Máster en Sistemas Inteligentes

Tecnología de contenedores

SJK006 - Cloud Computing

UNIVERSITAT JAUME I

Profesor:
Óscar Belmonte Fernández

Autores:
Pablo Muñoz Alcaide
Raquel Lázaro Belenguer
Arturo Gascó Compte
Miguel Pardo Navarro
Javier González Barreda

Curso:
2023/2024

Fecha:
20 de octubre de 2023

Dedicación horaria:

Alumno	*Horas presenciales*	*Horas individuales*	*Horas totales*
Pablo Muñoz Alcaide	3	2	5
Raquel Lázaro Belenguer	3	2	5
Arturo Gascó Compte	3	2	5
Miguel Pardo Navarro	3	2	5
Javier González Barreda	3	2	5
		Total	25

Introducción

La tecnología de contenedores [1][2] es un método de virtualización que consiste en el empaquetamiento de software con todos los elementos necesarios para crear un ejecutable liviano que se pueda ejecutar en cualquier entorno. La base de la tecnología de contenedores son las denominadas Imágenes de Contenedor [3], las cuales son un archivo estático con código ejecutable que contiene las librerías, binarios, código fuente y otras dependencias necesarias para desplegar un entorno de contenedores. A diferencia de las máquinas virtuales tradicionales, las imágenes de contenedor comparten el *kernel* del sistema operativo del *host*, por lo que no necesita incluir un sistema operativo completo. Esto confiere ciertas características tales como la portabilidad o la ligereza, sobre las cuales profundizaremos en el siguiente punto.

En resumen, un contenedor [4] se define como una instancia de una Imagen que, a su vez, consiste en un grupo de procesos que corren en un sistema Linux. Una de las características principales es que pueden ejecutar varios contenedores a partir de una misma imagen, ya que estos grupos de procesos se encuentran aislados unos de otros sobre la misma máquina Linux.

Características principales

La tecnología de contenedores destaca por su **eficiencia** y **ligereza**, puesto que, a diferencia de las máquinas virtuales que requieren un sistema operativo completo para cada instancia, los contenedores aprovechan el *kernel* del sistema operativo de la máquina anfitriona. La siguiente figura muestra la diferencia entre la arquitectura [5] de ambas tecnologías.

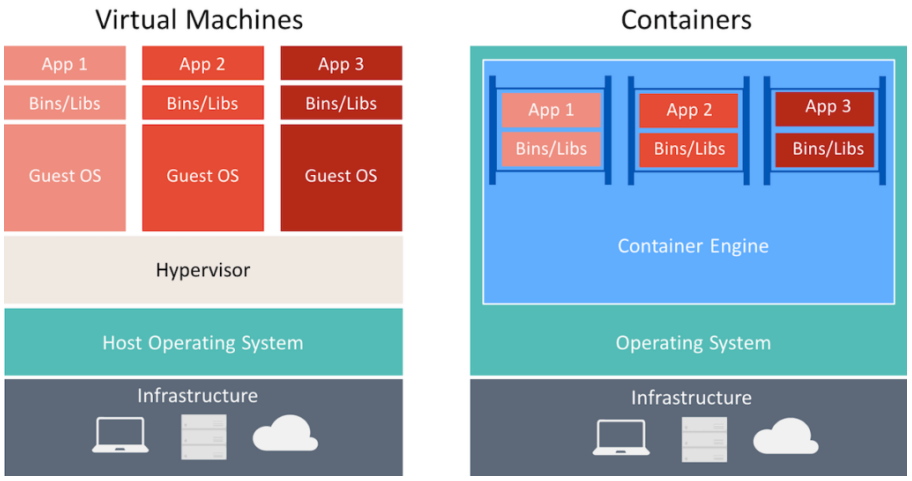

Figura 1. Arquitectura de las máquinas virtuales (izquierda) y de la tecnología de contenedores (derecha)

Además, la propiedad de que los contenedores compartan el *kernel* de la máquina anfitriona le otorga una característica muy relevante: la **portabilidad**. El contenedor, al estar formado por todas las dependencias necesarias para su funcionamiento, permite que este se ejecute en cualquier infraestructura. De esta forma, los desarrolladores pueden utilizar las mismas herramientas cuando trabajan con contenedores en un entorno o en otro, lo que simplifica enormemente el desarrollo y la implantación de aplicaciones en contenedores en distintos sistemas operativos.

Como ya se ha descrito, el propio contenedor se abstrae del sistema operativo anfitrión y solo tiene acceso limitado a los recursos subyacentes. Además, cada contenedor está aislado del otro, lo que proporciona ventajas en **seguridad** y **funcionalidad**. En términos de seguridad, el aislamiento de las aplicaciones de contenedores impide intrínsecamente que la invasión de código malicioso afecte a otros contenedores o al sistema anfitrión, mientras que en términos de funcionalidad, el fallo de un contenedor no afecta al funcionamiento de los demás. Por lo tanto, los equipos de desarrollo pueden identificar y corregir cualquier problema técnico dentro de un contenedor sin que haya tiempo de inactividad en otros contenedores.

En definitiva, las principales características de la tecnología de contenedores incluyen: eficiencia, ligereza, portabilidad, seguridad y amplia funcionalidad. Debido a la conjunción de todas estas propiedades, los contenedores son ampliamente utilizados para empaquetar los microservicios que componen las aplicaciones modernas.

Evolución histórica

Tal y como se ha descrito en secciones previas, la tecnología de contenedores se basa en empaquetar aplicaciones para que puedan funcionar en cualquier sistema de forma aislada. Las primeras nociones de aislamiento de procesos aparecieron con la utilidad chroot [6] en 1979 a partir de la versión 7 de Unix. Esta herramienta, chroot, permitía modificar el directorio raíz de un proceso, facilitando que se ejecutara en un entorno separado dentro del sistema operativo *host*, proporcionando así un nivel básico de aislamiento entre los procesos.

En la década de los 2000 se impulsaron en gran medida las tecnologías de contenerización con la introducción de FreeBDS Jails. Estas «jaulas» además de aislar los directorios raíz, también aíslan otras características del sistema como *networking* y usuarios. A partir de este sistema de jaulas, se creó Linux-VServer en el 2001, que añadía además la limitación de recursos de la CPU y la memoria. De esta forma, con Linux-VServer, se plantea la idea de dividir un servidor físico en múltiples entornos virtuales aislados y separados entre sí.

En 2004 se lanzaron los contenedores de Solaris, basados en control de recursos y una mejora en el aislamiento de los contenedores pero, esta vez, separados por zonas. De forma muy similar a estos contenedores, un año más tarde, OpenVZ habilita la posibilidad de crear contenedores en los que se pueden manejar los distintos recursos, además de hacer *checkpoints*, es decir, que se puede volver a un paso anterior si se desea.

En 2006 se crean los grupos de control, que permiten la limitación de los recursos que se le otorgan a cada contenedor. Google implementa los contenedores basados en procesos, que utilizan características del kernel para realizar el enjaulamiento. Un año después, en 2007, se integran los cgroups al kernel de Linux.

Gracias a la integración de los cgroups, se desarrolla LXC [7], los contenedores de Linux que permitían descargar imágenes ya formadas con las características que quisiera el desarrollador. Además, funcionaban directamente sobre el kernel de Linux sin tener que usar parches. Este sistema de contenerización ha sido la base de muchos otros, puesto que proporciona mucha estabilidad. En 2011 surge Warden, una tecnología que, al igual que LXC, permite la contenerización, pero con una diferencia significativa: no está limitada únicamente a Linux, sino que también es compatible con otros sistemas operativos.

En 2013 nace «*let me contain that for you*», una versión *open-source* del orquestador de Google para manejar los contenedores, a partir del cual se creó libcontainer. Ese mismo año se crea Docker, basado en LXC y libcontainer. Se trata de una alternativa a las propuestas hasta el momento que permite crear imágenes propias al desarrollador. Se hizo muy famoso porque la interfaz gráfica es muy intuitiva y permite tener aplicaciones con diferentes requisitos de sistema operativo en una sola máquina. Rkt, emergió a partir de Docker y abordó y mejoró algunos de los problemas de seguridad existentes en Docker, posicionándose como una solución más enfocada en la seguridad de los contenedores.

Tras el éxito de Docker se creó la OCI en 2015 (*Open Container Initiative*), con la idea de estandarizar las tecnologías de contenedores. Actualmente muchas empresas, como Google, Amazon, Microsoft y el propio Docker, forman parte de la OCI.

Finalmente, en 2016 Microsoft permite que se puedan utilizar los contenedores basados en Linux (como Docker) en ordenadores de Windows sin tener que utilizar máquinas virtuales con la creación de Windows containers.

Tecnologías de base

La propia naturaleza de Linux favorece a la contenerización, ya que al ser open-source se promueve el desarrollo de tecnologías que mejoran el rendimiento y la seguridad de los contenedores. Las soluciones de contenerización se basan en características del *kernel* de Linux [8] para permitir la creación de contenedores. A continuación se describen y se definen los conceptos base que permiten el funcionamiento de los contenedores.

***Kernel* de Linux**

El *kernel* de Linux es el encargado de la gestión y control de los recursos en el sistema. Permite el soporte de múltiples sistemas de archivos, media entre el *hardware* y los procesos, aporta seguridad y supervisa la memoria que se utiliza en cada elemento. De esta forma, se pueden aprovechar los recursos que ofrece el kernel de Linux para crear los contenedores.

Cgroups (*control groups*)

Los grupos de control son una de las características principales del *kernel* de Linux, se utilizan para limitar los recursos que se ponen a disposición de los distintos procesos. Mediante la partición de recursos del sistema se puede asegurar que no se consumen recursos innecesarios.

Namespaces

Un *namespace* es una agrupación de características del sistema cuyo objetivo es limitar la visibilidad o aislar un grupo de procesos, lo que permite que se puedan ejecutar varios contenedores en una misma máquina. Entre los distintos tipos de *namespaces* destacan:

- *Process ID namespace* (PID): controla los procesos de forma que cada contenedor tiene su espacio de nombres PID, es decir que cada proceso tiene su identificador dentro del contenedor, pero otro proceso en otro contenedor con otro namespace podría tener el mismo PID.
- *Network* (net): con este *namespace* cada contenedor tiene asignados unos recursos de red propios, dispositivos, protocolos, reglas de firewall, etc.
- *Mount namespace* (mnt): controla los puntos de montaje, aísla el sistema de archivos de forma que cada contenedor tenga su propio sistema de archivos independiente.
- *User namespace*: aísla ID de usuario y grupo, dentro de un contenedor puede haber unos nombres de usuario que no tengan el mismo significado fuera del contenedor. Mejora el aislamiento y, por tanto, la seguridad de los distintos contenedores.

En definitiva, gracias a estos recursos se favorece la contenerización. Además, debido al desarrollo continuo en la comunidad de Linux, se crean y mejoran constantemente soluciones que permiten trabajar mejor con los contenedores. Por ejemplo, hay distribuciones de Linux optimizadas específicamente para la ejecución de contenedores como CoreOS [9] o Project

Atomic [10]. También existen distintas APIs y bibliotecas en Linux que se pueden utilizar para desarrollar soluciones de contenedor.

Soluciones para trabajar con contenedores

A continuación se presentan las principales y más conocidas soluciones para trabajar con la tecnología de contenedores.

LXC

Linux Containers (LXC) es una solución de virtualización a nivel de sistema operativo para ejecutar múltiples entornos Linux aislados en un solo host de Linux. Es *open source* y conocido por su eficiencia y pequeña huella de memoria. LXC utiliza la API de espacio de nombres de Linux para proporcionar un aislamiento operativo y de red efectivo, permitiendo así que los contenedores se ejecuten en un entorno seguro y escalable [11].

LXD

LXD es una extensión de LXC, actualmente propiedad de la empresa Canonical, pero, previamente, formaba parte del paraguas Linux Containers, en los que se encuentra LXC y el nuevo sucesor de LXD, aunque todavía en estado de desarrollo: Incus [12].

OpenVZ

OpenVZ: Es otra solución de contenedorización para Linux de software libre, disponible bajo licencia GNU. OpenVZ te permite crear contenedores seguros y completamente aislados, también conocidos como VEs o VPSs, con diferentes usuarios, direcciones IP, procesos y aplicaciones [13].

Rkt

Rkt (pronunciado como «rocket») es una alternativa a Docker desarrollada por CoreOS con el objetivo de abordar ciertas preocupaciones de seguridad asociadas con Docker. Fue lanzado en 2014 como una opción más segura, interoperable y abierta, ya que versiones anteriores de Docker se ejecutaban como *root*, lo que podía otorgar a un atacante privilegios de superusuario si se encontraban vulnerabilidades en alguno de los contenedores [14][15]. Esta solución se diseñó, esencialmente, para mejorar la seguridad de los contenedores, adoptando un enfoque distribuido y más amigable con Linux [16].

Docker

Docker es una de las plataformas de contenedores más populares en la actualidad. Proporciona un entorno de ejecución que empaqueta una aplicación y sus dependencias en un contenedor virtual que puede ejecutarse en cualquier sistema operativo Linux o Windows que soporte Docker, facilitando la portabilidad entre diferentes entornos y mejorando la eficiencia al asegurar la coherencia en los entornos de desarrollo, prueba y producción. Con una vasta biblioteca de imágenes predefinidas en *Docker Hub*, los desarrolladores tienen la posibilidad de acelerar el despliegue de aplicaciones y servicios, al tiempo que mantienen la flexibilidad de personalizar las imágenes para sus necesidades específicas. Además, la tecnología de

Docker introduce una capa de abstracción y automatización sobre la virtualización a nivel de sistema operativo, lo que la convierte en una herramienta esencial para los equipos que buscan mejorar continuamente la integración y entrega de aplicaciones [17].

Containerd

Containerd es un entorno de ejecución de contenedores de código abierto que fue originalmente desarrollado por Docker y posteriormente donado a Cloud Native Computing Foundation (CNCF) [18]. Es un entorno estándar en la industria diseñado para ser simple, robusto y portátil, y está disponible como un demonio para sistemas Linux y Windows. Containerd puede gestionar el ciclo de vida completo del contenedor en su sistema *host*, incluyendo la transferencia y almacenamiento de imágenes, ejecución y supervisión de contenedores, y mucho más [19]. Se le suele describir como un «motor» que maneja la creación, ejecución y eliminación de contenedores, facilitando la gestión y ejecución de aplicaciones en entornos aislados [20].

Podman

Por otro lado, Podman es una herramienta de código abierto que facilita el desarrollo, gestión y ejecución de contenedores y *pods* (grupos de contenedores). Fue desarrollado originalmente por ingenieros de Red Hat junto con la comunidad de código abierto. Esta herramienta gestiona el ecosistema completo de contenedores utilizando la biblioteca libpod [21]. Es una herramienta nativa de Linux, sin *daemon*, diseñada para facilitar la búsqueda, ejecución, construcción, compartición y despliegue de aplicaciones usando contenedores e imágenes de contenedores de la Iniciativa de Contenedores Abiertos (OCI) [22]. Además, Podman proporciona una interfaz de línea de comandos (CLI) familiar para cualquier persona que haya empleado el motor de contenedores Docker. También puede manejar volúmenes montados en esos contenedores y *pods* creados a partir de grupos de contenedores. Aunque Podman se ejecuta en sistemas Linux, también puede ser utilizado en sistemas Mac y Windows mediante una máquina virtual gestionada por Podman [23].

Comparación entre las distintas soluciones

Solución	Propietario	Características Principales
 LXC	Open Source	Se integra estrechamente con el kernel de Linux y permite una administración detallada de los recursos del sistema.
 LXD	Canonical	Extensión de LXC, con la mejora en gestión y migración de contenedores.
 OpenVZ	Software Libre (GNU)	Contenedores seguros y completamente aislados. Permite una alta densidad de contenedores por host y ofrece una eficiente utilización de recursos.
	Docker Inc.	Tiene un extenso ecosistema y repositorio de imágenes (Docker Hub). Además, es ampliamente utilizado y posee una comunidad muy activa.
 rkt	CoreOS (ahora parte de Red Hat)	No requiere un demonio en tiempo de ejecución y ofrece mayor seguridad. También proporciona soporte a diferentes formatos de contenedores, incluyendo Docker y OCI.
	CNCF (originado por Docker)	Ciclo de vida completo del contenedor, simple y robusto. Además, es compatible con los estándares OCI.
	Red Hat	Permite la ejecución de contenedores sin privilegios (*rootless*) y compatible con comandos de Docker CLI.

Orquestadores de contenedores

Los orquestadores de contenedores son sistemas que ofrecen herramientas para controlar y automatizar diversas tareas relacionadas con contenedores, como el despliegue, el escalado, la gestión de redes y el equilibrio de carga. Estos orquestadores no solo se encargan de mantener las aplicaciones en funcionamiento según lo previsto, sino que también garantizan su resiliencia y disponibilidad, gestionando aspectos como la tolerancia a fallos y la recuperación ante errores.

Mientras que Kubernetes [24] es, sin duda, el nombre más reconocido en este ámbito, existen otras tecnologías y soluciones en el ecosistema de orquestadores que ofrecen capacidades únicas y abordan diferentes desafíos. Explorar estas tecnologías relacionadas nos ofrece una visión más amplia y nos permite comprender mejor las múltiples opciones disponibles para gestionar y orquestar contenedores en la era moderna de la computación.

A continuación se describen brevemente los principales orquestadores que existen actualmente.

Kubernetes

A menudo abreviado como K8s, es un sistema de código abierto diseñado para automatizar, desplegar, escalar y operar aplicaciones contenedorizadas. Originado en Google, basado en su experiencia con Borg [25], Kubernetes ha emergido como el estándar de facto en orquestación de contenedores debido a su robustez, flexibilidad y gran comunidad de soporte.

El corazón de Kubernetes reside en su capacidad para gestionar clústeres de contenedores. Utiliza abstracciones como *pods* para agrupar uno o más contenedores, *services* para definir cómo se accede a estos pods y *deployments* para mantener y actualizar los contenedores en producción. Estas abstracciones, junto con otras, permiten a Kubernetes garantizar que las aplicaciones contenedorizadas se ejecuten y escalen según las especificaciones del usuario. Además, tienen la capacidad de recuperarse automáticamente de posibles fallos.

A través de su rica API y su extensibilidad, Kubernetes puede integrarse con una amplia gama de herramientas y sistemas, lo que lo convierte en una pieza central en muchas infraestructuras modernas de *DevOps* y *cloud-native*.

Docker Swarm

Docker Swarm [26] es la solución nativa de orquestación de contenedores ofrecida por Docker. Diseñado para ser simple y fácil de usar, Swarm permite a los usuarios convertir un grupo de máquinas Docker en un clúster coordinado, denominado *swarm*, en el cual los contenedores pueden ser desplegados y gestionados de manera distribuida. Al estar integrado directamente en el motor Docker, los usuarios pueden aprovechar la familiaridad de los comandos y las herramientas de Docker mientras trabajan en un entorno de orquestación.

Swarm utiliza conceptos como *services* y *tasks* para definir y ejecutar aplicaciones contenedorizadas en el clúster. Un *service* define cómo se deben ejecutar los contenedores, mientras que una *task* es una instancia de un contenedor que se ejecuta en un nodo. Docker Swarm se encarga de tareas como equilibrar la carga entre los nodos, recuperarse de fallos y asegurarse de que haya la cantidad deseada de réplicas para cada servicio.

La integración y coherencia con el ecosistema Docker es uno de los principales atractivos de Swarm, ofreciendo a los usuarios una experiencia fluida y una curva de aprendizaje reducida, especialmente para aquellos ya familiarizados con Docker.

Openshift

OpenShift [27] es una plataforma de contenedores empresarial desarrollada por Red Hat, basada en Kubernetes. Aunque en su núcleo utiliza Kubernetes para la orquestación de contenedores, OpenShift amplía y mejora esta base con características adicionales, herramientas y una experiencia de usuario mejorada diseñada para satisfacer las necesidades de las empresas.

La plataforma OpenShift proporciona una amplia gama de herramientas de desarrollo y operaciones que facilitan el ciclo completo de desarrollo de aplicaciones: creación, despliegue y escalabilidad. Esto incluye una amplia integración con herramientas de CI/CD, una plataforma como servicio (PaaS) para desarrolladores, y soluciones de seguridad y gestión mejoradas que son esenciales para entornos empresariales. Además, OpenShift ofrece un catálogo de servicios, soporte para múltiples lenguajes y *frameworks*, y capacidades avanzadas de red y almacenamiento.

Nomad

Desarrollado por HashiCorp, Nomad [28] es una herramienta de orquestación de tareas y contenedores que destaca por su simplicidad y flexibilidad. A diferencia de otras soluciones de orquestación que se centran principalmente en contenedores, Nomad adopta un enfoque más generalista, siendo capaz de orquestar tanto tareas contenedorizadas, como aquellas que no lo están, como máquinas virtuales o aplicaciones binarias estáticas.

Nomad se ha diseñado con un enfoque en la facilidad de uso, la escalabilidad y la resiliencia. Su arquitectura distribuida y altamente disponible permite gestionar miles de nodos en múltiples regiones con mínima sobrecarga operativa. A través de abstracciones simples, los usuarios pueden definir trabajos que Nomad ejecutará en su infraestructura. La visión de HashiCorp con Nomad es ofrecer a las empresas una solución de orquestación ligera y altamente eficiente que pueda coexistir con otras herramientas y adaptarse a una amplia variedad de cargas de trabajo, desde microservicios hasta aplicaciones monolíticas y trabajos por lotes.

Comparación entre las diferentes tecnologías

Como se muestra en [29], [30] y [31], algunas de las principales ventajas y desventajas de cada una de las tecnologías descritas anteriormente se muestran a continuación.

Característica	Kubernetes	Docker Swarm	Openshift	Nomad
Desarrollador	CNCF (origen en Google)	Docker, Inc.	Red Hat	HashiCorp
Integración con herramientas	Amplio ecosistema	Integración con Docker	Amplio ecosistema + herramientas Red Hat	Integración con herramientas HashiCorp (Consul, Vault)
Extensibilidad	Alta (a través de CRDs)	Moderada	Alta (a través de K8s y herramientas propias)	Moderada
Comunidad	Muy grande y activa	Grande	Grande	Creciente
Escalabilidad	Alta	Alta	Alta	Alta
Servicios de red	A través de *plugins*	*Overlay* nativo	A través de *plugins* + soluciones Red Hat	Integración con Consul
Gestión de almacenamiento	A través de *plugins*	Volumes con *plugins*	A través de *plugins* + soluciones Red Hat	*Plugins* y drivers
Modelo de seguridad	RBAC [32], ServiceAccounts	RBAC a través de Docker	RBAC avanzado, SELinux	ACLs [33], integración con Vault

¿Cuándo se deben utilizar los contenedores?

Los contenedores y las máquinas virtuales representan dos estrategias distintas para la administración de recursos en entornos de tecnología de la información. Ambas tecnologías tienen sus pros y contras, y la elección entre una u otra depende de las necesidades específicas de cada proyecto.

Ventajas de Utilizar Contenedores:

- Tamaño Reducido: Los contenedores son ligeros, ya que comparten el kernel del sistema operativo del host y solo incluyen las bibliotecas y dependencias necesarias para ejecutar la aplicación específica.
- Arranque Rápido: Los contenedores pueden iniciarse en cuestión de segundos o incluso menos, lo que es beneficioso para aplicaciones que requieren escalabilidad rápida y despliegues frecuentes.
- Portabilidad: Los contenedores son portables entre diferentes sistemas y proveedores de la nube, lo que facilita la migración y el despliegue en diferentes entornos.
- Integración Continua y Entrega Continua (CI/CD): Los contenedores son ideales para entornos CI/CD, permitiendo despliegues y pruebas consistentes.

Desventajas de Utilizar Contenedores:

- Redes Complicadas: La comunicación entre contenedores puede requerir una configuración de red más compleja en comparación con las VMs.
- Seguridad: Los contenedores son considerados menos seguros que las VMs, especialmente en lo que respecta al aislamiento entre aplicaciones.
- Descomposición de Aplicaciones: Utilizar contenedores puede requerir descomponer aplicaciones monolíticas en microservicios, lo que puede ser un desafío.

La elección entre contenedores y VMs se reduce a las necesidades específicas del proyecto. Los contenedores son una excelente opción para aplicaciones modernas y escalables, mientras que las VMs son más adecuadas para aplicaciones legadas y entornos que requieren una seguridad robusta.

Caso práctico: Adidas

El caso de Adidas [34] ilustra cómo la orquestación de contenedores, con Kubernetes como pieza central, puede transformar las operaciones de desarrollo y entrega de *software* en una empresa. Anteriormente, obtener una máquina virtual para un desarrollador en Adidas implicaba un proceso burocrático que podía tardar desde media hora hasta una semana. Para solucionar esto, se adoptó una perspectiva centrada en el desarrollador, buscando agilizar la puesta en marcha de los proyectos en la infraestructura de Adidas.

La solución llegó con la adopción de la contenerización, el desarrollo ágil, la entrega continua y una plataforma nativa de la nube que incluía Kubernetes y Prometheus [35]. Como resultado, se logró una transformación notable: seis meses después del inicio del proyecto, el 100% del sitio de comercio electrónico de Adidas se ejecutaba en Kubernetes. Esto resultó en una reducción del tiempo de carga del sitio a la mitad y un incremento en la frecuencia de las entregas, pasando de cada 4-6 semanas a 3-4 veces al día. Con 4,000 *pods*, 200 nodos y 80,000 compilaciones por mes, Adidas ahora ejecuta el 40% de sus sistemas críticos en esta plataforma nativa de la nube.

Fernando Cornago y Daniel Eichten, Directores Senior de Ingeniería de Plataformas, compartieron que Kubernetes, para ellos, es una plataforma creada por ingenieros para ingenieros, liberando al equipo de desarrollo de tareas no deseadas, pero brindándoles visibilidad y control sobre lo que sucede tras bambalinas. Esta transición no solo aumentó la eficiencia operativa, sino que también mejoró la colaboración y el aprendizaje entre los equipos.

Un incidente notable ocurrió durante la preparación para la Cyber Week de 2017, donde el equipo tuvo que crear muchas métricas personalizadas. Enfrentaron un desafío con la base de datos de Prometheus, pero lograron implementar una solución federada en dos días con la colaboración entre los equipos de plataforma y de comercio electrónico.

La adopción de esta plataforma nativa de la nube se propagó rápidamente entre todos los ingenieros de Adidas, con cada miembro dedicando una semana completa a la integración y aprendizaje de la plataforma. Además, se introdujo una competición denominada «DevOps Cup» para fomentar la innovación técnica y la colaboración entre los equipos.

Cornago y Eichten enfatizan que no hay una solución única para todas las empresas, y aconsejan aplicar la cultura de la empresa a cada aspecto de la transición hacia una plataforma nativa de la nube. Esta experiencia de Adidas subraya cómo la orquestación de contenedores, con la ayuda de Kubernetes, puede agilizar las operaciones, fomentar la colaboración y conducir a entregas de software más rápidas y eficientes.

Conclusiones

Tras el desarrollo de este trabajo sobre la tecnología y el ecosistema de contenedores, se han identificado varias conclusiones significativas que se detallan a continuación.

En primer lugar, los contenedores ofrecen una solución eficiente y portátil para el desarrollo y despliegue de aplicaciones. Al encapsular aplicaciones y sus dependencias en un entorno aislado, se asegura que las aplicaciones funcionarán de la misma manera en cualquier entorno, ya sea local, en la nube o en un centro de datos. Esto garantiza una mayor coherencia en el ciclo de vida de la aplicación y facilita la colaboración entre equipos de desarrollo y operaciones.

Así mismo, la tecnología de contenedores permite un despliegue rápido y una escalabilidad eficiente. Las aplicaciones contenidas pueden iniciarse y detenerse en cuestión de segundos, facilitando la gestión de recursos y la adaptabilidad a fluctuaciones en la carga de trabajo. Esto es especialmente valioso en entornos donde la demanda del usuario varía constantemente, ya que los contenedores pueden adaptarse instantáneamente para satisfacer las necesidades del usuario.

Los contenedores también facilitan el desarrollo ágil al permitir la adopción de prácticas de desarrollo ágil y DevOps. Al dividir las aplicaciones en componentes modulares, los equipos de desarrollo pueden trabajar en partes específicas de la aplicación de forma independiente, lo que facilita la colaboración y la implementación continua. Además, proporcionan una base sólida para la automatización, agilizando así el proceso de desarrollo, prueba y despliegue. En cuanto a la gestión, la tecnología de contenedores ha llevado al desarrollo de orquestadores como Kubernetes, que simplifican la gestión y escalabilidad de aplicaciones en contenedores a gran escala. Estas herramientas ofrecen capacidades avanzadas de orquestación, como el balanceo de carga, la auto recuperación y la gestión dinámica de recursos, facilitando la administración de aplicaciones complejas distribuidas en múltiples contenedores.

Otro beneficio clave es la optimización de recursos. Los contenedores permiten una mejor utilización de los recursos del sistema al compartir el mismo núcleo del sistema operativo y otras dependencias subyacentes. Esto reduce el consumo de recursos en comparación con las máquinas virtuales tradicionales, mejorando así la eficiencia operativa y la economía de escala.

Finalmente, es importante destacar que la tecnología de contenedores ha sido ampliamente adoptada por la industria y la comunidad de desarrollo. Empresas líderes y organizaciones de código abierto han invertido significativamente en el desarrollo y mejora de soluciones de contenedores, lo que ha llevado a un ecosistema robusto y maduro. Esta adopción generalizada significa que hay una amplia cantidad de recursos, herramientas y conocimientos disponibles para cualquier persona interesada en trabajar con contenedores.

Además de las ventajas directas que las tecnologías de contenedores ofrecen en términos de eficiencia, portabilidad y gestión simplificada de aplicaciones, su impacto en la industria y en la forma en que se desarrollan y operan las aplicaciones ha sido profundo y transformador.

Por un lado, las tecnologías de contenedores han sido un pilar fundamental en la transformación digital de las empresas. Han permitido a las organizaciones migrar aplicaciones heredadas a entornos modernos y ágiles, lo que les ha permitido aprovechar los beneficios de la nube y la computación distribuida. Esta transformación ha mejorado la flexibilidad empresarial, permitiendo una rápida adaptación a las demandas del mercado y una entrega más rápida de servicios y productos.

Igualmente, la adopción de contenedores ha llevado a un cambio cultural en el desarrollo y las operaciones de TI. La colaboración estrecha entre los equipos de desarrollo y operaciones (DevOps) se ha vuelto esencial para aprovechar al máximo las capacidades de contenerización. Este enfoque colaborativo ha roto las barreras tradicionales entre los equipos, fomentando la comunicación constante y la automatización en todas las etapas del ciclo de vida de las aplicaciones.

Por último, es importante destacar que las tecnologías de contenedores también han tenido un impacto positivo en el medio ambiente. Al permitir una mejor utilización de los recursos de hardware y facilitar la gestión eficiente de aplicaciones a través de la virtualización a nivel de sistema operativo, los contenedores han contribuido a la reducción del consumo de energía y la huella de carbono en centros de datos a gran escala.

En resumen, la tecnología de contenedores ha revolucionado la forma en que desarrollamos, desplegamos y gestionamos aplicaciones. Su eficiencia, portabilidad y facilidad de gestión hacen que sea una opción atractiva para cualquier entorno de desarrollo y despliegue. Sin embargo, es importante considerar cuidadosamente los requisitos específicos del proyecto antes de decidir utilizar contenedores, ya que no son la solución ideal para todos los escenarios. En última instancia, la adopción inteligente de la tecnología de contenedores puede mejorar significativamente la velocidad, la agilidad y la confiabilidad de los procesos de desarrollo y despliegue de aplicaciones en la era de la computación en la nube y la informática distribuida.

Bibliografía

[1] *Containerization explained | IBM*. (n.d.). https://www.ibm.com/topics/containerization
[2] *What is containerization?* (n.d.). https://www.redhat.com/en/topics/cloud-native-apps/what-is-containerization
[3] *¿Qué son los contenedores? | Google Cloud*. (n.d.). Google Cloud. https://cloud.google.com/learn/what-are-containers?hl=es
[4] *What is a Container? | Docker*. (n.d.). Docker. https://www.docker.com/resources/what-container/
[5] Aqua Security. (2022, December 7). *Container Images: Architecture and best practices - AQUA*. Aqua. https://www.aquasec.com/cloud-native-academy/container-security/container-images/
[6] *A brief history of container technology*. (n.d.). Engineering Education (EngEd) Program | Section. https://www.section.io/engineering-education/history-of-container-technology/
[7] Qasim. (2023, August 22). LXC Vs Docker: A Highlight Of 6 Differences. *RedSwitches*. https://www.redswitches.com/blog/lxc-vs-docker/#Introducing-Linux-Containers-(LXC)
[8] *¿Qué es Docker y cómo funciona? Ventajas de los contenedores Docker*. (n.d.). https://www.redhat.com/es/topics/containers/what-is-docker
[9] *En qué consistían CoreOS y CoreOS Container Linux*. (n.d.). https://www.redhat.com/es/technologies/cloud-computing/openshift/what-was-coreos
[10] Mabe, D. (2019, November 21). *Project Atomic*. https://projectatomic.io/
[11] *Linux containers*. (n.d.). https://linuxcontainers.org/
[12] *Linux Containers - LXD - Has been moved to Canonical*. (n.d.). https://linuxcontainers.org/lxd/
[13] *OpenVZ Virtuozzo Containers Wiki*. (n.d.). https://wiki.openvz.org/Main_Page
[14] *Docker vs CoreOS Rkt | UpGuard*. (n.d.). https://www.upguard.com/blog/docker-vs-coreos#:~:text=CoreOS%20released%20rkt%20in%202014,user%20privileges
[15] Pedamkar, P. (2023, March 27). *RKT vs Docker*. EDUCBA. https://www.educba.com/rkt-vs-docker/
[16] Nolle, T. (2017, June 1). *When to use Docker alternatives rkt and LXD*. IT Operations. https://www.techtarget.com/searchitoperations/tip/When-to-use-Docker-alternatives-rkt-and-LXD#:~:text=The%20prime%20Docker%20alternative%20is,major%20problems%20with%20Docker%20security
[17] *¿Qué es Docker y cómo funciona? Ventajas de los contenedores Docker*. (n.d.-b). https://www.redhat.com/es/topics/containers/what-is-docker
[18] Mutai, J. (2023, July 21). How to interact with Containerd Runtime in Kubernetes | ComputingForGeeks. *ComputingForGeeks*. https://computingforgeeks.com/interact-with-containerd-runtime-in-kubernetes/#:~:text=Containerd%20is%20an%20open%20source%2C,OCI
[19] microsoft. (n.d.). *GitHub - microsoft/confidential-containers-containerd: An open and reliable container runtime*. GitHub. https://github.com/microsoft/confidential-containers-containerd
[20] Tilwani, R. (n.d.). *Containerd vs. Docker vs. Cri-o - Which Container Runtime to Choose?* Humalect.com. https://humalect.com/blog/containerd-vs-docker#:~:text=Containerd%20is%20an%20open,run%20applications%20in%20isolated%20environments
[21] *What is Podman?* (n.d.). https://www.redhat.com/en/topics/containers/what-is-podman#:~:text=Podman%20,ecosystem%20using%20the%20libpod%20library
[22] *What is Podman? — Podman documentation*. (n.d.). https://docs.podman.io/en/latest/

[23] Containers. (n.d.). *podman/README.md at main · containers/podman*. GitHub. https://github.com/containers/podman/blob/main/README.md
[24] *Orquestación de contenedores para producción*. (n.d.). Kubernetes. https://kubernetes.io/es/
[25] *Borg: The predecessor to Kubernetes*. (2023, January 16). Kubernetes. https://kubernetes.io/blog/2015/04/borg-predecessor-to-kubernetes/
[26] Losada, C. (2022, August 12). *¿Qué es Docker Swarm? - Making Science*. Making Science. https://www.makingscience.es/blog/que-es-docker-swarm/
[27] *Red Hat OpenShift simplifica la organización de los contenedores*. (n.d.). https://www.redhat.com/es/technologies/cloud-computing/openshift
[28] Autentia. (2018, September 20). *Nomad vs. Kubernetes - Autentia*. Autentia. https://www.autentia.com/2018/09/20/nomad-vs-kubernetes/
[29] Sahid. (2023, September 18). Openshift vs Kubernetes: What is the Difference? *Cloud Training Program*. https://k21academy.com/openshift/openshift-vs-kubernetes/
[30] *Docker Swarm vs Kubernetes: Top Differences*. (n.d.). https://www.knowledgehut.com/blog/devops/docker-swarm-vs-kubernetes
[31] *Nomad vs. Kubernetes | Nomad | HashiCorp Developer*. (n.d.). Nomad Vs. Kubernetes | Nomad | HashiCorp Developer. https://developer.hashicorp.com/nomad/docs/nomad-vs-kubernetes
[32] *¿Qué es el control de acceso basado en roles (RBAC)?* (n.d.). https://www.entrust.com/es/resources/faq/what-is-role-based-access-control
[33] colaboradores de Wikipedia. (2023, October 10). *Lista de control de acceso*. Wikipedia, La Enciclopedia Libre. https://es.wikipedia.org/wiki/Lista_de_control_de_acceso
[34] Thakur, S. (2023, September 16). "Revolutionizing Industries: How Kubernetes solves Real-World use Cases." *Medium*. https://sonamthakur7172.medium.com/revolutionizing-industries-how-kubernetes-solves-real-world-use-cases-608c4b666274
[35] Prometheus. (n.d.). *Prometheus - Monitoring system & time series database*. https://prometheus.io/