Instalación y uso básico de MongoDB

Antonio José Méndez Ramírez

1º DAM Punta del verde

Instalación de mongoDB y uso básico.

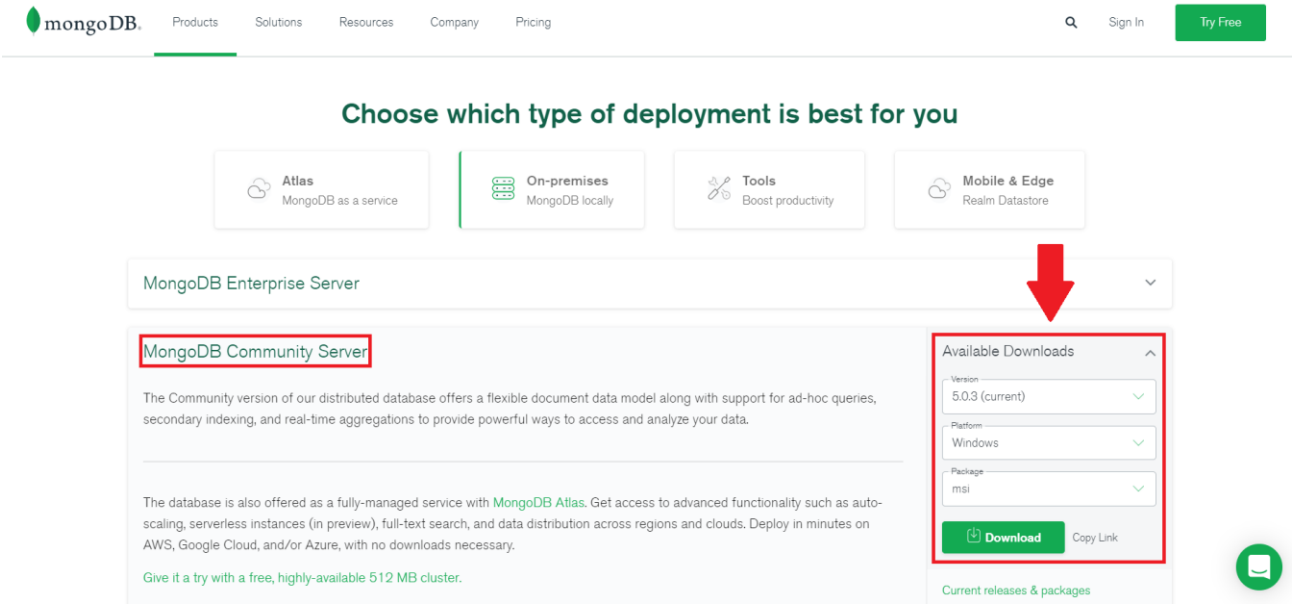
Contenido

1.) ¿Qué es MongoDB?

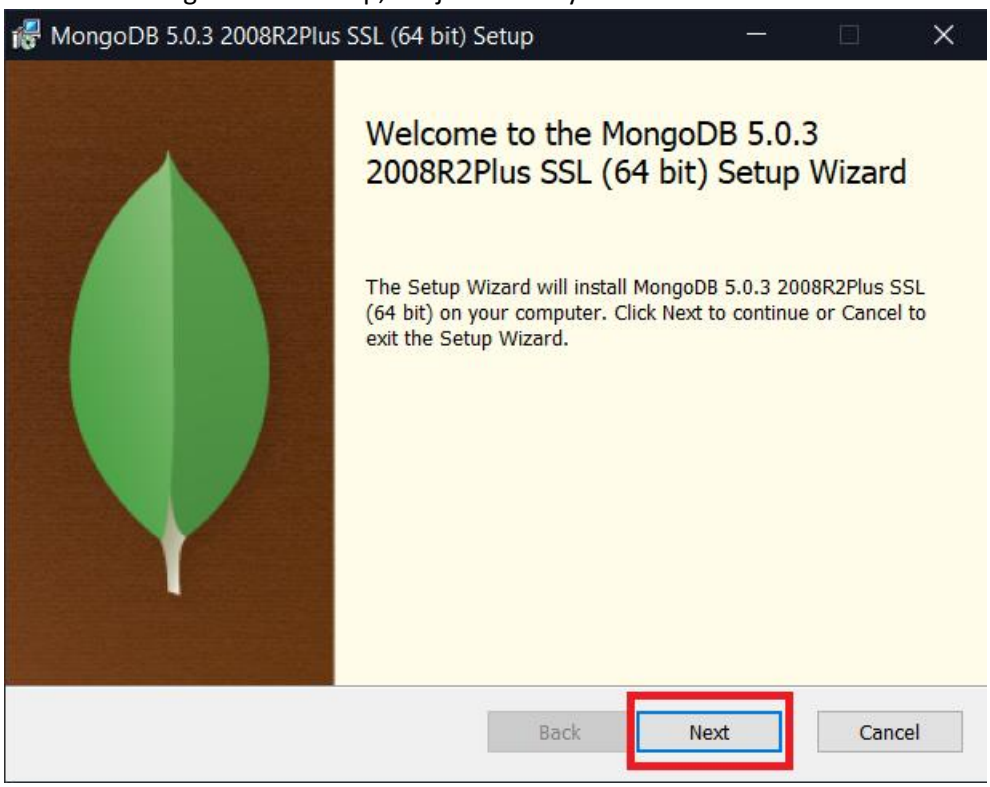
MongoDB es un sistema de base de datos NoSQL orientado a documentos de código abierto y escrito en C++, que en lugar de guardar los datos en tablas lo hace en estructuras de datos JSON con un esquema dinámico.

2.) Instalación de MongoDB.

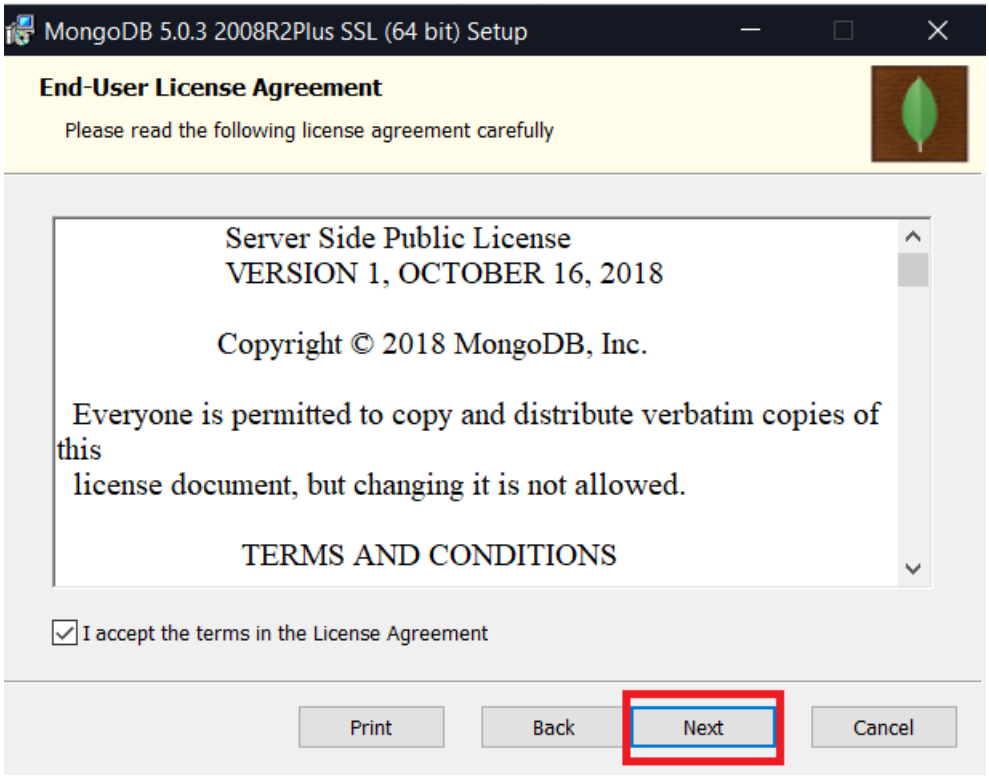
Primero, nos tendremos que ir a la [página oficial de descargas de MongoDB](). Nosotros elegiremos la versión de la comunidad que aunque es más limitada, es gratuita:



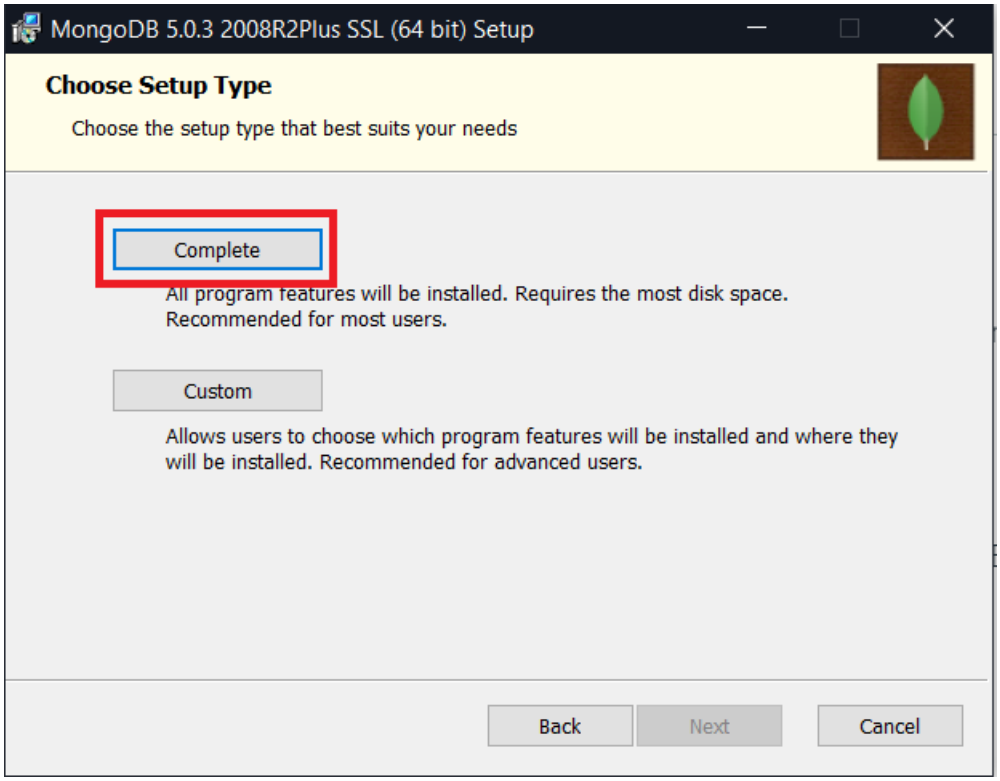
Una vez descarguemos el setup, lo ejecutamos y nos saldrá una ventana así:



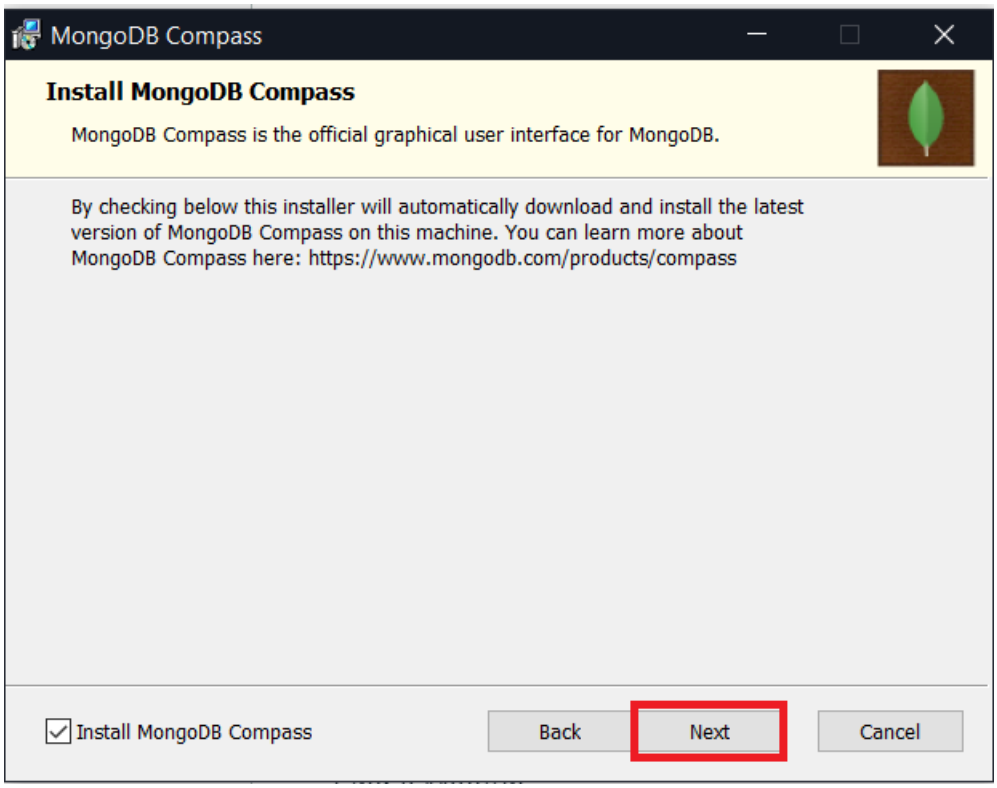
Aceptaremos los términos y condiciones:



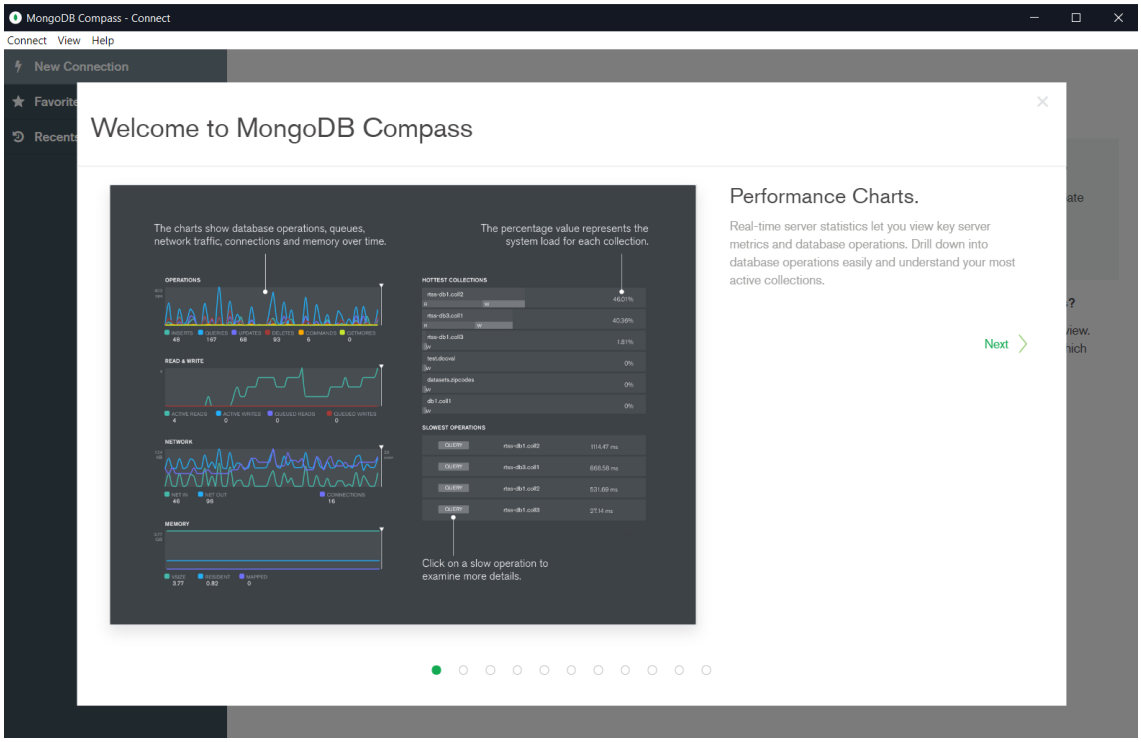
Elegiremos la instalación completa:



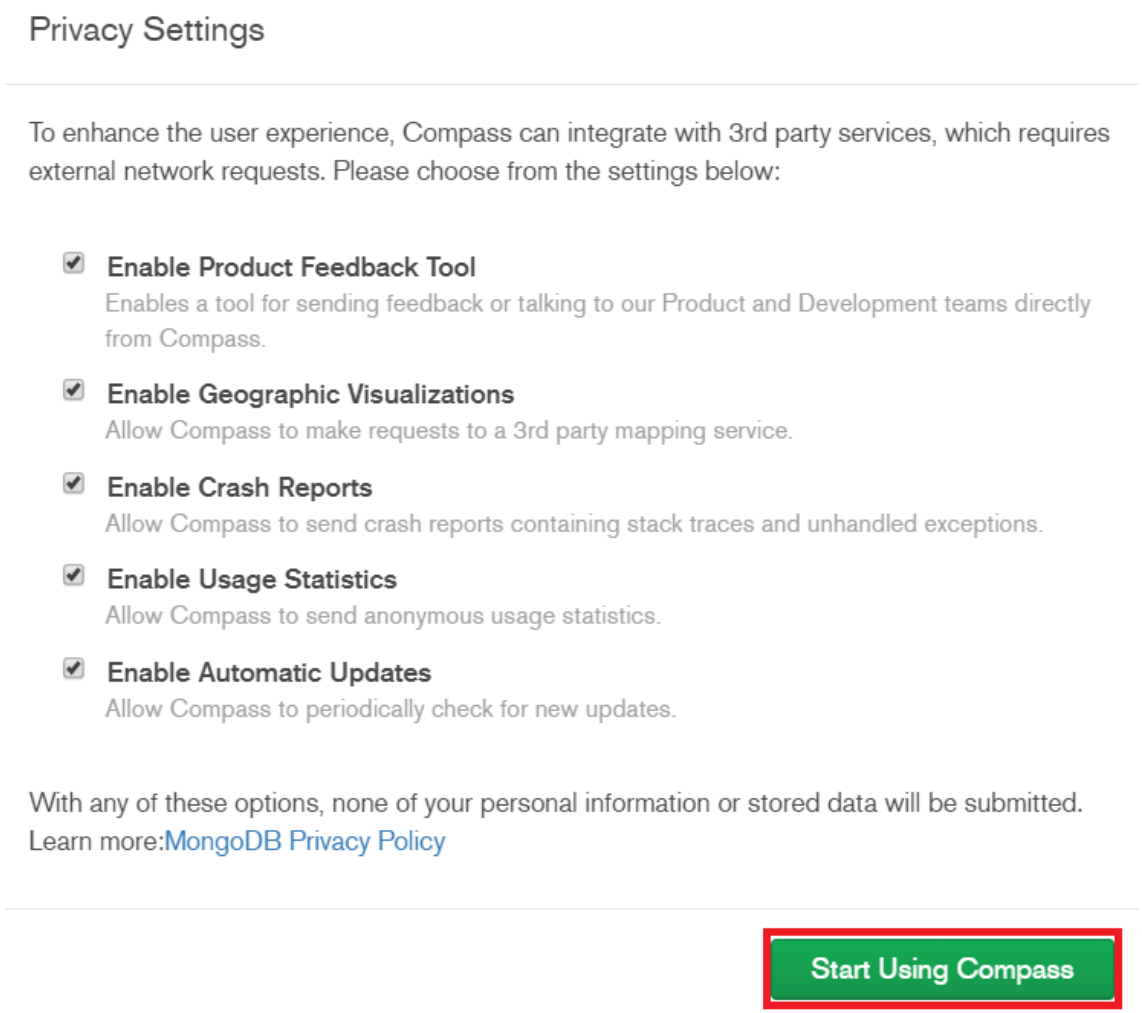
Nos saldrá una ventana para marcar la opción de instalar Mongo Compass; la interfaz gráfica de Mongo: la instalaremos para usarla en un futuro:



Y nos saldrá una ventana con una guía inicial de Mongo:



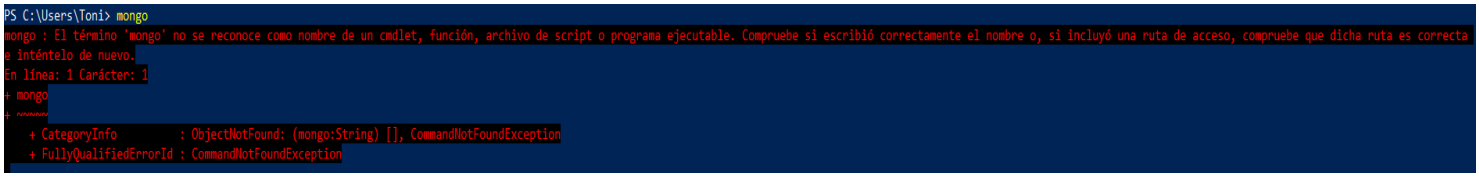
Después de que termine, nos saldrá una ventana para elegir los componentes a instalar del Mongo Compass; nosotros lo dejaremos por defecto, es decir; lo dejaremos todo marcado:



¿Cómo podremos ahora interactuar con Mongo?

Mongo tiene su propia consola, pero nosotros preferiremos usar la propia Powershell de Windows como hicimos en el anterior proyecto.

Para empezar a usar mongo dentro de la powershell hay que iniciarla con el comando "`mongo`".

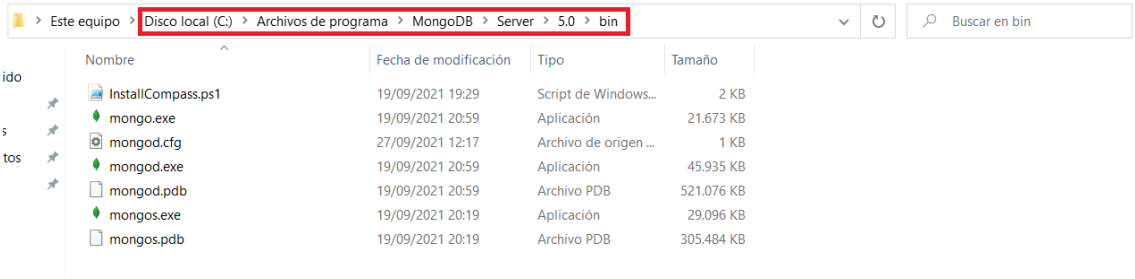


¿Por qué da error?

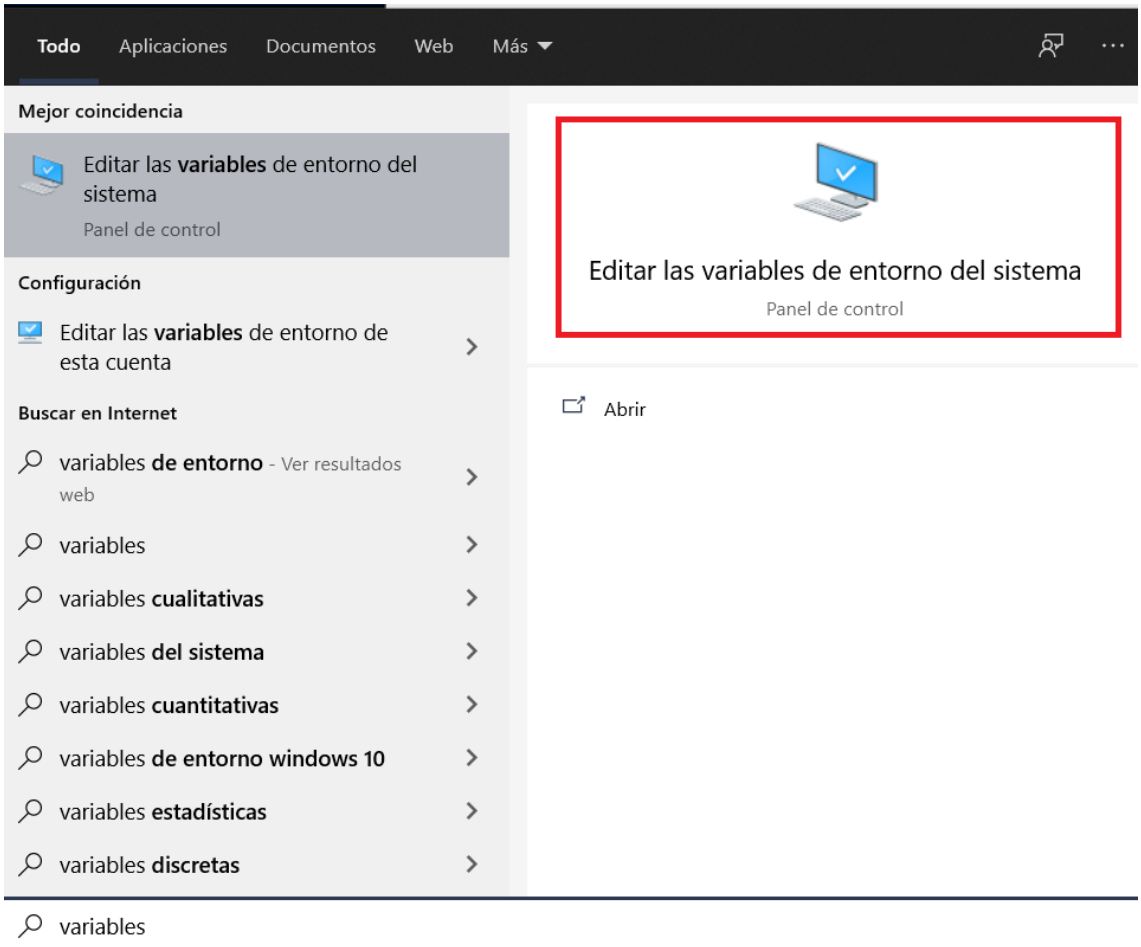
Hemos instalado Mongo, pero en ningún momento le hemos pasado a Powershell la ubicación de los comandos de Mongo, ya que no está agregado en el "Path"

Agregar MongoDB al Path de Windows

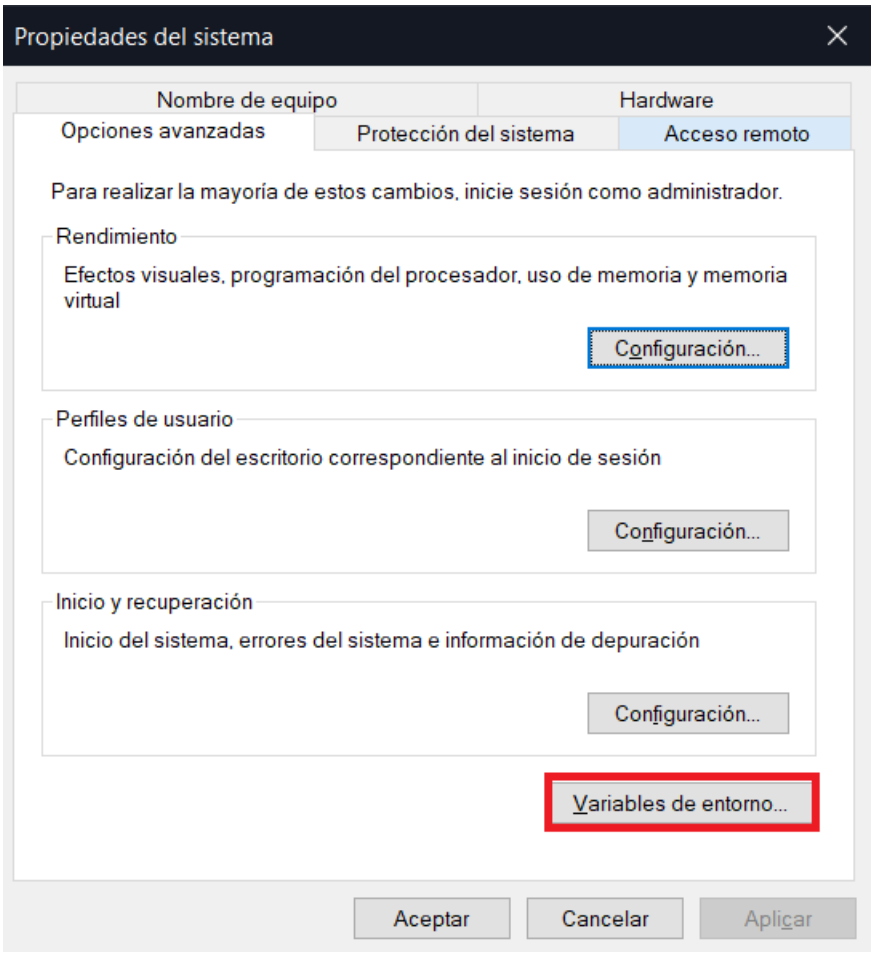
Primero, iremos a la ubicación donde tendremos instalado Mongo y la copiaremos, en mi caso:



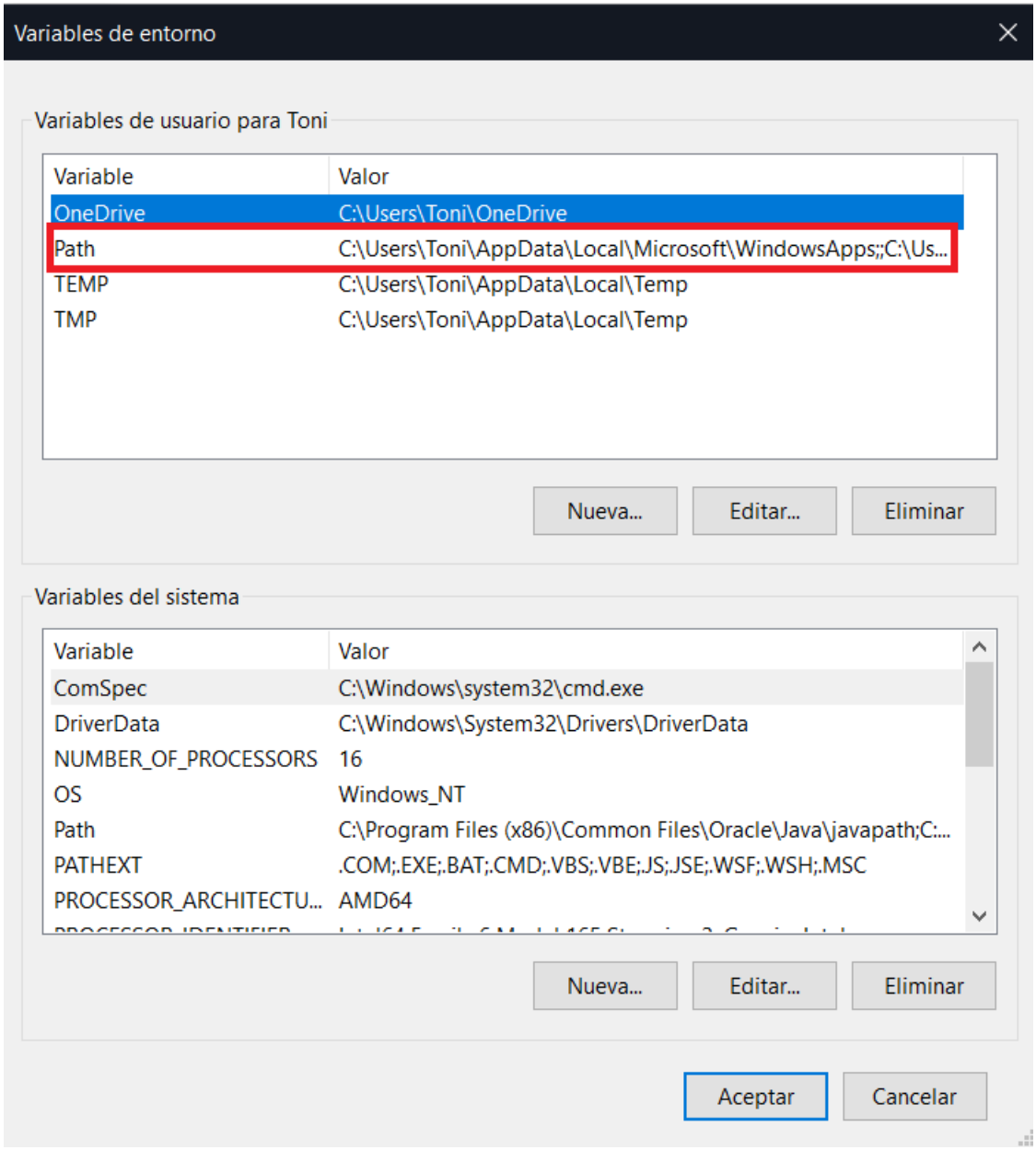
Copiaremos la ruta de nuestro directorio con Mongo instalado, y abriremos la sección de Windows “Editar variables del sistema”:



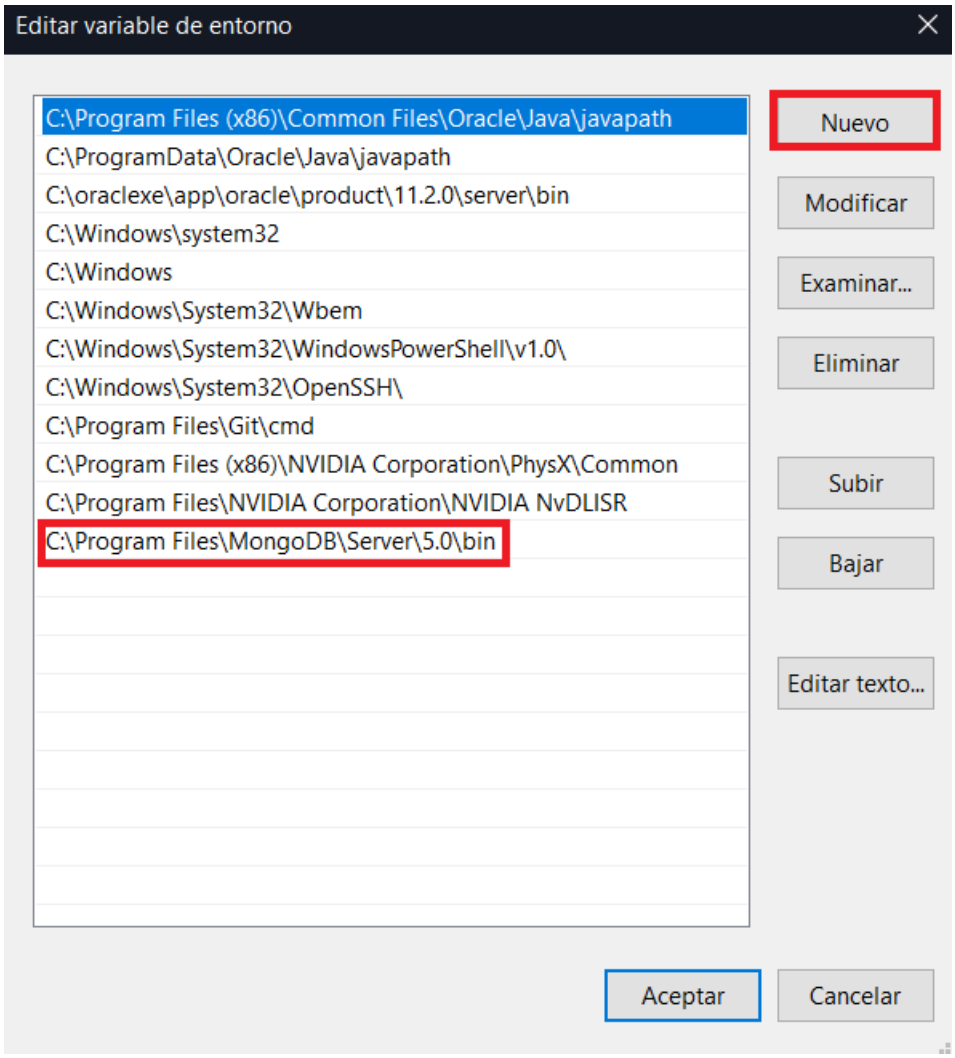
Una vez dentro, seleccionaremos la opción “Variables de entorno”:



Ahora, haremos doble click sobre el apartado “Path”:



Le daremos a “Nuevo” y tendremos que pegar la ruta que copiamos anteriormente:



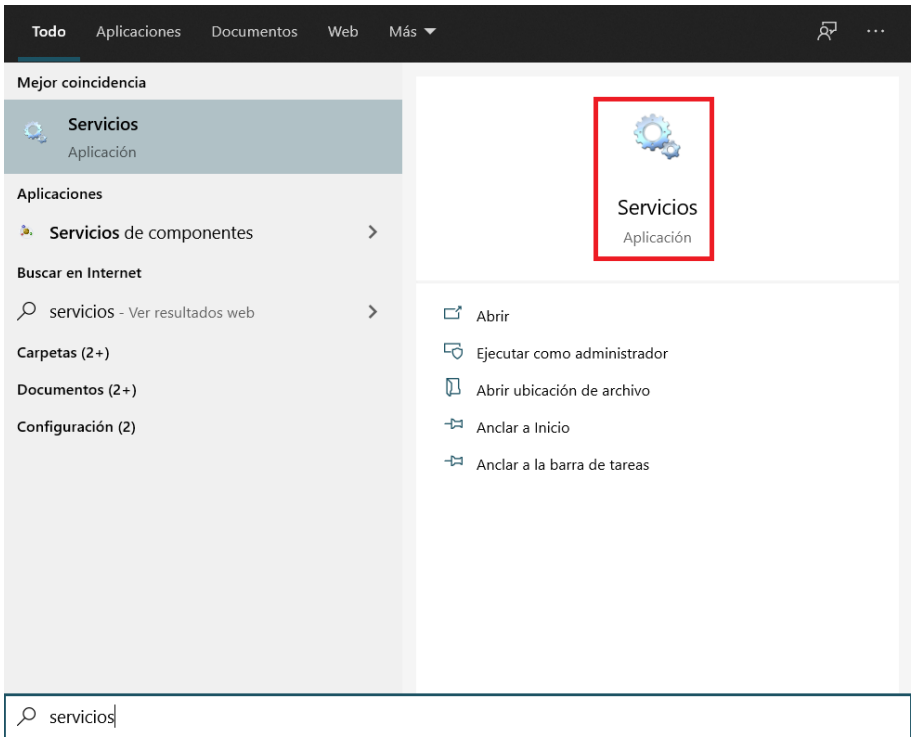
Ahora, si podremos usar Mongo perfectamente en nuestra Powershell desde cualquier directorio sin ningún tipo de problema:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\Toni> mongo
MongoDB shell version v5.0.3
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("04c38459-9488-4c67-99ec-057209f0bbf4") }
MongoDB server version: 5.0.3
================
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will be removed in
an upcoming release.
We recommend you begin using "mongosh".
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
================
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
        https://community.mongodb.com
---
The server generated these startup warnings when booting:
        2021-09-27T12:17:37.667+02:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> exit
bye
PS C:\Users\Toni> mongo --version
MongoDB shell version v5.0.3
Build Info: {
    "version": "5.0.3",
    "gitVersion": "657fea5a61a74d7a79df7aff8e4bcf0bc742b748",
    "modules": [],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "windows",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}
```

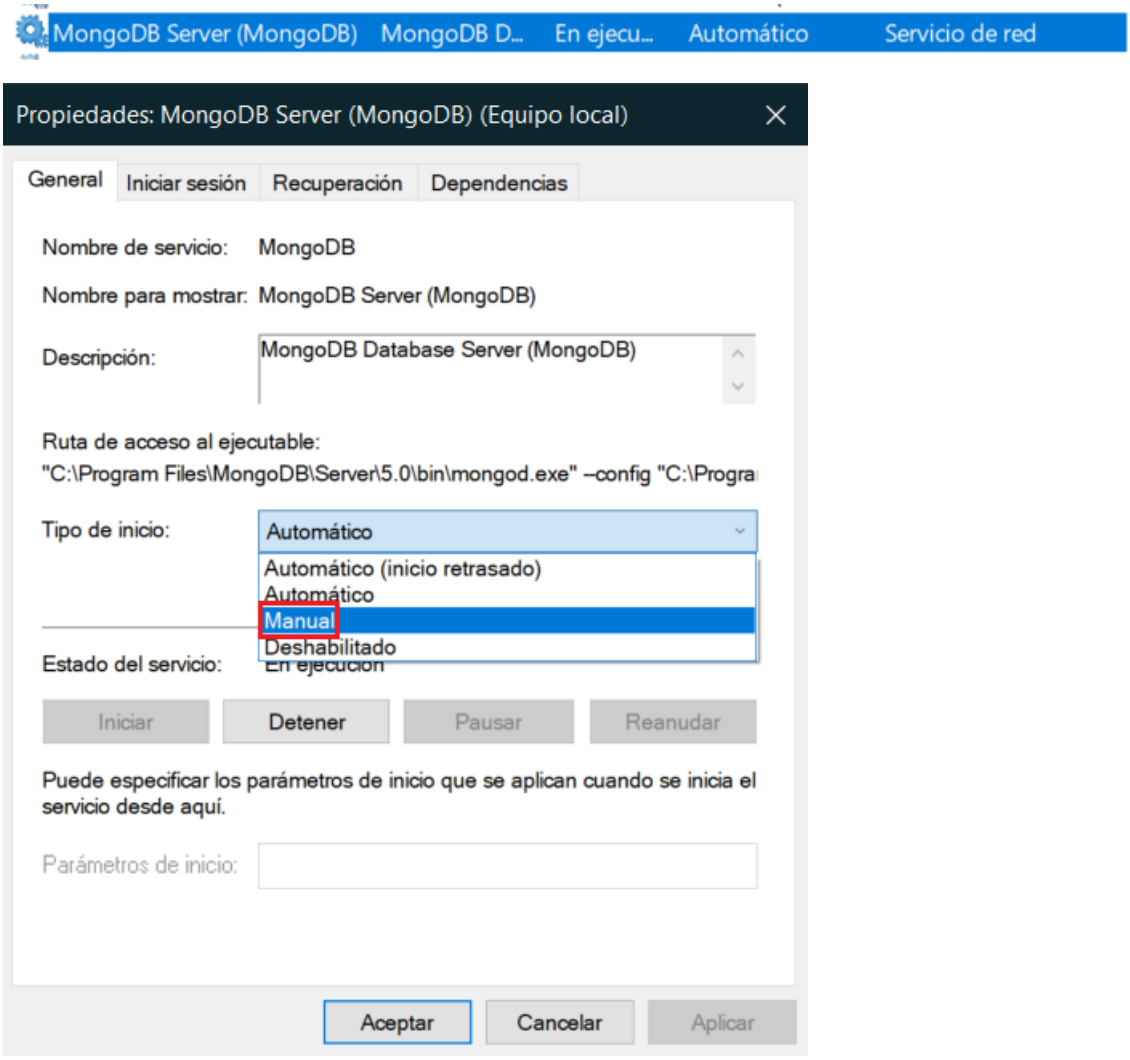
¿Cómo iniciar Mongo manualmente?

Mongo se inicia automáticamente, ¿Pero qué pasa si no queremos que se ejecute cada vez que encendamos el ordenador? ¿Y si se queda colgado cuando se inicia?

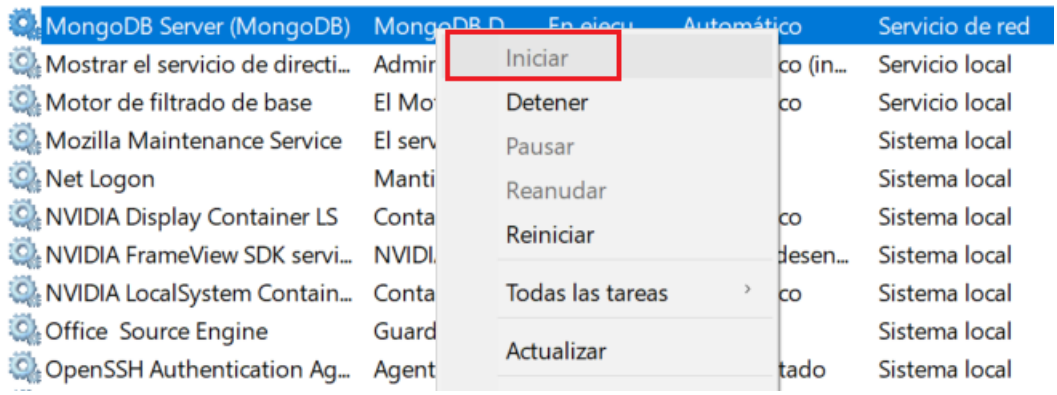
Para esto, tendremos que irnos a “Servicios”, donde Windows nos muestra todos los servicios en segundo plano que se ejecutan:



Una vez dentro, buscaremos el servicio de MongoDB y haremos click derecho sobre él y seleccionaremos sus propiedades:



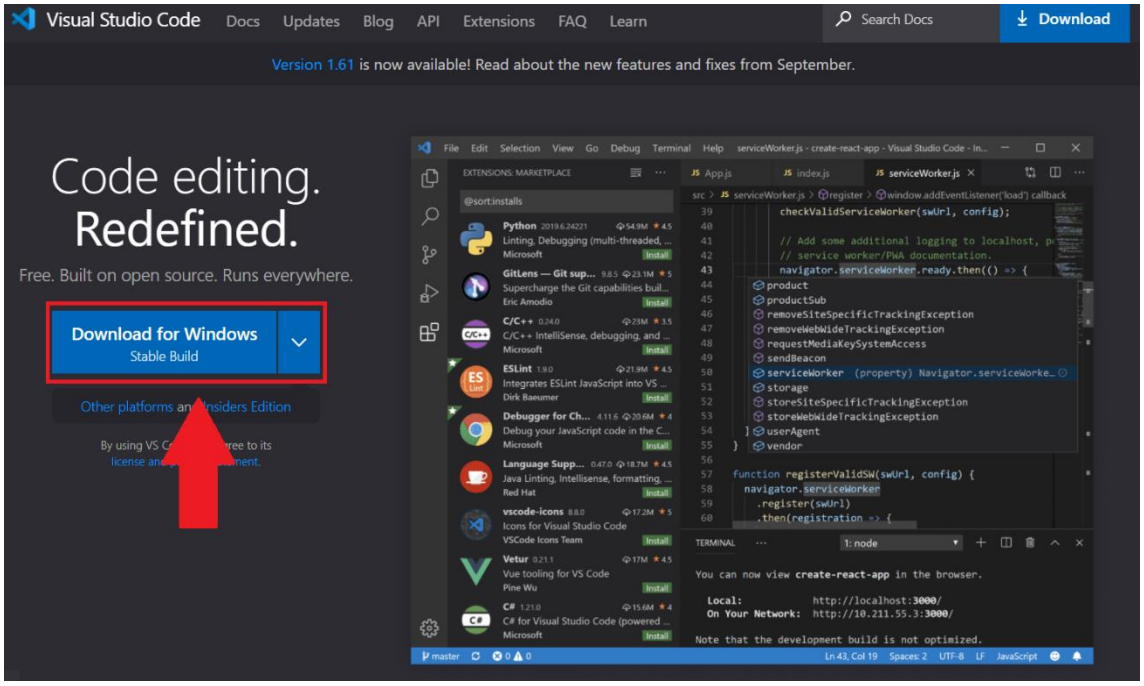
Si lo seleccionamos en manual, o si lo queremos iniciar nosotros manualmente aunque este automático (Ya se haya quedado el proceso colgado u alguna otra razón), tendremos que hacer click derecho sobre el servicio e iniciarlo:



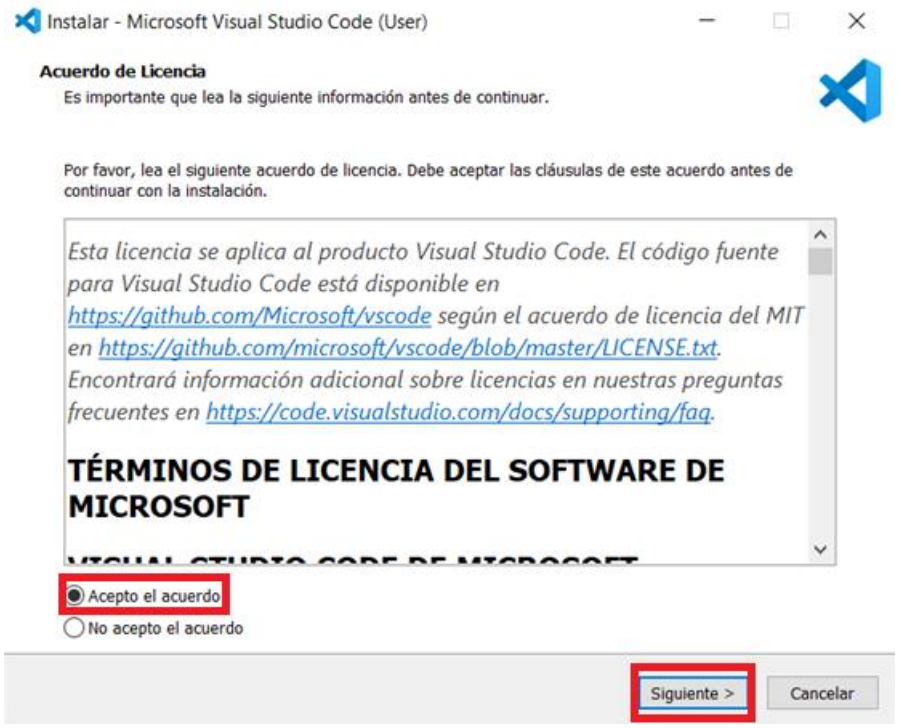
3.) Instalación Visual Studio Code.

Para Mongo, necesitaremos una editor de texto plano, y en nuestro caso elegiremos Visual Studio Code.

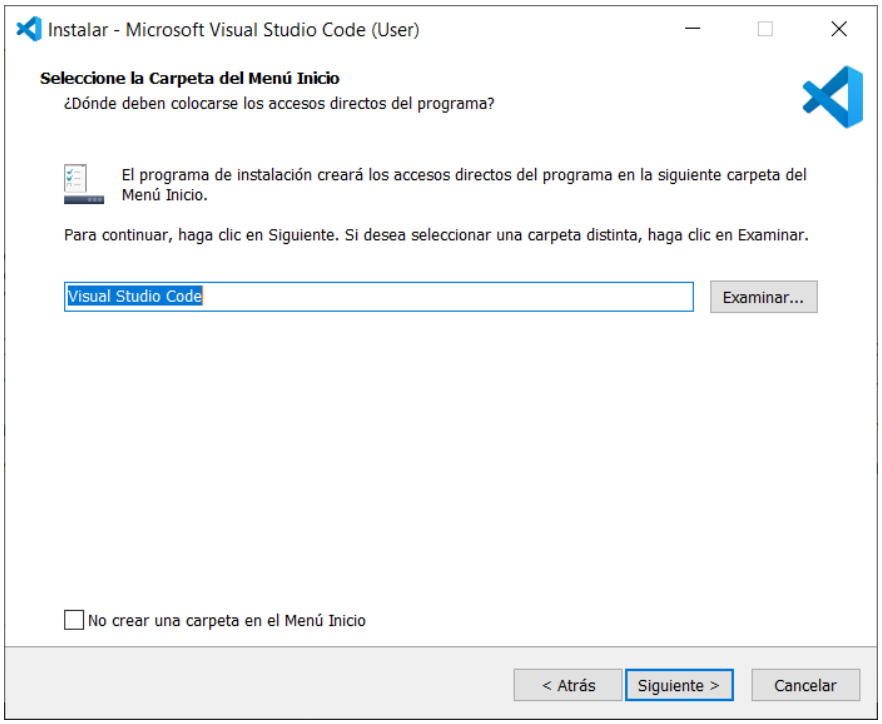
Para descargarlo, iremos al [apartado de descargas de su página oficial](), y le daremos al botón "Download":



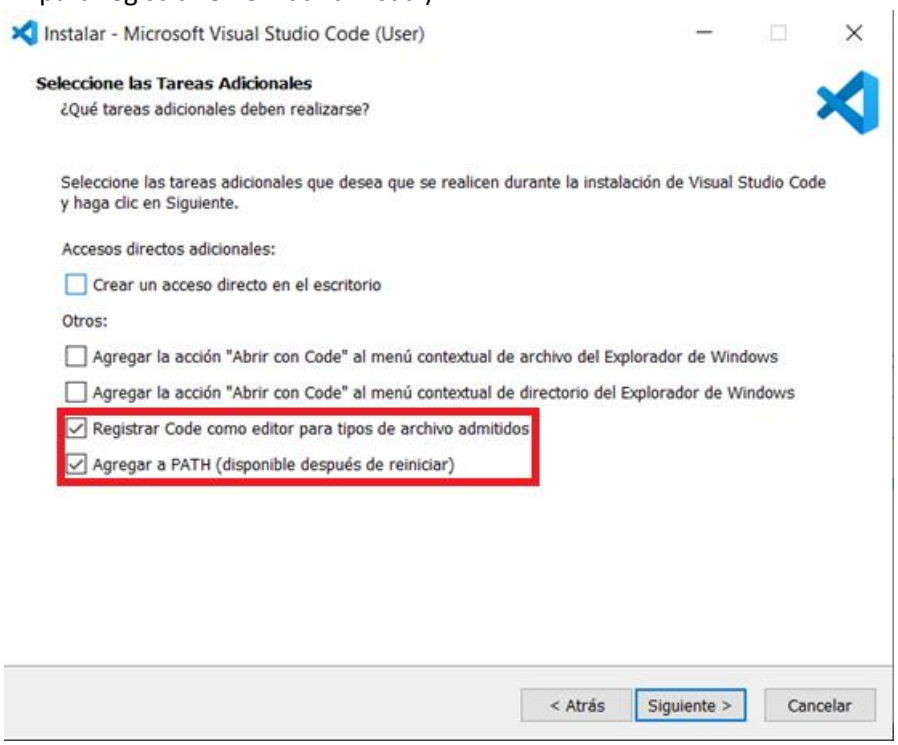
Una vez descargado, ejecutaremos su instalador, y lo 1º que nos saldrá es una ventana pidiéndonos aceptar los términos y condiciones, las aceptaremos y le daremos a siguiente:



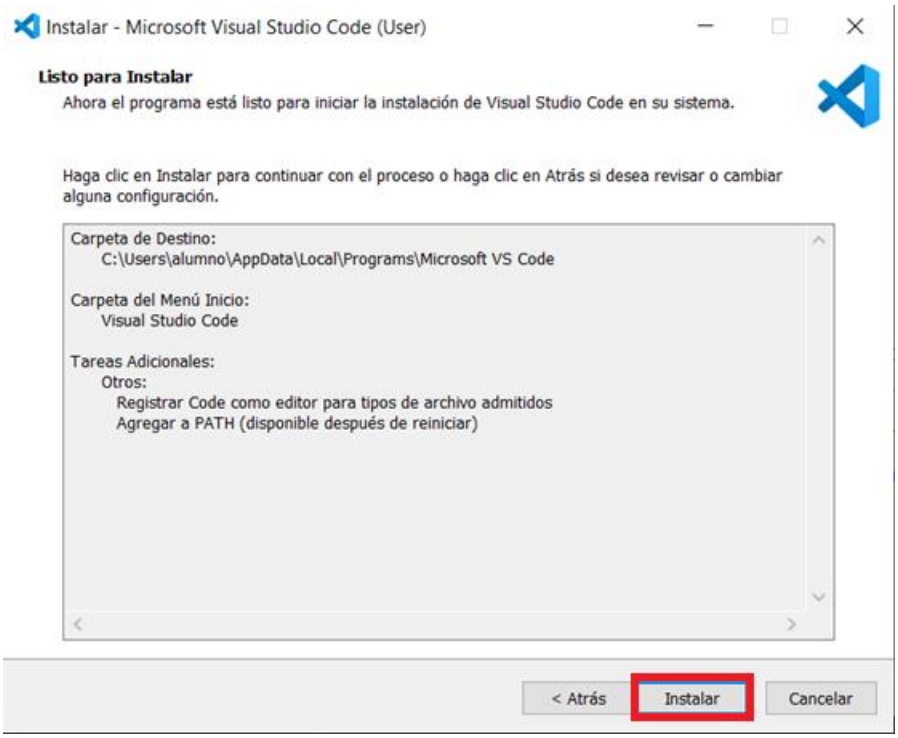
En la siguiente ventana no tendremos que tocar nada, simplemente le daremos a siguiente:



Ahora, nos saldrá una ventana con algunas opciones para configurar, nosotros solo marcaremos las dos últimas (La 1º para dejar por defecto Visual para archivos admitidos, y la 2º para registrar en el Path a Visual):



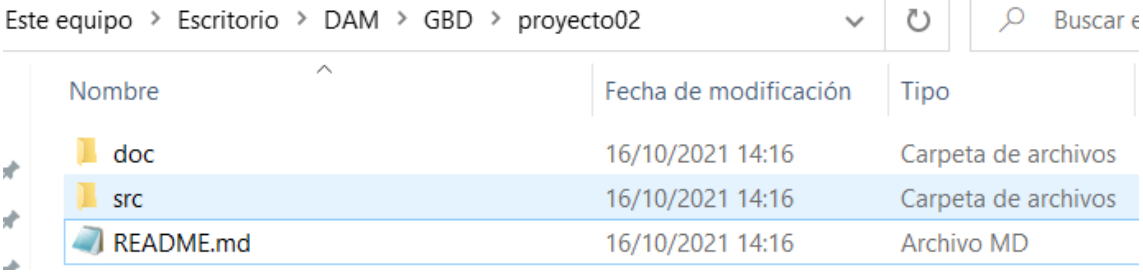
Y, lo instalaremos:



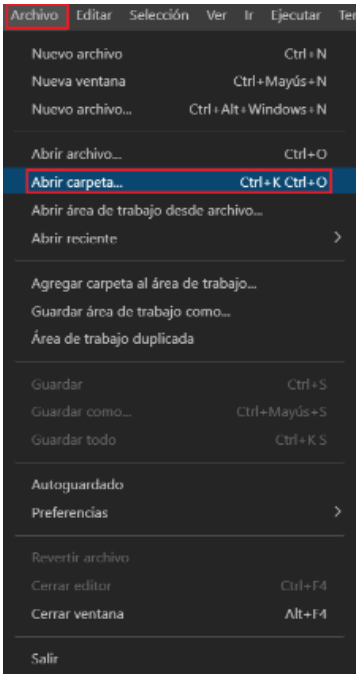
Y ya quedaría totalmente instalado.

Creación de un proyecto en Visual Studio Code.

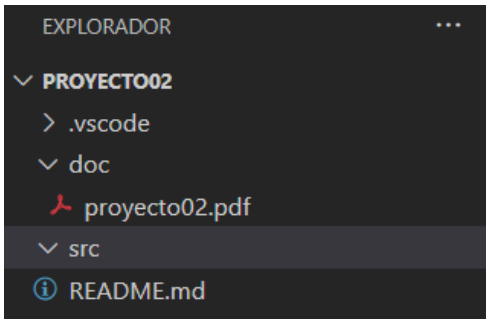
Antes de nada, creare mi estructura de directorios en la ruta que queramos. En mi caso, queda tal que así:



Una vez creado nuestra estructura de directorios preparada para subirse también a Github, abriremos el directorio “proyecto02” en Visual. Para esto, haremos clic en “Archivo” > “Abrir carpeta”:

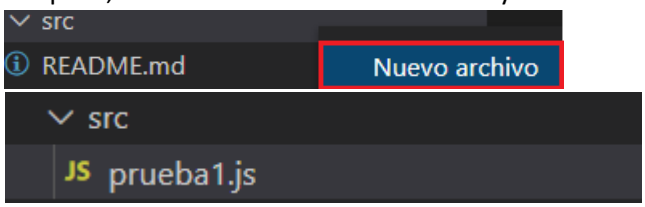


Nos quedara un explorador tal que así, mostrando la estructura que tengamos:

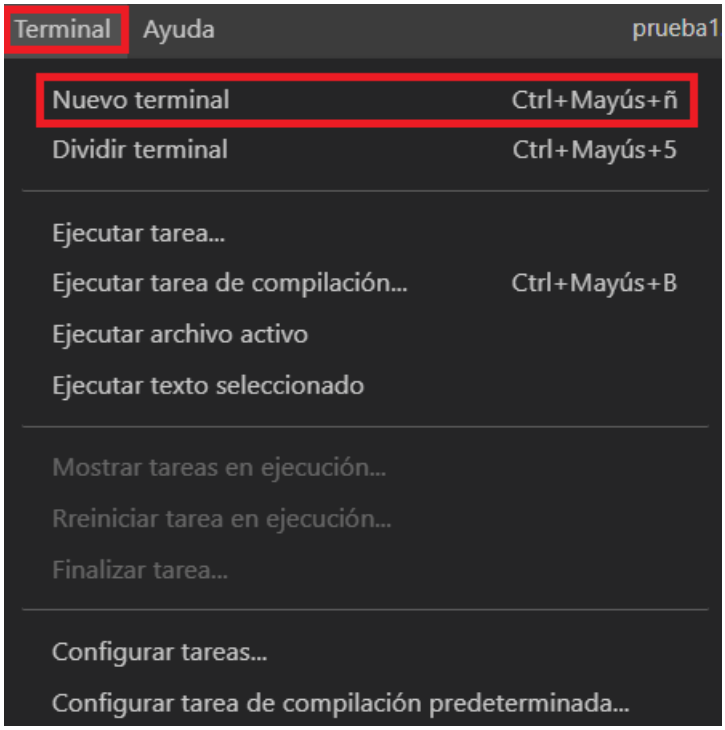


¿Cómo crear un archivo?

Primero seleccionaremos el directorio donde crear el fichero, y haremos click derecho sobre él. Después, le daremos en “Nuevo archivo” y le daremos el nombre y la extensión que queramos:



También, podemos manejar Mongo desde Visual, ya que trae una Shell en el propio Visual, para iniciar una, haremos click en “Terminal” > “Nuevo terminal”:



4.) Comandos básicos con MongoDB.

Todos los comandos básicos, se pueden ver en el apartado manual de la página oficial MongoDB.

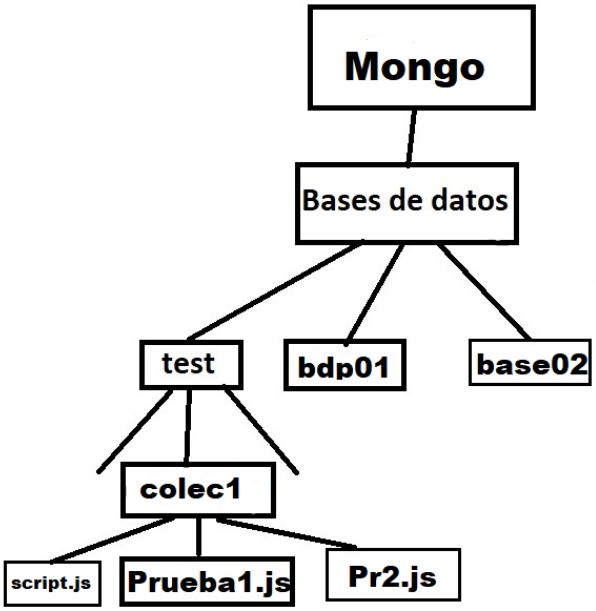
Primero, ejecutaremos la Powershell de Windows, y ejecutaremos el comando "`mongo`" para empezar a usar la Shell de Mongo:

```
PS C:\Users\Toni> mongo
```

Con el comando "test", podremos ver la base de datos con la que estamos trabajando; y en caso de que la base de datos sea "test", no estaremos trabajando sobre ninguna base de datos real:

```
> db
test
```

El gestor de base de datos tiene base de datos, como test, bdp01, etc… las bases de datos son como "directorios". Dentro de los directorios tenemos colecciones (como si fueran hojas de cálculo). Cada colección tiene documentos json, que serían como las filas de la hoja de cálculo:



Con el comando "`use <database>`" empezaremos a usar una base de datos. En caso de no estar creada, la creará:

```
> use bdp01
switched to db bdp01
> db
bdp01
```

Realmente, la base de datos no se creará hasta que introduzcamos información dentro de ella, así que ahora usaremos el comando "`db.<NombreColeccion>.insertOne ( { x:1 } )`" insertamos información en la colección nombrada como queramos dentro de la base de datos que estamos usando actualmente, donde x es el tipo de dato, y 1 el propio dato (Podremos pasarle más de 1 tipo de dato con su dato, pero siempre tendrán que ir separados por comas):

```
> db.col01.insertOne( { nombre: "Juan", edad: 21 } )
{
        "acknowledged" : true,
        "insertedId" : ObjectId("616afdee9f2285b1a484e25a")
}
```

Gracias al “ObjectID” que mongo asigna automáticamente a cada objeto que introducimos, podemos poner ese mismo objeto, que Mongo siempre lo va a diferenciar por su ID:

```
> db.col01.insertOne( { nombre: "Juan", edad: 21 } )
{
        "acknowledged" : true,
        "insertedId" : ObjectId("616afe929f2285b1a484e25b")
}
```

Podemos ver, que son ID diferentes.

IMPORTANTE: Todo lo que sea una cadena de texto dentro de un campo de valor tiene que ir encerrado entre comillas.

Con el comando "`db.NombreColeccion.find()`" consultamos los datos que están dentro de una colección:

```
> db.col01.find()
{ "_id" : ObjectId("616b04519f2285b1a484e25d"), "nombre" : "Juan", "edad" : 21 }
{ "_id" : ObjectId("616b04539f2285b1a484e25e"), "nombre" : "Juan", "edad" : 21 }
```

El comando "`db.NombreColeccion.deleteMany`" elimina datos de la colección. Si dejamos el campo “({})”, se eliminaran TODOS los datos de la colección:

```
> db.col01.deleteMany({})
{ "acknowledged" : true, "deletedCount" : 2 }
```

Además, nos indicara el nº de datos que ha eliminado.

Con el comando “`db.dropDatabase()`”, eliminaremos la base de datos que estemos usando en ese momento.

Con el comando “`db.<nombreColeccion>.drop()`” eliminaremos la base de datos que estemos usando en ese momento.

Con el comando "`show dbs`" muestra todas nuestras bases de datos:

```
> show dbs
admin   0.000GB
bdp01   0.000GB
config  0.000GB
local   0.000GB
test    0.000GB
```

Y, con el comando “`show collections`” muestra todas las colecciones de la base de datos que estemos usando en ese momento:

```
> show collections
col01
```

En Mongo, en vez de usar los comandos uno a uno en la Shell, podemos usar ficheros .json, el cual contenga estos mismos comandos y puedas poner la cantidad de comandos que tú quieras. Para esto, usaremos el fichero que crearemos:

JS insert01.js

Vamos a probar a insertar varios parámetros en el mismo script con "`insertOne`":

```
src > JS prueba1.js > ...
1
2   db.col01.insertOne( { nombre: "Pepe", edad: 28 } )
3   db.col01.insertOne( { nombre: "Alfredo", edad: 25 } )
4   db.col01.insertOne( { nombre: "Antonio", edad: 19 } )
5   db.col01.insertOne( { nombre: "Miguel", edad: 21 } )
6   db.col01.insertOne( { nombre: "Francisco", edad: 18 } )
7   db.col01.insertOne( { nombre: "Peter", edad: 17 } )
8   db.col01.insertOne( { nombre: "Raul", edad: 18 } )
```

Y lo ejecutaremos desde la propia terminal del Visual, con el comando "`load("<NombreFichero>")`" :

```
> load("prueba1.js")
true
```

Si hacemos un `Find`, podemos ver que se han agregado correctamente:

```
> db.col01.find()
{ "_id" : ObjectId("616b16dd1e584defa49644a6"), "nombre" : "Pepe", "edad" : 28 }
{ "_id" : ObjectId("616b16dd1e584defa49644a7"), "nombre" : "Alfredo", "edad" : 25 }
{ "_id" : ObjectId("616b16dd1e584defa49644a8"), "nombre" : "Antonio", "edad" : 19 }
{ "_id" : ObjectId("616b16dd1e584defa49644a9"), "nombre" : "Miguel", "edad" : 21 }
{ "_id" : ObjectId("616b16dd1e584defa49644aa"), "nombre" : "Francisco", "edad" : 18 }
{ "_id" : ObjectId("616b16dd1e584defa49644ab"), "nombre" : "Peter", "edad" : 17 }
{ "_id" : ObjectId("616b16dd1e584defa49644ac"), "nombre" : "Raul", "edad" : 18 }
```

También podemos buscar mediante filtros con el comando "`find`" (Metiendo datos entre {}):

```
> db.col01.find({edad:18})
{ "_id" : ObjectId("616c0ffc3142438993310b4b"), "nombre" : "Francisco", "edad" : 18 }
{ "_id" : ObjectId("616c0ffc3142438993310b4d"), "nombre" : "Raul", "edad" : 18 }
```

Insertar documentos en MongoDB.

Para esto, seguiremos usando el documento llamado "inser01.js", dentro de la carpeta src:

JS insert01.js

Y, vamos a crear un objeto, con varios campos de información, "Quantity", "Tag" y "Size", y dentro de "size" habrá diferentes parámetros también:

```
db.col01.insertOne(
    { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }
)
```

Y lo cargaremos en la Shell:

```
> load("insert01.js")
true
> db.col01.find()
{ "_id" : ObjectId("616c180d3142438993310b51"), "item" : "canvas", "qty" : 100, "tags" : [ "cotton" ], "size" : { "h" : 28, "w" : 35.5, "uom" : "cm" } }
```

Para insertar diferentes objetos en el mismo fichero y usando el mismo comando, usaremos el comando "`db.inventory.insertMany([])`", que lo que hará es insertar un **ARRAY** de **documentos:**

```
db.col01.insertMany([
    { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },
    { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
    { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm" } }
])
```

Y lo cargaremos en la Shell:

```
> load("insert01.js")
true
> db.col01.find()
{ "_id" : ObjectId("616c19ce3142438993310b55"), "item" : "journal", "qty" : 25, "tags" : [ "blank", "red" ], "size" : { "h" : 14, "w" : 21, "uom" : "cm" } }
{ "_id" : ObjectId("616c19ce3142438993310b56"), "item" : "mat", "qty" : 85, "tags" : [ "gray" ], "size" : { "h" : 27.9, "w" : 35.5, "uom" : "cm" } }
{ "_id" : ObjectId("616c19ce3142438993310b57"), "item" : "mousepad", "qty" : 25, "tags" : [ "gel", "blue" ], "size" : { "h" : 19, "w" : 22.85, "uom" : "cm" } }
>
```

Estas son las dos formas que tenemos de insertar documentos con **ARRAYS**.

Documentos QUERY en MongoDB.

Para esto, seguiremos usando el documento llamado "insert01.js":

JS insert01.js

Para esto, introduciremos una serie de documentos mediante un "`insertMany`", con distintos objetos, con diferentes parámetros al igual que antes, pero ahora insertando un campo llamado "status", que nos permitirá identificar mejor a un conjunto de objetos:

```
db.col01.insertMany([
    { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
    { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
    { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
    { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
    { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

Lo cargaremos en la Shell:

```
> load("insert01.js"))
true
> db.col01.find()
{ "_id" : ObjectId("616c28ea3142438993310b58"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("616c28ea3142438993310b59"), "item" : "notebook", "qty" : 50, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "A" }
{ "_id" : ObjectId("616c28ea3142438993310b5a"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
{ "_id" : ObjectId("616c28ea3142438993310b5b"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, "w" : 30, "uom" : "cm" }, "status" : "D" }
{ "_id" : ObjectId("616c28ea3142438993310b5c"), "item" : "postcard", "qty" : 45, "size" : { "h" : 10, "w" : 15.25, "uom" : "cm" }, "status" : "A" }
```

Y con estos datos, veremos el siguiente punto:

Consultas en MongoDB.

Para esto usaremos operadores query, operadores propios de mongo muy útiles a la hora de hacer consultas. Todos estos pueden verse en la sección de su manual "Query and Projection operators".

Crearemos el siguiente fichero .js para realizar consultas:

consulta01.js

¿Qué es una consulta? Una consulta es el método para acceder a los datos que nosotros deseemos buscar.

Si queremos buscar los objetos que solo tengan un campo en específico, tendremos que escribirlo de esta manera:

```
Buscamos solo los que tienen el valor "paper" en el campo item
db.col01.find( {item: "paper"} )
```

```
> db.col01.find( {item: "paper"} )
{ "_id" : ObjectId("616c28ea3142438993310b5a"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
```

Para esto, también podemos usar el operador "$eq", que es lo mismo que "igual":

```
//hacer la misma consulta usando el operador $eq
db.col01.find( {item: { $eq: "paper" } } )
```

```
> db.col01.find( {item: { $eq: "paper" } } )
{ "_id" : ObjectId("616c28ea3142438993310b5a"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
```

Si lo que hay dentro del campo, es un valor numérico se escribiría de esta manera:

```
// Saber los que tienen 25 en el campo qty (quantity)
db.col01.find( { qty: { $eq: 25 } } )
```

```
> db.col01.find( { qty: { $eq: 25 } } )
{ "_id" : ObjectId("616c88bfd4fbcf9373683789"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
```

Si queremos saber el nº de veces que se repite un valor dentro de un campo, usaremos el "`.count`":

```
//Cuenta el nº de valores iguales a "planner" en el campo item
db.col01.find({item: { $eq: "planner" } }).count()
```

```
> db.col01.find({item: { $eq: "planner" } }).count()
1
```

Dentro de los campos puede haber otros campos, "subcampos". Para buscar un valor dentro de estos, se escribiría de esta manera:

```
// Saber los que tienen el valor "in" en el subcampo uom, que pertenece al campo size.
db.col01.find( {"size.uom": "in" })
```

o usando el operador "$eq":

```
// Saber el subacampo uom que cumplen que son iguales a "in", que pertenece al campo size.
db.col01.find( {"size.uom": { $eq: "in"} })
```

```
> db.col01.find( {"size.uom": "in" })
{ "_id" : ObjectId("616c88bfd4fbcf937368378a"), "item" : "notebook", "qty" : 50, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "A" }
{ "_id" : ObjectId("616c88bfd4fbcf937368378b"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
> db.col01.find( {"size.uom": { $eq: "in"} })
{ "_id" : ObjectId("616c88bfd4fbcf937368378a"), "item" : "notebook", "qty" : 50, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "A" }
{ "_id" : ObjectId("616c88bfd4fbcf937368378b"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
```

Otros operadores query.

Tenemos el comparador "`$gt`", que significa "mayor que":

```
> db.col01.find( { qty: { $gt: 50 } } )
{ "_id" : ObjectId("616c88bfd4fbcf937368378b"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
{ "_id" : ObjectId("616c88bfd4fbcf937368378c"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, "w" : 30, "uom" : "cm" }, "status" : "D" }
```

En esta misma línea de código, podemos ver que hemos buscado objetos que tengan más de valor 50 en el campo qty.

Tenemos el comparador "`$gte`", que significa "mayor o igual que:

```
> db.col01.find( { qty: { $gte: 45 } } )
{ "_id" : ObjectId("616c88bfd4fbcf937368378a"), "item" : "notebook", "qty" : 50, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "A" }
{ "_id" : ObjectId("616c88bfd4fbcf937368378b"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
{ "_id" : ObjectId("616c88bfd4fbcf937368378c"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, "w" : 30, "uom" : "cm" }, "status" : "D" }
{ "_id" : ObjectId("616c88bfd4fbcf937368378d"), "item" : "postcard", "qty" : 45, "size" : { "h" : 10, "w" : 15.25, "uom" : "cm" }, "status" : "A" }
```

En esta misma línea de código, podemos ver que hemos buscado objetos que tengan más o igual de valor 45 en el campo qty.

Tenemos el comparador "`$lt`", que signficica "menor que", y el comparador "`$lte`", que significa "igual o menor que":

```
> db.col01.find( { qty: { $lte: 50 } } )
{ "_id" : ObjectId("616c88bfd4fbcf9373683789"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("616c88bfd4fbcf937368378a"), "item" : "notebook", "qty" : 50, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "A" }
{ "_id" : ObjectId("616c88bfd4fbcf937368378d"), "item" : "postcard", "qty" : 45, "size" : { "h" : 10, "w" : 15.25, "uom" : "cm" }, "status" : "A" }
```

Tenemos el comparador "`$ne`" que significa "diferente que":

```
> db.col01.find( { qty: { $ne: 50 } } )
{ "_id" : ObjectId("616c88bfd4fbcf9373683789"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("616c88bfd4fbcf937368378b"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
{ "_id" : ObjectId("616c88bfd4fbcf937368378c"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, "w" : 30, "uom" : "cm" }, "status" : "D" }
{ "_id" : ObjectId("616c88bfd4fbcf937368378d"), "item" : "postcard", "qty" : 45, "size" : { "h" : 10, "w" : 15.25, "uom" : "cm" }, "status" : "A" }
```

En esta misma línea de código, podemos ver que hemos buscado objetos que sean diferentes que 50 en el campo qty.

Tenemos el operador lógico "`and`" que significa "y…":

```
> db.col01.find( { $and: [ { qty: { $eq: 25} }, { status: { $eq: "A"} } ] } )
{ "_id" : ObjectId("616c88bfd4fbcf9373683789"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
```

En esta misma línea de código, podemos ver que con el operador "`$and`" estamos pidiendo que se cumplan 2 condiciones: que el campo "qty" contenga el valor 25 y que el campo "status" contenga el valor "A".

Tenemos el operador lógico "`$or`" que significa "o…":

```
> db.col01.find( { $or: [ { qty: { $eq: 25} }, { status: { $eq: "A"} } ] } )
{ "_id" : ObjectId("616c88bfd4fbcf9373683789"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("616c88bfd4fbcf937368378a"), "item" : "notebook", "qty" : 50, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "A" }
{ "_id" : ObjectId("616c88bfd4fbcf937368378d"), "item" : "postcard", "qty" : 45, "size" : { "h" : 10, "w" : 15.25, "uom" : "cm" }, "status" : "A" }
```

En esta misma línea de código, podemos ver que con el operador "`$or`" estamos pidiendo que se cumpla 1 de las 2 condiciones: que el campo "qty" contenga el valor 25 o que el campo "status" contenga el valor "A".

Actualizar documentos

¿Cómo puedo hacer para añadir más valores a mis documentos? ¿Y si me he equivocado, como puedo modificarlo?

Podemos solucionar estos problemas con el comando "`db.col01.update()`". Hay 2 maneras de hacerlo:

- Solo con el comando update: Con esto, lo que realmente estamos haciendo es sobrescribiendo sobre el objeto:

```
> db.col01.update({"item": "postcard"}, {"item": "pencil"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.col01.find({item:"pencil"})
{ "_id" : ObjectId("616c88bfd4fbcf937368378d"), "item" : "pencil" }
```

  Como podemos ver, lo que ha hecho es borrar todos los datos del item "postcard" y escribir los suyos encima.

- Con el operador "`$set`": Con este operador, lo que realmente haremos es añadir más campos al objeto en cuestión:

```
> db.col01.update({"item": "pencil"}, { $set: {"qty": 50} })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.col01.find({item:"pencil"})
{ "_id" : ObjectId("616c88bfd4fbcf937368378d"), "item" : "pencil", "qty" : 50 }
```

  Como podemos ver, al contrario que al anterior no ha sobreescribido los datos, si no que ha **AÑADIDO** datos.