

# MongoDB

Mikel Egaña Aranguren

[mikel-egana-aranguren.github.io](https://mikel-egana-aranguren.github.io)

[mikel.egana@ehu.eus](mailto:mikel.egana@ehu.eus)



# MongoDB

<https://github.com/mikel-egana-aranguren/ABD>



# MongoDB

<https://www.mongodb.com/>

BD NoSQL orientada a documentos (JSON)

Su nombre viene de la palabra Humongous (“gigantesco”)

# MongoDB

- MongoDB Community Edition: Versión libre
- MongoDB Enterprise: Versión comercial
- MongoSH: cliente Shell
- Atlas: Servicio en la nube
- Herramientas adicionales: Compass, Atlas CLI, VS Code Plugin, ...
- [Documentación](#)

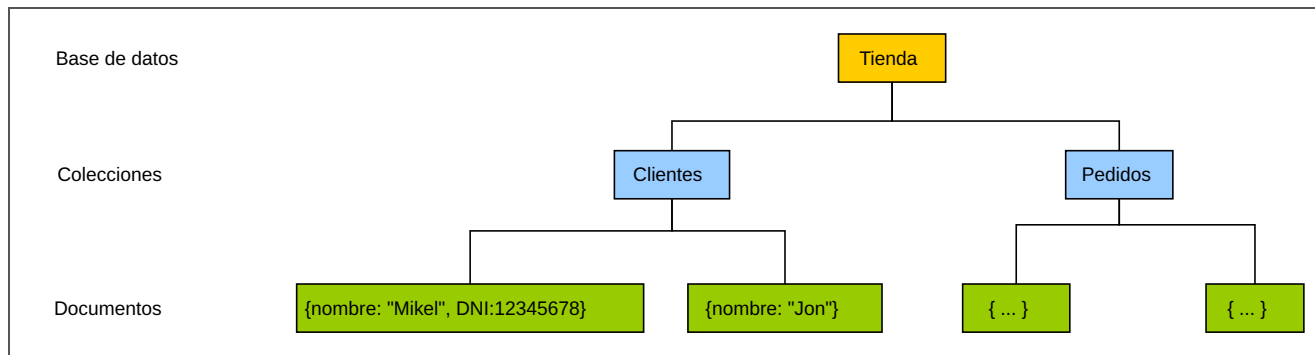
# Instalación Ubuntu

1. Importar clave GPG (Instalar gpg, curl)
2. Crear archivo sources.list (24.04, 22.04, 20.04)
3. Recargar los paquetes disponibles en APT
4. Instalar mediante APT

# Instalación Docker

1. <https://hub.docker.com/>
2. mongodb/mongodb-community-server
3. `docker pull mongodb/mongodb-community-server`
4. `docker run --name mongodb -p 27017:27017 -d mongodb/mongodb-community-server:latest`

# Estructura de datos



# Estructura de datos

Un documento se organiza en formato [JSON](#) (JavaScript Object Notation):

Internamente se almacena en formato [Binary JSON](#) (BSON)

Cada dato como clave valor



# Estructura de datos

```
{  
  nombre: "Mikel"  
  email: "mikel.egana@ehu.eus"  
  direccion:  
    {  
      ciudad: "Bilbao"  
    }  
  telefonos: [  
    {  
      despacho: 946014786  
    },  
    {  
      movil: 666777888  
    }  
  ]  
}
```

Un valor puede ser otro documento

Un valor puede contener un listado de documentos

# Estructura de datos

Equivalencia aproximada con modelo relacional:

- Tabla - colección
- Fila - documento
- Columna - clave
- Joins - integrados en documentos

# Estructura de datos

Diferencias principales con modelo relacional:

- No todos los documentos tienen por qué tener las mismas claves
- No es necesario definir relaciones explícitas entre documentos

# Directorios importantes

- /var/log/mongodb/
- /var/lib/mongodb/

# Comandos

- MongoSH:  
\$ mongosh
- Mostrar BBDD existentes:  
> show dbs
- Mostrar BBDD en uso:  
> db

# Comandos

- Crear/meterse en DB:  
> use nombre-BBDD
- Limpiar la pantalla:  
> cls
- Tabular!!!!  
> db.

# Comandos

Crear coleccion vacia (! NoSQL):

- `> use tienda`
- `> db.createCollection("clientes")`
- `> show dbs`
- `> show collections`

# Comandos

Crear coleccion con un documento:

- `> use tienda`
- `> db.[coleccion].insertOne([documento])`
- `> db.clientes.insertOne({"nombre": "mikel"})`



# Comandos

- Cada documento tiene un identificador único que se asigna automáticamente en su creación

- Asignar ID de forma manual:

```
> db.clientes.insertOne ( { _id:1 , nombre:"josu" } )
```

- Mostrar todos los documentos:

```
> db.[nombre-colección].find()
```

# Comandos

Insertar múltiples documentos en una colección:

```
> db.[nombre-colección].insertMany( [array-docs] )
```

```
>
```

```
db.departamentos.insertMany([{"nombre": "Contabilidad", "empleados": 5},  
{"nombre": "Almacén", "empleados": 5}])
```

# Comandos

- Mostrar los documentos que encajen con un patrón:

```
> db.[nombre-colección].find([patrón])
```

- Encontrar el 1er documento que encaje con un patrón:

```
> db.[nombre-colección].findOne([patrón])
```

# Comandos

- Condicion: {}
- Buscar en la colección libros los libros con editorial “Biblio”:  

```
> db.libros.find({editorial:"Biblio"})
```
- Buscar en la colección libros los libros con editorial “Biblio” y cantidad de 12:  

```
> db.libros.find({editorial:"Biblio", cantidad: 12})
```

# Comandos

- {columna: {operador:valor}, ...}
- Operadores de comparación (eq,ne,gt,gte,lt,lte):  

```
> db.libros.find({editorial:"Biblio", cantidad: { $gt: 20 } })
```
- Expresiones regulares:  

```
> db.libros.find({editorial:"Biblio", titulo : /Don.* / })
```

# Comandos

- AND
- `> db.libros.find({condicion1,condicion2,condicion3})`
- `> db.libros.find({$and : [{condicion1},{condicion2},{condicion3}]})`
- `> db.libros.find({ precio : {$gt: 25}, cantidad : {$lt : 25}})`

# Comandos

- OR
- ```
> db.libros.find({ $or : [ {precio : {$gt: 25}},  
  {cantidad : {$lt : 25}}]})
```

-----



# Comandos

- Campos a mostrar:

```
{ campoAMostrar: 1, campoAOCultar: 0}
```

- Recuperar sólo el DNI de los documentos en la colección “clientes”:

```
> db.clientes.find( {}, {DNI: 1, _id: 0})
```

# Comandos

- Eliminar el 1er documento que encaje con patrón:  

```
> db.[nombre-colección].deleteOne([patrón])
```
- Eliminar los documentos que encajen con patrón:  

```
> db.[nombre-colección].deleteMany([patrón])
```
- Eliminar todos los documentos de una colección:  

```
> db.[nombre-colección].deleteMany( {} )
```

# Comandos

- Modificar documentos en una colección:  

```
> db.[nombre-colección].updateMany([patrón-busq],  
{$set: [cambios]})
```
- Para los clientes cuyo DNI sea 22233, actualizar su nombre a “Nagore”:  

```
> db.clientes.updateMany({ DNI: 22233 }, {$set:{  
nombre: "Nagore" } })
```

# Comandos

- Reemplazar un documento por otro:

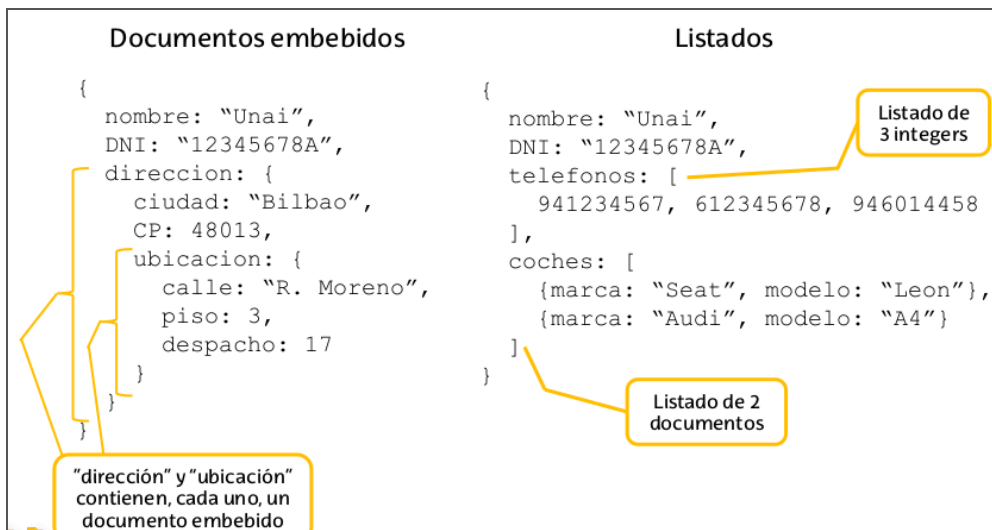
```
> db.[nombre-colección].replaceOne([patrón-busq],  
[nuevo-doc] )
```

# Comandos

- Mostrar colecciones en una BD:  
`> show collections`
- Eliminar una colección completa:  
`> db.[nombre-colección].drop()`
- Eliminar una BD:  
`> db.dropDatabase()`

# Relaciones

Un documento puede tener documentos embebidos o listados



# Relaciones

Normalmente:

- Varias colecciones, cada una representa una entidad del contexto: clientes, productos, pedidos
- Relaciones entre los datos de diferentes entidades

MongoDB no proporciona una técnica concreta para definir relaciones entre colecciones: las debemos definir nosotros

# Relaciones

Documentos embebidos

Utilizar campos concretos como referencia



# Documentos embebidos

## *Colección "clientes"*

```
{  
  nombre: "Unai",  
  DNI: "12345678A",  
  direccion:  
  {  
    ciudad: "Bilbao",  
    CP: 48013,  
    ...  
  }  
}
```

# Documentos embebidos

Adecuado para datos que no se solapan/repiten

(+) Los datos se agrupan lógicamente

(-) Puede generar duplicidades que debemos gestionar

# Referencias

## *Colección "clientes"*

```
{  
  nombre: "Unai",  
  DNI: "12345678A",  
  direcciones: "001",  
  ...  
}
```

## *Colección "direcciones"*

```
{  
  _id: "001",  
  ciudad: "Bilbao",  
  CP: 48013,  
  ...  
}
```

# Referencias

Adecuado para datos que se referencien en diferentes colecciones

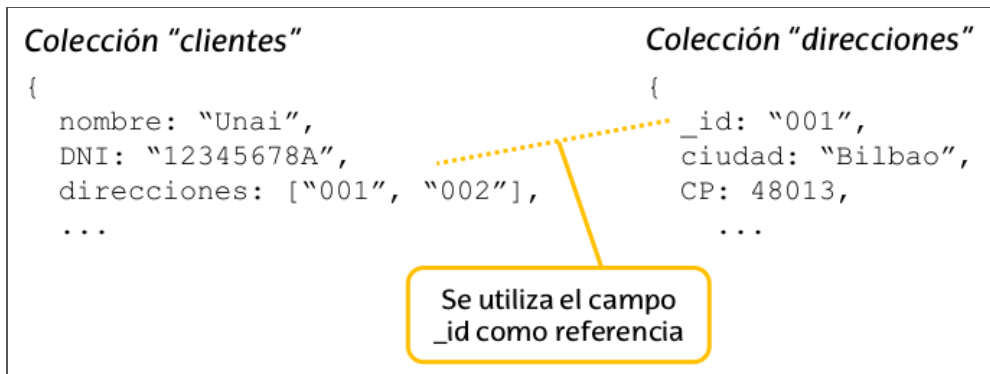
(+) Elimina posibles duplicidades

(-) Más complejo de gestionar

(-) Requiere agregaciones para obtener datos relacionados

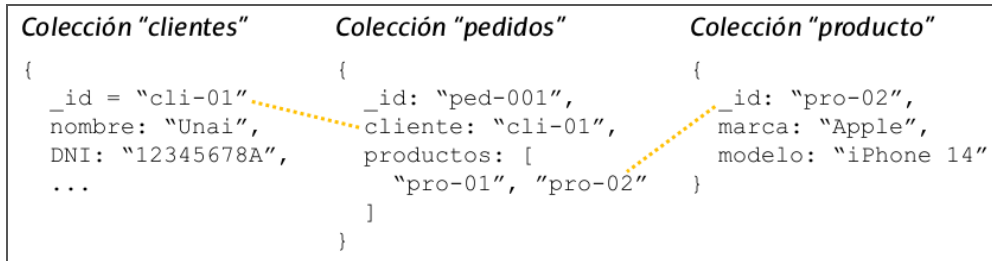
# Referencias

Relación entre los datos 1 a N: un cliente puede tener múltiples direcciones, una dirección pertenece sólo a 1 cliente



# Referencias

Relación entre los datos N a N: un cliente puede comprar múltiples productos, un producto puede ser comprado por múltiples clientes



# Referencias o documentos embebidos

Documentos embebidos:

- Datos que estén fuertemente relacionados y que no estén duplicados
- Relaciones 1 a 1: un cliente tendrá sólo una dirección asociada
- Relaciones 1 a N donde no haya duplicidades

# Referencias o documentos embebidos

## Referencias

- Datos de entidades independientes pero relacionadas
- Relaciones 1 a N
- Relaciones N a N



# Relaciones

Combinacion mediante agregacion ("Join")

## ***Colección "pedidos"***

```
{ _id: "ped-001",  
  cliente: "cli-01",  
  productos: ["pro-01"]  
}
```

## ***Colección "clientes"***

```
{ _id = "cli-01"  
  nombre: "Unai",  
  DNI: 45823150C } }
```

## ***Combinación***

```
{ _id: "ped-001",  
  cliente: "cli-01",  
  productos: ["pro-01"]  
  datosCliente: {  
    _id = "cli-01"  
    nombre: "Unai",  
    DNI: 45823150C  
  }  
}
```



# Agregacion mediante \$lookup

Ejemplo: Crear una colección que contenga los datos de los pedidos combinados con los datos de los clientes

The diagram illustrates the MongoDB `$lookup` aggregation stage. It shows a code snippet with callouts explaining its components:

- Colección origen**: Points to the `db.pedidos.aggregate` part of the code.
- Colección a combinar**: Points to the `from: "clientes"` field in the `$lookup` stage.
- Campo para combinación en origen**: Points to the `localField: "cliente"` field.
- Campo para combinación en colección a combinar**: Points to the `foreignField: "_id"` field.
- Nombre para el campo que incluirá combinados**: Points to the `as: "datosClientes"` field.

**Colección "pedidos"**

```
{ _id: "ped-001",  
  cliente: "cli-01",  
  productos: ["pro-01"]  
}
```

**Colección "clientes"**

```
{ _id = "cli-01"  
  nombre: "Unai",  
  DNI: 45823150C }  
}
```

# Esquemas

Es posible gestionar BBDD en MongoDB sin definir ningún tipo de estructura

Pero en algunas situaciones puede que queramos controlar los datos de forma automática: P.e. que todos los documento de una colección “productos” tienen un campo numérico “precio”

# Esquemas (Fuente: M. Schwarzmüller)

| Caos total                                                         | Punto intermedio                                                                       | Estilo SQL                                                     |
|--------------------------------------------------------------------|----------------------------------------------------------------------------------------|----------------------------------------------------------------|
| <b>Colección: Productos</b>                                        | <b>Colección: Productos</b>                                                            | <b>Colección: Productos</b>                                    |
| <pre>{<br/>  nombre: "Libro",<br/>  precio: 5.99<br/>}</pre>       | <pre>{<br/>  nombre: "Libro",<br/>  precio: 5.99<br/>}</pre>                           | <pre>{<br/>  nombre: "Libro",<br/>  precio: 5.99<br/>}</pre>   |
| <pre>{<br/>  titulo: "Botella",<br/>  disponible: true<br/>}</pre> | <pre>{<br/>  nombre: "Botella",<br/>  precio: 2.55,<br/>  disponible: true<br/>}</pre> | <pre>{<br/>  nombre: "Botella",<br/>  precio: 2.55<br/>}</pre> |

# Esquemas

Se puede definir la estructura que deben cumplir los documentos de una colección

Un esquema verifica cada documento insertado en una colección y genera un aviso/error si es incorrecto

# Esquemas

validationLevel: Controla cómo de estricta es la validación

- strict (valor por defecto): Se comprueba toda inserción y modificación
- moderate: Las modificaciones a documentos ya existentes no se comprueban

# Esquemas

`validationAction`: Indica qué hacer cuando un documento no cumple el esquema

- `error` (valor por defecto): Se emite un error y se impide la inserción
- `warn`: Se escribe un aviso en el log de MongoDB y se permite la inserción

# Esquemas

```
> db.createCollection("marcas", {  
  validator: {  
    $jsonSchema: {  
      bsonType: "object",  
      required: [ "nombre", "presupuesto" ],  
      properties: {  
        nombre: { bsonType: "string", description: "nombre de la marca"},  
        presupuesto: { bsonType: "int", minimum: 2000 }},  
      },  
    validationLevel: 'moderate',  
    validationAction: 'error'  
  }  
})
```

Listado de campos de la colección

Opcionales



# Esquemas

Mostrar el esquema de una colección:

```
> db.getCollectionInfos({name: "marcas"})
```

Modificar el esquema de una colección:

```
> db.runCommand( {collMod: "marcas",  
  validator: {  
    $jsonSchema: {  
      ...
```

# Esquemas

Si validationAction es “warn”, el resultado se escribe en el log de MongoDB, por defecto en `/var/log/mongodb/mongod.log`

# Esquemas

| Nombre           | Descripción                  | Ejemplo              |
|------------------|------------------------------|----------------------|
| string           | Texto plano                  | "Unai"               |
| Boolean          | Valor booleano               | true                 |
| int              | Número entero (int32)        | 55                   |
| NumberLong       | Número entero grande (int64) | 1000000000           |
| NumberDecimal    | Números decimales            | 12.99                |
| ObjectId         | Identificador único          | ObjectId("74121...") |
| ISODate          | Fecha en formato AAAA-MM-DD  | 2022-01-09           |
| Timestamp        | Fecha en formato Unix        | 11348822             |
| EmbeddedDocument | Documento embebido           | { "a": { ... } }     |
| Array            | Listado de elementos         | { "b": [ ... ] }     |