Informe sobre Docker, Contenedores, Docker Compose, Docker Swarm y Docker Stack

1. Introducción a Docker y la Contenerización:

Docker se presenta como una plataforma de software que facilita la construcción, prueba y despliegue rápido de aplicaciones [1]. Su funcionamiento se basa en empaquetar el software en unidades estandarizadas denominadas contenedores, que incluyen todos los elementos necesarios para su ejecución, como bibliotecas, herramientas del sistema, código y el entorno de ejecución [1]. Esta tecnología de contenedores permite automatizar el despliegue y la gestión de aplicaciones, ofreciendo una manera consistente y eficiente de empaquetar, distribuir y ejecutar software [3]. Los desarrolladores pueden, gracias a Docker, construir, compartir, ejecutar y verificar aplicaciones en cualquier lugar, sin la necesidad de lidiar con tediosas configuraciones o gestiones de entornos [5].

La idea central de Docker es abstraer la infraestructura subyacente, proporcionando un entorno consistente para las aplicaciones, lo que simplifica significativamente el proceso de desarrollo y despliegue. Este enfoque resuelve el conocido problema de "funciona en mi máquina", donde las aplicaciones pueden comportarse de manera diferente en distintos entornos debido a inconsistencias en la configuración [4]. La amplia adopción de Docker ha generado un ecosistema robusto de herramientas y aplicaciones listas para usar [2]. Esto implica que los desarrolladores pueden aprovechar imágenes y herramientas preexistentes, en lugar de tener que construir todo desde cero. De hecho, Docker se ha consolidado como la herramienta de contenerización más utilizada en el mercado, con una cuota significativa [8].

Docker ofrece una forma estándar de ejecutar código, actuando como un sistema operativo para contenedores [2]. De manera similar a cómo una máquina virtual virtualiza el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor. Docker se instala en cada servidor y proporciona comandos sencillos para construir, iniciar o detener contenedores. Esta capacidad permite gestionar la infraestructura de la misma manera en que se gestionan las aplicaciones [9]. La analogía de un "sistema operativo para contenedores" [2] y la posibilidad de administrar la infraestructura como si fueran aplicaciones [9] son representaciones poderosas que ilustran el nivel de abstracción y control que Docker ofrece sobre el entorno de ejecución de las aplicaciones.

Docker aborda una serie de problemas comunes en el ciclo de vida del software. Uno de los principales es la gestión de dependencias. Los problemas derivados de

dependencias conflictivas, versiones incompatibles e instalaciones a nivel de sistema se resuelven mediante el aislamiento de dependencias dentro de los contenedores [4]. Esto también garantiza la compatibilidad de los servicios con el sistema operativo subyacente y sus bibliotecas [11]. Al proporcionar entornos idénticos para desarrolladores y testers, Docker elimina el problema de "funciona en mi máquina" [4]. Además, ofrece un empaquetado, configuración, monitorización, escalabilidad y manejo de errores estandarizados [12].

La utilización de Docker se traduce en despliegues más rápidos y fiables gracias a la consistencia de los entornos [13]. Simplifica la configuración y acelera los procesos de construcción [13], mejorando los flujos de trabajo de desarrollo y la experiencia del desarrollador [13]. En comparación con las máquinas virtuales, Docker maximiza la eficiencia de los recursos [7] y facilita la portabilidad de las aplicaciones entre diferentes entornos [7]. La escalabilidad de las aplicaciones también se ve simplificada [7]. Docker también resuelve problemas como configuraciones redundantes y procesos de despliegue lentos [4]. La capacidad de Docker para proporcionar un empaquetado y configuración estandarizados [12] tiene implicaciones significativas para la automatización en las canalizaciones de integración y entrega continua (CI/CD) [2]. La consistencia de los entornos asegura que las construcciones y las pruebas sean reproducibles.

2. Entendiendo los Contenedores Docker:

En el contexto de Docker, un contenedor se define como una unidad estandarizada de software que encapsula todo lo necesario para que una aplicación se ejecute. Esto incluye bibliotecas, herramientas del sistema, código y el entorno de ejecución [1]. Los contenedores son instancias de ejecución aisladas y ligeras, que comparten el kernel del sistema operativo anfitrión [3]. Se crean a partir de imágenes Docker, que actúan como plantillas [3]. Un contenedor proporciona un entorno con un aislamiento flexible que contiene todos los componentes necesarios para ejecutar una aplicación [14]. A diferencia de las máquinas virtuales que virtualizan el hardware, los contenedores virtualizan el sistema operativo del servidor [2]. Son altamente portátiles y pueden ejecutarse sin modificaciones en diversos entornos [8]. La idea de un contenedor como una "unidad estandarizada" [1] es fundamental, ya que esta estandarización permite la portabilidad y una ejecución consistente en diferentes infraestructuras. La descripción de "entorno con un aislamiento flexible" [14] subraya el equilibrio entre el aislamiento y el uso compartido de recursos que hace que los contenedores sean eficientes.

La diferencia fundamental entre los contenedores Docker y las máquinas virtuales

radica en el nivel de virtualización. Las máquinas virtuales virtualizan el hardware del servidor, incluyendo su propio sistema operativo, aplicaciones, binarios y bibliotecas, lo que las hace más grandes y con un mayor consumo de recursos [2]. En cambio, los contenedores virtualizan el sistema operativo, empaquetando únicamente la aplicación y sus dependencias [2]. Múltiples contenedores pueden ejecutarse sobre el mismo kernel del sistema operativo, compartiendo el SO pero manteniéndose aislados entre sí [2]. Los contenedores son más ligeros que las máquinas virtuales, con tamaños típicamente medidos en megabytes, mientras que las VMs pueden ocupar gigabytes [8]. Los contenedores permiten que múltiples componentes de una aplicación compartan los recursos de una única instancia del sistema operativo anfitrión [8]. Gracias a esto, se pueden ejecutar muchas más copias de una aplicación en el mismo hardware utilizando contenedores en comparación con las máquinas virtuales [8]. Los contenedores tienen tiempos de inicio más rápidos, ya que no necesitan arrancar un sistema operativo completo [8]. Docker ofrece una alternativa rentable a las VMs basadas en hipervisores [9]. Los contenedores aprovechan características del kernel de Linux como los namespaces para crear espacios de trabajo aislados [9]. En contraste, las VMs típicamente involucran un hipervisor que gestiona y aísla sistemas operativos completos [9]. La distinción fundamental reside en que las VMs virtualizan el hardware, lo que conlleva una mayor sobrecarga, mientras que los contenedores virtualizan el SO, resultando en un aislamiento de aplicaciones ligero y eficiente.

Tabla 1: Comparación de Contenedores Docker y Máquinas Virtuales

Característica	Contenedor Docker	Máquina Virtual (VM)
Nivel de Virtualización	Sistema Operativo	Hardware
Uso de Recursos	Bajo	Alto
Tamaño	Megabytes	Gigabytes
Tiempo de Inicio	Rápido (segundos)	Lento (minutos)
SO por Instancia	Comparte el SO del host	Cada VM tiene su propio SO completo

Hipervisor	No requerido	Requerido
Portabilidad	Alta, diseñado para correr en cualquier entorno	Depende de la compatibilidad del SO virtualizado

La utilización de contenedores aporta ventajas significativas tanto en el desarrollo como en la producción. Permiten entregar código más rápidamente [2] y estandarizar las operaciones de las aplicaciones [2]. El código puede moverse sin problemas entre entornos de desarrollo y producción [2]. Además, se logra un ahorro económico al mejorar la utilización de los recursos [2]. Los contenedores posibilitan la creación de entornos consistentes y reproducibles [3] y simplifican los despliegues híbridos y multi-nube [3]. Facilitan la integración y entrega continua (CI/CD) [8] y son ideales para arquitecturas de microservicios [2]. La portabilidad de los contenedores es superior [8], y permiten actualizaciones más ligeras y granulares [8]. La creación y el versionado de contenedores se pueden automatizar [8]. Los desarrolladores pueden acceder a un registro de contenedores de código abierto, como Docker Hub, que contiene miles de contenedores aportados por la comunidad [3]. La migración a la nube se simplifica [8], y los contenedores proporcionan una base sólida para los equipos de DevOps [8]. Incluso el desarrollo de inteligencia artificial y aprendizaje automático se acelera con Docker [8]. Los contenedores mejoran el aislamiento y la seguridad [7] y ofrecen una escalabilidad flexible [7]. Las ventajas de utilizar contenedores abarcan todo el ciclo de vida del software, desde la velocidad y consistencia del desarrollo hasta la eficiencia operativa y la escalabilidad en producción.

3. Docker Compose para la Gestión de Aplicaciones Multi-Contenedor:

Docker Compose es una herramienta diseñada para definir y ejecutar aplicaciones multi-contenedor [3]. Su función principal es simplificar la gestión de aplicaciones complejas que requieren múltiples contenedores interconectados. Compose permite controlar toda la pila de la aplicación (servicios, redes y volúmenes) mediante un único archivo de configuración YAML [19]. Con un solo comando, docker compose up, se crean e inician todos los servicios definidos en este archivo [19]. Compose es una herramienta versátil que se puede utilizar en diversos entornos, incluyendo producción, staging, desarrollo, pruebas y flujos de trabajo de CI [19]. Automatiza el proceso de gestión simultánea de varios contenedores Docker [20] y encapsula los requisitos de una aplicación multi-contenedor en una "pila" [20]. Esto mejora la experiencia del desarrollador, facilita la reutilización de la configuración y ayuda a prevenir errores de configuración [20]. A pesar de la aparición de herramientas más avanzadas para la orquestación en producción, Docker Compose sigue siendo relevante para el desarrollo local, las pruebas y los despliegues más sencillos [20].

Docker Compose aborda la complejidad de gestionar múltiples contenedores interconectados proporcionando un enfoque declarativo a través del archivo docker-compose.yml y comandos simplificados.

El archivo docker-compose.yml es un archivo YAML que se utiliza para definir los servicios (contenedores), las redes y los volúmenes de una aplicación [20]. La sección principal services define los contenedores que requiere la aplicación [20]. Cada definición de servicio incluye la imagen Docker a utilizar, las asignaciones de puertos, las variables de entorno, los montajes de volúmenes, las dependencias y las instrucciones de construcción (si es necesario) [20]. La sección networks define las redes a las que se pueden conectar los contenedores [20]. Compose crea automáticamente una red predeterminada [20]. La sección volumes define los volúmenes de almacenamiento persistente [20]. La directiva build especifica un Dockerfile para construir una imagen para un servicio [22]. La directiva ports mapea los puertos entre el host y el contenedor [20]. La directiva volumes monta directorios del host o volúmenes con nombre dentro de los contenedores [20]. La directiva environment establece variables de entorno dentro del contenedor [23]. La directiva depends_on define las dependencias entre servicios y el orden de inicio [23]. La clave version especifica la versión del formato del archivo Compose [22]. Los archivos Compose se pueden combinar, y los archivos posteriores sobrescriben los anteriores [28]. Las rutas relativas en los archivos Compose se resuelven en función del primer archivo de configuración especificado [28]. El archivo docker-compose.yml actúa como un plano para toda la pila de la aplicación, permitiendo aplicar principios de infraestructura como código a los despliegues multi-contenedor.

Docker Compose es la herramienta ideal en varios casos de uso comunes. Principalmente, se utiliza para desarrollar y probar aplicaciones multi-contenedor localmente [19] y para configurar entornos de desarrollo consistentes [19]. También es útil para ejecutar pruebas de integración [19] y para despliegues sencillos en un solo host o para equipos más pequeños [20]. Compose facilita la definición y gestión de dependencias entre servicios [20] y automatiza la configuración de entornos de desarrollo y pruebas [20]. Docker Compose está principalmente enfocado en simplificar la experiencia de desarrollo y prueba local para aplicaciones multi-contenedor. Su alcance limitado a un solo host lo hace menos adecuado para despliegues de producción a gran escala en múltiples máquinas.

4. Docker Swarm: Orquestación Nativa de Contenedores Docker:

Docker Swarm es una herramienta de orquestación de contenedores diseñada para agrupar y programar contenedores Docker [17]. Su propósito principal es permitir a los

administradores de TI y desarrolladores establecer y gestionar un clúster de nodos Docker como un único sistema virtual [17]. Swarm permite unir múltiples máquinas físicas o virtuales en un clúster [33]. Se configura para que múltiples instancias de la aplicación Docker se unan en un clúster [34]. El modo Swarm extiende las capacidades de Docker proporcionando gestión de clústeres, escalado de contenedores, configuración declarativa y soporte para el descubrimiento automático de servicios [29]. Es la herramienta nativa de clustering y programación para contenedores Docker [30] y facilita el despliegue y la gestión de aplicaciones multi-contenedor en múltiples nodos [30]. El objetivo primordial de Docker Swarm es gestionar y escalar aplicaciones contenerizadas a través de un clúster de hosts Docker, proporcionando alta disponibilidad y equilibrio de carga.

La arquitectura de Docker Swarm se basa en nodos manager y worker [36]. Un Swarm consta de múltiples hosts Docker que se ejecutan en modo Swarm y actúan como managers, para gestionar la membresía y la delegación, y como workers, que ejecutan los servicios Swarm [36]. Un host Docker dado puede ser un manager, un worker o desempeñar ambos roles [36]. Los nodos manager asignan tareas a los nodos worker y realizan funciones de orquestación y gestión del clúster [17], mientras que los nodos worker reciben y ejecutan las tareas enviadas por los nodos manager [17]. Un servicio define el estado deseado de una aplicación (número de réplicas, red, almacenamiento, puertos) [17], y Docker trabaja para mantener ese estado deseado [36]. Una tarea es un contenedor en ejecución que forma parte de un servicio Swarm y es gestionado por un manager Swarm [17]. Swarm utiliza el consenso Raft para la coordinación distribuida entre los nodos manager y la tolerancia a fallos [32]. Proporciona una red overlay para una comunicación fluida entre contenedores en diferentes hosts [17] e incluye funciones de equilibrio de carga integradas [17]. Los servicios pueden ser replicados (un número especificado de réplicas) o globales (una instancia por nodo) [17]. Docker Swarm emplea una arquitectura manager-worker para distribuir y gestionar aplicaciones contenerizadas en un clúster, asegurando la resiliencia y la escalabilidad. Los conceptos de servicios y tareas proporcionan una capa de abstracción para definir y ejecutar aplicaciones.

Para desplegar una aplicación en un entorno Swarm, se envía una definición de servicio a un nodo manager [17]. El manager distribuye las tareas (contenedores) a los nodos worker [17]. Los servicios se pueden escalar aumentando o disminuyendo el número de réplicas [17]. Swarm automáticamente reprograma los contenedores en nodos saludables si un nodo falla [30]. Las actualizaciones continuas permiten desplegar nuevas versiones de contenedores sin tiempo de inactividad [17]. Swarm proporciona descubrimiento de servicios, permitiendo que los contenedores se

comuniquen utilizando nombres de servicio [17]. El escalado horizontal se puede lograr añadiendo más nodos o aumentando el número de instancias de contenedor [17]. Docker Swarm simplifica la gestión de aplicaciones distribuidas a través de la programación automatizada de tareas, el escalado y la monitorización de la salud. Esto reduce la sobrecarga operativa de ejecutar aplicaciones en múltiples máquinas.

Docker Swarm se utiliza típicamente para orquestar aplicaciones contenerizadas y garantizar la alta disponibilidad [34]. También se emplea para el equilibrio de carga y el escalado en múltiples nodos [34], simplificando el despliegue de arquitecturas de microservicios [15] y mejorando la utilización de recursos [34]. Es adecuado para pilas de aplicaciones de tamaño pequeño a mediano [17], como el despliegue de bases de datos, aplicaciones web y servicios de backend [29]. Swarm se integra bien en flujos de trabajo de DevOps para automatizar los procesos de despliegue [34] y para gestionar contenedores Docker de manera efectiva [34], garantizando una alta disponibilidad del servicio a través de la redundancia [34]. También se utiliza para aplicaciones de transmisión de eventos y aquellas con un uso intensivo de procesos basados en mensajes [40]. Es una buena opción para casos de uso más sencillos y organizaciones que ya están invertidas en el ecosistema Docker [31], así como para configuraciones de producción ligeras que requieren una orquestación rápida [31] y para equipos que utilizan intensivamente Docker para sus aplicaciones contenerizadas [31]. Docker Swarm es adecuado para organizaciones que necesitan una solución de orquestación de contenedores relativamente simple e integrada, particularmente para despliegues de pequeña a mediana escala y equipos que ya están familiarizados con Docker.

5. Docker Stack: Despliegue de Aplicaciones en Swarm con Compose:

Docker Stack se ejecuta sobre un clúster Docker Swarm [40]. Permite desplegar y agrupar lógicamente múltiples servicios, que son contenedores distribuidos a través de un swarm [40]. Stack ayuda a gestionar la orquestación de múltiples contenedores en varias máquinas [40]. Es una herramienta de despliegue y orquestación integrada en el motor Docker para el modo Docker Swarm [42]. Un stack contiene una colección de contenedores/servicios que pertenecen lógicamente juntos [43] y permite definir y ejecutar aplicaciones multi-contenedor en modo Docker Swarm [42]. Docker Stacks ocurren cuando un Swarm Manager está gestionando múltiples Swarms para múltiples Servicios en un clúster dado para una aplicación dada [44]. Docker Stack se basa en Docker Swarm proporcionando una abstracción de nivel superior para desplegar y gestionar aplicaciones multi-contenedor definidas utilizando archivos Docker Compose en un clúster Swarm.

Para desplegar aplicaciones completas en un clúster Swarm utilizando Docker Stack,

se utiliza un archivo Docker Compose [42]. El comando docker stack deploy se emplea para desplegar un stack en el swarm, aceptando un archivo Compose como entrada [40]. El archivo Compose utilizado con Docker Stack debe especificar la versión '3.0' o superior [27]. Docker Stack ignora las instrucciones build en el archivo Compose, requiriendo imágenes pre-construidas [27]. Las redes especificadas en el archivo Compose se crean como redes overlay en el Swarm [27]. El comando docker stack deploy debe ejecutarse desde un nodo manager en el Swarm [40]. El escalado y la actualización de los servicios en un stack se pueden realizar de forma declarativa volviendo a desplegar el archivo stack o utilizando docker service scale [27]. Docker Stack permite el despliegue de aplicaciones complejas multi-servicio en un clúster Docker Swarm utilizando el formato familiar de Docker Compose, uniendo la brecha entre los despliegues de Compose en un solo host y los entornos distribuidos de Swarm.

La utilización de Docker Stack para la gestión de aplicaciones en entornos de producción ofrece varias ventajas. Permite el despliegue y escalado distribuido de aplicaciones [42] y proporciona una forma sencilla de desplegar y gestionar todo el ciclo de vida de la aplicación (despliegue inicial, comprobaciones de salud, escalado, actualizaciones, rollbacks) [42]. Los servicios se gestionan a través de los nodos del swarm [42] y se crean redes especiales para los servicios dentro del stack [42]. Admite el escalado y las actualizaciones declarativas [42] y es ideal para aplicaciones de procesamiento sin estado [40]. Facilita el despliegue de servicios relacionados como una unidad [27]. Docker Stack está orientado a despliegues de producción al aprovechar las capacidades de Docker Swarm para la alta disponibilidad, la escalabilidad y las actualizaciones continuas, todo ello utilizando una configuración declarativa similar a Compose.

6. Análisis Comparativo: Docker Compose, Docker Swarm y Docker Stack:

Tabla 2: Comparación de Docker Compose, Docker Swarm y Docker Stack

Característica	Docker Compose	Docker Swarm	Docker Stack
Alcance	Un solo host (local)	Multi-host (clúster)	Despliegue en clúster Swarm

Objetivo de Orquestación	Contenedores	Nodos Docker	Servicios (definidos en Compose)
Configuración	docker-compose.yml	Comandos CLI de Docker	docker-compose.yml (versión 3+) y CLI Swarm
Complejidad	Simple	Moderada	Moderada
Escalabilidad	Limitada a los recursos del host	Escalado horizontal en múltiples nodos	Aprovecha la escalabilidad de Swarm
Alta Disponibilidad	No inherente	Tolerancia a fallos y equilibrio de carga	Beneficia de las características de Swarm
Casos de Uso	Desarrollo y pruebas locales	Gestión de aplicaciones distribuidas pequeñas/medianas	Despliegue en producción con Swarm
Curva de Aprendizaje	Baja	Moderada	Moderada
Soporte de Build	Sí	No	No (requiere imágenes pre-construidas)

Docker Compose, Docker Swarm y Docker Stack representan una progresión en términos de alcance y complejidad. Compose se centra en despliegues en un solo host, típicamente para el desarrollo local, mientras que Swarm se dedica a la gestión de clústeres multi-host. Stack, por su parte, actúa como un puente, permitiendo desplegar aplicaciones definidas con Compose en un entorno Swarm [27]. En cuanto al objetivo de la orquestación, Compose gestiona contenedores individuales, Swarm orquesta nodos Docker completos y Stack gestiona servicios, que son definidos en archivos Compose y se ejecutan en un clúster Swarm [44].

La configuración también varía. Compose utiliza el archivo docker-compose.yml para definir la aplicación multi-contenedor [19]. Swarm se configura principalmente a través de comandos de la interfaz de línea de comandos (CLI) de Docker [40]. Stack, en cambio, utiliza un archivo docker-compose.yml (que debe ser de versión 3 o superior)

para la definición de los servicios y los comandos CLI de Swarm para el despliegue [27].

En términos de complejidad, Compose es la herramienta más sencilla, ideal para entornos de desarrollo. Swarm introduce una mayor complejidad debido a la necesidad de gestionar un clúster distribuido. Stack se sitúa en un punto intermedio, ya que se basa en la infraestructura de Swarm pero utiliza un formato de configuración familiar para los usuarios de Compose [30].

La escalabilidad es otra diferencia clave. Compose está limitado por los recursos de un único host. Swarm, en cambio, ofrece escalabilidad horizontal al permitir la distribución de contenedores en múltiples nodos. Stack, al ejecutarse sobre Swarm, se beneficia de esta capacidad de escalado [2].

La alta disponibilidad no es una característica inherente a Compose, ya que se ejecuta en un solo host. Swarm, por otro lado, proporciona tolerancia a fallos y equilibrio de carga integrados, distribuyendo las cargas de trabajo entre los nodos del clúster. Stack, al desplegarse sobre Swarm, también se beneficia de estas características de alta disponibilidad [17].

Los casos de uso más apropiados para cada herramienta también son distintos. Compose es ideal para el desarrollo y las pruebas locales de aplicaciones multi-contenedor [17]. Swarm es más adecuado para la gestión de aplicaciones distribuidas de tamaño pequeño a mediano en entornos de producción. Stack se utiliza principalmente para desplegar aplicaciones multi-contenedor en entornos de producción utilizando un clúster Swarm [17].

La curva de aprendizaje para Compose es la más suave, lo que la hace accesible para principiantes. Swarm requiere una comprensión de los conceptos de clustering y orquestación, lo que implica una curva de aprendizaje moderada. Stack también tiene una curva de aprendizaje moderada, ya que requiere una comprensión tanto de Compose como de los conceptos básicos de Swarm [30].

Finalmente, Compose permite construir imágenes directamente desde el archivo docker-compose.yml utilizando la directiva build. En contraste, Stack no admite la construcción de imágenes y requiere que las imágenes se hayan construido previamente y estén disponibles en un registro de contenedores [27]. En resumen, estas tres herramientas representan una evolución en la gestión de contenedores, comenzando con la simplicidad de Compose para el desarrollo local, pasando a la capacidad de orquestación distribuida de Swarm y culminando en la facilidad de despliegue de Stack para aplicaciones complejas en producción.

7. Ejemplos Prácticos de Despliegue:

Para ilustrar el uso de Docker Compose, consideremos el despliegue de una aplicación web básica junto con una base de datos MySQL en un entorno local [20]. El archivo docker-compose.yml podría tener la siguiente estructura:

YAML

```yaml
version: '3.1'
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./html:/usr/share/nginx/html:ro
    depends_on:
      - db
    networks:
      - mynetwork

  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: mysecretpassword
      MYSQL_DATABASE: mydatabase
    volumes:
      - db_data:/var/lib/mysql
    networks:
      - mynetwork

volumes:
  db_data:

networks:
  mynetwork:
```

En este ejemplo, se definen dos servicios: web y db. El servicio web utiliza la imagen oficial de nginx y mapea el puerto 80 del host al puerto 80 del contenedor. Monta un volumen local ./html en el directorio de nginx para servir contenido web y declara una dependencia del servicio db. El servicio db utiliza la imagen oficial de MySQL 5.7, establece variables de entorno para la contraseña de root y la base de datos, y define un volumen llamado db_data para la persistencia de los datos. Ambos servicios están conectados a una red llamada mynetwork. Para desplegar esta aplicación, se guardaría este archivo como docker-compose.yml y se ejecutaría el comando docker compose up -d en el mismo directorio. Docker Compose construiría (si fuera necesario) y ejecutaría los contenedores, creando la red y el volumen definidos. La aplicación web sería accesible en http://localhost. Este ejemplo ilustra la facilidad con la que Docker Compose gestiona las dependencias y la red de una aplicación multi-capa típica en un entorno de desarrollo local.

Para desplegar la misma aplicación web y base de datos utilizando Docker Stack en un clúster Docker Swarm, se requeriría un archivo docker-compose.yml similar (versión 3 o superior), pero con imágenes pre-construidas para ambos servicios [27]. Asumiendo que ya se ha configurado un clúster Docker Swarm y que las imágenes my-nginx-image:latest y my-mysql-image:5.7 han sido construidas y pushadas a un registro Docker, el archivo docker-compose.yml podría ser:

YAML

```yaml
version: '3.1'
services:
  web:
    image: my-nginx-image:latest
    ports:
      - "80:80"
    volumes:
      - web_content:/usr/share/nginx/html:ro
    depends_on:
      - db
    networks:
      - mynetwork
    deploy:
      replicas: 3
```

```
    update_config:
      parallelism: 2
      delay: 10s
    restart_policy:
      condition: on-failure

  db:
    image: my-mysql-image:5.7
    environment:
      MYSQL_ROOT_PASSWORD: mysecretpassword
      MYSQL_DATABASE: mydatabase
    volumes:
      - db_data:/var/lib/mysql
    networks:
      - mynetwork
    deploy:
      placement:
        constraints: [node.role == worker]

volumes:
  web_content:
  db_data:

networks:
  mynetwork:
    driver: overlay
```

En este caso, la directiva build se ha reemplazado por image especificando las imágenes pre-construidas. Se ha añadido una sección deploy para el servicio web para indicar que se deben desplegar 3 réplicas, con una estrategia de actualización y una política de reinicio. Para el servicio de base de datos, se ha añadido una restricción de placement para que se ejecute solo en nodos worker. La red mynetwork se define como un overlay driver, necesario para la comunicación entre servicios en un Swarm. Para desplegar esta aplicación en el clúster Swarm, se ejecutaría el comando docker stack deploy --compose-file docker-compose.yml mywebapp desde un nodo manager del Swarm. Docker Stack desplegaría los servicios en el clúster, gestionando las réplicas y la red overlay. La aplicación web sería accesible a través de la dirección IP de cualquier nodo del Swarm en el puerto 80, ya que Swarm se encarga del equilibrio de carga. Este ejemplo muestra cómo Docker Stack extiende el

flujo de trabajo de Compose a un entorno distribuido.

8. Conclusión y Recomendaciones:

En este informe se han examinado los conceptos fundamentales de Docker y sus herramientas asociadas para la contenerización y la orquestación. Docker proporciona la base para la creación de contenedores, unidades estandarizadas que encapsulan todo lo necesario para ejecutar una aplicación. Docker Compose simplifica la gestión de aplicaciones multi-contenedor en un solo host a través de un archivo de configuración declarativo. Docker Swarm emerge como una solución de orquestación nativa de Docker, permitiendo la gestión y escalabilidad de contenedores en un clúster de máquinas. Finalmente, Docker Stack facilita el despliegue de aplicaciones definidas con Docker Compose en un clúster Docker Swarm, combinando la facilidad de configuración de Compose con las capacidades de orquestación distribuida de Swarm. Estas tecnologías se complementan, ofreciendo soluciones para diferentes escalas y necesidades de despliegue.

La elección de la herramienta más adecuada depende en gran medida de la escala de la aplicación, el entorno de despliegue (local vs. distribuido) y la complejidad de los requisitos de orquestación.

- **Utilice Docker Compose cuando:** necesite desarrollar y probar aplicaciones multi-contenedor localmente, gestionar dependencias entre servicios en un solo host o configurar entornos locales sencillos.
- **Utilice Docker Swarm cuando:** requiera orquestar contenedores en múltiples hosts Docker para lograr escalabilidad y alta disponibilidad, gestionar despliegues de producción de tamaño pequeño a mediano o si ya está familiarizado con el ecosistema Docker y busca una solución de orquestación nativa.
- **Utilice Docker Stack cuando:** necesite desplegar aplicaciones multi-contenedor definidas en un archivo Docker Compose en un clúster Docker Swarm para entornos de producción, aprovechando las capacidades de orquestación de Swarm con un formato de configuración Compose familiar.
- **Considere Kubernetes cuando:** gestione aplicaciones de microservicios complejas a gran escala, necesite funciones avanzadas de auto-escalado y auto-recuperación, o requiera una amplia integración con proveedores de nube y herramientas de terceros.

Comprender las fortalezas y limitaciones de cada herramienta es crucial para tomar la decisión correcta en función de las necesidades específicas de cada proyecto y

entorno.

Obras citadas

1. aws.amazon.com, fecha de acceso: marzo 28, 2025, https://aws.amazon.com/docker/#:~:text=Docker%20is%20a%20software%20platform,tools%2C%20code%2C%20and%20runtime.
2. What is Docker? | AWS, fecha de acceso: marzo 28, 2025, https://aws.amazon.com/docker/
3. What is Docker - DevZero, fecha de acceso: marzo 28, 2025, https://www.devzero.io/blog/what-is-docker-how-does-it-works
4. Introduction to Docker: how docker solves deployment problems - DevOps Maestro, fecha de acceso: marzo 28, 2025, https://devopsmaestro.hashnode.dev/what-problem-does-docker-solve-beginners-introduction-to-docker
5. Docker: Accelerated Container Application Development, fecha de acceso: marzo 28, 2025, https://www.docker.com/
6. Why Docker | Docker, fecha de acceso: marzo 28, 2025, https://www.docker.com/why-docker/
7. Solving Software Development Challenges with Docker: How Docker Streamlines Workflow Challenges | by Husayn Fakher | Medium, fecha de acceso: marzo 28, 2025, https://medium.com/@thecuriouschronicles/solving-software-development-challenges-with-docker-how-docker-streamlines-workflow-challenges-9c007849d047
8. What Is Docker? | IBM, fecha de acceso: marzo 28, 2025, https://www.ibm.com/think/topics/docker
9. What is Docker? | Docker Docs, fecha de acceso: marzo 28, 2025, https://docs.docker.com/get-started/docker-overview/
10. medium.com, fecha de acceso: marzo 28, 2025, https://medium.com/@thecuriouschronicles/solving-software-development-challenges-with-docker-how-docker-streamlines-workflow-challenges-9c007849d047#:~:text=Conflicting%20dependencies%2C%20version%20mismatches%2C%20and,thus%20streamlining%20the%20development%20workflow.
11. What problems Docker solves in the software development process? - TechWorld with Nana, fecha de acceso: marzo 28, 2025, https://www.techworld-with-nana.com/post/what-problems-docker-solves
12. What problems does Docker really solve? - DEV Community, fecha de acceso: marzo 28, 2025, https://dev.to/techworld_with_nana/what-problems-does-docker-really-solve-496a
13. Top Developer Productivity Challenges and How Docker Solves Them, fecha de acceso: marzo 28, 2025, https://www.docker.com/resources/top-developer-productivity-challenges-how-docker-solves-them-white-paper/

14. Exploring Docker for DevOps: What It Is and How It Works, fecha de acceso: marzo 28, 2025, https://www.docker.com/blog/docker-for-devops/
15. 15 Most Common Docker Use Cases in 2024 - Folio3 Cloud Services, fecha de acceso: marzo 28, 2025, https://cloud.folio3.com/blog/docker-use-cases/
16. what are the real world use cases of docker containers? [closed] - Stack Overflow, fecha de acceso: marzo 28, 2025, https://stackoverflow.com/questions/48470051/what-are-the-real-world-use-cases-of-docker-containers
17. What is Docker Swarm and How Does it Work? - LogicMonitor, fecha de acceso: marzo 28, 2025, https://www.logicmonitor.com/blog/what-is-docker-swarm-and-how-does-it-work
18. docs.docker.com, fecha de acceso: marzo 28, 2025, https://docs.docker.com/compose/#:~:text=Docker%20Compose%20is%20a%20tool,efficient%20development%20and%20deployment%20experience.
19. Docker Compose, fecha de acceso: marzo 28, 2025, https://docs.docker.com/compose/
20. Docker Compose - What is It, Example & Tutorial - Spacelift, fecha de acceso: marzo 28, 2025, https://spacelift.io/blog/docker-compose
21. docker/compose: Define and run multi-container applications with Docker - GitHub, fecha de acceso: marzo 28, 2025, https://github.com/docker/compose
22. docker/compose - Docker Image, fecha de acceso: marzo 28, 2025, https://hub.docker.com/r/docker/compose
23. Docker Compose Up Command Explained | Built In, fecha de acceso: marzo 28, 2025, https://builtin.com/articles/docker-compose-up
24. How Compose works - Docker Docs, fecha de acceso: marzo 28, 2025, https://docs.docker.com/compose/intro/compose-application-model/
25. How to Use Docker Compose and YAML for Multi-Container Applications | Day 18 of 90 Days Of DevOps - Ajit Fawade, fecha de acceso: marzo 28, 2025, https://ajitfawade.medium.com/how-to-use-docker-compose-and-yaml-for-multi-container-applications-day-18-of-90-days-of-devops-78261fbd7b37
26. The docker-compose.yml file | Divio Documentation, fecha de acceso: marzo 28, 2025, https://docs.divio.com/reference/docker-docker-compose/
27. Docker Compose, Docker Stack and Docker Swarm - Digi Hunch, fecha de acceso: marzo 28, 2025, https://www.digihunch.com/2020/05/docker-swarm-brief-notes/
28. docker compose - Docker Docs, fecha de acceso: marzo 28, 2025, https://docs.docker.com/reference/cli/docker/compose/
29. Docker Swarm vs. Kubernetes - Key Differences Explained - Spacelift, fecha de acceso: marzo 28, 2025, https://spacelift.io/blog/docker-swarm-vs-kubernetes
30. Kubernetes vs Docker Swarm: Which to Choose for Containers? - Last9, fecha de acceso: marzo 28, 2025, https://last9.io/blog/kubernetes-vs-docker-swarm/
31. Common Orchestrators, Docker Swarm | Cycle.io, fecha de acceso: marzo 28, 2025, https://cycle.io/learn/common-orchestrator-docker-swarm
32. What is Docker Swarm? - Sysdig, fecha de acceso: marzo 28, 2025,

https://sysdig.com/learn-cloud-native/what-is-docker-swarm/
33. www.techtarget.com, fecha de acceso: marzo 28, 2025, https://www.techtarget.com/searchitoperations/definition/Docker-Swarm#:~:text=Docker%20Swarm%20is%20a%20container,virtual%20machines%20into%20a%20cluster.
34. Docker Swarm - definition & overview - Sumo Logic, fecha de acceso: marzo 28, 2025, https://www.sumologic.com/glossary/docker-swarm/
35. Kubernetes vs. Docker Swarm: A Comprehensive Comparison (2023) - KodeKloud, fecha de acceso: marzo 28, 2025, https://kodekloud.com/blog/kubernetes-vs-docker-swarm/
36. Swarm mode key concepts - Docker Docs, fecha de acceso: marzo 28, 2025, https://docs.docker.com/engine/swarm/key-concepts/
37. Mastering Docker Swarm: A Deep Dive into Container Orchestration - Ayushmaan Srivastav, fecha de acceso: marzo 28, 2025, https://srivastavayushmaan1347.medium.com/mastering-docker-swarm-a-deep-dive-into-container-orchestration-2a383e8808ec
38. A Look At Docker Swarm & Use Cases | by Leroy Leow | Medium, fecha de acceso: marzo 28, 2025, https://medium.com/@leroyleowdev/a-look-at-docker-swarm-use-cases-c2c5a3a5077e
39. Mastering Docker Swarm: Setting Up a Cluster and Ensuring Fault Tolerance - Medium, fecha de acceso: marzo 28, 2025, https://medium.com/@priyamsanodiya340/mastering-docker-swarm-setting-up-a-cluster-and-ensuring-fault-tolerance-fcd6d526f7c9
40. What is Docker Stack and how do you use it? - Ronald James Group, fecha de acceso: marzo 28, 2025, https://www.ronaldjamesgroup.com/article/docker-stack
41. www.ronaldjamesgroup.com, fecha de acceso: marzo 28, 2025, https://www.ronaldjamesgroup.com/article/docker-stack#:~:text=Docker%20Stack%20is%20run%20across,be%20deployed%20and%20grouped%20logically.
42. Deploying App With Docker Stacks - Medium, fecha de acceso: marzo 28, 2025, https://medium.com/@leroyleowdev/deploying-app-with-docker-stacks-24ea5a863fb7
43. Using Docker Swarm & Docker Stack to Deploy WordPress & MySQL | by Nife Sofowoke, fecha de acceso: marzo 28, 2025, https://medium.com/@nifemi.sofowoke/using-docker-swarm-docker-stack-to-deploy-wordpress-mysql-553e747b34b0
44. Docker Swarms and Stacks: What's the difference?, fecha de acceso: marzo 28, 2025, https://stackoverflow.com/questions/44500394/docker-swarms-and-stacks-whats-the-difference
45. Deploy a stack to a swarm - Docker Docs, fecha de acceso: marzo 28, 2025, https://docs.docker.com/engine/swarm/stack-deploy/
46. The Difference Between Docker Compose And Docker Stack - vsupalov.com, fecha de acceso: marzo 28, 2025, https://vsupalov.com/difference-docker-compose-and-docker-stack/

47. Docker-compose vs docker stack for local development, fecha de acceso: marzo 28, 2025, https://forums.docker.com/t/docker-compose-vs-docker-stack-for-local-development/61422
48. docker - What's the difference between a stack file and a Compose file? - Stack Overflow, fecha de acceso: marzo 28, 2025, https://stackoverflow.com/questions/43099408/whats-the-difference-between-a-stack-file-and-a-compose-file
49. Use containerized databases | Docker Docs, fecha de acceso: marzo 28, 2025, https://docs.docker.com/guides/databases/
50. “A Beginner's Guide to Docker Compose: Building a Simple Web Application” | by Prajwal Patil | Medium, fecha de acceso: marzo 28, 2025, https://medium.com/@prajwalpatil0402/a-beginners-guide-to-docker-compose-building-a-simple-web-application-c33e32df5d0d
51. Sample apps with Compose - Docker Docs, fecha de acceso: marzo 28, 2025, https://docs.docker.com/compose/support-and-feedback/samples-for-compose/
52. Awesome Docker Compose samples - GitHub, fecha de acceso: marzo 28, 2025, https://github.com/docker/awesome-compose
53. Docker Compose Quickstart, fecha de acceso: marzo 28, 2025, https://docs.docker.com/compose/gettingstarted/
54. How can we create full stack application like frontend backend and database in one Docker image, fecha de acceso: marzo 28, 2025, https://stackoverflow.com/questions/70259105/how-can-we-create-full-stack-application-like-frontend-backend-and-database-in-o
55. Docker Stack – my full sample - Guerilla Programmer, fecha de acceso: marzo 28, 2025, https://guerillaprogrammer.com/2017/10/06/docker-stack-my-full-sample/
56. Dockerizing the Front- and Backend - Tutorial: Full Stack Web App - milanwittpohl.com, fecha de acceso: marzo 28, 2025, https://www.milanwittpohl.com/projects/tutorials/full-stack-web-app/dockerizing-our-front-and-backend