UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA EN COMPUTACIÓN

INGENIERA:

Karens Medrano

ACTIVIDAD:

Investigación Aplicada 2

ASIGNATURA:

Diseño y Programación de Software Multiplataforma G01T

FECHA DE ELABORACION: 14/9/2024

CICLO 02 / 2024

No.	INTEGRANTES:	CARNET	Grupo Teórico
1	Samuel Orlando Aguilar Recinos	AR200454	01T
2	Kevin Salvador Casamalhuapa Turcios	CT142074	01T
3.	Daniel Alexander Castellanos Romero	CR221376	01T
4	Luis Alberto Lino Mejía	LM151857	01T
5	Cristian Enrique Pineda Muñoz	PM190654	01T

Contenido

Introducción

En el ámbito del desarrollo de software moderno, la necesidad de entornos de ejecución consistentes y reproducibles se ha convertido en una prioridad. Los contenedores Docker han emergido como una solución clave para abordar estos desafíos, revolucionando la forma en que los desarrolladores crean, despliegan y gestionan aplicaciones. Docker es una plataforma que facilita la creación, despliegue y ejecución de aplicaciones en contenedores, los cuales encapsulan todo lo necesario para que una aplicación funcione correctamente, incluyendo el código, las dependencias y las configuraciones.

El uso de Docker ofrece numerosas ventajas en comparación con los enfoques tradicionales. Primero, proporciona un entorno de desarrollo homogéneo, asegurando que todos los miembros del equipo trabajen con las mismas configuraciones y versiones de software, eliminando así el clásico problema del "funciona en mi máquina". Segundo, Docker simplifica el proceso de pruebas al permitir la creación de entornos de prueba aislados que replican el entorno de producción, lo que resulta en una mayor fiabilidad en los resultados. Finalmente, Docker facilita el despliegue al permitir que las mismas imágenes que se utilizan en desarrollo y pruebas se implementen directamente en producción, garantizando la consistencia a lo largo de todo el ciclo de vida del software.

Este trabajo se centra exclusivamente en Docker y explora cómo esta herramienta puede mejorar significativamente cada etapa del ciclo de vida de una aplicación. A través de ejemplos prácticos y descripciones detalladas, se analizarán los beneficios de Docker en el desarrollo, pruebas y despliegue de aplicaciones, así como sus herramientas complementarias y mejores prácticas para superar posibles desafíos.

1. Definir qué son los contenedores:

a) Explicar el concepto de contenedores en el contexto del desarrollo de software.

Los contenedores de software son una forma de virtualización que permite ejecutar aplicaciones de forma aislada en un entorno compartido. En términos simples, los contenedores en informática son como cajas mágicas que pueden guardar programas y todo lo que necesitan para funcionar, dentro de estas cajas, ponemos un programa, como un juego en una videoconsola. Pero no solo ponemos el juego, también ponemos todas las cosas que el juego necesita, como los gráficos, sonidos y reglas del juego. Así, el juego funciona igual en cualquier lugar donde pongamos esa caja mágica.

Los contenedores ayudan a los programadores a hacer que sus programas sean más fáciles de mover y usar en diferentes ordenadores, sin importar el tamaño del sistema o el entorno operativo utilizado (Windows, Mac o Linux). Mantienen las aplicaciones aisladas entre sí y permiten automatizar el desarrollo de las mismas.

b) Diferenciar entre contenedores y máquinas virtuales.

	Contenedores	Máquinas virtuales
Funcionamiento	implica la creación de paquetes de software autosuficientes que funcionen de manera uniforme, independientemente de las máquinas en las que se ejecuten	implica la instalación de software de virtualización en un servidor o una computadora físicos. La computadora física es la computadora host y la máquina virtual es el invitado.

Tecnología Principal	utilizan un motor de contenedores o un tiempo de ejecución de los contenedores. Este es el software que actúa como un agente intermediario entre los contenedores y el sistema operativo. Esto proporciona y administra los recursos de sistema que necesita la aplicación. Docker es el motor de contenedores de código abierto más usado.	utilizan hipervisores que se comunican entre el sistema operativo invitado y el sistema operativo host. El hipervisor coordina el uso compartido de los recursos para que la máquina virtual se ejecute de forma aislada junto con otras en el mismo hardware.
Tamaño	Los archivos de contenedor son más ligeros y se pueden medir en MB. En los contenedores solo se empaquetan los recursos necesarios para ejecutar una única aplicación.	Los archivos de imagen de tienen un mayor tamaño (varios GB), ya que contienen su propio sistema operativo. El aumento de recursos significa que puede duplicar, dividir, abstraer y emular servidores, bases de datos, escritorios y redes enteros.
Sistema Operativo	Ejecuta la parte del modo de usuario de un sistema operativo y se puede personalizar para que contenga solo los servicios necesarios para la aplicación,	Ejecuta un sistema operativo completo incluido el kernel, lo que requiere más recursos del sistema (CPU, memoria y almacenamiento).

	con menos recursos del sistema.	
Redes	Usa una vista aislada de un adaptador de red virtual, lo que proporciona un poco menos de virtualización: el firewall del host se comparte con los contenedores, a la vez que se usan menos recursos.	Usa adaptadores de red virtual.
Tolerancia a errores	Si se produce un error en un nodo del clúster, el orquestador vuelve a crear rápidamente los contenedores que se ejecutan en él en otro nodo del clúster.	Las máquinas virtuales pueden conmutar por error a otro servidor de un clúster, reiniciando el sistema operativo de la máquina virtual en el nuevo servidor.

2. Describir el funcionamiento de los contenedores:

a) Analizar cómo los contenedores empaquetan aplicaciones y sus dependencias.

Las dependencias de los contenedores se refieren a los recursos y componentes necesarios para que un contenedor funcione correctamente. Estas dependencias pueden variar según el tipo de contenedor y la aplicación que se está ejecutando. Como las siguientes

1- Sistema Operativo Base: Los contenedores suelen estar construidos sobre una imagen base que proporciona el sistema operativo subyacente (como una imagen de Ubuntu, Alpine, etc.).

2- Librerías y Paquetes: Las aplicaciones dentro de los contenedores necesitan ciertas librerías y paquetes. Estas se incluyen en la imagen del contenedor y pueden variar según el software que se esté ejecutando.

3- Configuraciones y Archivos de Datos: Los contenedores a menudo dependen de archivos de configuración y datos específicos para funcionar correctamente. Estos pueden ser volúmenes montados desde el host o compartidos entre contenedores.

4- Redes: Los contenedores suelen requerir una configuración de red para comunicarse entre sí y con el mundo exterior. Esto incluye puertos expuestos y configuraciones de red.

5- Docker Engine o Plataforma de Contenedores: Los contenedores se ejecutan sobre un motor de contenedores como Docker, Podman, o una plataforma de orquestación como Kubernetes. La versión y configuración de este motor también son una dependencia crucial.

6- Recursos del Sistema: Los contenedores utilizan recursos del sistema como CPU, memoria y almacenamiento. La disponibilidad y asignación de estos recursos pueden afectar el rendimiento del contenedor.

7- Dependencias del Entorno de Ejecución: Algunas aplicaciones pueden necesitar servicios específicos (como bases de datos o colas de mensajes) que deben estar disponibles en el entorno donde el contenedor se ejecuta.

8- Imágenes de Contenedores: Las imágenes que se utilizan para crear contenedores pueden tener sus propias dependencias, como otras imágenes base o capas específicas.

Los contenedores empaquetan aplicaciones de manera eficiente al incluir todo lo necesario para que la aplicación se ejecute de forma consistente en cualquier entorno. Se realiza de la siguiente manera:

- Imagen de Contenedor: se basa en una imagen, que es una plantilla inmutable que contiene todo el código, las dependencias, las bibliotecas y las configuraciones necesarias para ejecutar la aplicación. La imagen se crea a partir de un Dockerfile (en Docker) o un archivo similar, que especifica cómo construir la imagen paso a paso. Esto incluye la base del sistema operativo, las herramientas y librerías necesarias, y la propia aplicación.

- Sistema de Archivos del Contenedor: Cada contenedor tiene su propio sistema de archivos que está basado en la imagen, aislado del sistema operativo subyacente y de otros contenedores. Esto se logra mediante un sistema de archivos en capas. Las imágenes de contenedores se construyen en capas. Cada instrucción en el Dockerfile crea una nueva capa en la imagen. Esto permite que las imágenes sean modulares y reutilizables, ya que las capas comunes pueden ser compartidas entre diferentes imágenes.

- Dependencias y Configuraciones: Las dependencias de la aplicación, como librerías y paquetes, se incluyen dentro de la imagen. Esto asegura que la aplicación tenga todas las dependencias necesarias sin importar dónde se ejecute. Las configuraciones específicas de la aplicación se pueden definir en archivos dentro de la imagen o a través de variables de entorno que se establecen cuando se ejecuta el contenedor.

- Entorno de Ejecución: Cuando se ejecuta un contenedor, se utiliza el motor de contenedores (como Docker, Podman, etc.) para crear una instancia de la imagen. El motor maneja el aislamiento de recursos, la red y la comunicación entre contenedores. Los contenedores utilizan tecnologías como namespaces y cgroups para garantizar que cada contenedor tenga su propio espacio de usuario, red, y recursos, proporcionando un entorno aislado para cada aplicación.

- Portabilidad y Consistencia: Dado que los contenedores incluyen todas las dependencias necesarias y se ejecutan de manera aislada, las aplicaciones pueden moverse y ejecutarse de manera consistente en diferentes entornos, como desarrollo, pruebas y producción. La aplicación dentro del contenedor se comportará de la misma manera independientemente del sistema operativo subyacente, ya que el contenedor incluye su propio entorno de ejecución.

b) Explicar el proceso de aislamiento y ejecución de contenedores en el sistema operativo.

El aislamiento y la ejecución de contenedores en un sistema operativo dependen de dos mecanismos clave del kernel de Linux: los encabezados y los grupos c. Al crear entornos distintos para cada contenedor, los namespaces permiten que tenga su propio sistema de archivos, red, usuarios y procesos. Esto crea la impresión de que cada contenedor se ejecuta en un sistema independiente porque asegura que los contenedores no interfieran entre sí ni con el sistema host.

Sin embargo, los grupos de control (cgroups) controlan y limitan el uso de recursos como la CPU, la memoria, la red y el disco para cada contenedor. Esto garantiza que ningún contenedor use más recursos de los asignados, manteniendo el rendimiento y la estabilidad del sistema.

Para aislar y controlar la ejecución de un contenedor, se crea un conjunto de namespaces y se aplican cgroups. El contenedor emplea un sistema de archivos en capas (UnionFS), donde la imagen base es solo de lectura y cualquier cambio se guarda en las capas superiores, lo que hace que el almacenamiento sea optimizado. Por lo tanto, los contenedores brindan un entorno portátil, seguro y eficiente para el desarrollo y la ejecución de aplicaciones.

3. Explorar las ventajas de utilizar contenedores:

a. Discutir los beneficios en términos de portabilidad, eficiencia y consistencia.

A Beneficios en términos de portabilidad, eficiencia y consistencia.

- Portabilidad:

Independencia del entorno: Los contenedores empaquetan una aplicación con todas sus dependencias, lo que permite que se ejecute en cualquier sistema que soporte contenedores sin necesidad de realizar ajustes. Esto permite que las aplicaciones se muevan entre varios entornos (desarrollo, pruebas, producción) y plataformas (local, nube, servidores físicos). Unión entre plataformas: al estar encapsulado en un contenedor, el software se comporta de la misma manera en diferentes entornos, eliminando el problema de "funciona en mi máquina, pero no en producción".

- Eficiencia:

Menor uso de recursos: Los contenedores comparten el kernel del sistema operativo host, lo que reduce la sobrecarga de recursos como CPU, memoria y almacenamiento. Son más ligeros y rápidos de arrancar, lo que permite que un host ejecute más contenedores.

Ejecución rápida: Los contenedores no necesitan cargar un sistema operativo completo como las VM, por lo que se inician rápidamente. Esto es particularmente ventajoso en áreas donde se requiere iniciar o expandir servicios rápidamente.

Mejor utilización de hardware: porque son más ligeros, pueden usar más hardware, maximizando la capacidad de cómputo disponible.

- Consistencia

Entorno reproducible: los contenedores eliminan las inconsistencias entre el entorno de desarrollo y producción asegurando que las aplicaciones funcionen en el mismo entorno, independientemente de dónde estén desplegadas.
Configuraciones estandarizadas: Los desarrolladores pueden asegurarse de que todos los miembros del equipo trabajen con la misma configuración y versiones de dependencias utilizando archivos de configuración como Dockerfile, lo que reduce los errores y los malentendidos.
Ciclo de vida predecible: Las aplicaciones dentro de contenedores permiten pruebas automatizadas, integración y despliegue continuos (CI/CD), mejorando la calidad y confiabilidad del software.

b. Ejemplificar cómo los contenedores facilitan el desarrollo, pruebas y despliegue de aplicaciones.

El uso de contenedores en el desarrollo de aplicaciones React simplifica y agiliza todo el ciclo de vida del software: desarrollo, pruebas y despliegue. Aquí tienes un ejemplo práctico de cómo Docker facilita cada etapa:

1. Desarrollo

Al usar Docker, puedes crear un entorno de desarrollo homogéneo para todos los miembros del equipo. Con un simple archivo Dockerfile, puedes definir las dependencias necesarias, como Node.js y las herramientas de desarrollo de React, asegurando que todos trabajen en el mismo entorno.
Ejemplo de Dockerfile para una aplicación React:

```
# Usa una imagen oficial de Node.js
FROM node:16

# Establece el directorio de trabajo en el contenedor
WORKDIR /app

# Copia el package.json e instala las dependencias
COPY package*.json ./
RUN npm install

# Copia el código de la aplicación
COPY . .

# Expone el puerto donde correrá la app
EXPOSE 3000

# Comando para iniciar la aplicación en modo desarrollo
CMD ["npm", "start"]
```

Este archivo asegura que cualquier desarrollador pueda clonar el repositorio, construir la imagen con Docker (docker build -t react-app .) y correr la aplicación (docker run -p 3000:3000 react-app). Todo el entorno de desarrollo estará listo sin conflictos de versiones o configuraciones.

2. Pruebas

Las pruebas también se benefician de los contenedores, ya que puedes definir un entorno de pruebas aislado. En lugar de ejecutar pruebas localmente con configuraciones variables, puedes correrlas en un contenedor que tenga las mismas dependencias y configuraciones que el entorno de desarrollo y producción.
Añadiendo pruebas en el Dockerfile:

```
# Comando para correr las pruebas
CMD ["npm", "test"]
```

Al ejecutar docker run react-app, puedes asegurarte de que las pruebas se ejecuten en un entorno limpio y consistente. Incluso se puede integrar con sistemas de CI/CD (como Jenkins o GitLab CI) para ejecutar las pruebas automáticamente cada vez que se hace un commit.

3. Despliegue

El despliegue con contenedores es sencillo y confiable. Después de probar la aplicación en contenedores, puedes usar la misma imagen de Docker para desplegarla en producción, asegurando que lo que funcionó en desarrollo y pruebas, funcionará en producción sin sorpresas.

Para desplegar, simplemente construyes la imagen y la subes a un registro de contenedores como Docker Hub o Amazon ECR. Luego, puedes desplegar la imagen en un servicio como Kubernetes, AWS ECS, o directamente en un servidor con Docker:

```
docker build -t react-app .
docker tag react-app your-registry/react-app:v1
docker push your-registry/react-app:v1
```

En producción, el comando docker run -p 80:3000 your-registry/react-app:v1 levantará la aplicación React usando la misma imagen que se usó en desarrollo y pruebas, garantizando consistencia en todos los entornos.
Resumen:

Desarrollo: Los contenedores crean un entorno consistente y aislado para evitar conflictos de dependencias.
Pruebas: Las pruebas se ejecutan en un entorno controlado, asegurando fiabilidad.
Despliegue: La misma imagen de Docker se utiliza en producción, garantizando que el código que funcionó localmente funcionará en producción.

Esto simplifica el ciclo completo de desarrollo de una aplicación React, desde la codificación hasta el despliegue.

4. Presentar herramientas populares en el ecosistema de contenedores:

a. Describir y comparar Docker, Kubernetes y otras herramientas relevantes.

¿Qué es Docker? Puede ser una de dos cosas: la empresa promotora del proyecto Docker, Inc que a su vez promueve el desarrollo del proyecto de código abierto, basado en Linux, que permite a los programadores desarrollar, crear, e implementar aplicaciones de manera rápida.

¿Pero cómo lo hace? Los entornos de Docker son ejecutables, independientes e integran todo lo necesario para que una aplicación funcione correctamente. Estos entornos poseen sistemas de archivos, como el de un sistema operativo o librerías necesarias para su ejecución, el propio código de la aplicación con sus dependencias, archivos de configuración, el uso de puertos de red específicos y scripts o comandos, ejecutados al momento de inicializar el contenedor.

Podríamos comparar su funcionamiento con inicializar una máquina virtual que funcione con un servidor específico, como uno web, donde el entorno donde se ejecuta posee lo necesario para su correcto funcionamiento.

Existen dos versiones: el Docker Engine, que funciona únicamente en Linux, y el Docker Engine, diseñado para funcionar en Windows y MacOS debido a no poder ejecutar contenedores de manera nativa, sirviéndose de la virtualización para poder crear el entorno de ejecución.

¿Pero qué otras herramientas existen? Gracias al estándar OCI, Open Container Initiative, permitió la creación de nuevos entornos de creación de contenedores, entre ellos se encuentra Podman, creado por RedHat, que también es posible ejecutar en entornos de Windows y MacOS.

En un inicio Podman fue competencia de Docker, pero no poseía las ventajas de Docker, cómo lo son el uso de un proceso central, en caso de Docker, que evita al sistema operativo manejar al contenedor el mismo, como Podman.

Pero Podman fue diseñado para trabajar en entornos donde la seguridad es uno de los aspectos más importantes, ya que Podman solo puede trabajar con un usuario designado que posea los permisos adecuados, buscando evitar que cualquier usuario pueda ejecutar el entorno, , incluido el root, como es el caso de Docker.

Ahora, sabemos que Podman es otra alternativa a la creación de contenedores, pero lo hace como si fuera una capsula, estas capsulas permiten que varios contenedores puedan ser ejecutados al mismo tiempo dentro de ellas. De aquí es donde viene el nombre de Podman, es un administrador de estas capsulas, pod-manager.

¿Pero qué otros contenedores pueden ser usados dentro de Podman? Los Kubernetes, también llamados K8s.

Al igual que Docker y Podman, Kubernetes también se encuentra basado en código libre y puede manipular contenedores que permiten la ejecución de aplicaciones, como lo es Docker.

Los Kubernetes poseen una capacidad de escalar, crecer, sin la necesidad de aumentar nuestros equipos donde ejecutamos aplicaciones, son muy flexibles y, al igual que los contenedores mencionados anteriormente, se encuentra en código abierto.

Se puede trabajar en ellos de manera local/física, en la nube o de manera híbrida, combinando los dos casos.

¿Pero en qué se diferencian Kubernetes y Docker? Mientras que Docker es solo una plataforma para contener una aplicación durante su tiempo de ejecución mientras que Kubernetes permite ejecutar y gestionar varios contenedores, así como monitorizar su estado de ejecución y llegar a reiniciarlos en caso de presentar fallos.

Al utilizar Kubernetes solo hay que preocuparse de escribir el código de las aplicaciones, ya que toda infraestructura ya estaría equipada con lo necesario para trabajar el código.

A largo plazo, en comparación, si se busca tener una mayor flexibilidad y escalabilidad, la capacidad de expandirse mientras más eventos y necesidades existan, Kubernetes es una opción más confiable que Docker, aunque su instalación en un inicio es mucho más complicada.

b. Mostrar casos de uso y ejemplos prácticos de cada herramienta.

1. Docker
    - Para la creación de entornos de desarrollo aislados para nuestros proyectos, como puede ser un entorno para una aplicación de NodeJS en que se encuentren instaladas y listas para usar diferentes librerías, mismas que tendrán un rol en cada paso del desarrollo.
    - Cuando se necesite almacenar la aplicación o servicio realizado en equipos, como discos duros externos, que puedan ser movilizados a otro sitio sin problemas.
    - Si necesitamos crear una imagen del contenedor, como si se tratase de la imagen ISO de un sistema operativo, que pueda ser luego visualizado en otro equipo.
    - Sin importar donde se use el contenedor, tendremos la seguridad de que el funcionamiento dentro de este no variaría.
2. Podman
    - Si se trabaja en una mediana o gran empresa, pero no se tienen los fondos para trabajar con una licencia de Docker, se puede usar para trabajar de manera libre y gratuita sin pagar licencias.
    - Cuando la aplicación almacenada en el contenedor sea una con la que se trabaje con datos sumamente delicados, ya que solo un usuario, con los privilegios o permisos adecuados, pueden acceder y ejecutar la aplicación. Aclarando, este usuario no es el mismo que un usuario root, ya que es posible que más de una persona tenga acceso al usuario root y por ende al contenedor.
    - Cuando se tenga planeado el usar Kubernetes, debido a la compatibilidad existente entre ambos.
    - Relacionado al punto anterior, si se busca compatibilidad, con una opción diferente a Kubernetes, se puede usar otro contenedor que se encuentre basado en OCI, como lo puede ser Buildah o Skopeo.
    - Para administrar contenedores durante todo su ciclo de vida: la ejecución, su networking, su eliminación.
    - Cuando no se busca utilizar un deamon, un proceso aparte que se encargara de la ejecución del contenedor, sino que el propio sistema operativo se encargara del manejo del contenedor.

3. Kubernetes
    - Cuando se tiene planeado el crecimiento a gran escala de la aplicación o servicio que se tiene dentro del contenedor.
    - Para el desarrollo e implementación de una inteligencia artificial, así como también la subsiguiente enseñanza que esta tendrá para poder realizar bien su trabajo.
    - Para la administración de microservicios, esto es visto normalmente con aplicaciones basadas en la nube.
    - Relacionado al punto anterior, se puede usar en entornos para guardar y acceder a aplicaciones en entornos locales, en la nube o híbridos.

5. Discutir los desafíos y consideraciones al usar contenedores:

a. Identificar posibles limitaciones y problemas comunes.

- Limitaciones
    - Persistencia de Datos: Los contenedores no guardan la información por siempre, ya que al ser eliminados se pierde toda la información almacenada.
    - Tiempo de Inicio: Algunos contenedores pueden ser lentos para iniciar, sobre todo si la información almacenada por ejemplo imágenes, tiene dependencias pesadas.
    - Redes complejas: La configuración de redes entre contenedores puede ser compleja, especialmente en entornos distribuidos.
    - Orquestación y escalabilidad: Gestionar una gran cantidad de contenedores de forma manual no es sencillo, es por eso que herramientas como kubernetes son necesarias para escalarlos adecuadamente.

- Problemas comunes
    - Seguridad: Los OS compartidos en los contenedores pueden aumentar la probabilidad de ataque si se compromete el host o las imágenes no están en un lugar seguro.

- **Gestión de recursos:** Cuando no se gestionan bien los recursos, un contenedor consume más de lo esperado, afectando al resto.
- **Depuración:** Depurar aplicaciones dentro de contenedores puede ser más complejo que hacerlo desde el OS anfitrión, requiriendo herramientas adicionales.
- **Sobrecarga de Red:** Cuando se despliega a gran escala, la comunicación entre varios contenedores puede sobrecargar la red si no se gestiona correctamente.

b. Proponer soluciones y mejores prácticas para abordar estos desafíos

- **Limitaciones**
  - **Persistencia de Datos:** Implementar volúmenes de datos persistentes (Persisent Volumes, PV) y Persistent Volume Claims (PVC) en Kubernetes, o utilizar servicios externos como bases de datos gestionadas y almacenamiento en la nube, de este modo sea seguro que los datos no se perderán al momento de eliminar el contenedor.
  - **Tiempo de inicios:** Minimizar el tamaño de las imágenes de todos los contenedores, eliminando dependencias innecesarias y utilizando imágenes base más ligeras. Algunos ejemplos son: Photon OS, Distroless, Alpine Linux, etc.
  - **Redes Complejas:** Utilizar herramientas como Weave o Calico para simplificar la gestión de redes, y emplear políticas de red para gestionar el tráfico entre contenedores. Además, habilitar la segmentación de red entre pods para mejorar el aislamiento.
  - **Orquestación y Escalabilidad:** Implementar Kubernetes para automatizar la gestión y escalado de contenedores en lugar de hacerlo manualmente. Asegurarse de configurar el autoescalado y supervisar el uso de recursos en tiempo real para evitar sobrecargas.

- **Problemas comunes**
  - **Seguridad:** Mantener los contenedores aislados adecuadamente del sistema operativo host mediante el uso de espacios de nombres y cgroups. Utilizar soluciones como kata Containers para mejorar el aislamiento de los contenedores.

- **Gestión de recursos:** Implementar límites y cuotas de recursos para cada contenedor en Kubernetes o Docker para evitar que un contenedor consuma más de lo esperado. Se debe configurar los límites en CPU, memoria y almacenamiento.
- **Depuración:** Desactivar el modo debug en entornos de producción. Este modo puede exponer información sensible y ser una puerta de entrada para cualquier atacante.
- **Sobrecarga de Red:** Implementar soluciones de monitoreo y auditoría continua para detectar actividades sospechosas o vulnerabilidades en los contenedores. Herramientas como Prometheus o Sysdig pueden alterar y proporcionar métricas de seguridad en tiempo real.

Conclusión

El uso de contenedores ha revolucionado el enfoque tradicional de desarrollo, pruebas y despliegue de aplicaciones al proporcionar un entorno consistente y aislado que elimina muchos de los problemas asociados con la variabilidad de los entornos de ejecución. A través de herramientas como Docker, Podman y Kubernetes, los desarrolladores pueden gestionar aplicaciones de manera más eficiente y fiable, asegurando que cada etapa del ciclo de vida del software se realice con la misma confiabilidad y previsibilidad.

En el contexto del ejemplo de la API para una aplicación de torneos y competiciones, los contenedores han demostrado ser una solución invaluable. Al encapsular la base de datos MySQL y la API en contenedores Docker, hemos conseguido simplificar la configuración del entorno de desarrollo y garantizar que el mismo entorno se reproduzca durante las pruebas y en producción. Este enfoque no solo ha facilitado la integración continua y la entrega continua (CI/CD) al permitir que las pruebas se realicen en un entorno controlado y replicable, sino que también ha optimizado el despliegue al garantizar que la aplicación funcione de manera consistente sin importar el entorno subyacente.

La implementación de Kubernetes para la orquestación de contenedores ha añadido una capa adicional de flexibilidad y escalabilidad, permitiendo gestionar y escalar la aplicación y su infraestructura de manera eficiente. La capacidad de Kubernetes para monitorizar, reiniciar y escalar contenedores automáticamente ha mejorado significativamente la fiabilidad y el rendimiento de la aplicación en entornos de producción.

En resumen, la adopción de contenedores en el desarrollo y despliegue de la API para nuestra aplicación de torneos ha resuelto problemas clave relacionados con la consistencia del entorno, la gestión de dependencias y la escalabilidad. Los beneficios de esta tecnología no solo han optimizado el ciclo de vida del software, sino que también han proporcionado una base sólida para la creación y gestión de aplicaciones complejas y dinámicas. A medida que el ecosistema de contenedores continúa evolucionando, es probable que su papel en la simplificación del desarrollo y la mejora de la eficiencia siga creciendo, ofreciendo aún más ventajas a los desarrolladores y equipos de operaciones.

Bibliografía

Lumigo. (s.f.). Containerized applications: Benefits, challenges, and best practices. *Lumigo*. https://lumigo.io/container-monitoring/containerized-applications-benefits-challenges-best-practices/

Simform. (2023, June 22). Containerization best practices: Expert guide. *Simform*. https://www.simform.com/blog/containerization-best-practices/

¿Qué son los contenedores? | IBM. (s. f.-b). https://www.ibm.com/es-es/topics/containers

Explicación sobre los contenedores: concepto e importancia. (s. f.). https://www.redhat.com/es/topics/containers

Vrapolinario. (2023, 31 marzo). Contenedores frente a máquinas virtuales. Microsoft Learn. https://learn.microsoft.com/es-es/virtualization/windowscontainers/about/containers-vs-vm

Tomás, E. (s. f.-b). *¿Docker o Podman?: similitudes, diferencias, ventajas e inconvenientes a la hora de manejar contenedores - campusMVP.es*. campusMVP.es. https://www.campusmvp.es/recursos/post/docker-o-podman-similitudes-diferencias-ventajas-e-inconvenientes-a-la-hora-de-manejar-contenedores.aspx

Atlassian. (s. f.-b). *Comparación entre Kubernetes y Docker | Atlassian*. https://www.atlassian.com/es/microservices/microservices-architecture/kubernetes-vs-docker#:~:text=Docker%20es%20una%20plataforma%20de,tiempos%20de%20ejecuci%C3%B3n%20de%20contenedores.

What is Podman? (s. f.-b). https://www.redhat.com/en/topics/containers/what-is-podman#why-podman

¿Qué es Docker y cómo funciona? Ventajas de los contenedores Docker. (s. f.). https://www.redhat.com/es/topics/containers/what-is-docker