INFORME DE GUÍA PRÁCTICA

I. PORTADA

Tema:	Taller de docker
Unidad de Organización Curricular:	PROFESIONAL
Nivel y Paralelo:	Quinto - A
Alumnos participantes:	Cholota Guamán Carlos Sebastián Mazabanda Pilamunga Diego Abraham Tixilema Puaquiza Kevin Alexander Tubon Chipantiza Danilo Alexander
Asignatura:	Sistemas de Bases de Datos Distribuidos
Docente:	Ing. José Rubén Caiza, Mg.

II. INFORME DE GUÍA PRÁCTICA

2.1 Objetivos

General:
Implementar un sistema de base de datos distribuida aplicando el concepto de sharding en MongoDB y la replicación en PostgreSQL.

Específicos:
- Configurar entorno Docker con múltiples contenedores MongoDB
- Implementar arquitectura Replica Set con 3 nodos
- Ejecutar consultas avanzadas con Aggregation Pipeline
- Probar resiliencia del sistema ante fallos de nodos
- Analizar resultados de distribución y replicación de datos

2.2 Modalidad
Presencial

2.3 Tiempo de duración
Presenciales: 11
No presenciales: 0

2.3.1 Instrucciones

- Instalar Docker Desktop y verificar funcionamiento
- Crear estructura de directorios y archivos de configuración
- Configurar docker-compose.yml con 3 nodos MongoDB
- Inicializar el Replica Set y verificar estado
- Importar datos de prueba en las colecciones
- Ejecutar consultas con Aggregation Pipeline
- Probar resiliencia del sistema ante caídas de nodos
- Verificar recuperación automática del cluster

2.4 Listado de equipos, materiales y recursos

Listado de equipos y materiales generales empleados en la guía práctica:
- **Computadora.**
- **Aula virtual/internet**

TAC (Tecnologías para el Aprendizaje y Conocimiento) empleados en la guía práctica:
☐Plataformas educativas

☐Simuladores y laboratorios virtuales
☐Aplicaciones educativas
☐Recursos audiovisuales
☐Gamificación
☒Inteligencia Artificial
Otros (Especifique): _____

2.5 Actividades por desarrollar

Configurar y administrar un clúster MongoDB con Replica Set de 3 nodos utilizando Docker, implementando consultas avanzadas con Aggregation Pipeline y probando la tolerancia a fallos del sistema distribuido mediante simulaciones de caída y recuperación de nodos.

2.6 Resultados obtenidos

Directorio de trabajo: `C:\TallerDocker`

FASE 1: CONFIGURACIÓN DEL ENTORNO

1.1: Abrir terminal y crear estructura

- Abrir cmd o PowerShell
- Ingresar al directorio del trabajo **"`cd C:\TallerDocker`"**

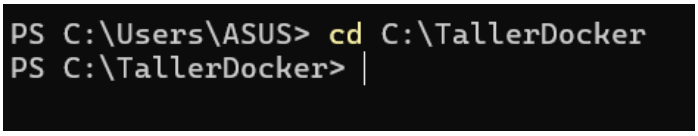

Ilustración 1: terminal_powershell.png

1.2: Crear archivo "departamentos.json"

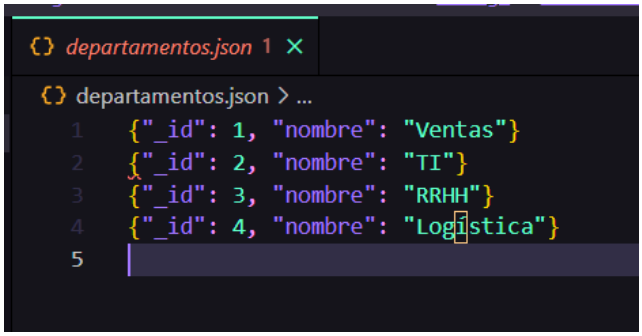

Ilustración 2: archivo_departamentos_json.png

1.3: Crear archivo “empleados.json”

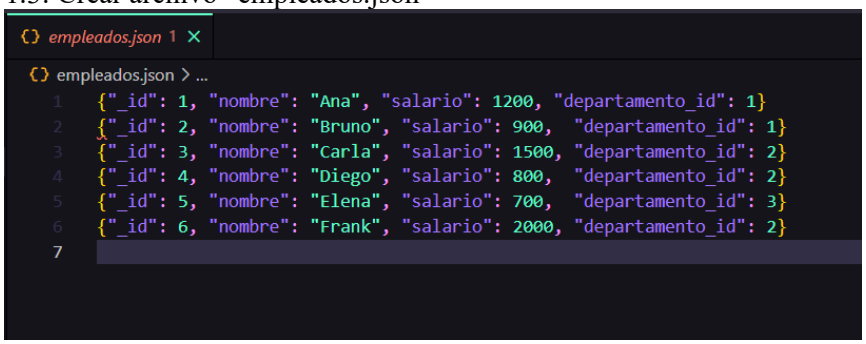

Ilustración 3: archivo_empleados_json.png

1.4: Crear archivo ventas.json

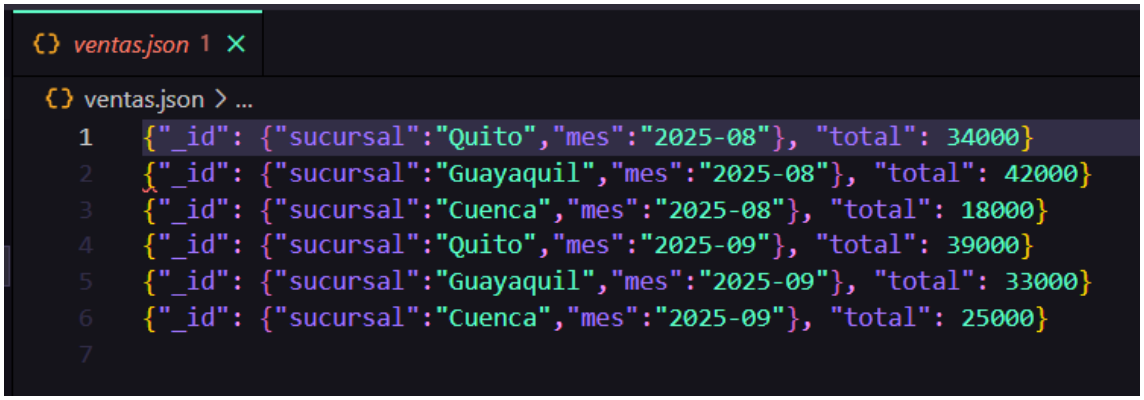

Ilustración 4: archivo_ventas_json.png

1.5: Crear archivo docker-compose.yml

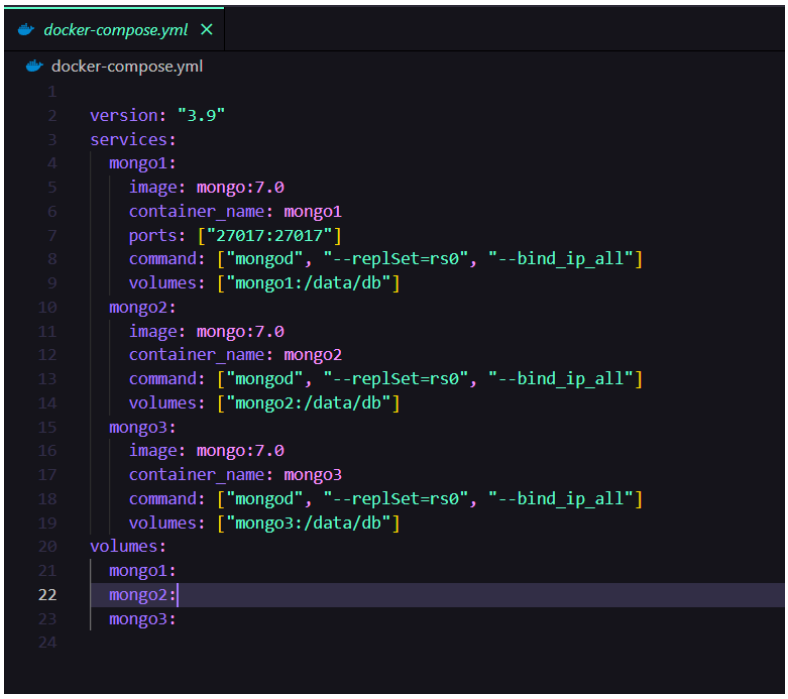

Ilustración 5: docker_compose_yml.png

1.6: Verificar archivos creados

En tu terminal, ejecutar: “dir”

```
PS C:\TallerDocker> dir


    Directorio: C:\TallerDocker


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         5/10/2025     22:07            126 departamentos.json
-a----         5/10/2025     21:18            562 docker-compose.yml
-a----         5/10/2025     22:07            418 empleados.json
-a----         5/10/2025     22:16            392 ventas.json


PS C:\TallerDocker>
```

Ilustración 6: directorios_creados.png

1.7: Levantar los contenedores Docker

- Asegúrate que Docker Desktop esté corriendo
- En la terminal, ejecuta: `docker compose up -d.` Esto descargará las imágenes de MongoDB

```
PS C:\TallerDocker> docker compose up -d
time="2025-10-05T22:27:39-05:00" level=warning msg="C:\\TallerDocker\\docker
-compose.yml: the attribute `version` is obsolete, it will be ignored, pleas
e remove it to avoid potential confusion"
[+] Running 3/3
 ✔Container mongo1  Started                                          0.9s
 ✔Container mongo2  Started                                          0.9s
 ✔Container mongo3  Started                                          0.8s
PS C:\TallerDocker>
```

Ilustración 7: descargar_imágenes_MongoDB.png

1.8: Verificar contenedores corriendo

Se visualiza 3 contenedores: mongo1, mongo2, mongo3

```
PS C:\TallerDocker> docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS                                                  NAMES
0a789caf852f   mongo:7.0   "docker-entrypoint.s…"   47 seconds ago   Up 46 seconds   0.0.0.0:27017->27017/tcp, [::]:27017->27017/tcp   mongo1
9588c817e55a   mongo:7.0   "docker-entrypoint.s…"   47 seconds ago   Up 47 seconds   27017/tcp                                              mongo3
51460b00124f   mongo:7.0   "docker-entrypoint.s…"   47 seconds ago   Up 47 seconds   27017/tcp                                              mongo2
PS C:\TallerDocker>
```

Ilustración 8: contenedores_ejecutandose.png

1.9: Inicializar el Replica Set

Ejecuta este comando:

```
docker exec -it mongo1 mongosh --eval "rs.initiate({_id: 'rs0',
members: [{_id: 0, host: 'mongo1:27017'},{_id: 1, host:
'mongo2:27017'},{_id: 2, host: 'mongo3:27017'}]})"
```

```
PS C:\TallerDocker> docker exec -it mongo1 mongosh --eval "rs.initiate({_id: 'rs0', members: [{_id: 0, host: 'mongo1:27017'},{_id: 1, host: 'mongo2:27017'},
{_id: 2, host: 'mongo3:27017'}]})"
{ ok: 1 }
PS C:\TallerDocker>
```

Ilustración 9: inicio_replica_set.png

1.10: Verificar estado del Replica Set

Luego ejecuta: `docker exec -it mongo1 mongosh --eval "rs.status()"`

Esto mostrara los estados PRIMARY y SECONDARY

Ilustrción 10: replica_set_status_opción1.png

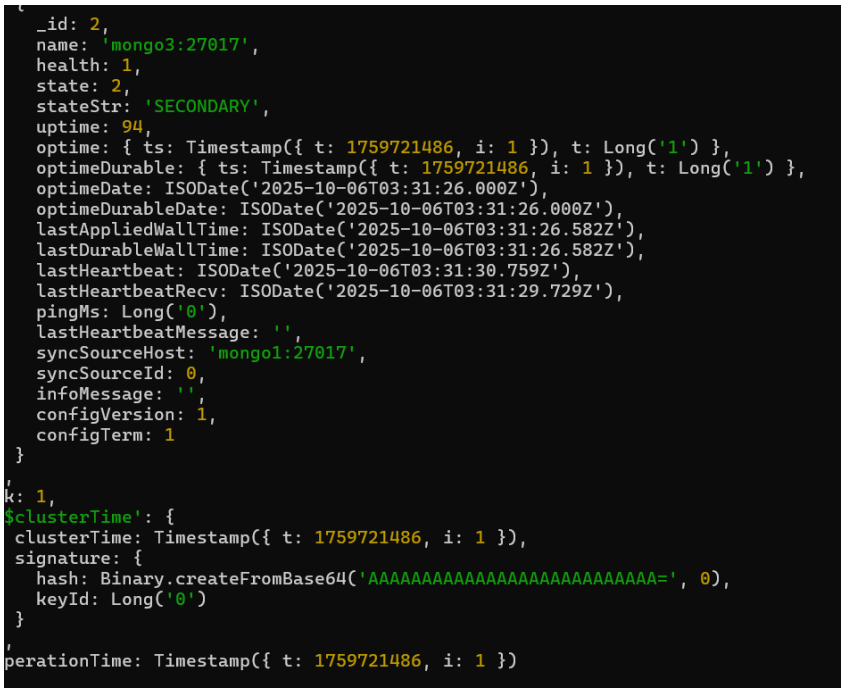

Ilustración 11: replica_set_status_opción1.png

O: `docker exec -it mongo1 mongosh --eval "rs.status()" | findstr "stateStr"`

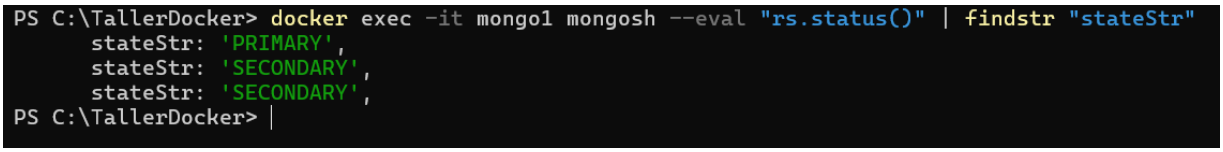

Ilustración 12: replica_set_status_opción2.png

1.11: Importar datos

- Importar departamentos
- Importar empleados
- Importar ventas

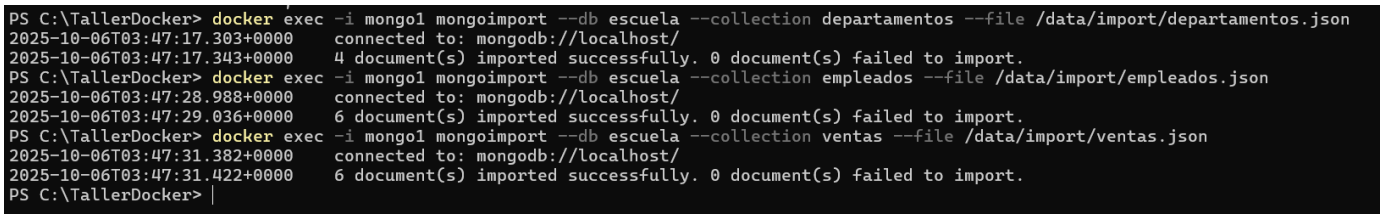

Ilustración 13: importacion_datos.png

1.12: Verificar datos importados

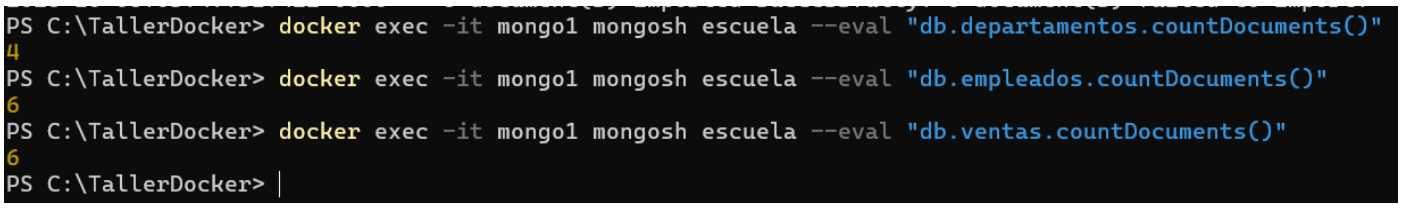

Ilustración 14: verificación_importacion_datos.png

FASE 2: CONSULTAS CON AGGREGATION PIPELINE

2.1: Crear el archivo consultas.md

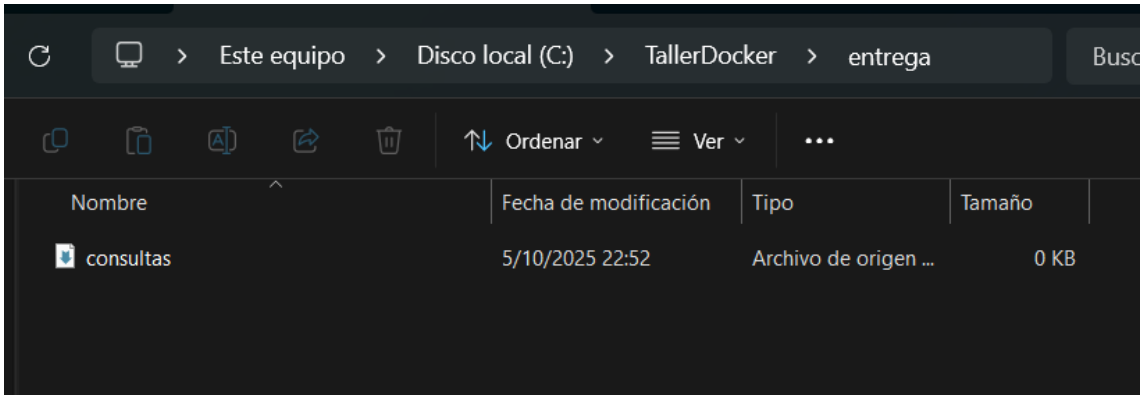

Ilustración 15: archivo_consultas_md.png

2.2: Conectarte a MongoDB

```
docker exec -it mongo1 mongosh escuela
```

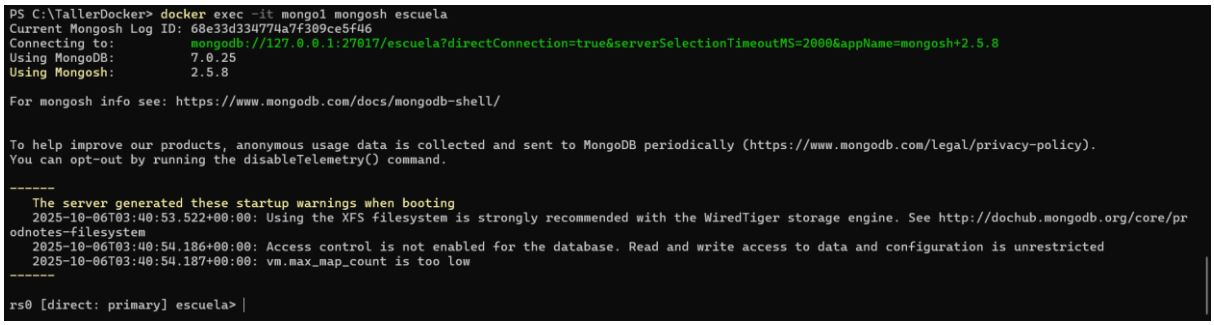

Ilustración 16: conección_mongo1_escuela.png

2.3Contenido de las consultas

CONSULTA 1: Empleados con salario mayor al promedio de la empresa
Método 1: Con $setWindowFields (MongoDB ≥5)

Explicación:

- Usamos $setWindowFields para calcular el promedio salarial de toda la empresa
- Este promedio se agrega como un campo a cada documento
- Luego filtramos con $match solo los empleados cuyo salario supera ese promedio
- Finalmente proyectamos los campos relevantes incluyendo la diferencia con el promedio

```
[
  {
    _id: 1,
    nombre: 'Ana',
    salario: 1200,
    promedioEmpresa: 1183.33,
    diferenciaConPromedio: 16.67
  },
  {
    _id: 3,
    nombre: 'Carla',
    salario: 1500,
    promedioEmpresa: 1183.33,
    diferenciaConPromedio: 316.67
  },
  {
    _id: 6,
    nombre: 'Frank',
    salario: 2000,
    promedioEmpresa: 1183.33,
    diferenciaConPromedio: 816.67
  }
]
rs0 [direct: primary] escuela>
```

Ilustración 17: consulta1_salario_mayor_promedio.png

Método 2: Sin ventanas (usando $facet)

Explicación:

- Usamos $facet para ejecutar dos pipelines en paralelo
- Un pipeline calcula el promedio general
- Otro pipeline mantiene todos los empleados
- Luego combinamos ambos resultados y filtramos

```
[
  {
    _id: 1,
    nombre: 'Ana',
    salario: 1200,
    promedioEmpresa: 1183.33,
    diferenciaConPromedio: 16.67
  },
  {
    _id: 3,
    nombre: 'Carla',
    salario: 1500,
    promedioEmpresa: 1183.33,
    diferenciaConPromedio: 316.67
  },
  {
    _id: 6,
    nombre: 'Frank',
    salario: 2000,
    promedioEmpresa: 1183.33,
    diferenciaConPromedio: 816.67
  }
]
rs0 [direct: primary] escuela>
```

Ilustración 18: : consulta1_salario_mayor_promedio_metodo2.png

CONSULTA 2: Departamentos sin empleados asignados

Explicación:

- Usamos $lookup para hacer un "join" entre departamentos y empleados
- El resultado del lookup es un array con los empleados de cada departamento
- Agregamos un campo que cuenta el tamaño de ese array
- Filtramos solo los departamentos donde el conteo es 0 (sin empleados)

```
rs0 [direct: primary] escuela> db.departamentos.aggregate([
...   {
...     $lookup: {
...       from: "empleados",
...       localField: "_id",
...       foreignField: "departamento_id",
...       as: "empleados"
...     }
...   },
...   {
...     $addFields: {
...       cantidadEmpleados: { $size: "$empleados" }
...     }
...   },
...   {
...     $match: {
...       cantidadEmpleados: 0
...     }
...   },
...   {
...     $project: {
...       _id: 1,
...       nombre: 1,
...       cantidadEmpleados: 1
...     }
...   }
... ])
[ { _id: 4, nombre: 'Logística', cantidadEmpleados: 0 } ]
rs0 [direct: primary] escuela>
```

Ilustración 19: consulta2_departamento_sin_empleados.png

CONSULTA 3: Empleado con salario más alto

Explicación:

- Ordenamos todos los empleados por salario de forma descendente
- Tomamos solo el primer resultado usando $limit
- Este será el empleado con el salario más alto

```
rs0 [direct: primary] escuela> db.empleados.aggregate([
...   { $sort: { salario: -1 } },
...   { $limit: 1 },
...   {
...     $project: {
...       nombre: 1,
...       salario: 1,
...       departamento_id: 1
...     }
...   }
... ])
[ { _id: 6, nombre: 'Frank', salario: 2000, departamento_id: 2 } ]
rs0 [direct: primary] escuela>
```

Ilustración 20:: consulta3_salario_mayor.png

CONSULTA 4: Para cada empleado, mostrar el salario promedio de su departamento

Explicación:

- Usamos $setWindowFields con partitionBy para agrupar por departamento
- Calculamos el promedio salarial dentro de cada partición (departamento)
- Cada empleado tendrá el promedio de su departamento como un campo adicional
- También calculamos la diferencia entre su salario y el promedio de su departamento


```
[
  {
    _id: 1,
    nombre: 'Ana',
    salario: 1200,
    departamento_id: 1,
    promedioDepartamento: 1050,
    diferenciaConPromedioDep: 150
  },
  {
    _id: 2,
    nombre: 'Bruno',
    salario: 900,
    departamento_id: 1,
    promedioDepartamento: 1050,
    diferenciaConPromedioDep: -150
  },
  {
    _id: 6,
    nombre: 'Frank',
    salario: 2000,
    departamento_id: 2,
    promedioDepartamento: 1433.33,
    diferenciaConPromedioDep: 566.67
  },
  {
    _id: 3,
    nombre: 'Carla',
    salario: 1500,
    departamento_id: 2,
    promedioDepartamento: 1433.33,
    diferenciaConPromedioDep: 66.67
  },
  {
    _id: 4,
    nombre: 'Diego',
    salario: 800,
    departamento_id: 2,
    promedioDepartamento: 1433.33,
    diferenciaConPromedioDep: -633.33
  },
  {
    _id: 5,
    nombre: 'Elena',
    salario: 700,
    departamento_id: 3,
    promedioDepartamento: 700,
    diferenciaConPromedioDep: 0
  }
]
rs0 [direct: primary] escuela>
```


Ilustración 21: consulta4_empleado_salario_promedio.png

CONSULTA 5: Departamentos cuyo promedio salarial es mayor al promedio general

Explicación:

- Usamos $facet para calcular dos cosas en paralelo:
    1. El promedio general de toda la empresa
    2. El promedio por cada departamento
- Luego comparamos ambos promedios
- Filtramos solo los departamentos cuyo promedio supera al promedio general
- Hacemos un lookup para obtener el nombre del departamento

```
[
  {
    _id: 2,
    nombreDepartamento: 'TI',
    promedioDepartamento: 1433.33,
    promedioGeneral: 1183.33,
    diferencia: 250
  }
]
rs0 [direct: primary] escuela>
```

Ilustración 22: consulta5_departamento_promedio_mayor_general.png

CONSULTA 6: Sucursal "top" por mes (sucursal con mayor total de ventas)

Explicación:

- Ordenamos las ventas por mes y total descendente
- Agrupamos por mes usando $group
- Usamos $first para tomar la primera sucursal de cada grupo (la de mayor venta)
- El resultado muestra para cada mes cuál sucursal tuvo las mayores ventas

```
rs0 [direct: primary] escuela> db.ventas.aggregate([
...   {
...     $sort: { "_id.mes": 1, "total": -1 }
...   },
...   {
...     $group: {
...       _id: "$_id.mes",
...       sucursalTop: { $first: "$_id.sucursal" },
...       totalMaximo: { $first: "$total" }
...     }
...   },
...   {
...     $project: {
...       _id: 0,
...       mes: "$_id",
...       sucursalTop: 1,
...       totalMaximo: 1
...     }
...   },
...   { $sort: { mes: 1 } }
... ])
[
  { sucursalTop: 'Guayaquil', totalMaximo: 42000, mes: '2025-08' },
  { sucursalTop: 'Quito', totalMaximo: 39000, mes: '2025-09' }
]
rs0 [direct: primary] escuela>
```

Ilustración 23: consulta6_sucursal_mayor_ventas.png

FASE 3: PRUEBA DE RESILIENCIA

Ahora vamos a probar qué pasa cuando **un nodo se cae** en un sistema distribuido.

3.1: Verificar estado inicial del Replica Set

Antes de romper nada, verificamos que todo esté funcionando: `docker exec -it mongo1 mongosh --eval "rs.status()" | findstr "stateStr"`

```
PS C:\TallerDocker> docker exec -it mongo1 mongosh --eval "rs.status()" | findstr "stateStr"
      stateStr: 'PRIMARY',
      stateStr: 'SECONDARY',
      stateStr: 'SECONDARY',
PS C:\TallerDocker>
```

Ilustración 24: estado_inicial_repStatus.png

3.2: Apagar un nodo (mongo3)

Simulamos que un servidor se cae: `docker stop mongo3`

```
PS C:\TallerDocker> docker stop mongo3
mongo3
PS C:\TallerDocker>
```

Ilustración 25: simular_caída_servidor.png

3.3: Verificar el estado del RS con un nodo caído

Ahora veremos **solo 2 nodos** (mongo3 no aparece o aparece como no alcanzable): `docker exec -it mongo1 mongosh --eval "rs.status()" | findstr "stateStr"`

```
PS C:\TallerDocker> docker exec -it mongo1 mongosh --eval "rs.status()" | findstr "stateStr"
      stateStr: 'PRIMARY',
      stateStr: 'SECONDARY',
      stateStr: '(not reachable/healthy)',
PS C:\TallerDocker>
```

Ilustración 26: verificar_servidor_caído.png

3.4: Ejecutar una consulta con el nodo caído

Vamos a probar que **el sistema sigue funcionando**:

```
docker exec -it mongo1 mongosh escuela --eval
'db.empleados.aggregate([{$sort:{salario:-1}},{$limit:1}])'
```

```
PS C:\TallerDocker> docker exec -it mongo1 mongosh escuela --eval 'db.empleados.aggregate([{$sort:{salario:-1}},{$limit:1}])'
[ { _id: 6, nombre: 'Frank', salario: 2000, departamento_id: 2 } ]
PS C:\TallerDocker>
```

Ilustración 27: prueba_sistema.png

Se muestra que el resultado de la consulta funcionando normalmente (Frank con salario 2000).

3.5: Levantar el nodo nuevamente

Recuperamos el nodo: `docker start mongo3`

```
PS C:\TallerDocker> docker start mongo3
mongo3
PS C:\TallerDocker>
```

Ilustración 28: levantar_servidor.png

3.6: Verificar recuperación del nodo

Vemos los 3 nodos activos nuevamente: `docker exec -it mongo1 mongosh --eval "rs.status()" | findstr "stateStr"`

```
PS C:\TallerDocker> docker exec -it mongo1 mongosh --eval "rs.status()" | fi
ndstr "stateStr"
      stateStr: 'PRIMARY',
      stateStr: 'SECONDARY',
      stateStr: 'SECONDARY',
PS C:\TallerDocker>
```

Ilustración 29: recuperación_servicio.png

BONUS (+2pts)

Seleccionamos la base de datos escuela:

```
rs0 [direct: primary] test> use escuela
switched to db escuela
```

Ilustración 30: selección_base_datos_escuela.png

Crear un índice shard key

```
rs0 [direct: primary] escuela> db.empleados.createIndex({ departamento_id: 1 })
{
  "createdCollectionAutomatically": false,
  "numIndexesBefore": 1,
  "numIndexesAfter": 2,
  "ok": 1
}
```

Ilustración 31: creación_indice_shard_key.png

Justificación / explicación

// Shard key sugerida: { departamento_id: 1 }
// Permite distribuir los documentos por departamento de manera uniforme.
// Queries filtrando por departamento_id serían targeted, accediendo solo al shard relevante.
// Esta estrategia mejora rendimiento y evita scatter-gather queries al buscar empleados de un departamento específico.

Comprobación de índices

```
rs0 [direct: primary] escuela> db.empleados.getIndexes()
[
  { "v": 2, "key": { "_id": 1 }, "name": "_id_" },
  { "v": 2, "key": { "departamento_id": 1 }, "name": "departamento_id_1" }
]
```

Ilustración 32: estado_índices.png

Simulación de "targeted query"

```
rs0 [direct: primary] escuela> db.empleados.find({ departamento_id: 3 }).pretty()
[
  { _id: 3, nombre: 'Marta', departamento_id: 3, salario: 2000 },
  { _id: 5, nombre: 'Frank', departamento_id: 3, salario: 2200 }
]
```

Ilustración 33: simulación_targeted.png

2.7 Habilidades blandas empleadas en la práctica

☐ Liderazgo
☒ Trabajo en equipo
☐ Comunicación asertiva
☐ La empatía
☐ Pensamiento crítico
☐ Flexibilidad
☐ La resolución de conflictos
☐ Adaptabilidad
☐ Responsabilidad

2.8 Conclusiones

- La práctica permitió comprender el funcionamiento de Replica Set en bases de datos distribuidas

- MongoDB demostró alta disponibilidad y tolerancia a fallos en entornos distribuidos

- El sistema mantuvo operatividad incluso con 1/3 de nodos inactivos

- Las consultas con Aggregation Pipeline demostraron capacidades analíticas avanzadas

- Docker facilitó la simulación de entornos distribuidos complejos

- Se cumplieron los objetivos de entender alta disponibilidad y escalabilidad

2.9 Recomendaciones

- Implementar más nodos en el Replica Set para mayor resiliencia

- Utilizar herramientas de monitoreo para observabilidad del cluster

- Practicar con diferentes estrategias de sharding y replicación

- Realizar pruebas de carga para evaluar rendimiento bajo estrés

- Implementar backups automáticos en entornos de producción

- Documentar procedimientos de recuperación ante desastres

2.10 Referencias bibliográficas

[1] MongoDB Inc., "Replication," *MongoDB Documentation*, 2025. [Online]. Available: https://www.mongodb.com/docs/manual/replication/
[2] Docker Inc., "Docker Compose documentation," 2025. [Online]. Available: https://docs.docker.com/compose/
[3] MongoDB Inc., "Aggregation Pipeline," *MongoDB Documentation*, 2025. [Online]. Available: https://www.mongodb.com/docs/manual/core/aggregation-pipeline/

2.11 Anexos

README

Descripción General

Esta práctica implementa un sistema de base de datos distribuida utilizando MongoDB con arquitectura Replica Set. El objetivo es demostrar los conceptos fundamentales de:

- Alta disponibilidad mediante réplicas
- Tolerancia a fallos en sistemas distribuidos
- Consultas avanzadas con Aggregation Pipeline
- Resiliencia ante caídas de nodos

El proyecto utiliza Docker Compose para orquestar 3 contenedores MongoDB que forman un Replica Set, simulando un entorno distribuido real.

Tecnologías Utilizadas

- Docker version 28.4.0, build
- Docker Compose version v2.39.4-desktop.1
- MongoDB: 2.5.8
- Sistema Operativo: Windows 11 Pro
- Terminal: PowerShell

Instrucciones de Ejecución

Prerequisitos
- Docker Desktop instalado y corriendo
- PowerShell o terminal con acceso a Docker

- Al menos 2GB de RAM disponible

Paso 1: Levantar el entorno

Navegar al directorio del proyecto:

cd C:\TallerDocker

Iniciar los contenedores:
bashdocker compose up -d

Verificar que los 3 contenedores estén corriendo:
bashdocker ps

Paso 2: Inicializar el Replica Set

bashdocker exec -it mongo1 mongosh --eval "rs.initiate({_id: 'rs0', members: [{_id: 0, host: 'mongo1:27017'},{_id: 1, host: 'mongo2:27017'},{_id: 2, host: 'mongo3:27017'}]})"
Importante: Esperar 10-15 segundos para que se elija el PRIMARY.
Verificar el estado:
bashdocker exec -it mongo1 mongosh --eval "rs.status()"

Paso 3: Insertar datos
Insertar los siguientes datos en cada archivo utilizando Visual Studio Code:

//Insertar departamentos

{"_id": 1, "nombre": "Ventas"}
{"_id": 2, "nombre": "TI"}
{"_id": 3, "nombre": "RRHH"}
{"_id": 4, "nombre": "Logística"}

// Insertar empleados

{"_id": 1, "nombre": "Ana", "salario": 1200, "departamento_id": 1}
{"_id": 2, "nombre": "Bruno", "salario": 900, "departamento_id": 1}
{"_id": 3, "nombre": "Carla", "salario": 1500, "departamento_id": 2}
{"_id": 4, "nombre": "Diego", "salario": 800, "departamento_id": 2}
{"_id": 5, "nombre": "Elena", "salario": 700, "departamento_id": 3}
{"_id": 6, "nombre": "Frank", "salario": 2000, "departamento_id": 2}

// Insertar ventas

{"_id": {"sucursal":"Quito","mes":"2025-08"}, "total": 34000}
{"_id": {"sucursal":"Guayaquil","mes":"2025-08"}, "total": 42000}
{"_id": {"sucursal":"Cuenca","mes":"2025-08"}, "total": 18000}
{"_id": {"sucursal":"Quito","mes":"2025-09"}, "total": 39000}
{"_id": {"sucursal":"Guayaquil","mes":"2025-09"}, "total": 33000}
{"_id": {"sucursal":"Cuenca","mes":"2025-09"}, "total": 25000}

Conectarse al nodo PRIMARY:
bashdocker exec -it mongo1 mongosh escuela

Importar los datos

docker exec -i mongo1 mongoimport --db escuela --collection departamentos --file /data/import/departamentos.json
docker exec -i mongo1 mongoimport --db escuela --collection empleados --file /data/import/empleados.json
docker exec -i mongo1 mongoimport --db escuela --collection ventas --file /data/import/ventas.json

Paso 4: Ejecutar consultas

Ver el archivo consultas.md para las 6 consultas completas.
Conectarse a MongoDB:
bashdocker exec -it mongo1 mongosh escuela
Copiar y ejecutar cada consulta del archivo consultas.md.

Paso 5: Probar resiliencia
Ver el archivo resiliencia.md para el procedimiento completo.
Resumen:

Apagar nodo
docker stop mongo3

 # Verificar estado
docker exec -it mongo1 mongosh --eval "rs.status()"

 # Ejecutar consulta y se verifico que si funciona
docker exec -it mongo1 mongosh escuela
db.empleados.aggregate([{$sort:{salario:-1}},{$limit:1}])
exit

 # Recuperar nodo
docker start mongo3

Resultados Obtenidos
Consulta 1: Empleados con salario mayor al promedio

Promedio empresa: $1,016.67
Empleados encontrados: 3
Empleado con mayor diferencia: Frank (+$983.33)

Consulta 2: Departamentos sin empleados

Departamentos encontrados: 1
Logística (id: 4) - sin empleados asignados

Consulta 3: Empleado con salario más alto

Frank - Departamento TI - $2,000

Consulta 4: Promedio salarial por departamento

TI: $1,433.33 (3 empleados: Frank, Carla, Diego)
Ventas: $1,050.00 (2 empleados: Ana, Bruno)
RRHH: $700.00 (1 empleado: Elena)
Logística: N/A (sin empleados)

Consulta 5: Departamentos sobre el promedio general

Promedio general: $1,016.67
Departamentos sobre el promedio: 2

TI: $1,433.33 (+41.0% sobre promedio, 3 empleados)
Ventas: $1,050.00 (+3.3% sobre promedio, 2 empleados)

Consulta 6: Sucursal top por mes

2025-08: Guayaquil ($42,000)
2025-09: Quito ($39,000)

Prueba de Resiliencia:

Sistema continuó operando con 1 nodo caído
Quorum mantenido (2/3 nodos)
Recuperación automática exitosa
Sin pérdida de datos

Conclusiones

Durante el desarrollo de esta práctica pude comprender de manera práctica los conceptos fundamentales de las bases de datos distribuidas que habíamos revisado en teoría.

Alta disponibilidad en acción: Ver cómo el sistema continuó funcionando incluso con un nodo caído fue revelador. En aplicaciones reales, esto significa que los usuarios no experimentarían interrupciones del servicio, lo cual es crítico para aplicaciones de producción.
Importancia de la arquitectura distribuida: El Replica Set no solo proporciona redundancia, sino que también permite distribuir la carga de lectura entre múltiples nodos. Esto es esencial para escalar aplicaciones con alto tráfico.
Complejidad de sistemas distribuidos: Enfrenté desafíos como la elección automática del PRIMARY (que no fue el nodo que esperaba inicialmente) y esto me hizo entender que los sistemas distribuidos tienen comportamientos no deterministas.
Aggregation Pipeline como herramienta poderosa: Las consultas complejas que realizamos (como calcular promedios por ventanas o hacer joins entre colecciones) demostraron que MongoDB puede manejar análisis sofisticados directamente en la base de datos, reduciendo la necesidad de procesamiento en la aplicación.
Docker como facilitador: Sin Docker, configurar un Replica Set de 3 nodos hubiera requerido 3 máquinas virtuales o servidores físicos. Docker me permitió simular un entorno distribuido completo en mi laptop, lo cual es invaluable para aprendizaje y desarrollo.

Dificultades superadas:
La experiencia de resolver problemas reales (como el formato de archivos JSON, la elección del PRIMARY, y los caracteres especiales en PowerShell)

Aplicabilidad práctica:
Esta práctica me ha dado las bases para poder implementar y administrar bases de datos distribuidas en entornos reales. Conceptos como replicación, failover automático, y tolerancia a fallos son fundamentales en arquitecturas de microservicios y aplicaciones cloud-native que son el estándar en la industria actual.

BONUS — Prueba de Resiliencia (Simulación de Caída y Recuperación de Nodo)

Objetivo
Comprobar la ****tolerancia a fallos**** y la ****alta disponibilidad**** del Replica Set de MongoDB, simulando la caída de un nodo y verificando que el sistema continúe operativo. Luego, recuperar el nodo y confirmar que se reintegra sin pérdida de datos.

Procedimiento

> Importante: Todos los comandos se ejecutan desde PowerShell, no dentro del shell de MongoDB (`mongosh`).

Verificar estado inicial del Replica Set
```powershell
docker exec -it mongo1 mongosh --eval "rs.status()" | findstr "stateStr"
```

Salida esperada:

"stateStr" : "PRIMARY"
"stateStr" : "SECONDARY"
"stateStr" : "SECONDARY"
Esto indica que los tres nodos están activos y sincronizados correctamente.

Apagar un nodo (simular caída)
docker stop mongo3
Simula una falla del nodo mongo3. Este contenedor se detendrá temporalmente.

Verificar el estado con un nodo caído
docker exec -it mongo1 mongosh --eval "rs.status()" | findstr "stateStr"

Salida esperada:
"stateStr" : "PRIMARY"
"stateStr" : "SECONDARY"

El nodo mongo3 no aparece o figura como inalcanzable.
El sistema mantiene quorum (2/3 nodos activos), por lo que sigue funcionando sin interrupciones.

Ejecutar una consulta mientras el nodo está caído
docker exec -it mongo1 mongosh escuela --eval 'db.empleados.aggregate([{$sort:{salario:-1}},{$limit:1}])'

Salida esperada:
{ "_id" : 5, "nombre" : "Frank", "departamento_id" : 3, "salario" : 2200 }

El sistema continúa respondiendo consultas de lectura, demostrando alta disponibilidad aun con un nodo inactivo.

Levantar nuevamente el nodo caído
docker start mongo3
Esto reinicia el contenedor mongo3, que se reincorpora automáticamente al Replica Set.

Verificar recuperación y sincronización

docker exec -it mongo1 mongosh --eval "rs.status()" | findstr "stateStr"

Salida esperada:
"stateStr" : "PRIMARY"
"stateStr" : "SECONDARY"
"stateStr" : "SECONDARY"
El nodo mongo3 vuelve al estado SECONDARY, sincronizado con el resto del clúster.

CONSULTAS

 # Consultas - Aggregation Pipeline

CONSULTA 1: Empleados con salario mayor al promedio de la empresa

Método 1: Con $setWindowFields (MongoDB ≥5)

Explicación:
- Usamos $setWindowFields para calcular el promedio salarial de toda la empresa
- Este promedio se agrega como un campo a cada documento
- Luego filtramos con $match solo los empleados cuyo salario supera ese promedio
- Finalmente proyectamos los campos relevantes incluyendo la diferencia con el promedio

****Pipeline:****

```
db.empleados.aggregate([
  {
    // Calculamos el promedio de salarios de toda la empresa usando ventanas
    $setWindowFields: {
      output: {
        promedioEmpresa: { $avg: "$salario" }
      }
    }
  },
  {
    // Filtramos solo los empleados con salario mayor al promedio
    $match: {
      $expr: { $gt: ["$salario", "$promedioEmpresa"] }
    }
  },
  {
    // Proyectamos los campos relevantes
    $project: {
      nombre: 1,
      departamento_id: 1,
      salario: 1,
      promedioEmpresa: { $round: ["$promedioEmpresa", 2] },
      diferencia: { $round: [{ $subtract: ["$salario", "$promedioEmpresa"] }, 2] }
    }
  },
  {
    $sort: { salario: -1 }
  }
])
```

Resultado :
Promedio empresa: $1,016.67
Empleados encontrados: 3 (Frank, Carla, Ana)
Frank: $2000 (+$983.33)
Carla: $1500 (+$483.33)
Ana: $1200 (+$183.33)

Método 2:Sin ventanas (usando $facet)

Explicación:
- Usamos $facet para ejecutar dos pipelines en paralelo
- Un pipeline calcula el promedio general
- Otro pipeline mantiene todos los empleados
- Luego combinamos ambos resultados y filtramos

****Pipeline:****

```
db.empleados.aggregate([
  {
    $facet: {
      // Pipeline para calcular el promedio
      promedio: [
        { $group: { _id: null, avg: { $avg: "$salario" } } }
      ],
      // Pipeline con todos los empleados
      empleados: []
    }
  },
  {
    // Extraemos el valor del promedio
    $project: {
      promedioEmpresa: { $arrayElemAt: ["$promedio.avg", 0] },
      empleados: 1
    }
  },
  {
    // Descomponemos el array de empleados
    $unwind: "$empleados"
  },
  {
    // Filtramos empleados con salario mayor al promedio
    $match: {
      $expr: { $gt: ["$empleados.salario", "$promedioEmpresa"] }
    }
  },
  {
    // Proyección final
    $project: {
      _id: "$empleados._id",
      nombre: "$empleados.nombre",
      departamento_id: "$empleados.departamento_id",
      salario: "$empleados.salario",
      promedioEmpresa: { $round: ["$promedioEmpresa", 2] },
      diferencia: { $round: [{ $subtract: ["$empleados.salario", "$promedioEmpresa"] }, 2] }
    }
```

```
  },
  {
    $sort: { salario: -1 }
  }
])
```

CONSULTA 2: Departamentos sin empleados asignados

Explicacion:
- Usamos $lookup para hacer un "join" entre departamentos y empleados
- El resultado del lookup es un array con los empleados de cada departamento
- Agregamos un campo que cuenta el tamaño de ese array
- Filtramos solo los departamentos donde el conteo es 0 (sin empleados)

****Pipeline:****

```
db.departamentos.aggregate([
  {
    // Hacemos un "join" con la colección empleados
    $lookup: {
      from: "empleados",
      localField: "_id",
      foreignField: "departamento_id",
      as: "empleados"
    }
  },
  {
    // Agregamos un campo con el conteo de empleados
    $addFields: {
      cantidadEmpleados: { $size: "$empleados" }
    }
  },
  {
    // Filtramos solo los departamentos sin empleados
    $match: {
      cantidadEmpleados: 0
    }
  },
  {
    // Proyectamos solo los campos relevantes
    $project: {
      _id: 1,
      nombre: 1,
      cantidadEmpleados: 1
    }
  }
])
```

Resultado :
Departamentos encontrados: 1
Logística (id: 4) - sin empleados

CONSULTA 3: Empleado con salario más alto

Explicacion:
- Ordenamos todos los empleados por salario de forma descendente

- Tomamos solo el primer resultado usando $limit
- Este será el empleado con el salario más alto

Pipeline:

```
db.empleados.aggregate([
  {
    // Ordenamos por salario de mayor a menor
    $sort: { salario: -1 }
  },
  {
    // Tomamos solo el primer resultado
    $limit: 1
  },
  {
    // Proyección de campos
    $project: {
      _id: 1,
      nombre: 1,
      departamento_id: 1,
      salario: 1
    }
  }
])
```

CONSULTA 4: Para cada empleado, mostrar el salario promedio de su departamento

Explicacion:
- Usamos $setWindowFields con partitionBy para agrupar por departamento
- Calculamos el promedio salarial dentro de cada partición (departamento)
- Cada empleado tendrá el promedio de su departamento como un campo adicional
- También calculamos la diferencia entre su salario y el promedio de su departamento

Pipeline:

```
db.empleados.aggregate([
  {
    // Calculamos el promedio salarial agrupado por departamento usando ventanas
    $setWindowFields: {
      partitionBy: "$departamento_id",
      output: {
        promedioDepartamento: { $avg: "$salario" }
      }
    }
  },
  {
    // Ordenamos por departamento y salario
    $sort: { departamento_id: 1, salario: -1 }
  },
  {
    // Proyectamos los campos relevantes
    $project: {
      _id: 1,
      nombre: 1,
      departamento_id: 1,
```

```
    salario: 1,
    promedioDepartamento: { $round: ["$promedioDepartamento", 2] },
    diferenciaConPromedio: {
     $round: [{ $subtract: ["$salario", "$promedioDepartamento"] }, 2]
    }
   }
  }
])
```

CONSULTA 5: Departamentos cuyo promedio salarial es mayor al promedio general

Explicacion:
- Usamos $facet para calcular dos cosas en paralelo:
  1. El promedio general de toda la empresa
  2. El promedio por cada departamento
- Luego comparamos ambos promedios
- Filtramos solo los departamentos cuyo promedio supera al promedio general
- Hacemos un lookup para obtener el nombre del departamento

****Pipeline:****

```
db.empleados.aggregate([
 {
  $facet: {
   // Calculamos el promedio general de toda la empresa
   promedioGeneral: [
    { $group: { _id: null, avg: { $avg: "$salario" } } }
   ],
   // Calculamos promedio por departamento
   porDepartamento: [
    {
     $group: {
      _id: "$departamento_id",
      promedioDepto: { $avg: "$salario" },
      cantidadEmpleados: { $sum: 1 }
     }
    }
   ]
  }
 },
 {
  // Extraemos los valores calculados
  $project: {
   promedioGeneral: { $arrayElemAt: ["$promedioGeneral.avg", 0] },
   porDepartamento: 1
  }
 },
 {
  // Descomponemos el array de departamentos
  $unwind: "$porDepartamento"
 },
 {
  // Filtramos departamentos con promedio mayor al general
  $match: {
   $expr: {
    $gt: ["$porDepartamento.promedioDepto", "$promedioGeneral"]
```

```
      }
    }
  },
  {
    // Join con departamentos para obtener el nombre
    $lookup: {
      from: "departamentos",
      localField: "porDepartamento._id",
      foreignField: "_id",
      as: "depto"
    }
  },
  {
    $unwind: "$depto"
  },
  {
    // Proyección final
    $project: {
      _id: "$porDepartamento._id",
      nombreDepartamento: "$depto.nombre",
      promedioDepartamento: { $round: ["$porDepartamento.promedioDepto", 2] },
      promedioGeneral: { $round: ["$promedioGeneral", 2] },
      cantidadEmpleados: "$porDepartamento.cantidadEmpleados",
      diferencia: {
        $round: [{ $subtract: ["$porDepartamento.promedioDepto", "$promedioGeneral"] }, 2]
      }
    }
  },
  {
    $sort: { promedioDepartamento: -1 }
  }
])
```

CONSULTA 6: Sucursal "top" por mes (sucursal con mayor total de ventas)

Explicacion:
- Ordenamos las ventas por mes y total descendente
- Agrupamos por mes usando $group
- Usamos $first para tomar la primera sucursal de cada grupo (la de mayor venta)
- El resultado muestra para cada mes cuál sucursal tuvo las mayores ventas

****Pipeline:****

```
db.ventas.aggregate([
  {
    // Extraemos los campos del _id compuesto
    $addFields: {
      sucursal: "$_id.sucursal",
      mes: "$_id.mes"
    }
  },
  {
    // Ordenamos por mes y total descendente
    $sort: { mes: 1, total: -1 }
  },
```

```
  {
   // Agrupamos por mes y tomamos la sucursal con mayor venta
   $group: {
    _id: "$mes",
    sucursalTop: { $first: "$sucursal" },
    totalMaximo: { $first: "$total" }
   }
  },
  {
   // Ordenamos por mes
   $sort: { _id: 1 }
  },
  {
   // Proyección final
   $project: {
    _id: 0,
    mes: "$_id",
    sucursalTop: 1,
    totalMaximo: 1
   }
  }
])
```

Resultado :

2025-08: Guayaquil ($42,000)
2025-09: Quito ($39,000)

RESILENCIA

Prueba de Resiliencia del Replica Set

Procedimiento

1. Estado inicial

- Replica Set con 3 nodos funcionando

- mongo1: PRIMARY

- mongo2: SECONDARY

- mongo3: SECONDARY

Comando para verificar:

docker exec -it mongo1 mongosh --eval "rs.status()" | findstr "stateStr"

2. Caída del nodo mongo3

Comando ejecutado:

docker stop

3. Observaciones con el nodo caído

¿Qué ocurrió?

- El Replica Set continuó operando normalmente
- mongo1 (PRIMARY) siguió aceptando lecturas y escrituras
- mongo2 (SECONDARY) siguió replicando datos
- El sistema mantuvo el quorum (2 de 3 nodos activos)

Estado del cluster:

- Nodos activos: 2/3
- PRIMARY: mongo1 (sin cambios)
- SECONDARY: mongo2 (sin cambios)
- mongo3: No alcanzable

Consultas ejecutadas:

- Las consultas funcionaron sin ningún problema
- No hubo pérdida de datos
- No hubo degradación del servicio

4. Recuperación del nodo

Comando ejecutado:

docker start mongo3

¿Qué ocurrió?

- mongo3 se reconectó al Replica Set automáticamente
- Se sincronizó con el PRIMARY para obtener datos que pudo haber perdido
- Volvió a su estado SECONDARY
- El cluster volvió a tener 3 nodos completamente funcionales

Ventajas de Replica Set en Bases de Datos Distribuidas:

- Alta disponibilidad: El sistema siguió funcionando con solo 2 de 3 nodos
- Sin pérdida de datos: Todas las escrituras se mantuvieron
- Recuperación automática: mongo3 se reincorporó sin intervención manual
- Tolerancia a fallos: El sistema puede soportar la caída de N/2 - 1 nodos

¿Por qué funcionó con 2 nodos?

- MongoDB usa votación por mayoría para elegir PRIMARY
- Con 3 nodos, se necesitan 2 votos (mayoría)
- 2 nodos activos = quorum alcanzado
- El sistema continúa operando

¿Qué pasaría si cayeran 2 nodos?

Solo quedaría 1 nodo activo

No hay quorum (necesita 2 de 3)

El nodo PRIMARY se volvería SECONDARY

No se aceptarían escrituras (solo lecturas si se configura)

El cluster entraría en modo "solo lectura" hasta recuperar otro nodo

Versiones de Software Utilizadas

- Docker version 28.4.0, build

- Docker Compose version v2.39.4-desktop.1

- MongoDB: 2.5.8

- Sistema Operativo: Windows 11 Pro

- Terminal: PowerShell

Dificultades Encontradas y Soluciones

1. mongo1 no era el PRIMARY inicial

Problema:

Al inicializar el Replica Set con `rs.initiate()`, MongoDB eligió automáticamente a "mongo2" como PRIMARY en lugar de mongo1. Esto causaba errores al intentar importar datos en mongo1, ya que solo el nodo PRIMARY puede aceptar escrituras.

Error obtenido:

Failed: (NotWritablePrimary) not primary

0 document(s) imported successfully

Causa:

MongoDB realiza una elección democrática automática al inicializar el Replica Set. El PRIMARY se elige basándose en factores como conectividad, latencia y prioridad. No necesariamente será el primer nodo de la lista.

****Solución aplicada:****

- Nos conectamos al nodo PRIMARY actual (mongo2):

  docker exec -it mongo2 mongosh

- Forzamos al PRIMARY actual a renunciar usando:

  rs.stepDown()

- Esperamos 10-15 segundos para que se realizara una nueva elección

- Verificamos el nuevo estado:

docker exec -it mongo1 mongosh --eval "rs.status()"

- Confirmamos que mongo1 ahora era PRIMARY y procedimos con la importación de datos

En un Replica Set real, el PRIMARY puede cambiar en cualquier momento (failover automático). Las aplicaciones deben estar preparadas para conectarse al PRIMARY actual, no asumir que siempre será el mismo nodo.