

UD 2 - Procesado y Presentación Datos Almacenados - MongoDB

El nombre **MongoDB** proviene de la palabra inglesa "**homongous**" que significa enorme. **MongoDB es, por tanto, un sistema de base de datos NoSQL, open source, orientado a documentos y escrito en lenguaje C++.** Este sistema de bases de datos está disponible no solo para múltiples plataformas y sistemas operativos (Windows, Linux, OS X) sino también como servicio empresarial en la nube y se puede integrar con otros servicios como, por ejemplo, Amazon Web Services (AWS).

1. Características

Tal y como se ha presentado anteriormente, MongoDB es un sistema de bases de datos documental u orientado a documentos. Esta orientación hace que los datos se almacenen de forma estructurada en forma de documentos, los cuales se acoplan sin problema en los tipos de datos utilizados por los lenguajes de programación. Además, esta concepción hace que una base de datos MongoDB disponga de un esquema dinámico y fácilmente modificable.

En este apartado se pretende poner de relieve **las principales características de MongoDB** que hacen que sea en la actualidad uno de los sistemas de bases de datos NoSQL más utilizados tanto en el ámbito académico como en el profesional.

- **Alto rendimiento:** Gracias a la definición de los documentos y la creación de índices que hace que las lecturas y escrituras se realicen de forma más rápida.
- **Alta disponibilidad:** MongoDB dispone de servidores replicados con restablecimiento automático maestro.
- **Fácil escalabilidad:** Permitiendo distribuir colecciones de documentos entre diferentes máquinas de forma muy sencilla.
- **Indexación:** Similar al concepto de índice en una base de datos relacional, permite crear índices e índices secundarios para mejorar el rendimiento de las consultas.
- **Consultas ad-hoc:** Al igual que en una base de datos relacional, MongoDB da soporte a la búsqueda por campos, consultas de rangos, uso de expresiones regulares... A todo lo anterior, se añade la posibilidad de ejecutar y devolver una función JavaScript definida por el programador.
- **Replicación:** Siguiendo el modelo maestro-esclavo. El maestro puede ejecutar comandos de lectura y escritura mientras que el esclavo solo tiene acceso de lectura y la posibilidad

de realizar copias de seguridad. En caso de que el maestro caiga, el esclavo puede elegir un nuevo maestro para mantener el servicio de replicación.

- **Balanceo de carga:** MongoDB es capaz de ejecutarse en múltiples servidores, pudiendo balancear la carga y/o duplicar los datos para mantener el correcto funcionamiento del sistema aunque se produzca un fallo hardware.
- **Almacenamiento de archivos:** Todo lo anterior, facilita que este sistema de base de datos pueda ser usado con un sistema de archivos GridFS con balanceo de carga y replicación.
- **Agregación:** Proporciona operadores de agregación y la posibilidad de utilizar funciones MapReduce para el procesamiento de datos por lotes.
- **Ejecución de Javascript del lado del servidor:** Es posible realizar consultas utilizando JavaScript, de forma que éstas son enviadas directamente a la base de datos para ser ejecutadas.

Todas estas características hacen que, en la actualidad, MongoDB sea uno de los principales sistemas de bases de datos NoSQL elegidos en multitud de aplicaciones como: **almacenamiento y registro de eventos, comercio electrónico, juegos, aplicaciones móviles, almacenes de datos, gestión de estadísticas en tiempo real y cualquier aplicación que requiera llevar a cabo analíticas sobre grandes volúmenes de datos.**

2. Conceptos básicos

Aunque MongoDB es un sistema de bases de datos NoSQL con todo lo que ello implica, también ofrece toda la funcionalidad de la que disponen las bases de datos relacionales. Sin embargo, **la estructura de una base de datos MongoDB difiere de la de una base de datos relacional.**

En un servidor MongoDB es posible crear tantas bases de datos como se desee. El concepto de **base de datos** es en este caso equivalente al de **base de datos** en los sistemas relacionales. Una vez creada, la base de datos estará compuesta por una o más **colecciones**. El término colección en el ámbito NoSQL es el equivalente al concepto de **tabla** en los sistemas relacionales.

Cada colección, por tanto, estará formada por un conjunto de **documentos** (o incluso ninguno, en cuyo caso la colección estaría vacía). El concepto de documento en NoSQL se corresponde con el concepto de **registro** en una base de datos relacional. Un documento estará compuesto por una serie de **campos**, al igual que lo están los registros de una base de datos relacional.

En el ámbito NoSQL, y más concretamente en MongoDB, cada documento viene dado por un archivo **JSON** <http://www.json.org/> (*BSON, en realidad, que veremos justo en el siguiente punto*) en el cual, siguiendo una estructura clave-valor se especifican las características de cada documento. El siguiente código muestra un ejemplo de documento que define un barco.

```
1  {
2      name: 'USS Enterprise-D',
3      operator: 'Starfleet',
4      type: 'Explorer',
5      class: 'Galaxy',
6      crew: 750,
7      codes: [10, 11, 12]
8  }
```

2.1 BSON

BSON es un formato de intercambio de datos usado principalmente para su almacenamiento y transferencia en la base de datos **MongoDB**. Es una representación binaria de estructuras de datos y mapas. El nombre BSON está basado en el término **JSON** y significa **Binary JSON (JSON Binario)**. Consulta su especificación en su página oficial <https://bsonspec.org/>



BSON fue diseñado para tener las siguientes características

- **Ligero:** Mantener la sobrecarga espacial al mínimo es importante para cualquier formato de representación de datos, especialmente cuando se utiliza a través de la red.
- **Transitable:** BSON está diseñado para ser recorrido fácilmente. Esta es una propiedad vital en su papel como la principal representación de datos para MongoDB.
- **Eficiente:** La codificación de datos a BSON y la decodificación desde BSON puede realizarse muy rápidamente en la mayoría de lenguajes debido al uso de tipos de datos C.

Un **objeto BSON** consiste en una lista ordenada de elementos. Cada elemento consta de un campo nombre, un tipo y un valor. Los nombres son de tipo String y los tipos pueden ser:

Type	Number	Alias	Notes
Double	1	"double"	
String	2	"string"	
Object	3	"object"	
Array	4	"array"	
Binary data	5	"binData"	
Undefined	6	"undefined"	Deprecated.
ObjectId	7	"objectId"	
Boolean	8	"bool"	
Date	9	"date"	
Null	10	"null"	
Regular Expression	11	"regex"	
DBPointer	12	"dbPointer"	Deprecated.
JavaScript	13	"javascript"	
Symbol	14	"symbol"	Deprecated.
32-bit integer	16	"int"	
Timestamp	17	"timestamp"	
64-bit integer	18	"long"	
Decimal128	19	"decimal"	
Min key	-1	"minKey"	
Max key	127	"maxKey"	

Figura BSON 1: Tipos de datos BSON. (Fuente: MongoDB)

Un ejemplo de un objeto BSON podría ser

```

1 var mydoc = {
2     _id: ObjectId("5099803df3f4948bd2f98391"),
3     name: { first: "Alan", last: "Turing" },
4     birth: new Date('Jun 23, 1912'),
5     death: new Date('Jun 07, 1954'),
6     contribs: [ "Turing machine", "Turing test", "Turingery" ],
7     views : NumberLong(1250000)
8 }
```

Donde los campos del ejemplo anterior tienen los siguientes tipos de datos:

- **_id** contiene un `ObjectId`.
- **name** contiene un documento embebido que contiene los campos **first** y **last**.
- **birth** y **death** mantienen valores del tipo `Date`.
- **contribs** contiene un `string`.
- **views** tiene un valor del tipo `NumberLong`.

Además, BSON también contiene expresiones que permiten la representación de tipos de datos que no forman parte de la especificación JSON. Puedes consultarlos todos en <https://www.mongodb.com/docs/manual/reference/bson-types/>

De todos estos tipos de datos, en MongoDB debemos hacer mayor consideración a estos particulares:

Binary Data

El valor de un Binary Data `bindata` es un array de bytes. Un valor de `bindata` tiene un subtipo que indica cómo interpretar los datos binarios. La siguiente tabla muestra los subtipos.

Number	Description
0	Generic binary subtype
1	Function data
2	Binary (old)
3	UUID (old)
4	UUID
5	MD5
6	Encrypted BSON value
7	Compressed time series data <i>New in version 5.2.</i>
8	Sensitive data, such as a key or secret. MongoDB does not log literal values for binary data with subtype 8. Instead, MongoDB logs a placeholder value of <code>###</code> .
9	Vector
128	Custom data

Figura BSON 2: Subtipos de datos BSON Binary Data. (Fuente: MongoDB)

ObjectIds

Los ObjectIds `object` son pequeños, probablemente únicos, rápidos de generar y ordenados. Los valores de ObjectId tienen una longitud de 12 bytes y constan de:

- 4-byte timestamp, que representa la creación del ObjectId, medida en segundos desde la época de Unix.
- Un valor aleatorio de 5 bytes generado una vez por proceso. Este valor aleatorio es exclusivo de la máquina y del proceso.
- Un contador incremental de 3 bytes, inicializado a un valor aleatorio.



Note

En MongoDB, cada documento almacenado en una colección **requiere un campo _id único que actúa como clave principal**. Si un documento insertado omite el campo _id, el controlador MongoDB genera automáticamente un ID de objeto para el campo _id.

String

BSON String `string` son UTF-8. En general, los controladores para cada lenguaje de programación convierten del formato de cadena del lenguaje a UTF-8 al serializar y deserializar BSON. Esto hace posible almacenar la mayoría de los caracteres internacionales en cadenas BSON con facilidad. Además, las consultas \$regex de MongoDB admiten UTF-8 en la cadena de expresiones regulares.

Timestamp

BSON tiene un tipo `timestamp` especial para uso interno de MongoDB y no está asociado con el estándar Tipo Date. Este tipo de marca de tiempo interna es un valor de 64 bits donde:

- los 32 bits más significativos son un valor `time_t` (segundos desde la época Unix)
- los 32 bits menos significativos son un `ordinal` incremental para operaciones dentro de un segundo determinado.

2.2 Document Limitations



Info

Puedes consultar toda la información relacionada con documentos en
<https://www.mongodb.com/docs/manual/core/document/>

Límite de tamaño del documento

El tamaño máximo de documento BSON es de 16MB (ayuda a garantizar que un solo documento no pueda utilizar una cantidad excesiva de RAM ni ancho de banda). Para almacenar documentos que superen el tamaño máximo, MongoDB proporciona la API GridFS.

Orden de campos del documento

A diferencia de los objetos JavaScript, los campos de un documento BSON están ordenados. Pero no asegura que el orden de los campos se respete

Case sensitive

Es sensible a las mayúsculas y minúsculas

2.3 Comandos de Administración

Para la administración del sistema de bases de datos, MongoDB pone a disposición de los usuarios las siguientes utilidades:

- **mongo**: Se trata del shell interactivo de MongoDB que permite insertar, eliminar, actualizar datos y realizar consultas, además de replicar la información, apagar servidores y ejecutar código JavaScript.
- **mongostat**: Herramienta de línea de comandos que muestra las estadísticas de una instancia en ejecución de MongoDB
- **mongotop**: Herramienta de línea de comandos que muestra la cantidad de tiempo empleado en la lectura y escritura de datos por parte de la instancia en ejecución
- **mongosniff**: Herramienta de línea de comandos que permite hacer un rastreo del tráfico de la red que va desde y hacia MongoDB.
- **mongoimport/mongoexport**: Herramienta de línea de comandos para importar y exportar contenido en o desde .json, .csv o .tsv entre otros.
- **mongodump/mongorestore**: Herramienta de línea de comandos para la creación de una exportación binaria del contenido de la base de datos.



BSON => JSON

Si necesitamos transformar un fichero BSON a JSON, usaríamos el comando (bsondump) [<https://www.mongodb.com/docs/database-tools/bsondump/>]:

```
1 bsondump file.bson > file.json
```

3. Instalación y uso

3.1 Docker

Instalación

Instalamos Docker Desktop. Sigue los pasos de la documentación oficial de tu Sistema Operativo.



Docker engine vs Docker Desktop on Linux

En Linux, ya tenemos por defecto docker engine para la ejecución de contenedores. Si instalamos docker desktop hay que tener en cuenta que docker desktop trabaja sobre un maquina virtual **KVM**. Esto significa que las imágenes desplegadas en docker engine no están disponibles en docker desktop y viceversa. Para ello podemos "jugar" con el contexto de docker.

Para más información observa la documentación oficial de docker: [What is the difference between Docker Desktop for Linux and Docker Engine](#)

The screenshot shows the Docker documentation website. On the left, there's a sidebar with a tree structure under 'Manuals': Docker Build, Docker Build Cloud, Docker Compose, Docker Desktop (which is expanded), and Install. Under Install, there are sections for Mac, Windows, and Linux, with Linux being the selected option. The main content area has a breadcrumb navigation: Home / Manuals / Docker Desktop / Install / Linux. The main title is 'Install Docker Desktop on Linux'. Below it, there's a section titled 'Docker Desktop terms' and another about 'Commercial use of Docker Desktop'. A note states that using Docker Desktop in larger enterprises requires a paid subscription. Another section, 'Important', explains that Docker Desktop runs a VM which creates a custom docker context named 'desktop-linux' on startup. It also notes that images and containers deployed on the Linux Docker Engine before installation are not available in Docker Desktop for Linux. At the bottom, there's a question box asking 'What is the difference between Docker Desktop for Linux and Docker Engine?'.

Figura Docker 1. Docker Desktop on Linux. (Fuente: Docker)

Para más información observa los siguientes recursos:

- [Why does Docker Desktop for Linux run a VM?](#)
- [Switching between Docker Desktop and Docker Engine](#). Docker Docs
- [Docker Desktop vs. Docker Engine en Ubuntu LTS](#). Video explicativo



Ubuntu 24.04 LTS is not yet supported

La última versión de Ubuntu 24.04 LTS aún no es compatible. Docker Desktop no se iniciará. Debido a un cambio en la forma en que la última versión de Ubuntu [restringe los espacios de nombres](#) sin privilegios.

Para solucionar este restricción debemos cambiar la configuración de esta restricción. Necesitamos ejecutarlo al menos una vez cada vez que iniciemos el sistema.

```
1 sudo sysctl -w kernel.apparmor_restrict_unprivileged_userns=0
```

Prerequisites

To install Docker Desktop successfully, you must:

- Meet the [general system requirements](#).
- Have a 64-bit version of either the LTS version Ubuntu Jammy Jellyfish 22.04, or the current non-LTS version. Docker Desktop is supported on `x86_64` (or `amd64`) architecture.

Note

The latest Ubuntu 24.04 LTS is not yet supported. Docker Desktop will fail to start. Due to a change in how the latest Ubuntu release restricts the unprivileged namespaces,

`sudo sysctl -w kernel.apparmor_restrict_unprivileged_userns=0` needs to be run at least once. Refer to the [Ubuntu Blog](#) for more details.

- For non-Gnome Desktop environments, `gnome-terminal` must be installed:

```
$ sudo apt install gnome-terminal
```

Figura Docker 2.Restricted Unprivileged User Namespaces. (Fuente: Docker)

Uso

Vamos a explicar como levantar un contenedor MongoDB con Docker

1. Instalar MondoDB en Docker
2. Vamos a la [imagen oficial de mondodb en docker](#)
3. Observa con detenimiento la información que nos facilita la imagen oficial para el despliegue y configuración de los contenedores
4. Seguimos los pasos, que tienen este **formato**

```
1 docker run -d --network some-network --name some-mongo \
2   -e MONGO_INITDB_ROOT_USERNAME=mongoadmin \
3   -e MONGO_INITDB_ROOT_PASSWORD=secret \
4   mongo:tag
```

En mi caso voy a añadir un **bind mount** para poder luego importar un fichero para el ejemplo. Por tanto sería:

```
1 | docker run -d --name mongodb-bda1 -e MONGO_INITDB_ROOT_USERNAME=admin -e MONGO_INITDB_ROOT_PASSWORD=admin --mount type=bind,src=/home/jaime/BigData/BDA/,dst=/home/bda mongo:latest
```

También podemos crear el contenedor sin bind mount y copiarla directamente con `docker cp`:

```
1 | docker run -d --name mongodb-bda1 -e MONGO_INITDB_ROOT_USERNAME=admin -e MONGO_INITDB_ROOT_PASSWORD=admin mongo:latest
2 | docker cp fichero mongo:/directorio
```

5. Para añadir documentos, podemos importar datos de un dataset `mongoimport`, por ejemplo de kaggle, o restaurar una Base de datos `mongorestore`

Dataset restaurantes1.csv

Descarga [restaurantes1.csv](#)

sampledata (muestra de datos de MongoDB, la misma de Mongo Atlas)

```
1 | wget https://atlas-education.s3.amazonaws.com/sampledata.archive
```

6. Copiamos los archivos que vamos a importar/restaurar

restaurantes1.csv

```
1 | docker cp restaurantes1.csv mongodb-bda1:/tmp
```

sampledata

```
1 | docker cp sampledata.archive mongodb-bda1:/tmp
```

7. Abre una terminal en el contenedor (*puedes abrirla por Docker Desktop también*)

```
1 | docker exec -it mongodb-bda1 sh
```

8. Restauramos/importamos

Importamos restaurantes1.csv

```
1 | mongoimport --db=db_ej1_restaurantes --collection=restaurantes --file=tmp/restaurantes1.csv --authenticationDatabase=admin --username=admin --password=admin
```

Restauramos sampledata

```
1 | mongorestore --archive=/tmp/sampledata.archive --  
authenticationDatabase=admin --username=admin --password=admin  
  
1 | 2024-10-14T14:21:57.418+0000      preparing collections to restore from  
2 | 2024-10-14T14:21:57.429+0000      reading metadata for  
sample_training.zips from archive '/tmp/sampledata.archive'  
3 | 2024-10-14T14:21:57.429+0000      reading metadata for  
sample_geospatial.shipwrecks from archive '/tmp/sampledata.archive'  
4 | 2024-10-14T14:21:57.429+0000      reading metadata for  
sample_mflix.theaters from archive '/tmp/sampledata.archive'  
5 | ...  
6 | 2024-10-14T14:22:05.162+0000      425367 document(s) restored  
successfully. 0 document(s) failed to restore.
```

9. Comprobamos. Entrar a la consola de mongo y autorizarnos

```
1 | mongosh  
2 | use admin  
3 | db.auth("admin","admin")
```

10. Ver las bases de datos

```
1 | show dbs  
  
admin> show dbs  
admin                  100.00 KiB  
config                72.00 KiB  
db_ej1_restaurantes  4.07 MiB  
local                 72.00 KiB  
sample_airbnb          51.87 MiB  
sample_analytics        9.44 MiB  
sample_geospatial       980.00 KiB  
sample_guides          40.00 KiB  
sample_mflix            107.62 MiB  
sample_restaurants      5.61 MiB  
sample_supplies         968.00 KiB  
sample_training          40.55 MiB  
sample_weatherdata      2.39 MiB  
admin> █
```

Figura Docker 3. Mostrar las Bases de datos con Mongo. (Fuente: Propia)

11. Usar la Base de datos (`use nombre_bd`, también sirve para crear una Base de datos)

```
1 | use db_ej1_restaurantes
```

12. Ver las colecciones de la base de datos que estamos usando

```
1 | show collections
```

13. Mostramos los documentos

```
1 | db.restaurantes.find()
```

3.2 Soluciones Mongo

En la actualidad, MongoDB se comercializa mediante tres productos:

- [MongoDB Enterprise Advanced](#), versión de pago con soporte, herramientas avanzadas de monitorización y seguridad, y administración automatizada.
- [MongoDB Community Edition](#), Software gratuito para trabajar on-premise, con versiones para diferentes sistemas operativos.
- [Mongo Atlas](#), como plataforma cloud, con una opción gratuita mediante un cluster de 512MB.

MongoDB Community Edition

Instalación on-promise de MongoDB. Si deseas una instalación en local puedes seguir las instrucciones de la [documentación oficial](#).

3.3 Mongo Atlas

Podemos hacer uso de una solución gratuita cloud de Mongo Atlas ofrecida por MongoDB, con 512MB de RAM compartida y 5GB de almacenamiento

MongoDB Atlas

The multi-cloud developer data platform available on AWS, Azure, and Google Cloud

The screenshot shows the MongoDB Atlas pricing page with three main options:

- Serverless**: from \$0.10/million reads. Includes a "Sign Up" button and a note "(i) Per 1 million reads". Description: For serverless applications with variable or infrequent traffic. Minimal configuration required. Benefits: ✓ Up to 1TB of storage, ✓ Resources scale seamlessly to meet your workload, ✓ Pay only for the operations you run, ✓ Always-on security and backups. View pricing.
- Dedicated**: Recommended, from \$57/month. Includes a "Sign Up" button and a note "(i) Estimated based on \$0.08 per hour". Description: For production applications with sophisticated workload requirements. Advanced configuration controls. Benefits: ✓ 10GB to 4TB of storage, ✓ 2GB to 768GB RAM, ✓ Network isolation and fine-grained access controls, ✓ Multi-region and multi-cloud options available. View pricing.
- Shared**: from \$0/month. Includes a "Try for Free" button and a note "(i) Free forever for free clusters". Description: For learning and exploring MongoDB in a cloud environment. Basic configuration options. Benefits: ✓ 512MB to 5GB of storage, ✓ Shared RAM, ✓ Upgrade to dedicated clusters for full functionality, ✓ No credit card required to start. View pricing.

Figura Atlas 1: MongoDB Atlas. (Fuente: Propia)

Para ello seguimos todos los pasos: [Getting Started MongoDB](#)

1 Create an Atlas account.

[Register for an Atlas account](#) using your Google Account or an email address.

2 Deploy a free cluster.

[Create and deploy a free cluster](#). You can use Atlas free clusters as a small-scale development environment to host your data. Free clusters never expire, and provide access to a [subset](#) of Atlas features.

3 Manage database users for your cluster.

[Manage database users for your cluster](#). For security purposes, Atlas requires clients to authenticate as MongoDB database users to access clusters.

4 Manage the IP access list.

[Manage the list of trusted IP addresses](#). An IP uniquely identifies a device connecting to a network. In Atlas, you can connect to a cluster only from a trusted IP address. Within Atlas, you can create a list of trusted IP addresses, referred to as an IP access list. An IP access list defines the IP addresses that can connect to your cluster and access your data.

5 Connect to your cluster.

[Connect to your cluster](#) using the [mongosh](#), the [Node.js driver](#), the [PyMongo driver](#), or [Compass](#).

6 Insert and view a document.

[Insert a document into your cluster](#) using one of the supported [MongoDB Drivers](#). MongoDB drivers let you interact with your databases programmatically with a supported programming language.

7 Load sample data.

[Load sample data into your Atlas clusters](#). Atlas provides sample data that you can load into your Atlas clusters. You can use this data to quickly get started experimenting with data in MongoDB and using tools such as the [Atlas UI](#) and [MongoDB Charts](#).

You can also generate synthetic data that aligns to your real data's schema. To learn more,

see [Generate Synthetic Data](#).

Figura Atlas 2: MongoDB Atlas. Get Started. (Fuente: Propia)

1. Creamos cuenta
2. Despliega un cluster
 - Creamos una nueva organización

The screenshot shows the MongoDB Atlas 'Create Organization' interface. On the left, a sidebar has 'Organizations' selected. The main area shows the 'Create Organization' form with the 'Name and Service' tab active. The organization name is set to 'BDA_IESGranCapitan'. Below this, the 'Select Cloud Service' section compares two options:

Features	MongoDB Atlas	Cloud Manager
Automated database configuration	✓	✓
Continuous backup and point-in-time recovery	✓	✓
Queryable backup snapshots	✓	✓
Fine grained database monitoring & customizable alerts	✓	✓
Real-time performance panel	✓	✓
Data explorer	✓	✓
Automated cluster lifecycle management	✓	

Figura Atlas 3: MongoDB Atlas. New Organization 1. (Fuente: Propia)

- Configuración de miembros y privilegios de la organización

The screenshot shows the MongoDB Atlas 'Create Organization' interface. On the left, a sidebar menu includes 'PREFERENCES' (Legacy 2FA, Personalization, Invitations, Organizations), with 'Organizations' highlighted in green. The main area has a breadcrumb path '← Organizations' and a title 'Create Organization'. A navigation bar at the top right shows 'Name and Service' (with a green checkmark) and 'Members and Security'. Below this, a section titled 'Add Members and Set Permissions' contains a text input field 'Invite new or existing users via email address...'. A note says 'Give your members access permissions below.' A table lists a member: 'jaimerabasco@iesgrancapitan.org (you)' with a dropdown menu set to 'Organization Owner'. A toggle switch labeled 'Require IP Access List for the Atlas Administration API' is turned on. A note explains: 'This will force all Atlas Administration API operations for your organization to originate from an IP Address added to your Access List by the API Key.' Below the table are 'Back', 'Cancel', and 'Create Organization' buttons, with 'Create Organization' being the active button.

Figura Atlas 4: MongoDB Atlas. New Organization 2. (Fuente: Propia)

- Creamos un proyecto

ORGANIZATION

Projects

Alerts 0

Activity Feed

Settings

Integrations

Access Manager

Billing

Support

Live Migration

Create a Project

Name Your Project

Project0

Add Tags (Optional)

Use tags to efficiently label and categorize your projects. A project can have a maximum of 50 tags. You can modify tags for the project later. [Learn more](#)

Key	Value	Actions
Select a key or enter your own	Select a value or enter your own	Delete
+ Add tag		0 TAGS

Cancel **Next**

Figura Atlas 5: MongoDB Atlas. New Project. (Fuente: Propia)

- Creamos un cluster

Project0

Data Services

Overview

DEPLOYMENT

Database

SERVICES

Atlas Search

Stream Processing

Triggers

Migration

Data Federation

Programmatic Access

SECURITY

Backup

Database Access

Network Access

Advanced

New On Atlas

Goto

CREATE A CLUSTER

Choose your cloud provider, region, and specs.

+ Create

Toolbar

Featured Resources

GENERAL

Get Started with Atlas

Reference MongoDB Documentation

Develop Applications with the Developer Center

Ask the MongoDB Community

New On Atlas

Learn about the latest feature enhancements on Atlas.

Figura Atlas 6: MongoDB Atlas. New Cluster. (Fuente: Propia)

- Desplegamos cluster gratuito

Deploy your cluster

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

M10 **\$0.09/hour**

For production applications with sophisticated workload requirements.

STORAGE	RAM	vCPU
10 GB	2 GB	2 vCPUs

Serverless **\$0.10/1M reads**

For application development and testing, or workloads with variable traffic.

STORAGE	RAM	vCPU
Up to 1TB	Auto-scale	Auto-scale

M0 **Free**

For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM	vCPU
512 MB	Shared	Shared

✓ **Free forever!** Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Name
You cannot change the name once the cluster is created.

Automate security setup i
 Preload sample dataset i

Provider

Region

[I'll do this later](#)

[Go to Advanced Configuration](#)

[Create Deployment](#)

Figura Atlas 7: MongoDB Atlas. Development new Cluster. (Fuente: Propia)

3. Conectamos a la Base de Datos

- Creamos usuario y contraseña.

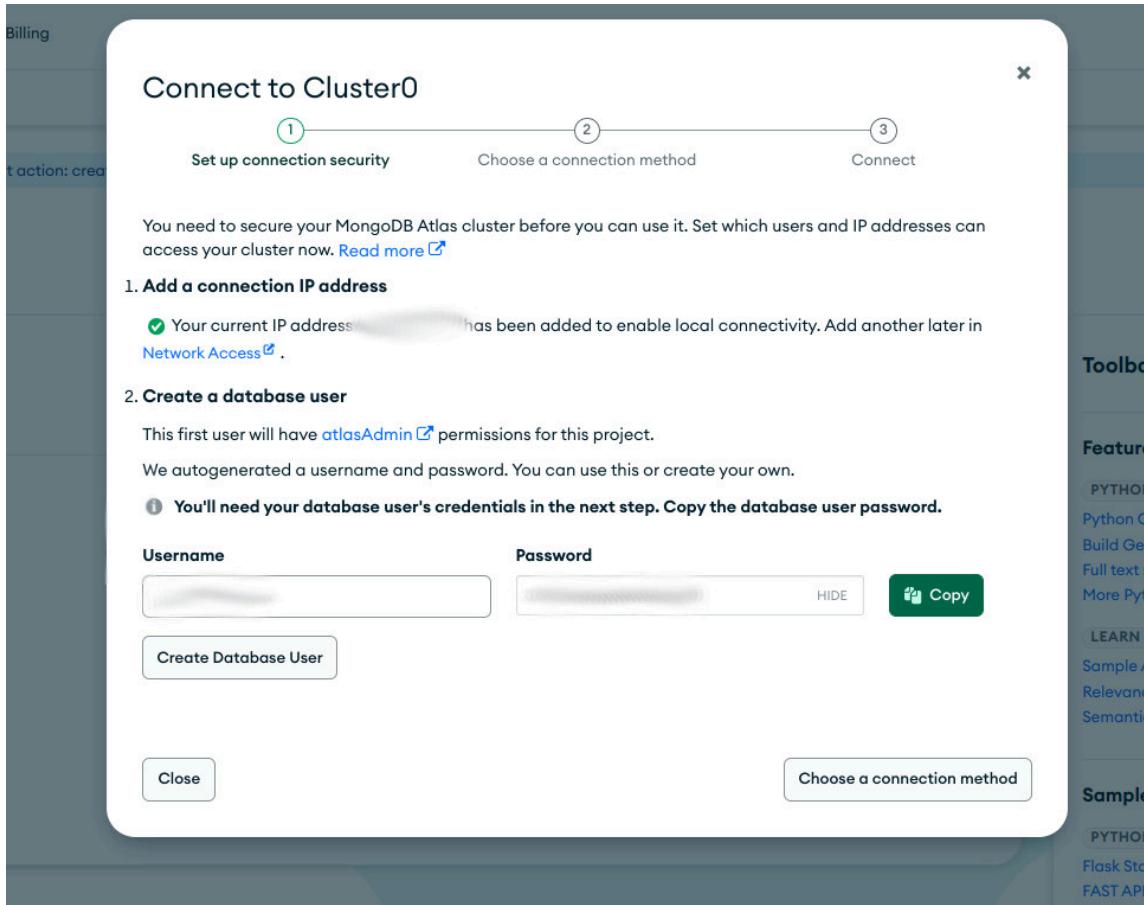


Figura Atlas 8: MongoDB Atlas. New Connect 1. (Fuente: Propia)

- Copiamos las credenciales para un posterior acceso y pulsamos en "Create Database User". Observa que se añadido un usuario y la ip de nuestro equipo para autorizar la conexión.

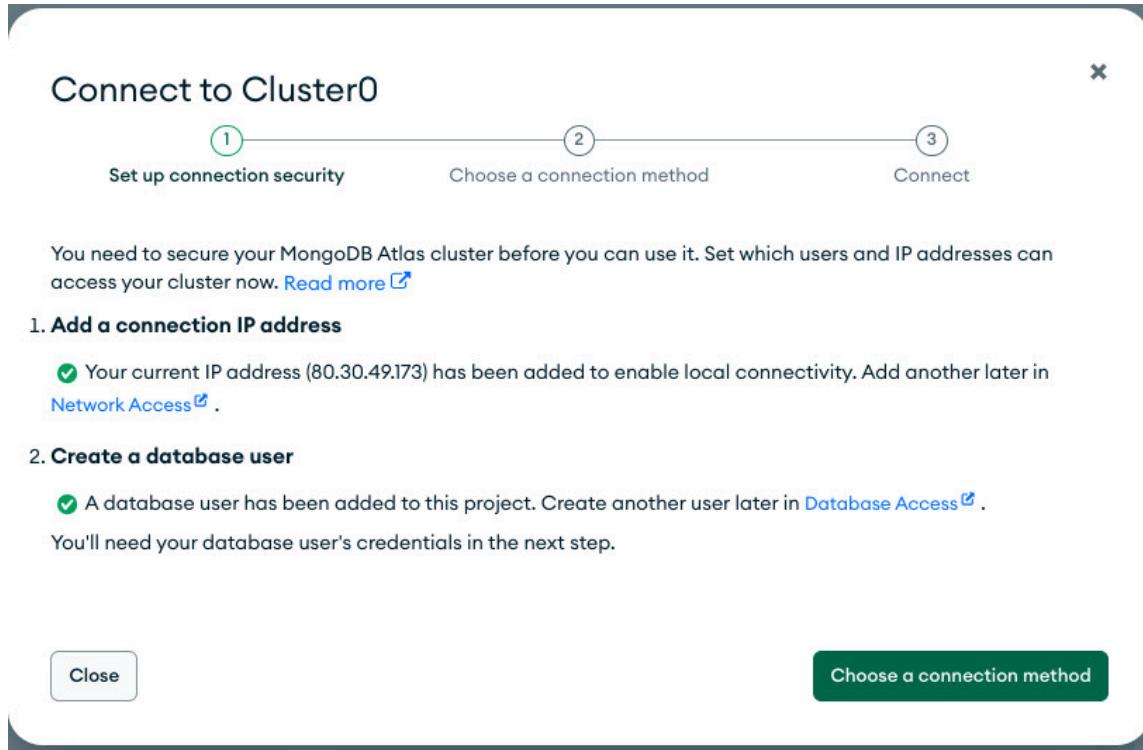


Figura Atlas 9: MongoDB Atlas. New Connect 2. (Fuente: Propia)

- Una vez hemos creado el usuario de la Base de Datos, ya podemos conectarnos a este cluster creado que albergará las Bases de Datos. Tenemos muchas opciones diferentes de conexión, que veremos más adelante.

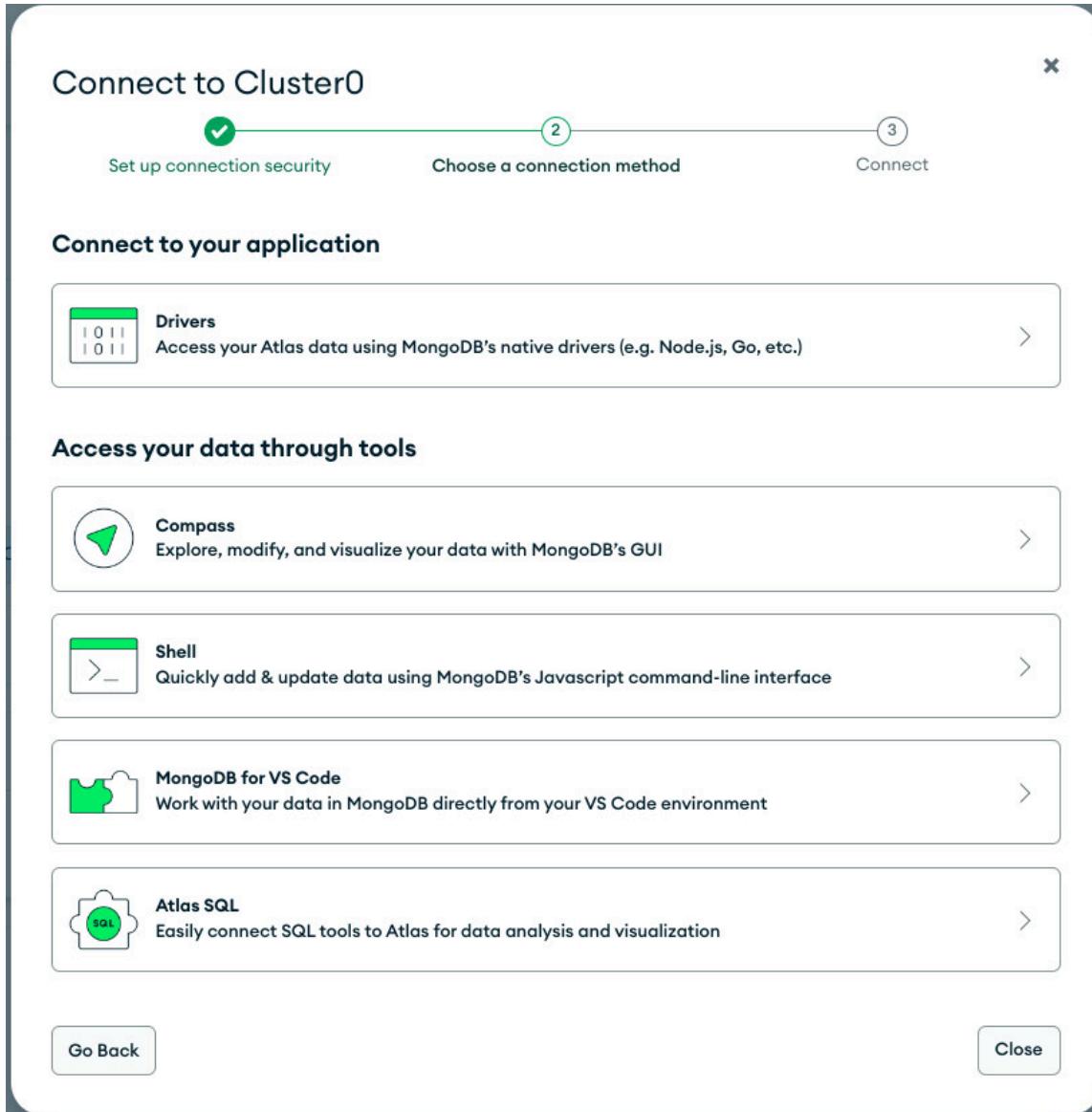


Figura Atlas 10: MongoDB Atlas. New Connect 3. (Fuente: Propia)

4. Añadir tus propios datos. Podemos añadir datos a través de un json pulsando en "Add data" y seguir los pasos correspondientes

The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with sections for Overview, DATABASE (Clusters), SERVICES (Atlas Search, Stream Processing, Triggers, Migration, Data Federation, Programmatic Access), SECURITY (Quickstart, Backup, Database Access, Network Access, Advanced), and a 'Create cluster' button. The main area is titled 'BDA_IESTRANCAPITAN > PROJECTO' and has a large 'Overview' heading. Below it, the 'Clusters' section displays 'Cluster0'. It includes a 'Connect' button and an 'Edit configuration' button. A tooltip for 'Add data' says: 'Looking to interact with Atlas using CLI/IaC tools? View Programmatic Access options.' Below the clusters, there are four main buttons: 'Add data' (with a tooltip), 'Migrate data', 'Load sample data' (which is highlighted with a mouse cursor), and 'Data modeling templates'.

Figura Atlas 11: MongoDB Atlas. Add data. (Fuente: Propia)

5. Cargar una muestra de datos ya establecido por MongoDB:

- Pulsando en "Load sample data"

This screenshot is identical to Figura 11, showing the MongoDB Atlas Overview page. The 'Load sample data' button is now highlighted with a mouse cursor, indicating the user action described in the accompanying text.

Figura Atlas 12: MongoDB Atlas. Load data 1. (Fuente: Propia)

- Esperamos la carga de los datos

The screenshot shows the MongoDB Atlas Database Deployments interface. At the top, a message says "We are deploying your changes (current action: configuring MongoDB)". Below it, the path is "BDA_IESGRANCAPITAN > HELLO_MONGODB". The main title is "Database Deployments". A search bar and a "Create" button are at the top right. The central area shows a progress bar for "Loading your sample dataset...". It includes metrics for "R: 0" and "W: 0" (Last 6 hours), "In: 0.0 B/s" and "Out: 0.0 B/s" (Last 6 hours), and "Data Size: 0.0 B / 512.0 MB (0%)". A tooltip for the "Terminate" button is visible. Below the progress bar, there's a section titled "Visualize Your Data" with a "Dismiss" button and a "Explore Charts" button. At the bottom, there's a table with cluster details: VERSION 6.0.10, REGION AWS / N. Virginia (us-east-1), CLUSTER TIER MO Sandbox (General), TYPE Replica Set - 3 nodes, BACKUPS Inactive, LINKED APP SERVICES None Linked, ATLAS SQL Connect, and ATLAS SEARCH Create Index. A "+ Add Tag" button is also present.

Figura Atlas 13: MongoDB Atlas. Load data 2. (Fuente: Propia)

- Browser Collections. Ya podemos visualizar los datos

The screenshot shows the MongoDB Atlas Browser Collections interface. The left sidebar shows "Project0" and "Data Services". Under "Clusters", "ClusterO" is selected. The main area shows the "sample_airbnb.listingsAndReviews" collection. It displays "DATABASES: 9" and "COLLECTIONS: 23". A "Create Database" button and a "Search Namespaces" input field are present. The collection details show "VERSION: 7.0.12" and "REGION: AWS Paris (eu-west-3)". A "VISUALIZE YOUR DATA" button and a "REFRESH" button are at the top right. Below, there are tabs for "Find", "Indexes", "Schema Anti-Patterns", "Aggregation", and "Search Indexes". A search bar with "Type a query: { field: 'value' }" and a "Reset", "Apply", and "Options" button are shown. The "QUERY RESULTS: 1-20 OF MANY" section lists several document snippets, such as one for a listing with id 10000540 and name "Ribera Charming Duplex".

Figura Atlas 14: MongoDB Atlas. Browser Collections. (Fuente: Propia)

6. Añadir una nueva dirección ip a la lista de ips de acceso al cluster (**En el caso de no haberlo configurado anteriormente o estamos accediendo desde otro lugar**)

- Menú izquierdo => Network Access
- Add IP Address (Derecha)

The screenshot shows the 'Network Access' section of the MongoDB Atlas interface. On the left, there's a sidebar with 'DEPLOYMENT', 'SERVICES', 'SECURITY', and 'Quickstart' sections. Under 'SERVICES', 'Database Access' is selected. Under 'SECURITY', 'Network Access' is selected. The main area is titled 'IP Access List' and shows a table of IP addresses. A yellow banner at the top states: 'You will only be able to connect to your cluster from the following list of IP Addresses:'. The table has columns for 'IP Address', 'Comment', 'Status', and 'Actions'. There are seven entries, all marked as 'Active' with edit and delete buttons.

Figura Atlas 15: MongoDB Atlas. Añadir IP. (Fuente: Propia)

7. Crear otro usuario para la Base de Datos

- Menú izquierdo => Database Access
- Add New Database User (Derecha)

The screenshot shows the 'Quickstart' section of the MongoDB Atlas interface. On the left, 'Quickstart' is selected in the sidebar. The main area is titled 'How would you like to authenticate your connection?'. It offers two options: 'Username and Password' (selected) and 'Certificate'. A tooltip message says: 'We autogenerated a username and password for your first database user in this project using your MongoDB Cloud registration information.' Below this, there's a form to create a database user. It includes fields for 'Username' (set to 'jaimerabasco'), 'Password' (with a placeholder 'Enter password' and an 'Autogenerate Secure Password' button), and a 'Create User' button. There's also a 'Copy' button next to the password field.

Figura 16 Atlas: New user. (Fuente: Propia)

8. Conectarte a tu cluster. Hay varias formas de conexión a tu cluster.

- **A MongoDB driver**, un controlador para comunicarse con su base de datos MongoDB mediante programación. Soporta varios lenguajes de programación.

- [MongoDB Compass](#), una GUI para sus datos de MongoDB. Puede utilizar Compass para explorar, modificar y visualizar sus datos.
- [The MongoDB Shell](#), una interfaz de línea de comandos interactiva para MongoDB. Puedes usar mongosh para insertar e interactuar con datos en su clúster Atlas.
- [MongoDB for VS Code](#). Para trabajar MongoDB en VS Code
- [Atlas SQL](#). Para conectar fácilmente a Atlas para visualización y análisis de datos

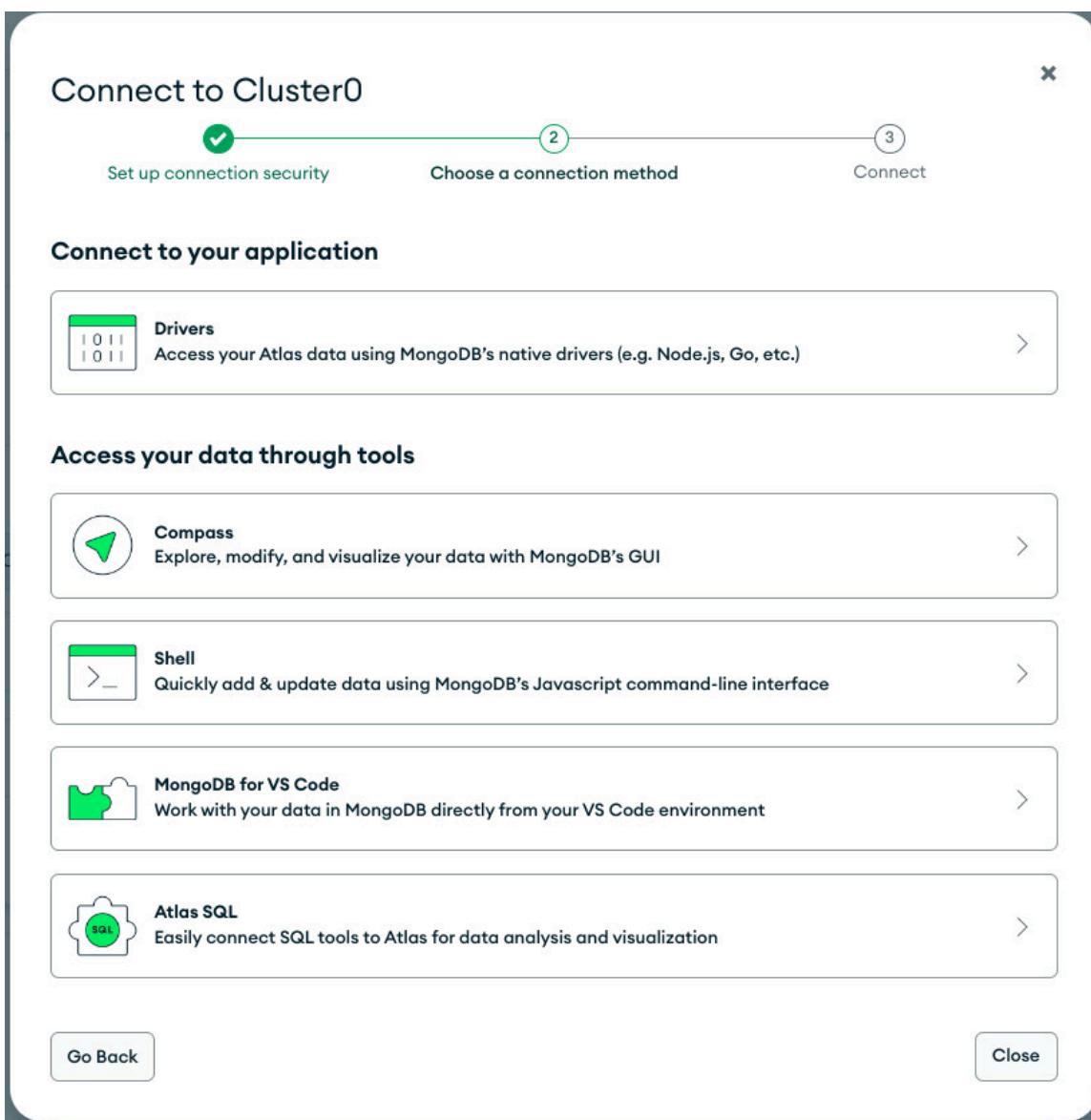


Figura Atlas 10: MongoDB Atlas. New Connect 3. (Fuente: Propia)

3.4 MongoDB Shell

Para esta conexión, solo necesitamos usar el comando `mongosh` con las credenciales configuradas anteriormente. Siguiendo los pasos dentro de **connect en la opción Mongo Shell**

que nos da Mongo Atlas (figura anterior)

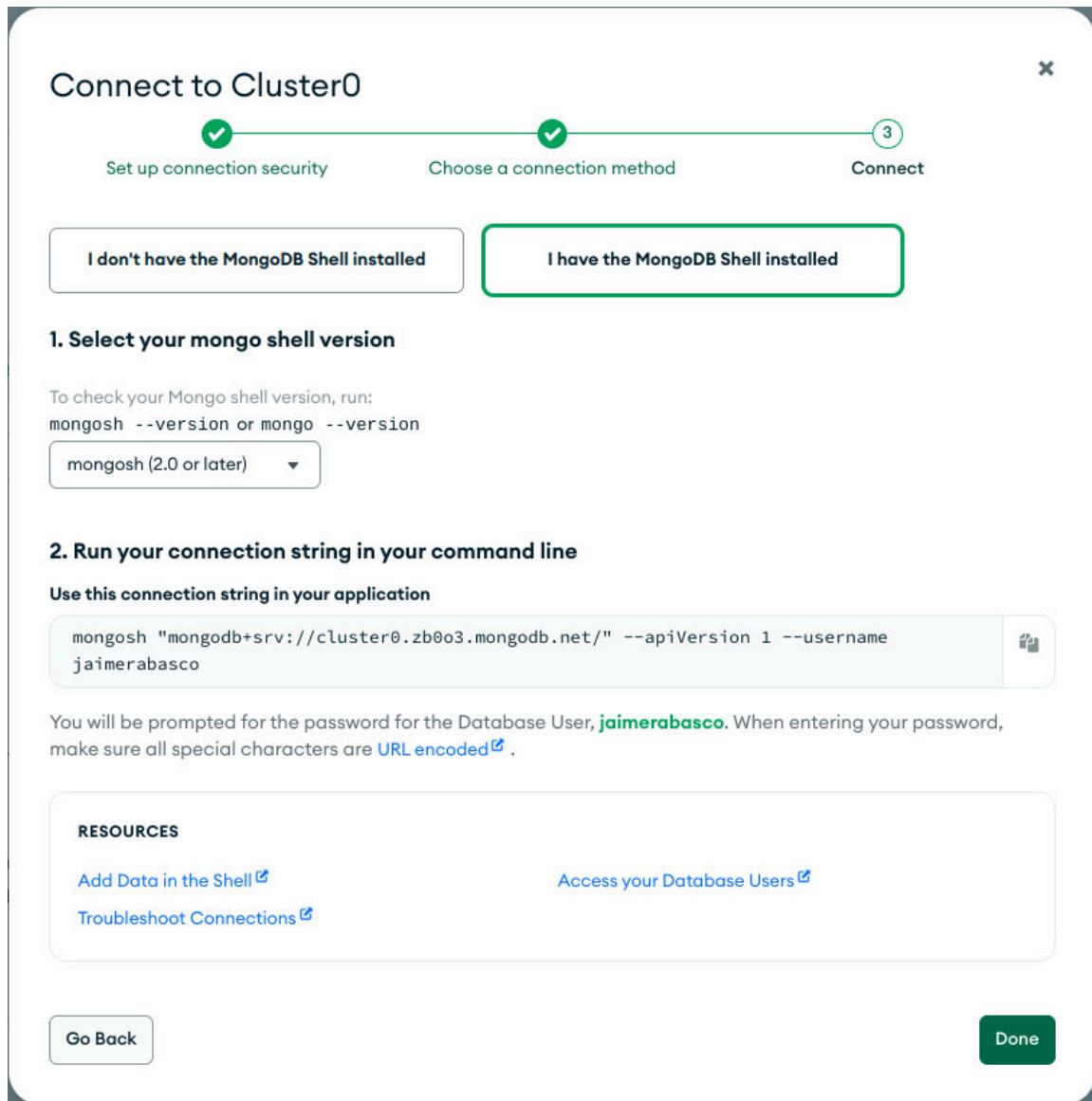
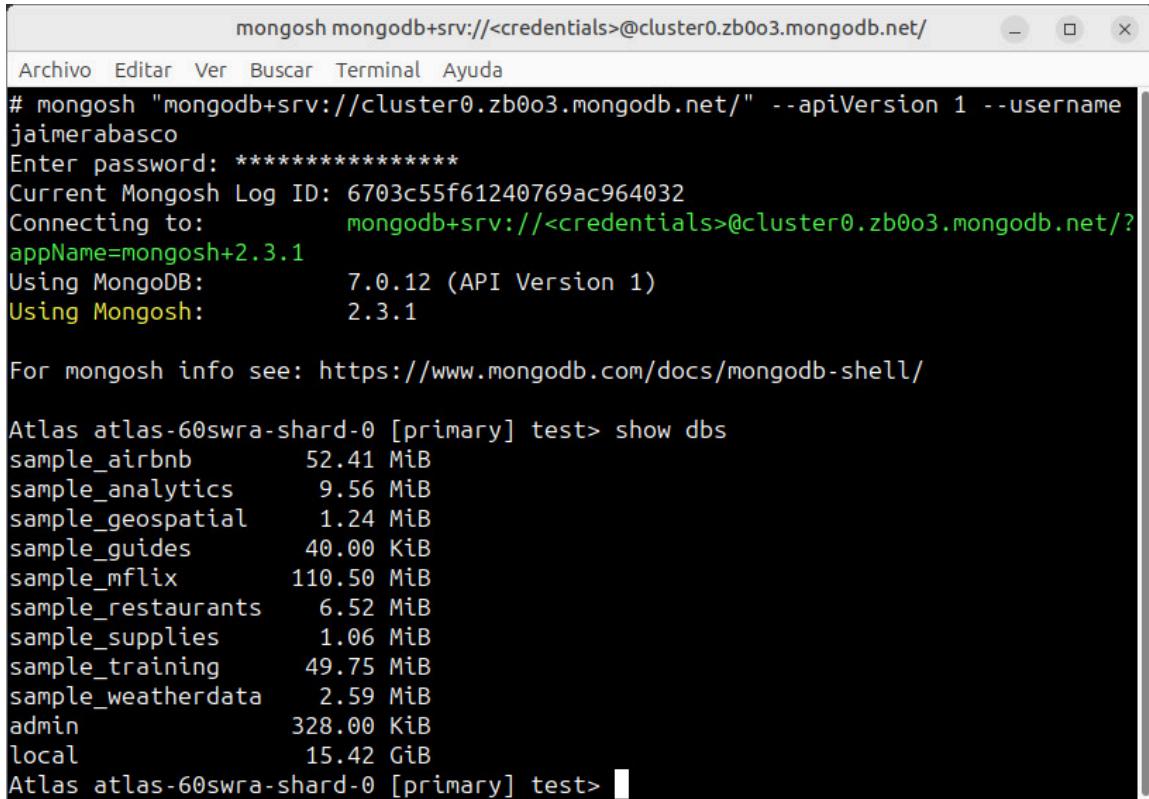


Figura Atlas 17. Connect Mongo Shell. (Fuente: Propia)

Y escribiendo después la contraseña establecida nos conectamos a nuestro al servicio cloud



```

mongosh mongodb+srv://<credentials>@cluster0.zb0o3.mongodb.net/
Archivo Editar Ver Buscar Terminal Ayuda
# mongosh "mongodb+srv://cluster0.zb0o3.mongodb.net/" --apiVersion 1 --username
jaimerabasco
Enter password: *****
Current Mongosh Log ID: 6703c55f61240769ac964032
Connecting to: mongodb+srv://<credentials>@cluster0.zb0o3.mongodb.net/?appName=mongosh+2.3.1
Using MongoDB: 7.0.12 (API Version 1)
Using Mongosh: 2.3.1

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

Atlas atlas-60swra-shard-0 [primary] test> show dbs
sample_airbnb      52.41 MiB
sample_analytics    9.56 MiB
sample_geospatial   1.24 MiB
sample_guides       40.00 KiB
sample_mflix        110.50 MiB
sample_restaurants   6.52 MiB
sample_supplies      1.06 MiB
sample_training      49.75 MiB
sample_weatherdata   2.59 MiB
admin                328.00 KiB
local                15.42 GiB
Atlas atlas-60swra-shard-0 [primary] test>

```

Figura Atlas 18. Connect Mongo Shell. (Fuente: Propia)

3.5 MongoDB para VS Code

1. Para ello usamos la extensión MongoDB for VS Code

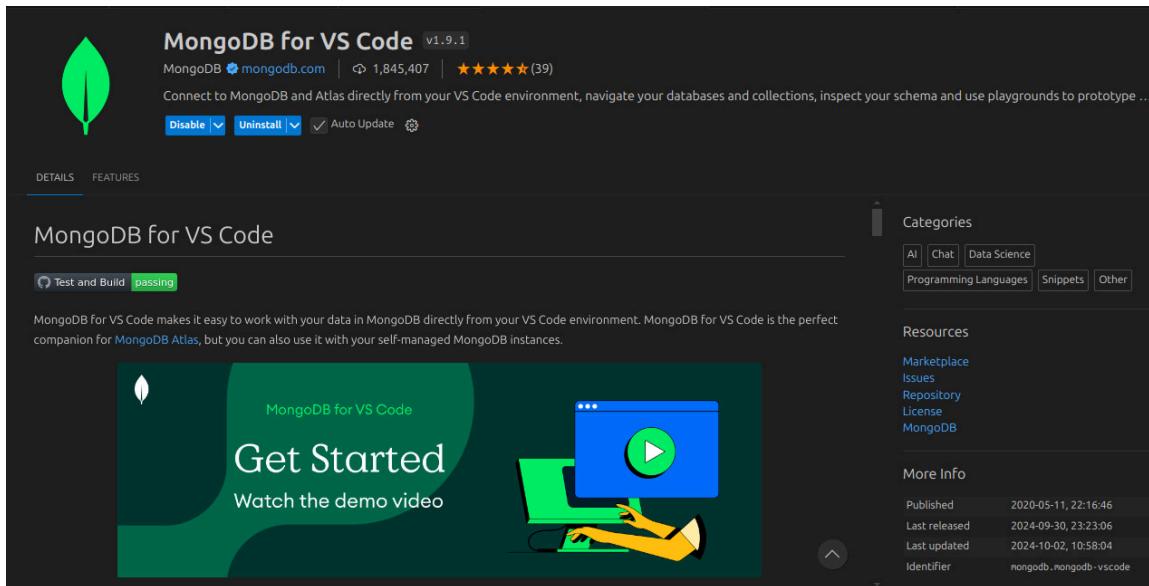


Figura MongoDB for VSCode 1. Extensión. (Fuente: Propia)

2. Damos a la opción de connect en nuestro MongoDB Atlas y seleccionamos MongoDB for VSCode. Seguimos los pasos que nos indican. Copiamos la url del punto 3 y le añadimos nuestra contraseña

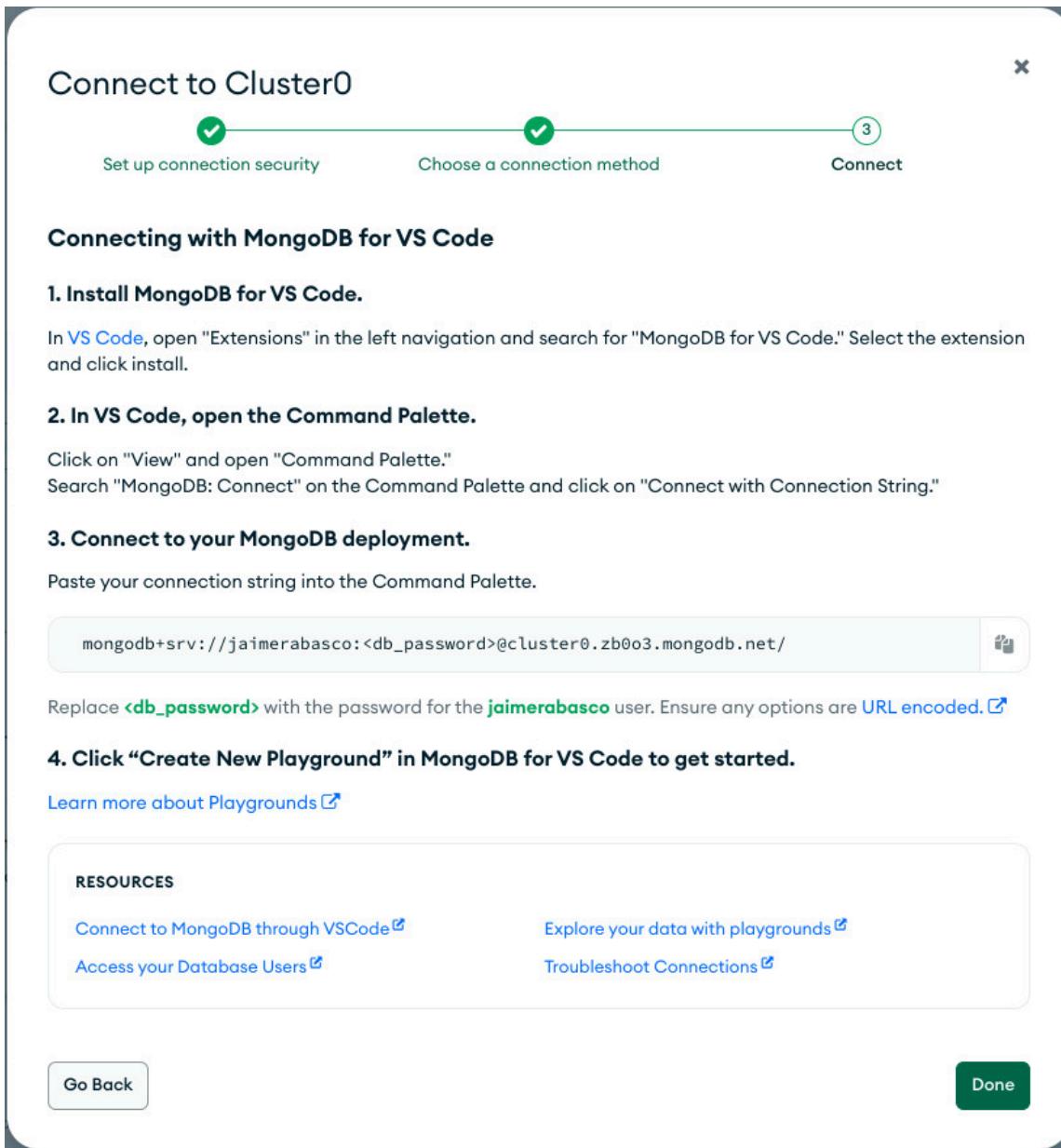


Figura MongoDB for VSCode 2. Conectando con Mondo Atlas. (Fuente: Propia)

3. Vamos a nuestra extensión y le damos a **connecting string** y añadimos la url con nuestra contraseña.

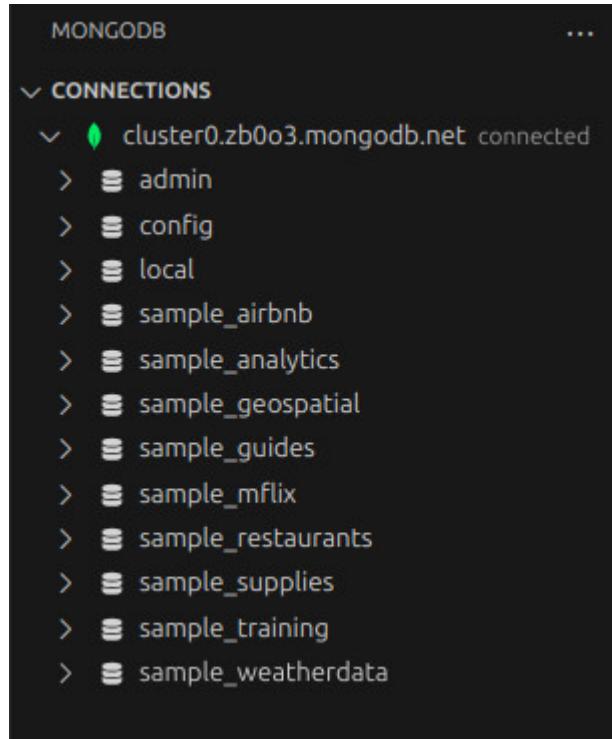


Figura MongoDB for VSCode 3. Conectado con Mondo Atlas. (Fuente: Propia)

4. Seguimos los pasos y creamos un **New Playground**. Como test, ejecutamos (botón *play* arriba a la derecha) el playground creado por defecto. Observa el resultado, la salida en consola y los nuevos datos creados en nuestra Cloud Database.

The screenshot shows the MongoDB Compass interface. In the top left, it says "DATABASES: 10 COLLECTIONS: 24". On the left sidebar, under "mongdbVSCodePlaygrou...", there's a list of collections: sales, sample_airbnb, sample_analytics, sample_geospatial, sample_guides, sample_mflix, sample_restaurants, sample_supplies, sample_training, and sample_weatherdata. The "sales" collection is selected and expanded. In the main panel, the title is "mongodbVSCodePlaygroundDB.sales". It shows storage details: STORAGE SIZE: 20KB, LOGICAL DATA SIZE: 608B, TOTAL DOCUMENTS: 8, INDEXES TOTAL SIZE: 20KB. Below this are tabs for Find, Indexes, Schema Anti-Patterns, Aggregation, and Search Indexes. A search bar says "Generate queries from natural language in Compass". A "Filter" button and a text input "Type a query: { field: 'value' }" are also present. The results section is titled "QUERY RESULTS: 1-8 OF 8" and lists three documents:

```

_id: ObjectId('6703c7d0fc8c011f49723159')
item: "abc"
price: 10
quantity: 2
date: 2014-03-01T08:00:00.000+00:00

_id: ObjectId('6703c7d0fc8c011f4972315a')
item: "jkl"
price: 20
quantity: 1
date: 2014-03-01T09:00:00.000+00:00

_id: ObjectId('6703c7d0fc8c011f4972315b')
item: "xyz"
price: 5
quantity: 10
date: 2014-03-15T09:00:00.000+00:00

```

Figura MongoDB for VSCode 4. Ejecutando playground. (Fuente: Propia)

5. Observa el código del playground. Presta especial atención a las palabras reservadas `db`, `use`, `console.log()`, `print()`, `printjson()`. Para mas información de uso consulta la documentación oficial
6. También puedes ejecutar una shell de mongoDB usando botón derecho sobre la conexión y pulsando **Launch MongoDB Shell**. Para ello debes tener en tu sistema [MongoDB Shell](#)

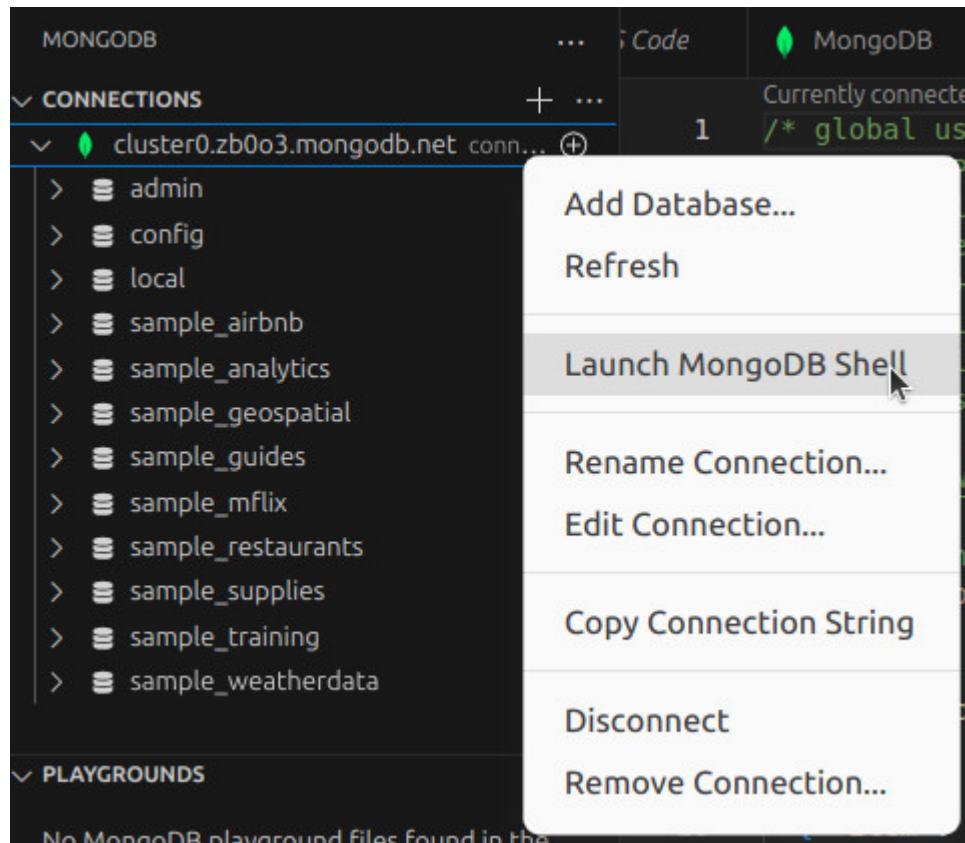


Figura MongoDB for VSCode 5. MongoDB Shell en VSCode. (Fuente: Propia)

3.6 MongoDB Compass

MongoDB [Compass](#) es una potente GUI para consultar, agregar y analizar sus datos de MongoDB en un entorno visual. MongoDB Compass viene con 3 versiones diferentes:

- **Compass:** La **full version** de MongoDB Compass, con todas las características y capacidades.
- **Readonly Edition:** Esta versión se limita estrictamente a operaciones de lectura, y se eliminan todas las capacidades de escritura y eliminación (ideal para Analista de Datos).
- **Isolated Edition:** Esta versión deshabilita todas las conexiones de red excepto la conexión a la instancia de MongoDB.

Info

Compass es de uso gratuito y está disponible en origen, y puede ejecutarse en macOS, Windows y Linux.

Puedes ver toda la [documentación](#) y los pasos de [instalación](#) en la documentación oficial

Para **comprobar su funcionamiento**, vamos a conectarnos a nuestra Base de Datos en Atlas:

1. Instalamos siguiendo los pasos de la documentación oficial.
2. Vamos a las opciones de conexión y elegimos la opción compass. Copiamos el string de conexión añadiendo nuestra contraseña.

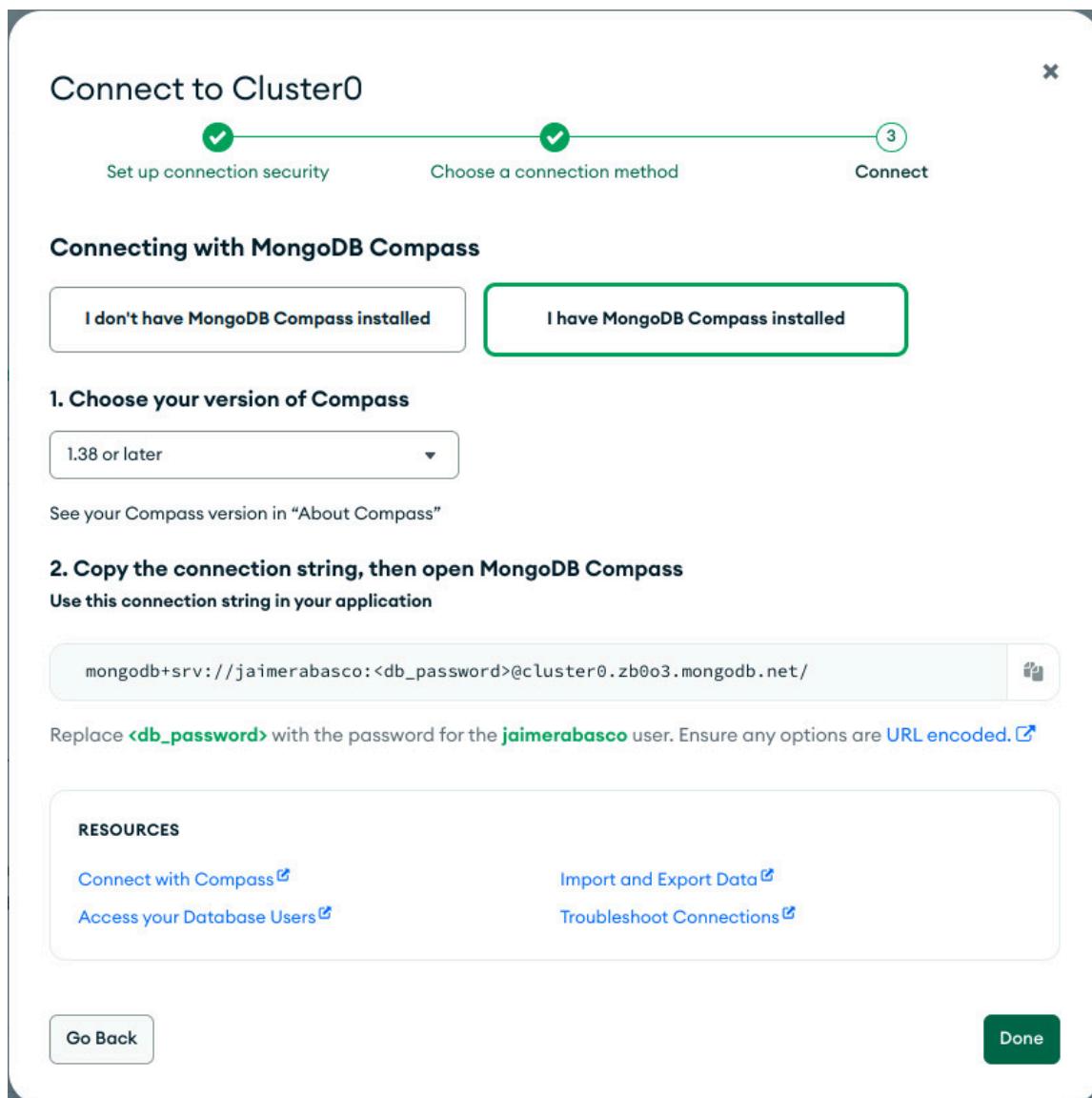


Figura MongoDB Compass 1. Conexión mediante Compass. (Fuente: Propia)

3. En Compass, abrimos la opción de conexión y copiamos nuestro string de conexión.

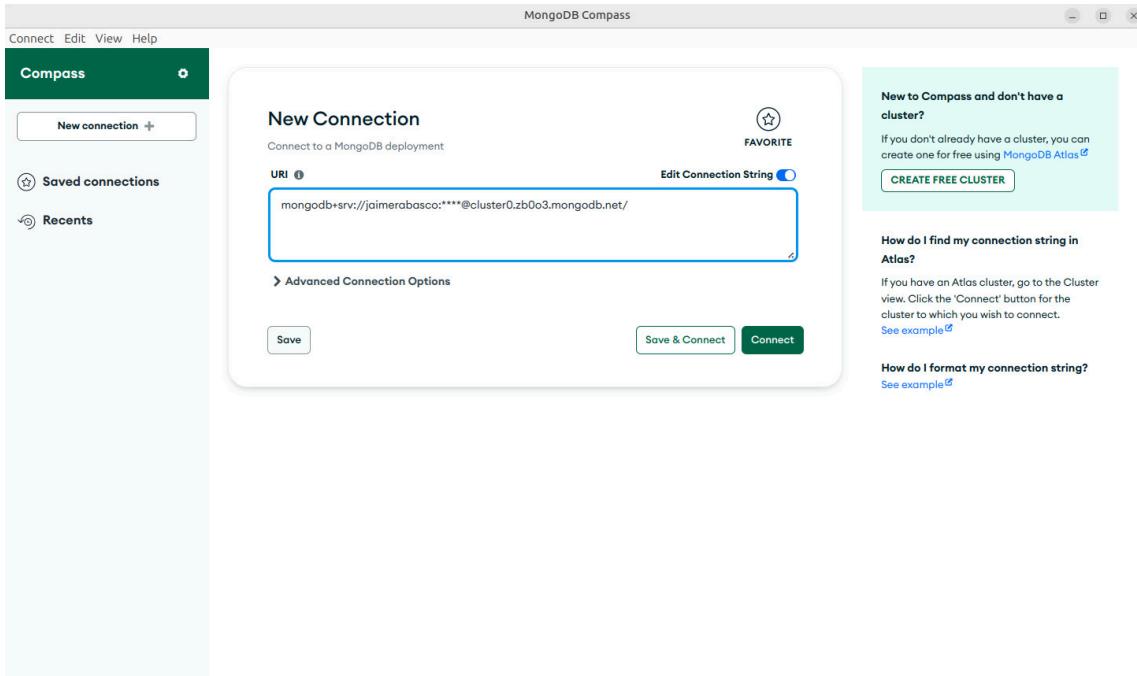


Figura MongoDB Compass 2. String de conexión. (Fuente: Propia)

4. Ya podemos usar la GUI de Compass.

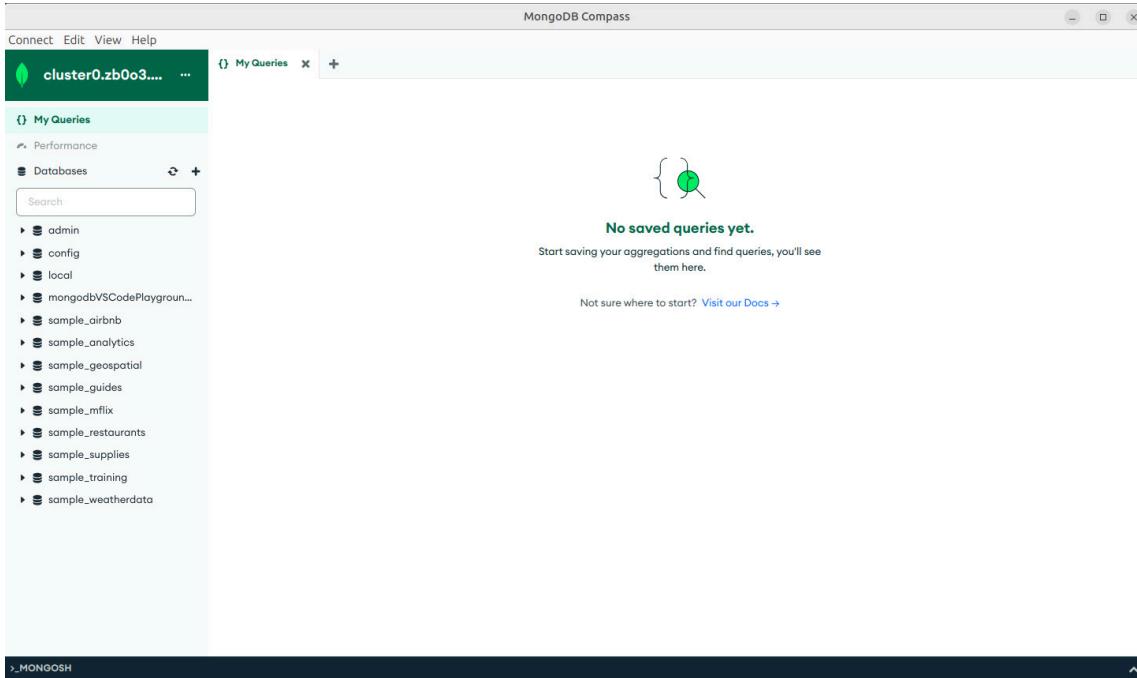


Figura MongoDB Compass 3. GUI de Compass 1. (Fuente: Propia)

5. Visualizar las Base de Datos, Collections y Documentos. Debajo también se nos abre una conexión con `mongosh` para realizar las operaciones que queramos en terminal si lo deseamos (*la cual podemos minimizar*).

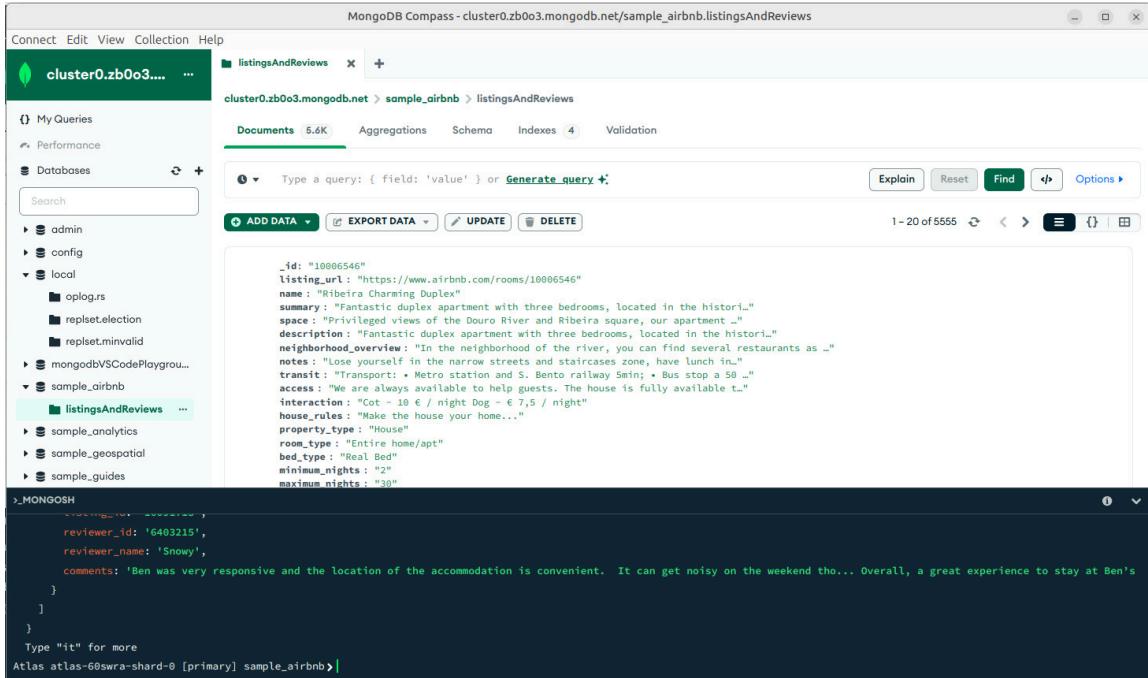


Figura MongoDB Compass 4. GUI de Compass 2. (Fuente: Propia)

3.7 Atlas SQL

Para la conexión de herramientas externas para el análisis y visualización de los datos en MongoDB Atlas

1. Creamos la conexión. Se crea un **SQL Schema** para su correcto acceso desde herramientas externas

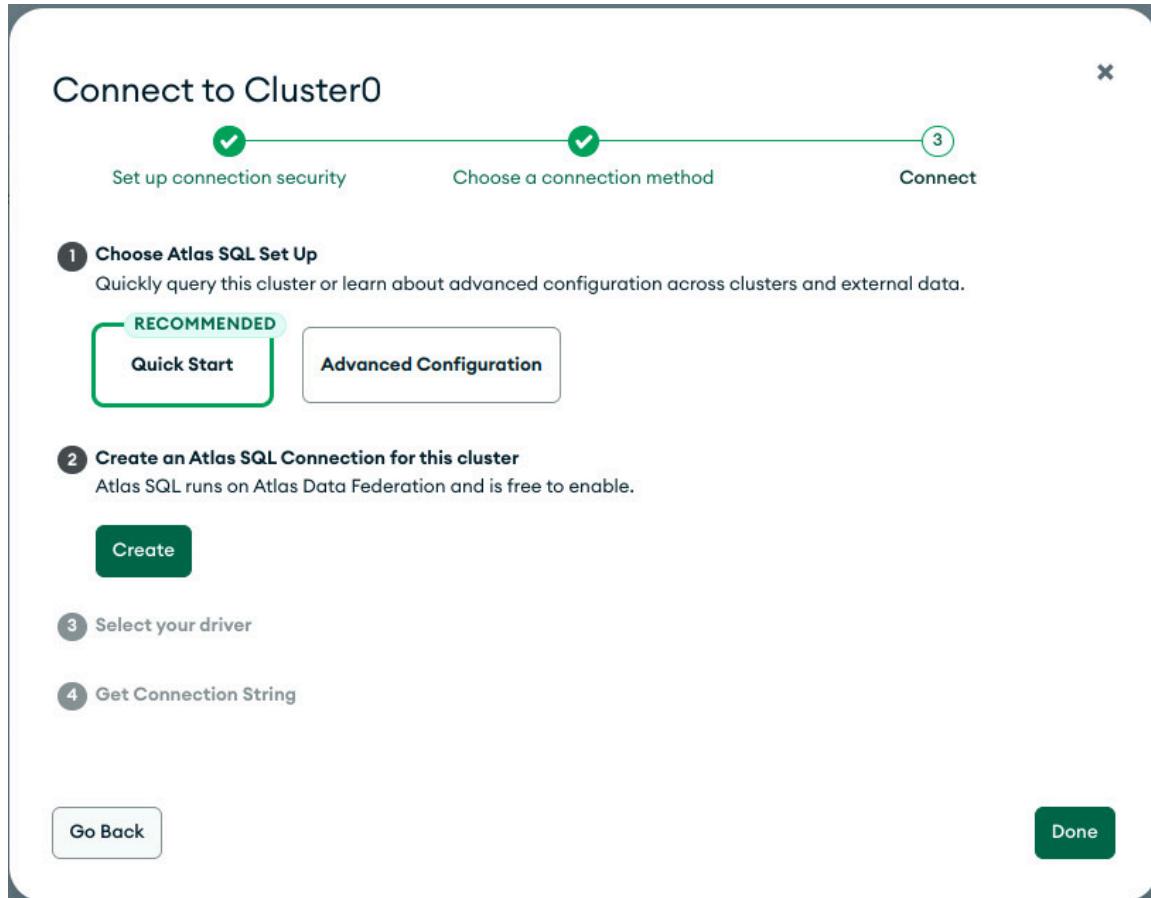


Figura MongoDB Atlas SQL 1. (Fuente: Propia)

2. Elegimos la herramienta externa (PowerBI, Tableau, JDBC Driver, ODBC Driver).

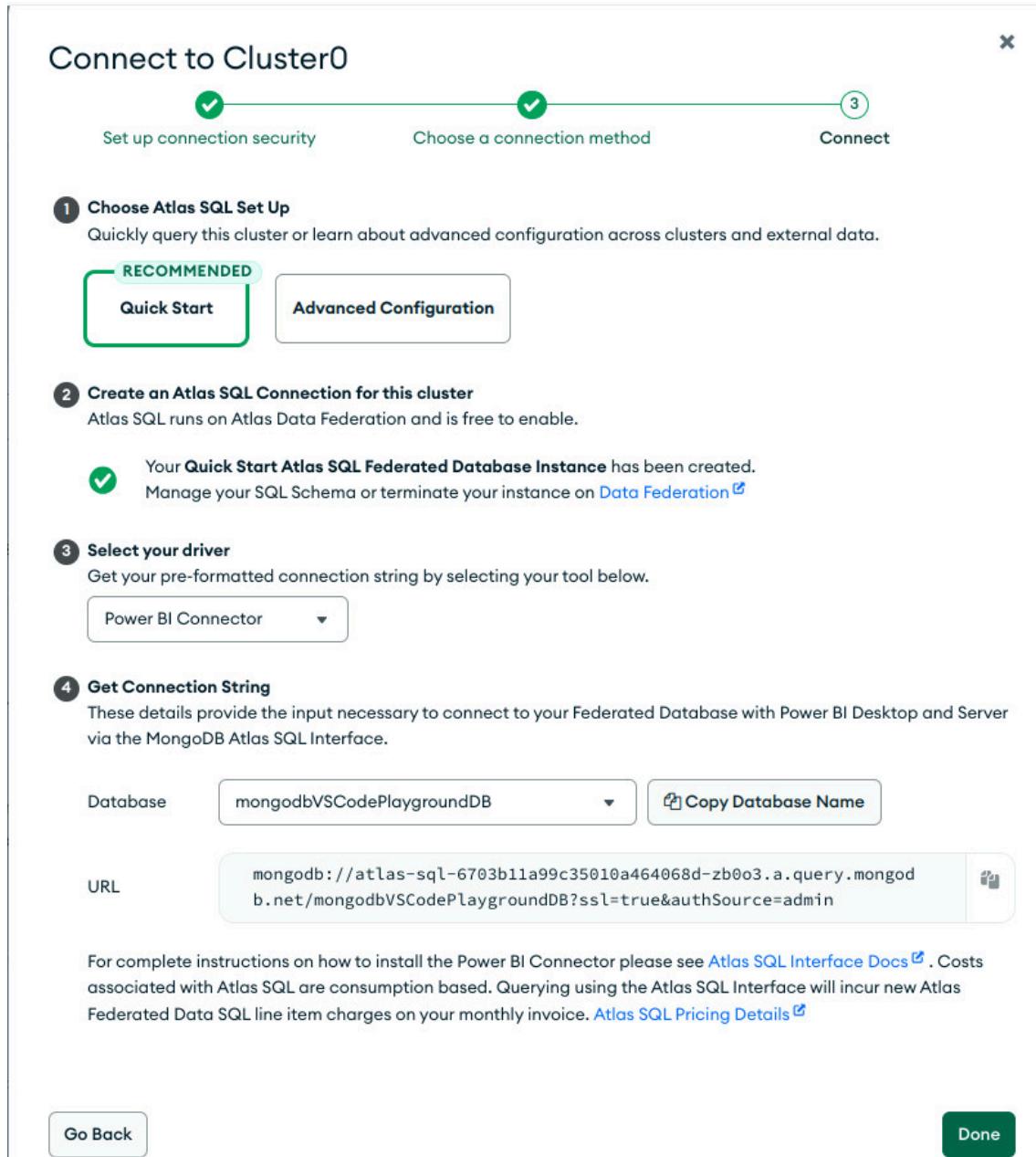


Figura MongoDB Atlas SQL 1. (Fuente: Propia)

3.8 MongoDB Tools

The [MongoDB Database Tools](#) son una colección de utilidades de línea de comandos para trabajar con una implementación de MongoDB. Las herramientas de base de datos incluyen los siguientes binarios:

1. Binary Import / Export:

- `mongodump` : Crea una exportación binaria del contenido de un demonio *mongod*.

- `mongorestore` : Restaura datos de un volcado de base de datos mongodump en un *mongod* o *mongos* (*Para un shared cluster, las instancias de _mongos proporcionan la interfaz entre las aplicaciones cliente y el clúster fragmentado.*)
- `bsondump` : Convierte archivos *BSON dump* into *JSON*.

2. Data Import / Export:

- `mongoimport` : Importa el contenido desde un archivo externo E_xtended JSON, CSV, or TSV._
- `mongoexport` : Produce una exportación JSON o CSV de datos almacenados en un instancia de *mongod*.

3. Diagnostic Tools:

- `mongostat` : Proporciona una descripción general rápida del estado de una instancia de *mongod* o *mongos* actualmente en ejecución.
- `mongotop` : Proporciona una descripción general del tiempo que una instancia de *mongod* dedica a leer y escribir datos.

4. GridFS Tools:

- `mongofiles` : admite la manipulación de archivos almacenados en su instancia de MongoDB en objetos *GridFS*.

Puedes optar por descargar estas herramientas en local o utilizar algunas de las opciones mostradas anteriormente. La instancia de MongoDB creada en Docker también tiene por defecto todas estas herramientas.

4. Operaciones CRUD

4.1 Create Operations

Las operaciones de creación o inserción agregan nuevos documentos a una colección. Si la colección no existe actualmente, las operaciones de inserción crearán la colección.

MongoDB proporciona los siguientes métodos para insertar documentos en una colección:

- `db.collection.insertOne()`
- `db.collection.insertMany()`

En MongoDB, las operaciones de inserción tienen como objetivo una única colección. Todas las operaciones de escritura en MongoDB son **atómicas** al nivel de un solo documento.

```
db.users.insertOne( ← collection
{
    name: "sue", ← field: value
    age: 26, ← field: value
    status: "pending" ← field: value
}
)
```

The diagram illustrates the structure of a MongoDB document. It shows a single brace on the right side of the code block, labeled 'document'. Three green arrows point from the 'name', 'age', and 'status' fields to their respective values, indicating they are 'field: value' pairs.

Figura MongoDB Create Operations. (Fuente: MongoDB)

Insertar un documento

`db.collection.insertOne()` inserta un único documento dentro de una collection.



Note

El siguiente código inserta un nuevo documento en la colección de inventario. Si el documento no especifica un campo `_id`, MongoDB agrega el campo `_id` con un valor ObjectId al nuevo documento.

```
1 db.inventory.insertOne(
2     { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5,
uom: "cm" } }
3 )
```

`insertOne()` devuelve un documento que incluye el valor del campo `_id` del documento recién insertado.

Insertar varios documentos

`db.collection.insertMany()` puede insertar múltiples documentos a una collection. Pasa un array of documentos al método.

```
1 db.inventory.insertMany([
2     { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w:
21, uom: "cm" } },
3     { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom:
"cm" } },
4     { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w:
22.85, uom: "cm" } }
5 ])
```

4.2 Read Operations

En primer lugar, antes de aprender consultas en MongoDB, vamos a ver cuales son los operadores que facilita MongoDB para poder realizar todo tipo de consultas.

Una vez estudiados los distintos operadores, veremos como hacer consultas.

4.2.1 Query Operators

Mostraremos un resumen de todos los operadores (salvo *aggregation operators* que veremos más adelante cuando expliquemos agregaciones). Para más detalle, explicación y ejemplo de cada operador, la puedes encontrar en la [documentación oficial](#)

Query Selectors

1. Comparison

Name	Description
<code>\$eq</code>	Matches values that are equal to a specified value.
<code>\$gt</code>	Matches values that are greater than a specified value.
<code>\$gte</code>	Matches values that are greater than or equal to a specified value.
<code>\$in</code>	Matches any of the values specified in an array.
<code>\$lt</code>	Matches values that are less than a specified value.
<code>\$lte</code>	Matches values that are less than or equal to a specified value.
<code>\$ne</code>	Matches all values that are not equal to a specified value.
<code>\$nin</code>	Matches none of the values specified in an array.

Figura MongoDB Operators 1. Comparison Operators (Fuente: MongoDB)

2. Logical

Name	Description
<code>\$and</code>	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
<code>\$not</code>	Inverts the effect of a query expression and returns documents that do <i>not</i> match the query expression.
<code>\$nor</code>	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
<code>\$or</code>	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

Figura MongoDB Operators 2. Logical Operators (Fuente: MongoDB)

3. Element

Name	Description
<code>\$exists</code>	Matches documents that have the specified field.
<code>\$type</code>	Selects documents if a field is of the specified type.

Figura MongoDB Operators 3. Element Operators (Fuente: MongoDB)

4. Evaluation

Name	Description
<code>\$expr</code>	Allows use of aggregation expressions within the query language.
<code>\$jsonSchema</code>	Validate documents against the given JSON Schema.
<code>\$mod</code>	Performs a modulo operation on the value of a field and selects documents with a specified result.
<code>\$regex</code>	Selects documents where values match a specified regular expression.
<code>\$text</code>	Performs text search.
<code>\$where</code>	Matches documents that satisfy a JavaScript expression.

Figura MongoDB Operators 4. Evaluation Operators (Fuente: MongoDB)

5. Geospatial

Name	Description
<code>\$geoIntersects</code>	Selects geometries that intersect with a GeoJSON geometry. The <code>2dsphere</code> index supports <code>\$geoIntersects</code> .
<code>\$geoWithin</code>	Selects geometries within a bounding GeoJSON geometry . The <code>2dsphere</code> and <code>2d</code> indexes support <code>\$geoWithin</code> .
<code>\$near</code>	Returns geospatial objects in proximity to a point. Requires a geospatial index. The <code>2dsphere</code> and <code>2d</code> indexes support <code>\$near</code> .
<code>\$nearSphere</code>	Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The <code>2dsphere</code> and <code>2d</code> indexes support <code>\$nearSphere</code> .

Figura MongoDB Operators 5. Geospatial Operators (Fuente: MongoDB)

6. Array

Name	Description
<code>\$all</code>	Matches arrays that contain all elements specified in the query.
<code>\$elemMatch</code>	Selects documents if element in the array field matches all the specified <code>\$elemMatch</code> conditions.
<code>\$size</code>	Selects documents if the array field is a specified size.

Figura MongoDB Operators 6. Array Operators (Fuente: MongoDB)

7. Bitwise

Name	Description
<code>\$bitsAllClear</code>	Matches numeric or binary values in which a set of bit positions <i>all</i> have a value of 0.
<code>\$bitsAllSet</code>	Matches numeric or binary values in which a set of bit positions <i>all</i> have a value of 1.
<code>\$bitsAnyClear</code>	Matches numeric or binary values in which <i>any</i> bit from a set of bit positions has a value of 0.
<code>\$bitsAnySet</code>	Matches numeric or binary values in which <i>any</i> bit from a set of bit positions has a value of 1.

Figura MongoDB Operators 7. Bitwise Operators (Fuente: MongoDB)

Projection Operators

Name	Description
<code>\$</code>	Projects the first element in an array that matches the query condition.
<code>\$elemMatch</code>	Projects the first element in an array that matches the specified <code>\$elemMatch</code> condition.
<code>\$meta</code>	Projects the document's score assigned during <code>\$text</code> operation.
<code>\$slice</code>	Limits the number of elements projected from an array. Supports skip and limit slices.

Figura MongoDB Operators 8. Projection Operators (Fuente: MongoDB)

Miscellaneous Operators

Name	Description
<code>\$comment</code>	Adds a comment to a query predicate.
<code>\$rand</code>	Generates a random float between 0 and 1.

Figura MongoDB Operators 9. Miscellaneous Operators (Fuente: MongoDB)

4.2.2 Consultas MondoDB

Para usar como ejemplo añadimos los siguientes documentos. Primero nos aseguramos de tener la collection vacía.

```

1 db.inventory.deleteMany({})

1 db.inventory.insertMany([
2   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status:
  "A" },
3   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" },
  status: "A" },
4   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status:
  "D" },
5   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" },
  status: "D" },
6   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" },
  status: "A" }
7 ]);
```

Seleccionar todos los documentos de un Collection

Para seleccionar todos los documentos de la colección, pasamos un documento vacío como parámetro de filtro de consulta al método de búsqueda. Esta operación utiliza un predicado de filtro de {}:

Mongosh

```
1 db.inventory.find( {} )
```

Equivalente en SQL

```
1 SELECT * FROM inventory
```

Para poder realizar cualquier tipo de consulta/búsqueda usaremos dentro de las {} los filtros/operadores (vistos en el punto anterior) para realizar todo tipo de consultas.

Especificando una condición de igualdad

Para especificar condiciones de igualdad, utiliza la expresión `<field>:<value>` como filtro de la consulta del documento:

```
1 | { <field1>: <value1>, ... }
```

El siguiente ejemplo selecciona de la colección de inventario todos los documentos cuyo estado es "D":

Mongosh

```
1 | db.inventory.find( { status: "D" } )
```

Equivalente en SQL

```
1 | SELECT * FROM inventory WHERE status = "D"
```

Especificando condiciones mediante operadores de consulta

Un filtro de consulta de un documento puede utilizar los operadores de consulta para especificar condiciones de la siguiente forma:

```
1 | { <field1>: { <operator1>: <value1>, ... } }
```

El siguiente ejemplo recupera todos los documentos de la colección donde el estado es "A" o "D":

Mongosh

```
1 | db.inventory.find( { status: { $in: [ "A", "D" ] } } )
```

Podría usarse también el operador `$or`

Equivalente en SQL

```
1 | SELECT * FROM inventory WHERE status IN ("A", "D")
```

Usando condición AND

Una consulta compuesta puede especificar condiciones para más de un campo en los documentos de la colección. Implícitamente, una conjunción lógica `AND` conecta las cláusulas de una consulta compuesta de modo que la consulta selecciona los documentos de la colección que coinciden con todas las condiciones.

El siguiente ejemplo recupera todos los documentos de la colección donde el estado es igual a "A" y la cantidad es menor que `$lt 30`:

Mongosh

```
1 | db.inventory.find( { status: "A", qty: { $lt: 30 } } )
```

Equivalente en SQL

```
1 | SELECT * FROM inventory WHERE status = "A" AND qty < 30
```

Usando condición OR

Con el operador `$or`, puedes especificar una consulta compuesta que une cada cláusula con una conjunción OR lógica para que la consulta seleccione los documentos de la colección que coincidan con al menos una condición.

El siguiente ejemplo recupera todos los documentos de la colección donde el estado es igual a "A" o la cantidad es menor que `$lt 30`:

Mongosh

```
1 | db.inventory.find( { $or: [ { status: "A" }, { qty: { $lt: 30 } } ] } )
```

Equivalente en SQL

```
1 | SELECT * FROM inventory WHERE status = "A" OR qty < 30
```

Especificando condiciones AND y OR

En el siguiente ejemplo, el documento de consulta compuesta selecciona todos los documentos de la colección donde el estado es igual a "A" y la cantidad es menor que `$lt 30` o el elemento comienza con el carácter p:

Mongosh

```
1 | db.inventory.find( {
2 |   status: "A",
3 |   $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ]
4 | })
```

Podría usarse otro operador de filtro:

```
1 | db.inventory.find( {
2 |   status: "A",
3 |   $or: [ { qty: { $lt: 30 } }, { item: { $regex: '^p' } } ]
4 | })
```

Equivalente en SQL

```
1 | SELECT * FROM inventory WHERE status = "A" AND ( qty < 30 OR item LIKE
"p%")
```

Con esta introducción podemos tener una buena introducción a las *query operations* de MongoDB. Pero existen otros muchos tipos:

- [Query on Embedded/Nested Documents](#)
- [Query an Array](#)
- [Query an Array of Embedded Documents](#)
- [Project Fields to Return from Query](#)
- [Query for Null or Missing Fields](#)

Puedes consultarlas en la [documentación oficial](#)

4.3 Update Operations

Como en el caso de las consultas, vamos a ver primero los operadores de actualización y seguidamente como las realizamos en MongoDB

4.3.1 Update Operators

Los siguientes modificadores están disponibles para su uso en operaciones de actualización.

Sintaxis

```
1 | {
2 |     <operator1>: { <field1>: <value1>, ... },
3 |     <operator2>: { <field2>: <value2>, ... },
4 |     ...
5 | }
```

Para más detalle, explicación y ejemplo de cada operador, la puedes encontrar en la [documentación oficial. Update Operator](#)

1. Field update operators

Name	Description
<code>\$currentDate</code>	Sets the value of a field to current date, either as a Date or a Timestamp.
<code>\$inc</code>	Increments the value of the field by the specified amount.
<code>\$min</code>	Only updates the field if the specified value is less than the existing field value.
<code>\$max</code>	Only updates the field if the specified value is greater than the existing field value.
<code>\$mul</code>	Multiplies the value of the field by the specified amount.
<code>\$rename</code>	Renames a field.
<code>\$set</code>	Sets the value of a field in a document.
<code>\$setOnInsert</code>	Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.
<code>\$unset</code>	Removes the specified field from a document.

Figura MongoDB Update Operators 1. Field (Fuente: MongoDB)

2. Array update operators

Name	Description
<code>\$</code>	Acts as a placeholder to update the first element that matches the query condition.
<code>\$[]</code>	Acts as a placeholder to update all elements in an array for the documents that match the query condition.
<code>\$[<identifier>]</code>	Acts as a placeholder to update all elements that match the <code>arrayFilters</code> condition for the documents that match the query condition.
<code>\$addToSet</code>	Adds elements to an array only if they do not already exist in the set.
<code>\$pop</code>	Removes the first or last item of an array.
<code>\$pull</code>	Removes all array elements that match a specified query.
<code>\$push</code>	Adds an item to an array.
<code>\$pullAll</code>	Removes all matching values from an array.

Figura MongoDB Update Operators 2. Array (Fuente: MongoDB)

3. Modifiers update operators

Name	Description
<code>\$each</code>	Modifies the <code>\$push</code> and <code>\$addToSet</code> operators to append multiple items for array updates.
<code>\$position</code>	Modifies the <code>\$push</code> operator to specify the position in the array to add elements.
<code>\$slice</code>	Modifies the <code>\$push</code> operator to limit the size of updated arrays.
<code>\$sort</code>	Modifies the <code>\$push</code> operator to reorder documents stored in an array.

Figura MongoDB Update Operators 3. Modifiers (Fuente: MongoDB)

4. Bitwise update operators

Name	Description
<code>\$bit</code>	Performs bitwise AND, OR, and XOR updates of integer values.

Figura MongoDB Update Operators 4. Bitwise (Fuente: MongoDB)

4.3.2 Actualizaciones

Para usar como ejemplo añadimos los siguientes documentos. Primero nos aseguramos de tener la collection vacía.

```

1 db.inventory.deleteMany({})

1 db.inventory.insertMany( [
2   { item: "canvas", qty: 100, size: { h: 28, w: 35.5, uom: "cm" },
3     status: "A" },
4   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status:
5     "A" },
6   { item: "mat", qty: 85, size: { h: 27.9, w: 35.5, uom: "cm" }, status:
7     "A" },
8   { item: "mousepad", qty: 25, size: { h: 19, w: 22.85, uom: "cm" },
9     status: "P" },
10  { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" },
11    status: "P" },
12  { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "cm" }, status:
13    "D" },
14  { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" },
15    status: "D" },
16  { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" },
17    status: "A" },
18  { item: "sketchbook", qty: 80, size: { h: 14, w: 21, uom: "cm" },
19    status: "A" },
20  { item: "sketch pad", qty: 95, size: { h: 22.85, w: 30.5, uom: "cm" },
21    status: "A" }
22 ] );

```

MongoDB proporciona los siguientes métodos para actualizar documentos en una colección:

- db.collection.updateOne(,)
- db.collection.updateMany(,)
- db.collection.replaceOne(,)

Para la actualización de documentos de una collection usamos la siguiente sintaxis

```

1  {
2      <update operator>: { <field1>: <value1>, ... },
3      <update operator>: { <field2>: <value2>, ... },
4      ...
5  }
```

Update a Single Document

El siguiente ejemplo utiliza el método `db.collection.updateOne()` en la colección de inventario para actualizar **el primer documento** donde elemento es igual a "paper":

```

1  db.inventory.updateOne(
2      { item: "paper" },
3      {
4          $set: { "size.uom": "cm", status: "P" },
5          $currentDate: { lastModified: true }
6      }
7  )
```

La operación de actualización:

- utiliza el operador `$set` para actualizar el valor del campo `size.uom` a "cm" y el valor del campo de estado a "P",
- utiliza el operador `$currentDate` para actualizar el valor del campo `lastModified` a la fecha actual. Si el campo `lastModified` no existe, `$currentDate` creará el campo.

Update Multiple Documents

El siguiente ejemplo utiliza el método `db.collection.updateMany()` en la colección de `inventory` para actualizar **todos los documentos** donde la cantidad es inferior a 50:

```

1  db.inventory.updateMany(
2      { "qty": { $lt: 50 } },
3      {
4          $set: { "size.uom": "in", status: "P" },
5          $currentDate: { lastModified: true }
6      }
7  )
```

La operación de actualización:

- utiliza el operador `$set` para actualizar el valor del campo `size uom` a "in" y el valor del campo de estado a "P",
- utiliza el operador `$currentDate` para actualizar el valor del campo `lastModified` a la fecha actual. Si el campo `lastModified` no existe, `$currentDate` creará el campo.

Replace a Document

Para reemplazar todo el contenido de un documento excepto el campo `_id`, pase un

documento completamente nuevo como segundo argumento a

```
db.collection.replaceOne()
```

Al reemplazar un documento, el documento de reemplazo debe consistir únicamente en pares de campo/valor; es decir, no incluir actualizar operadores expresiones.

El documento de reemplazo puede tener campos diferentes a los del documento original. En el documento de reemplazo, puede omitir el campo `_id` ya que el campo `_id` es **immutable**; sin embargo, si incluye el campo `_id`, debe tener el mismo valor que el valor actual.

El siguiente ejemplo reemplaza el primer documento de la colección de inventario donde elemento: "papel":

```
1 db.inventory.replaceOne(
2   { item: "paper" },
3   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 } ] }
4 )
```

Otros ejemplos

```
1 db.ships.updateOne({name : {$eq: 'USS-Enterprise-D'}}, {$set : {name: 'USS Something'}})
2 db.ships.updateOne({name : {$eq: 'USS Something'}}, {$set : {name: 'USS Prometheus', class: 'Prometheus'}})
3 db.ships.updateOne({name : {$eq: 'USS Prometheus'}}, {$unset : {operator: 1}})
```

4.3.3 Métodos adicionales

MongoDB tiene otros métodos menos usados para la actualización de documentos en las colecciones:

- `db.collection.findOneAndReplace()`
- `db.collection.findOneAndUpdate()`
- `db.collection.findAndModify()`
- `db.collection.bulkWrite()`

Puedes obtener más información en la [documentación oficial. Update methods](#)

4.4 Delete Operations

Para usar como ejemplo añadimos los siguientes documentos. Primero nos aseguramos de tener la collection vacía.

```
1 db.inventory.deleteMany({})  
  
1 db.inventory.insertMany( [  
2   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status:  
"A" },  
3   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" },  
status: "P" },  
4   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status:  
"D" },  
5   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" },  
status: "D" },  
6   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" },  
status: "A" },  
7 ] );
```

MongoDB proporciona los siguientes métodos para borrar documentos en una colección:

- db.collection.deleteMany()
- db.collection.deleteOne()

Delete All Documents

Para eliminar todos los documentos de una colección, pasamos un documento de filtro vacío {} al método `db.collection.deleteMany()`.

```
1 db.inventario.deleteMany({})
```

El método devuelve un documento con el estado de la operación.

Delete All Documents that Match a Condition

Podemos especificar criterios o filtros que identifiquen los documentos que se eliminarán. Los filtros utilizan la misma sintaxis que las [operaciones de lectura](#).

Para especificar condiciones de igualdad, utilizamos expresiones <campo>:<valor> en el documento de filtro de consulta:

```
1 { <campo1>: <valor1>, ... }
```

Una consulta de documento con filtro podemos utilizar los operadores de consulta para especificar condiciones de la siguiente forma:

```
1 | { <campo1>: { <operador1>: <valor1> }, ... }
```

Entonces, para eliminar todos los documentos que coincidan con un criterio de eliminación, pasamos un filtro por parámetro al método `deleteMany()`.

Lo vemos en el siguiente ejemplo, donde eliminamos todos los documentos de la colección de inventario donde el campo de estado es "A":

```
1 | db.inventory.deleteMany({ status: "A" })
```

El método devuelve un documento con el estado de la operación.

Delete Only One Document that Matches a Condition

Para eliminar como máximo un único documento que coincida con un filtro específico (aunque varios documentos puedan coincidir con el filtro especificado), utilizamos el método `db.collection.deleteOne()`.

En el siguiente ejemplo elimina el primer documento cuyo estado es "D":

```
1 | db.inventory.deleteOne( {status: "D" } )
```

Otros Ejemplos

```
1 | db.ships.deleteOne({name : 'USS Prometheus'})  
2 | db.ships.deleteOne({name:$regex:'^A*' })
```

5. Aggregations

Las operaciones de agregación procesan múltiples documentos y devuelven resultados calculados. Puede utilizar operaciones de agregación para:

- Agrupar valores desde varios documentos a la vez.
- Realice operaciones sobre los **datos agrupados** para devolver un único resultado.
- Analizar los cambios de datos a lo largo del tiempo.

Para realizar operaciones de agregación, puede utilizar:

- **Aggregation pipelines**, que son el método preferido para realizar agregaciones.
- **Single purpose aggregation methods**, que son simples pero carecen de las capacidades de una Aggregation pipelines.

Un proceso de agregación consta de **una o más etapas** que procesan documentos:

- Cada etapa realiza una operación en los documentos de entrada. Por ejemplo, una etapa puede filtrar documentos, agrupar documentos y calcular valores.
- Los documentos que salen de una etapa pasan a la siguiente etapa.
- Una Aggregation Pipelines puede devolver resultados por grupos de documentos. Por ejemplo, devuelva los valores total, promedio, máximo y mínimo.

Para poder realizar estas agrupaciones de datos por etapas MongoDB proporciona una larga lista de Operadores

5.1 Operator Aggregation Pipelines Stages

Hay una larga lista de operadores que te permiten establecer cada una de las etapas de los Aggregation Pipelines. En la documentación oficial tienes la [lista completa](#).

A continuación extraemos los más utilizados a modo de resumen:

Estado	Descripción
\$project	Cambia la forma de cada documento en la secuencia, por ejemplo agregando nuevos campos o eliminando campos existentes. Por cada documento de entrada, genera un documento.
\$match	Filtrá el <i>stream</i> de documentos para permitir que solo los documentos coincidentes pasen sin modificaciones a la siguiente etapa del proceso. <code>\$match</code> utiliza consultas estándar de MongoDB. Para cada documento de entrada, genera un documento (una coincidencia) o cero documentos (ninguna coincidencia).
\$group	Agrupa los documentos de entrada por una expresión de identificador especificada y aplica las expresiones del acumulador, si se especifican, a cada grupo. Consumo todos los documentos de entrada y genera un documento por cada grupo distinto. Los documentos de salida solo contienen el campo identificador y, si se especifica, campos acumulados.
\$sort	Reordena el flujo de documentos según una clave de clasificación especificada. Sólo cambia el orden; los documentos permanecen sin modificaciones. Por cada documento de entrada, genera un documento.

Estado	Descripción
\$skip	Omite los primeros n documentos donde n es el número de omisión especificado y pasa los documentos restantes sin modificar a la canalización. Para cada documento de entrada, genera cero documentos (para los primeros n documentos) o un documento (si está después de los primeros n documentos).
\$limit	Pasa los primeros n documentos sin modificar al <i>pipeline</i> , donde n es el límite especificado. Para cada documento de entrada, genera un documento (para los primeros n documentos) o cero documentos (después de los primeros n documentos).
\$unwind	Deconstruye un campo de matriz a partir de los documentos de entrada para generar un documento para cada elemento. Cada documento de salida reemplaza la matriz con un valor de elemento. Para cada documento de entrada, genera n documentos donde n es el número de elementos de la matriz y puede ser cero para una matriz vacía.

Tabla 1: Operadores de aggregations más usados en MongoDB

5.2 Aggregation Pipelines

Una vez vista la información sobre Aggregation vamos a aprender a usarlo usando pequeños ejemplos. Para ello vamos a tomar como base la siguiente colección `orders` de pedidos de pizzas. Recuerda asegurarte de tener la collection vacía.

```
1 db.orders.deleteMany({})
```

Info

Aggregation Pipelines que se ejecutan con el método `db.collection.aggregate()` no modifican los documentos de una colección, a menos que la canalización contenga una etapa `$merge` o `$out`.

```
1 db.orders.insertMany([
2   { _id: 0, name: "Pepperoni", size: "small", price: 19,
3     quantity: 10, date: ISODate( "2021-03-13T08:14:30Z" ) },
4   { _id: 1, name: "Pepperoni", size: "medium", price: 20,
5     quantity: 20, date : ISODate( "2021-03-13T09:13:24Z" ) },
6   { _id: 2, name: "Pepperoni", size: "large", price: 21,
7     quantity: 30, date : ISODate( "2021-03-17T09:22:12Z" ) },
8   { _id: 3, name: "Cheese", size: "small", price: 12,
```

```

9      quantity: 15, date : ISODate( "2021-03-13T11:21:39.736Z" ) },
10     { _id: 4, name: "Cheese", size: "medium", price: 13,
11       quantity:50, date : ISODate( "2022-01-12T21:23:13.331Z" ) },
12     { _id: 5, name: "Cheese", size: "large", price: 14,
13       quantity: 10, date : ISODate( "2022-01-12T05:08:13Z" ) },
14     { _id: 6, name: "Vegan", size: "small", price: 17,
15       quantity: 10, date : ISODate( "2021-01-13T05:08:13Z" ) },
16     { _id: 7, name: "Vegan", size: "medium", price: 18,
17       quantity: 10, date : ISODate( "2021-01-13T05:10:13Z" ) }
18   ] )

```

Calcular la cantidad total del pedido

Este ejemplo contiene dos etapas y devuelve la cantidad total del pedido de pizzas de tamaño "medium" agrupadas por nombre de pizza:

```

1 db.orders.aggregate( [
2
3   // Etapa 1: Filtramos los documentos por el tamaño de la pizza
4   {
5     $match: { size: "medium" }
6   },
7
8   // Etapa 2: Agrupe los documentos restantes por nombre de pizza y
9   // calcule la cantidad total
10  {
11    $group: { _id: "$name", totalQuantity: { $sum: "$quantity" } }
12  }
13 ]
)

```

La etapa de `$match`:

- Filtra los documentos de pedido de pizza a pizzas de tamaño mediano.
- Pasa los documentos restantes a la etapa de `$group`.

La fase de `$group`:

- Agrupa los documentos restantes por nombre de pizza.
- Utiliza `$sum` para calcular la cantidad total del pedido para cada `$name` de pizza. El total se almacena en el campo `totalQuantity` devuelto por la aggregation pipeline.

La salida del ejemplo sería:

```

1 [
2   { _id: 'Cheese', totalQuantity: 50 },
3   { _id: 'Vegan', totalQuantity: 10 },
4   { _id: 'Pepperoni', totalQuantity: 20 }
5 ]

```

Calcular el valor total del pedido y la cantidad promedio del pedido

El siguiente ejemplo calcula el valor total del pedido de pizza y la cantidad promedio del pedido entre dos fechas:

```

1 db.orders.aggregate( [
2
3     // Etapa 1: Filtramos los documentos de pedidos de pizza por rango de
fecha
4     {
5         $match:
6         {
7             "date": { $gte: new ISODate( "2020-01-30" ), $lt: new ISODate(
"2022-01-30" ) }
8         }
9     },
10
11     // Etapa 2: Agrupamos los documentos recibidos por $match por fecha y
calculamos los resultados
12     {
13         $group:
14         {
15             _id: { $dateToString: { format: "%Y-%m-%d", date: "$date" } },
16             totalOrderValue: { $sum: { $multiply: [ "$price", "$quantity" ] }
17         },
18         averageOrderQuantity: { $avg: "$quantity" }
19     },
20
21     // Etapa 3: Ordenamos los documentos por por totalOrderValue en orden
descendente
22     {
23         $sort: { totalOrderValue: -1 }
24     }
25
26 ] )

```

La etapa de `$match`:

- Filtra los documentos de pedido de pizza en un rango de fechas especificado usando `$gte` y `$lt`.
- Pasa los documentos restantes a la fase de `$group`.

La etapa `$group`:

- Agrupa los documentos por fecha usando `$dateToString`.
- Para cada grupo, calcula:
 - Valor total del pedido usando `$sum` y `$multiply`.
 - Cantidad promedio de pedido usando `$avg`.
- Pasa los documentos agrupados a la etapa `$sort`.

La etapa de `$sort`:

- Ordena los documentos por el valor total del pedido para cada grupo en orden descendente (-1).
- Devuelve los documentos ordenados.

La salida del ejemplo sería:

```

1   [
2     { _id: '2022-01-12', totalOrderValue: 790, averageOrderQuantity: 30 },
3     { _id: '2021-03-13', totalOrderValue: 770, averageOrderQuantity: 15 },
4     { _id: '2021-03-17', totalOrderValue: 630, averageOrderQuantity: 30 },
5     { _id: '2021-01-13', totalOrderValue: 350, averageOrderQuantity: 10 }
6   ]

```

6. Ejemplo

Usando la collection de restaurantes cargada anteriormente, vamos a resolver como ejemplo, algunas operaciones:

Ejemplo en Atlas

Si quieras hacerlo desde Mongo Atlas, podemos importarla con `mongoimport`. **Recuerda tener correctamente habilitado tu usuario y la ip añadir la ip a la whitelist**

```

1 mongoimport --uri "mongodb+srv://cluster0.zb0o3.mongodb.net/" --db=clase --
collection=restaurantes --file=/tmp/restaurantes1.csv --username tuusuario --
password tucontraseña

```

1. Comprobamos la carga

```

1 show dbs
2 db.restaurantes.find()

```

2. Crear una consultar para encontrar qué restaurantes no tienen dirección (Todas tienen dirección)

```

1 db.restaurantes.find({address:{$exists:false}})

```

3. Contar el número de restaurantes que si tienen dirección

```

1 db.restaurantes.find({address:{$exists:true}}).count()

```

4. Crear una consulta para encontrar aquellos restaurantes de cocina italiana que se encuentren en la zona geográfica con código postal 10075

```
1 | db.restaurantes.find({$and:[ {"cuisine":"Italian"}, {"address.zipcode":"10075"} ]})
```

5. Encontrar aquellos restaurantes que tengan grado A, puntuación 11 y fecha 2014-10-01T00:00:00Z

```
1 | db.restaurantes.find({grades:{ "date":ISODate("2014-10-01T00:00:00Z"), "grade":"A", "score":11}})
```

5. Contabiliza cuántos restaurantes tienen una puntuación menor o igual a 5

```
1 | db.restaurantes.find({"grades.score":{$lte:5}}).count()
```

6. Obtener los nombres del segundo y el tercer restaurante de cocina italiana ordenados por nombre

```
1 | db.restaurantes.find({"cuisine":"Italian"}).sort({name:1}).limit(2).skip(1)
```

7. Añadir una valoración al restaurante 41156888

```
1 | db.restaurantes.find({"restaurant_id":"41156888"}) // Listamos primero para ver las que tiene actualmente
2 | db.restaurantes.updateOne({"restaurant_id":"41156888"}, {$push:{grades: {"date":ISODate("2016-01-02T00:00:00.000Z"), "grade":"A", "score":14}}})
3 | db.restaurantes.find({"restaurant_id":"41156888"}) // Comprobamos que se ha añadido correctamente
```