MONGO INSTALLATION

- Mongo ha una potente capacità di archiviare e interrogare i dati in vari modi.
- Ciò che rende unici i database dei documenti è la loro capacità di gestire in modo efficiente documenti di dati nidificati arbitrariamente e senza schemi.
- Finora, abbiamo eseguito Mongo come un singolo server.
- Ma se si dovesse eseguire Mongo in produzione, si vorrebbe eseguirlo come un **cluster di macchine**, che fornirebbe una disponibilità molto più elevata e consentirebbe di re**plicare i dati su server, frammentare raccolte in molti pezzi ed eseguire query in parallelo**.

REPLICA SET

- Mongo doveva ridimensionarsi, non funzionare in modalità standalone.
- È stato creato per la coerenza dei dati e la tolleranza della partizione, ma la condivisione dei dati ha un costo: se una parte di una raccolta viene persa, l'intera struttura viene compromessa.
  Mongo affronta questa debolezza implicita di sharding in un modo semplice: **duplicazione**.
  Raramente dovresti eseguire una singola istanza Mongo in produzione e invece dovresti replicare i dati memorizzati su più servizi.

Oggi inizieremo da zero e genereremo alcuni nuovi server.
La porta predefinita di Mongo è 27017, quindi avvieremo ogni server su altre porte.
Ricorda che devi prima creare le directory dei dati, quindi creane tre:

mkdir mongo1 mongo2 mongo3

Successivamente, accenderemo i server Mongo.
Questa volta aggiungeremo il flag replSet con il libro dei nomi e specificheremo le porte.

mongod --replSet book --dbpath ./mongo1 --port 27011

Ora, occorre aprire un'altra finestra del terminale ed eseguire il comando successivo, che avvia un altro server, indicando una directory diversa, disponibile su un'altra porta.
Quindi aprire un terzo terminale per avviare il terzo server.

mongod --replSet book --dbpath ./mongo2 --port 27012
mongod --replSet book --dbpath ./mongo3 --port 27013

Si nota che si ottiene molto di questo messaggio sull'output, con messaggi di errore come questo:
[initandlisten] Did not find local voted for document at startup
È una buona cosa perché non abbiamo ancora inizializzato il nostro set di repliche e Mongo ce lo sta facendo sapere.

Avvia una shell mongo su uno dei server ed esegui la funzione rs.initiate ().

mongo localhost:27011

rs.initiate({_id: 'book',
members: [
{_id: 1, host: 'localhost:27011'},
{_id: 2, host: 'localhost:27012'},
{_id: 3, host: 'localhost:27013'}
]
})

rs.status().ok

Si noti che stiamo utilizzando un nuovo oggetto chiamato **rs (set di repliche) anziché db (database).**
Come altri oggetti, ha un metodo *help()* che puoi chiamare.
L'esecuzione del comando *status()* ci farà sapere quando il nostro set di repliche è in esecuzione, quindi continua a controllare lo stato per il completamento prima di continuare.
Se guardi i tre output del server, dovresti vedere che un server emette questa linea:

Member ... is now in state PRIMARY

E gli altri 2 server emetteranno il seguente output:

Member ... is now in state SECONDARY

PRIMARY sarà il server principale. È probabile che questo sarà il server sulla porta 27011 (perché è stato avviato per primo); tuttavia, in caso contrario, vai avanti e accendi una console sul primario. Inserisci qualsiasi cosa vecchia nella riga di comando e proveremo un esperimento.

db.echo.insert({ say : 'HELLO!' })

Dopo l'inserimento, esci dalla console e quindi testiamo che la nostra modifica sia stata replicata chiudendo il nodo principale; premendo Ctrl + C è sufficiente. Se guardi i log dei restanti due server, dovresti vedere che uno dei due è stato promosso a master.

Apri una console in quella macchina (per noi era localhost:27012) e *db.echo.find()* dovrebbe contenere il tuo valore.

Per sapere il nodo master:

db.isMaster().primary

Note:

- non possiamo né scrivere su un nodo secondario né leggere direttamente da esso.
- Esiste un solo master per set di repliche e devi interagire con esso.
- È il guardiano del set.
- La replica dei dati comporta dei problemi (di sincronizzazione delle copie con il dato master).
  Nella configurazione di Mongo, un problema è decidere chi viene promosso quando un nodo principale si interrompe.
  Mongo si occupa di questo dando un voto a ciascun servizio mongod e quello con i dati più recenti viene eletto nuovo padrone.
  In questo momento, dovresti avere ancora due servizi mongod in esecuzione.
  Vai avanti e spegni il master corrente.
  Ricorda, quando l'abbiamo fatto con tre nodi, uno degli altri è stato appena promosso a diventare il nuovo maestro.
  Ora l'ultimo nodo rimanente è implicitamente il master.
  Vai avanti e riavvia gli altri server e guarda i registri.
  Quando i nodi vengono ripristinati, entrano in uno stato di recupero e tentano di risincronizzare i loro dati con il nuovo nodo principale.
  "Apetta un minuto!?"
  Quindi, se il master originale avesse dati che non si sono ancora propagati?
  Tali operazioni vengono eliminate.
  Una scrittura in un set di repliche Mongo non viene considerata corretta fino a quando la maggior parte dei nodi non ha una copia dei dati.

Il problema dei nodi pari
Il concetto di replica è abbastanza semplice da comprendere:

- si **scrive su un server MongoDB e i dati vengono duplicati su altri all'interno del set di repliche**.
- Se un server non è disponibile, uno degli altri può essere promosso e soddisfare le richieste.
- Ma un server può non essere disponibile in più modi di un arresto anomalo del server.
- A volte, la connessione di rete tra i nodi è inattiva.
- In tal caso, **Mongo impone che la maggior parte dei nodi che possono ancora comunicare ora costituisce la rete**.
- MongoDB prevede un **numero dispari di nodi totali nel set di repliche**.
  Senza una chiara maggioranza, non è stato possibile raggiungere un quorum.

SHARDING

Uno degli obiettivi principali di Mongo è fornire una **gestione rapida e sicura di set di dati molto grandi**. Il metodo più chiaro per raggiungere questo obiettivo è attraverso lo **sharding orizzontale** per intervalli di valori.

Invece di un singolo server che ospita tutti i valori in una raccolta, alcuni intervalli di valori vengono suddivisi o suddivisi su altri server.

- Ad esempio, nella nostra raccolta di numeri di telefono, possiamo inserire tutti i numeri di telefono inferiori a 1.500.000.000 sul server Mongo.
- E metti numeri maggiori o uguali a 1.500.000.0001 sul server B.
  Mongo rende tutto più semplice autosharding, gestendo questa divisione per te → fa lo sharding in maniera automatica!

Ora, vediamo come lanciare un paio di server mongod (non replicanti).
Come per i set di repliche, è necessario un parametro speciale per essere considerato un server shard (il che significa che questo server è in grado di eseguire il sharding).

mkdir mongo4 mongo5

mongod --shardsvr --dbpath ./mongo4 --port 27014
mongod --shardsvr --dbpath ./mongo5 --port 27015

Ora si ha bisogno di un **server per tenere traccia delle tue chiavi**.
Immagina di aver creato una tabella per memorizzare i nomi delle città in ordine alfabetico.
È necessario un modo per sapere che, ad esempio, le città che iniziano da A a N vanno al server mongo4 e da O a Z al server mongo5.
In Mongo, crei un server di configurazione (che è solo un normale mongod) che tiene traccia di quale server (mongo4 o mongo5) possiede quali valori.
Dovrai creare e inizializzare un **secondo set di repliche per la configurazione del cluster** (chiamiamolo configSet).

mkdir ./mongoconfig

mongod --configsvr --replSet configSet --dbpath ./mongoconfig --port 27016

Ora accedi alla shell Mongo per il server di configurazione eseguendo

mongo localhost: 27016

e avvia il cluster di server di configurazione (con un solo membro per questo esempio):

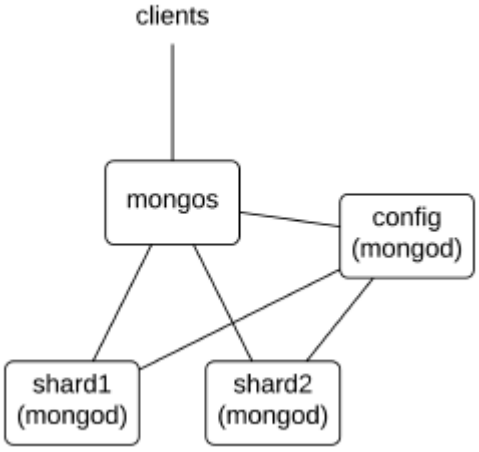
```
rs.initiate({
_id: 'configSet',
configsvr: true,
members: [{
        _id: 0,
        host: 'localhost:27016'
}]
})
```

rs.status().ok

Infine, devi eseguire ancora un altro server chiamato mongos, che è l'unico punto di accesso per i nostri clienti.
Il server mongos si connetterà al server di configurazione mongoconfig per tenere traccia delle informazioni di sharding ivi memorizzate.
Puntate i mongos al replSet / server: port con il flag --configdb.

mongos --configdb configSet/localhost:27016 --port 27020

La seguente immagine della configurazione del nostro server può essere d'aiuto.



Ora passiamo alla console del server mongos nel database admin eseguendo

mongo localhost: 27020/admin

Configureremo 2 sharding.

sh.addShard('localhost:27014')

sh.addShard('localhost:27015')

Con questa configurazione, ora occorre fronire il database e la raccolta per frammentare i dati (nel nostro caso, il nome della città).

db.runCommand({ enablesharding : "test" })

db.runCommand({ shardcollection : "test.cities", key : {name : 1} })
{ "collectionsharded" : "test.cities", "ok" : 1 }

Con tutto ciò che è installato, carichiamo alcuni dati.
Se scarichi il codice libro, troverai un file di dati da 12 MB chiamato *mongoCities100000.json* che contiene dati per ogni città del mondo con una popolazione di oltre 1.000 persone.
Scarica quel file ed esegui il seguente script di importazione che importa i dati nel tuo server mongos:

*mongoimport *
*--host localhost:27020 *
*--db test *
*--collection cities *
*--type json *
mongoCities100000.json

Se l'importazione ha esito positivo, dovresti vedere 99838 documenti importati nell'output.