



MongoDB: tutto quello che devi sapere per usarlo da subito

Webinar



Marco Breveglieri

*Sviluppatore software
consulente e trainer*

👉 <https://www.breveglieri.it>





Introduzione



NoSQL: una definizione

"A **NoSQL database** provides a mechanism for storage and retrieval of data that employs less constrained consistency models than traditional relational databases. [...]

NoSQL systems are also referred to as "Not only SQL" to emphasize that they may in fact allow SQLlike query languages to be used".

Wikipedia



Caratteristiche principali

NoSQL si riferisce a un tipo di storage che

- Non si basa sul modello relazionale (non è un RDBMS)
- Non utilizza SQL come linguaggio di interrogazione
- Favorisce la scalabilità e la disponibilità rispetto a consistenza e atomicità
- Può basarsi su tipi diversi di strutture dati



Vantaggi dei database NoSQL

- **Rappresentazione “schema-less”**: consente di definire la struttura del DB facendola evolvere nel tempo senza doverla necessariamente fissare a priori
- **Riduzione dei tempi di sviluppo**: evitare l'uso di SQL, di JOIN e di altri costrutti complessi per accedere ai dati consente di scrivere codice più rapidamente
- **Elevate performance**: i dati possono essere trasferiti a velocità molto elevate, rispondendo a molti più utenti
- **Estrema scalabilità**: possono gestire picchi di utilizzo garantendo comunque il servizio agli utenti e scalando orizzontalmente (*scaling out*)



Svantaggi dei database NoSQL

- **Supporto limitato o assente a join**: l'accesso ai dati è fatto a livello di documento, quindi i legami tra oggetti vanno istituiti in modo applicativo
- **Transazioni (spesso) assenti**: si basano su “eventual consistency”, non lavorano su aggiornamenti batch, bensì su singoli documenti
- **Constraint non implementati**: non esistono foreign key, né le relative restrizioni, né integrità referenziale



Tipologie di database NoSQL

- **Column Store** (*Big Table, Cassandra, ...*)
 - Memorizza i dati organizzati in colonne
- **Document Store** (*MongoDB, CouchDB, ...*)
 - Conosciuto anche come “Document Oriented Database”
 - Consente di inserire, manipolare ed estrarre dati strutturati a documento (es. JSON)
 - Consente di creare indici basati su proprietà dei documenti
- **Key-Value Store** (*Redis, MemCacheDB, ...*)
 - Simile a Document Store, ma i documenti vengono recuperati tramite chiave e sono “opachi”
 - Ottimizzano al massimo la ricerca sulla chiave
- **Graph** (*Neo4J, FlockDB, ...*)
 - Generalmente basato su grafi (come suggerisce il nome)
 - Indicato per archivi pesantemente basati su relazioni tra i dati



Let's meet MongoDB!



MongoDB

MongoDB è un “document oriented database” di tipo NoSQL, open-source, rilasciato gratuitamente dietro General Public License (GPL).

Scateniamoci alla tastiera!



Vediamo le principali distribuzioni di MongoDB, dove scaricarlo, come configurare un'istanza e lanciarlo per iniziare a lavorare...





Primi passi



Fondamenti di MongoDB

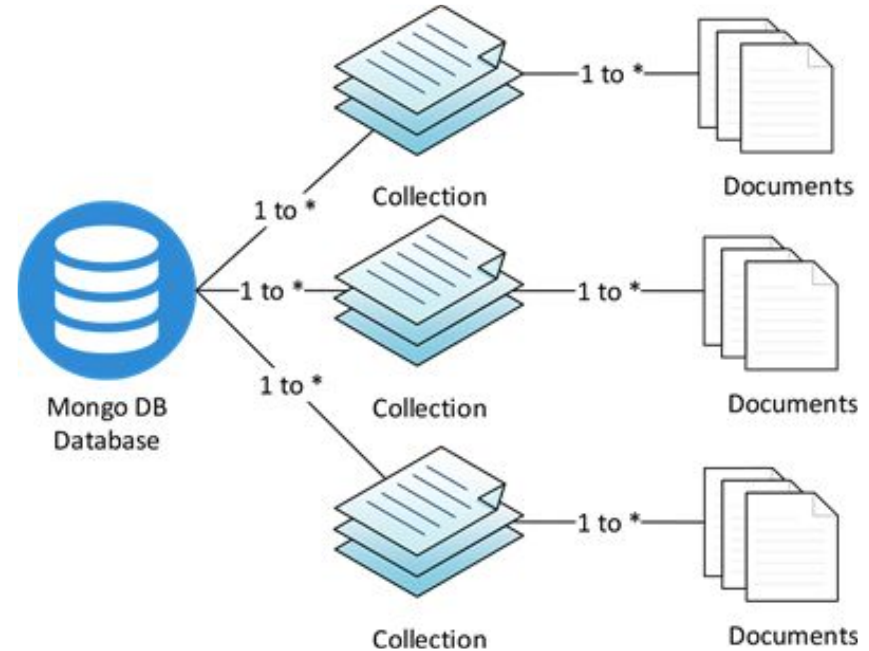
MongoDB è un database “document oriented” che si basa su due elementi fondamentali:

- **Document**: è il singolo documento che contiene i dati
 - *Struttura formata da coppie chiave/valore*
 - *E' rappresentato da un oggetto JSON*
 - *Viene salvato su disco in formato binario (BSON)*
- **Collection**: è un contenitore di uno o più documenti
 - *Simile (in un certo senso) alla tabella di un RDBMS*
 - *E' schema-less (contiene oggetti di qualsiasi tipo)*



Pensare a “documenti”

Una delle differenze principali tra i database RDBMS e MongoDB risiede nell'organizzazione dei dati.



Scateniamoci alla tastiera!



Testiamo database, collection e documenti, muovendo i primi passi con i dati...





Relazioni tra documenti

Le relazioni tra i documenti si possono implementare con due differenti modalità:

- **Embedded**: il documento correlato viene “inglobato” nel documento principale
- **Reference**: il documento correlato contiene un riferimento al documento principale, o viceversa

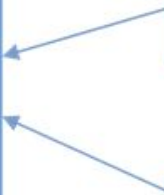


Relazioni tramite “reference”

```
user document:
{
  _id: <User1>,
  username: "johndoe",
  firstname: "john",
  lastname: "doe"
}
```

```
address document:
{
  _id: <address1>,
  user_id: <User1>,
  city: "Zürich",
  country: "Switzerland"
}
```

```
contact document:
{
  _id: <contact1>,
  user_id: <User1>,
  phone: "+41 123-123-123",
  email: "jd@jd.com"
}
```





Relazioni tramite “embedding”

```
user document:
{
  _id: <User1>,
  username: "johndoe",
  firstname: "john",
  lastname: "doe"
  address: {
    city: "Zürich",
    country: "Switzerland"
  },
  contact: {
    phone: "+41 123-123-123",
    email: "jd@jd.com"
  }
}
```



Document Design Strategy

- Usare l'embedding quando possibile
 - Eliminare la necessità di join tra i dati
 - Salvare e recuperare dati correlati risulta "atomico" e veloce
 - Favorire i dati che non sono usati da altri documenti
- Normalizzare i dati da riutilizzare
 - Creare una Collection distinta per ogni dato riusabile o ricercabile
- Non eccedere la dimensione massima del documento
 - MongoDB prevede un massimo di 16 MB per ciascun singolo documento



Database Tools



Tool e loro utilizzo

Tool principali

- **mongod**: è il processo che esegue un'istanza di MongoDB su una singola macchina
- **mongos**: è il servizio che si pone tra l'applicazione e i database per recuperare o salvare i dati sugli shard in base a quanto comunicato dal servizio di configurazione
- **mongoosh**: è la (nuova) shell a riga di comando che consente di interagire con MongoDB usando una interfaccia basata su JavaScript

Tool aggiuntivi

- **mongodump**: è una utility per creare una esportazione binaria dei dati su DB
- **mongorestore**: scrive i dati binari creati da un dump in una istanza del DB
- **bsondump**: è un tool diagnostico per ispezionare i file BSON e convertirli in formati leggibili dall'utente (es. viene usato per i file creati da mongodump)
- **mongoexport**: esporta dati dal DB all'interno di file JSON, CSV, TSV ecc.
- **mongoimport**: importa dati da file JSON, CSV, TSV generati da mongoexport
- **mongostat**: fornisce informazioni rapide sullo stato di una istanza di MongoDB
- **mongofiles**: gestisce i file su DB organizzati come oggetti GridFS



Scateniamoci alla tastiera!



Importiamo dati da file JSON
all'interno del nostro database.





Indicizziamo...



L'importanza degli indici

Gli indici di MongoDB sono basati su *B-tree*, e consentono di

- *ottimizzare l'esecuzione delle query*
- *sfruttare l'hardware in modo efficace*

In sintesi, riducono il lavoro necessario al reperimento dei documenti ricercati.



Tipologie di indici

Indici a **chiave singola** (*Single Key*):

- Sono indici composti da un singolo valore per ogni documento
- Ogni collection ne ha uno predefinito (basato sul campo “_id”)

Indici a **chiave composta** (*Compound Key*):

- Sono composti da più valori per ogni singolo documento
- Sono consigliati per velocizzare le query su più campi

Attenzione: quando si crea un indice, valutare bene i benefici osservando il rapporto tra dimensioni/performance (sia in fase di ricerca che su inserimento/aggiornamento)!



Caratteristiche degli indici

- Solo un indice verrà selezionato per eseguire una determinata query
- Se una query richiede più campi nella ricerca, occorre avere un indice composto per ottimizzarla appieno
- Un indice semplice può essere rimosso se esiste un indice composto che lo include



Scateniamoci alla tastiera!



Testiamo gli indici sui nostri dati di esempio...





GridFS



Che cos'è GridFS?

GridFS è una specifica proposta come standard per salvare e recuperare file.

- *Viene utilizzata in MongoDB per la gestione dei file*
- *Consente di superare il limite di 16 MB per documento*
- *Permette di collegare uno o più file ai documenti*
- *I file possono essere “ridondati” e sottoposti a backup*



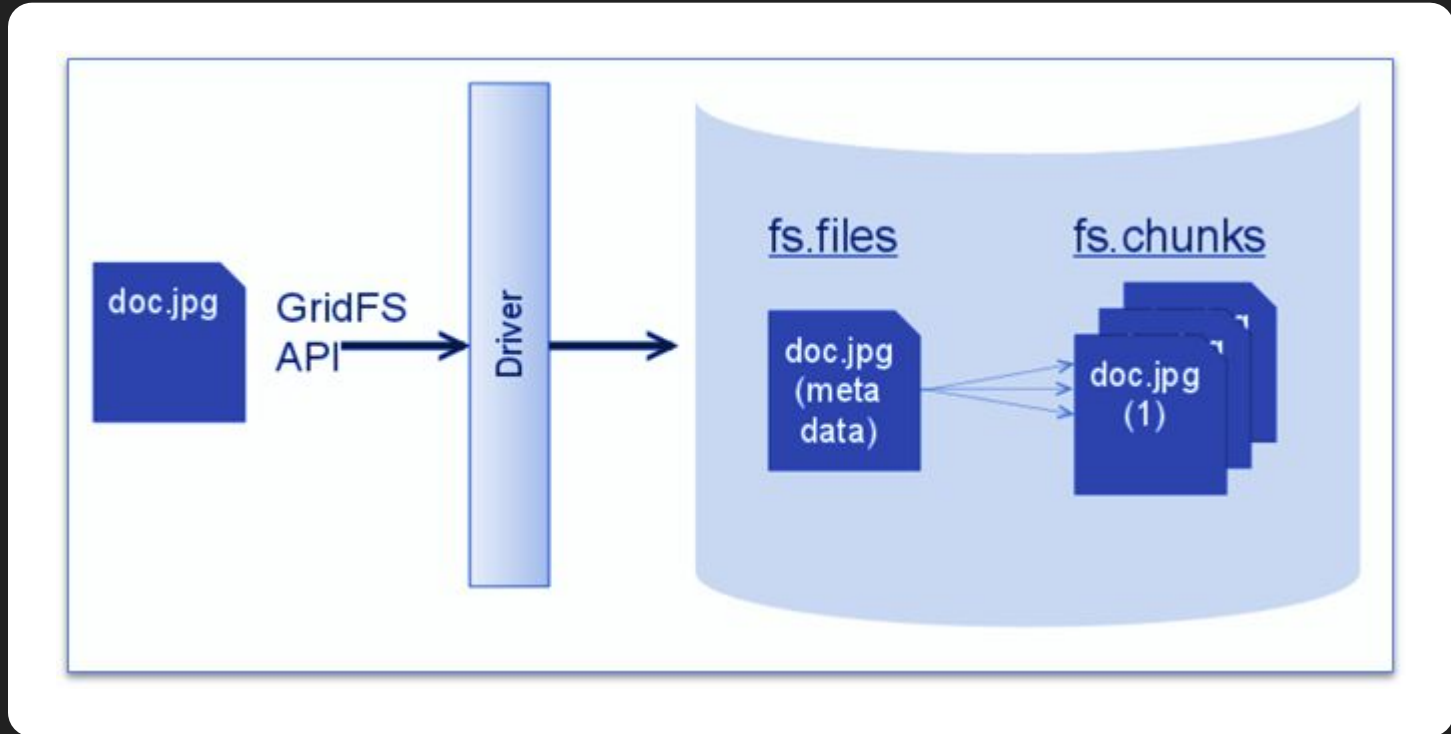
GridFS in MongoDB /1

GridFS usa due *collection* su DB per memorizzare i file:

- ***fs.files*** → *contiene i metadati dei file memorizzati*
- ***fs.chunks*** → *contiene i dati veri e propri dei file, in formato binario, suddivisi in segmenti (255 Kb max)*



GridFS in MongoDB /2





Scateniamoci alla tastiera!



Importiamo un file nel nostro database
ed esploriamo le collection di GridFS...





MongoDB Drivers



Caratteristiche dei driver

- Tutti i driver, come qualsiasi linguaggio, parlano con MongoDB
- Possono essere usati in applicazioni Web, backend, ecc.
- Oltre a quelli “ufficiali”, ve ne sono una grande quantità di terze parti
- Il nome delle funzioni e dei parametri è **identico** a quelli della shell
- Nella documentazione ufficiale si trovano le istruzioni per tutti i client



Driver per ogni linguaggio

 C

 C++


 C#

 Go

 Java

 Node.js


 PHP

 Python

 Ruby

 Rust

 Scala

 Swift



Piccolo demo in Python



Replica & Sharding



Cos'è il Replica Set?

- Legge di Murphy: “Se qualcosa può andare male... lo farà!”
- In ambiente di produzione, è preferibile avere una ridondanza dei dati e adottare misure per il cosiddetto “*disaster recovery*”.

L'uso di *Replica Set* garantisce durabilità ai dati.



Nella configurazione, tenere presente il motto “*Two is One, One is None*”.



Nodi del “Replica Set”

Il *Replica Set* è formato da **nodi**.

- **Primario** → esso riceve le operazioni di lettura/scrittura
- **Secondario** → nodo con istanza avente ruolo “non primario”, che funge quindi da “gregario” (consente solo le letture)
- **Arbiter** → nodo con il ruolo di esprimere solo votazioni per eleggere il nodo “primario” quando necessario (facoltativo)

Ciascun nodo è gestito da una istanza di MongoDB (`mongod`) che può essere installato su altre macchine o reti, o geo-localizzato se in cloud.



Election

- In caso di “Fail Over”, i nodi svolgono un’elezione per stabilire il nuovo server primario (*PRIMARY*)
- L’elezione si svolge tra i restanti nodi disponibili, inclusi gli arbiter
- Il 50% o più dei voti espressi dai membri possono eleggere un “primario”
- Se risulta impossibile eleggere un server primario, i dati rimangono in sola lettura (*read only*)



Scateniamoci alla tastiera!



Realizziamo un Replica Set...





Sharding

Distribuire i dati del database su più di un server (o meglio, “replica set”).

E' necessario quando

- la dimensione della base dati cresce al punto da incidere sulle query
- aumentano le richieste di lettura e scrittura
- la RAM non copre l'indicizzazione della quantità di dati crescente
- il numero di CPU e di core non è sufficiente a gestire il lavoro
- eseguire il backup diventa oneroso per tempistiche e dimensioni



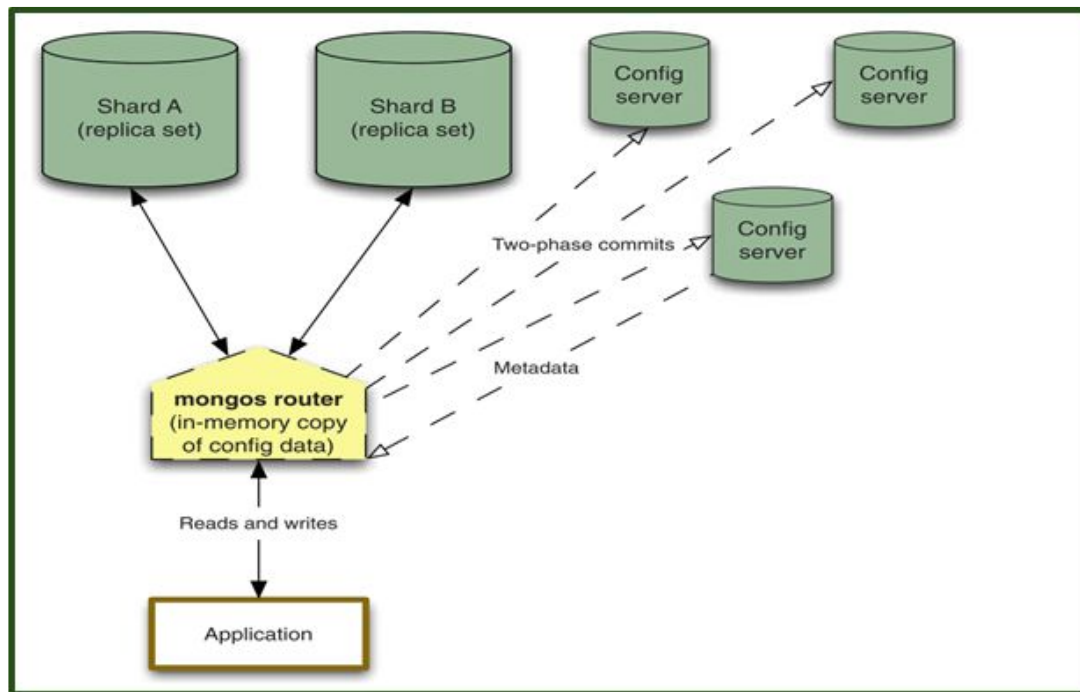
Lo “sharded cluster”

Uno “shared cluster” è costituito da

- **Shards** → contiene una parte dei dati gestito con “replica set”
 - La distribuzione può avvenire “per database” o “per collection” (più frequente)
 - La suddivisione avviene in base a una “Shard Key” e ai rispettivi range di valori
- **Config Servers** → mantiene in modo persistente la configurazione e lo stato del cluster di shard
- **Mongos Routers** → rappresenta il punto di ingresso allo “shared cluster”, al quale rivolgere le richieste



Panoramica dell'architettura





Panoramica dell'architettura

Start	End	Shard
$-\infty$	abbot	B
abbot	dayton	A
dayton	harris	B
harris	norris	A
norris	$+\infty$	B



Conclusioni



Risorse e approfondimenti

- Sito ufficiale di **MongoDB**
<https://www.mongodb.com/>
- Versione **Atlas** di MongoDB
<https://cloud.mongodb.com>
- **Download server e tools**
<https://www.mongodb.com/try/download/community>
<https://www.mongodb.com/try/download/tools>
- Driver per linguaggi di programmazione
<https://www.mongodb.com/docs/drivers>



Q & A





Grazie!

