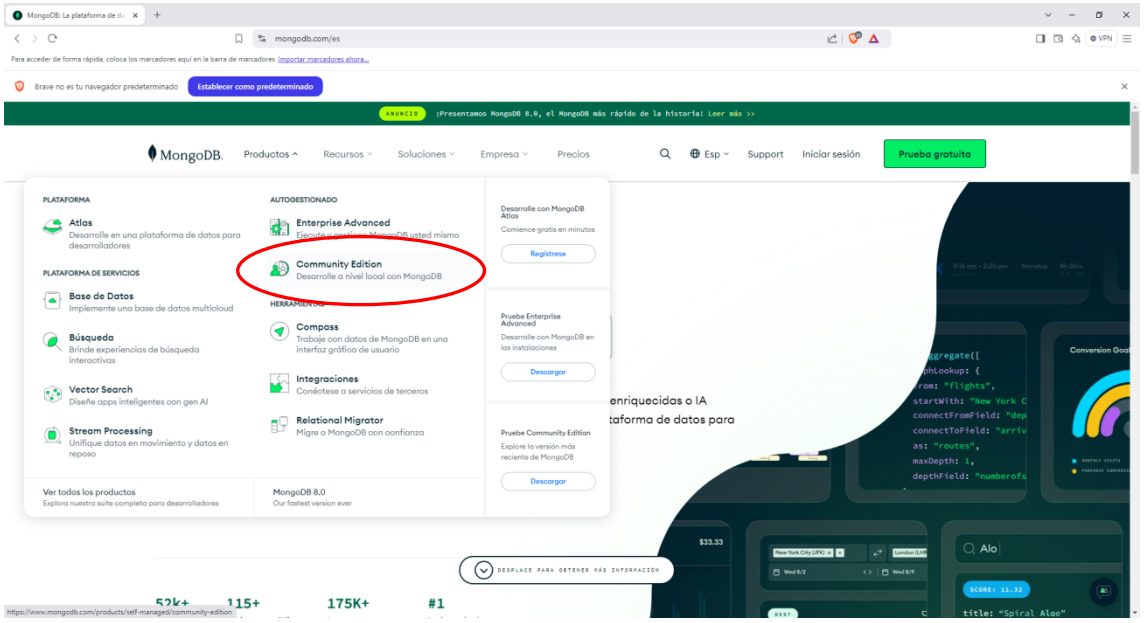
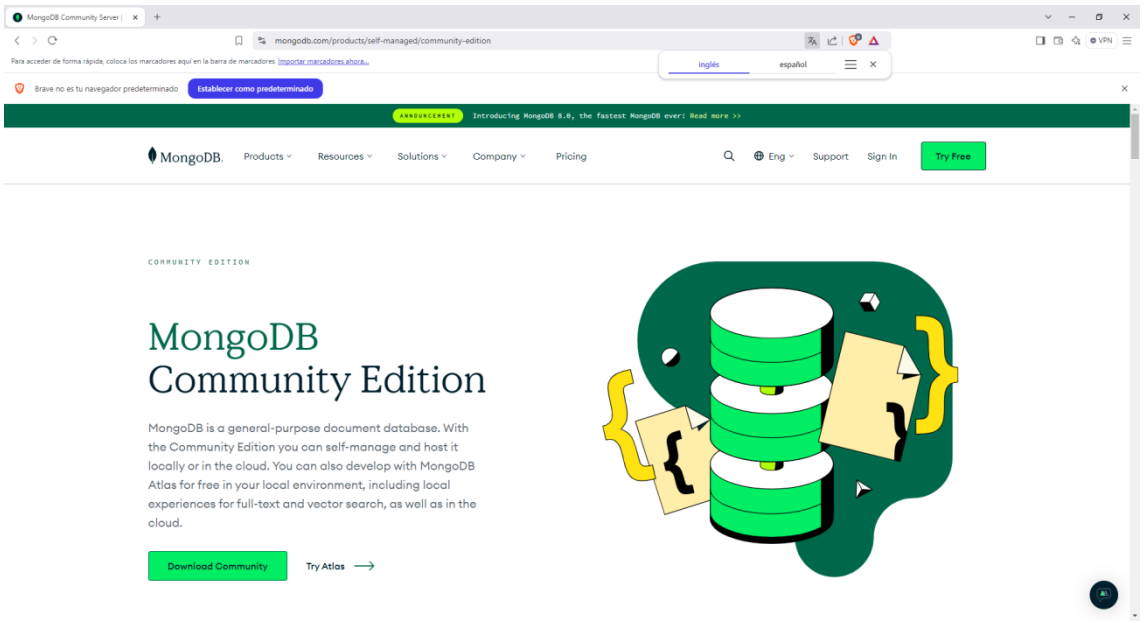
Mongo DB

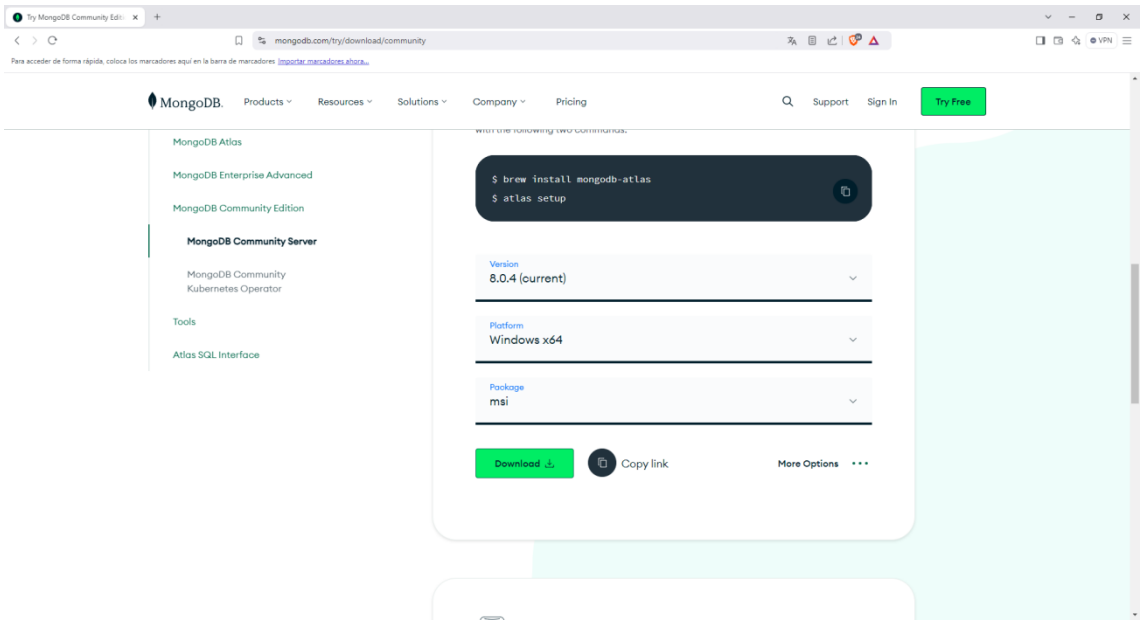
Instalación:

Descargaremos la versión comunity

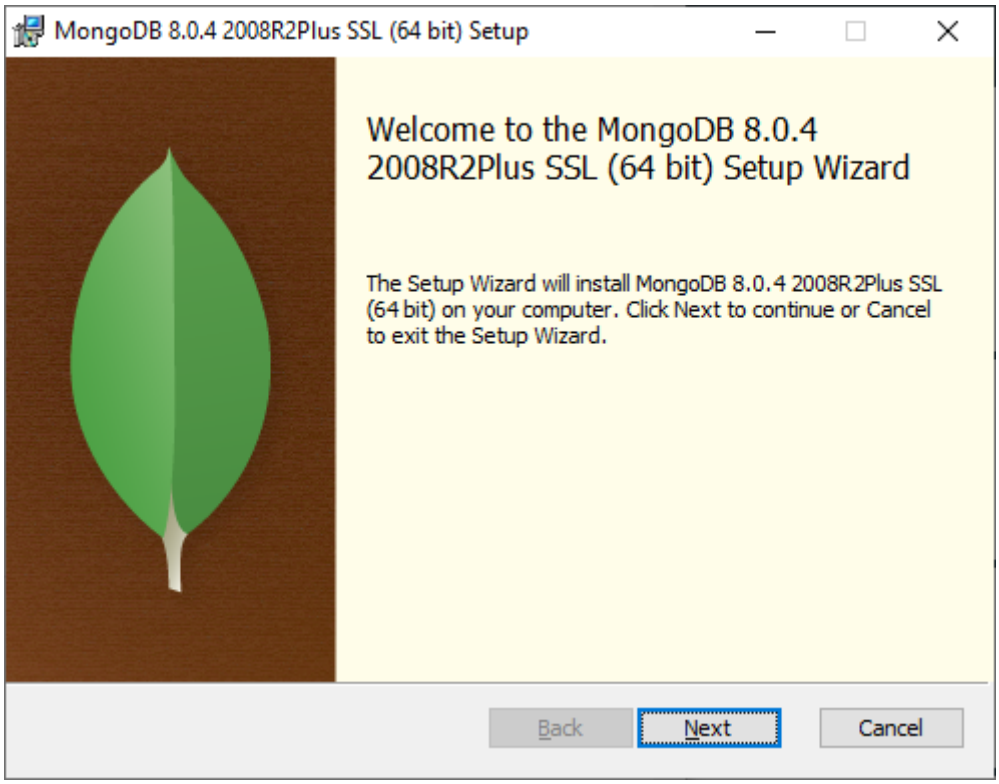
https://www.mongodb.com/try/download/community

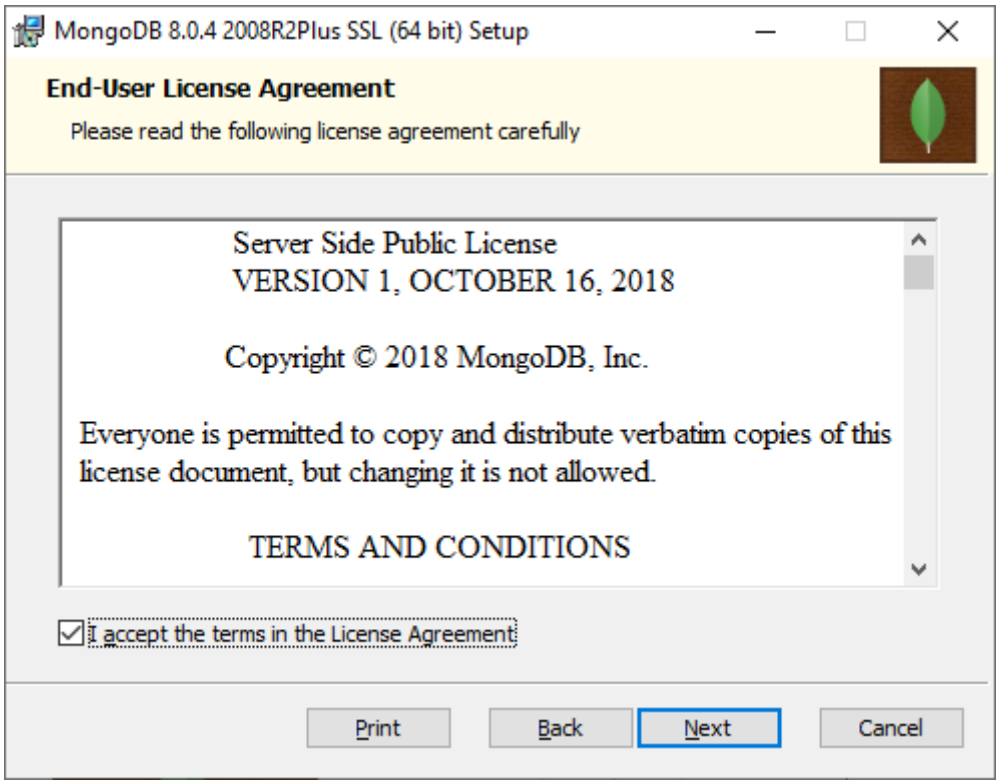


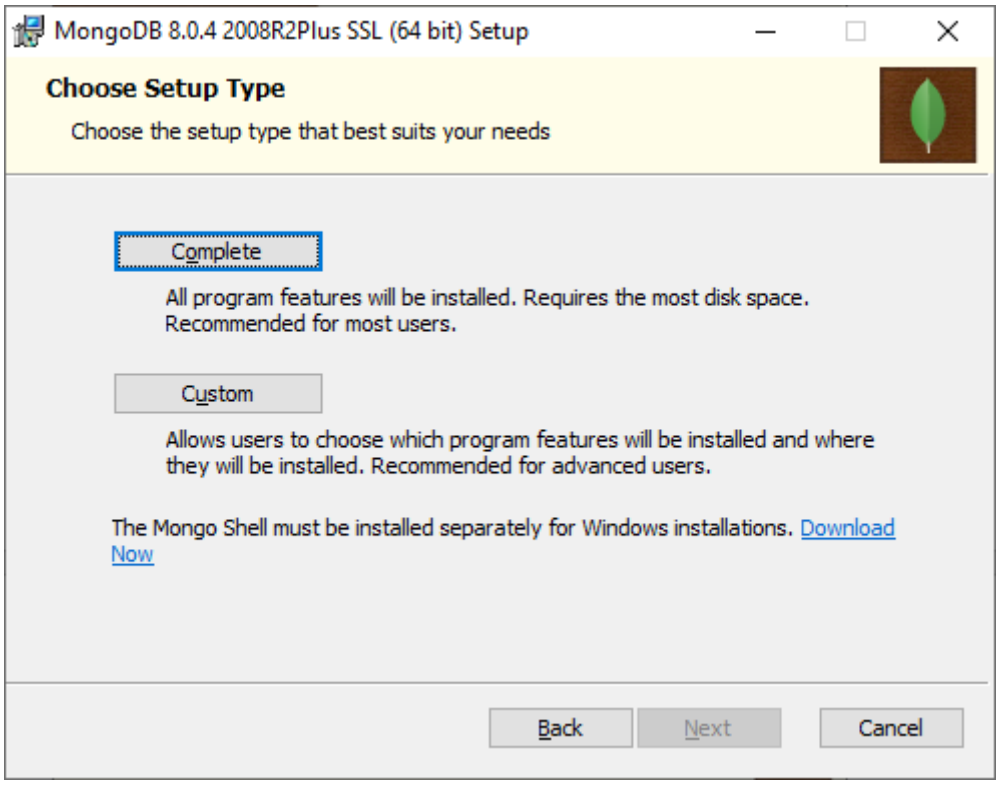


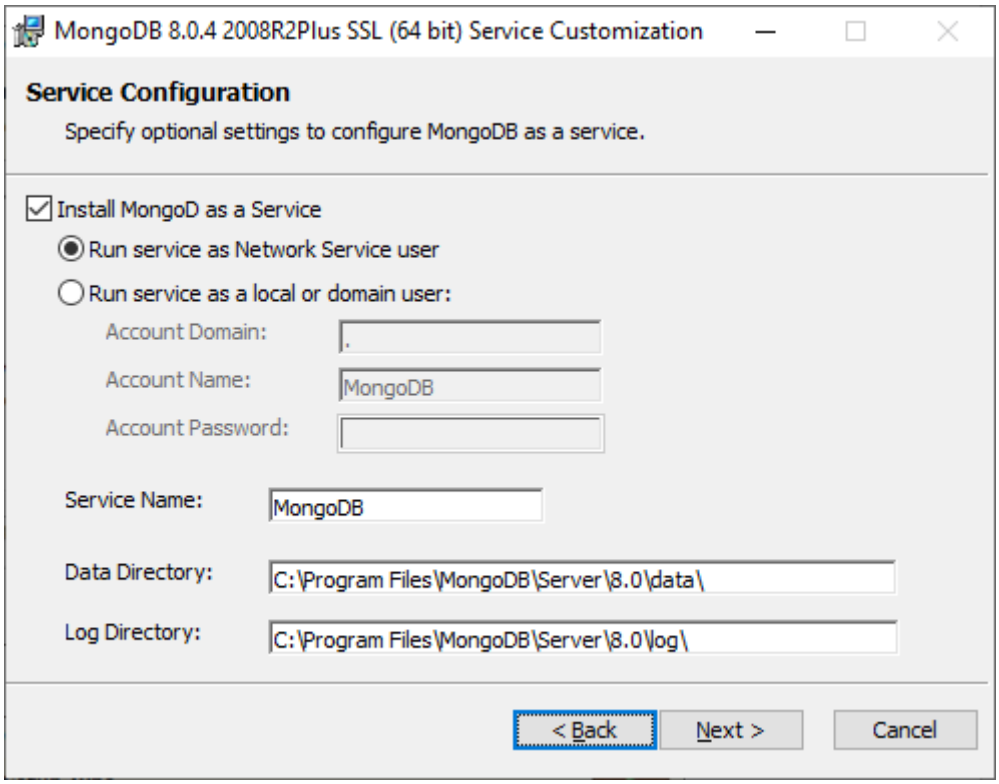


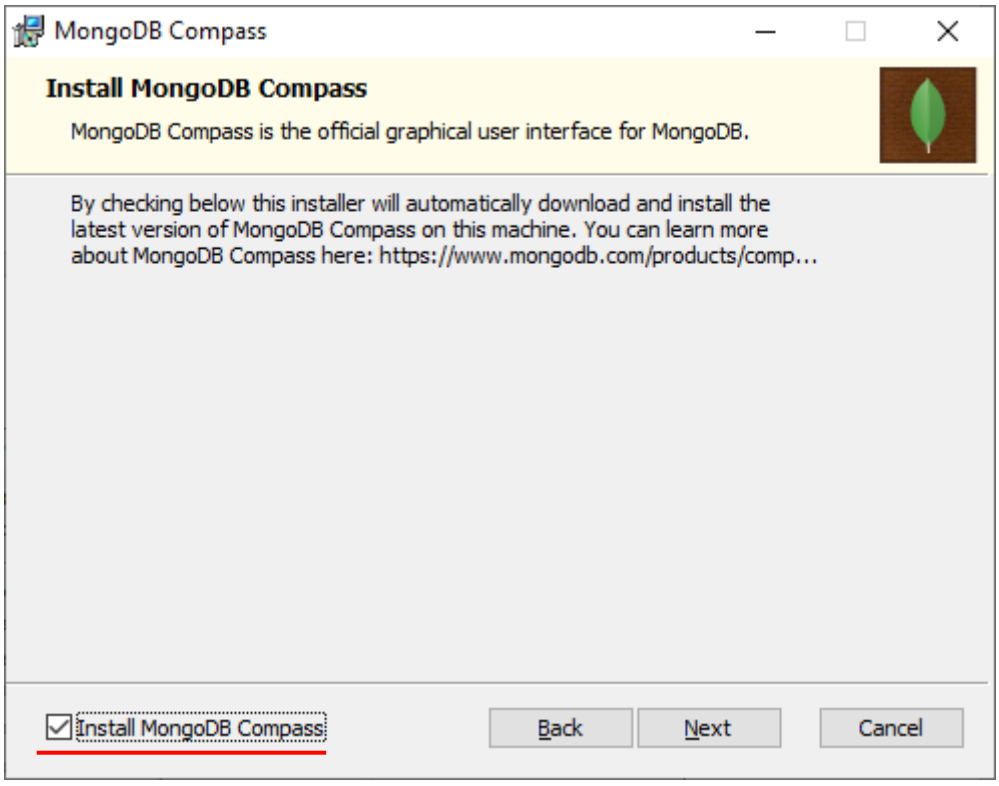
Típica instalación, siguiente, siguiente, aceptar.




MongoDB 8.0.4 2008R2Plus SSL (64 bit) Setup
End-User License Agreement
Please read the following license agreement carefully
Server Side Public License
VERSION 1, OCTOBER 16, 2018
Copyright © 2018 MongoDB, Inc.
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.
TERMS AND CONDITIONS
I accept the terms in the License Agreement
Print
Back
Next
Cancel

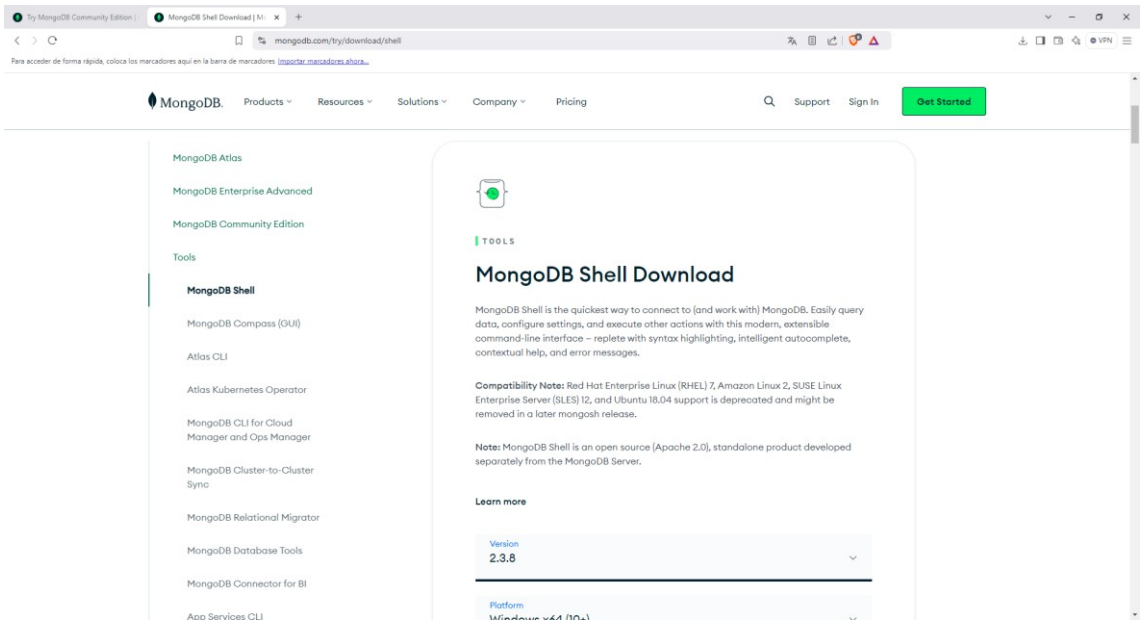

MongoDB 8.0.4 2008R2Plus SSL (64 bit) Setup
Choose Setup Type
Choose the setup type that best suits your needs
Complete
All program features will be installed. Requires the most disk space.
Recommended for most users.
Custom
Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.
The Mongo Shell must be installed separately for Windows installations. Download Now
Back
Next
Cancel



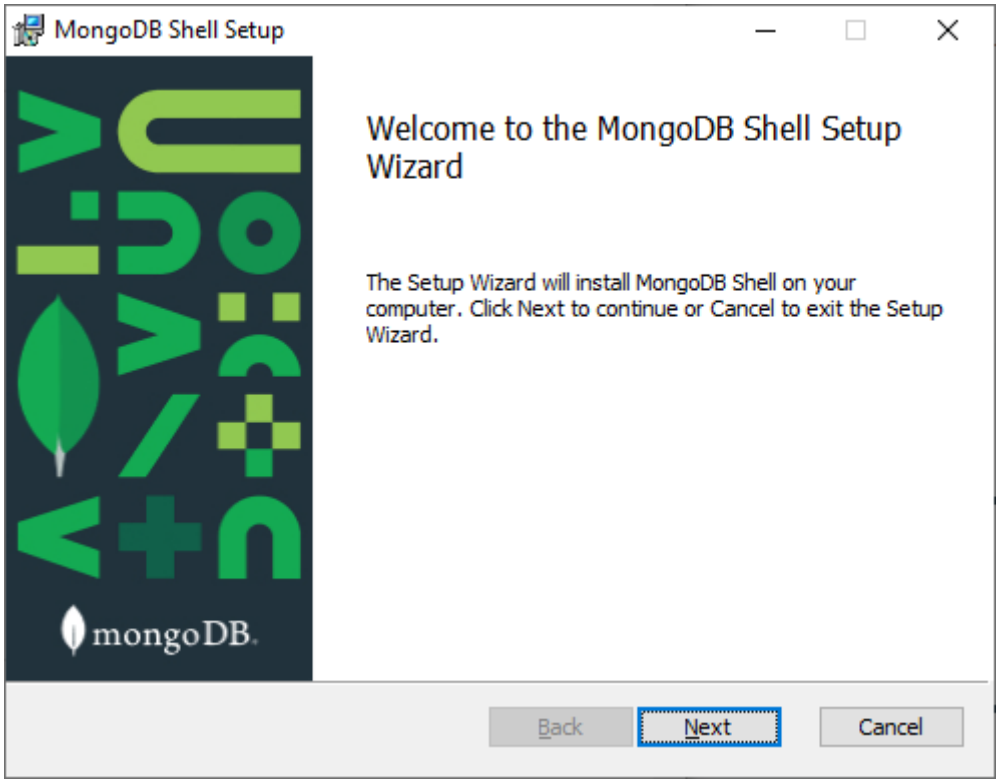


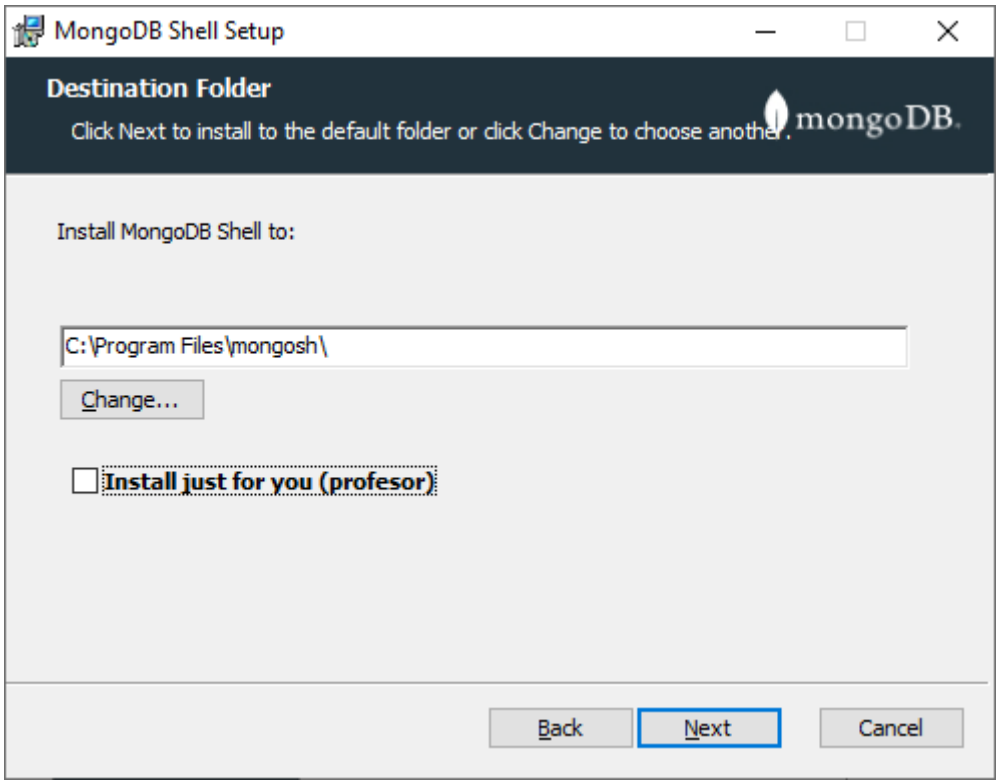
Mongo Compass es una interfaz gráfica.

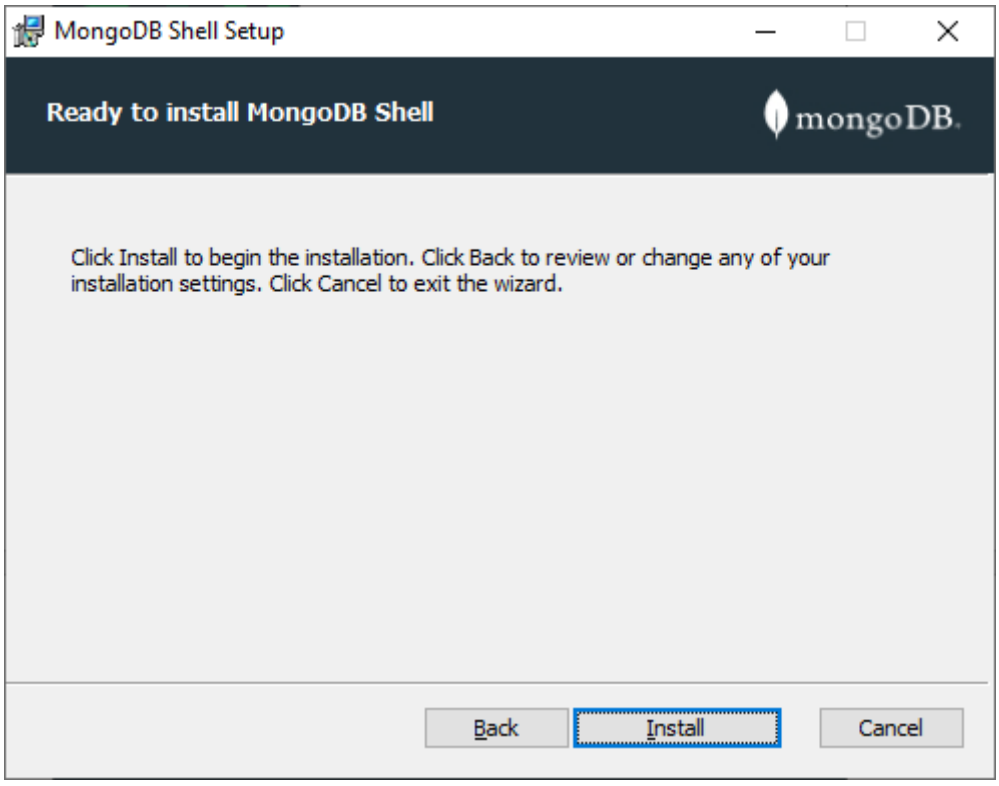
Instalación de Mongo Shell:



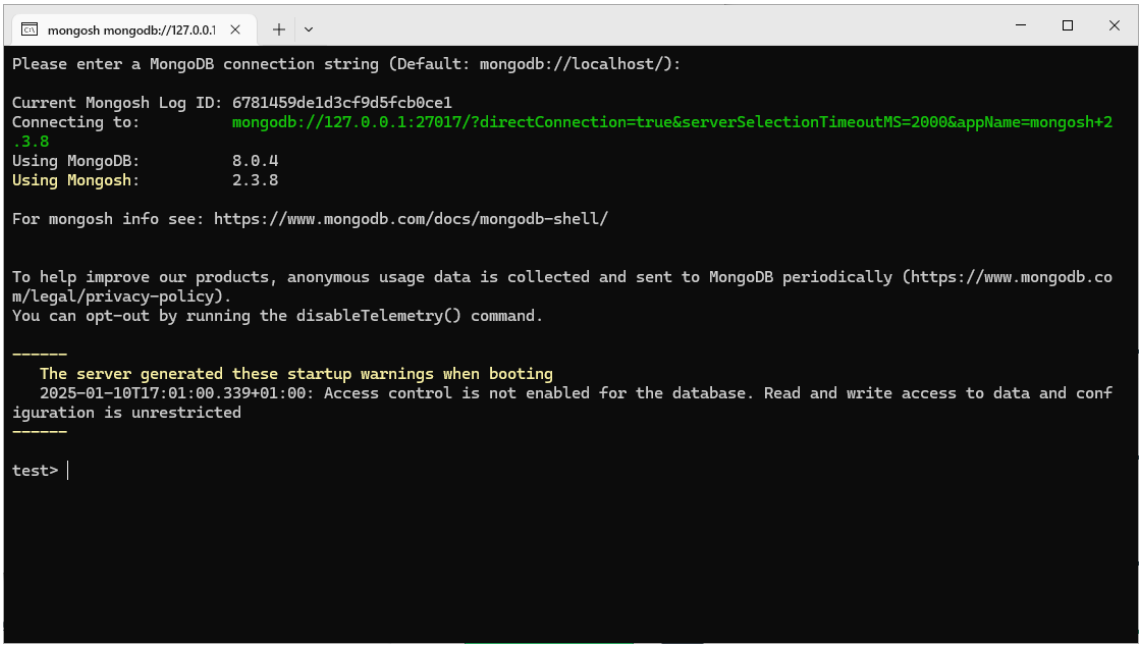
Típica instalación siguiente, siguiente aceptar.




MongoDB Shell Setup
Destination Folder
Click Next to install to the default folder or click Change to choose another.
mongoDB.
Install MongoDB Shell to:
C:\Program Files\mongosh\
Change...
Install just for you (profesor)
Back
Next
Cancel


MongoDB Shell Setup
Ready to install MongoDB Shell
mongoDB.
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
Back
Install
Cancel

Mongo Shell:



Historia de MongoDB

MongoDB fue desarrollado por la empresa 10gen (ahora MongoDB Inc.) en 2007. Surgió como una solución a las limitaciones de las bases de datos relacionales tradicionales en términos de escalabilidad y flexibilidad. La idea era crear una base de datos que pudiera manejar grandes volúmenes de datos y proporcionar una estructura más flexible para el almacenamiento y la recuperación de datos.

¿Qué es MongoDB?

MongoDB es una base de datos NoSQL orientada a documentos que utiliza documentos similares a JSON con esquemas dinámicos. Esto significa que los datos se almacenan en documentos BSON (una extensión binaria de JSON), lo que permite una representación más rica y flexible de los datos.

Características importantes de MongoDB

1.- Modelo de datos flexible: MongoDB permite almacenar datos en documentos BSON, lo que proporciona una estructura flexible y dinámica. No es necesario definir un esquema fijo, lo que facilita la evolución de la base de datos a medida que cambian los requisitos de la aplicación.

2.- Escalabilidad horizontal: MongoDB está diseñado para escalar horizontalmente mediante el sharding, que distribuye los datos en múltiples servidores. Esto permite manejar grandes volúmenes de datos y altas tasas de tráfico.

3.- Alta disponibilidad: MongoDB proporciona alta disponibilidad mediante la replicación. Los datos se replican en múltiples servidores, lo que garantiza la redundancia y la tolerancia a fallos.

4.- Consultas avanzadas: MongoDB soporta consultas ad-hoc, índices secundarios, y agregaciones, lo que permite realizar consultas complejas y análisis de datos.
5.- Alto rendimiento: MongoDB está optimizado para el rendimiento, con características como la indexación avanzada, el almacenamiento en memoria y la compresión de datos.

Mejoras respecto a las bases de datos relacionales

1.- Flexibilidad del esquema: A diferencia de las bases de datos relacionales, MongoDB no requiere un esquema fijo. Esto permite una mayor flexibilidad en la estructura de los datos y facilita la adaptación a cambios en los requisitos de la aplicación.
2.- Escalabilidad horizontal: Las bases de datos relacionales suelen escalar verticalmente (añadiendo más recursos a un solo servidor), lo que puede ser costoso y limitado. MongoDB, en cambio, está diseñado para escalar horizontalmente, distribuyendo los datos en múltiples servidores.
3.- Desempeño en grandes volúmenes de datos: MongoDB está optimizado para manejar grandes volúmenes de datos y altas tasas de tráfico, lo que puede ser un desafío para las bases de datos relacionales.
4.- Desarrollo ágil: La flexibilidad del esquema y la capacidad de almacenar datos en documentos BSON permiten un desarrollo más ágil y rápido, ya que no es necesario definir y modificar esquemas rígidos.
5.- Consultas y agregaciones avanzadas: MongoDB proporciona potentes capacidades de consulta y agregación que permiten realizar análisis de datos complejos de manera eficiente.

Concepto de NoSQL

NoSQL (Not Only SQL) se refiere a una categoría de sistemas de gestión de bases de datos que no se basan en el modelo relacional tradicional. Estas bases de datos están diseñadas para manejar grandes volúmenes de datos, alta velocidad de procesamiento y estructuras de datos flexibles.

Diferencias entre NoSQL y bases de datos relacionales

1. Modelo de datos:

→ Relacional: Utiliza tablas con filas y columnas, y requiere un esquema fijo.
→ NoSQL: Utiliza varios modelos de datos como documentos, grafos, clave-valor y columnas anchas, y no requiere un esquema fijo.

2. Escalabilidad:

→ Relacional: Escala verticalmente (añadiendo más recursos a un solo servidor).
→ NoSQL: Escala horizontalmente (añadiendo más servidores).

3. Flexibilidad del esquema:

→ Relacional: Requiere un esquema predefinido y rígido.

→ NoSQL: Permite esquemas dinámicos y flexibles.

4. Consultas:

→ Relacional: Utiliza SQL para consultas complejas y transacciones ACID.
→ NoSQL: Utiliza consultas específicas del modelo de datos y puede no soportar transacciones ACID completas.

5. Consistencia:

→ Relacional: Garantiza consistencia fuerte.
→ NoSQL: Ofrece consistencia eventual, lo que puede ser más adecuado para aplicaciones distribuidas.

- BSON: Breve descripción del formato de almacenamiento.

- **BSON**: MongoDB almacena datos en un formato llamado BSON (Binary JSON), que es una representación binaria de datos JSON. BSON extiende el modelo JSON con tipos de datos adicionales, como “date” y “binary data”, y permite una codificación más eficiente y un acceso más rápido a los datos. Aunque los documentos se presentan y pueden ser manipulados como JSON en las interfaces de usuario y en el código, internamente MongoDB trabaja con BSON para optimizar el almacenamiento y la velocidad de acceso.

Configuración Inicial y Creación de una Base de Datos de Prueba

Una vez instalado MongoDB, puedes realizar una configuración inicial y crear una base de datos de prueba:

1. Acceder al Shell de MongoDB: Si MongoDB está instalado localmente, abre una terminal y escribe “mongo”. Si usas Docker, conecta al contenedor con “docker exec -it mongodb bash” y luego ejecuta “mongo”.

2. Crear una Base de Datos de Prueba: En el shell de MongoDB, usa el comando “use nombreDeLaBaseDeDatos” para crear una nueva base de datos. Por ejemplo:

```
use miBaseDeDatosTest
```

3. Insertar Datos de Prueba: Puedes insertar algunos datos en tu nueva base de datos para probarla. Por ejemplo:

```
db.miColeccion.insert({nombre: "Ejemplo", valor: 123})
```

(En mongo, si no insertas un dato, la colección no se llega a crear.)

4. Consultar los Datos: Verifica que los datos se hayan insertado correctamente con:

```
db.miColeccion.find()
```

5. Mostrar las bases de datos existentes:

```
show dbs
```

Insertar datos:

Ya hemos visto “insertOne”, aunque también podemos insertar varios documentos:

```
db.collectionName.insertMany(<json>)
```

Filtrando Resultados

Para filtrar documentos basados en un campo específico:

```
db.collection.find({ campo: "valor" })
```

Consulta LIKE en SQL

Para realizar una operación similar al LIKE de SQL (buscar patrones en cadenas de texto), utiliza expresiones regulares en MongoDB:

```
db.collection.find({ campo: /patrón/ })
```

Por ejemplo, para buscar documentos donde el campo "nombre" contenga "Juan":

```
db.collection.find({ nombre: /Juan/ })
```

Operadores Comunes en MongoDB

A continuación, se muestra una lista de los operadores más comunes en MongoDB con su significado:

```
$gt (Greater Than) - Mayor que

$lt (Less Than) - Menor que

$gte (Greater Than or Equal) - Mayor o igual que

$lte (Less Than or Equal) - Menor o igual que

$ne (Not Equal) - No igual a

$in (In) - Dentro de una lista de valores

$nin (Not In) - No dentro de una lista de valores

$exists (Exists) - Campo existe o no

$regex (Regular Expression) - Expresión regular
```

Consulta con AND/OR

Para realizar consultas con condiciones lógicas AND y OR:

// AND

```
db.collection.find({ $and: [{ campo1: "valor1" }, { campo2: "valor2" }] })
```

// OR

```
db.collection.find({ $or: [{ campo1: "valor1" }, { campo2: "valor2" }] })
```

Ordenar Resultados

Para ordenar resultados en una consulta:

```
db.collection.find({}).sort({ campo: 1 }) // Ascendente

db.collection.find({}).sort({ campo: -1 }) // Descendente
```

Limitar Resultados

Para limitar la cantidad de resultados devueltos:

```
db.collection.find({}).limit(5)
```

Consultas Anidadas

Para realizar consultas dentro de documentos anidados:

```
db.collection.find({ "campo.subcampo": "valor" })
```

Eliminar documentos:

Un solo documento:

```
db.collectionName.deleteOne(<json>);
```

Varios documentos:

```
db.collectionName.deleteMany(<json>);
```

Actualizar documentos:

```
db.collection.updateOne(<filter>, <update>)

db.collection.updateMany(<filter>, <update)

db.collection.replaceOne(<filter>, <update>)
```

Diseño de Bases de datos NoSQL:

Se conoce como SCHEMALESS o Sin esquema.

En las bases de datos SQL, antes de guardar datos debemos saber los campos y tipos de datos a guardar, lo que se traduce en la existencia de restricciones.

NoSQL:

Datos desestructurados o semiestructurados, unos pueden tener unos campos, otros campos.

Mejor gestión de memoria, al no tener que guardar campos vacios unos veces sí y otras no. Aporta flexibilidad

Tipos de diseño:

Documentos embebidos

- Hace referencia a guardar una cosa dentro de otra.
- En este caso, guardar un documento JSON dentro de otro como valor de una de sus propiedades

Imagina que en la universidad se imparte un curso de especialización sobre física, y dicho curso se compone de varias asignaturas:

```
{
"id" :1,
"referencta": "C001"
"nombre": "Física" ,
"finicio": "12/01/2019" ,
"esGratuito": true,
"asignaturas": [

{
"id" : 2,
"nombre": "Física elemental"
"duracion": 5
},
{
"id" : 3,
"nombre": "Termodinámica"
"duracion": 5
}
]
}
```

Documentos referenciados:

A diferencia de los documentos embebidos, en un documento JSON se guarda solo el valor de una o varias propiedades, en lugar del documento completo.

Normalmente, se guarda el valor de una propiedad que identifica unívocamente, al documento referenciado:

```
{
"id" :1,
"referencta": "C001"
"nombre": "Física" ,
"finicio": "12/01/2019" ,
"esGratuito": true,
"asignaturas": [2,3]
}
{
"id" : 2,
"nombre": "Física elemental"
"duracion": 5
},
{
"id" : 3,
"nombre": "Termodinámica"
"duracion": 5
}
```

Ventajas y desventajas:

Documentos embebidos:

Ventajas

- Al recuperar un curso, podemos traernos toda la información relacionada en otras colecciones (p.e. asignaturas)

Desventajas:

- Al hacer consultas sobre la colección externa (cursos), si uno de los criterios afecta a la información de los documentos embebidos, el tiempo para realizar dicha consulta se verá incrementando.
- Las actualizaciones serán más costosas si alguna de las propiedades a actualizar pertenece al documento embebido

Documentos referenciados

Ventajas

- Las consultas sobre la colección principal y las relacionadas se ejecutará más rápido.
- Al actualizar información relacionada, se hace directamente en sus documentos sin tener que revisar la colección externa.

Desventajas:

- Al recuperar un curso, habrá que consultar los identificadores que relacionan a los documentos en otras colecciones y hacer consultas adicionales para obtener la información relacionada.

¿Qué diseño elegir?

Dependerá de:

- Cómo se quiere almacenar la información.
- La naturaleza y el contexto de las aplicaciones que vayan a consumir la información.
- Las preferencias de roles como arquitectos de software y de bases de datos teniendo en cuenta factores futuros como la escalabilidad en cuanto a volumen de datos, usuarios / aplicaciones y sus formas de acceder a la información, etc.