

# MongoDB

# ¿Qué es MongoDB?

Una base de datos NoSQL...

- Sin esquema
- Alto rendimiento
- Almacena documentos BSON
- Enfocada en escalabilidad horizontal
- Lenguaje de consultas potente
- Sin transacciones
- Agregado de datos

# ¿Qué es MongoDB?

Una base de datos de **documentos**

- No impone forma a los datos
- No necesita migraciones/reestructuración de la BBDD
- Permite estructuras muy complejas
- Herramientas potentes de agregado con JavaScript
  - Map-Reduce
  - Aggregation Pipeline

# ¿Qué es MongoDB?

Con algunos extras interesantes...

- Índices geoespaciales
- Búsquedas FTS
- Almacenamiento eficiente de blobs y ficheros
- Sharding automático
- Replicación

# ¿Qué es MongoDB?

Es una buena alternativa para... ¡muchas cosas!

- Prototipos y aplicaciones simples
- Hacer la transición de front a back
- Aplicaciones con mucha carga de escritura
- Agregado de datos a un nivel medio/alto
- Aplicaciones con datos muy heterogéneos
- Enormes colecciones de datos (sharding)
- Almacenar ficheros (sharding)

# ¿Qué NO es MongoDB?

No te dejes seducir demasiado:

- **Mongo no puede hacer JOINS!**
- El lenguaje de consulta menos potente que SQL
- **No tiene transacciones!**
- La velocidad baja al subir la seguridad (escritura)
- Ten cuidado:
  - Es muy fácil empezar con MongoDB
  - Si tu app crece mucho... vas a necesitar JOINS

# La idea general

Un almacén de documentos

- Básicamente, objetos JSON (BSON)
- Dividido en colecciones
- Consultas basadas en la estructura del documento
- ¡Se integra genial con JavaScript!

# Requisitos

## Necesitas

- Instalar y arrancar mongod
  - `mongod --dbpath <path> --nojournal`
- Tener la documentación a mano
  - ➔ <http://docs.mongodb.org/manual/>
  - ➔ <http://mongodb.github.io/node-mongodb-native/>
- `npm install mongodb`
- En c9.io
  - <comandos para instalar mongodb>



# Primeros pasos

Para conectarte al servidor

```
var Promise = require("bluebird"),
    MongoDB = Promise.promisifyAll(require("mongodb")),
    MongoClient = MongoDB.MongoClient,
    ObjectID = require("mongodb").ObjectID;

var url = "mongodb://127.0.0.1:27017/dbname"
var client = MongoClient.connectAsync(url);
client.then(function(db) {
    console.log("Conectado!");
});
client.fail(function(e) {
    console.log("ERROR conectando a Mongo: ", e)
});
```

# Primeros pasos

La BD está dividida en colecciones. Para abrir una colección:

```
var collection = client.then(function(db) {  
    return db.collection("coleccion");  
});
```

# Documento

Un documento es un objeto BSON

```
{  
  "_id" : ObjectId("524872a99c50880000000001"),  
  "email" : "test@asdf.com",  
  "password" : "asdf1234",  
  "name" : "Test User",  
  "date" : 1380479657300,  
  "token" : "hm6ly43v.0o1or"  
}
```

# Documento

Un documento es un objeto BSON

```
{  
  "_id" : ObjectId("524872a99c50880000000001"),  
  "email" : "test@asdf.com",  
  "password" : "asdf1234",  
  "name" : "Test User",  
  "date" : 1380479657300,  
  "token" : "hm6ly43v.0o1or"  
}
```

# Documento

Un documento puede contener arrays y otros documentos

```
{
  "_id" : ObjectId( "5249a2e9b90687d56453b2f3" ),
  "text" : "Soy un comentario",
  "user" : {
    "_id" : ObjectId( "524872a99c5088000000000001" ),
    "nombre" : "Test User",
    "avatar" : "/img/as09a8sd09.jpg"
  },
  "tags" : [ "test", "prueba" ]
}
```

# Documento

Un documento puede contener arrays y otros documentos

```
{
  "_id" : ObjectId("5249a2e9b90687d56453b2f3"),
  "text" : "Soy un comentario",
  "user" : {
    "_id" : ObjectId("524872a99c5088000000000001"),
    "nombre" : "Test User",
    "avatar" : "/img/as09a8sd09.jpg"
  },
  "tags" : [ "test", "prueba" ]
}
```

# Documento

Un documento puede contener arrays y otros documentos

```
{
  "_id" : ObjectId( "5249a2e9b90687d56453b2f3" ),
  "text" : "Soy un comentario",
  "user" : {
    "_id" : ObjectId( "524872a99c50880000000001" ),
    "nombre" : "Test User",
    "avatar" : "/img/as09a8sd09.jpg"
  },
  "tags" : [ "test", "prueba" ]
}
```

# Documentos

MongoDB no puede hacer JOINS

- Sin embargo, se pueden empotrar documentos y arrays
- Profundidad y complejidad arbitraria
- El límite: documento < 16MB
- Se suelen favorecer los diseños desnormalizados
  - ✓ Mucho más cómodos
  - ✓ Más eficientes en Mongo (con cuidado)
  - ⊙ Redundancia...
  - ⊙ Posible inconsistencia
  - ⊙ Actualizar los datos a mano cuando cambian



# Colecciones

Una colección es una agrupación de documentos

- Puede alojar cualquier documento (no impone estructura)
- Puede alojar documentos con diferentes formas
- Operaciones de consulta
- Es donde se ponen los índices

# Colecciones

## Operaciones sobre una colección:

- `collection.save`: guardar/actualizar un documento
- `collection.insert`: inserta un documento
- `collection.findOne`: recuperar un documento
- `collection.find`: recuperar varios documentos
- `collection.remove`: borrar uno o varios documentos
- `collection.drop`: elimina la colección
- `collection.rename`: cambia de nombre la colección
- `collection.count`: número de documentos

# Colecciones

MongoDB trae un cliente de línea de comandos

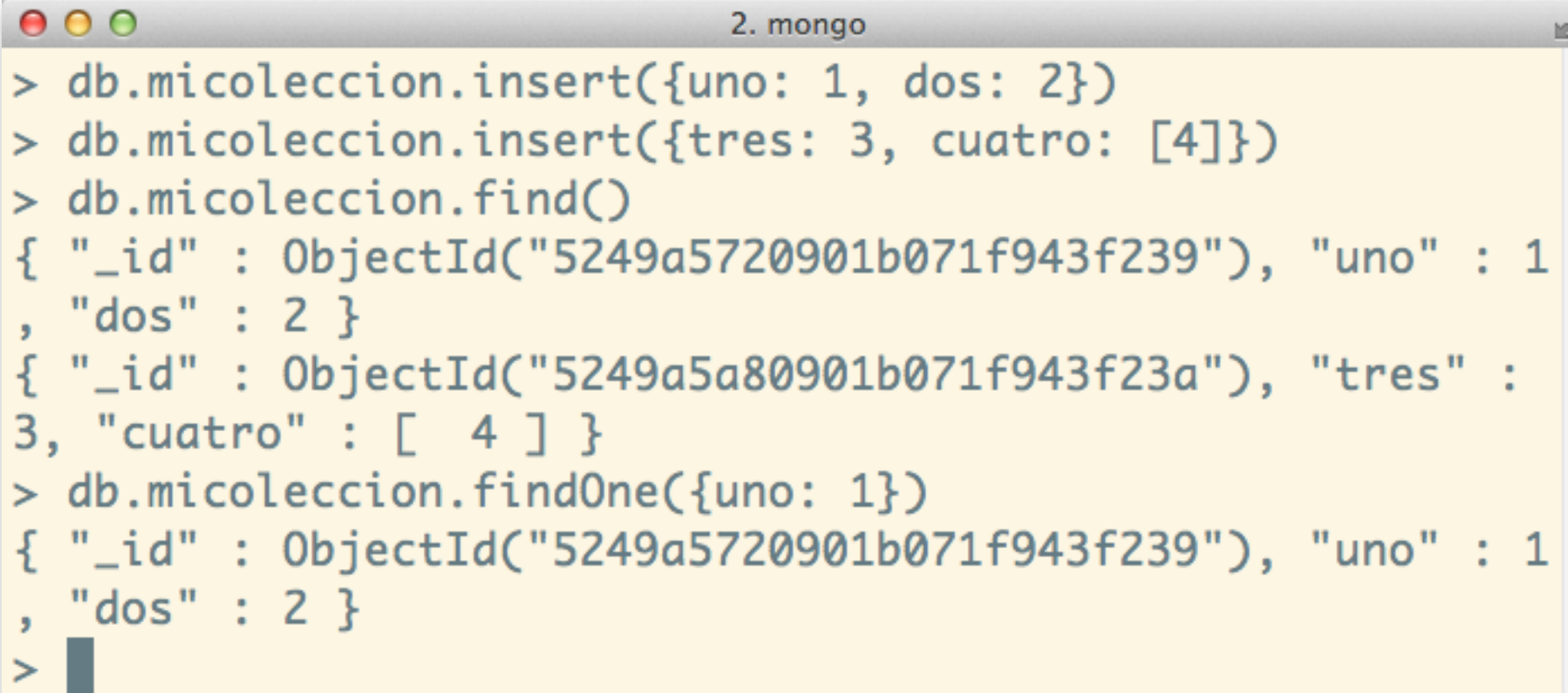
- `mongo <host>/<dbname>`
- Ejecuta JavaScript
- Muy práctico para explorar

# Colecciones

A terminal window titled "2. mongo" with a yellow background. It shows the command "courseExports \$ mongo localhost/pruebas" being executed. The output displays "MongoDB shell version: 2.4.5" and "connecting to: localhost/pruebas". The prompt ">" is followed by a dark grey cursor bar.

```
courseExports $ mongo localhost/pruebas
MongoDB shell version: 2.4.5
connecting to: localhost/pruebas
> █
```

# Colecciones



```
> db.micoleccion.insert({uno: 1, dos: 2})
> db.micoleccion.insert({tres: 3, cuatro: [4]})
> db.micoleccion.find()
{ "_id" : ObjectId("5249a5720901b071f943f239"), "uno" : 1
, "dos" : 2 }
{ "_id" : ObjectId("5249a5a80901b071f943f23a"), "tres" :
3, "cuatro" : [ 4 ] }
> db.micoleccion.findOne({uno: 1})
{ "_id" : ObjectId("5249a5720901b071f943f239"), "uno" : 1
, "dos" : 2 }
> █
```

# Colecciones

## Desde Node.js

```
/// conexión a mongo
client.then(function(db){
  var collection = db.collection("micoleccion");
  collection.insertAsync({ uno: 1, dos: 2})
    .then(function(){
      return collection.findOne({ uno: 1 })
        .toArrayAsync();
    });
});
```

# Consulta

Dos operaciones fundamentales:

- `findOne`: devuelve un documento (`findOneAsync`)
- `find`: devuelve varios documentos en un *cursor*
- Tanto `find` como `findOne` aceptan criterios `{ clave: valor, clave: valor }`
- *Sobre el cursor* podemos hacer (promisificado):
- **`.toArrayAsync()`** para devolver una promesa que resuelve con el array de documentos

# Consulta

Los operadores de búsqueda:

- \$gt / \$gte: mayor/mayor o igual
- \$lt / \$lte: menor/menor o igual
- \$ne: diferente
- \$in / \$nin: en/no en array de valores

```
micol.findOne({ valor: { $in: [ 5, 15 ] } }, cb)
```



# Consulta

## Los operadores lógicos:

- \$or: se cumple alguna cláusula
- \$and: se cumplen todas las cláusulas
- \$nor: el resultado opuesto
- \$not: no se cumplen todas las cláusulas

```
micol.findOne({$or: [  
  {valor: 5},  
  {precio: {$gt: 15 }}  
]}, callback)
```

# Consulta

Más operadores interesantes:

- Operadores de evaluación
  - Regex
  - Código JavaScript arbitrario
- Operaciones geoespaciales

➡ <http://docs.mongodb.org/manual/reference/operator/nav-query/>

# Cursores

El operador `find(...)` devuelve un cursor

- Representa un conjunto de resultados
- `cursor.count(callback)`: cantidad de documentos
- `cursor.limit(n)`: limitar a `n` documentos
- `cursor.skip(n)`: saltarse los `n` primeros documentos
- `cursor.nextObject(callback)`: siguiente documento
- `cursor.each(callback)`: para cada doc, en orden
- `cursor.toArray(callback)`: convierte el cursor en array

# Cursores

`cursor.sort(opciones, [callback])`

- Ordenar los resultados
- Opciones del tipo:
  - ▶ `[["campo", 1], ["otroCampo", -1]]`
  - ▶ 1 para ascendente, -1 para descendente

```
coleccion.find()  
    .sort([['a', -1]])  
    .nextObject(function(err, item) {  
        // ...  
    })
```

# Modificación

El operador más sencillo para modificar: save

- Si el documento es nuevo (no tiene `_id`), lo inserta
- Si el documento ya existe, lo modifica

```
db.micol.save( { nombre: "Test User" } )
```

# Modificación

insert(<documento o array>)

- Inserta uno o varios documentos en la colección

```
db.micol.insert([  
  { nombre: "Test User" },  
  { nombre: "Test User 2" }  
])
```

# Escritura

remove(<patrón>)

- Elimina los documentos que satisfagan la búsqueda

```
col.saveAsync({a: 1})  
  .then(function() {  
    return col.removeAsync({a: 1});  
  })  
  .then(function(doc) {  
    return col.countAsync();  
  })  
  .then(function(n) {  
    console.log(n);  
  });
```

# Esto es solo el principio

MongoDB tiene **muchas más funcionalidades** que nos harán falta en el mundo real:

- Data Aggregation (group, filter, funciones de agregado...)
- Índices
- Replicación, sharding



# Ejercicio: Clón de Digg

Vamos a hacer un clon de Digg/HN/Reddit/...

- El usuario se puede registrar/loguear
- Puede añadir links (url+título+descripción)
- Puede votar +1/-1 los links (solo una vez)
- Puede comentar los links posteados
- Puede votar +1/-1 los comentarios
- Recientes/Populares

# Ejercicio: Clón de Digg

La mecánica es ligeramente distinta al ej. anterior

- El cliente es una *Single Page App* en Backbone que sólo hace peticiones a una API JSON (autenticadas por token)
- No hay vistas, solo respondemos con datos JSON
- Algunas rutas tienen parámetros extra...

# Ejercicio: Clón de Digg

## Peculiaridades:

- /me: información sobre el usuario logueado
- /posts?s=<seccion>&page=<page>
  - seccion == "hottest": ordenados por voto
  - else: ordenados por fecha desc.
- /posts/:postsid/vote/up
- /posts/:postsid/vote/down
- /comments/:commentid/vote/up
- /comments/:commentid/vote/down

# Ejercicio: Clón de Digg

Para loguearse:

- El usuario manda user+pass a POST /session
- El servidor:
  1. Genera un token para el usuario
  2. Lo guarda en user.token
  3. Devuelve el JSON del usuario al cliente

# Ejercicio: Clón de Digg

Datos del usuario

```
{  
  "email" : "asdf@asdf.com",  
  "name" : "Test User",  
  "password": "secret",  
  "date" : 1380479657300,  
  "_id" : ObjectId("524872a99c50880000000001"),  
  "token" : "hm6ly43v.0o1or"  
}
```

# Ejercicio: Clón de Digg

## Datos del post

```
{
  "_id" : ObjectId("524889c998ea73000000000001"),
  "date" : 1380485577004,
  "description" : "asdf",
  "link" : "http://www.google.com",
  "ncomments" : 9,
  "title" : "titulo",
  "user" : {
    "name" : "Test User",
    "_id" : ObjectId("524872a99c5088000000000001")
  },
  "votes" : 0
}
```

# Ejercicio: Clón de Digg

## Datos del comentario

```
{
  "text" : "Comentario!",
  "post_id" : ObjectId("524889c998ea730000000001"),
  "user" : {
    "name" : "Test User",
    "_id" : ObjectId("524872a99c50880000000001")
  },
  "votes" : 0,
  "date" : 1380485577013,
  "_id" : ObjectId("524889c998ea730000000002")
}
```