Docker

¿Qué es?

Es una tecnología que permite encapsular aplicaciones y sus entornos en contenedores individuales. Múltiples contenedores pueden correr en la misma máquina ya que cada uno de ellos se ejecuta en un entorno protegido de resto, lo que se conoce como `sandbox`. Es la evolución lógica del alojamiento web, ya que se ha pasado de contratar máquinas físicas a hacerlo con máquinas lógicas. Este movimiento permite alcanzar mayor fiabilidad, accesibilidad, escalabilidad y seguridad.

Tipos de hosting

En los orígenes de Internet allá por los años 90, las compañías alojaban los sitios webs en ordenadores donde compartían los recursos del mismo (CPU, RAM, HD, …) con cientos de sitios webs. Era la época del hosting compartido. Después se pasó al Co-located hosting o Housing, donde el sitio web compra o alquila un espacio físico dentro de un centro de datos donde el cliente coloca su propio ordenador. Por el contrario, el Self-hosting consiste en la práctica de correr y mantener un sitio web usando un servidor web privado. Por último están los Data Centers

Data centers

Los beneficios de contratar un data center son numerosos ya que proveen una buena conexión a Internet, copias de seguridad, servicio 24/4/365, clima controlado, protección anti incendios, etc. Este nivel de confianza es difícil de alcanzar para la mayoría de las empresas por lo que se recurre a empresas punteras como Google (**Google Cloud Platform**), Microsoft (**Azure**), Amazon (**Amazon Web Services - AWS**) y otros actores.

Usar virtualización para economizar el uso de recursos

La virtualización de servidores permite ejecutar múltiples sistemas operativos en un solo servidor físico por medio de máquinas virtuales que ofrecen un elevado

Figure 1: Figure 1.3 – A server room at CERN (Switzerland)

rendimiento. Entre las ventajas principales, se incluyen las siguientes:

- Mayor eficiencia del entorno de TI
- Reducción de los costes operativos
- Implementación más rápida de las cargas de trabajo
- Mejora del rendimiento de las aplicaciones
- Mayor disponibilidad del servidor
- Eliminación de la complejidad y la proliferación de servidores

Por ejemplo, una máquina con 128 GB de RAM y 8 CPUs puede hacer con software de virtualización 4 máquinas con 32 GB de RAM y 2 CPUs cada una y cada una de estas puede correr su propio sistema operativo.

La virtualización ya empezó en los años 60 por IBM aunque realmente emulaban por software el juego de instrucciones de las máquinas. En 1998 ya se funda WMWare y VirtualBox fue liberado como open source en 2007. Los fabricantes de CPUS basadas en x86, empezaron a correr máquinas o software a una velocidad prácticamente nativa. Pero para ello hace falta un `hypervisor`. El `hypervisor`muestra la máquina virtual al SO elegido y después gestiona los recursos y la ejecución de las mismas en el tiempo.

Según la Wikipedia

> Un hipervisor (en inglés hypervisor) o monitor de máquina virtual (virtual machine monitor) es una capa de software para realizar una virtualización de hardware que permite utilizar, al mismo tiempo, diferentes sistemas operativos (sin modificar o modificados, en el caso de paravirtualización) en una misma computadora. Es una extensión de un término anterior, «supervisor», que se aplicaba a los kernels de los sistemas operativos de computadora.

Desde el punto de vista de la ciberseguridad, hay una vulnerabilidad llamada VM Scape que permite a un atacante salir de la máquina virtual e interactuar sobre el sistema operativo anfitrión tal como se de describe en la vulnerabilidad CVE-2008-0923

Gracias a estas técnicas de virtualización, hoy en día es muy sencillo escalar aplicaciones tanto vertical como horizontalmente, pues es tan fácil como entrar en el panel de control de pongamos AWS (Amazon Web Services) y contratar más RAM (escalado vertical) o contratar más servidores (escalado horizontal). Hay que prever los picos que pueda tener nuestra aplicación web. Por ejemplo, una dedicada a deportes de invierno necesitará mas potencia durante en el invierno y en el resto del año se pueden disminuir los recursos y, por tanto, reducir el coste.

En estas instalaciones en la nube incluso se puede mover una máquina virtual en *caliente* a otra máquina física sin la interrupción del servicio. Esta característica se denomina Teleport

Usar contenedores para optimizar aún más los recursos de los Data Center

Docker hace un uso inteligente de la virtualización a nivel del SO soportando múltiples contenedores ejecutándose en una sola máquina, cada uno de ellos es una instancia de una imagen y, por defecto, están aislados de la máquina anfitrión (host) y entre ellos. Este tipo de aislamiento se denomina sandboxing

Como los contenedores comparten su kernel linux con el anfitrión no se necesita instalar un sistema operativo completo como sí es necesario en una máquina virtual. Otra diferencia es que no bloquean todo un recurso sino que lo comparten. Por ejemplo, dos contenedores idénticos usan la RAM del anfitrión en vez de un bloque de la misma configurado antes de poner en marcha la máquina virtual.

Sandboxing o aislamiento de procesos

> Según la Wikipedia
>
> In computer security, a **sandbox** is a security mechanism for separating running programs, usually in an effort to mitigate system failures and/or software vulnerabilities from spreading. It is often used

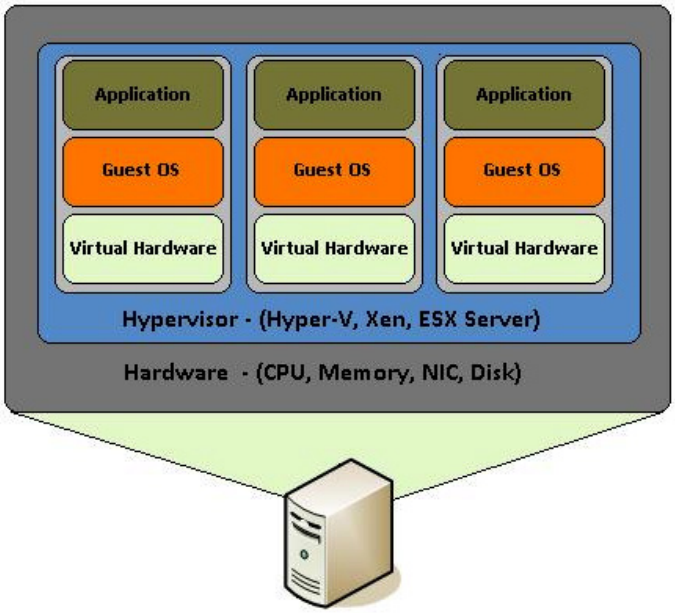

Figure 2: Figure 1.4 – Hardware virtualization

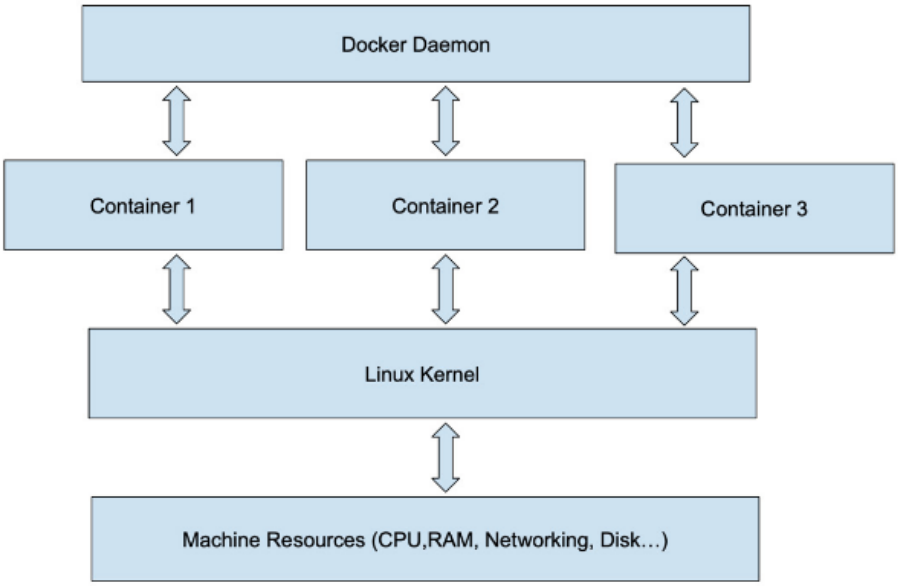

Figure 3: Docker

> to execute untested or untrusted programs or code, possibly from unverified or untrusted third parties, suppliers, users or websites, without risking harm to the host machine or operating system.[1] A sandbox typically provides a tightly controlled set of resources for guest programs to run in, such as storage and memory scratch space. Network access, the ability to inspect the host system or read from input devices are usually disallowed or heavily restricted.

Este término también se utiliza en el terreno del desarrollo de software. Por ejemplo, al hablar de un entorno de desarrollo y uno de pruebas, ya estamos trabajando con sandboxes pues cada uno de ellos nos aísla del servidor de producción. Y evidentemente, en ciberseguridad pues nos permite ejecutar un programa en un espacio cerrado y limitado y de esta forma no estamos dando acceso al resto de recursos sensibles del equipo.

También existen los propios programas Sandbox (como Sandboxie) cuyo propósito es ejecutar una aplicación en un entorno estando que, en casa de error, no afecte a los demás programas.

Incluso hay Sistemas Operativos como Qubes OS que se autodefine como "un sistema operativo razonablemente seguro" basados completamente en la virtualización

> Qubes OS is a free and open-source, security-oriented operating system for single-user desktop computing. Qubes OS leverages Xen-based virtualization to allow for the creation and management of isolated compartments called qubes.

Usar Docker para desarrollo.

Uno de los problemas con los que se enfrenta el desarrollador de software es la cantidad ingente de software (y versiones distintas del mismo) que necesita tener instalado en su equipo de desarrollo. Este problema desaparece con los contenedores pues aíslan al software del sistema operativo. Simplemente se borrar el contenedor y esto hace que automáticamente desaparezca todo el software instalado en el mismo.

Realmente Docker se ejecuta como una máquina virtual Linux en aquellos SO que no están basados en Linux y todo ello de forma transparente. Es como una *headless* Virtual Machine.

Por ejemplo, podemos correr Apache en un contenedor sin instalarlo en la máquina de trabajo. O podemos correr en un contenedor la versión 8 de Node.js y la versión 10 en otro. Esto es un problema obvio si no usamos contenedores.

Cuando ya tienes preparado tu container para desplegarlo, lo único que debes hacer es un push a cualquier hosting que admita contenedores, cono Docker Hub.

Este es un repositorio excelente para encontrar contenedores ya preparados para usar en tu proyecto. Por ejemplo de Apache, de PHP, de MongoDB y MySql, etc.

Para desarrollar los propios contenedores, es bastante habitual, empezar con un contenedor llamado Alpine Linux y depués se pueden ir añadiendo paquetes con `apt`

En esta práctica se ejecutará un contenedor popular, gratuito y ligero y se explorarán los fundamentos de cómo funcionan los contenedores, cómo el Motor Docker ejecuta y aísla los contenedores entre sí.

Conceptos de este ejercicio:

- Motor Docker
- Contenedores e imágenes
- Registros de imágenes y Docker Hub
- Aislamiento del contenedor

Correr tu primer contenedor

¡Es hora de ensuciarse las manos! Como con todas las cosas técnicas, una aplicación de "hola mundo" es un buen lugar para empezar. Escribe el código de abajo para ejecutar tu primer contenedor Docker:

```
docker container run hello-world
```

Eso es todo: su primer contenedor. La salida del contenedor "hola *mundo*" te dice un poco sobre lo que acaba de pasar. Esencialmente, el motor Docker que funciona en tu terminal trató de encontrar una **imagen** llamada `hola-mundo`. Como acabas de empezar no hay imágenes almacenadas localmente (`Unable to find image...`) así que el motor Docker va a su **registro** Docker por defecto, que es Docker Hub, para buscar una imagen llamada `hola-mundo`. Encuentra la imagen allí, la descarga, y luego la corre en un contenedor. Y la única función de "hello-world" es emitir el texto que ves en tu terminal, después de lo cual el contenedor sale.

Si estás familiarizado con las máquinas virtuales, puedes pensar que esto es más o menos como ejecutar una máquina virtual, excepto con un repositorio central de imágenes de máquinas virtuales. Y en este simple ejemplo, eso es básicamente cierto. Pero a medida que vayas realizando estos ejercicios, empezarás a ver importantes formas en que Docker y los contenedores difieren de las máquinas virtuales. Por ahora, la explicación simple es esta:

- La VM es una abstracción de *hardware*: toma las CPUs físicas y la RAM de un host, y las divide y comparte entre varias máquinas virtuales más

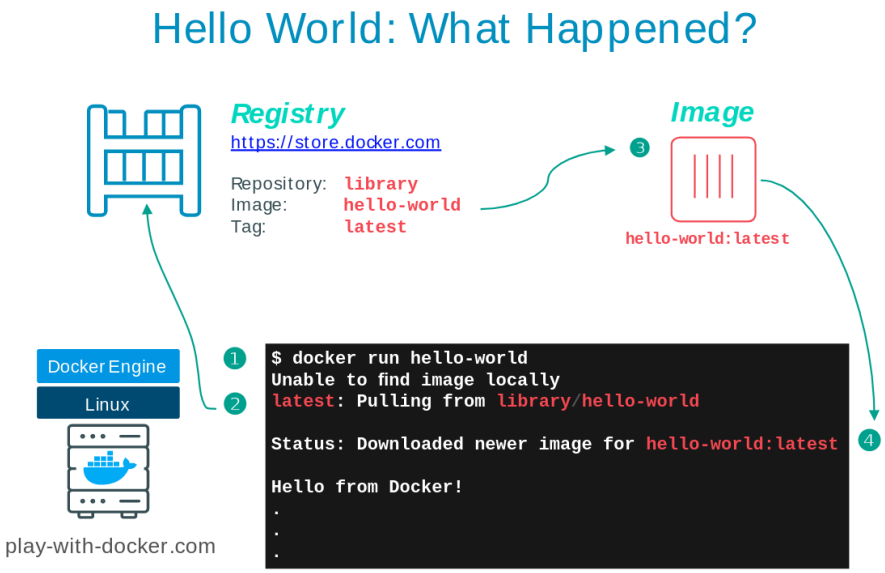

Figure 4: Hello world explainer

pequeñas. Hay un sistema operativo y una aplicación que se ejecuta dentro de la máquina virtual, pero el software de virtualización no suele tener un conocimiento real de eso.

- Un contenedor es una abstracción de *la* aplicación: el foco está realmente en el SO y la aplicación, y no tanto en la abstracción del hardware.

Muchos clientes usan hoy en día tanto máquinas virtuales como contenedores en sus entornos y, de hecho, pueden ejecutar contenedores dentro de las máquinas virtuales.

Imágenes del Docker

En el resto de este laboratorio, vais a ejecutar un contenedor Alpine Linux. Alpine es una distribución Linux ligera, por lo que es rápida de bajar y ejecutar, lo que la convierte en un popular punto de partida para muchas otras imágenes.

Para empezar, vamos a ejecutar lo siguiente en nuestra terminal:

```
docker image pull alpine
```

El `pull` El comando obtiene la **imagen** alpina del **registro del Docker** y la guarda en nuestro sistema. En este caso el registro es **Docker Hub**.

Puedes usar el `docker image` para ver una lista de todas las imágenes de su sistema.

```
docker image ls
```

```
REPOSITORY              TAG                 IMAGE ID            CREATED             VIRTUAL
alpine                 latest              c51f86c28340        4 weeks ago         1.109 MB
hello-world             latest              690ed74de00f        5 months ago        960 B
```

Docker Container Run

¡Grandioso! Ahora vamos a ejecutar un **contenedor** Docker basado en esta imagen. Para hacer eso vas a usar el comando `docker container run`

```
docker container run alpine ls -l
```

```
total 48
drwxr-xr-x    2 root     root          4096 Mar  2 16:20 bin
drwxr-xr-x    5 root     root           360 Mar 18 09:47 dev
drwxr-xr-x   13 root     root          4096 Mar 18 09:47 etc
drwxr-xr-x    2 root     root          4096 Mar  2 16:20 home
drwxr-xr-x    5 root     root          4096 Mar  2 16:20 lib
......
......
```

Mientras que la salida del comando `ls` puede no ser muy emocionante, entre bastidores han ocurrido bastantes cosas. Cuando llamas a `run` el cliente Docker encuentra la imagen (`Alpine` en este caso), crea el contenedor y luego ejecuta un comando en ese contenedor. Cuando se ejecuta `docker container run alpine`… proporcionaste una orden…`ls -l`), así que Docker ejecutó este comando dentro del contenedor para el que vio el listado de directorios. Después de que el `ls` comando termina, se cierra el contenedor.

El hecho de que el contenedor saliera después de ejecutar nuestro comando es importante, como empezarás a ver. Intentemos algo más emocionante. Escribe lo siguiente:

```
docker container run alpine echo "hello from alpine"
```

Y deberías obtener la siguiente salida:

```
hello from alpine
```

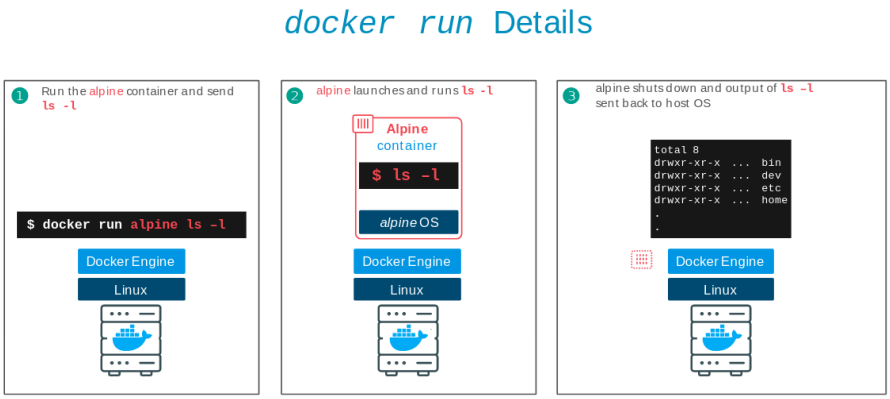

Figure 5: docker run explainer

En este caso, el cliente de Docker ejecutó obedientemente el `echo` dentro de nuestro contenedor `Alpine` y luego salió. Si te has dado cuenta, todo eso pasó muy rápido y de nuevo nuestro contenedor salió. Como verás en unos pocos pasos más, el comando `echo` se ejecutó en una instancia de contenedor separada. Imagina que arrancas una máquina virtual (VM), ejecutas un comando y luego la cierras; tardaría un minuto o dos para arrancar la VM antes de ejecutar el comando. Una VM tiene que emular una pila de hardware completa, arrancar un sistema operativo y luego lanzar su aplicación, es un entorno de *hardware* virtualizado. Los Docker Containers funcionan en la capa de aplicación, por lo que se saltan la mayoría de los pasos que requieren las VMs y sólo ejecutan lo que se requiere para la aplicación. Ahora ya sabes por qué dicen que los contenedores son rápidos!

Intenta otro comando.

```
docker container run alpine /bin/sh
```

Espera, ¡no ha pasado nada! ¿Es eso un error? No! De hecho, algo pasó. Iniciaste una tercera instancia del contenedor alpino y ejecutó el comando `/bin/sh` y luego salió. No suministró ningún comando adicional a `/bin/sh` así que sólo lanzó la cáscara, salió de la cáscara y luego detuvo el contenedor. Lo que se podría haber *esperado* era un shell interactivo donde se podían escribir algunos comandos. Docker tiene una facilidad para eso al agregar una bandera para ejecutar el contenedor en una terminal interactiva. Para este ejemplo, escribe lo siguiente:

```
docker container run -it alpine /bin/sh
```

Ahora estás dentro del contenedor ejecutando un shell de Linux y puedes probar algunos comandos como `ls -l`, `uname -a` y otros. Ten en cuenta que `Alpine` es un pequeño sistema operativo Linux, por lo que pueden faltar varios comandos. Sal del shell y del contenedor tecleando el comando `exit`.

Ok, dijimos que habíamos ejecutado cada uno de los comandos anteriores en un contenedor separado. Podemos ver estas instancias usando el comando `docker container ls` que te muestra todos los contenedores que están funcionando actualmente:

```
docker container ls
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
```

Como no hay contenedores en funcionamiento, se ve una línea en blanco. Intentemos una variante más útil: `docker container ls -a`

```
docker container ls -a
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
36171a5da744        alpine              "/bin/sh"                5 minutes ago       Exited
a6a9d46d0b2f        alpine              "echo 'hello from alp"   6 minutes ago       Exited
ff0a5c3750b9        alpine              "ls -l"                  8 minutes ago       Exited
c317d0a9e3d2        hello-world         "/hello"                 34 seconds ago      Exited
```

Lo que ves ahora es una lista de todos los contenedores que has ejecutado. Notar que el `STATUS` muestra que estos contenedores salieron hace algún tiempo.

Aquí está la misma salida del `docker container ls -a` que se muestra en forma de diagrama (tenga en cuenta que las identificaciones y los nombres de los contenedores serán diferentes):

Tiene sentido pasar algún tiempo poniéndose cómodo con el comandos `docker run` . Para saber más sobre `run` usa `docker container run --help` para ver una lista de todas las banderas que apoya. A medida que avanzas, veremos algunas variantes más de `docker container run` pero siéntete libre de experimentar aquí antes de proceder.

Aislamiento del contenedor

En los pasos anteriores ejecutamos varios comandos a través de instancias de contenedores con la ayuda de `docker container run`. El comando `docker container ls -a` nos mostró que había varios contenedores en la lista. ¿Por qué hay tantos contenedores en la lista si todos son de la imagen `Alpine`?

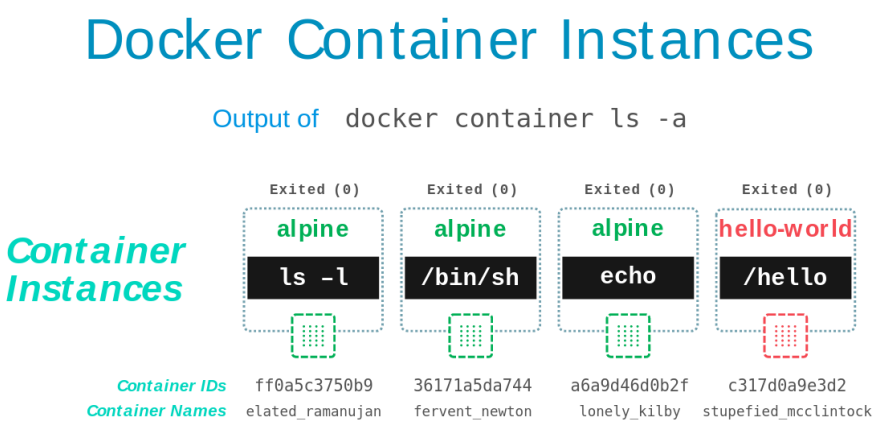

Figure 6: Docker container instances

¡Este es un concepto de seguridad crítico en el mundo de los contenedores Docker! Aunque cada uno de los comandos `docker container run` usaron la misma ***imagen*** `Alpine` cada ejecución fue separada, aislada en un ***contenedor***. Cada contenedor tiene un sistema de archivos separado y se ejecuta en un espacio de nombres diferente; por defecto, un contenedor no tiene forma de interactuar con otros contenedores, ni siquiera con los de la misma imagen. Intentemos otro ejercicio para aprender más sobre el aislamiento.

```
docker container run -it alpine /bin/ash
```

El `/bin/ash` es otro tipo de shell disponible en la imagen `Alpine`. Una vez que el contenedor se lanza y estás en el símbolo de comando del contenedor, escriba los siguientes comandos:

```
echo "hello world" > hello.txt
```

```
ls
```

El primero `echo` crea un archivo llamado “hello.txt” con las palabras “hello world” en su interior. El segundo comando le da un listado de los archivos y debería mostrar su recién creado archivo “hello.txt”. Ahora escribe `exit` para dejar este contenedor.

Para mostrar cómo funciona el aislamiento, haz lo siguiente:

```
docker container run alpine ls
```

Es el mismo `ls` que usamos en el interior del shell interactivço del contenedor, pero esta vez, ¿te das cuenta de que tu archivo "hello.txt" ha desaparecido? ¡Eso es aislamiento! El comando se ejecutó en una nueva y separada *instancia*, aunque está basado en la misma *imagen*. La 2ª instancia no tiene forma de interactuar con la 1ª instancia porque el Docker Engine los mantiene separados y no hemos configurado ningún parámetro extra que permita que estas dos instancias interactúen.

En el trabajo diario, los usuarios de Docker aprovechan esta característica no sólo por seguridad, sino para probar los efectos de hacer cambios en la aplicación. El aislamiento permite a los usuarios crear rápidamente copias de prueba separadas y aisladas de una aplicación o servicio y hacerlas funcionar en paralelo sin interferir unas con otras. De hecho, existe todo un ciclo de vida en el que los usuarios toman sus cambios y los trasladan a la producción utilizando este concepto básico y las capacidades incorporadas de Docker Enteprise.

Ahora mismo, la pregunta obvia es "¿cómo vuelvo al contenedor que tiene mi archivo 'hello.txt'?"

Una vez más ejecuta el

```
docker container ls -a
```

de nuevo y deberías ver una salida similar a la siguiente:

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
36171a5da744        alpine              "ls"                     2 minutes ago       Exited
3030c9c91e12        alpine              "/bin/ash"               5 minutes ago       Exited
a6a9d46d0b2f        alpine             "echo 'hello from alp"    6 minutes ago       Exited
ff0a5c3750b9        alpine             "ls -l"                   8 minutes ago       Exited
c317d0a9e3d2        hello-world         "/hello"                 34 seconds ago      Exited
```

Gráficamente, esto es lo que pasó en nuestro Motor Docker:

El contenedor en el que creamos el archivo "hello.txt" es el mismo en el que usamos el `shell ash`, que podemos ver listada en la columna de "COMANDO". El número de identificación del contenedor *de* la primera columna identifica de forma exclusiva esa instancia de contenedor en particular. En la salida de la muestra anterior el ID del contenedor es `3030c9c91e12`. Podemos usar un comando ligeramente diferente para decirle a Docker que ejecute esta instancia específica de contenedor. Intenta escribir:

```
docker container start <container ID>
```

- **Consejo profesional:** En lugar de usar la identificación del contenedor completo, puedes usar solo los primeros caracteres, siempre y cuando

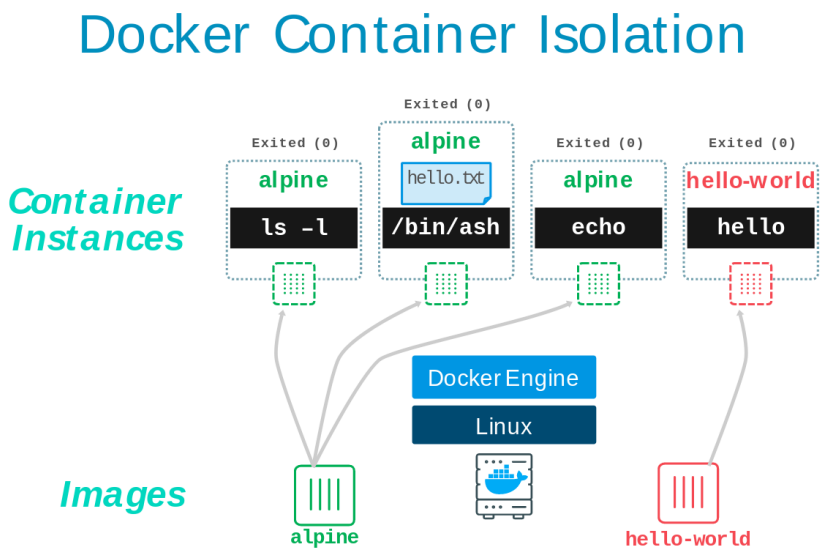

Figure 7: Docker container isolation

sean suficientes para identificar un contenedor de forma única. Así que podríamos usar simplemente "3030" para identificar la instancia de contenedor en el ejemplo anterior, ya que ningún otro contenedor de esta lista comienza con estos caracteres.

Ahora usa el `docker container ls` para listar los contenedores en funcionamiento.

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
3030c9c91e12        alpine              "/bin/ash"               2 minutes ago       Up 14
```

Notar esta vez que nuestra instancia de contenedores sigue funcionando. Usamos el `shell ash` esta vez, así que en lugar de salir simplemente como lo hizo `/bin/sh` antes, `ash` espera un comando. Podemos enviar un comando al contenedor para que se ejecute usando el `exec` de la siguiente manera:

```
docker container exec <container ID> ls
```

Esta vez obtenemos un listado de directorios y muestra nuestro archivo "hello.txt" porque usamos la instancia de contenedor donde creamos ese archivo.

Ahora estás empezando a ver algunos de los conceptos importantes de los contenedores. En el próximo ejercicio empezaremos a ver cómo puedes crear tus

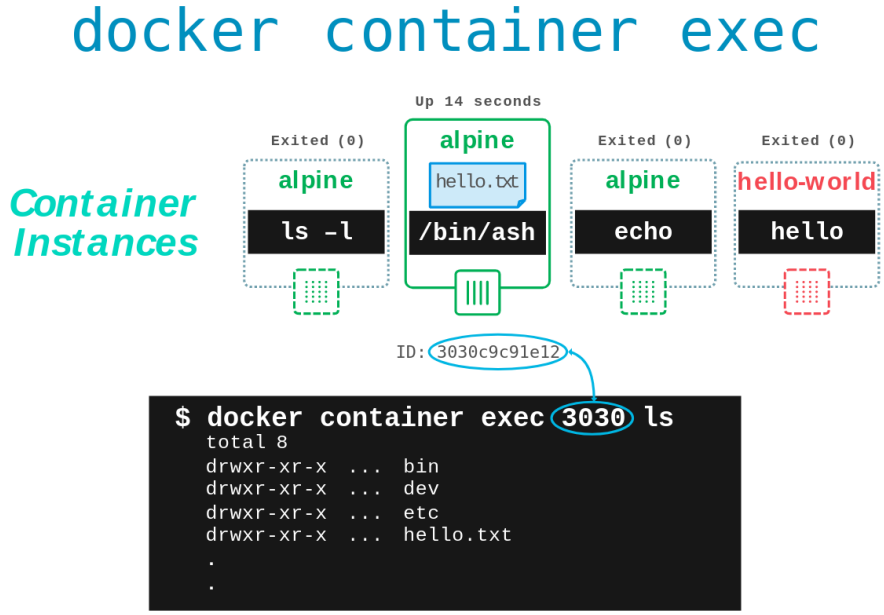

Figure 8: Docker container exec command

propias imágenes Docker y cómo usar un archivo Docker para estandarizar las imágenes de tal manera que puedas crear imágenes más grandes y complejas de una manera simple y automatizada.

Terminología

En la última sección, viste un montón de jerga específica de Docker que podría ser confusa para algunos. Así que antes de ir más lejos, vamos a aclarar algunos términos que se utilizan con frecuencia en el ecosistema de Docker.

- *Imágenes* - El sistema de archivos y la configuración de nuestra aplicación que se utilizan para crear contenedores. Para saber más acerca de una imagen Docker, ejecuta `docker image inspect alpine`. En la demostración anterior, usaste el `docker image pull` para descargar la imagen `alpine`. Cuando ejecutaste el comando `docker container run hello-world` también hiciste un `docker image pull` entre bastidores para descargar la imagen de **hola mundo**.
- *Contenedores* - Ejecutan instancias de imágenes de Docker - los contenedores ejecutan las aplicaciones reales. Un contenedor incluye una aplicación y todas sus dependencias. Comparte el núcleo con otros contenedores, y se ejecuta como un proceso aislado en el espacio de usuario en el

sistema operativo anfitrión. Creaste un contenedor usando `docker run` lo cual hiciste usando la imagen `alpine` que descargaste. Se puede ver una lista de contenedores en funcionamiento usando el `docker container ls` …comando.

- Demonio *Docker* - El servicio de fondo que se ejecuta en el host que gestiona la construcción, ejecución y distribución de los contenedores Docker.
- Cliente *Docker* - La herramienta de línea de comandos que permite al usuario interactuar con el demonio Docker.
- Docker*Hub* - La tienda es, entre otras cosas, un registro de imágenes Docker. Puedes pensar en el registro como un directorio de todas las imágenes Docker disponibles.