UNIVERSIDAD TÉCNICA ESTATAL DE QUEVEDO

FACULTAD DE CIENCIAS DE LA INGENIERÍA

ASIGNATURA:

APLICACIONES DISTRIBUIDAS

INTEGRANTES:

CASANOVA MORANTE HÉCTOR

HERRERA SILVA ALEXANDER

MOLINA MOSQUERA VICTOR

TEMA:

KUBERNETES Y DOCKER

DOCENTE:

GUERRERO ULLOA GLEISTON

1. INTRODUCCIÓN

Las aplicaciones son cada vez más complejas y se despliegan en entornos distribuidos. La gestión de estas aplicaciones y su infraestructura se vuelve crítica a gran escala y a su vez puede complicar la coordinación entre el equipo de desarrollo. Para abordar estos desafíos, se han desarrollado soluciones como los contenedores, que son unidades de software que encapsulan todo lo necesario para ejecutar una aplicación, incluyendo el código, las bibliotecas, las herramientas del sistema y las dependencias.

Los contenedores proporcionan una serie de beneficios, como la portabilidad, la consistencia y el aislamiento. Esto significa que una aplicación puede ser empaquetada en un contenedor en un entorno de desarrollo y luego ejecutarse de manera consistente en cualquier otro entorno, independientemente de las diferencias en el sistema operativo, la configuración del hardware o el software. Entre las principales herramientas que permiten realizar estas operaciones se encuentran Docker y Kubernetes.

Docker es una tecnología que revolucionó el desarrollo de software al introducir el concepto de contenedores. Los contenedores son unidades de software autónomas que incluyen todo lo necesario para ejecutar una aplicación, desde el código hasta las bibliotecas, las herramientas del sistema y las dependencias. Esto garantiza que la aplicación funcione de manera consistente en cualquier entorno, ya sea en un servidor local o en la nube. Docker también facilita la gestión de imágenes, que son plantillas para crear contenedores, y registros, que son repositorios donde se almacenan las imágenes para su distribución.

Por otro lado, Kubernetes es un sistema de orquestación de contenedores de código abierto que automatiza el despliegue, la escalabilidad y la gestión de aplicaciones en contenedores. A diferencia de Docker, que se centra en la creación y gestión de contenedores, Kubernetes se ocupa de la gestión de los contenedores en un clúster, proporcionando características como el balanceo de carga, la escalabilidad automática y la tolerancia a fallos. Kubernetes utiliza un enfoque declarativo para la configuración, lo que significa que se define el estado deseado de la aplicación y Kubernetes se encarga de alcanzarlo.

Uno de los aspectos clave de Kubernetes es su capacidad para manejar aplicaciones de manera distribuida y escalable. Esto significa que puede manejar una gran cantidad de contenedores y ajustar la cantidad de recursos asignados a cada contenedor según las necesidades de la aplicación. Además, Kubernetes proporciona mecanismos para la comunicación entre contenedores y para la distribución de datos entre ellos, lo que facilita la construcción de aplicaciones microservicios.

En resumen, Docker y Kubernetes son dos tecnologías fundamentales en el ecosistema de la computación en la nube y la infraestructura como código. Docker proporciona la capacidad de empaquetar aplicaciones en contenedores, mientras que Kubernetes proporciona la orquestación para ejecutar y gestionar estos contenedores de manera eficiente y escalable. Juntos, forman la base para la construcción y despliegue de aplicaciones modernas, permitiendo a los desarrolladores centrarse en la lógica de la aplicación en lugar de las complejidades de la infraestructura.

2. OBJETIVOS

2.1. OBJETIVO GENERAL

- Reconocer la importancia de las herramientas y plataformas como Docker y Kubernetes en el desarrollo de aplicaciones distribuidas.

2.2. OBJETIVOS ESPECÍFICOS

- Definir los conceptos, funcionamiento y arquitectura de Docker y Kubernetes para una correcta interpretación.
- Comprender las características, ventajas y desventajas que se presentan en Docker y Kubernetes.
- Desarollar un ejemplo práctico con las herramientas Docker y Kubernetes.

3. PLATAFORMAS DISTRIBUIDAS

Sistemas en los que los elementos de hardware y software, presentes en computadoras interconectadas, interactúan y sincronizan sus funciones mediante el intercambio de mensajes para alcanzar un objetivo específico, la comunicación se realiza mediante un protocolo previamente definido.[1] Un sistema distribuido se refiere a programas informáticos que emplean recursos computacionales en varios nodos de procesamiento para alcanzar un objetivo compartido.[2]

Las aplicaciones distribuidas son aquellas en las que la funcionalidad se divide en múltiples partes independientes, conocidas como microservicios.[2] Cada microservicio se encarga de una responsabilidad específica dentro de la aplicación y puede ejecutarse en un contenedor separado, lo que permite una mayor flexibilidad y escalabilidad.[2], [3]También, proporcionan transparencia, ya que los usuarios pueden interactuar con el sistema como si fuera una única computadora, sin tener que preocuparse por la configuración y mantenimiento de las máquinas individuales. Esto facilita el desarrollo y la gestión de aplicaciones, ya que los desarrolladores pueden centrarse en la lógica de la aplicación y no en la arquitectura.[3]

Las plataformas distribuidas son fundamentales en la computación moderna, especialmente en la computación en la nube.[3] Estas plataformas permiten que las aplicaciones se ejecuten en múltiples nodos de computación, lo que facilita la escalabilidad y la coordinación entre equipos de desarrollo. La computación en la nube, en particular, se basa en la distribución de recursos de computación y almacenamiento a través de Internet, lo que permite a las empresas acceder a estos recursos de manera flexible y rentable.[2], [4]

La computación en la nube y la computación distribuida se complementan en la medida en que la nube distribuida facilita el despliegue y la gestión de aplicaciones en múltiples ubicaciones geográficas.[5] Esto se puede notar principalmente en aquellas aplicaciones y sistemas que procesan grandes volúmenes de datos o que requieren una baja latencia, como las aplicaciones de análisis en tiempo real o las aplicaciones de IoT.[6]

4. COMPUTACIÓN EN LA NUBE

La computación en la nube, también conocida como cloud computing, es un servicio informático ofrecido a través de Internet.[3] Las entidades proveedoras de servicios en la nube generalmente ofrecen un servicio básico de manera gratuita, mientras que cobran por funcionalidades más avanzadas.[4] Este enfoque permite almacenar archivos en la nube en lugar del disco duro de la computadora, facilitando el acceso desde cualquier dispositivo, en cualquier parte del mundo y en cualquier momento.[3], [4]

La computación en la nube, conocido también como servicios en la nube, informática en la nube, nube de cómputo o nube de conceptos, es un paradigma que permite ofrecer servicios de computación a través de Internet.[5] Esta representa un importante desafío para los departamentos de Tecnología de la Información (TI), que están a punto de enfrentarse a sus efectos en las empresas actuales. Los líderes de los departamentos de TI deben evaluar cómo se necesita adquirir y compartir información en entornos colaborativos, para salvaguardar los intereses organizativos.[6]

4.1. CARACTERISTICAS

Disponibilidad: La computación en la nube puede ser vista como un servicio continuo o disponible las 24 horas, los 7 días de la semana. Se trata de un servicio accesible en cualquier momento y lugar con requisitos mínimos de conexión.[7]

Accesibilidad: Presenta diversas formas y métodos de acceso, permitiendo la entrada desde dispositivos que solo cuenten con un navegador web, así como desde dispositivos más avanzados como tabletas, teléfonos móviles, entre otros.[5], [8]

Servicio bajo demanda: Los usuarios tienen la capacidad de requerir recursos tanto de hardware como de software según sus necesidades. En términos de hardware, esto podría implicar la solicitud de una mayor capacidad de almacenamiento, mientras que, en el ámbito del software, se trata de solicitar nuevos servicios o aplicaciones informáticas.[5]

Reducción de costes: Las empresas tienen la oportunidad de reducir costos al adoptar la computación en la nube, ya que abonan por el uso del servicio en lugar de invertir en los equipos de hardware. Este enfoque resulta transparente para el usuario.[7]

Adaptabilidad: Las aplicaciones basadas en la nube tienen la capacidad de adaptarse a cualquier sistema en el que estén implementadas y pueden manejar un gran número de usuarios simultáneos. [8], [9]

4.2. VENTAJAS

- La tecnología es adecuada para su uso en cualquier dispositivo que cuente con conexión a internet, tales como teléfonos celulares, tabletas y computadoras de diversos tipos.[5], [9]
- Reduce los gastos de adquisición de equipos de cómputo y al pago de licencias por cada programa informático necesario.[4]
- En caso de pérdida de información, su recuperación es fácil debido al servicio de almacenamiento en la nube.[6], [7]

- Las compañías tienen la opción de transformar la seguridad en servicios y soluciones, mejorar la gestión de las Tecnologías de la Información, actualizar las estrategias empresariales y liberar recursos financieros.[10]

4.3. DESVENTAJAS

- Ausencia de supervisión sobre los recursos, cuando la infraestructura y la aplicación operan en servidores en la nube, controlados por el proveedor, el cliente carece de autoridad sobre los recursos y, incluso, sobre su información tras su transferencia.[9]
- Existe la posibilidad de perder información debido a las vastas cantidades de datos que se gestionan.[7], [8]
- Genera una relación de dependencia entre el usuario y el proveedor de servicios, dada la conveniencia que este último proporciona.[11]

5. DOCKER

Docker es una plataforma de código abierto (open source) que proporciona una solución para el desarrollo, envío y ejecución de aplicaciones. Permite aislar las aplicaciones de la infraestructura subyacente, lo que facilita la entrega rápida de software [12]. Con Docker, los desarrolladores pueden gestionar su infraestructura de la misma manera que gestionan sus aplicaciones, aprovechando las metodologías de Docker para el envío, pruebas y despliegue de código, lo que reduce significativamente el retraso entre la escritura de código y su ejecución en producción [13].

Entre las funciones que ofrece Docker, se encuentra la capacidad de empaquetar y ejecutar una aplicación en un entorno aislado llamado contenedor. Este aislamiento y seguridad permiten ejecutar múltiples contenedores simultáneamente en un host dado. Los contenedores son incluyen las funciones y operaciones necesarias para ejecutar la aplicación, por lo que no es necesario depender de lo que esté instalado en el host [13]. Docker proporciona herramientas y una plataforma para gestionar el ciclo de vida de los contenedores, desde el desarrollo de la aplicación y sus componentes de soporte hasta el despliegue en el entorno de producción [14].

Al trabajar con su propio ciclo de vida e implementar funciones que optimizan las pruebas e implementación del sistema permite a los desarrolladores trabajar en entornos estandarizados utilizando contenedores locales que proporcionan las aplicaciones y servicios [15]. Los contenedores son ideales para los flujos de trabajo de integración continua y entrega continua (CI/CD). Los desarrolladores escriben código localmente y comparten su trabajo con sus colegas utilizando contenedores Docker [13], [14]. Utilizan Docker para enviar sus aplicaciones a un entorno de prueba y ejecutar pruebas automatizadas y manuales. Cuando se encuentran errores, pueden corregirlos en el entorno de desarrollo y volver a desplegarlos en el entorno de prueba para realizar pruebas y validación [15], [16].

Docker utiliza una arquitectura cliente-servidor. El cliente se comunica con el daemon de Docker, que realiza el trabajo pesado de construir, ejecutar y distribuir los contenedores. El cliente y el daemon pueden ejecutarse en el mismo sistema, o se puede conectar un cliente a un daemon remoto [13]. El cliente y el daemon se comunican utilizando API REST, a través de sockets UNIX o una interfaz de red. También es posible trabajar con Docker Compose, que permite trabajar con aplicaciones compuestas por un conjunto de contenedores [14].

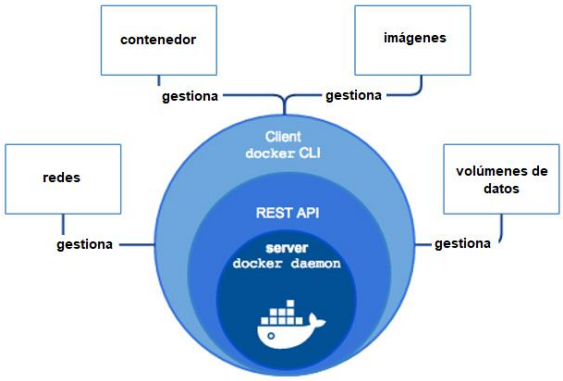


Ilustración 1. Representación de las relaciones entre los componentes de Docker (Traducida) [13].

Cuando se utiliza Docker, se crean y utilizan imágenes, contenedores, redes, volúmenes, plugins y otros objetos; una imagen es una plantilla de solo lectura con instrucciones que permiten a los usuarios crear un contenedor [16]. Los usuarios o clientes pueden manipular la imagen con las operaciones básicas; es decir, crear, iniciar, detener, mover o eliminar un contenedor utilizando la API de Docker o la CLI (Interfaz de línea de comandos) [13], [17].

Docker está escrito en el lenguaje de programación Go y aprovecha varias características del kernel de Linux para ofrecer su funcionalidad [13]. Docker utiliza una tecnología llamada espacios de nombres para proporcionar el espacio de trabajo aislado llamado contenedor. Cuando se ejecuta un contenedor, Docker crea un conjunto de espacios de nombres para ese contenedor. Estos espacios de nombres proporcionan una capa de aislamiento [17]. Cada aspecto de un contenedor se ejecuta en un espacio de nombres separado y su acceso está limitado a ese espacio de nombres [14], [16].

5.1. HISTORIA Y EVOLUCIÓN DE DOCKER

Docker es una plataforma que ha revolucionado la forma en que los desarrolladores despliegan y gestionan aplicaciones [12]. Fue fundada en 2010 por Solomon Hykes con el objetivo de ofrecer una alternativa más eficiente a las máquinas virtuales para el despliegue de aplicaciones en la nube; es decir, en lugar de las máquinas virtuales, que son más pesadas y consumen más recursos, Docker propuso el uso de contenedores de software, que son unidades de software livianas y portables [17], [18].

La primera versión de Docker fue lanzada en 2013, y desde entonces ha experimentado un rápido crecimiento y adopción en la industria del software [18]. En 2013, Docker se convirtió en un proyecto de código abierto, lo que permitió a la comunidad de desarrolladores contribuir y mejorar la plataforma. En 2017, Docker fue adquirida por Microsoft, pero se mantuvo como una entidad separada dentro de la estructura de Microsoft [19].

A lo largo de los años, Docker ha continuado desarrollándose y mejorando sus características. Las imágenes de Docker, que son la base de los contenedores, se han vuelto más eficientes y fáciles de usar. Docker también ha expandido su ecosistema con herramientas como Docker Compose y Docker Swarm para facilitar la orquestación de contenedores [17], [18].

Hoy en día, Docker es ampliamente adoptado en la industria del software y es una parte integral de muchas prácticas de DevOps y desarrollo de aplicaciones en la nube [16]. La historia de Docker demuestra cómo una idea innovadora puede transformar la forma en que se desarrollan y despliegan las aplicaciones, permitiendo a los desarrolladores empaquetar sus aplicaciones con todas sus dependencias en un contenedor que puede ejecutarse en cualquier sistema operativo compatible [15], [18].

5.2. ARQUITECTURA DE DOCKER

La arquitectura cliente-servidor es un modelo de diseño de software en el que las funcionalidades de una aplicación se dividen en dos partes: el cliente y el servidor. En este modelo, el cliente solicita servicios o recursos al servidor, y el servidor responde a estas solicitudes [12], [13]. Para cumplir la comunicación entre el cliente y el servidor, Docker implementa algunas características adicionales para la comunicación, actualización e implementación de imágenes de contenedores [17].

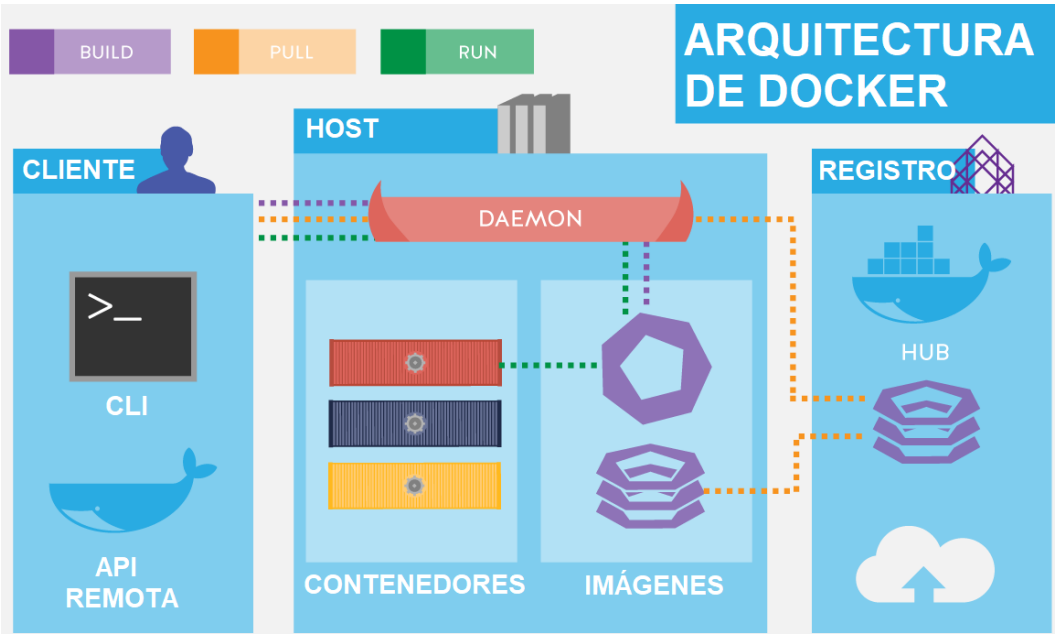


Ilustración 2. Arquitectura de Docker: Comunicación entre componentes para la construcción y manipulación de contenedores (Traducida). Obtenido de: https://devopedia.org/docker

5.3. COMPONENTES DE DOCKER

Existe una serie de componentes superficiales o que son conocidos por los usuarios de Docker para identificar las acciones que pueden ser realizadas [15]. Las más utilizadas y conocidas son el daemon o demonio de Docker, el registro las imágenes y las redes; sin embargo, existen otros componentes que son el complemento que permiten el correcto funcionamiento y de cierta forma optimizan las operaciones [17], [19]. A continuación, se presentan los componentes con una breve descripción de la función que realizan.

- **Registro de Docker (Docker Registry):** Es un repositorio centralizado donde se almacenan las imágenes de Docker. Docker Hub es una instancia pública de Docker Registry que proporciona imágenes preconstruidas para diversas aplicaciones y servicios [12].
- **Imágenes de Docker (Docker Images):** Son plantillas que contienen el sistema operativo base y las aplicaciones necesarias para ejecutar un contenedor. Las imágenes se construyen en capas y se pueden descargar o subir a un Docker Registry [13].
- **Motor de Docker API (Docker Engine REST API):** Es una API que permite a las aplicaciones interactuar con el daemon Docker. Puede ser accedida por un cliente HTTP y es utilizada por el cliente Docker para enviar comandos al daemon [12].
- **Daemon de Docker (Docker Daemon):** Es el proceso de fondo que administra las imágenes de Docker, los contenedores, las redes y los volúmenes de almacenamiento. El daemon Docker escucha y procesa las solicitudes de la API de Docker y realiza las acciones necesarias, como construir, ejecutar y distribuir contenedores Docker [14].
- **Docker Cliente:** Es la interfaz de usuario con la que interactúan los desarrolladores y administradores de sistemas. El cliente Docker se comunica con el daemon Docker para enviar comandos y recibir respuestas. Puede ejecutarse en el mismo sistema que el daemon o conectarse a un daemon Docker remoto [12], [13] .
- **Docker Compuesto (Docker Compose):** Es una herramienta que permite definir y administrar aplicaciones multi-contenedor con un archivo de configuración en formato YAML. Facilita la creación de aplicaciones que requieren múltiples servicios interconectados [14].
- **Docker de escritorio (Docker Desktop):** Es una aplicación para Mac, Windows o Linux que incluye el daemon Docker, el cliente Docker, Docker Compose, Docker Content Trust, Kubernetes y el Asistente de Credenciales. Proporciona un entorno para construir y compartir aplicaciones contenerizadas y microservicios [13], [15].

- **Redes (Networking):** Docker implementa la red en una forma orientada a la aplicación, proporcionando varias opciones de red mientras mantiene suficiente abstracción para los desarrolladores de aplicaciones. Hay dos tipos de redes disponibles: la red predeterminada de Docker y las redes definidas por el usuario [12], [13].
- **Almacenamiento (Storage):** Docker gestiona el almacenamiento de los contenedores, permitiendo montar volúmenes y utilizar mecanismos de almacenamiento persistente para los datos de la aplicación [16].

5.4. ¿CÓMO FUNCIONA DOCKER?

El objetivo de docker es estandarizar una forma de agrupar un conjunto de aplicaciones en contenedores que contengan la lógica, archivos y recursos necesarios [19]. Estos se crean a partir de una máquina virtual con el kernel de Linux. Es decir, no requiere instalar completamente todos los drivers y componentes de alguna distribución de Linux, sino solo el kernel. Para entenderlo, se procede a realizar una comparación entre los contenedores creados en el kernel versus los hipervisores que crean un sistema operativo virtual completo [13], [20].

Docker opera a nivel de sistema operativo, permitiendo la ejecución de aplicaciones dentro de contenedores que comparten el kernel del sistema operativo del host [12], [13]. Esto significa que los contenedores son más ligeros y rápidos de iniciar en comparación con las máquinas virtuales, ya que no requieren un sistema operativo completo para cada aplicación [15]. Los contenedores de Docker son portátiles y pueden moverse entre diferentes hosts con Docker instalado, lo que facilita su implementación y escalabilidad [14]. Además, Docker utiliza un sistema de capas de archivos de solo lectura, lo que mejora la eficiencia en el uso del espacio y la escritura de datos. Sin embargo, los contenedores son menos seguros que las máquinas virtuales, ya que, si el host se ve comprometido, también pueden ser vulnerables [20].

Por otro lado, los hipervisores proporcionan virtualización a nivel de hardware, permitiendo la ejecución de múltiples sistemas operativos en una sola máquina física [15]. Cada máquina virtual en un hipervisor tiene su propio sistema operativo, lo que significa que son más pesadas y menos portátiles que los contenedores de Docker [21]. Las máquinas virtuales son aisladas entre sí y tienen acceso a los recursos del sistema asignados por el hipervisor, lo que las hace más seguras que los contenedores, ya que están aisladas y no comparten el mismo kernel del sistema operativo host. Sin embargo, las máquinas virtuales son más costosas y tardan más en arrancar debido a que cada una necesita su propio sistema operativo [14], [16].

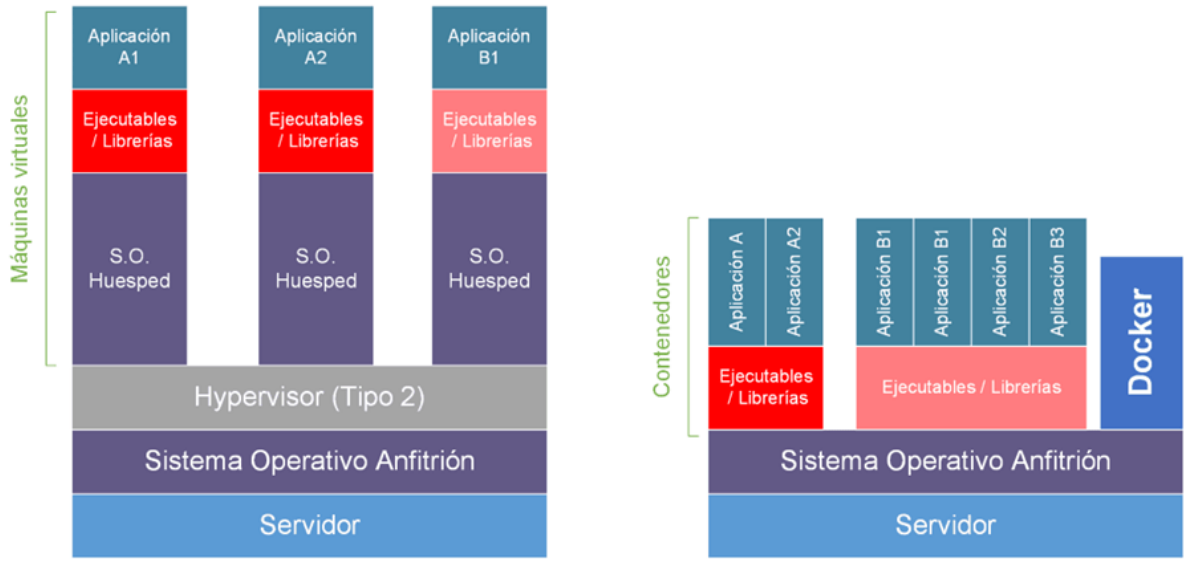


Ilustración 3. Diferencia entre las máquinas virtuales y los contenedores de Docker. Autor: Héctor Casanova.

El cliente es la interfaz que interactúa con el usuario y envía las solicitudes al servidor. Puede ser una aplicación de escritorio, una aplicación web, un dispositivo móvil o cualquier otro tipo de cliente que pueda comunicarse con el servidor a través de una red o de sockets UNIX [20]. Por otro lado, el servidor, es el componente que procesa las solicitudes del cliente. En el caso de Docker, el servidor es el daemon Docker, que es un proceso de fondo que se encarga de gestionar los contenedores, imágenes, redes y almacenamiento [21]. El daemon Docker escucha las solicitudes de la API Docker del cliente Docker y realiza las acciones correspondientes [13], [22].

La comunicación entre el cliente y el servidor se realiza a través de una API (Interfaz de Programación de Aplicaciones) [12]. Esta API permite que los componentes de software se comuniquen entre sí de manera estructurada y estandarizada, independientemente del lenguaje de programación o la plataforma en la que se ejecuten [22].

Esta comunicación vía API permite que los usuarios puedan controlar el ciclo de vida de los contenedores, construir imágenes, ejecutar y distribuir contenedores Docker [15], [23]. La arquitectura cliente-servidor de Docker facilita la gestión de contenedores y la interacción con la infraestructura de Docker, proporcionando una capa de abstracción que simplifica el proceso de desarrollo y despliegue de aplicaciones [24].

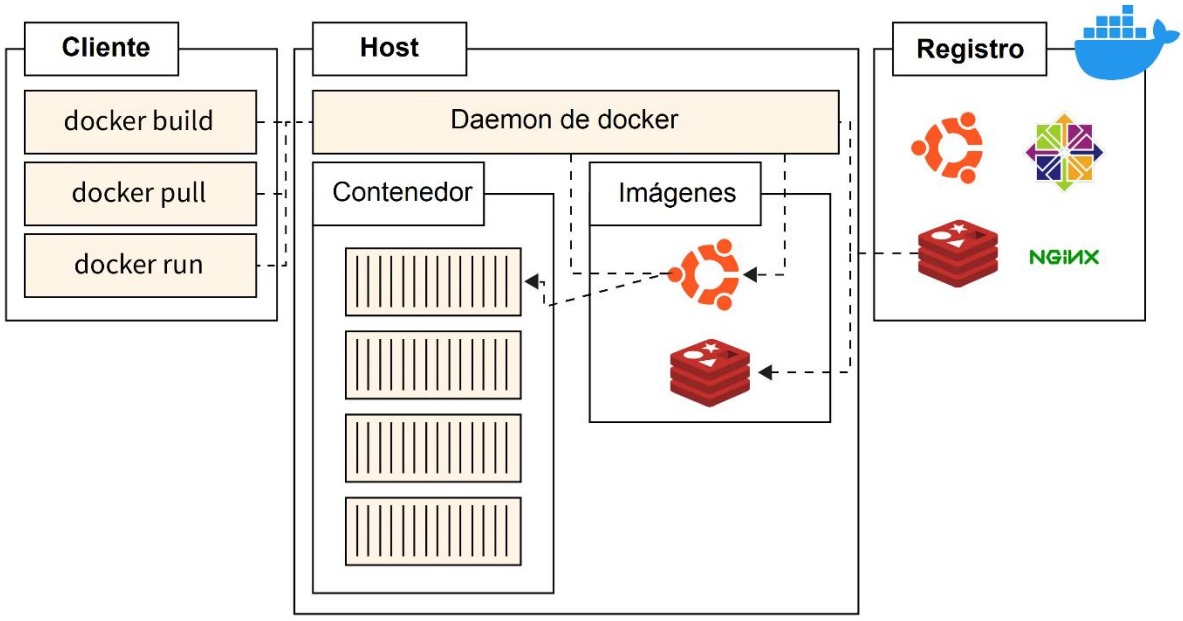


Ilustración 4. Relación entre componentes principales de Docker (Cliente, Host y Registro) [12].

Como resultado de ejecutar diferentes acciones sobre contenedores en Docker, esto asignará un proceso único sobre el kernel de Linux. Estas operaciones o procesos son detallados en el siguiente punto donde se define la composición de Docker, abarcando las funciones que utiliza para la gestión de procesos y red hasta la publicación y mantenimiento de los mismos.

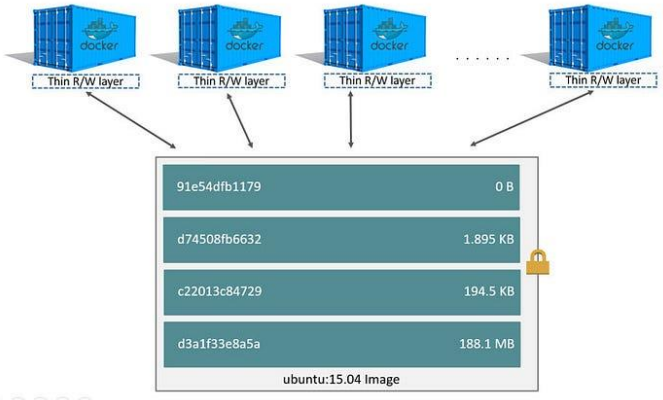


Ilustración 5. Generación de identificador para contenedores en el kernel de Linux [13].

5.5. COMPOSICIÓN DE DOCKER

Docker está programado en el lenguaje de programación Go y está basado en sistemas operativos Linux, Windows y macOS [13], [15]. Para su funcionamiento, Docker utiliza características del kernel de Linux, tales como:

5.5.1. LENGUAJE DE PROGRAMACIÓN GO

Docker, una plataforma de contenedores de código abierto, hace uso del lenguaje de programación Go por una serie de razones que se centran en la eficiencia, la simplicidad y la

robustez. Go, también conocido como Golang, es un lenguaje de programación compilado y estáticamente tipado diseñado por Google, conocido por su simplicidad y rendimiento eficiente [12], [17].

Una de las principales razones es la eficiencia y el rendimiento que Go proporciona. Los programas escritos en Go son eficientes y rápidos, lo que es esencial para un sistema como Docker que debe manejar la creación y gestión de contenedores de manera eficiente [20]. Otra ventaja de Go es su soporte para la concurrencia a través de gorutinas, lo que es útil para manejar múltiples tareas al mismo tiempo [15]. Esto es especialmente relevante en el contexto de Docker, donde se pueden estar ejecutando múltiples contenedores al mismo tiempo.

El tipado estático de Go es otra característica que lo hace atractivo para Docker [12]. El tipado estático ayuda a detectar errores durante la compilación en lugar de en tiempo de ejecución, lo que puede ser útil para la robustez del código de Docker. La portabilidad de los binarios de Go es otra ventaja importante [21]. El código realizado en el lenguaje de programación es portátil y pueden compilarse para diferentes sistemas operativos, lo que facilita la distribución y el uso de Docker en diferentes plataformas [22].

5.5.2. NAMESPACES

Los namespaces son una característica del kernel de Linux que permite aislar y compartir recursos de manera eficiente. Esto es fundamental para la plataforma de Docker, ya que permite a los contenedores ser ligeros, portátiles y seguros [14].

En un entorno de computación general, los procesos comparten el mismo conjunto de recursos, lo que puede llevar a conflictos y desafíos en la gestión de recursos [23]. Los namespaces proporcionan una capa de aislamiento, permitiendo que un conjunto de procesos vea un conjunto de recursos mientras que otro conjunto de procesos ve un conjunto diferente de recursos [24]. Docker utiliza namespaces para proporcionar un espacio de trabajo aislado llamado contenedor. Cuando se ejecuta un contenedor, Docker crea un conjunto de namespaces para ese contenedor. Esto significa que cada contenedor tiene su propio namespace aislado para procesos, redes, puntos de montaje y más [15], [25].

- **PID Namespace**: Este namespace aísla los identificadores de procesos (PIDs). Cada contenedor tiene su propio conjunto de PIDs, lo que significa que los procesos dentro de un contenedor no pueden ver o interactuar con los procesos de otros contenedores ni con los procesos del host [26].
- **NET Namespace**: Este namespace aísla la pila de red del contenedor. Cada contenedor tiene su propia pila de red con su propio conjunto de interfaces de red, tablas de

enrutamiento y reglas de firewall. Esto permite que los contenedores tengan su propia configuración de red y evite conflictos entre los contenedores en cuanto a la red [27].

- **IPC Namespace**: Este namespace aísla los recursos de comunicación entre procesos (IPC). Esto significa que los contenedores pueden tener su propia comunicación IPC sin interferir con la de otros contenedores o con el host [15].
- **MNT Namespace**: Este namespace aísla los puntos de montaje del sistema de archivos. Cada contenedor tiene su propio sistema de archivos raíz y los recursos montados solo aparecen dentro del contenedor que los montó. Esto evita que los contenedores interfieran con los archivos del host o con los archivos de otros contenedores [14], [16].
- **UTS Namespace**: Este namespace permite a cada contenedor tener su propio nombre de host y nombre de dominio. Esto evita conflictos entre los nombres de host y dominios de los contenedores y del host [15].

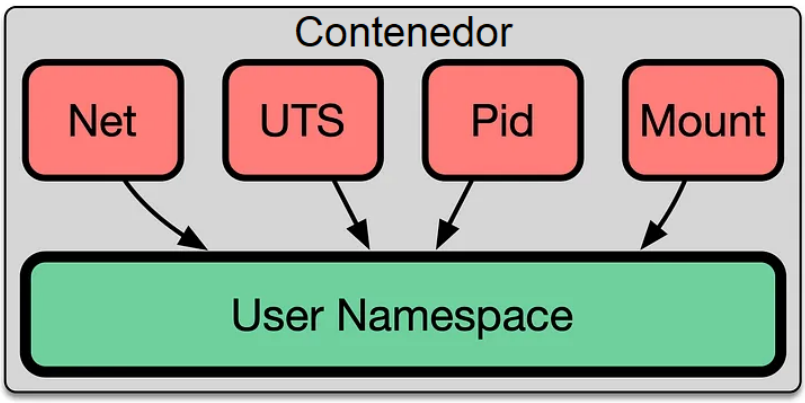


Ilustración 6. Representación gráfica de los tipos de namespaces ligados al del usuario creador del contenedor. Obtenido de: https://netflixtechblog.com/evolving-container-security-with-linux-user-namespaces-afbe3308c082?gi=c50d0d6604e4

5.5.3. CONTROL GROUPS

Los Control Groups (cgroups) son una característica del núcleo de Linux que permite limitar el acceso de los procesos y contenedores a los recursos del sistema, como la CPU, RAM, IOPS y la red [12], [13]. Los cgroups no solo agrupan procesos, sino que también proporcionan información sobre el uso de CPU, memoria y E/S de bloques, y también se pueden obtener métricas de uso de red [18].

Cuando se inicia un contenedor, Docker genera un conjunto de cgroups para ese contenedor, lo que permite controlar y restringir el acceso de un contenedor a los recursos del sistema host de Docker [22]. Por ejemplo, se pueden establecer límites en la cantidad de CPU que un contenedor puede usar, limitar la cantidad de memoria que puede consumir, o incluso limitar la cantidad de procesos que puede iniciar. Además, los cgroups se utilizan para recopilar

métricas cuando un contenedor se detiene, lo que es útil para conocer cuánta CPU, memoria, etc., ha utilizado un contenedor durante su ejecución [23].

En resumen, los cgroups son una herramienta esencial en el ecosistema de contenedores que permite un control detallado sobre los recursos del sistema que un contenedor puede consumir, asegurando que los contenedores no consuman todos los recursos disponibles y que puedan funcionar de manera eficiente en un entorno compartido [12], [17].

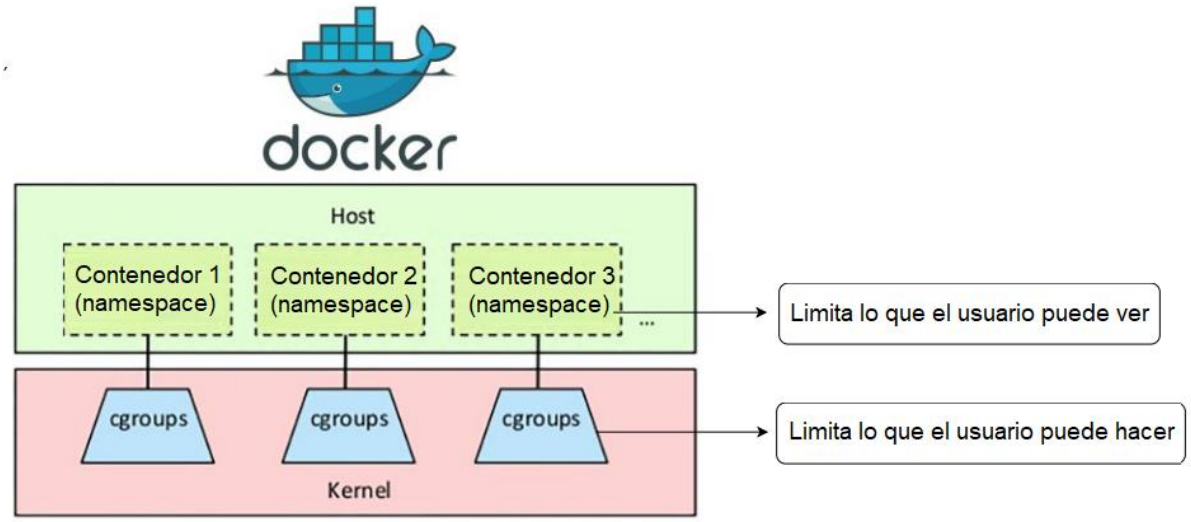


Ilustración 7. Uso de los cgroups en Docker mediante el kernel de una distribución de Linux. Obtenido de: https://medium.com/@mrdevsecops/namespace-vs-cgroup-60c832c6b8c8

5.5.4. UNIONFS

UnionFS en Docker es un sistema de archivos que combina diferentes capas de almacenamiento en una sola vista [26]. Esto se logra superponiendo las capas de imágenes de Docker, que son inmutables, y permitiendo que los contenedores modifiquen su propio espacio de trabajo sin afectar las demás capas [27].

Cuando Docker inicia un contenedor, crea un sistema de archivos que incluye todas las capas de la imagen y una capa de trabajo donde se pueden realizar cambios. Esto se hace para que los contenedores puedan tener acceso a los archivos de la imagen y modificarlos según sea necesario sin alterar la imagen original [15].

Docker utiliza varias implementaciones de UnionFS, como AUFS, Btrfs, ZFS, Overlay, Overlay2 y DeviceMapper, para lograr esta superposición de archivos. La elección de la implementación depende de la configuración del sistema operativo y los requisitos específicos de la aplicación [12]. UnionFS permite la eficiente gestión del almacenamiento de imágenes y contenedores, proporcionando aislamiento y optimización del espacio al evitar la duplicación de datos [14], [15].

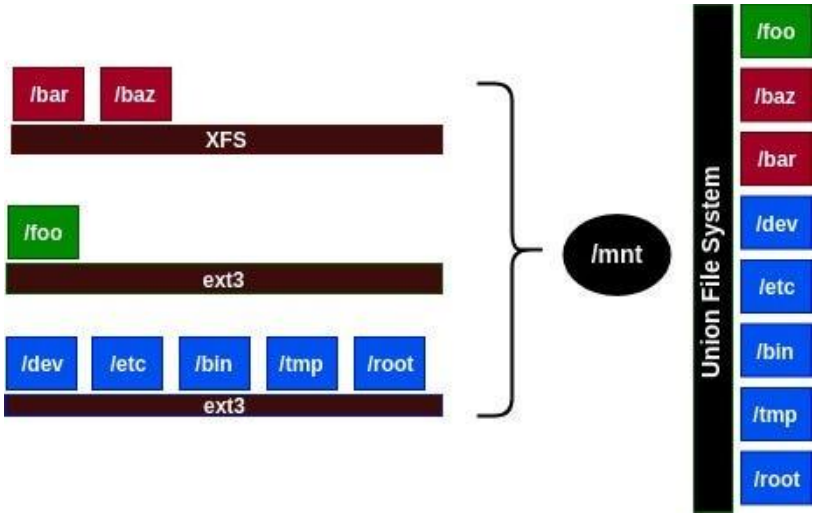


Ilustración 8. Agrupación de recursos mediante UnionFS de las distribuciones de Linux. Obtenido de: https://medium.com/@knoldus/unionfs-a-file-system-of-a-container-2136cd11a779

Es decir, Docker al trabajar con UFS o UnionFS se agrupan diferentes carpetas y su contenido dentro de la carpeta mnt que es donde se almacenan las particiones para la virtualización de otros sistemas operativos y los recursos necesarios para su funcionamiento. Así, el usuario puede acceder a esta carpeta y acceder a los archivos que sean requeridos sin que dejen de estar ordenados. Al permitir agrupar en un contenedor diferentes imágenes y aplicaciones, se utiliza el sistema UFS para el acceso en una distribución del sistema operativo Linux con los recursos necesarios.

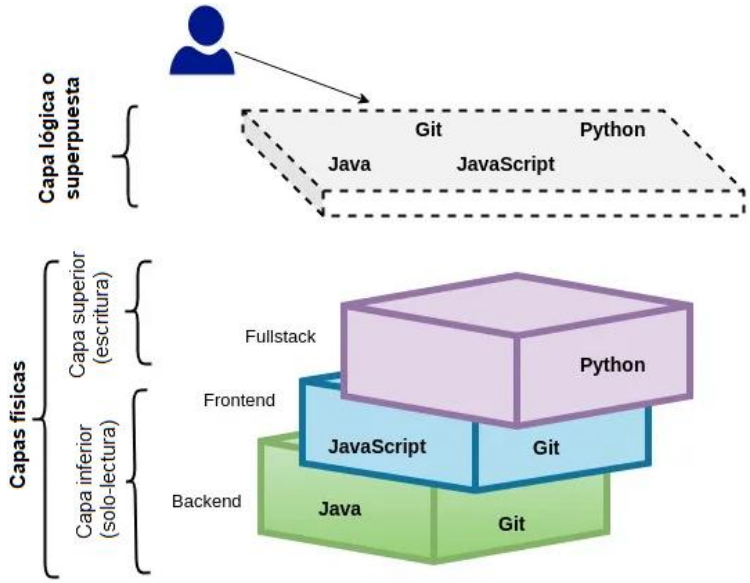


Ilustración 9. Representación de un ejemplo real que agrupa las diferentes imágenes disponibles para un contenedor. Obtenido de: https://medium.com/@knoldus/unionfs-a-file-system-of-a-container-2136cd11a779

Finalmente, un ejemplo real de los sistemas de archivos y su unión se puede visualizar en la ilustración anterior, donde existen diferentes capas o imágenes de los lenguajes de programación y son adjuntadas en diferentes capas según el tipo de acceso (lectura y escritura).

A la capa de imágenes y aplicaciones correspondientes a un contenedor se les llama capas físicas; y la agrupación de todos los recursos y archivos de la capa física es llamada capa superpuesta o capa lógica.

5.5.5. CAPABILITIES

Las capacidades o capabilities en Docker son un conjunto de privilegios que se pueden asignar a los contenedores, permitiendo controlar qué operaciones puede realizar un contenedor en el sistema host [12]. Estas capacidades son una forma de ajustar los privilegios de un contenedor para que solo tenga los permisos necesarios para funcionar correctamente, reduciendo así el riesgo de que un contenedor pueda realizar acciones maliciosas o no deseadas en el host del sistema [22].

Por defecto, Docker inicia los contenedores con un conjunto limitado de capacidades, lo que significa que los procesos dentro del contenedor no necesitan tener todos los privilegios de root para realizar ciertas operaciones [23]. Las capacidades pueden ser agregadas o eliminadas, permitiendo un perfil de seguridad no predeterminado. Esto puede hacer que Docker sea más seguro al eliminar capacidades o menos seguro al agregarlas. La mejor práctica es eliminar todas las capacidades excepto las que se requieren explícitamente para los procesos. Para ello, existen las siguientes funciones [13], [15], [16]:

- **Capabilities drop**: Es la capacidad de quitarle privilegios a un contenedor. Por ejemplo, si un contenedor no necesita ejecutarse como root, se puede quitarle el privilegio de root para que no tenga acceso a ciertos recursos del sistema [21].
- **Capabilities add**: Es la capacidad de añadir privilegios específicos a un contenedor. Por ejemplo, si un contenedor necesita acceso a la red, se puede añadirle el privilegio de network [25].
- **Capabilities full**: Es la capacidad de darle a un contenedor todos los privilegios. Esto no se recomienda por razones de seguridad, ya que un contenedor con todos los privilegios podría hacer cualquier cosa en el sistema host [27].

5.6. COMPONENTES EN TIEMPO DE EJECUCIÓN DOKER

El motor de Docker consta de dos elementos que permiten manipular los contenedores que son creados y las imágenes, estos son: Containerd y runc son elementos fundamentales dentro de la infraestructura de contenedores utilizada por Docker y otros sistemas de gestión de contenedores como Kubernetes [13].

Containerd es un entorno de ejecución de contenedores que se emplea para operar contenedores en un sistema operativo anfitrión. Es el componente que administra la creación, operación y

desactivación de contenedores [25]. Containerd se encarga de la gestión del ciclo de vida del contenedor, que incluye la preparación del entorno de ejecución, la configuración de la red, la asignación de volúmenes y la administración de cgroups para el control de recursos [14].

Containerd es conforme a la especificación OCI (Open Container Initiative), lo que implica que sigue los estándares de la industria para la ejecución de contenedores. Esto garantiza que los contenedores creados y ejecutados con Containerd sean portátiles y consistentes en diferentes plataformas y sistemas operativos [15], [24].

Runc es un componente a un nivel inferior en la capa de software de contenedores. Es un ejecutable ligero que se emplea para ejecutar un contenedor individual de acuerdo con las especificaciones OCI [23]. Runc se encarga de crear un entorno aislado para el contenedor utilizando namespaces de Linux, cgroups y otros mecanismos del kernel de Linux. Es responsable de gestionar el ciclo de vida de un contenedor, incluyendo la creación del entorno de ejecución, la configuración de la red y la asignación de volúmenes, y la administración de cgroups para el control de recursos [17], [25].

5.7. MOTOR DE DOCKER

La última capa del funcionamiento interno de Docker, se basa en la utilización de características propias del sistema y el uso de libnetwork que es una librería de red para gestionar un modelo de red basado en contenedores robustos mediante una interfaz de programación [13], [15].

Así mismo, existen otros elementos como graph, libcontainerd y plugins que son propios de Docker y permiten definir instrucciones para el almacenamiento de recursos dentro de los contenedores. Finalmente, con estas descripciones se muestra una representación de la estructura de Docker [27].

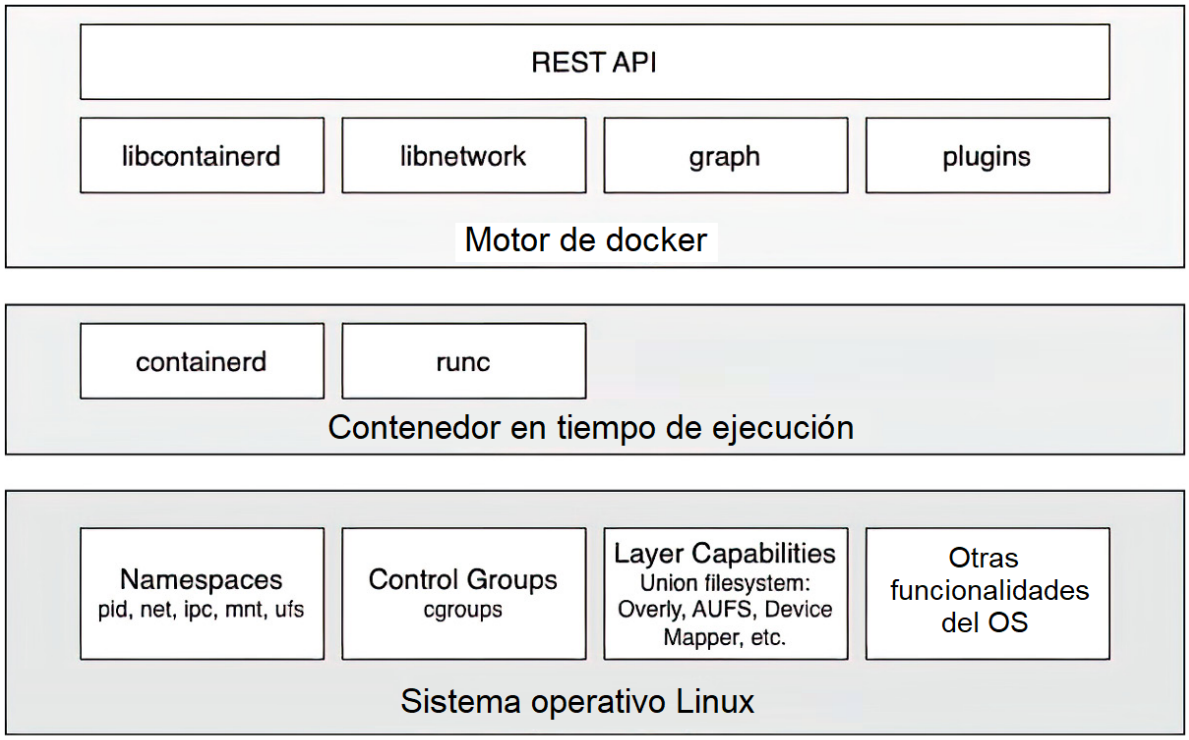


Ilustración 10. Arquitectura interna de Docker que describe los elementos utilizados para manipular los recursos de un contenedor. Obtenido de: https://www.amazon.com/Ultimate-Docker-Container-Book-containers/dp/1804613983

5.7.USOS DE DOCKER

Los usos de Docker abarcan procesos donde se requieren actualizaciones y despliegues constantes de aplicaciones. Además, Docker fue desarrollado con el objetivo de facilitar la implementación y desarrollo de cada fase del ciclo de vida del software; permitiendo así, que el trabajo sea fácil de realizar para los desarrolladores y fácil de acceder a revisar entregables por parte de los revisores o usuarios finales [16]. A continuación, se presenta algunos casos de uso:

- **Salud**: Los Institutos Nacionales de Salud (NIH) de Estados Unidos utilizan Docker para su movilidad y flexibilidad en la entrega de servicios de software de imágenes a más de 40 hospitales en el país. Docker permite al NIH probar nuevas tecnologías de imágenes en el sector hospitalario y asegurarse de que se estén utilizando las últimas tecnologías [24].
- **Educación**: Wiley Education Services (WES) proporciona servicios educativos en línea a más de 60 instituciones de educación superior. Utilizan Docker para innovar y experimentar con la creación de arquitecturas escalables. Docker les permite poner en marcha uno de sus sitios web en el mercado en poco tiempo, lo que ayuda a conectar a los estudiantes con los socios más rápidamente [25].

- **Desarrollo de software**: Las empresas pueden utilizar Docker para estandarizar los entornos de desarrollo, lo que facilita la colaboración entre los desarrolladores y acelera el proceso de desarrollo y pruebas. Docker también facilita la implementación continua, permitiendo a los desarrolladores desplegar sus aplicaciones de manera eficiente y rápida [27].
- **Implementación de microservicios**: Docker es fundamental en el desarrollo de aplicaciones basadas en microservicios, ya que permite el aislamiento de servicios y facilita la colaboración entre ellos [14]. Esto es especialmente útil en entornos donde las aplicaciones están compuestas por varios servicios que necesitan ser ejecutados en contenedores separados [16].
- **Procesamiento de datos**: Docker puede ser utilizado para proporcionar servicios de procesamiento de big data, permitiendo la ejecución de análisis y paquetes de datos en contenedores portátiles que pueden ser utilizados por usuarios sin conocimientos técnicos [17].
- **Entornos de prueba**: Docker es útil para crear entornos de prueba aislados para probar aplicaciones antes de desplegarlas en producción. Esto asegura que las aplicaciones funcionen como se espera y minimiza el riesgo de errores en producción [15], [16].

Finalmente, la ejecución de las acciones realizables por los usuarios puede resumirse en la siguiente ilustración.

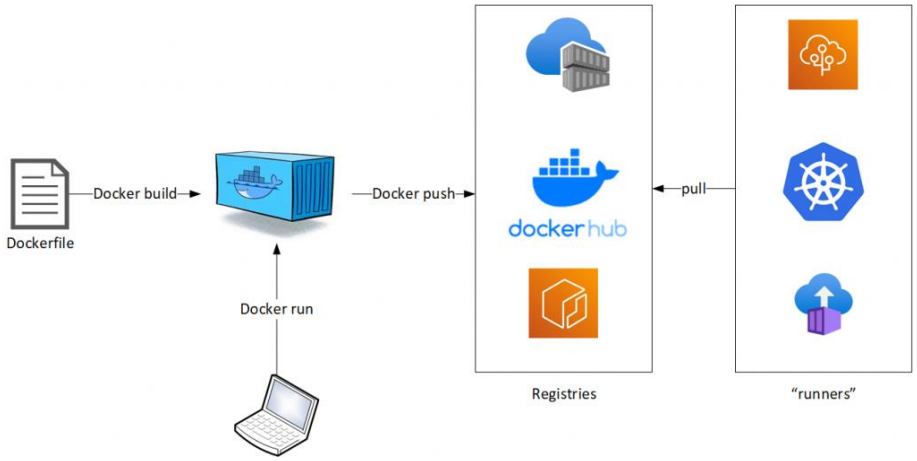


Ilustración 11. Elementos necesarios para la creación y manipulación de los recursos de un contenedor. Obtenido de: https://consultorinternet.com/tag/docker

5.8.CARACTERÍSTICAS DE DOCKER

Así mismo se describen las características proporcionadas por Docker, que la convierten en una herramienta óptima para la creación y gestión de contenedores.

- **Contenedores**: Docker utiliza contenedores, que son unidades de software ligero y autónomo que incluyen todo lo necesario para ejecutar una aplicación, incluyendo el código, el tiempo de ejecución, las bibliotecas del sistema, y las herramientas del sistema [13], [14].
- **Estandarización**: Permite estandarizar las operaciones, lo que facilita la implementación, la identificación de problemas y el retorno a una fase anterior para remediarlos [15].
- **Integración y entrega continuas (CI/CD)**: Facilita la integración y entrega continuas al estandarizar los entornos de desarrollo y producción, lo que ayuda a eliminar los conflictos entre paquetes de lenguaje y versiones [14], [17].
- **Portabilidad**: Los contenedores de Docker son portátiles, lo que significa que pueden ejecutarse en cualquier sistema que tenga Docker instalado, independientemente del sistema operativo o del entorno de host [15].
- **Seguridad**: Docker ofrece características de seguridad como la capacidad de aislar los contenedores y controlar el acceso a los recursos del sistema [17].
- **Ecosistema**: Docker tiene un ecosistema activo con una gran cantidad de imágenes y herramientas disponibles para facilitar el desarrollo y la implementación de aplicaciones [13], [20].
- **Compatibilidad con la nube**: Docker es compatible con muchos proveedores de servicios en la nube, lo que permite desplegar contenedores en la nube de manera eficiente [4], [12].

5.9.VENTAJAS

- **Automatización**: Docker facilita la automatización de tareas, lo que puede ayudar a los desarrolladores a evitar tareas repetitivas y ahorrar tiempo [12], [13].
- **Comunidad**: La comunidad es activa con muchos recursos disponibles, incluyendo un canal de Slack dedicado, un foro de la comunidad y una gran cantidad de imágenes de contenedores en Docker Hub [20].
- **Ahorro de recursos**: Al ejecutar múltiples contenedores en un solo host, Docker permite aprovechar mejor los recursos del servidor, lo que puede resultar en ahorros significativos [17], [19].

5.10. DESVENTAJAS

- **Seguridad**: Aunque Docker ofrece características de seguridad, las máquinas virtuales pueden ser consideradas más seguras porque el sistema operativo de seguridad se mantiene independiente del hardware [16].
- **Complejidad**: Docker puede ser complejo de configurar y administrar, especialmente para usuarios que no están familiarizados con la tecnología de contenedores [18].
- **Dependencia del kernel**: Depende del kernel del sistema operativo, lo que significa que las características y el rendimiento pueden variar entre diferentes sistemas operativos [14], [25].
- **Limitación de la persistencia de datos**: Aunque facilita la creación de contenedores, la persistencia de datos puede ser un desafío, ya que los datos almacenados en un contenedor pueden perderse si el contenedor se detiene o se elimina [16], [26].

6. KUBERNETES

Es un sistema de código abierto diseñado para automatizar la implementación, escalado y gestión de aplicaciones en contenedores que luego los organiza de una aplicación en unidades lógicas para simplificar su administración y localización [28]. Kubernetes se ha desarrollado a partir de 15 años de experiencia en la ejecución de cargas de trabajo en entornos de producción en Google, integrando las mejores ideas y prácticas de la comunidad [29].

Ofrece una respuesta a la orquestación de contenedores al abordar las limitaciones de Docker al gestionar aplicaciones con múltiples contenedores y servicios, especialmente aquellas con un alto tráfico entrante [30]. Facilita tareas como la asignación de ubicación para los contenedores, el monitoreo, la reiniciación en caso de problemas, la escalabilidad automática de 1 a 20 réplicas según la carga, y diversas acciones adicionales. Además, proporciona una abstracción que permite implementar aplicaciones en un clúster sin preocuparse por las máquinas subyacentes [28],[29].

6.1.HISTORIA Y EVOLUCIÓN DE KUBERNETES

Kubernetes fue lanzado al público en 2015 bajo la Fundación de Software de Código Abierto de Linux (CNCF). Este proyecto de código abierto destinado a la orquestación de contenedores ha experimentado una adopción significativa en la industria desde entonces [47]. Su evolución ha estado marcada por continuas actualizaciones y mejoras, cada versión introduciendo nuevas características y correcciones de errores para adaptarse a las cambiantes necesidades de la comunidad y la industria [48].

La comunidad de Kubernetes ha crecido de manera exponencial, con la participación activa de empresas como Microsoft, Red Hat e IBM, contribuyendo al desarrollo y consolidación del proyecto. Además, ha surgido un ecosistema vibrante de herramientas y proyectos complementarios que amplían y enriquecen las funcionalidades de Kubernetes [47],[48].

Este sistema ha demostrado ser versátil, abordando una amplia gama de casos de uso, desde implementaciones en la nube hasta entornos locales y híbridos. Su capacidad para escalar eficientemente, gestionar recursos y automatizar procesos le ha conferido un estatus de estándar de facto en la administración de contenedores [49].

6.2.ARQUITECTURA DE KUBERNETES

Kubernetes presenta una arquitectura que proporciona un método descentralizado para la identificación de servicios dentro de un conjunto. En un clúster de Kubernetes, se encuentran presentes uno o varios planos de control, así como uno o varios nodos de computación [34].

Existen dos categorías de nodos dentro de un clúster de Kubernetes, cada uno desempeñando un conjunto definido de procesos específicos [35]:

- Nodo principal: También conocido como nodo maestro o maestro, actúa como la cabeza y el cerebro del sistema, realizando todas las funciones de pensamiento y toma de decisiones. Aquí reside la inteligencia central del clúster.
- Nodo de trabajo: También denominado nodo o Minion, representa las manos y los pies que ejecutan las tareas prácticas y operativas.

El maestro controla los nodos y, en la mayoría de los casos, la comunicación se realiza solo con el maestro.

Una interfaz muy utilizada para interactuar con el clúster es la herramienta de línea de comandos kubectl. Esta se instala como una aplicación cliente, ya sea en el propio nodo maestro o en un equipo independiente, como una PC. No importa desde dónde se realice, la comunicación con el maestro se lleva a cabo a través de la API de REST que éste expone [36].

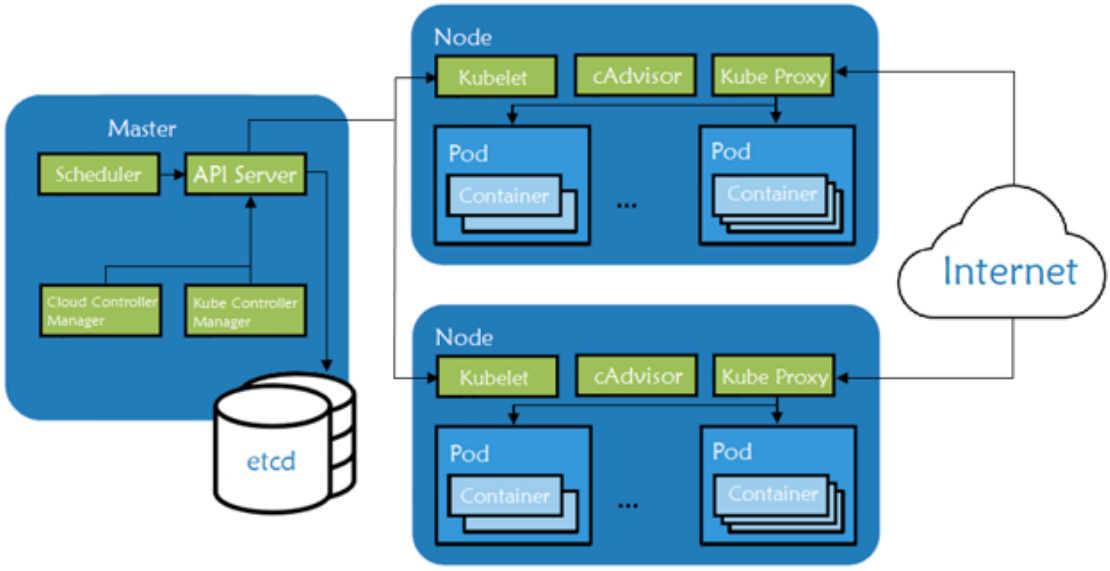


Ilustración 12 Arquitectura de Kubernetes

6.3.COMPONENTES DE KUBERNETES

A continuación, se describirán los distintos componentes que son necesarios para operar un clúster de Kubernetes.

6.3.1. **COMPONENTES DEL PLANO DE CONTROL**

Los elementos que integran el plano de control son responsables de tomar decisiones a nivel global en el clúster, como la planificación, así como de detectar y reaccionar ante eventos específicos, como la generación de un nuevo pod cuando la propiedad de réplicas de un controlador de replicación no se cumple. Estos componentes tienen la capacidad de operar en cualquier nodo del clúster [39], [50]. Sin embargo, con fines de simplificación, los scripts de instalación suelen activarse exclusivamente en el mismo nodo, evitando así la ejecución de contenedores de usuarios en esos nodos, la ejecución del panel de control se distribuye en varios nodos para garantizar una alta disponibilidad del sistema [50].

6.3.2. **KUBE-APISERVER**

El componente del plano de control de Kubernetes encargado de exponer la API de Kubernetes es el servidor de la API. Este actúa como la interfaz principal de Kubernetes, recibiendo solicitudes y actualizando consecuentemente el estado en etcd [46].

La implementación principal de este servidor en Kubernetes es conocida como kube-apiserver. Esta implementación está diseñada para operar con alta disponibilidad y cuenta con la capacidad de escalar horizontalmente, distribuyendo la carga de manera equitativa entre múltiples instancias [46], [47].

6.3.3. **ETCD**

Es un sistema de almacenamiento persistente, coherente y distribuido basado en clave-valor es empleado como el depósito de información integral del clúster de Kubernetes. Si nuestro clúster está utilizando etcd como su sistema de almacenamiento, se recomienda revisar la documentación referente a estrategias de respaldo [51].

6.3.4. **KUBE-SCHEDULER**

Este componente del plano de control se encarga de supervisar los Pods que aún no han sido asignados a ningún nodo y elige uno adecuado para su ejecución.

Al determinar en qué nodo se desplegará el pod, se consideran diversos factores como requisitos de recursos, restricciones de hardware/software/políticas, afinidad y anti-afinidad, así como la ubicación de datos dependientes, entre otros.[52]

6.3.5. **KUBE- CONTROLLER-MANAGER**

Un elemento del plano de control que ejecuta los controladores de Kubernetes. Aunque lógicamente cada controlador opera como un proceso independiente, para simplificar la

estructura, se compilan en un único binario que se ejecuta como un solo proceso [51], [52]. Entre estos controladores se encuentran:

- Controlador de nodos: encargado de detectar y responder cuando un nodo deja de funcionar.
- Controlador de replicación: responsable de mantener la cantidad adecuada de pods para cada controlador de replicación en el sistema.
- Controlador de endpoints: crea el objeto Endpoints, es decir, establece una conexión entre los Services y los Pods.
- Controladores de tokens y cuentas de servicio: generan cuentas y tokens de acceso a la API de manera predeterminada para los nuevos Namespaces.

6.3.6. **CLOUD-CONTROLLER-MANAGER**

El cloud-controller-manager ejecuta controladores que mantienen interacciones con proveedores de servicios en la nube. La funcionalidad del binario cloud-controller-manager se introdujo inicialmente como una característica alpha en la versión 1.6 de Kubernetes.[53]

Este componente específico, cloud-controller-manager, realiza ciclos de control destinados exclusivamente para cada proveedor de servicios en la nube. Existe la posibilidad de desactivar estos ciclos en el kube-controller-manager al incluir la opción --cloud-provider=external durante el inicio del kube-controller-manager.[51],[53]

La presencia del cloud-controller-manager facilita la evolución independiente del código de Kubernetes y el código del proveedor de la nube. Anteriormente, el código de Kubernetes estaba vinculado a funcionalidades específicas de cada proveedor de servicios en la nube. En un enfoque futuro, se espera que el código específico de una plataforma sea mantenido por el proveedor de servicios en la nube y vinculado al cloud-controller-manager durante la ejecución de Kubernetes.[50]

Los siguientes controladores dependen, de alguna manera, del proveedor de servicios en la nube:[53],[50],[52]

- Controlador de nodos: responsable de identificar y responder cuando un nodo deja de estar operativo.
- Controlador de rutas: se encarga de configurar rutas en la infraestructura subyacente de la nube.
- Controlador de servicios: encargado de crear, actualizar y eliminar balanceadores de carga en la nube.

- Controlador de volúmenes: se ocupa de la creación, conexión y montaje de volúmenes, interactuando con el proveedor de la nube para orquestar estas operaciones.

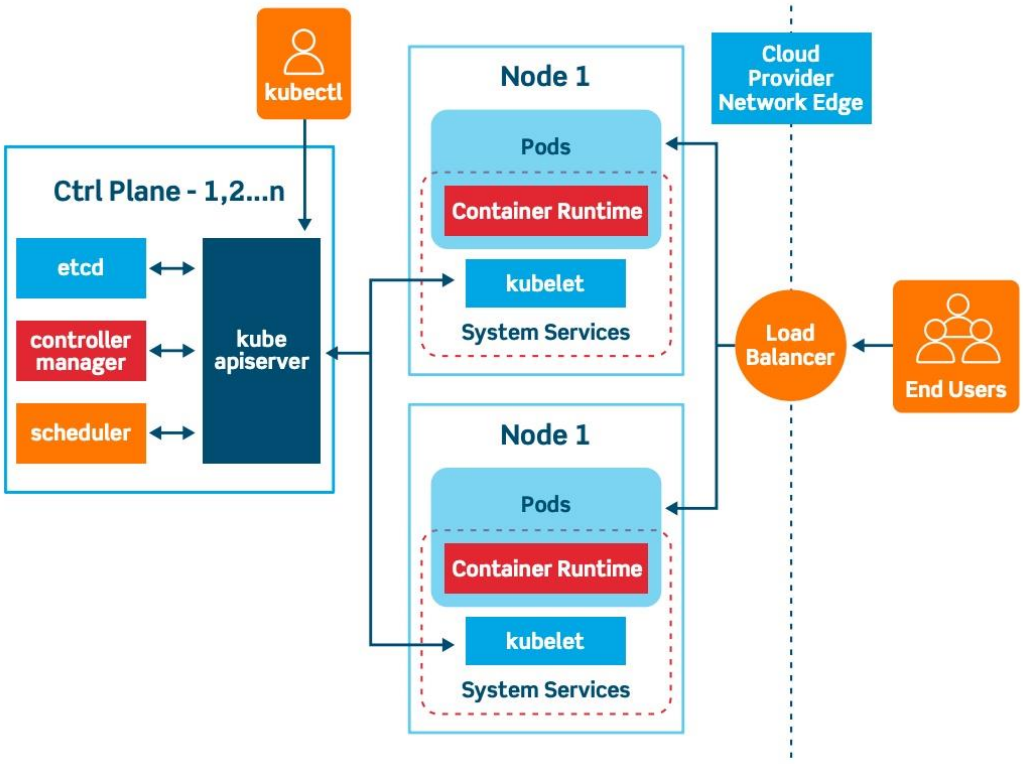


Ilustración 13 Herramientas que componen la arquitectura de Kubernetes

6.4. MÁSTER KUBERNETES

Es un nodo principal en Kubernetes, también denominado patrón, actúa como el cerebro central, este nodo maestro del clúster despliega el plano de control, encargado de tomar decisiones a nivel global para el conjunto [37]. Por ejemplo, cuando surge la necesidad de generar un contenedor en el clúster, el maestro determina qué nodo ejecutará la tarea y procede a iniciar un nuevo contenedor, en un proceso conocido como programación [36],[37].

La responsabilidad principal del maestro recae en mantener el estado deseado para el clúster. Al asignar una solicitud para un servidor web, se asegura de que siempre existan dos contenedores entre sí. El maestro monitorea el estado de ejecución y genera un nuevo contenedor si hay menos de dos debido a errores [38].

Normalmente, se requiere solo un nodo maestro en el clúster, aunque existe la opción de replicarlo para lograr una mayor disponibilidad y redundancia. Diversos procesos que se ejecutan en el nodo maestro implementan funciones específicas, tales como [37], [38].

- kube-apiserver: Actúa como la interfaz frontal del plano de control y proporciona una API de REST.
- kube-scheduler: Encargado de la programación, decide la ubicación de los contenedores según los requisitos del sistema (CPU, memoria, almacenamiento, etc.) y otros parámetros o restricciones personalizados, como especificaciones de afinidad.
- kube-controller-manager: Único proceso que supervisa diversos tipos de controladores, garantizando que el estado del sistema sea el esperado. Ejemplos de controladores incluyen la controladora de duplicación, ReplicaSet, Implementación y Controlador de servicio.
- etcd: Base de datos que almacena el estado del sistema.

6.5.¿CÓMO FUNCIONA KUBERNETES?

Kubernetes opera gestionando el estado deseado y real del clúster. Los objetos representan el estado actual, mientras que los controladores aseguran la coherencia entre el estado presente y el deseado. Desarrolladores definen el estado a través de archivos YAML o JSON, enviándolos a la API de Kubernetes [39]. El estado deseado incluye configuraciones como aplicaciones, imágenes en contenedores y recursos.

La base de datos etcd almacena estos datos. Kubernetes automatiza la gestión, ajustando el clúster mediante controladores integrados. Los conjuntos de réplicas garantizan un número específico de pods, siendo gestionados por implementaciones de Kubernetes [40]. La escalabilidad automática permite ajustar servicios según la demanda, como aumentar réplicas durante picos de actividad. En resumen, Kubernetes automatiza y gestiona la coherencia entre el estado actual y el deseado del clúster, ofreciendo flexibilidad y escalabilidad [41].

Kubernetes requiere de los contenedores de Docker para ser agrupados en los clústers y las propiedades o características que son consideradas son las mismas de Docker. Esto se puede visualizar en la siguiente imagen.

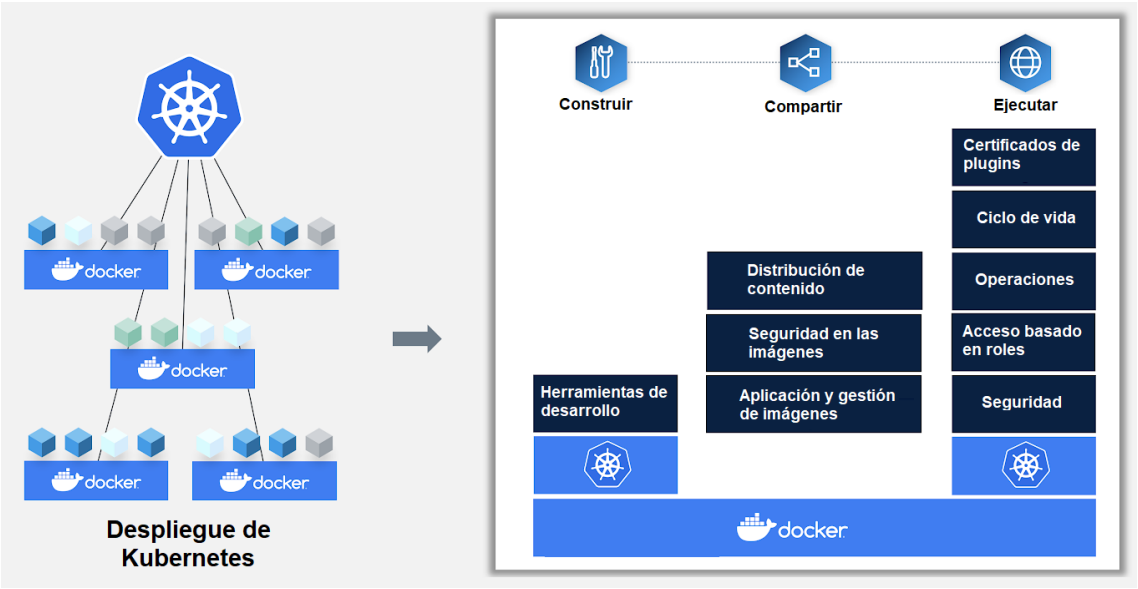


Ilustración 14. Representación de la comunicación y orquestación de múltiples contenedores de Docker (Traducida) [14].

6.6.USOS DE KUBERNETES

Kubernetes se emplea para desarrollar aplicaciones que son sencillas de gestionar y desplegar en cualquier entorno. Cuando se presenta como un servicio administrado, Kubernetes proporciona diversas soluciones adaptadas a tus requerimientos [42], [43].

- **Aumento de la velocidad de desarrollo:** Kubernetes facilita la construcción de aplicaciones basadas en microservicios y diseñadas para la nube. Además, respalda la containerización de aplicaciones preexistentes, convirtiéndose en el fundamento para la modernización de aplicaciones y acelerando el proceso de desarrollo de aplicaciones.
- **Implementación de aplicaciones en cualquier lugar:** Kubernetes ha sido concebido para ser utilizado en diversos entornos, permitiéndote ejecutar aplicaciones tanto en implementaciones locales como en nubes públicas o entornos híbridos. Esto posibilita que puedas ejecutar tus aplicaciones en el lugar que mejor se adapte a tus necesidades.
- **Ejecución de servicios eficientes:** Kubernetes tiene la capacidad de ajustar automáticamente el tamaño de un clúster para satisfacer los requisitos de ejecución de un servicio específico. Esto posibilita la escalabilidad automática de tus aplicaciones según la demanda, permitiendo su ejecución eficiente de manera automática.

6.7.CARACTERÍSTICAS DE KUBERNETES

1. **Orquestación de Contenedores:** Kubernetes facilita la gestión automatizada de contenedores, permitiendo escalar, desplegar y gestionar aplicaciones de forma eficiente [32].

2. **Escalabilidad:** Permite escalar horizontalmente aplicaciones de manera sencilla, adaptándose a las demandas de tráfico [31].
3. **Autoreparación:** Kubernetes ofrece capacidades de autoreparación, detectando y reemplazando contenedores que no responden [31], [32].
4. **Despliegue Declarativo:** Permite definir el estado deseado de la aplicación y Kubernetes se encarga de llevarla a ese estado, facilitando la gestión [32].
5. **Servicio de Descubrimiento:** Proporciona un servicio de descubrimiento para que los contenedores puedan encontrarse y comunicarse entre sí [32].

6.8.VENTAJAS

1. **Escalabilidad:** Permite escalar aplicaciones de forma dinámica [31].
2. **Autoreparación:** Capacidad de autoreparación mejora la disponibilidad de las aplicaciones [32].
3. **Orquestación Avanzada:** Ofrece una orquestación avanzada de contenedores, facilitando la gestión de aplicaciones complejas [33].
4. **Portabilidad:** Permite la portabilidad de aplicaciones entre entornos de desarrollo, pruebas y producción [31], [32].
5. **Comunidad Activa:** Cuenta con una amplia comunidad de usuarios y desarrolladores que contribuyen a su mejora continua [33].

6.9.DESVENTAJAS

1. **Complejidad:** Puede resultar complejo de configurar y gestionar para usuarios no familiarizados con la tecnología [31], [33].
2. **Requisitos de Recursos:** Requiere recursos significativos en términos de CPU y memoria para su despliegue [31].
3. **Curva de Aprendizaje:** La curva de aprendizaje puede ser empinada para aquellos nuevos en el uso de Kubernetes [33].
4. **Costo:** Puede implicar costos adicionales en términos de infraestructura y recursos para su implementación y mantenimiento [31], [32].
5. **Posible Sobredimensionamiento:** Si no se configura adecuadamente, puede llevar a un sobredimensionamiento de recursos [32].

7. DIFERENCIAS ENTRE DOCKER Y KUBERNETES

1. **Docker:** Es una plataforma de contenedores que permite empaquetar, distribuir y ejecutar aplicaciones en entornos aislados [31].
2. **Kubernetes:** Es un sistema de orquestación de contenedores que facilita la gestión, escalabilidad y despliegue de aplicaciones en contenedores [32].
3. **Funcionalidad:** Docker se centra en la creación y ejecución de contenedores individuales, mientras que Kubernetes se enfoca en la gestión de múltiples contenedores y su orquestación [31], [32].
4. **Escalabilidad:** Kubernetes ofrece capacidades avanzadas de escalabilidad y autoreparación, mientras que Docker se centra en la creación y ejecución de contenedores [32], [33].
5. **Complejidad:** Kubernetes es más complejo que Docker, ya que proporciona una capa adicional de abstracción para la gestión de múltiples contenedores y su interacción [31].

8. CASOS DE USO E IMPLEMENTACIÓN DE DOCKER Y KUBERNETES

Docker Containers and Images for Robot Operating System (ROS)–Based Applications

Presentaron una investigación sobre el uso de contenedores Docker para implementar y ejecutar una carga de pago (payload) basada en ROS en plataformas robóticas. Su enfoque aprovechó Docker para empaquetar el código, bibliotecas y dependencias en una imagen portable, facilitando el despliegue en múltiples plataformas.

Desarrollaron un Dockerfile integral para crear dicha imagen reusable, con controladores y repositorios necesarios para un payload de sensores específico en robots móviles. Su solución redujo drásticamente el tiempo de configuración e instalación en cada robot, de días a 30 minutos.

El uso de Docker mejoró así la portabilidad, escalabilidad y facilidad de implementación del software robótico entre plataformas heterogéneas; también habilitó el control de versiones y mantenimiento. Si bien su estudio se centra en la implementación eficiente mediante contenedores, no explora aspectos más profundos de ejecución en contenedores como rendimiento computacional, networking o seguridad. Ofrece entonces una solución práctica para agilizar despliegues robóticos, sobre la cual se podría avanzar investigando otras dimensiones de este paradigma [44].

Studying the Practices of Deploying Machine Learning Projects on Docker

Se llevó a cabo un estudio para comprender cómo se utiliza Docker en el proceso de despliegue de proyectos de software basados en aprendizaje automático (ML). Analizaron 406 proyectos open-source de GitHub que utilizan Docker y tienen imágenes Docker correspondientes en Docker Hub.Categorizaron los proyectos en: sistemas de aplicación ML, MLOps/AIOps, toolkits, frameworks de deep learning, modelos y documentación.

El análisis concluyó con la identificación de 21 propósitos principales de uso de Docker, como gestión de modelos, testing de software, desarrollo interactivo, gestión de datos, entre otros. Encontraron que la instrucción Docker más utilizada es ***RUN***, principalmente para gestionar sistemas de archivos, dependencias, permisos, compilación/ejecución y variables de entorno. Las imágenes base más comunes están relacionadas con sistemas operativos, runtimes de plataformas/lenguajes y frameworks de deep learning. Sin embargo, se requieren más recursos para ejecutar estas imágenes Docker debido a la gran cantidad de archivos anidados.

Un pequeño conjunto de archivos ocupa relativamente mucho espacio, lo cual requiere enfoques más eficientes de almacenamiento. El estudio categorizó los proyectos de ML que usan Docker, identificó varios propósitos de uso, analizó las funcionalidades más utilizadas en los Dockerfiles y caracterizó las imágenes Docker resultantes [45].

A case analysis for Kubernetes network security of financial service industry in Indonesia using zero trust model

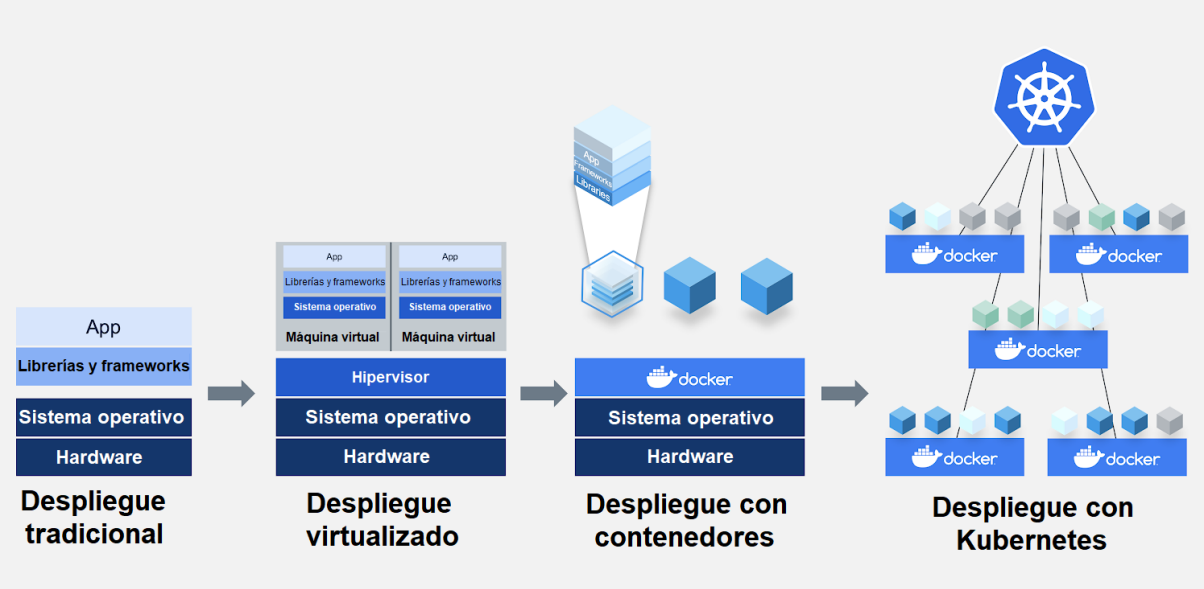
Presentaron un análisis de caso sobre la implementación de Kubernetes en una empresa de servicios financieros en Indonesia, para mejorar sus aplicaciones basadas en servicios digitales desarrolladas con arquitectura de microservicios. Construyeron y evaluaron un diseño de infraestructura para una red Kubernetes segura, utilizando el centro de datos de la empresa.

Se enfocaron en dos aspectos claves: redes y seguridad. Emplearon recomendaciones de red de VMware, Cisco y el modelo de confianza cero de Forrester como guía de seguridad. El diseño propuesto de infraestructura de red segura se aplicó exitosamente en las redes de contenedores, cumpliendo con los requerimientos de confianza cero y las necesidades y restricciones de la empresa.

Esperan que este estudio sobre el diseño de seguridad de red pueda ser utilizado y adaptable a redes existentes, reduciendo el riesgo y la interrupción al negocio causadas por la red. Si bien se centra en la implementación segura mediante Kubernetes, no explora otros aspectos relevantes como rendimiento, escalabilidad o monitorización. Ofrece una solución práctica

para asegurar despliegues en Kubernetes, que se podría complementar investigando esas otras dimensiones [46].

9. COMPARACIÓN ENTRE DESPLIEGUES DE APLICACIONES



10. CONCLUSIÓN

Docker y Kubernetes son herramientas esenciales en la computación actual, especialmente en la nube y las arquitecturas de microservicios. Docker se centra en la contenerización, permitiendo un empaquetado sencillo y eficiente de aplicaciones y sus dependencias, asegurando un entorno de ejecución uniforme en distintos sistemas. Esto facilita el desarrollo, distribución y ejecución de aplicaciones contenerizadas.

Kubernetes, por otro lado, es una plataforma de orquestación de contenedores que supera las limitaciones de Docker, proporcionando una solución para la gestión de contenedores a gran escala. Kubernetes maneja la escalabilidad, balanceo de carga, tolerancia a fallos y configuración de redes en un clúster de contenedores, lo que lo convierte en una opción ideal para desplegar y gestionar aplicaciones en entornos distribuidos.

Por lo tanto, Docker y Kubernetes son herramientas fundamentales en el desarrollo y despliegue de aplicaciones modernas. Docker ofrece contenerización y gestión de imágenes, mientras que Kubernetes se encarga de la orquestación a gran escala. Juntos, ofrecen una solución completa para el empaquetado, distribución y gestión de aplicaciones, permitiendo a los equipos de desarrollo trabajar de manera más eficiente y escalable.

11. REFERENCIAS

[1] F. de Asís López Fuentes, *Sistemas distribuidos*. 2015. [Online]. Available: www.cua.uam.mx

[2] I. Aviv, A. Barger, A. Kofman, and R. Weisfeld, "Reference Architecture for Blockchain-Native Distributed Information System," *IEEE Access*, vol. 11, pp. 4838–4851, 2023, doi: 10.1109/ACCESS.2023.3235838.

[3] M. Fazio *et al.*, "A Note on the Convergence of IoT, Edge, and Cloud Computing in Smart Cities," 2018. [Online]. Available: www.computer.org/cloud

[4] M. De Donno, K. Tange, and N. Dragoni, "Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog," *IEEE Access*, vol. 7, pp. 150936–150948, 2019, doi: 10.1109/ACCESS.2019.2947652.

[5] E. R. Morales, C. Fernando, and X. A. Capelo, "Computación en la nube Con Google Drive ESPOCH 2016," 2016.

[6] Dr. Rafael Belloso Chacín, "LA NUEVA ERA DE LOS NEGOCIOS: COMPUTACIÓN EN LA NUBE," Nov. 2015.

[7] F. Nadeem, "A Unified Framework for User-Preferred Multi-Level Ranking of Cloud Computing Services Based on Usability and Quality of Service Evaluation," *IEEE Access*, vol. 8, pp. 180054–180066, 2020, doi: 10.1109/ACCESS.2020.3027775.

[8] P. Bo, S. Fenzhen, and M. Yunshan, "A Cloud and Cloud Shadow Detection Method Based on Fuzzy c-Means Algorithm," *IEEE J Sel Top Appl Earth Obs Remote Sens*, vol. 13, pp. 1714–1727, 2020, doi: 10.1109/JSTARS.2020.2987844.

[9] M. Yahuza *et al.*, "Systematic review on security and privacy requirements in edge computing: State of the art and future research opportunities," *IEEE Access*, vol. 8, pp. 76541–76567, 2020, doi: 10.1109/ACCESS.2020.2989456.

[10] A. Varol, Institute of Electrical and Electronics Engineers. Portugal Section., and Institute of Electrical and Electronics Engineers, *7th International Symposium on Digital Forensics and Security : 10-12 June 2019, Barcelos, Portugal*.

[11] R. Wang, X. Dong, Q. Li, and Z. Ren, "Distributed Adaptive Formation Control for Linear Swarm Systems with Time-Varying Formation and Switching Topologies," *IEEE Access*, vol. 4, pp. 8995–9004, 2016, doi: 10.1109/ACCESS.2016.2646399.

[12] A. Mouat, Using Docker: Developing and Deploying Software with Containers. 2016.

[13] I. Álvaro and I. Martínez, Docker para DevOps de noob a experto. 2021.

[14] Jairo José Pérez Calvo, “Plataforma web para la gestión de contenedores Docker,” 2021.

[15] J. Turnbull, “The Docker Book,” Aging (Albany. NY)., vol. 7, no. 11, pp. 956–963, 2015.

[16] Nitin, Naik. (2017). Docker container-based big data processing system in multiple clouds for everyone. 1-7. Available from: 10.1109/SYSENG.2017.8088294

[17] Moses, Openja., Forough, Majidi., Foutse, Khomh., Bhagya, Chembakottu., Heng, Li. (2022). Studying the Practices of Deploying Machine Learning Projects on Docker. Available from: 10.1145/3530019.3530039

[18] Osama, Ennasr., A., Soylemezoglu., Garry, Glaspell. (2023). Docker containers and images for Robot Operating System (ROS)--based applications. Available from: 10.21079/11681/47279

[19] Giovanni, Rosa., Simone, Scalabrino., Gabriele, Bavota., Rocco, Oliveto. (2023). What Quality Aspects Influence the Adoption of Docker Images?. ACM Transactions on Software Engineering and Methodology, Available from: 10.1145/3603111

[20] Hyunjin, Shim. (2023). CRISPR-Cas-Docker: Web-based in silico docking and machine learning-based classification of crRNAs with Cas proteins. doi: 10.1101/2023.01.04.522819

[21] Haneul, Lee., Soonhong, Kwon., Jong-Hyouk, Lee. (2023). Experimental Analysis of Security Attacks for Docker Container Communications. Electronics, 12(4), 940-940. Available from: 10.3390/electronics12040940

[22] ChangHyuk, Kwon., Jason, Kim., Jaegyoon, Ahn. (2018). DockerBIO: web application for efficient use of bioinformatics Docker images.. PeerJ, 6 Available from: 10.7717/PEERJ.5954

[23] Moses, Openja., Forough, Majidi., Foutse, Khomh., Bhagya, Chembakottu., Heng, Li. (2022). Studying the Practices of Deploying Machine Learning Projects on Docker. Available from: 10.1145/3530019.3530039

[24] Dhanush P, et al., (2023). Campus cloud: empowering university management web application with cloud-hosted docker technology on aws. Indian Scientific Journal Of Research In Engineering And Management, 07(04) Available from: 10.55041/ijsrem20366

[25] Hanji, Ju., Jie, Wang., Enguo, Zhu., Xiaoli, zhang., Feng, Zheng. (2022). Design Scheme of a Docker Container File Isolation against Computer Virus Spreading. Mathematical Problems in Engineering, 2022, 1-6. Available from: 10.1155/2022/5348370

[26] Ahmet, Efe., Ulaş, Aslan., Aytekin, Mutlu, Kara. (2020). Securing Vulnerabilities in Docker Images. 4(1), 31-39. Available from: 10.46460/IJIEA.617181

[27] Ivan, A., Burenkov. (2022). Using Docker virtual containers to launch services. Lit′ë i Metallurgiâ, 108-111. Available from: 10.21122/1683-6065-2022-3-108-111

KUBERNETES

[28] Stoneman, E., & Safari, an O. M. Company. (n.d.). *Learn Kubernetes in a Month of Lunches*. 592.

[29] Torres, M., Antonio, G., & García, L. (2019). *UNIVERSIDAD DE ALMERIA ESCUELA SUPERIOR DE INGENIERÍA TRABAJO FIN DE GRADO Despliegue de aplicaciones escalables con Kubernetes*.

[30] Strastocole. (n.d.). *EVERYTHING KUBERNETES: A PRACTICAL GUIDE CONTENTS INTRODUCTION KUBERNETES-BIRD'S EYE VIEW HIGH LEVEL ARCHITECTURE KUBERNETES BUILDING BLOCKS THE BASICS BLOCKS USING LABELS AND SELECTORS FOR FINE-GRAINED CONTROL SERVICE DISCOVERY 3 STORAGE BUILDING BLOCKS CHOOSING THE RIGHT BLOCK FOR THE JOB IMPERATIVE VS. DECLARATIVE ORCHESTRATION HANDS-ON: GETTING STARTED INSTALLATION LOGGING MONITORING WORKING WITH MULTIPLE CLUSTERS HANDS-ON: DEPLOYING AN APPLICATION*.

[31] Kaur, K., Guillemin, F., & Sailhan, F. (n.d.). Container placement and migration strategies for Cloud, Fog and Edge data centers: A survey. *International Journal of Network Management*, *2022*(6), 10. https://doi.org/10.1002/nem.2212ï

[32] Sarah R Nadaf1, H. K. K. (2022). *View of Kubernetes in Microservices*. https://doi.org/10.47679/ijasca.v2i1.19

[33] Turin, G., Borgarelli, A., Donetti, S., Damiani, F., Johnsen, B., & Lizeth Tapia Tarifa, S. (n.d.). *Predicting Resource Consumption of Kubernetes Container Systems using Resource Models ⋆*. https://github.com/GoogleCloudPlatform/microservices-demo

[34] Lee, J. B., Yoo, T. H., Lee, E. H., Hwang, B. H., Ahn, S. W., & Cho, C. H. (2021). High-Performance Software Load Balancer for Cloud-Native Architecture. *IEEE Access*, *9*, 123704–123716. https://doi.org/10.1109/ACCESS.2021.3108801

[35] Li, Z., Wei, H., Lyu, Z., & Lian, C. (2021). Kubernetes-Container-Cluster-Based Architecture for an Energy Management System. *IEEE Access*, *9*, 84596–84604. https://doi.org/10.1109/ACCESS.2021.3081559

[36] Rostami, G. (2023). Role-based Access Control (RBAC) Authorization in Kubernetes. *Journal of ICT Standardization*, *11*(3), 237–260. https://doi.org/10.13052/jicts2245-800X.1132

[37] García, J. N., & Bernardos, C. J. (2018). *Orquestación de contenedores con Kubernetes*.

[38] Gao, R., Xie, X., & Guo, Q. (2023). K-TAHP: A Kubernetes Load Balancing Strategy Base on TOPSIS+AHP. *IEEE Access*, *11*, 102132–102139. https://doi.org/10.1109/ACCESS.2023.3313643

[39] Fe, I., Nguyen, T. A., Soares, A., Son, S., Choi, E., Min, D., Lee, J. W., & Silva, F. A. (2023). Model-driven Dependability and Power Consumption Quantification of Kubernetes based Cloud-Fog Continuum. *IEEE Access*. https://doi.org/10.1109/ACCESS.2023.3340195

[40] Han, J., Hong, Y., & Kim, J. (2020). Refining microservices placement employing workload profiling over multiple kubernetes clusters. *IEEE Access*, *8*, 192543–192556. https://doi.org/10.1109/ACCESS.2020.3033019

[41] Femminella, M., Palmucci, M., Reali, G., & Rengo, M. (2024). *XXXX; Date of publication XX Month, XXXX; date of current version XX Month, XXXX. Attribute-Based Management of Secure Kubernetes Cloud Bursting*. https://doi.org/10.1109/OJCOMS.2022.1234567

[42] Rostami, G. (2023). Role-based Access Control (RBAC) Authorization in Kubernetes. *Journal of ICT Standardization*, *11*(3), 237–260. https://doi.org/10.13052/jicts2245-800X.1132

[43] Tran, M. N., Vu, X. T., & Kim, Y. (2022). Proactive Stateful Fault-Tolerant System for Kubernetes Containerized Services. *IEEE Access*, *10*, 102181–102194. https://doi.org/10.1109/ACCESS.2022.3209257

[44] Amir Naser, O. E. A. S. and G. G. (2023). *Docker Containers and Images for*.

[45] Openja, M., Majidi, F., Khomh, F., Chembakottu, B., & Li, H. (2022). Studying the Practices of Deploying Machine Learning Projects on Docker. *ACM International Conference Proceeding Series*, 190–200. https://doi.org/10.1145/3530019.3530039

[46] Nico Surantha1, 2, Felix Ivan1, Ritchie Chandra. (2022). *A case analysis for Kubernetes network security of financial service industry in Indonesia using zero trust model _ Surantha _ Bulletin of Electrical Engineering and Informatics*.

[47] National Security Agency (NSA) Cybersecurity Directorate Endpoint Security. (2022). *Kubernetes Hardening Guide Notices and history Document change history Disclaimer of warranties and endorsement Trademark recognition*.

[48] Song, P., Aborabh, A., & Mariappan, Y. (n.d.). *Day One: Building Containers Using Kubernetes and Contrail*. www.juniper.net/books.

[49] Truyen, E., Kratzke, N., van Landuyt, D., Lagaisse, B., & Joosen, W. (2020). Managing Feature Compatibility in Kubernetes: Vendor Comparison and Analysis. *IEEE Access*. https://doi.org/10.1109/ACCESS.2020.3045768

[50] Scano, D., Giorgetti, A., Paolucci, F., Sgambelluri, A., Chammanara, J., Rothman, J., Al-Bado, M., Marx, E., Ahearne, S., & Cugini, F. (2023). Enabling P4 Network Telemetry in Edge Micro Data Centers With Kubernetes Orchestration. *IEEE Access*, *11*, 22637–22653. https://doi.org/10.1109/ACCESS.2023.3249105

[51] Kashyap, A., & Lu, X. (2022). NVMe-oAF: Towards Adaptive NVMe-oF for IO-Intensive Workloads on HPC Cloud. *HPDC 2022 - Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, 56–70. https://doi.org/10.1145/3502181.3531476

[52] Schien, D., & James, A. (2019). *A Low Carbon Kubernetes Scheduler*. https://www.researchgate.net/publication/364315086

[53] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9751692 [53]