INFORME DE TALLER

I. PORTADA

Tema:	Docker – Creación de un cluster
Unidad de Organización Curricular:	PROFESIONAL
Nivel y Paralelo:	5 A
Alumnos participantes:	Analuiza Castillo Jimmy Sebastián Gordillo Guevara Luis Josué Manobanda Chango Ana Patricia Peñaloza Narváez Johnny Alexander
Asignatura:	Sistemas de Base de Datos Distribuidos
Docente:	Ing. José Caiza

II. INFORME DE TALLER

2.1 Objetivos

General:

Implementar y administrar un clúster de bases de datos distribuidas utilizando Docker y MongoDB, con el fin de comprender y aplicar los conceptos de réplica, escalabilidad y resiliencia en entornos de bases de datos distribuidas.

Específicos:

- Configurar un Replica Set de MongoDB en un entorno contenerizado mediante Docker Compose.
- Importar y manipular datos en un esquema distribuido utilizando herramientas de MongoDB.
- Ejecutar consultas avanzadas con agregaciones y ventanas para análisis de datos.
- Evaluar la resiliencia del clúster ante la caída de uno de sus nodos.
- Analizar el comportamiento de las consultas mediante el uso de explain().

2.2 Modalidad

- Práctica guiada en laboratorio con supervisión del docente.
- Individual o en equipos de 2 personas (según indicaciones del profesor).

2.3 Duración

- **Presenciales:** 2 horas (tiempo en clase).
- **No presenciales:** 1 hora adicional para redacción del informe.

2.4 Instrucciones

1. Preparar el entorno: Asegurarse de tener Docker Desktop instalado y en ejecución.
2. Levantar el clúster: Ejecutar docker compose up -d desde el directorio del proyecto para desplegar los contenedores.
3. Inicializar el Replica Set: Conectarse a mongo1 y ejecutar el script de inicialización del Replica Set rs0.
4. Importar datos: Utilizar el comando mongoimport para cargar los archivos JSON (departamentos.json, empleados.json, ventas.json) en la base de datos escuela.
5. Ejecutar consultas obligatorias (Parte C): Desarrollar y ejecutar en mongosh las consultas de agregación solicitadas, utilizando tanto métodos con ventanas ($setWindowFields) como tradicionales.
6. Probar resiliencia (Parte D): Detener un nodo (docker stop mongo3), ejecutar una consulta para observar el comportamiento del clúster, y luego reiniciarlo (docker start mongo3).

7. Generar entregables: Documentar los resultados con capturas de pantalla de las consultas, el estado del Replica Set (rs.status()) y la prueba de resiliencia.

2.5 Listado de materiales

Listado de equipos y materiales generales empleados en la guía práctica:

- Computadora con Windows/Linux/MacOS
- Docker Desktop
- MongoDB

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

2.6 Desarrollo de la actividad

Levantar el clúster

docker compose up -d

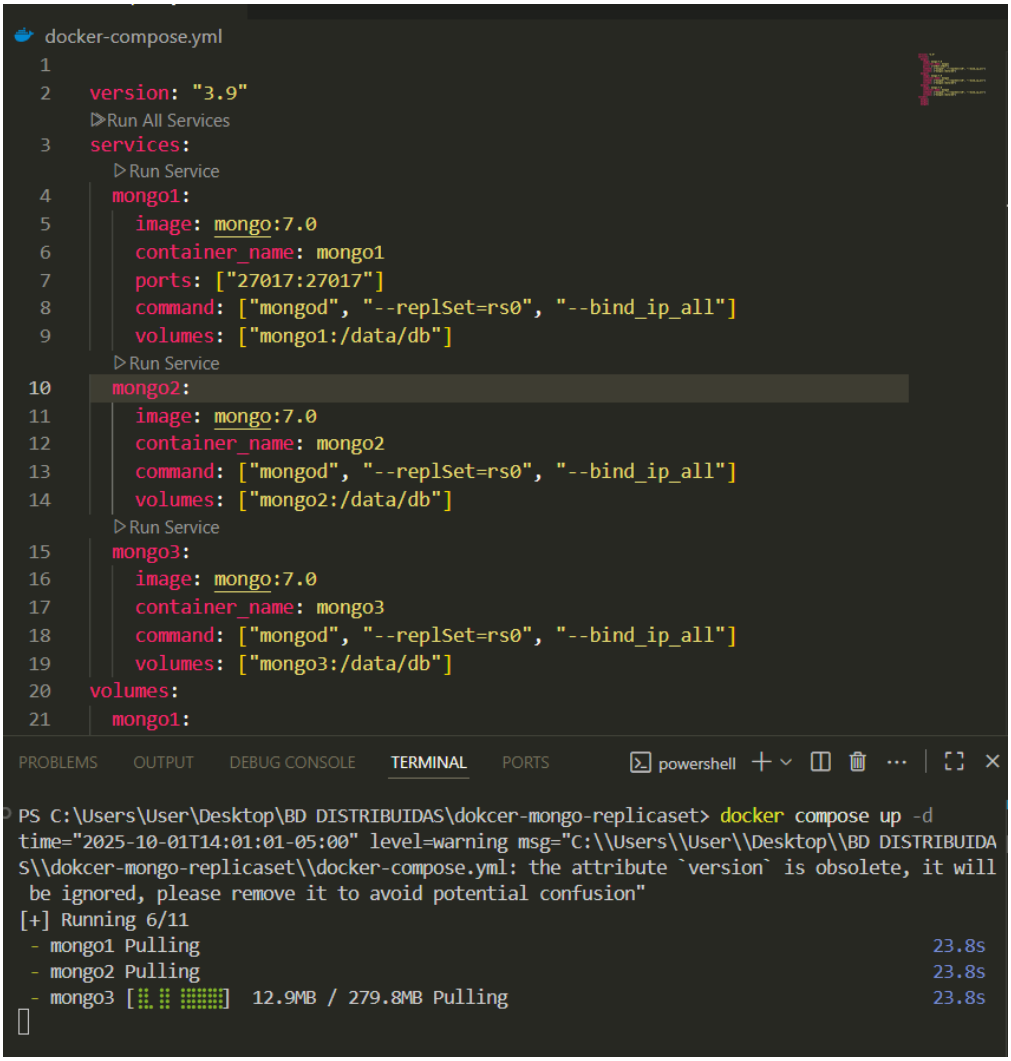

Fig. 1. Configuración inicial del clúster con Docker Compose.

Inicializar el Replica Set

docker exec -it mongo1 mongosh --eval '

```
PS C:\Users\User\Desktop\BD DISTRIBUIDAS\dokcer-mongo-replicaset> docker exec -it mongo1 mongosh
Current Mongosh Log ID: 68dd7ea5b4de2e8cffce5f46
```

Fig. 2. Comando para inicializar el Replica Set en MongoDB.

Se verifica la creación y estado de las replicas (Fig.3-5):

```
test> rs.initiate({
...   _id: "rs0",
...   members: [
...     { _id: 0, host: "mongo1:27017" },
...     { _id: 1, host: "mongo2:27017" },
...     { _id: 2, host: "mongo3:27017" }
...   ]
... })
...
{ ok: 1 }
rs0 [direct: other] test> exit
```

Fig. 3. Verificación del estado del Replica Set después de la inicialización.

```
PS C:\Users\User\Desktop\BD DISTRIBUIDAS\dokcer-mongo-replicaset> docker exec -it mongo1 mongosh --eval 'rs.status()'
{
  set: 'rs0',
  date: ISODate('2025-10-01T19:19:54.146Z'),
  myState: 1,
  term: Long('1'),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long('2000'),
  majorityVoteCount: 2,
  writeMajorityCount: 2,
  votingMembersCount: 3,
  writableVotingMembersCount: 3,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1759346391, i: 1 }), t: Long('1') },
    lastCommittedWallTime: ISODate('2025-10-01T19:19:51.515Z'),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1759346391, i: 1 }), t: Long('1') },
    appliedOpTime: { ts: Timestamp({ t: 1759346391, i: 1 }), t: Long('1') },
    durableOpTime: { ts: Timestamp({ t: 1759346391, i: 1 }), t: Long('1') },
    lastAppliedWallTime: ISODate('2025-10-01T19:19:51.515Z'),
    lastDurableWallTime: ISODate('2025-10-01T19:19:51.515Z')
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1759346351, i: 1 }),
  electionCandidateMetrics: {
    lastElectionReason: 'electionTimeout',
```

Fig. 4. Confirmación de la creación de las réplicas en el clúster.

```
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1759346391, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1759346391, i: 1 })
}
```

Fig. 5. Estado de los miembros del Replica Set mostrando nodos primarios y secundarios.

Importar datos

Get-Content departamentos.json | docker exec -i mongo1 mongoimport --db escuela --collection departamentos --file /dev/stdin
Get-Content empleados.json | docker exec -i mongo1 mongoimport --db escuela --collection empleados --file /dev/stdin
Get-Content ventas.json | docker exec -i mongo1 mongoimport --db escuela --collection ventas --file /dev/stdin

```
PS C:\Users\User\Desktop\BD DISTRIBUIDAS\dokcer-mongo-replicaset> Get-Content departamentos.json | docker exec -i mongo1 mongoimpo
rt --db escuela --collection departamentos --file /dev/stdin
2025-10-01T19:24:48.337+0000    connected to: mongodb://localhost/
2025-10-01T19:24:48.357+0000    4 document(s) imported successfully. 0 document(s) failed to import.
PS C:\Users\User\Desktop\BD DISTRIBUIDAS\dokcer-mongo-replicaset> docker -v
Docker version 28.4.0, build d8eb465
PS C:\Users\User\Desktop\BD DISTRIBUIDAS\dokcer-mongo-replicaset> Get-Content empleados.json | docker exec -i mongo1 mongoimport -
-db escuela --collection empleados --file /dev/stdin
2025-10-01T19:25:14.371+0000    connected to: mongodb://localhost/
2025-10-01T19:25:14.396+0000    6 document(s) imported successfully. 0 document(s) failed to import.
PS C:\Users\User\Desktop\BD DISTRIBUIDAS\dokcer-mongo-replicaset> Get-Content ventas.json     | docker exec -i mongo1 mongoimport
--db escuela --collection ventas --file /dev/stdin
2025-10-01T19:25:17.798+0000    connected to: mongodb://localhost/
2025-10-01T19:25:17.824+0000    6 document(s) imported successfully. 0 document(s) failed to import.
PS C:\Users\User\Desktop\BD DISTRIBUIDAS\dokcer-mongo-replicaset>
```

Fig. 6. Importación de datos JSON a las colecciones de MongoDB.

Consultas obligatorias (Parte C)

1. **Empleados con salario mayor al promedio de la empresa**
   - **Método 1 (ventanas, MongoDB ≥5): usar $setWindowFields para calcular promEmpresa y luego $match.**

```
rs0 [direct: primary] escuela> db.empleados.aggregate([
...   {
...     $setWindowFields: {
...       partitionBy: null,
...       output: {
...         promEmpresa: { $avg: "$salario" }
...       }
...     }
...   },
...   {
...     $match: { $expr: { $gt: ["$salario", "$promEmpresa"] } }
...   },
...   {
...     $project: { nombre: 1, salario: 1, promEmpresa: 1, _id: 0 }
...   }
... ])
[
  { nombre: 'Ana', salario: 1200, promEmpresa: 1183.3333333333333 },
  { nombre: 'Frank', salario: 2000, promEmpresa: 1183.3333333333333 },
  { nombre: 'Carla', salario: 1500, promEmpresa: 1183.3333333333333 }
]
```

Fig. 7. Consulta de empleados con salario mayor al promedio usando $setWindowFields.

$setWindowFields crea un campo promEmpresa con el promedio de todos los salarios.
$match compara cada salario contra ese promedio.
$project muestra solo la información relevante para la entrega.

- **Método 2 (sin ventanas): calcular promedio con un pipeline y usar ese valor con $expr.**

```
rs0 [direct: primary] escuela> var promedio = db.empleados.aggregate([
...   { $group: { _id: null, promEmpresa: { $avg: "$salario" } } }
... ]).toArray()[0].promEmpresa;
...
... db.empleados.aggregate([
...   {
...     $match: {
...       $expr: { $gt: ["$salario", promedio] }  // Compara cada salario con el promedio calculado
...     }
...   },
...   {
...     $project: { nombre: 1, salario: 1, _id: 0 }  // Solo campos relevantes
...   }
... ])
...
[
  { nombre: 'Ana', salario: 1200 },
  { nombre: 'Frank', salario: 2000 },
  { nombre: 'Carla', salario: 1500 }
]
```

Fig. 8. Consulta de empleados con salario mayor al promedio usando método tradicional.

Calculamos el promedio global (promedio) de manera independiente.
$match con $expr compara cada documento con ese promedio.
$project limpia la salida para la entrega.

2. **Departamentos sin empleados asignados**
   - **Usar $lookup de departamentos → empleados y filtrar con $match donde el arreglo resultante tenga tamaño 0.**
   - **Pista: $lookup + $addFields: {count: {$size: "$empleados"}} + $match: {count: 0}.**

```
rs0 [direct: primary] escuela> db.departamentos.aggregate([
...   // Hacemos un $lookup para traer los empleados que pertenecen a cada departamento
...   {
...     $lookup: {
...       from: "empleados",       // Colección a unir
...       localField: "_id",        // Campo de departamentos
...       foreignField: "departamento_id", // Campo correspondiente en empleados
...       as: "empleados"           // Nombre del arreglo resultante
...     }
...   },
...   //Contamos cuántos empleados tiene cada departamento
...   {
...     $addFields: {
...       count: { $size: "$empleados" } // Creamos un campo 'count' con la cantidad de empleados
...     }
...   },
...   //Filtramos solo los departamentos que no tienen empleados
...   {
...     $match: { count: 0 }
...   },
...   //Seleccionamos los campos que queremos mostrar
...   {
...     $project: { nombre: 1, _id: 0 }
...   }
... ])
[ { nombre: 'Log??stica' } ]
```

Fig. 9. Consulta de departamentos sin empleados asignados usando $lookup y $size.

$lookup une cada departamento con sus empleados correspondientes.
$addFields calcula la cantidad de empleados por departamento.
$match filtra solo los departamentos sin empleados (count: 0).
$project limpia la salida para mostrar solo el nombre del departamento.

3. **Empleado con salario más alto**
   - **Opción A: $sort descendente y $limit: 1.**

```
rs0 [direct: primary] escuela> db.empleados.aggregate([
...   // Ordenamos los empleados por salario de mayor a menor
...   { $sort: { salario: -1 } },
...   // Tomamos solo el primer documento (el salario más alto)
...   { $limit: 1 },
...   //Proyectamos solo los campos relevantes para la entrega
...   { $project: { nombre: 1, salario: 1, departamento_id: 1, _id: 0 } }
... ])
[ { nombre: 'Frank', salario: 2000, departamento_id: 2 } ]
```

Fig. 10. Empleado con salario más alto usando $sort y $limit.

$sort organiza todos los empleados de mayor a menor salario.
$limit: 1 devuelve solo el empleado con salario máximo.
$project muestra solo la información relevante.

   - **Opción B: $group + $topN (si está disponible) o $max y luego $match.**

```
rs0 [direct: primary] escuela> db.empleados.aggregate([
...   // Agrupamos todos los empleados en un solo grupo para calcular el salario máximo
...   { $group: { _id: null, salarioMax: { $max: "$salario" } } },
...   // Hacemos un $lookup para obtener el empleado que tenga ese salario máximo
...   {
...     $lookup: {
...       from: "empleados",
...       localField: "salarioMax",
...       foreignField: "salario",
...       as: "empleadoTop"
...     }
...   },
...   // Desenrollamos el arreglo para acceder al documento
...   { $unwind: "$empleadoTop" },
...   // Proyectamos solo los campos necesarios
...   { $project: { nombre: "$empleadoTop.nombre", salario: "$empleadoTop.salario", departamento_id: "$empleadoTop.departamento_id", _id: 0 } }
... ])
...
[ { nombre: 'Frank', salario: 2000, departamento_id: 2 } ]
```

Fig. 11. Empleado con salario más alto usando $group y $lookup.

$group obtiene el salario máximo de toda la empresa.
$lookup busca los empleados que tengan ese salario máximo.
$unwind transforma el arreglo de resultados en documento único.
$project selecciona los campos que queremos mostrar.

4. **Para cada empleado, mostrar el salario promedio de su departamento**
   - **Opción con ventanas: $setWindowFields con partitionBy: "$departamento_id" y {$avg: "$salario"}.**

```
rs0 [direct: primary] escuela> db.empleados.aggregate([
...   // Calculamos el promedio de salario de cada departamento
...   {
...     $setWindowFields: {
...       partitionBy: "$departamento_id",      // Agrupamos por departamento
...       output: {
...         promDepartamento: { $avg: "$salario" }  // Promedio de salarios en ese departamento
...       }
...     }
...   },
...   //  Seleccionamos los campos a mostrar
...   {
...     $project: { nombre: 1, salario: 1, departamento_id: 1, promDepartamento: 1, _id: 0 }
...   }
... ])
```

Fig. 12. Cálculo del salario promedio por departamento usando $setWindowFields.

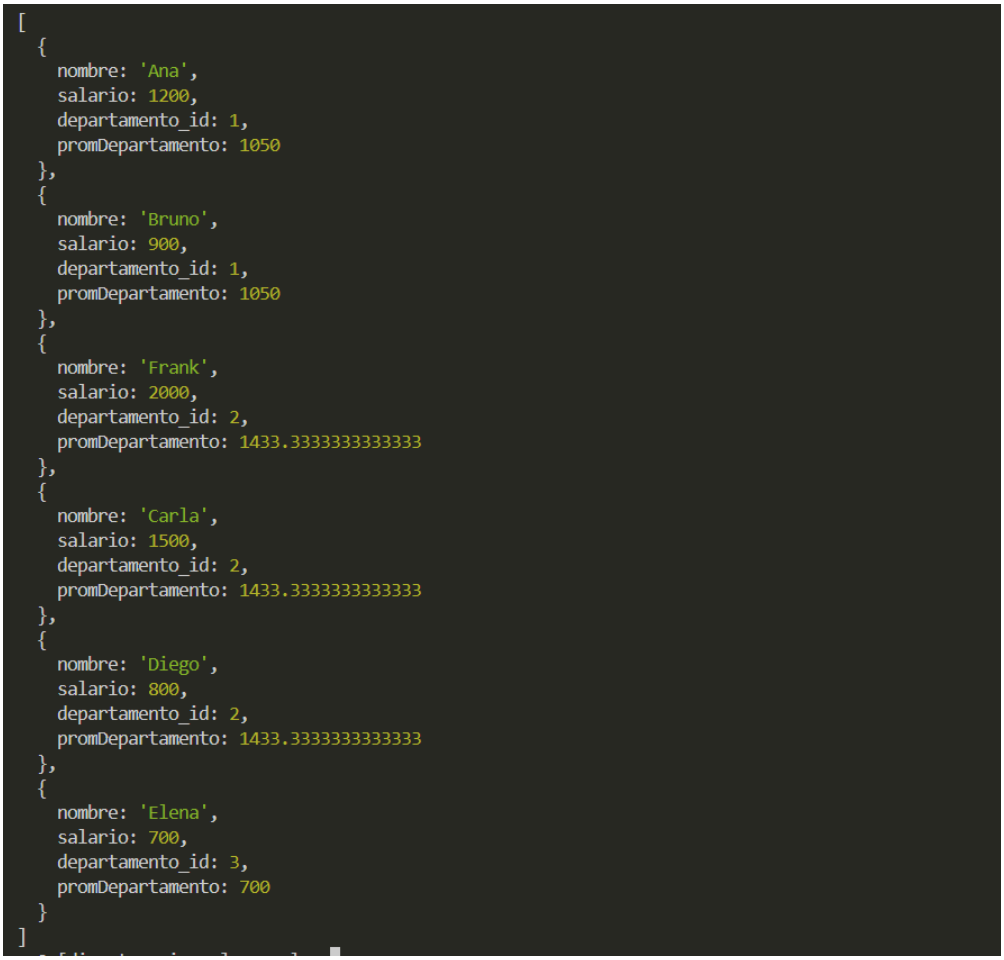

Fig. 13. Resultados del promedio salarial por departamento.

$setWindowFields permite calcular el promedio de cada grupo (partitionBy: "$departamento_id") y lo agrega como nuevo campo promDepartamento. $project muestra solo la información necesaria para la entrega.

- **Opción sin ventanas: $lookup hacia empleados con let: {dep:"$departamento_id"} y un pipeline que haga $match y $group para obtener el promedio; luego proyectar.**

```
rs0 [direct: primary] escuela> db.empleados.aggregate([
...   // Hacemos un $lookup hacia la misma colección para traer todos los empleados del mismo departamento
...   {
...     $lookup: {
...       from: "empleados",
...       let: { dep: "$departamento_id" },   // Variable con el id del departamento actual
...       pipeline: [
...         { $match: { $expr: { $eq: ["$departamento_id", "$$dep"] } } }, // Filtramos por el mismo departamento
...         { $group: { _id: null, promDepartamento: { $avg: "$salario" } } } // Calculamos el promedio
...       ],
...       as: "promDep"
...     }
...   },
...   // Desenrollamos el arreglo resultante para acceder al valor
...   { $unwind: "$promDep" },
...   //Proyectamos los campos finales
...   {
...     $project: { nombre: 1, salario: 1, departamento_id: 1, promDepartamento: "$promDep.promDepartamento", _id: 0 }
...   }
... ])
```

Fig. 14. Cálculo del salario promedio por departamento usando método tradicional.

Fig. 15. Resultados del promedio salarial por departamento con método tradicional.

$lookup trae todos los empleados del mismo departamento usando la variable dep.
$group calcula el promedio de salarios de ese departamento.
$unwind convierte el arreglo de resultados en documento único.
$project selecciona los campos que se deben entregar.

5. **Departamentos cuyo promedio salarial es mayor al promedio general**
   - **Pipeline en dos niveles:**
     a) $group por departamento_id para obtener promDep.

```
rs0 [direct: primary] escuela> db.empleados.aggregate([
...   // Calculamos el promedio salarial por departamento
...   {
...     $group: {
...       _id: "$departamento_id",
...       promDep: { $avg: "$salario" }
...     }
...   },
...   // Calculamos el promedio general como un campo adicional usando ventanas
...   {
...     $setWindowFields: {
...       partitionBy: null,   // Todos los documentos juntos
...       output: {
...         promGlobal: { $avg: "$promDep" }  // Promedio de todos los departamentos
...       }
...     }
...   },
...   // Filtramos los departamentos cuyo promedio es mayor al promedio global
...   {
...     $match: { $expr: { $gt: ["$promDep", "$promGlobal"] } }
...   },
...   // Proyectamos campos finales
...   {
...     $project: { departamento_id: "$_id", promDep: 1, promGlobal: 1, _id: 0 }
...   }
... ])
[
  {
    promDep: 1433.3333333333333,
    promGlobal: 1061.111111111111,
    departamento_id: 2
  }
]
```

Fig. 16. Departamentos con promedio salarial mayor al general usando $setWindowFields.

$group calcula el promedio salarial de cada departamento.
$setWindowFields calcula el promedio global de los departamentos.
$match filtra los que superan el promedio global.
$project muestra solo la información relevante.

b) Comparar contra promedio global (ventanas o $lookup/doble pipeline).

```
rs0 [direct: primary] escuela> db.departamentos.aggregate([
...   // Hacemos $lookup para calcular promedio salarial de cada departamento
...   {
...     $lookup: {
...       from: "empleados",
...       localField: "_id",
...       foreignField: "departamento_id",
...       as: "empleadosDept"
...     }
...   },
...   {
...     $addFields: {
...       promDep: { $avg: "$empleadosDept.salario" }  // Promedio por departamento
...     }
...   },
...   // Calculamos promedio global de todos los departamentos
...   {
...     $group: {
...       _id: null,
...       promGlobal: { $avg: "$promDep" },
...       departamentos: { $push: { departamento_id: "$_id", promDep: "$promDep" } }
...     }
...   },
...   // Desenrollamos los departamentos para filtrar
...   { $unwind: "$departamentos" },
...   // Filtramos solo los departamentos cuyo promedio > promedio global
...   {
...     $replaceRoot: { newRoot: "$departamentos" }
...   },
...   {
...     $match: { $expr: { $gt: ["$promDep", "$promGlobal"] } }
...   }
... ])
...
[
  { departamento_id: 2, promDep: 1433.3333333333333 },
  { departamento_id: 3, promDep: 700 },
  { departamento_id: 4, promDep: null },
  { departamento_id: 1, promDep: 1050 }
]
```

Fig. 17. Departamentos con promedio salarial mayor al general usando método tradicional.

$lookup une cada departamento con sus empleados.
$addFields calcula el promedio por departamento (promDep).
$group calcula el promedio global de todos los departamentos (promGlobal) y guarda los departamentos en un arreglo.
$unwind y $replaceRoot permiten filtrar individualmente.
$match selecciona los departamentos cuyo promedio supera al promedio global.

6. **Ventas: sucursal “top” por mes**
   - **A partir de ventas, obtener por cada mes la sucursal con mayor total.**

```
rs0 [direct: primary] escuela> db.ventas.aggregate([
...   // Ordenamos por mes ascendente y total descendente
...   { $sort: { "_id.mes": 1, total: -1 } },
...   // Agrupamos por mes y tomamos la primera sucursal (la de mayor total)
...   {
...     $group: {
...       _id: "$_id.mes",
...       sucursalTop: { $first: "$_id.sucursal" },
...       totalTop: { $first: "$total" }
...     }
...   },
...   //  Proyectamos los campos finales
...   { $project: { mes: "$_id", sucursalTop: 1, totalTop: 1, _id: 0 } },
...   // Ordenamos por mes (opcional)
...   { $sort: { mes: 1 } }
... ])
...
[
  { sucursalTop: 'Guayaquil', totalTop: 42000, mes: '2025-08' },
  { sucursalTop: 'Quito', totalTop: 39000, mes: '2025-09' }
]
```

Fig. 18. Sucursal top por mes usando $sort y $first.

$sort asegura que el documento con mayor total para cada mes esté primero.
$group por mes y $first selecciona la sucursal top.
$project limpia la salida para entregar solo los campos deseados.

Sugerencia: ordenar por total desc, agrupar por mes y tomar el primero, o usar $group + $topN.

```
rs0 [direct: primary] escuela> db.ventas.aggregate([
...   // Agrupamos por mes y usamos $topN para obtener la sucursal con mayor total
...   {
...     $group: {
...       _id: "$_id.mes",
...       topSucursal: {
...         $topN: {
...           output: { sucursal: "$_id.sucursal", total: "$total" },
...           sortBy: { total: -1 },
...           n: 1
...         }
...       }
...     }
...   },
...   // Desenrollamos el arreglo topSucursal
...   { $unwind: "$topSucursal" },
...   // Proyectamos los campos finales
...   { $project: { mes: "$_id", sucursalTop: "$topSucursal.sucursal", totalTop: "$topSucursal.total", _id: 0 } },
...   { $sort: { mes: 1 } }
... ])
[
  { mes: '2025-08', sucursalTop: 'Guayaquil', totalTop: 42000 },
  { mes: '2025-09', sucursalTop: 'Quito', totalTop: 39000 }
]
```

Fig. 19. Sucursal top por mes usando $group y $topN.

$group agrupa las ventas por mes.
$topN selecciona la sucursal con mayor total en cada mes.
$unwind transforma el arreglo topSucursal en documento único.
$project muestra los campos finales para la entrega.

Parte D — Resiliencia (obligatorio, breve)

1. **Con el rs0 funcionando, apaga un nodo:**

   docker stop mongo3

2. **Ejecuta una consulta del punto C y explica qué ocurre con el primario/secundario.**

   docker start mongo3

NODO MONGO1

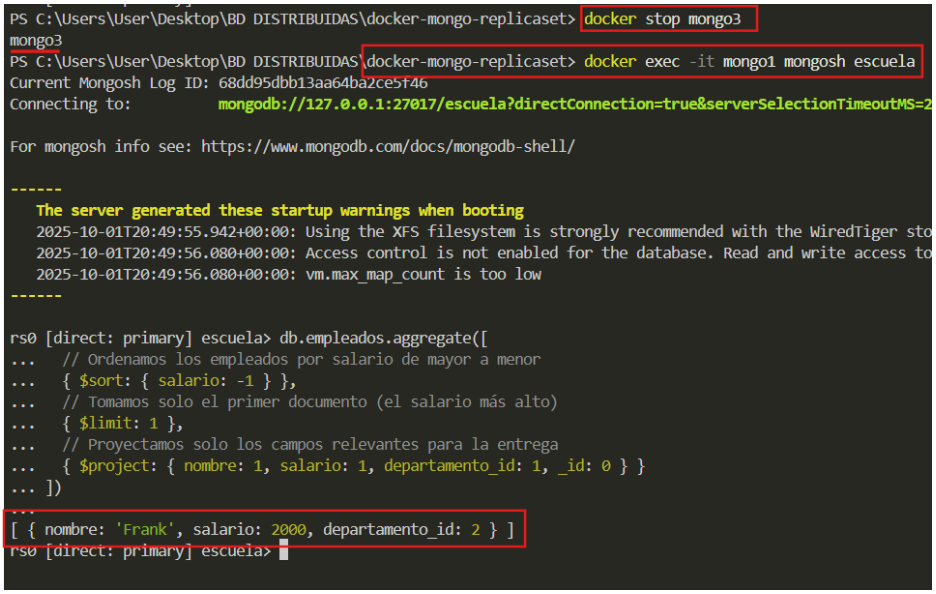

Fig. 20. Estado del Replica Set con el nodo mongo3 detenido.

NODO MONGO2

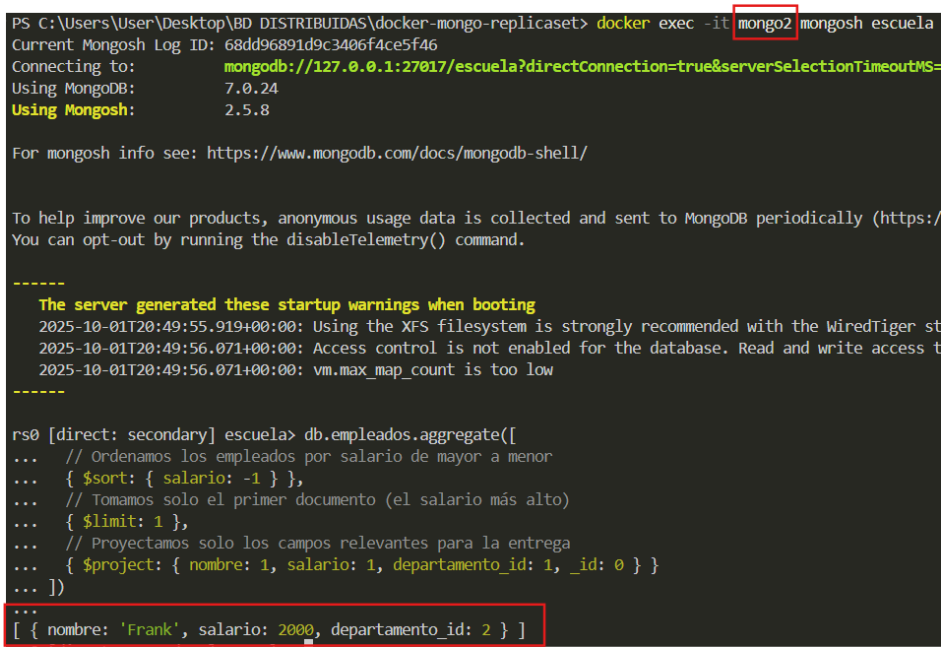

Fig. 21. Comportamiento del clúster durante la caída de un nodo secundario.

3. Vuelve a levantar el nodo: docker start mongo3

```
PS C:\Users\User\Desktop\BD DISTRIBUIDAS\docker-mongo-replicaset> docker start mongo3
mongo3
PS C:\Users\User\Desktop\BD DISTRIBUIDAS\docker-mongo-replicaset> docker exec -it mongo1 mongosh escuela
Current Mongosh Log ID: 68dd96cd11704fd174ce5f46
Connecting to:          mongodb://127.0.0.1:27017/escuela?directConnection=true&serverSelectionTimeoutMS=
Using MongoDB:          7.0.24
Using Mongosh:          2.5.8

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-10-01T20:49:55.942+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger st
   2025-10-01T20:49:56.080+00:00: Access control is not enabled for the database. Read and write access t
   2025-10-01T20:49:56.080+00:00: vm.max_map_count is too low
------

rs0 [direct: primary] escuela> db.empleados.aggregate([
...   // Ordenamos los empleados por salario de mayor a menor
...   { $sort: { salario: -1 } },
...   // Tomamos solo el primer documento (el salario más alto)
...   { $limit: 1 },
...   // Proyectamos solo los campos relevantes para la entrega
...   { $project: { nombre: 1, salario: 1, departamento_id: 1, _id: 0 } }
... ])
...
[ { nombre: 'Frank', salario: 2000, departamento_id: 2 } ]
```

Fig. 22. Reconexión y resincronización del nodo mongo3 al Replica Set.

ENTREGABLES

CAPTURA rs.status() (mongo3 apagado)

```
rs0 [direct: primary] escuela> rs.status()
```

Fig. 23. Comando para verificar el estado del Replica Set.

```
members: [
  {
    _id: 0,
    name: 'mongo1:27017',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 1041,
    optime: { ts: Timestamp({ t: 1759352827, i: 1 }), t: Long('1') },
    optimeDate: ISODate('2025-10-01T21:07:07.000Z'),
    lastAppliedWallTime: ISODate('2025-10-01T21:07:07.340Z'),
    lastDurableWallTime: ISODate('2025-10-01T21:07:07.340Z'),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1759351847, i: 1 }),
    electionDate: ISODate('2025-10-01T20:50:47.000Z'),
    configVersion: 1,
    configTerm: 1,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: 'mongo2:27017',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 999,
    optime: { ts: Timestamp({ t: 1759352827, i: 1 }), t: Long('1') },
    optimeDurable: { ts: Timestamp({ t: 1759352827, i: 1 }), t: Long('1') },
    optimeDate: ISODate('2025-10-01T21:07:07.000Z'),
    optimeDurableDate: ISODate('2025-10-01T21:07:07.000Z'),
    lastAppliedWallTime: ISODate('2025-10-01T21:07:07.340Z'),
    lastDurableWallTime: ISODate('2025-10-01T21:07:07.340Z'),
    lastHeartbeat: ISODate('2025-10-01T21:07:15.977Z'),
    lastHeartbeatRecv: ISODate('2025-10-01T21:07:14.937Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: '',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 1,
    configTerm: 1
  },
```

Fig. 24. Estado detallado del Replica Set con un nodo fuera de servicio.

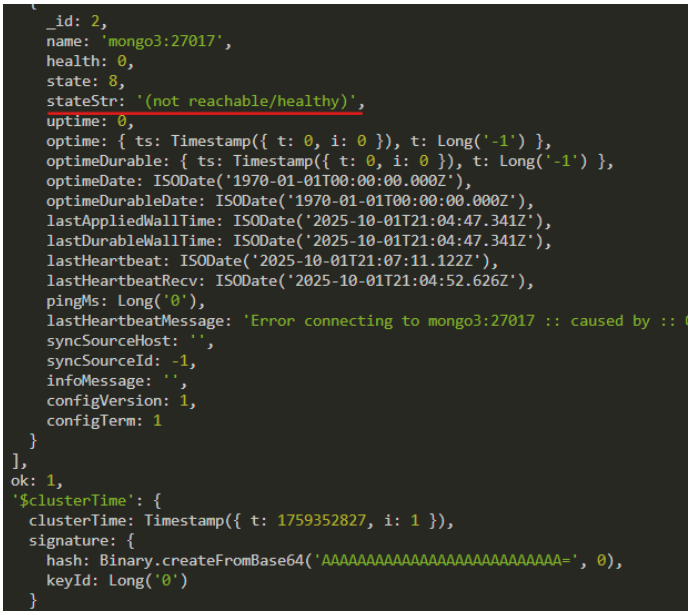

Fig. 25. Información de los miembros del Replica Set durante la prueba de resiliencia.

BONUS

sh.status() (Bonus)

- Solo aplica si se tiene un sharded cluster con mongos.
- El comando sh.status() corresponde a los entornos shardeados, donde los datos se dividen entre distintos shards para balancear la carga y escalar horizontalmente. Dado que en este caso se trabaja únicamente con un ReplicaSet y no con un clúster shardeado, el uso de sh.status() no resulta aplicable.

explain() (Bonus).- Para ver cómo MongoDB resuelve las consultas:

```
db.empleados.find({ salario: { $gt: 2000 } }).explain()
```

```
rs0 [direct: primary] escuela> db.empleados.find({ salario: { $gt: 2000 } }).explain()
...
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'escuela.empleados',
    indexFilterSet: false,
    parsedQuery: { salario: { '$gt': 2000 } },
    queryHash: 'C07C90B3',
    planCacheKey: 'C07C90B3',
    optimizationTimeMillis: 0,
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { salario: { '$gt': 2000 } },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  command: {
    find: 'empleados',
    filter: { salario: { '$gt': 2000 } },
    '$db': 'escuela'
  },
  serverInfo: {
    host: '613392d8990b',
    port: 27017,
    version: '7.0.24',
    gitVersion: '332b0e6c30fdc41a0228dc55657e2e0784b0fe24'
  },
```

Fig. 26. Análisis de rendimiento de consulta usando explain().

```
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
    internalQueryFrameworkControl: 'forceClassicEngine'
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1759353017, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1759353017, i: 1 })
}
```

Fig. 27. Comprobación de transacciones en el clúster distribuido.

2.7 Resultados obtenidos

- Configuración exitosa de un Replica Set de MongoDB con tres nodos (mongo1, mongo2, mongo3) utilizando Docker.
- Importación correcta de los conjuntos de datos: departamentos.json, empleados.json y ventas.json.
- Ejecución de consultas complejas utilizando pipelines de agregación, tanto con métodos de ventana ($setWindowFields) como con enfoques tradicionales ($lookup, $group).

- Demostración de la tolerancia a fallos: al detener el nodo mongo3, el clúster continuó operando sin interrupciones gracias a la replicación.
- Uso de explain() para analizar el rendimiento y la estrategia de ejecución de una consulta simple.

2.8 Habilidades blandas

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

2.9 Conclusiones

- Docker facilita la implementación de entornos de bases de datos distribuidas, permitiendo una configuración reproducible y aislada.
- MongoDB Replica Set garantiza alta disponibilidad y continuidad del servicio ante fallos en uno o más nodos.
- Las consultas con etapas de ventana ($setWindowFields) ofrecen una alternativa eficiente y legible para cálculos agregados sin necesidad de subconsultas complejas.
- La resiliencia del sistema se verifica mediante la capacidad de mantener operaciones de lectura y escritura incluso con la pérdida de un nodo secundario.
- El uso de explain() es fundamental para optimizar consultas y entender el comportamiento interno de MongoDB.

2.10 Recomendaciones

- Utilizar siempre Replica Sets en entornos productivos para asegurar la disponibilidad y durabilidad de los datos.
- Profundizar en el uso de índices y explain() para mejorar el rendimiento de consultas en colecciones grandes.

2.11 Referencia Bibliográfica

[1] M. García, “Deploy a MongoDB Cluster with Docker”, DEV Community. Consultado: el 6 de octubre de 2025. [En línea]. Disponible en: https://dev.to/mattdark/deploy-a-mongodb-cluster-with-docker-1fal

[2] “Deploying A MongoDB Cluster With Docker”, MongoDB. Consultado: el 6 de octubre de 2025. [En línea]. Disponible en: https://www.mongodb.com/resources/products/compatibilities/deploying-a-mongodb-cluster-with-docker

[3] “MongoDB queries: cómo funcionan las consultas de datos”, IONOS Digital Guide. Consultado: el 6 de octubre de 2025. [En línea]. Disponible en: https://www.ionos.com/es-us/digitalguide/paginas-web/desarrollo-web/mongodb-queries/