

Introducción a Docker

UD 01. Introducción a los contenedores y a Docker



Fons Social Europeu

L'FSE inverteix en el teu futur

Autor: Sergi García Barea

Actualizado Marzo 2022

Licencia



Reconocimiento – NoComercial - CompartirIgual (BY-NC-SA): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

Importante

Atención

Interesante

ÍNDICE DE CONTENIDO

1. Introducción	3
2. Conceptos previos	3
2.1. Virtualización	3
2.2. ¿Qué es una máquina virtual?	3
2.3. ¿Qué es una máquina virtual de proceso?	3
2.4. ¿Qué es un hipervisor?	4
3. Contenedores	4
3.1. ¿Qué son los contenedores?	4
3.2. Analogía con contenedores de transporte marítimo	5
3.3. Contenedores para desarrollo y despliegue de aplicaciones	5
3.4. Contenedores para despliegue de servicios	5
3.5. Ventajas e inconvenientes del uso de contenedores	6
3.6. En resumen ¿Cuándo es adecuado usar contenedores?	6
4. Contenedores en sistemas Linux	7
4.1. ¿Es nuevo el concepto de entornos privados en sistemas Unix?	7
4.2. Sistemas privados modernos en Linux: contenedores	7
4.3. ¿Cómo funcionan los contenedores modernos en Linux?	7
4.4. ¿Puedo poner en marcha un contenedor Linux "A mano"?	7
4.5. Los contenedores Linux ¿Pueden funcionar en sistemas como Windows o MacOS?	8
5. Contenedores Docker	8
5.1. ¿Qué es Docker?	8
6. Docker en sistemas Windows y MacOS	9
7. Docker corriendo contenedores Windows Server Core y contenedores MacOS	10
8. Conclusión	10
9. Bibliografía	10
10. Licencias de elementos externos utilizados	10

UD01. INTRODUCCIÓN A LOS CONTENEDORES Y A DOCKER

1. INTRODUCCIÓN

En esta unidad realizaremos una introducción al concepto de contenedores. Nos centraremos en contenedores Linux y en concreto en la tecnología de Docker.

2. CONCEPTOS PREVIOS

2.1 Virtualización

La virtualización es un conjunto de tecnologías de hardware y software que permiten la abstracción de hardware, creando así la “ilusión” de administrar recursos virtuales como si fueran recursos reales, de forma transparente para los usuarios.

La virtualización es muy utilizada para el despliegue de sistemas, desarrollo de software, análisis de malware, escalado horizontal, etc. Ya que es relativamente sencilla de implementar y puede ahorrar significativamente costes (consumo de energía, mantenimiento, etc.)

2.2 ¿Qué es una máquina virtual?

A veces, necesitamos probar un nuevo sistema operativo, una determinada configuración, probar a desplegar un software, etc. pero no está disponible para ello una máquina real. La creación de una máquina virtual utilizando técnicas de virtualización es la solución a este problema.

De este modo, una máquina virtual permite simular una máquina (con su sistema operativo) y ejecutar programas como si estuvieran utilizando una máquina real e independiente.

Para la creación de máquinas virtuales generalmente existen varios tipos de tecnologías:

- Máquinas virtuales de proceso.
- Hipervisores.
- Contenedores. **Docker se engloba en esta categoría.**

2.3 ¿Qué es una máquina virtual de proceso?

Las máquinas virtuales de proceso, son un tipo de máquinas virtuales que permiten ejecutar un programa diseñado para un sistema operativo/arquitectura concreta (distinta de la máquina actual), como un proceso más de nuestra máquina actual.

Esto se consigue implementando una máquina virtual de proceso que emula la arquitectura necesaria. Teóricamente, podremos lanzar nuestro programa en cualquier sistema que tenga la máquina virtual de proceso implementada.

Algunos de los principales ejemplos de este tipo de virtualización son:

- Máquina virtual de Java (JVM): ejecuta los bytecodes de Java en cualquier sistema y arquitectura que la tenga implementada.
- Wine: ejecutar aplicaciones Windows en otros sistemas operativos.

2.4 ¿Qué es un hipervisor?

Un hipervisor, es una máquina virtual que simula total o parcialmente un hardware de una máquina, permitiendo la instalación de distintos sistemas operativos (por ejemplo, virtualizar un sistema Windows 10 Home en una máquina real Linux).

Algunos softwares conocidos que implementan un hipervisor son: Virtualbox, VMWare, emuladores de consolas, etc.

Para saber más: <https://es.wikipedia.org/wiki/Hipervisor>

3. CONTENEDORES

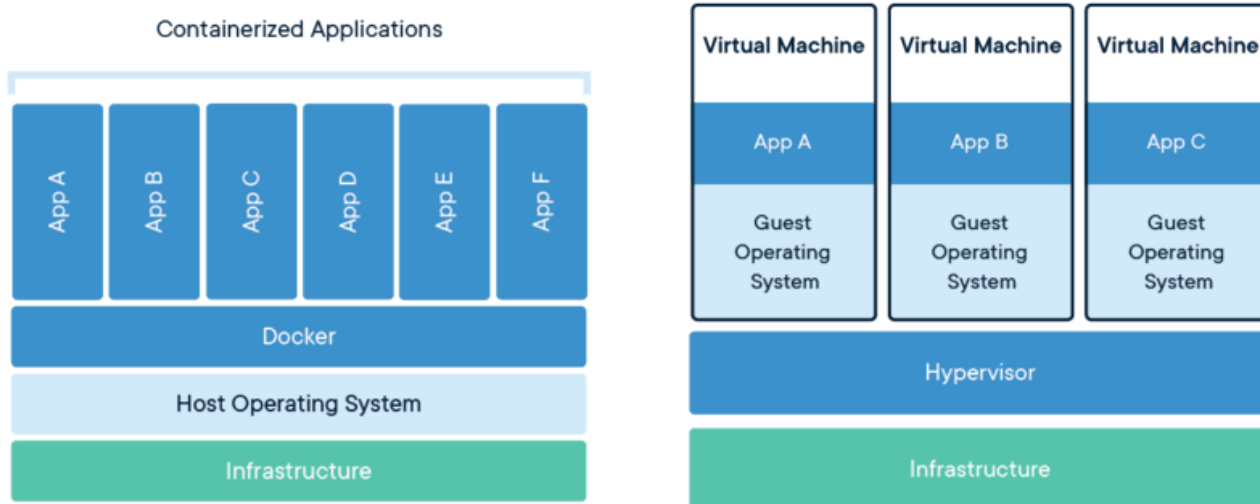
3.1 ¿Qué son los contenedores?

Los contenedores son una tecnología de virtualización, que al contrario que un hipervisor (que trata de emular un sistema completo), utiliza el sistema base de la máquina anfitrión y actúa realmente como un “entorno privado” que comparte recursos con el sistema anfitrión, sin virtualizar el hardware completo. En concreto, los contenedores suelen tener entornos privados aislados **a nivel de procesos, memoria, sistema de ficheros y red**.

Técnicamente, los contenedores son un tipo de virtualización englobada en lo que se llama “OS Level virtualization”. Para saber más: https://en.wikipedia.org/wiki/OS-level_virtualization

! Atención: esto implica, que de forma nativa, no puedes ejecutar un contenedor en un sistema operativo distinto del que utiliza la tecnología de contenedores.

La siguiente imagen puede ayudarnos a entender el concepto de contenedor.



Fuente imagen: <https://commons.wikimedia.org/wiki/File:Docker-containerized-and-vm-transparent-bg.png>

A la derecha observamos el funcionamiento de un hipervisor, encargado de virtualizar el hardware y donde cada máquina virtual tiene su propio sistema operativo. A la izquierda, observamos un sistema de contenedores, donde no existe esa virtualización del hardware y cada contenedor es un entorno privado.

3.2 Analogía con contenedores de transporte marítimo



Fuente imagen: <https://www.flickr.com/photos/68359921@N08/50125348052/>

Para facilitar la comprensión del funcionamiento de los contenedores, vamos a hacer una analogía con los contenedores de transporte marítimo:

- Los contenedores de transporte marítimo, deben cumplir unos estándares (tamaño, peso y forma) para ser transportados.
 - Lo mismo ocurre con los contenedores en virtualización. Mientras cumplan un estándar, pueden ser virtualizados en cualquier máquina que lo soporte (local, servidor, etc.).
- Una vez cumplido el estándar, el tipo de carga del contenedor marítimo es independiente.
 - Lo mismo ocurre con los contenedores en virtualización. Si se cumple el estándar, el software que contenga podrá ser ejecutado sin problemas

3.3 Contenedores para desarrollo y despliegue de aplicaciones

Uno de los principales usos de los contenedores (aunque no el único) es facilitar el desarrollo, distribución y el despliegue de aplicaciones.

- Compilar software es tedioso. Utilizando un contenedor, tenemos el entorno de compilación/depuración montado con las versiones que necesitamos.
- Usar contenedores facilita el testeo, permitiendo la creación de distintos entornos de prueba con diferentes configuraciones, etc.
- Los contenedores nos evitan problemas de compatibilidad al desplegar nuestras aplicaciones, teniendo siempre las versiones adecuadas para ejecutar nuestro software.

Interesante: Muchos sistemas de CI/CD (Continuous Integration/Continuous Delivery) se basan en el uso de contenedores.

3.4 Contenedores para despliegue de servicios

Otro de los principales usos de los contenedores es el despliegue de servidores de distinto tipo (web, correo, bases de datos, DNS, etc.).

Además de las ventajas anteriormente citadas de mantener versiones de software, los contenedores nos permiten unificar configuraciones de servidores en local, incluso involucrando a distintos servicios en distintos contenedores, de forma que al desplegarlos en la nube, funcionen exactamente igual que en las pruebas realizadas localmente.

! **Atención:** *“En mi máquina funcionaba... falla solo al subirlo al servidor...”*. El uso de contenedores contribuye a que esta situación desaparezca :)

Además, los contenedores facilitan el “escalado horizontal” de servicios, especialmente si se apoyan de herramientas llamadas orquestadores.

Para saber más https://es.wikipedia.org/wiki/Escalabilidad#Escalabilidad_horizontal

3.5 Ventajas e inconvenientes del uso de contenedores

Algunas de las ventajas del uso de contenedores son:

- Los contenedores ocupan menos espacio, al no tener que replicar en cada uno el sistema operativo que están virtualizando, ya que utilizan el sistema de la máquina anfitrión.
- Al no tener que realizar una virtualización de hardware, la ejecución del software de los contenedores es mucho más rápida, con velocidades cercanas a las nativas.
- Multitud de empresas de software (Microsoft, Apache, Nginx, MySQL, Oracle, Wordpress, Moodle, y un largo etc.) apoyan estas tecnologías y dan soporte tanto incorporando sistemas de contenedores a sus sistemas operativos, como ofreciendo imágenes oficiales de sus productos para que con una sencilla orden, se pueda poner en marcha alguno de sus servicios o aplicaciones.

Algunas de las principales desventajas de los contenedores son:

- Pese a que mejoran enormemente la velocidad respecto a una virtualización por hipervisor, siguen teniendo un rendimiento peor que una ejecución “bare metal” (sobre un sistema real), ya que el aislamiento consume recursos.
- La persistencia y el acceso/modificación a datos persistentes entre contenedores es más tedioso que si se realiza sobre una máquina real.
- Los contenedores están pensados generalmente para el uso vía línea de comandos. Aunque técnicamente es posible configurar los contenedores para tener su propio entorno gráfico, este proceso es tedioso.

3.6 En resumen ¿Cuándo es adecuado usar contenedores?

El uso de contenedores, suele ser adecuado en los contextos:

- Como usuarios: queremos probar algo rápido y sin complicarnos mucho en la configuración (por ejemplo, montar un servicio en local para aprender).
 - Para ello, podemos utilizar servicios de distribución de imágenes de contenedores públicas como Docker Hub <https://hub.docker.com/>
- Como desarrolladores: queremos desarrollar una aplicación que se pueda distribuir en local o desplegar en la nube sin problemas de configuración
 - Podemos usar contenedores, tanto para tener el entorno de desarrollo listo, como para distribuir la aplicación en sí.
- Queremos testear nuestra aplicación con distintas configuraciones, límites de recursos, juegos de prueba, etc.
 - Útil para generar entornos de prueba y despliegue utilizando CI/CD (Continuous Integration/Continuous Delivery) <https://es.wikipedia.org/wiki/CI/CD>
- Queremos realizar “escalado horizontal” de servicios, es decir ejecutar múltiples copias de una misma aplicación/conjunto de aplicaciones que funcionan como un cluster.
 - https://es.wikipedia.org/wiki/Escalabilidad#Escalabilidad_horizontal

4. CONTENEDORES EN SISTEMAS LINUX

4.1 ¿Es nuevo el concepto de entornos privados en sistemas Unix?

El concepto de entornos privados, utilizado en los controladores, no es algo novedoso de los sistemas Unix modernos. Desde hace muchos años existían algunas soluciones tales como:

- Chroot (Sistemas Unix): <https://es.wikipedia.org/wiki/Chroot> (1982)
- Jail (FreeBSD): https://es.wikipedia.org/wiki/FreeBSD_jail (1999)

Estas utilidades son los “abuelos” del concepto actual de contenedor en los sistemas Unix.

4.2 Sistemas privados modernos en Linux: contenedores

En el año 2008, aparecen los contenedores modernos de sistemas Linux, con el sistema LXC (Linux Container). Para su desarrollo, se introdujeron nuevas capacidades en el kernel de Linux, que han sido aprovechadas por otros sistemas de contenedores.

Aunque en este curso nos vamos a centrar en contenedores Docker, siempre podéis obtener más información de contenedores Linux como LXC, LXI y LXCFS en <https://linuxcontainers.org/>

Además, aquí os presento un ejemplo práctico desarrollado por José Castillo donde usa LXI en sus clases: <https://www.youtube.com/watch?v=ynglk64Hecg>

4.3 ¿Cómo funcionan los contenedores modernos en Linux?

Los sistemas más populares de contenedores sobre Linux, han utilizado (entre otras) dos características del kernel aparecidas en versiones relativamente recientes:

- **Linux namespaces:** permite aislar procesos de forma que vean unos recursos concretos. Los procesos que tienen un “namespace” común pueden ver recursos comunes.
 - Esto entre otras cosas, nos permite tener procesos “diferentes” de la máquina real a los contenedores, incluso con privilegios diferentes (un proceso puede ser “root” en el contenedor, pero no tiene esos privilegios en la máquina real).
 - Para saber más:
 - https://en.wikipedia.org/wiki/Linux_namespaces
 - <https://www.linux.com/news/understanding-and-securing-linux-namespaces/>
- **Cgroups:** permite aislar, configurar y limitar el uso de recursos(memoria, procesos, E/S, etc.). Para saber más <https://en.wikipedia.org/wiki/Cgroups>

En resumen, Linux namespaces nos facilita aislar el sistema y cgroups facilita la limitación/configuración de la disponibilidad de recursos de cada contenedor.

4.4 ¿Puedo poner en marcha un contenedor Linux “A mano”?

Si, es posible. Tal como nos muestra la conocida autora de materiales en formato comic **Julia Evans** aquí un ejemplo de script donde se muestra como crear un contenedor de Linux “a mano”.

<https://gist.github.com/jvns/ea2e4d572b4e2285148b8e87f70eed73>

Aprovecho para recomendar su web <https://wizardzines.com/>, donde parte de su trabajo es Creative Commons, y en concreto su “WizardZine” sobre contenedores <https://wizardzines.com/zines/containers/>


4.5 Los contenedores Linux ¿Pueden funcionar en sistemas como Windows o MacOS?

En principio, es posible ejecutar contenedores Linux en sistemas diferentes a Linux, aunque es posible que el rendimiento no sea el mismo. Algunos sistemas, para poder utilizar contenedores Linux, necesitan que un hipervisor virtualice un sistema Linux completo y que desde ahí se lancen los contenedores Linux.

En sistemas Windows y MacOS la estrategia era la siguiente: mediante un hipervisor, por ejemplo VirtualBox, se virtualizaba un sistema Linux y ahí se ponía en marcha el sistema de contenedores

En el caso concreto de Docker, para hacer este proceso de forma transparente, se usaba la hoy “discontinuada” herramienta “Docker Toolbox” <https://github.com/docker/toolbox>.

Actualmente, dado el crecimiento de Docker, existen otras optimizaciones que comentaremos más adelante. Aun así esta estrategia sigue siendo posible utilizarla para virtualizar contenedores Linux (LXC, LXD, Docker, etc.) en otros sistemas.

 **Importante:** Estos casos pueden ser útiles en algún contexto (pruebas, aprendizaje, desarrollo para otra plataforma), pero se pierden ventajas relativas al rendimiento.

5. CONTENEDORES DOCKER

5.1 ¿Qué es Docker?

Docker es un sistema de contenedores Linux que utiliza las características del núcleo de Linux para permitir el desarrollo y despliegue de aplicaciones.

Su web oficial <https://www.docker.com/> y su entrada en la wikipedia donde se da información detallada del proyecto [https://es.wikipedia.org/wiki/Docker_\(software\)](https://es.wikipedia.org/wiki/Docker_(software))

Docker es un proyecto de código abierto. Generalmente dispone de varias versiones:

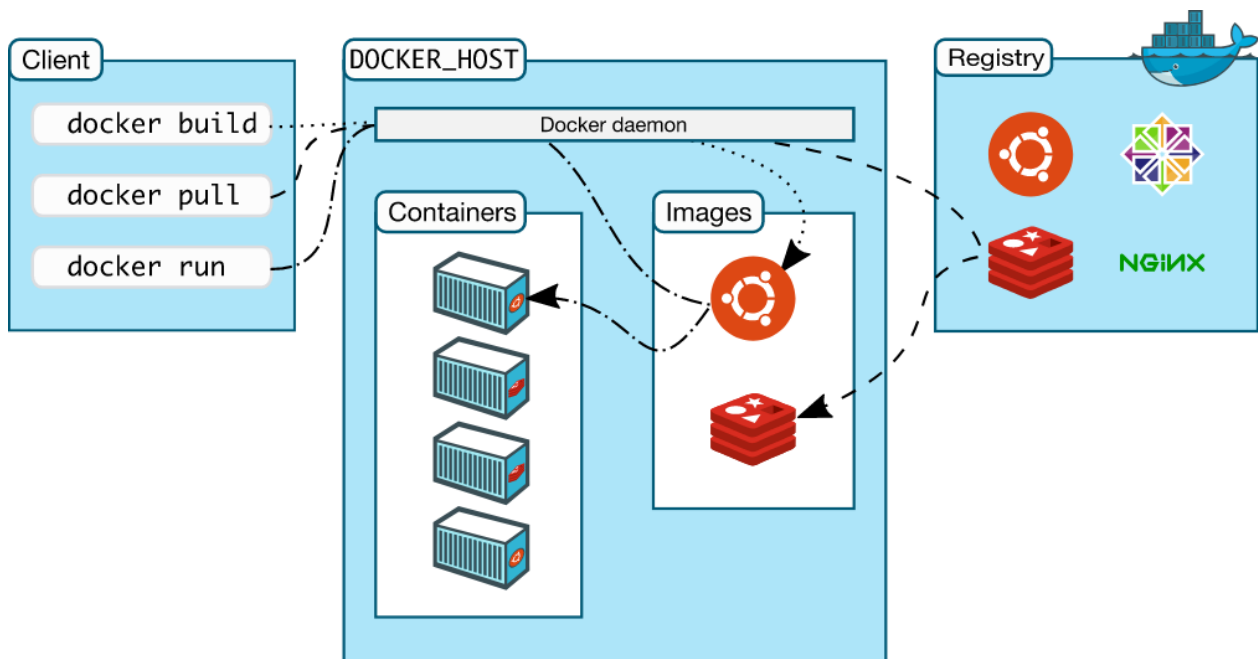
- **Docker CE (Community Edition):** el motor de Docker, de código abierto..
- **Docker EE (Enterprise Edition):** lo mismo que la versión CE, solo que además incluye certificación de funcionamiento en algunos sistemas concretos y soporte con la empresa Docker Inc.

El sistema de contenedores de Docker es integrable con otros servicios populares en la nube, tales como Google Cloud, Amazon AWS, Microsoft Azure, Digital Ocean y OVH, entre otros.

- AWS: <https://aws.amazon.com/es/getting-started/hands-on/deploy-docker-containers/>
- Azure: <https://docs.microsoft.com/es-es/learn/modules/run-docker-with-azure-container-instances/>
- Google Cloud: <https://cloud.google.com/container-optimized-os?hl=es-419>

5.2 La arquitectura de Docker

En esta imagen podemos ver como funciona la arquitectura básica de Docker.



Fuente imagen: <https://github.com/docker/docker.github.io/blob/master/engine/images/architecture.svg>

Esta arquitectura, la podemos resumir en 3 partes:

- **Cliente:** es el software encargado de comunicarse con el servidor Docker.
- **Servidor (Docker Host):** servicio Docker, donde se atienden a las peticiones de los clientes y se gestionan los contenedores e imágenes.
- **Registro (Registry):** lugar donde se almacenan imágenes Docker (públicas o privadas). Incluso, de una misma imagen, se almacenan las distintas versiones. El registro más popular y configurado por defecto en Docker es “**Docker Hub**” <https://hub.docker.com/>

5.3 DOCKER EN SISTEMAS WINDOWS Y MacOS

En apartados anteriores, comentamos que una de las estrategias para lanzar Docker (y otros contenedores Linux) en sistemas Windows y MacOS, era virtualizar un sistema Linux y ahí lanzar el sistema de contenedores.

Actualmente, en sistemas Windows y MacOS se plantea como alternativa el uso del software “Docker Desktop” para utilizar contenedores en estos sistemas. Docker Desktop nos instala todas las aplicaciones necesarias para correr contenedores en estos sistemas, de la forma más óptima posible.

A continuación, realizamos una pequeña comparativa a nivel de rendimiento entre la ejecución de Docker en diferentes sistemas:

- **Ejecución de Docker en Linux:**
 - Están implementados por el kernel, con velocidad cercana a la nativa.
- **Ejecución de Docker en Windows:**
 - Los contenedores Linux de Docker, funcionan usando Hyper-V en sistemas Windows Server y WSL2 (Windows Subsystem for Linux 2) en Windows Home <https://www.docker.com/blog/docker-hearts-wsl-2/>
 - Los contenedores “Windows Server Core”, están implementados por el núcleo de

Windows.

- **Ejecución de Docker en MacOS:**

- Los contenedores Linux funcionan usando Hyperkit.
 - <https://github.com/moby/hyperkit>

Más información en:

- <https://docs.docker.com/desktop/>
- <https://docs.docker.com/docker-for-windows/release-notes/>
- <https://docs.docker.com/docker-for-mac/release-notes/>

5.4 DOCKER CORRIENDO CONTENEDORES WINDOWS SERVER CORE Y CONTENEDORES MacOS

Aunque el uso habitual de Docker es lanzar contenedores con sistemas Linux, nuevas mejoras han permitido que puedan utilizarse contenedores que lancen otros sistemas operativos.

Docker en sistemas Windows puede lanzar contenedores que corren el sistema operativo “Windows Server Core”. Debe virtualizarse con un sistema anfitrión Windows. Más información en:

- https://hub.docker.com/_/microsoft-windows-servercore
- <https://blog.ipswitch.com/creating-your-first-windows-container-with-docker-for-windows>

Asimismo, es posible lanzar un contenedor que ejecute MacOS en un sistema Linux que tenga instalado KVM, utilizando el proyecto disponible en <https://github.com/sickcodes/Docker-OSX>

6. CONCLUSIÓN

En esta unidad hemos repasado conceptos básicos sobre virtualización. Tras ello, hemos procedido a introducir el concepto de contenedor y sus características, centrándonos en contenedores Linux. Comprendidos los conceptos de contenedores, hemos introducido la solución Docker, la cual instalaremos y utilizaremos en futuras unidades.

7. BIBLIOGRAFÍA

- [1] WizardZines “How containers work” <https://wizardzines.com/zines/containers/>
- [2] Docker Docs <https://docs.docker.com/>
- [3] Linux containers <https://linuxcontainers.org/>
- [4] OS Level virtualization https://en.wikipedia.org/wiki/OS-level_virtualization

8. LICENCIAS DE ELEMENTOS EXTERNOS UTILIZADOS

Figura 1: Imagen con licencia Apache 2.0. Fuente:

<https://github.com/docker/docker.github.io/blob/master/engine/images/architecture.svg>

Figura 2: Imagen con licencia CC BY SA. Fuente:

<https://commons.wikimedia.org/wiki/File:Docker-containerized-and-vm-transparent-bg.png>

Figura 3: Imagen con licencia CC BY SA. Fuente:

<https://www.flickr.com/photos/68359921@N08/50125348052/>