**UNIVERSIDAD DON BOSCO**

**FACULTAD DE INGENIERÍA**

**ESCUELA DE INGENIERÍA EN COMPUTACIÓN**

**Interfaz de usuario gráfica

Descripción generada automáticamente**

**INGENIERA**:

Karens Medrano

**ACTIVIDAD**:

Investigación Aplicada 2

**ASIGNATURA**:

Diseño y Programación de Software Multiplataforma G01T

**FECHA DE ELABORACION**: 12/9/2024

**CICLO 02 / 2024**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **INTEGRANTES:** | **CARNET** | **Grupo Teorico** |
| 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 Mejia | LM151857 | 01T |
| 5 | Cristian Enrique Pineda Muñoz | PM190654 | 01T |

Contenido

[Introducción 3](#_Toc176173542)

[*1.* Definir qué son los contenedores: 4](#_Toc176173543)

[a) Explicar el concepto de contenedores en el contexto del desarrollo de software. 4](#_Toc176173544)

[b) b. Diferenciar entre contenedores y máquinas virtuales. 4](#_Toc176173545)

[*2.* Describir el funcionamiento de los contenedores: 5](#_Toc176173546)

[a) Analizar cómo los contenedores empaquetan aplicaciones y sus dependencias. 5](#_Toc176173547)

[b) Explicar el proceso de aislamiento y ejecución de contenedores en el sistema operativo. 5](#_Toc176173548)

[*3.* Explorar las ventajas de utilizar contenedores: 6](#_Toc176173549)

[a. Discutir los beneficios en términos de portabilidad, eficiencia y consistencia. 6](#_Toc176173550)

[b. Ejemplificar cómo los contenedores facilitan el desarrollo, pruebas y despliegue de aplicaciones. 6](#_Toc176173551)

[*4.* Presentar herramientas populares en el ecosistema de contenedores: 7](#_Toc176173552)

[a. Describir y comparar Docker, Kubernetes y otras herramientas relevantes. 7](#_Toc176173553)

[b. Mostrar casos de uso y ejemplos prácticos de cada herramienta. 7](#_Toc176173554)

[*5.* Discutir los desafíos y consideraciones al usar contenedores: 8](#_Toc176173555)

[a. Identificar posibles limitaciones y problemas comunes. 8](#_Toc176173556)

[b. Proponer soluciones y mejores prácticas para abordar estos desafíos 8](#_Toc176173557)

[Conclusión 9](#_Toc176173558)

[Bibliografía 9](#_Toc176173559)

# Introducción

# Definir qué son los contenedores:

## Explicar el concepto de contenedores en el contexto del desarrollo de software.

## b. Diferenciar entre contenedores y máquinas virtuales.

# Describir el funcionamiento de los contenedores:

## Analizar cómo los contenedores empaquetan aplicaciones y sus dependencias.

## 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.

# Explorar las ventajas de utilizar contenedores:

## 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 continua y despliegue continuo (CI/CD), mejorando la calidad y confiabilidad del software.

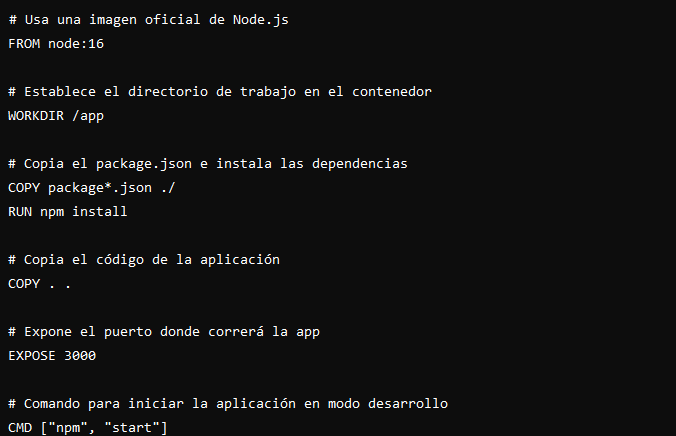
## Ejemplificar cómo los contenedores facilitan el desarrollo, pruebas y despliegue de aplicaciones.

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:



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:

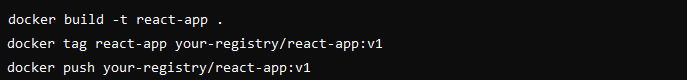


Al ejecutar docker run react-app, puedes asegurarte de que las pruebas se ejecuten en un entorno limpio y consistente. Incluso se puede integrar con sistemas de CI/CD (como Jenkins o GitLab CI) para ejecutar las pruebas automáticamente cada vez que se hace un commit.

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:



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.

# Presentar herramientas populares en el ecosistema de contenedores:

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

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

# Discutir los desafíos y consideraciones al usar contenedores:

## 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.

## 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

# 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/>