# MÓDULO 1(anexos):

# Database: MongoDB y Redis

## Paso 1: Instalar MongoDB

### 1.1. Agregar repositorio y clave GPG de MongoDB:

```
# Instalar dependencias iniciales
sudo apt-get install -y wget gnupg

# Agregar clave GPG
wget -qO - https://www.mongodb.org/static/pgp/server-7.0.asc | sudo gpg --
dearmor -o /usr/share/keyrings/mongodb.gpg

# Crear archivo de repositorio
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/mongodb.gpg]
https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-7.0.list

# Actualizar repositorios
sudo apt-get update
```

### 1.2. Instalar MongoDB:

```
sudo apt-get install -y mongodb-org

# Iniciar y habilitar el servicio
sudo systemctl enable mongod --now  # <-- ¡Nota el nombre correcto del servicio:
"mongod"!
```

---

## Paso 2: Instalar Redis

El error original no aplica para Redis, pero así debe instalarse:

```
sudo apt-get install -y redis-server
sudo systemctl enable redis-server --now
```

---

## Verificar Servicios

```
# MongoDB
sudo systemctl status mongod  # Debe mostrar "active (running)"

# Redis
sudo systemctl status redis-server  # Debe mostrar "active (running)"
```

---

## Solución de Errores Comunes

**Si MongoDB no inicia:**

1.Verifica los logs:

```
journalctl -u mongod -b
```

2.Asegúrate de que el puerto 27017 esté libre:

```
sudo lsof -i :27017
```

**Si Redis no inicia:**

1.Revisa la configuración:

```
sudo nano /etc/redis/redis.conf
```

2.Reinicia el servicio:

```
sudo systemctl restart redis-server
```

**TEST**

**Aquí tienes una implementación de la aplicación Flask con microservicios y pruebas de rendimiento:**

```
from flask import Flask, jsonify, request
from flask_cors import CORS
import pymongo
import redis
import time
import os

app = Flask(__name__)
CORS(app)

# Conexión a MongoDB
mongo_client = pymongo.MongoClient("mongodb://localhost:27017/")
mongo_db = mongo_client["microservices_db"]
users_collection = mongo_db["users"]

# Conexión a Redis
redis_client = redis.Redis(host='localhost', port=6379, db=0)

# Helpers
def mongo_user_exists(user_id):
    return users_collection.find_one({"_id": user_id}) is not None

# Rutas para MongoDB
@app.route('/mongo/user', methods=['POST'])
def create_mongo_user():
    data = request.json
    result = users_collection.insert_one(data)
    return jsonify({"_id": str(result.inserted_id)}), 201

@app.route('/mongo/user/<user_id>', methods=['GET'])
def get_mongo_user(user_id):
    user = users_collection.find_one({"_id": user_id})
    if user:
        user['_id'] = str(user['_id'])
        return jsonify(user)
    return jsonify({"error": "Usuario no encontrado"}), 404

@app.route('/mongo/user/<user_id>', methods=['PUT'])
def update_mongo_user(user_id):
    data = request.json
    result = users_collection.update_one({"_id": user_id}, {"$set": data})
    if result.modified_count:
        return jsonify({"message": "Usuario actualizado"})
    return jsonify({"error": "Usuario no encontrado"}), 404

@app.route('/mongo/user/<user_id>', methods=['DELETE'])
def delete_mongo_user(user_id):
    result = users_collection.delete_one({"_id": user_id})
    if result.deleted_count:
        return jsonify({"message": "Usuario eliminado"})
    return jsonify({"error": "Usuario no encontrado"}), 404
```

```python
# Rutas para Redis
@app.route('/redis/user/<user_id>', methods=['POST'])
def create_redis_user(user_id):
    data = request.json
    redis_client.hset(user_id, mapping=data)
    return jsonify({"message": "Usuario creado", "user_id": user_id}), 201

@app.route('/redis/user/<user_id>', methods=['GET'])
def get_redis_user(user_id):
    user = redis_client.hgetall(user_id)
    if user:
        return jsonify({k.decode(): v.decode() for k, v in user.items()})
    return jsonify({"error": "Usuario no encontrado"}), 404

@app.route('/redis/user/<user_id>', methods=['PUT'])
def update_redis_user(user_id):
    data = request.json
    if not redis_client.exists(user_id):
        return jsonify({"error": "Usuario no encontrado"}), 404

    for key, value in data.items():
        redis_client.hset(user_id, key, value)
    return jsonify({"message": "Usuario actualizado"})

@app.route('/redis/user/<user_id>', methods=['DELETE'])
def delete_redis_user(user_id):
    if redis_client.delete(user_id):
        return jsonify({"message": "Usuario eliminado"})
    return jsonify({"error": "Usuario no encontrado"}), 404

# Ruta de pruebas de rendimiento
@app.route('/test', methods=['GET'])
def run_performance_tests():
    # Configuración de prueba
    num_operations = 100
    test_results = {}

    # Limpiar datos anteriores
    users_collection.delete_many({})
    redis_client.flushdb()

    # Pruebas para MongoDB
    mongo_times = {}

    # Operación de inserción
    start_time = time.time()
    for i in range(num_operations):
        users_collection.insert_one({
            "name": f"User {i}",
            "email": f"user{i}@test.com",
            "age": i % 100
        })
    mongo_times['insert'] = time.time() - start_time

    # Operación de lectura
    start_time = time.time()
    for i in range(num_operations):
```

```
        users_collection.find_one({"email": f"user{i}@test.com"})
    mongo_times['read'] = time.time() - start_time

    # Operación de actualización
    start_time = time.time()
    for i in range(num_operations):
        users_collection.update_one(
            {"email": f"user{i}@test.com"},
            {"$set": {"age": (i % 100) + 1}}
        )
    mongo_times['update'] = time.time() - start_time

    # Operación de eliminación
    start_time = time.time()
    for i in range(num_operations):
        users_collection.delete_one({"email": f"user{i}@test.com"})
    mongo_times['delete'] = time.time() - start_time

    # Pruebas para Redis
    redis_times = {}

    # Operación de inserción
    start_time = time.time()
    for i in range(num_operations):
        redis_client.hset(f"user_{i}", mapping={
            "name": f"User {i}",
            "email": f"user{i}@test.com",
            "age": i % 100
        })
    redis_times['insert'] = time.time() - start_time

    # Operación de lectura
    start_time = time.time()
    for i in range(num_operations):
        redis_client.hgetall(f"user_{i}")
    redis_times['read'] = time.time() - start_time

    # Operación de actualización
    start_time = time.time()
    for i in range(num_operations):
        redis_client.hset(f"user_{i}", "age", (i % 100) + 1)
    redis_times['update'] = time.time() - start_time

    # Operación de eliminación
    start_time = time.time()
    for i in range(num_operations):
        redis_client.delete(f"user_{i}")
    redis_times['delete'] = time.time() - start_time

    # Calcular resultados
    test_results = {
        "mongodb": {op: f"{time:.4f}s" for op, time in mongo_times.items()},
        "redis": {op: f"{time:.4f}s" for op, time in redis_times.items()},
        "observaciones": [
            "Redis muestra mejores tiempos en operaciones CRUD debido a su
naturaleza en memoria",
            "MongoDB ofrece mayor flexibilidad en consultas complejas a cambio
de un pequeño costo en rendimiento",
```

```
            "Las bases de datos estructuradas (SQL) típicamente tendrían tiempos
más altos en inserciones/actualizaciones debido a la validación de esquema y
ACID"
        ]
    }

    return jsonify(test_results)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

## Instrucciones para ejecutar la aplicación:

1. **Requisitos previos:**
   - Instalar MongoDB y Redis
   - Ejecutar ambos servicios en sus puertos predeterminados
2. **Instalar dependencias:**

```
pip install flask pymongo redis flask-cors
```

3. **Ejecutar la aplicación:**

```
python app.py
```

## Ejercicio de comprobación:

1. **Pruebas básicas de funcionalidad:**
   - Usar herramientas como Postman o curl para probar los endpoints CRUD
   - Ejemplo para MongoDB:

```
curl -X POST -H "Content-Type: application/json" -d
'{"_id":"1","name":"John","email":"john@test.com","age":30}'
http://localhost:5000/mongo/user
```

   - Ejemplo para Redis:

```
curl -X POST -H "Content-Type: application/json" -d
'{"name":"Alice","email":"alice@test.com","age":25}'
http://localhost:5000/redis/user/user_1
```

2. **Prueba de rendimiento:**
   - Ejecutar:

```
curl http://localhost:5000/test
```

   - Analizar los resultados comparativos entre MongoDB y Redis

## Análisis esperado:

- **Redis** mostrará tiempos significativamente menores en todas las operaciones debido a:
    - Almacenamiento en memoria
    - Estructura de datos simple
    - Ausencia de overhead de validación de esquema
- **MongoDB** tendrá tiempos mayores pero:
    - Ofrece mayor capacidad de consultas complejas
    - Persistencia duradera por defecto
    - Escalabilidad horizontal
- **Bases de datos estructuradas (SQL):**
    - Tiempos típicamente mayores que ambos
    - Ventajas en consistencia ACID y relaciones complejas
    - Esquema rígido proporciona integridad de datos

Este ejercicio demuestra los casos de uso típicos para cada tecnología, ayudando a elegir la adecuada según los requisitos específicos de cada proyecto.