $ docker config create proxy nginx.conf

- Revisar la configuración

  $ docker config inspect proxy

- Crear una red tipo overlay

  $ docker network create --driver overlay front

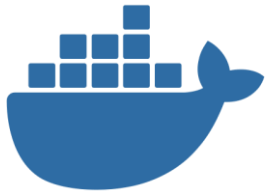
- Crear un servicio api

  $ docker service create --name api --network front edelpa/api

- Crear el proxy

  $ docker service create --name proxy --name proxy --network front --config src=proxy,target=/etc/nginx/nginx.conf -p 8000:8000 nginx:1.13.6

- Navegar *a http://localhost:8000/api*



Docker swarm

Docker Configs - Práctica

- Crear el siguiente *docker-compose.yaml*

```
version: "3.8"
services:
  proxy:
    image: nginx:1.13.6
    configs:
      - source: proxy
        target: /etc/nginx/nginx.conf
    ports:
      - 8000:8000
    networks:
      - front
  api:
    image: edelpa/api
    networks:
      - front
configs:
  proxy:
    external: true
networks:
  front:
```

- Desplegar con docker stack

  $ docker stack deploy -c docker-compose.yaml test-nginx

- Navegar a *http://localhost:8000/api*
- Crear una nueva configuración (*duplicar el anterior*)

  $ docker config create proxy-v2 nginx-v2.conf

- Actualizamos el archivo de configuración

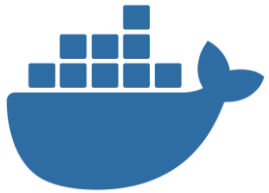
  $ docker service update --config-rm proxy --config-add src=proxy-v2,target=/etc/nginx/nginx.conf test-nginx_proxy

- Eliminamos todos los recursos

  $ docker stack rm test-nginx

  $ docker config rm proxy

  $ docker config rm proxy-v2



Docker

Contenido adicional

Seguridad de contenedores

Componentes de seguridad que establecerá el futuro de la contenerización, junto con prácticas recomendadas para la seguridad de los contenedores

Depurar remotamente un contenedor

En esta actividad adicional veremos como se puede depurar directamente desde un contenedor.

Seguridad de contenedores

Imágenes

- Protección de imágenes
  - Comience con fuentes de confianza:

    Deben utilizarse imágenes de contenedores oficiales, ya que son objeto de un mantenimiento continuo y de auditorías de seguridad. Deben obtenerse de fuentes fiables.
  - Verifique las firmas

    Antes del despliegue, confirme la integridad de las imágenes de contenedores y que están firmadas.
  - Escanear imágenes

    Las imágenes deben escanearse utilizando software de escaneo de imágenes para encontrar fallos, malware y problemas de configuración.
  - Hardening de imágenes:

    Para minimizar las posibles superficies de ataque, elimine los componentes, servicios y paquetes no utilizados de las imágenes de contenedores.

Seguridad de contenedores

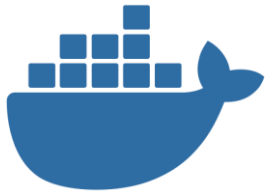
Registros

- Seguridad de los registros
  - Controles de acceso

    Para restringir quién puede cargar y descargar imágenes, aplique estrictos controles de acceso a los registros de contenedores.
  - Utilice registros privados

    Para regular la distribución y el acceso a las imágenes, piense en recurrir a registros privados de contenedores.
  - Habilite la autenticación

    Para evitar accesos no autorizados, haga necesario que los usuarios se autentiquen antes de acceder al registro.



Seguridad de contenedores

Despliegue

- Seguridad de la implantación
  - Límite los privilegios

    Abstenerse de ejecutar contenedores como root. Deben utilizarse siempre las cuentas de usuario con menos privilegios.
  - Aislamiento

    Utilizar herramientas de orquestación de contenedores para mantenerlos separados entre sí y del sistema anfitrión.
  - Segmentación de red

    Implemente políticas de red y cortafuegos para gestionar el tráfico entre contenedores y pods mediante el uso de la segmentación de red.
  - Protección en tiempo de ejecución

    Para vigilar la actividad sospechosa y los riesgos de seguridad en los contenedores, utilice soluciones de seguridad en tiempo de ejecución.

Seguridad de contenedores

Host y runtime

- Host y runtime
  - Proteja el sistema operativo anfitrión:

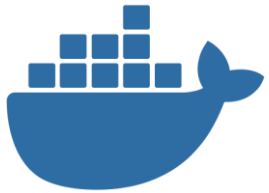
    Parchee y actualice el sistema operativo host con regularidad. Para resolver vulnerabilidades que puedan ser explotadas, aplique rápidamente actualizaciones de seguridad.
  - Implemente controles de acceso:

    Emplee controles de acceso y procedimientos de autenticación para impedir el acceso no autorizado al sistema host. Considere el uso de autenticación multifactor y la creación de contraseñas seguras para aumentar la seguridad.
  - Supervise la actividad del host:

    Utilice la supervisión a nivel de host para detectar actividades fuera de lo común o sospechosas. Las amenazas pueden identificarse con sistemas de detección de intrusiones (IDS) y sistemas de prevención de intrusiones (IPS).
  - Aplique prácticas de fortalecimiento:

    Para reducir la superficie de ataque, siga las técnicas recomendadas de endurecimiento del host. Reduzca los posibles puntos de explotación desactivando los servicios y funciones que no se utilicen.
  - Haga copias de seguridad periódicas

    Asegúrese de que se realizan copias de seguridad periódicas de los datos vitales del host. Tener copias de seguridad es beneficioso en caso de un evento de seguridad o una violación del sistema.



Seguridad de contenedores

Contenedores

- Contenedores

  Uso de contenedores ligeros y efímeros para reducir la superficie de ataque.

  - Adopte imágenes mínimas

    Para disminuir la superficie de ataque de los contenedores, utilice imágenes base simples.
  - Contenedores de un solo proceso

    Cree contenedores que sólo ejecuten un proceso para reducir la complejidad y los riesgos de seguridad.
  - Contenedores de vida corta

    Promueva la utilización de contenedores de vida corta para reducir la exposición a peligros
- Monitorización de la actividad de los contenedores
  - Logging

    Habilite el registro a nivel de contenedor para registrar actividades e incidentes de seguridad.
  - Logging centralizado

    Reúna los registros de los contenedores en un sistema centralizado para su análisis mediante el registro centralizado.
  - Monitorización en tiempo real

    Implemente la supervisión en tiempo real para identificar y responder rápidamente a los incidentes de seguridad.

Seguridad de contenedores

Secretos

- Gestión de secretos

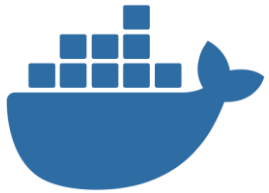
  Para distribuir y preservar los datos sensibles de forma segura, ponga en marcha una sólida solución de gestión de secretos.

  - Secretos cifrados:

    Para evitar el acceso no autorizado, cifre los secretos tanto en tránsito como en reposo.

  - Acceso basado en funciones:

    Implemente un control de acceso basado en funciones para limitar quién tiene acceso a los secretos



Docker

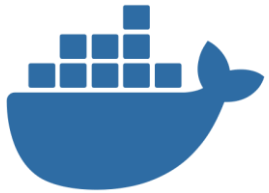
Contenido adicional

Seguridad de contenedores

Componentes de seguridad que establecerá el futuro de la contenerización, junto con prácticas recomendadas para la seguridad de los contenedores

Depurar remotamente un contenedor

En esta actividad adicional veremos como se puede depurar directamente desde un contenedor.



Depurar remotamente un contenedor

Práctica

- Para esta práctica es necesario Visual Studio Code y .NET SDK
- Desde el terminal de Visual Studio Code crear una carpeta “pr-dr” y ejecutar

```
$ dotnet new webapi --no-https –n SampleAPI
```

- Entrar en la carpeta (cd SampleAPI) y ejecutar $ dotnet run
- Ahora se puede acceder al endpoint a través de la url http://localhost:5275/WeatherForecast
- En Visual Studio Code, agregamos las extensiones de Docker y luego expandimos la paleta de comandos (***Crtl+Shift+P***) y buscamos *Docker: Add Docker Files to Workspace…*
- De la lista de opciones para Plataforma de Aplicaciones, seleccionamos *.NET: ASP.NET Core*
- De la lista de opciones de Sistema Operativo, seleccionamos *Linux.*
- En el valor de puerto, ingresamos el valor del puerto donde deseamos que se despliegue
- Respondemos que sí cuando pregunta incluir los archivos de *Docker Compose*

Docker – depurando remotamente

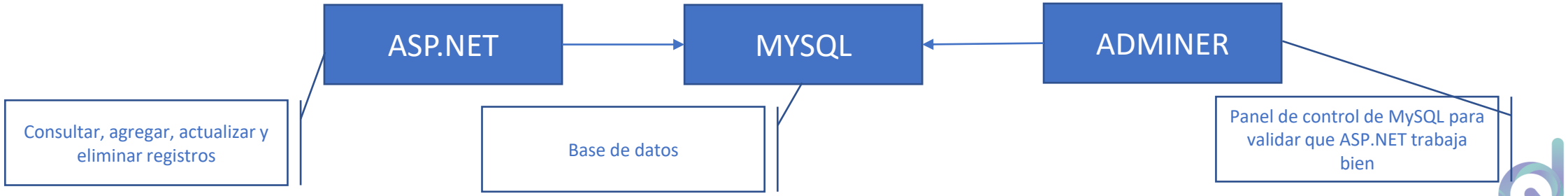
Práctica

- Esta extensión creará los archivos de *Dockerfile*, *.dockerignore*, *docker-compose.yaml* y *docker-compose.debug.yml*
- Luego ejecutar $ docker compose up -d
- Volver a acceder al endpoint a través de la url http://localhost:5275/WeatherForecast
- Desde el tab de Depuración, seleccionar de la lista de configuración, agregar configuración y seleccionamos la opción *.NET Core Attach (Preview)* y automáticamente se agregará la configuración a *launch.json*
- Click derecho en el archivo *docker-compose.yml* y seleccionar *Docker compose up*, aparecerá un menú preguntando si se desea copiar el depurador al contenedor, respondemos que sí. Si vamos con el archivo *docker-compose.debug.yml*, el depurados se copiará automáticamente.
- Volver a acceder al endpoint a través de la url http://localhost:5275/WeatherForecast
- En el menú desplegable seleccionamos *Docker .NET Core Attach (Preview)*, escogemos el grupo y el contenedor. Agregamos un punto de ruptura en el código y volvemos a cargar la url anterior. El depurador debe detenerse en el punto de ruptura.

Docker – depurando remotamente

Práctica libre

- A partir del proyecto en .NET de nombre *ASP.NET_WebAPI6*
  - Crear el archivo dockerfile desde visual studio code.
  - Agregar las variables de entorno dbServer, dbName, dbPassword para configurar estos valores en docker-compose
  - La solución de docker compose deberá de incluir un mySql donde el proyecto creará automáticamente las tablos y agregará los registros.
  - Adicionalmente deberá de incluir un adminer para validar que los registros están realmente en la bd
  - Para probar el proyecto una vez desplegado. Deberá de cargar la página en el puesto configurado en el docker file. En esta página encontrará los métodos para listar (GET), agregar (POST), modificar (UPDATE) y eliminar (DELETE) registros en la base de datos.
  - El proyecto carga los valores desde un archivo appsettings.json si no se encuentran los valores en variables de entorno. Por esto, luego modificará la solución de docker-compose para que en lugar de que cargue estos valores a través de variables de entorno, lo realice desde un secreto.



Muchas gracias

datahack
INSPIRING BIG DATA