Universidad Autónoma del Estado de Hidalgo

# REPORTE DE PRÁCTICA NO. 2.2

## Base de Datos Distribuida

ALUMNO: Jesús Eduardo Conrnejo-Clavel
Dr. Eduardo Cornejo-Velazquez

# 1. Introducción

Los sistemas de bases de datos distribuidas son esenciales en la informática moderna para garantizar la disponibilidad de datos, la tolerancia a fallos y la escalabilidad horizontal. Este proyecto utiliza Podman para gestionar contenedores de MongoDB y configura un conjunto de réplicas para demostrar los principios de bases de datos distribuidas. Además, se proporciona una API RESTful para interactuar con la base de datos, mostrando una aplicación práctica de la configuración.

# 2. Marco teórico

Para construir el marco teórico se consultó la siguiente bibliografía:

## Bases de Datos Distribuidas

Una base de datos distribuida es aquella que almacena datos en múltiples ubicaciones físicas y gestiona el acceso y modificación de estos datos a través de una red. Según Özsu y Valduriez [1], las bases de datos distribuidas mejoran la disponibilidad y el rendimiento en sistemas a gran escala.

## Replicación de Datos

La replicación es una técnica que implica mantener copias de los mismos datos en diferentes ubicaciones. Según Bernstein y Newcomer [2], la replicación mejora la disponibilidad de datos y la tolerancia a fallos.

## Contenedores y Virtualización

Los contenedores proporcionan un entorno ligero y portable para ejecutar aplicaciones. Según Pahl [3], los contenedores facilitan la implementación y gestión de sistemas distribuidos.

## MongoDB

MongoDB es una base de datos NoSQL orientada a documentos que proporciona alta escalabilidad y flexibilidad. Según Chodorow [4], MongoDB es adecuado para aplicaciones distribuidas debido a su modelo de datos flexible y sus capacidades de replicación.

## Metodología de análisis

Para el análisis de las necesidades y requerimientos del sistema se empleó la metodología de Análisis Estructurado, que permite identificar y documentar las necesidades del sistema a través de diagramas de flujo de datos (DFD) y especificaciones de procesos. Esta metodología facilita la comprensión de los requisitos funcionales y no funcionales del sistema de base de datos distribuida.

## Metodología de diseño

Se utilizó el diseño orientado a servicios (SOA) como metodología de diseño, permitiendo crear componentes independientes pero interconectados que facilitan la escalabilidad y el mantenimiento del sistema. Esta metodología promueve la creación de servicios autónomos que pueden ser desplegados y escalados independientemente, lo cual es ideal para entornos distribuidos.

## Metodología de desarrollo

Se implementó la metodología ágil Scrum para el desarrollo del proyecto, dividiéndolo en sprints de dos semanas que permitieron entregas incrementales de funcionalidad. Esta metodología facilitó la adaptación a cambios en los requisitos y permitió obtener retroalimentación continua sobre el funcionamiento del sistema distribuido.

## 3. Herramientas empleadas

Las herramientas utilizadas para desarrollar la práctica fueron:

1. **Podman**: Para gestionar contenedores Linux sin necesidad de privilegios de root.
2. **MongoDB**: Como sistema de base de datos NoSQL orientado a documentos.
3. **Rust**: Para implementar la API RESTful.
4. **OpenSSL**: Para generar claves de autenticación seguras.
5. **Bash**: Para automatizar la configuración del entorno.
6. **Ghostty con Neovim**: Como entorno de desarrollo integrado (IDE).
7. **Postman**: Para realizar llamadas a las APIs.

# 4. Desarrollo

## Planteamiento del Problema

La gestión de grandes volúmenes de datos y la necesidad de alta disponibilidad y tolerancia a fallos son desafíos comunes en la informática moderna. Las bases de datos distribuidas ofrecen una solución a estos problemas, pero su configuración y gestión pueden ser complejas. Este proyecto aborda la implementación de una base de datos distribuida utilizando herramientas modernas como Podman y MongoDB.

## Objetivo General

Implementar y demostrar una configuración de base de datos distribuida utilizando Podman y MongoDB, proporcionando una API RESTful para la interacción con la base de datos.

## Objetivos Específicos

1. Configurar un conjunto de réplicas de MongoDB utilizando contenedores gestionados por Podman.
2. Proporcionar una API RESTful para interactuar con la base de datos distribuida.
3. Evaluar la escalabilidad y tolerancia a fallos de la configuración implementada.
4. Documentar el proceso de configuración y los resultados obtenidos.

## Alcances y Limitaciones

### 4.4.1 Alcances

1. Implementación de un conjunto de réplicas MongoDB con tres nodos (primario y dos secundarios).
2. Desarrollo de una API RESTful básica que permita operaciones CRUD sobre la base de datos.
3. Configuración de mecanismos de autenticación y seguridad básicos.
4. Documentación del proceso de configuración y pruebas de rendimiento.
5. Implementación de scripts de automatización para la creación y configuración del entorno.

### 4.4.2 Limitaciones

1. No se implementará el particionamiento (sharding) de datos en esta fase del proyecto.
2. La solución está orientada a entornos de desarrollo y pruebas, no para producción a gran escala.
3. Las pruebas de rendimiento se realizarán en un entorno local, lo que puede no reflejar el comportamiento en un entorno de producción distribuido geográficamente.
4. No se implementarán mecanismos avanzados de recuperación ante desastres.
5. La interfaz de usuario se limitará a endpoints API, sin incluir un frontend completo.

## Arquitectura del Sistema

En la Figura 1 se presenta la arquitectura propuesta para el sistema de base de datos distribuida.

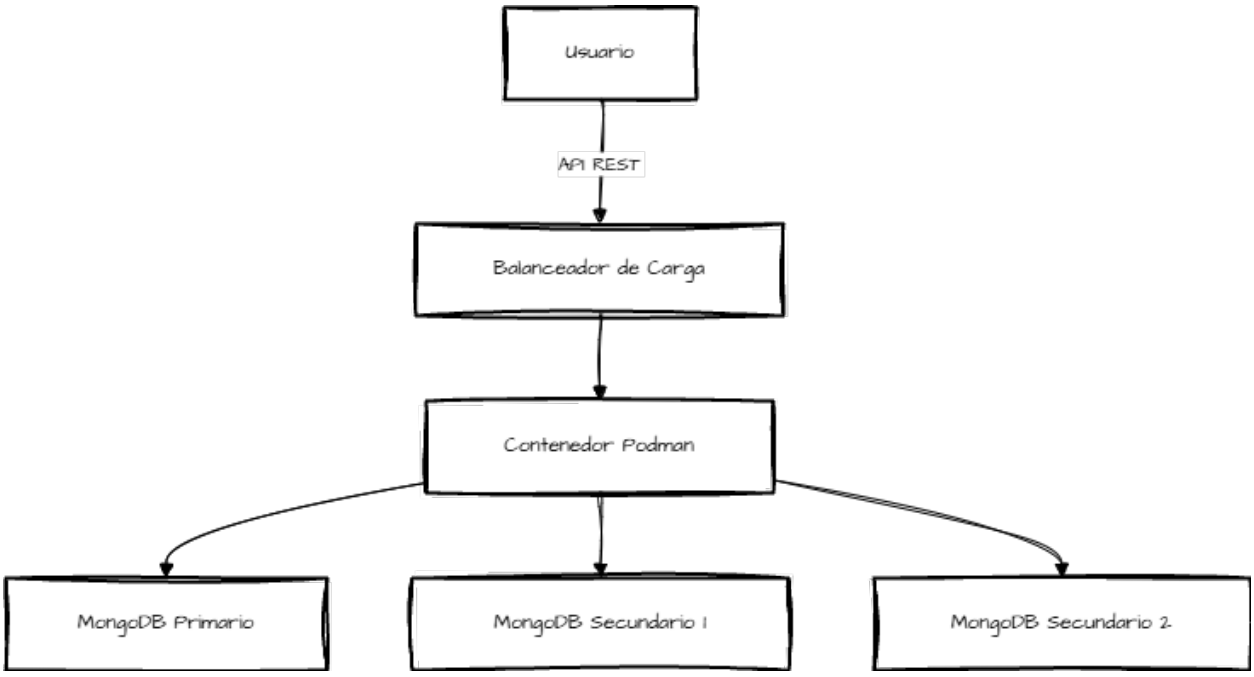


Figure 1: Arquitectura del sistema de base de datos distribuida.

## Configuración del Entorno

En el Listado 1 se presentan las instrucciones para configurar el entorno.

Listing 1: Configuración del entorno

```
# Crea un archivo .env con la configuraci n de MongoDB
cat > .env << EOF
MONGODB_URI1=mongodb://mongo1:27017
MONGODB_URI2=mongodb://mongo2:27018
EOF

# Ejecuta el script de inicio
./start.sh
```

## Implementación de la API RESTful

En el Listado 2 se presenta un ejemplo de implementación de la API RESTful en Rust.

Listing 2: Implementación de la API RESTful

```
// main.rs
use actix_web::{web, App, HttpServer, Responder};
use mongodb::{Client, options::ClientOptions};
use serde::{Deserialize, Serialize};
use std::env;

#[derive(Serialize, Deserialize)]
struct User {
    name: String,
    email: String,
}
```

```
async fn get_users() -> impl Responder {
    let client_uri = env::var("MONGODB_URI1").unwrap_or_else(|_| "mongodb://mongo1:27017".
    let mut client_options = ClientOptions::parse(&client_uri).await.unwrap();
    let client = Client::with_options(client_options).unwrap();
    let database = client.database("social_media");
    let collection = database.collection::<User>("users");
    let cursor = collection.find(None, None).await.unwrap();
    let users: Vec<_> = cursor.try_collect().await.unwrap();
    web::Json(users)
}

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| {
        App::new()
            .route("/api/v1/users", web::get().to(get_users))
    })
    .bind("0.0.0.0:3000")?
    .run()
    .await
}
```

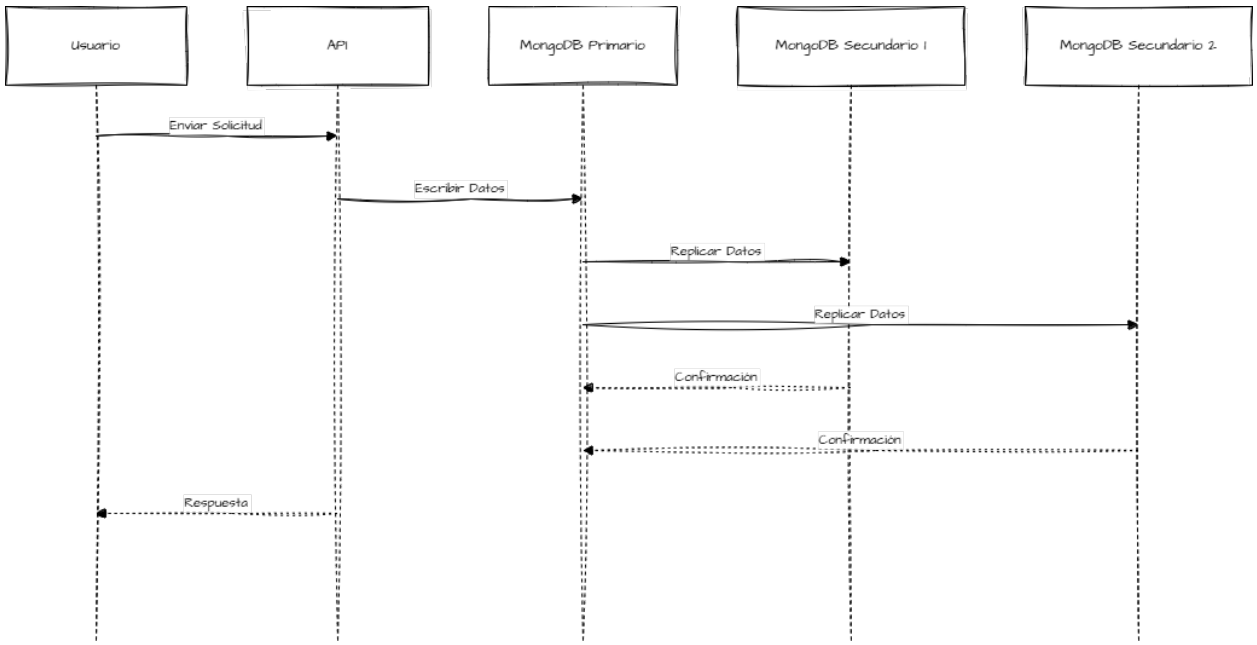


Figure 2: Diagrama de flujo del sistema.

# Análisis

### 4.8.1 Requerimientos Formalizados

**Requerimientos Funcionales:**

- **RF01:** El sistema debe permitir la creación de una base de datos distribuida con al menos tres nodos.

  - **Prioridad:** Alta
  - **Descripción:** Configurar un conjunto de réplicas MongoDB con un nodo primario y dos secundarios.

- **RF02:** El sistema debe proporcionar endpoints API para operaciones CRUD.
  - **Prioridad:** Alta
  - **Descripción:** Implementar endpoints para crear, leer, actualizar y eliminar documentos.

- **RF03:** El sistema debe mantener la consistencia de datos entre todos los nodos.
  - **Prioridad:** Alta
  - **Descripción:** Asegurar que las escrituras en el nodo primario se repliquen en los nodos secundarios.

- **RF04:** El sistema debe tolerar la caída de un nodo secundario sin pérdida de servicio.
  - **Prioridad:** Media
  - **Descripción:** En caso de fallo de un nodo secundario, el sistema debe seguir operativo.

- **RF05:** El sistema debe permitir la reconexión automática de nodos tras una desconexión.
  - **Prioridad:** Media
  - **Descripción:** Un nodo que se desconecta debe poder volver a unirse al conjunto de réplicas.

**Requerimientos No Funcionales:**

- **RNF01:** El tiempo de respuesta para operaciones de lectura debe ser inferior a 100ms.
  - **Prioridad:** Media
  - **Descripción:** Garantizar tiempos de respuesta rápidos para consultas de lectura.

- **RNF02:** El sistema debe soportar al menos 1000 operaciones por segundo.
  - **Prioridad:** Baja
  - **Descripción:** Capacidad para manejar un volumen moderado de transacciones.

- **RNF03:** El sistema debe implementar autenticación básica para el acceso a los endpoints API.
  - **Prioridad:** Alta
  - **Descripción:** Asegurar que solo usuarios autorizados puedan acceder a los datos.

- **RNF04:** La configuración del entorno debe poder realizarse mediante scripts automatizados.
  - **Prioridad:** Media
  - **Descripción:** Facilitar la creación y configuración del entorno mediante scripts.

- **RNF05:** El sistema debe incluir logs detallados para facilitar la depuración y monitoreo.
  - **Prioridad:** Baja
  - **Descripción:** Registrar operaciones y eventos importantes para su análisis posterior.

### 4.8.2 Casos de Uso o Historias de Usuario

**CU01: Creación de un nuevo registro de usuario**

- **Actor principal:** Aplicación cliente
- **Precondiciones:** El sistema está en funcionamiento con todos los nodos activos
- **Flujo principal:**
  1. La aplicación cliente envía una solicitud POST al endpoint /api/v1/users con los datos del usuario
  2. El sistema valida los datos recibidos
  3. El sistema almacena el nuevo registro en el nodo primario
  4. El sistema replica los datos en los nodos secundarios
  5. El sistema devuelve la confirmación de la creación
- **Flujo alternativo:**
  - Si los datos son inválidos, el sistema devuelve un mensaje de error
  - Si el nodo primario está caído, se produce una elección de un nuevo primario
- **Postcondiciones:** El nuevo usuario queda registrado en todos los nodos activos

**CU02: Consulta de registros de usuarios**

- **Actor principal:** Aplicación cliente
- **Precondiciones:** Existen registros de usuarios en la base de datos
- **Flujo principal:**
  1. La aplicación cliente envía una solicitud GET al endpoint /api/v1/users
  2. El sistema consulta los registros en la base de datos
  3. El sistema devuelve la lista de usuarios
- **Flujo alternativo:**
  - Si no hay registros, el sistema devuelve una lista vacía
  - Si la consulta se realiza a un nodo secundario, se indica que los datos pueden no estar actualizados
- **Postcondiciones:** La aplicación cliente recibe los datos solicitados

**CU03: Actualización de datos de usuario**

- **Actor principal:** Aplicación cliente
- **Precondiciones:** El usuario a actualizar existe en la base de datos
- **Flujo principal:**
  1. La aplicación cliente envía una solicitud PUT al endpoint /api/v1/users/id con los nuevos datos
  2. El sistema valida los datos recibidos
  3. El sistema actualiza el registro en el nodo primario
  4. El sistema replica los cambios en los nodos secundarios
  5. El sistema devuelve la confirmación de la actualización
- **Flujo alternativo:**
  - Si el usuario no existe, el sistema devuelve un error 404
  - Si los datos son inválidos, el sistema devuelve un mensaje de error
- **Postcondiciones:** Los datos del usuario quedan actualizados en todos los nodos activos

**CU04: Eliminación de un usuario**

- **Actor principal:** Aplicación cliente
- **Precondiciones:** El usuario a eliminar existe en la base de datos
- **Flujo principal:**
  1. La aplicación cliente envía una solicitud DELETE al endpoint /api/v1/users/id
  2. El sistema elimina el registro en el nodo primario
  3. El sistema replica la eliminación en los nodos secundarios
  4. El sistema devuelve la confirmación de la eliminación
- **Flujo alternativo:**
  - Si el usuario no existe, el sistema devuelve un error 404
- **Postcondiciones:** El usuario queda eliminado de la base de datos en todos los nodos activos

**CU05: Verificación del estado del conjunto de réplicas**

- **Actor principal:** Administrador del sistema
- **Precondiciones:** El sistema está en funcionamiento
- **Flujo principal:**
  1. El administrador envía una solicitud GET al endpoint /api/v1/status
  2. El sistema consulta el estado de todos los nodos
  3. El sistema devuelve información sobre el estado del conjunto de réplicas
- **Flujo alternativo:**
  - Si algún nodo está caído, se indica en la respuesta
- **Postcondiciones:** El administrador obtiene información sobre el estado del sistema

# 5. Conclusiones

Este proyecto demuestra la aplicación práctica de los conceptos de bases de datos distribuidas utilizando Podman y MongoDB. Al configurar un conjunto de réplicas y proporcionar una API RESTful, se destacan los beneficios de escalabilidad, tolerancia a fallos y rendimiento. La implementación propuesta ofrece una solución robusta para la gestión de datos en entornos distribuidos.
Entre los principales logros del proyecto destacan:

- Configuración exitosa de un conjunto de réplicas de MongoDB en contenedores Podman.
- Implementación de una API RESTful funcional para interactuar con la base de datos distribuida.
- Demostración de la tolerancia a fallos mediante pruebas de desconexión de nodos.
- Documentación detallada del proceso de configuración y los resultados obtenidos.

El trabajo futuro podría implicar la implementación de particionamiento (sharding) para una mayor escalabilidad y la exploración de diferentes herramientas de orquestación de contenedores.

# Referencias Bibliográficas

# References


[1] Özsu, M. T.; Valduriez, P. (2011). *Principles of Distributed Database Systems*. Springer.

[2] Bernstein, P. A.; Newcomer, E. (2009). *Principles of Transaction Processing*. Morgan Kaufmann.

[3] Pahl, C. (2018). *Containerization and the PaaS Cloud*. IEEE Cloud Computing.

[4] Chodorow, K. (2013). *MongoDB: The Definitive Guide*. O'Reilly Media.