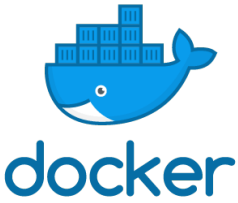
UNIVERSIDAD
POLITÉCNICA
DE MADRID



Análisis de contenedores Docker

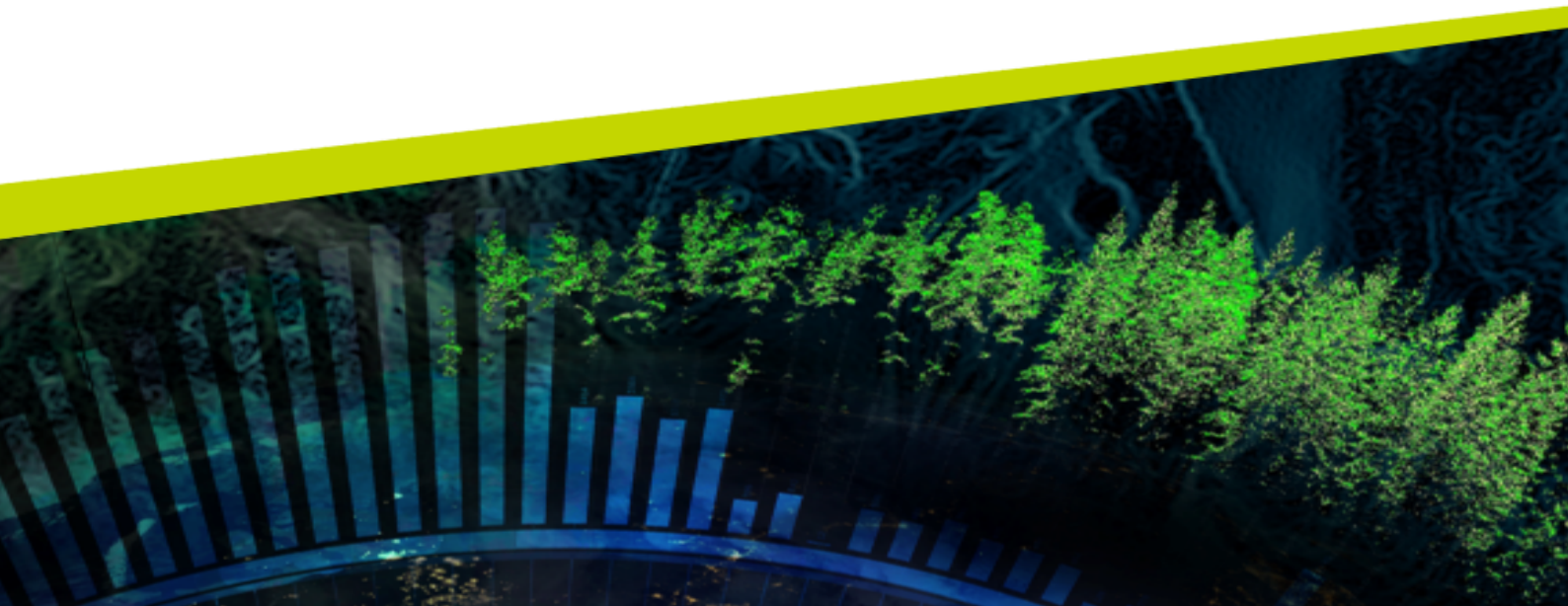
- y sus implicaciones de seguridad

Javier Alonso Silva

Seguridad en Sistemas y Redes

Universidad Politécnica de Madrid

2021



Resumen

TO–DO

Índice

1. Introducción

La era tecnológica ha avanzado en los últimos años a pasos agigantados, y las demandas del sector han crecido junto a ella. No hace más de 200 años se "descubría" la electricidad; hace 90 años nacía la primera computadora básica capaz de realizar operaciones aritméticas; hace 70 años nacía el transistor que sustituyó las válvulas de vacío (figura 1); y desde entonces, el crecimiento ha sido exponencial [1].

Figura 1: Comparativa de una válvula de vacío (izquierda) frente a un transistor (centro) y un circuito integrado (derecha).

Otro de los ejemplos de tecnologías que han crecido exponencialmente son los dispositivos de almacenamiento, donde no hacía más de 20 años las capacidades máximas se estimaban en torno a los MB (megabytes) y ahora se hablan de EB (exabytes) [2]. Esta evolución es muy representativa también a nivel económico, ya que el coste del almacenamiento ha ido bajando a medida pasaba el tiempo, así como el espacio físico que ocupan los dispositivos (figura 2):

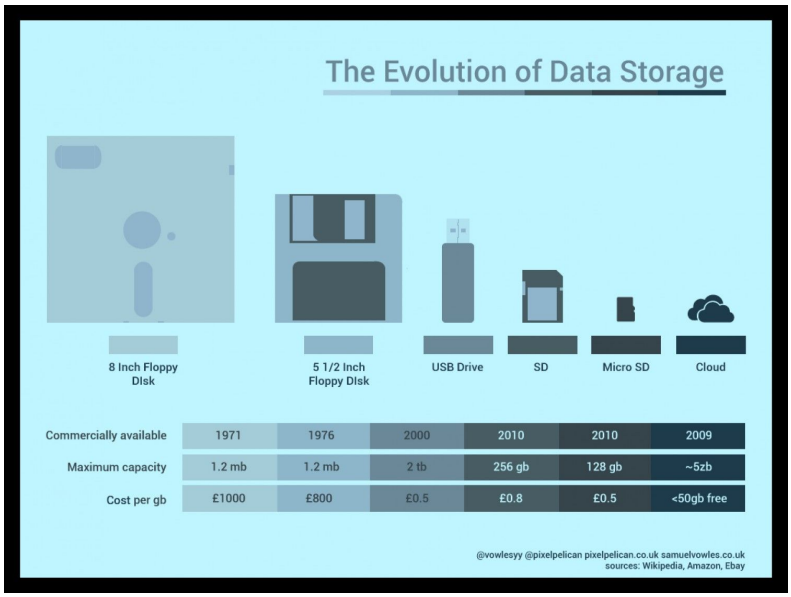


Figura 2: Evolución del espacio de almacenamiento en términos económicos y cuantitativos [3].

Finalmente, el gran salto tecnológico se ha producido con la aparición de Internet y las comunicaciones ya no eran únicamente personales sino entre dispositivos. En relación con el punto anterior, la aparición de Internet ha permitido descentralizar el espacio donde ya el usuario no guarda su información en su equipo personal sino en un clúster de servidores distribuidos a nivel mundial al cual accede, de forma simultánea, desde Internet y desde cualquier dispositivo. Así, lo que comenzó como una red de conexión de unos pocos usuarios ha acabado convirtiéndose en la red global que todos usamos y que conecta más de 4 billones de dispositivos (figura 3).

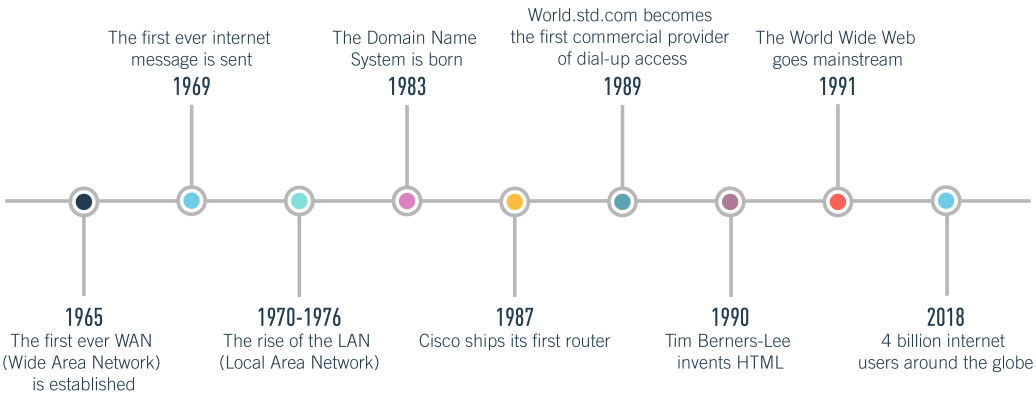


Figura 3: Evolución de Internet a lo largo del tiempo, hasta llegar a hoy [4].

El problema a esto es evidente: con una mayor capacidad de cómputo, con más opciones de comunicación y con más posibilidad de almacenar datos, los requisitos de las aplicaciones van creciendo y creciendo y cada vez son más complejos de satisfacer, no necesariamente a nivel *hardware* (que por lo general suele acompañar) sino a nivel *software*. Como las aplicaciones se orientan a los usuarios es necesario añadir capas de abstracción (como el sistema operativo) para facilitar la labor a la persona. Sin embargo, cada capa nueva que se añade dificulta las tareas de despliegue y mantenimiento dado que existe una gran variedad de combinaciones *hardware* y cada una puede estar con un sistema operativo distinto.

Por otra parte, la extensión de dependencias y posible incompatibilidad entre ellas suele desembocar en el uso de versiones desactualizadas de una librería ya que tendríamos "paquetes rotos". Esto es tan común que tiene hasta su propio término coloquial "*dependency hell*" [5]. Contar con dependencias obsoletas que ya han cumplido con su ciclo de vida *software* conlleva unas implicaciones de seguridad bastante severas:

- Si un *software* no ha mejorado a lo largo del tiempo, existe una malicia humana que puede aprovecharse de distintos *exploits* existentes y comprometan nuestra aplicación.
- Un *software* no actualizado puede tener implicaciones directas sobre el sistema en que se ejecuta, pudiendo producir fallos en el mismo. Esto se debe principalmente a que el *hardware* sigue mejorando y creciendo y un *software* antiguo puede presentar *bugs* en dispositivos modernos que no presentaría en antiguos.

- Un *software* no actualizado puede comprometer otros elementos del sistema en que se ejecuta. Por ejemplo, una aplicación 'A' hace uso de dicho *software* y una aplicación 'B' también. Sin embargo, la última aplicación se ha diseñado para trabajar con la última versión del *software* pero la aplicación 'A' solo puede funcionar con una versión antigua e insegura. Por consiguiente, pese a que la aplicación 'B' funcionaría correctamente el hecho de usar una versión antigua e insegura del *software* compromete directamente al sistema y a la aplicación.

Es por eso que existen alternativas como "`chroot`" y máquinas virtuales para subsanar estos problemas. Sin embargo, en los últimos años ha aparecido una herramienta muy sonada y con gran éxito: Docker y los contenedores.

1.1. ¿Qué es Docker?

1.2. *Real–life usages*

1.3. *Docker rules*

2. Docker

2.1. Estructura de un Docker

2.2. Creación de un contenedor

2.3. Comunicación entre contenedores

2.4. Despliegue de aplicaciones multi–contenedores. `docker-compose`

2.5. "Orquestación" de contenedores

2.6. Líneas futuras de desarrollo e innovación

3. Seguridad en Docker

3.1. Análisis de la pila Docker

3.2. Diferencias fundamentales con `chroot`

3.3. Seguridad en las comunicaciones de red – *firewall*

3.4. Seguridad en las comunicaciones inter–contenedores

Referencias

[1] «History of Technology Timeline,» Encyclopedia Britannica. (), dirección: `https://www.britannica.com/story/history-of-technology-timeline` (visitado 07-05-2021).

[2] «Evolution of Data Storage Timeline,» The Gateway. (), dirección: `/gateway/data-storage-timeline/` (visitado 07-05-2021).

[3] WeComputingTech. «Storage devices london | We Computing Blog.» (), dirección: `http://www.wecomputing.com/blog/tag/storage-devices-london/` (visitado 07-05-2021).

[4] «How To Become A Web Developer in 2021 — Everything You Need To Know.» (), dirección: https://careerfoundry.com/en/blog/web-development/what-does-it-take-to-become-a-web-developer-everything-you-need-to-know-before-getting-started/ (visitado 07-05-2021).

[5] *Dependency hell*, en *Wikipedia*, 29 de mayo de 2021. dirección: https://en.wikipedia.org/w/index.php?title=Dependency_hell&oldid=1025704309 (visitado 03-06-2021).