**Guía Didáctica de MongoDB: Local, Atlas y CRUD en Consola y Python**

**📂 1. Introducción a MongoDB y NoSQL**

* **Base de datos NoSQL:** no usa tablas, puede guardar datos como documentos JSON, clave/valor, grafos, etc.
* **Ejemplos:**
  + Documento: { nombre: "Ana", edad: 25 }
  + Clave/valor: "usuario123": "Juan Pérez"
  + Grafo: (Juan) ---es amigo de---> (Ana)

**📚 2. Crear Base de Datos con MongoDB Local (MongoDB Compass y Consola)**

**2.1 Con MongoDB Compass**

1. Abrir Compass.
2. Crear conexión a localhost:27017.
3. Ir a "Create Database":
   * Nombre de base: base1
   * Nombre de colección: libros
   * Insertar documentos manualmente.

**2.2 Con Consola (Mongo Shell)**

use base1

db.libros.insertOne({

codigo: 1,

nombre: 'El aleph',

autor: 'Borges',

editoriales: ['Planeta', 'Siglo XXI']

})

**Mostrar datos:**

db.libros.find().pretty()

**💡 3. Realizar CRUD desde la Consola**

**Insertar documentos:**

db.libros.insertOne({...})

db.libros.insertMany([{...}, {...}])

**Leer documentos:**

db.libros.find()

db.libros.find({precio: {$gte: 50}})

**Modificar documentos:**

db.libros.updateOne({\_id: 1}, {$set: {precio: 30}})

**Borrar documentos:**

db.libros.deleteOne({\_id: 1})

db.libros.deleteMany({precio: {$lt: 20}})

**📅 4. Ejercicios de CRUD por consola**

**✍️ Ejercicio 1: Colección "clientes"**

use base1

db.clientes.drop()

db.clientes.insertOne({ \_id: 1, nombre: 'Ana', ciudad: 'Rosario' })

db.clientes.insertOne({ \_id: 2, nombre: 'Luis', ciudad: 'Córdoba' })

db.clientes.insertMany([

{ \_id: 3, nombre: 'Elena', ciudad: 'Mendoza' },

{ \_id: 4, nombre: 'Pablo', ciudad: 'Buenos Aires' }

])

db.clientes.find()

db.clientes.updateOne({\_id: 2}, {$set: {ciudad: 'Santa Fe'}})

db.clientes.deleteOne({\_id: 3})

db.clientes.find().pretty()

**✍️ Ejercicio 2: Colección "productos"**

use base2

db.productos.drop()

db.productos.insertMany([

{ \_id: 1, nombre: 'Teclado', precio: 3500 },

{ \_id: 2, nombre: 'Mouse', precio: 2500 },

{ \_id: 3, nombre: 'Monitor', precio: 15000 }

])

db.productos.find()

db.productos.updateOne({\_id: 1}, {$set: {precio: 3000}})

db.productos.deleteMany({precio: {$gt: 10000}})

db.productos.find()

**✍️ Ejercicio 3: Colección "empleados"**

use base3

db.empleados.drop()

db.empleados.insertOne({ \_id: 1, nombre: 'Marcos', puesto: 'Contador', salario: 80000 })

db.empleados.insertMany([

{ \_id: 2, nombre: 'Lucia', puesto: 'Diseñadora', salario: 70000 },

{ \_id: 3, nombre: 'Diego', puesto: 'Programador', salario: 90000 }

])

db.empleados.find()

db.empleados.updateMany({}, {$inc: {salario: 10000}})

db.empleados.deleteOne({nombre: 'Lucia'})

db.empleados.find().pretty()

**✍️ Ejercicio 4: Colección "cursos"**

use base4

db.cursos.drop()

db.cursos.insertMany([

{ \_id: 1, nombre: 'Matemática', profesor: 'Laura', alumnos: 25 },

{ \_id: 2, nombre: 'Historia', profesor: 'Carlos', alumnos: 30 }

])

db.cursos.find()

db.cursos.updateOne({nombre: 'Matemática'}, {$inc: {alumnos: 5}})

db.cursos.deleteOne({nombre: 'Historia'})

db.cursos.find().pretty()

**✍️ Ejercicio 5: Colección "vehiculos"**

use base5

db.vehiculos.drop()

db.vehiculos.insertMany([

{ \_id: 1, marca: 'Toyota', modelo: 'Corolla', año: 2020 },

{ \_id: 2, marca: 'Ford', modelo: 'Focus', año: 2018 },

{ \_id: 3, marca: 'Honda', modelo: 'Civic', año: 2022 }

])

db.vehiculos.find()

db.vehiculos.updateOne({modelo: 'Focus'}, {$set: {año: 2019}})

db.vehiculos.deleteMany({año: {$lt: 2020}})

db.vehiculos.find().pretty()

**🚀 5. Crear una base en MongoDB Atlas**

1. Ir a <https://www.mongodb.com/cloud/atlas>
2. Crear cuenta y clúcster gratuito.
3. Crear base de datos y colección.
4. Ir a "Database Access" y crear usuario.
5. Ir a "Network Access" y permitir IP actual.
6. Copiar la URL de conexión y usarla en tu app:

mongodb+srv://usuario:clave@cluster0.xxxxxx.mongodb.net/test

**📊 6. Replica Set y Sharding**

**Diagrama Replica Set**

**Replica Set**

* Grupo de servidores: 1 primario y varios secundarios.
* Si el primario falla, otro lo reemplaza automáticamente.
* Protocolos:
  + **Heartbeat**: verifica que los servidores estén vivos.
  + **Oplog Replication**: copia los cambios del primario.

**🧠 ¿Qué es un Replica Set en MongoDB?**

Un **Replica Set** es un **grupo de servidores** que trabajan juntos para **guardar una copia exacta de tus datos**. Esto se hace para tener **seguridad, disponibilidad y tolerancia a fallos**.

**🖧 ¿Cómo funciona?**

Imaginemos que tenemos tres servidores:

* 🟢 **Servidor Primario**: el principal. Todas las escrituras y lecturas importantes van acá.
* 🔵 **Servidor Secundario 1**
* 🔵 **Servidor Secundario 2**

Estos dos últimos son copias del primario. Están ahí por si el primario **se cae** o **necesita reiniciarse**.  
Si eso pasa, uno de los secundarios **toma el control automáticamente**.

Este sistema se llama **Replica Set**.

**🔄 ¿Cómo se mantienen sincronizados?**

Usan dos protocolos:

* ❤️‍🔥 **Heartbeat**: Los servidores se "preguntan" constantemente:

“¿Estás vivo?”

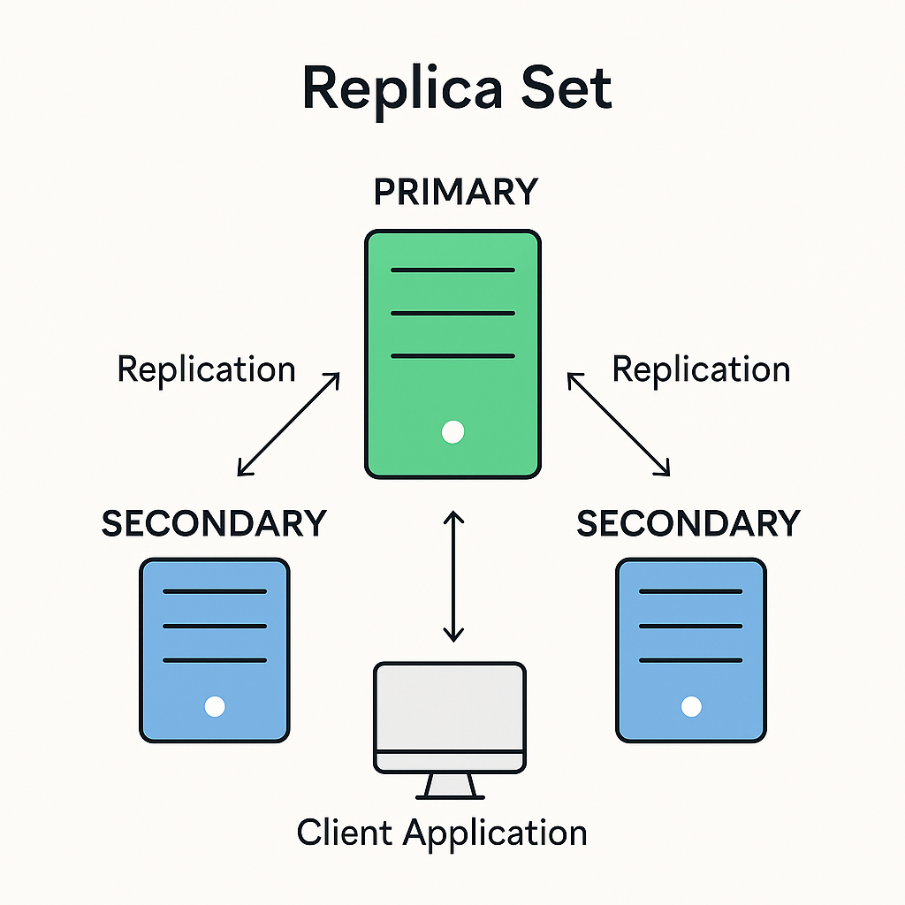
* 📜 **Oplog Replication**: Se copian todos los cambios que hace el primario en una lista de operaciones. Los secundarios leen esa lista y se actualizan.

**✅ Ventajas**

1. **Alta disponibilidad**: si un servidor falla, otro toma el control.
2. **Copia de seguridad automática**: siempre hay réplicas de los datos.
3. **Escalabilidad de lectura**: podés leer desde los secundarios para repartir la carga.

**⚠️ Limitaciones**

1. Si hay **mucha demanda**, el sistema puede **saturarse**.
2. No todos los tipos de lectura se pueden hacer desde los secundarios.
3. Requiere **más recursos y configuración**.



**Sharding**

* Divide los datos en varios **shards** (fragmentos).
* Usa mongos para enrutar las consultas.
* **Config Server** almacena metadatos sobre los shards.

**🧩 ¿Qué es Sharding?**

**Sharding** es una técnica que usa MongoDB para **dividir los datos en varios grupos llamados "Shards"**. Esto permite que la base de datos pueda **manejar muchísimos datos y muchas consultas al mismo tiempo**.

**🚀 ¿Por qué usar Sharding?**

Imaginá que tenés una biblioteca con millones de libros. Si solo tenés una persona buscando libros (un solo servidor), se tarda mucho.  
Pero si dividís los libros en diferentes sectores y ponés varias personas buscando, **todo se vuelve más rápido**.  
Eso hace el **sharding**.

**🏗️ ¿Cómo funciona?**

**1️⃣ Shards**

Cada **Shard** es como una mini base de datos (puede ser un Replica Set). Guarda **una parte** de la información total.

**2️⃣ mongos (Query Router)**

Es el **encargado de recibir las consultas** de la aplicación y decir:

“Esta consulta va al Shard 2, esta otra al Shard 1...”

Cada aplicación se conecta a un **mongos**, no a los shards directamente.

**3️⃣ Config Servers**

Son los cerebritos del sistema. Guardan los **metadatos**, es decir, dónde está cada parte de la información.

🔴 ¡Cuidado!: Si se pierde el **config server**, podés perder acceso a los datos, aunque los shards sigan existiendo.

**🔄 Resumen del flujo:**

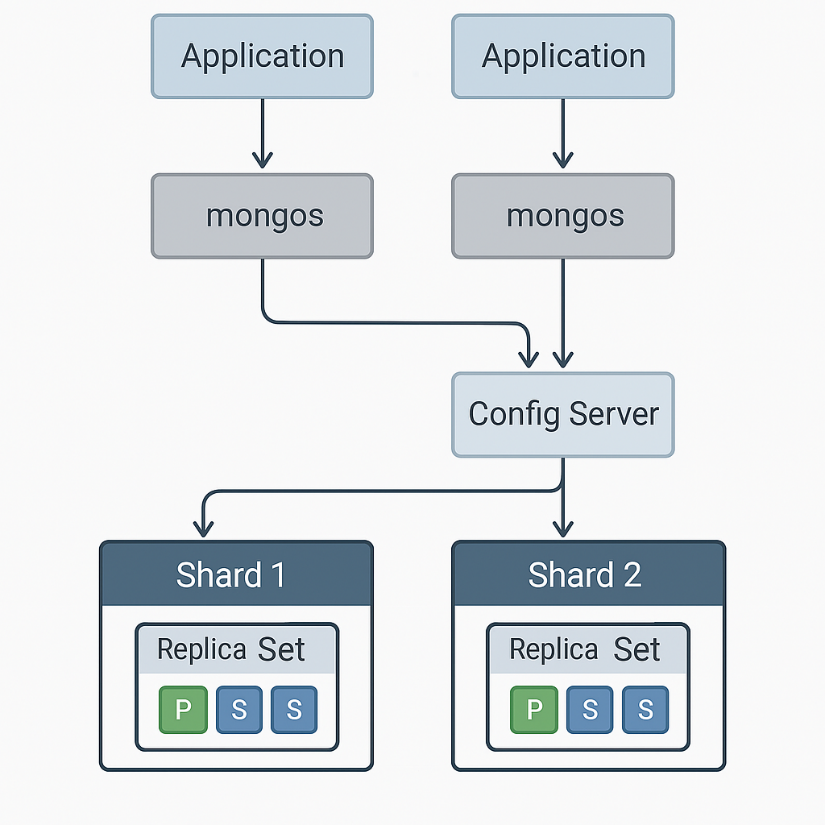
1. 📱 La aplicación se conecta a **mongos**
2. 🧭 **mongos** consulta al **Config Server** dónde está la info.
3. 🚀 Lanza la consulta al **Shard** correcto.
4. ✅ Devuelve el resultado a la aplicación.

**✅ Ventajas de Sharding**

* Escala horizontalmente: se agregan más servidores, no necesitas hacer uno gigante.
* Mejora rendimiento con muchos datos y usuarios.
* Cada shard puede ser un Replica Set (resistente a fallos).

**⚠️ Limitaciones**

* Es más complejo de configurar y mantener.
* Dependencia del mongos y config server.
* Si se mal distribuyen los datos (mal sharding key), puede haber cuellos de botella.



**📆 8. Sugerencia de trabajo final**

Crear una aplicación CRUD para una biblioteca que permita:

* Agregar libros
* Consultarlos por autor o precio
* Editar precio y cantidad
* Eliminar libros agotados
* Usar Mongo local y luego migrar a Atlas