INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

TECNICATURA SUPERIOR EN

Telecomunicaciones

Proyecto Integrador I

Unidad 2

Practicas con BD

Dirección General de EDUCACIÓN TÉCNICA Y FORMACIÓN PROFESIONAL

Ministerio de EDUCACIÓN

CÓRDOBA Seguimos haciendo

ÍNDICE

Desarrollo de la Capa de Almacenamiento

Introducción

Este documento describe el enfoque y la implementación de la capa de almacenamiento dentro de nuestra arquitectura de cloud, centrada en el uso de Docker para contenerizar las bases de datos MongoDB y MySQL. Esta implementación proporciona una base robusta y flexible para manejar los datos generados y consumidos por las aplicaciones IoT (APIs).

Estructura Cloud Implementada

La estructura cloud está diseñada para ser modular y escalable, utilizando Docker como la tecnología central para la orquestación de contenedores. Esto permite un despliegue eficiente y manejable de los servicios de bases de datos, que son esenciales para la recopilación y análisis de datos en tiempo real provenientes de dispositivos IoT.

Componentes de la Estructura:

- **Docker Containers**: Contenedores para cada servicio de base de datos, permitiendo aislamiento y configuración independiente.
- **Docker Compose**: Utilizado para definir y ejecutar aplicaciones multi-contenedor, facilitando la configuración y el despliegue de nuestros servicios de MongoDB y MySQL.

Herramientas del Stack Tecnológico y Sus Ventajas

1. **MongoDB**:

- **Descripción**: Una base de datos NoSQL que ofrece alta flexibilidad y es ideal para manejar grandes volúmenes de datos desestructurados.
- **Ventajas**:
    - Esquemas dinámicos que facilitan la integración y adaptación a las necesidades cambiantes de datos IoT.
    - Escalabilidad horizontal, esencial para manejar el crecimiento exponencial de datos en aplicaciones IoT.

2. **MySQL**:
    - **Descripción**: Un sistema de gestión de base de datos relacional conocido por su confiabilidad y robustez.
    - **Ventajas**:
        - Estructuras de datos bien definidas y la integridad de los datos a través de transacciones ACID, proporcionando una base sólida para operaciones críticas.
        - Amplio soporte y documentación que facilita la implementación y el mantenimiento.
3. **Docker**:
    - **Descripción**: Plataforma de contenedorización que permite empaquetar y ejecutar aplicaciones en un entorno aislado.
    - **Ventajas**:
        - Consistencia entre los entornos de desarrollo, prueba y producción, eliminando el problema "funciona en mi máquina".

- Portabilidad y microservicios, apoyando la arquitectura de despliegues en cloud y facilitando la gestión de dependencias.

4. **Docker Compose**:
    - **Descripción**: Herramienta para definir y correr aplicaciones Docker multi-contenedor.
    - **Ventajas**:
        - Simplifica la configuración de múltiples contenedores, permitiendo una configuración declarativa y fácil de seguir.
        - Automatiza la orquestación de contenedores, haciendo más sencillos los despliegues y la escalabilidad.

Documentación y Control de Versiones

Utilizamos **Git** y **GitHub** para el control de versiones y la documentación del proyecto. Esto asegura que todos los cambios estén bien documentados y que el proceso de desarrollo sea transparente y colaborativo.

- **GitHub**: Repositorios para almacenar el código, las configuraciones de Docker y los scripts de la base de datos, así como para rastrear problemas y cambios.

Conclusión

La implementación de la capa de almacenamiento utilizando las tecnologías seleccionadas establece una fundación sólida para el manejo de datos en nuestro proyecto de IoT. Esta configuración no solo optimiza el rendimiento y la escalabilidad sino que también garantiza la flexibilidad necesaria para adaptarse a las necesidades cambiantes de las aplicaciones IoT.

Docker

1. Conceptos Básicos

Docker es una plataforma de software que permite la creación, prueba y despliegue de aplicaciones rápidamente a través de contenedores de software. Los contenedores permiten empaquetar una aplicación con todas sus dependencias en una unidad estándar para el desarrollo de software. Esto asegura que la aplicación funcione en cualquier otro sistema Linux, independientemente de las configuraciones personalizadas que podrían diferir de la máquina utilizada para escribir y probar el código.

2. Componentes Clave de Docker

- **Imágenes de Docker**: Una imagen de Docker es una plantilla de solo lectura que contiene un sistema operativo con aplicaciones y dependencias. La imagen se utiliza para crear contenedores.
- **Contenedores de Docker**: Los contenedores son instancias ejecutables de una imagen, que encapsulan un pedazo de software en un paquete completo que contiene todo lo necesario para que el software funcione.
- **Dockerfile**: Un archivo de texto que contiene todas las instrucciones necesarias para construir una imagen de Docker.
- **Docker Compose**: Una herramienta que permite definir y gestionar aplicaciones multi-contenedor con Docker.

3. Instalación y Configuración

- **Instalación de Docker**: Los estudiantes deben instalar Docker Desktop si utilizan Windows o Mac, o Docker Engine en Linux.

- **Verificación de la instalación**: Ejecutar **docker run hello-world** para verificar que Docker está correctamente instalado y funcionando.

Que puedo Dockerizar

Docker ofrece una gran flexibilidad y varias aplicaciones en el desarrollo y despliegue de software, pero tiene algunas limitaciones y especificidades que es importante entender:

1. **Cross-Platform Compatibility**: Docker se basa en características del kernel de Linux para aislar funcionalidades (namespaces, cgroups, etc.). En Windows, Docker utiliza una máquina virtual de Linux para proporcionar estas características. Esto significa que puedes ejecutar contenedores basados en Linux en un sistema Windows, pero esto se realiza dentro de un entorno virtualizado de Linux. Lo mismo se aplica de forma inversa. Sin embargo, no puedes ejecutar directamente un contenedor de Windows en un sistema Linux o viceversa sin una capa de emulación o virtualización específica.
2. **Usos de Docker**:
    - **Consistencia en el entorno de desarrollo, pruebas y producción**: Docker asegura que el software se ejecute de la misma manera en cualquier entorno, eliminando los problemas de "funciona en mi máquina".
    - **Microservicios**: Docker es ideal para desplegar microservicios ya que cada servicio puede ser encapsulado en su propio contenedor, gestionando sus dependencias de manera independiente.

- **Escalabilidad y Orquestación**: Docker se utiliza junto con plataformas de orquestación como Kubernetes para manejar el escalado automático y la gestión de múltiples contenedores.
- **CI/CD**: Integración y despliegue continuos. Los contenedores de Docker se pueden utilizar en pipelines de CI/CD para construir, probar y desplegar aplicaciones automáticamente con herramientas como Jenkins, GitLab y GitHub Actions.
- **Despliegue rápido de aplicaciones**: Docker reduce significativamente el tiempo de puesta en marcha de una aplicación ya que los contenedores pueden ser creados y destruidos en segundos.
- **Aislamiento de Aplicaciones**: Cada contenedor opera de forma aislada, lo que mejora la seguridad y reduce las interferencias entre aplicaciones.

3. **Dockerización de Aplicaciones y Servicios**:
    - **Aplicaciones web y APIs**: Puedes dockerizar aplicaciones backend y frontend, garantizando que funcionen en cualquier entorno sin problemas de dependencias.
    - **Bases de Datos**: Docker permite ejecutar instancias de bases de datos como MySQL, PostgreSQL, MongoDB, etc., de manera aislada y portable.
    - **Entornos de Desarrollo**: Puedes crear entornos de desarrollo completos dentro de Docker, lo cual facilita a nuevos desarrolladores iniciar rápidamente sin configurar todo su entorno localmente.

- **Herramientas de Análisis de Datos**: Herramientas como Jupyter Notebook, RStudio, y sistemas de big data como Apache Spark se pueden ejecutar en contenedores para facilitar la análisis de datos.
- **Aplicaciones de Escritorio y Herramientas Gráficas**: Aunque Docker está más orientado a aplicaciones de servidor, también es posible ejecutar aplicaciones de GUI usando X11 forwarding o soluciones como VNC.

En resumen, Docker proporciona una solución robusta y versátil para el desarrollo y despliegue de aplicaciones en una amplia gama de contextos, facilitando la gestión de dependencias y la coherencia entre diferentes entornos de ejecución.

Similitudes entre Docker y Máquinas Virtuales

1. **Aislamiento**: Tanto Docker como las VM proporcionan un entorno aislado para ejecutar aplicaciones, asegurando que las operaciones en un contenedor o VM no afecten a otras.
2. **Portabilidad**: Ambos permiten que las aplicaciones se empaqueten con sus dependencias, facilitando el despliegue en diferentes sistemas y entornos de nube.
3. **Control de Versiones y Replicabilidad**: Permiten versionar y replicar entornos de manera consistente, lo cual es esencial para pruebas y despliegues escalables.

Diferencias entre Docker y Máquinas Virtuales

1. **Tipo de Aislamiento**:

- **Docker**: Utiliza contenedores, que comparten el mismo kernel del sistema operativo subyacente pero aíslan los procesos de la aplicación. Esto resulta en un menor consumo de recursos, ya que no es necesario cargar un sistema operativo completo para cada contenedor.
- **Máquinas Virtuales**: Cada VM ejecuta su propio sistema operativo completo, incluyendo el kernel, sobre un hypervisor que gestiona la VM en el hardware físico. Esto conlleva un mayor uso de recursos debido a la necesidad de virtualizar hardware y ejecutar múltiples sistemas operativos completos.

2. **Rendimiento**:
    - **Docker**: Generalmente ofrece un mejor rendimiento porque hay menos sobrecarga; los contenedores inician más rápido y utilizan menos memoria y CPU en comparación con las VM.
    - **Máquinas Virtuales**: Pueden experimentar una latencia mayor y un uso más intensivo de recursos debido a la necesidad de emular hardware y ejecutar sistemas operativos completos.
3. **Compatibilidad de Sistema Operativo**:
    - **Docker**: Principalmente diseñado para aplicaciones basadas en Linux, aunque también se pueden ejecutar contenedores en Windows y MacOS mediante máquinas virtuales de Linux o capas de compatibilidad.
    - **Máquinas Virtuales**: Pueden ejecutar casi cualquier sistema operativo, independientemente del sistema operativo host, lo que ofrece una flexibilidad superior en términos de compatibilidad de software.

4. **Uso de Casos**:
    - **Docker**: Ideal para desarrollo de aplicaciones, microservicios, aplicaciones web, y cualquier entorno donde la eficiencia, la escalabilidad y la automatización de despliegues son cruciales.
    - **Máquinas Virtuales**: Adecuadas para situaciones que requieren una emulación completa del hardware o la ejecución de múltiples sistemas operativos diferentes, como en el caso de centros de datos, grandes entornos de servidor, y pruebas de software en varios sistemas operativos.

Conclusión

Docker no reemplaza completamente a las máquinas virtuales; más bien, complementa y ofrece una alternativa más eficiente y ligera para ciertos usos. La elección entre Docker y máquinas virtuales depende de los requisitos específicos de aislamiento, rendimiento, compatibilidad de sistema operativo y administración de recursos. En muchas arquitecturas modernas, se utilizan ambas tecnologías para aprovechar sus respectivas ventajas.

Uso de VS Code con Docker

Extensión de Docker para VS Code: Esta extensión facilita la gestión de imágenes, contenedores, volúmenes y redes. Permite a los usuarios construir, gestionar y desplegar contenedores desde el IDE.

Estructura Básica de un Proyecto con Docker

```
proyecto/
|
├── Dockerfile              # Instrucciones para construir la imagen del contenedor
├── docker-compose.yml      # Configuración para gestionar servicios relacionados
├── .dockerignore           # Especifica los archivos que no se deben copiar
└── src/                    # Código fuente del proyecto
    ├── api/
    ├── microservicios/
    └── appweb/
```

Ejercicios Clásicos para BDNR

Vamos a desarrollar dos ejercicios clásicos que demuestran el uso de MongoDB en escenarios de bases de datos no relacionales (BDNR). Estos ejercicios incluirán la modelización de los datos y la implementación utilizando MongoDB para reflejar situaciones del mundo real.

Ejercicio 1: Gestión de Contenido para un Blog

Situación del Mundo Real: Un sistema de gestión de contenido (CMS) para un blog que necesita manejar publicaciones, comentarios y autores.

Planteo del Ejercicio:

El sistema debe permitir:

- Crear publicaciones con título, contenido, autor y fecha de publicación.
- Añadir comentarios a las publicaciones, con referencia al autor del comentario y la fecha.
- Mantener un registro de autores con nombre y biografía.

Modelo de Datos No Relacional:

- **Publicaciones:** Documentos que contienen título, contenido, información del autor (referencia a Autores), y comentarios (lista de documentos embebidos).
- **Autores:** Documentos que contienen nombre y biografía.
- **Comentarios:** Documentos embebidos en Publicaciones con contenido, autor del comentario (referencia a Autores) y fecha.

Ejercicio 2: Sistema de Tickets para Soporte Técnico

Situación del Mundo Real: Un sistema para manejar tickets de soporte técnico que registra problemas de usuarios, asigna técnicos y sigue el progreso hasta la resolución.

Planteo del Ejercicio:

El sistema debe:

- Registrar tickets con detalles del problema, usuario afectado, técnico asignado, estado y historial de acciones.

- Actualizar el estado de los tickets y añadir acciones al historial.
- Mantener un registro de usuarios y técnicos.

Modelo de Datos No Relacional:

- **Tickets:** Documentos que contienen detalles del problema, **usuario_id**, **tecnico_id**, estado, y un historial de acciones (lista de documentos embebidos).
- **Usuarios:** Documentos que contienen nombre y contacto.
- **Técnicos:** Documentos similares a Usuarios pero con información adicional sobre sus habilidades.

Ejercicios Clásicos para BDR (SQL)

Ejemplo 1: Sistema de Gestión de Biblioteca

Situación del Mundo Real: Una biblioteca local necesita un sistema para gestionar sus libros, préstamos y miembros. El sistema debe permitir registrar nuevos libros, mantener información sobre los miembros y gestionar los préstamos.

Planteo del Ejercicio:

Se requiere un sistema que permita:

- Registrar libros con detalles como título, autor, ISBN y estado (disponible o prestado).
- Registrar miembros con información como nombre, dirección, teléfono y fecha de inscripción.
- Gestionar los préstamos de libros a los miembros, incluyendo fechas de préstamo y devolución.

Modelo Entidad-Relación (E-R):

Entidades:

1. **Libro:** Representa los libros en la biblioteca.
    - Atributos: ISBN (clave primaria), Título, Autor, Estado.
2. **Miembro:** Personas registradas para tomar libros prestados.
    - Atributos: ID Miembro (clave primaria), Nombre, Dirección, Teléfono.

3. **Préstamo:** Transacciones de préstamo de libros a los miembros.
    - Atributos: ID Préstamo (clave primaria), Fecha de préstamo, Fecha de devolución, ISBN, ID Miembro.

Relaciones:

- **Prestar:** Relaciona un libro con un miembro a través de la entidad Préstamo.

Modelo Relacional:

- **Libro**
    - ISBN (PK)
    - Título
    - Autor
    - Estado
- **Miembro**
    - ID Miembro (PK)
    - Nombre
    - Dirección
    - Teléfono
- **Préstamo**
    - ID Préstamo (PK)
    - Fecha de préstamo
    - Fecha de devolución
    - ISBN (FK a Libro)
    - ID Miembro (FK a Miembro)

Ejemplo 2: Sistema de Registro de Estudiantes y Cursos

Situación del Mundo Real: Una universidad desea implementar un sistema para gestionar estudiantes, cursos y las inscripciones de los estudiantes en los cursos.

Planteo del Ejercicio:

El sistema debe:

- Mantener información de los estudiantes como nombre, dirección y número de matrícula.

- Registrar cursos ofrecidos con detalles como código de curso, nombre y créditos.
- Gestionar inscripciones de estudiantes en cursos, incluyendo calificaciones finales.

Modelo Entidad-Relación (E-R):

Entidades:

1. **Estudiante:** Alumnos inscritos en la universidad.
    - Atributos: Número de Matrícula (clave primaria), Nombre, Dirección.
2. **Curso:** Cursos ofrecidos por la universidad.
    - Atributos: Código de Curso (clave primaria), Nombre, Créditos.
3. **Inscripción:** Registro de estudiantes en cursos específicos.
    - Atributos: Número de Matrícula, Código de Curso, Calificación Final.

Relaciones:

- **Inscribirse:** Relaciona estudiantes con cursos.

Modelo Relacional:

- **Estudiante**
    - Número de Matrícula (PK)
    - Nombre
    - Dirección
- **Curso**
    - Código de Curso (PK)
    - Nombre
    - Créditos
- **Inscripción**
    - Número de Matrícula (FK a Estudiante)
    - Código de Curso (FK a Curso)
    - Calificación Final

Referencias Bibliográficas

1. **"MongoDB: La guía definitiva"**, Kristina Chodorow, Edición 2013.
    - Una guía completa sobre MongoDB, abarcando desde conceptos básicos hasta aspectos avanzados del uso de esta base de datos NoSQL.
2. **"MySQL 5: Guía de aprendizaje"**, Michael Kofler, Edición 2015.
    - Una introducción exhaustiva a MySQL, ofreciendo desde la instalación y las primeras consultas hasta la optimización y la administración de la base de datos.
3. **"Docker: Up & Running"**, Sean P. Kane y Karl Matthias, Edición 2016.
    - Explica cómo lanzar, desarrollar y operar aplicaciones eficaces usando Docker.
4. **"El libro definitivo de Docker"**, James Turnbull, Edición 2015.
    - Cubre todo sobre Docker incluyendo Docker Compose, Docker Machine y orquestación de contenedores.
5. **"Git: De principiante a experto"**, Edgar Gutiérrez, Edición 2017.
    - Guía desde los fundamentos de Git hasta técnicas avanzadas de gestión de versiones para control de código.
6. **"Control de versiones con Git"**, Jon Loeliger y Matthew McCullough, Edición 2012.
    - Profundiza en el uso de Git para el control de versiones, ideal para entender la integración de Git con entornos de desarrollo modernos.
7. **"Microservicios prácticos con Docker y Kubernetes"**, Jaime Buelta, Edición 2019.
    - Un libro que introduce los conceptos de microservicios usando Docker y Kubernetes, perfecto para entender la arquitectura de aplicaciones modernas.
8. **"Sistemas de Bases de Datos"**, Thomas Connolly y Carolyn Begg, Edición 2015.
    - Un texto académico que aborda tanto bases de datos relacionales como algunas no relacionales, proporcionando una base sólida en teoría de bases de datos.