**Infrastructure**

<https://docs.mongodb.com/manual/replication/>

<https://stackoverflow.com/questions/11571273/in-mongo-what-is-the-difference-between-sharding-and-replication>

## Pros

With multiple copies of data on different database servers. Provides:

* a level of fault tolerance against the loss of a single database server.
* increased read capacity as clients can send read operations to different servers.

## Sharding – write

Write to different Shard (could be a Replica set). mongos – router – decides by shard key.

Horizontal scaling.

[*sharding*](http://www.mongodb.org/display/DOCS/Sharding+Introduction) allows for horizontal scaling of data writes by partitioning data across multiple servers using a shard key. It's important to [choose a good shard key](http://www.mongodb.org/display/DOCS/Choosing+a+Shard+Key). For example, a poor choice of shard key could lead to "hot spots" of data only being written on a single shard.

Sharding sits on top of replication, usually.

## Replication – read

Read from many mongod instances which have the same data set.

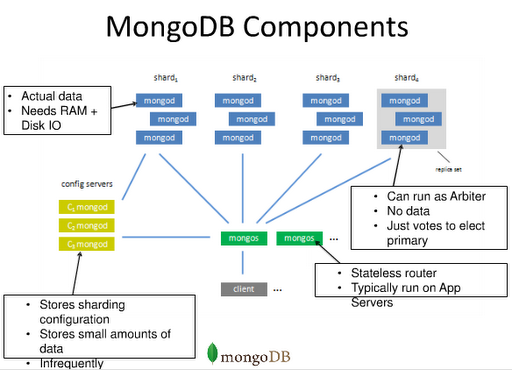
Automatic failover.

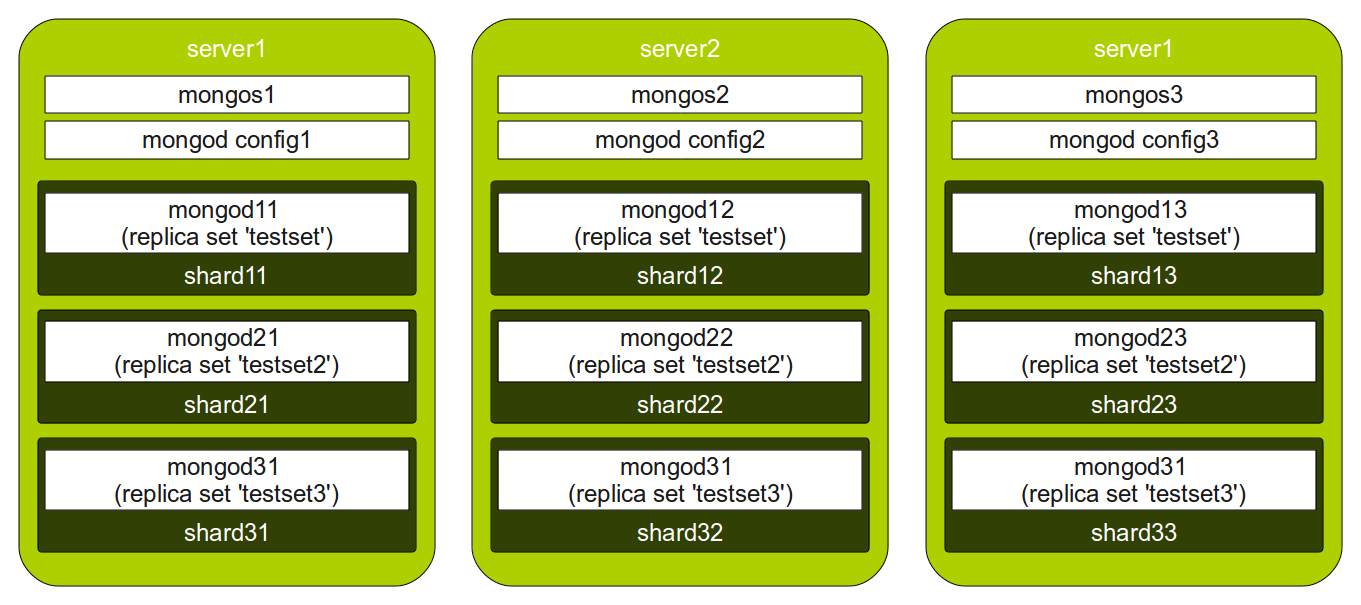
[*replication*](http://docs.mongodb.org/manual/core/replication/) creates additional copies of the data and allows for automatic failover to another node. Replication may help with horizontal scaling of reads if you are OK to read data that potentially isn't the latest.

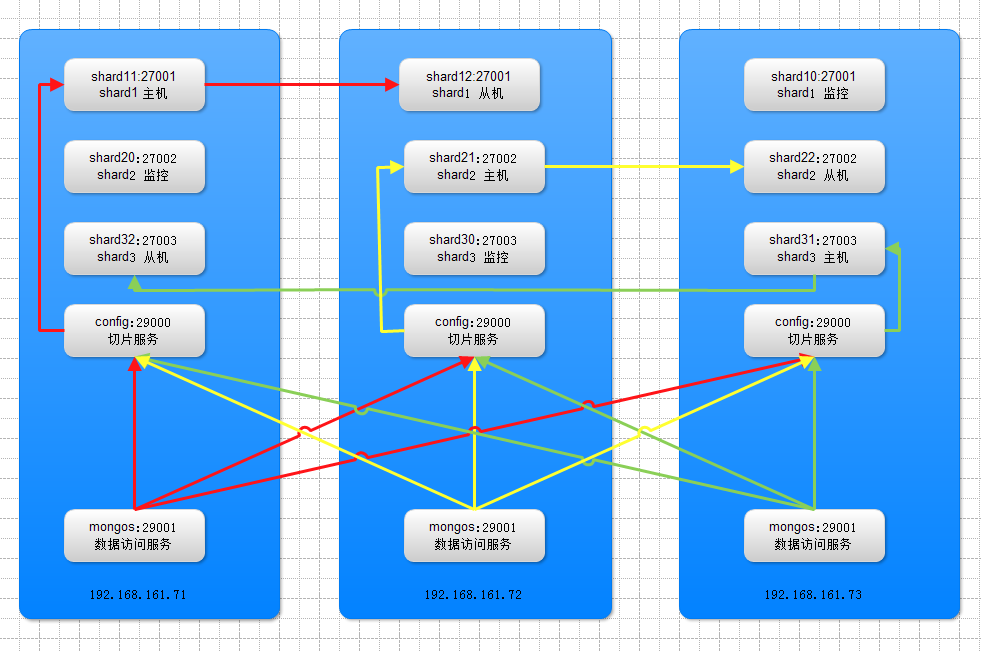
### Replica set

A replica set (Logically, each replica set can be seen as a **shard**) is a group of [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) instances that maintain the same data set. A replica set – master/slave setup – keeps the data in sync across several different instances so that if one of them goes down, we won't lose any data.

* several data bearing nodes
  + one and only one member is deemed the primary node -> receives all write operations
  + other nodes are deemed secondary nodes
* optionally one arbiter node

[](https://www.google.com.vn/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&ved=0ahUKEwiVgpGhuavVAhVqhlQKHcPIAE8QjRwIBw&url=http://www.fastcampus.co.kr/data_camp_mgdb_instructor_1/&psig=AFQjCNEsqnW4eVlOJNB3v6cdBjXqIgbXnw&ust=1501311754160811)





## So, what do we need to really know as a developer?

* insert must include a shard key, so if it's a multi-parted shard key, we must include the entire shard key
* we've to understand what the shard key is on collection itself
* for an update, remove, find - if mongos is not given a shard key - then it's going to have to broadcast the request to all the different shards that cover the collection.
* for an update - if we don't specify the entire shard key, we have to make it a multi update so that it knows that it needs to broadcast it

# Ops Manager

## Install

<https://docs.opsmanager.mongodb.com/current/tutorial/install-on-prem-from-archive/>

env: Linux, windows server.

### Download

<https://www.mongodb.com/download-center?jmp=nav#ops-manager>

### Extract the archive

tar -zxf mongodb-mms-<version>.x86\_64.tar.gz

## Configure connection to the Ops Manager Application Database

On a server that is to run Ops Manager, open <install\_directory>/conf/conf-mms.properties  
[mongo.mongoUri](https://docs.opsmanager.mongodb.com/current/reference/configuration/#mongo.mongoUri): mongodb://10.55.16.254:27017/?maxPoolSize=150  
db for this Ops Manager

### Start Ops Manager

<install\_directory>/bin/mongodb-mms start

### Open the Ops Manager home page and register the first user

http://<host>:8080

### Configure Ops Manager

Possible:

* running multiple Ops Manager instances behind a load balancer
* using Automation or Backup without an internet connection

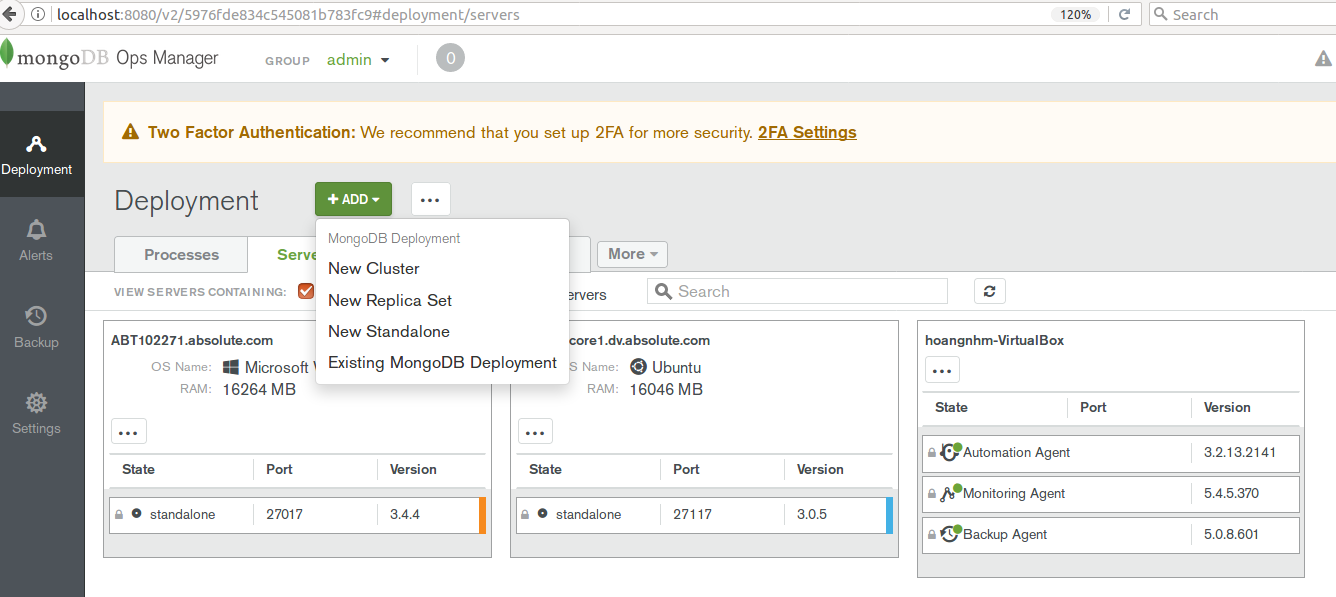
## Setup

### Setup Agents (Automation, Monitoring, Backup)

* On server that run Ops Manager for basic monitoring.
* On server that run MongoDB deployments for advanced controls.

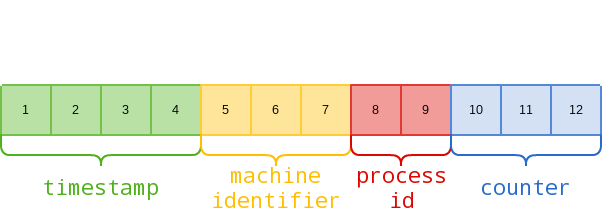
`

### Add Deployment



Note: using hostname not ip.

**\_id** là một số thập lục phân 12 byte

[](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&ved=0ahUKEwju8u__m5rVAhWKxVQKHYZCBtwQjRwIBw&url=http://techidiocy.com/_id-objectid-in-mongodb/&psig=AFQjCNGByahfS3S9b20Ykv7MP74d09Smwg&ust=1500721122920084)

4 byte đầu là cho Timestamp hiện tại, 3 byte tiếp theo cho ID của thiết bị, 2 byte tiếp là process id của MongoDB Server và 3 byte còn lại là giá trị có thể tang.

**db**

Trong MongoDB, cơ sở dữ liệu mặc định là test. Nếu bạn đã không tạo bất kỳ cơ sở dữ liệu nào, thì các Collection sẽ được lưu giữ trong test.

>show dbs

local 0.78125GB

test 0.23012GB

Cơ sở dữ liệu mydb đã được tạo của bạn không có trong danh sách này. Để hiển thị nó, bạn cần chèn ít nhất một Collection vào trong đó.

Query

And: ,

Or: $or

**Relationship**

Embeded

Điểm hạn chế ở đây là, nếu Document được nhúng tiếp tục tăng kích cỡ quá nhiều, nó sẽ ảnh hưởng đến hiệu suất đọc/ghi.

{

"\_id":ObjectId("52ffc33cd85242f436000001"),

"contact": "987654321",

"dob": "01-01-1991",

"name": "Tom Benzamin",

"address": [

{

"building": "22 A, Indiana Apt",

"pincode": 123456,

"city": "Los Angeles",

"state": "California"

},

{

"building": "170 A, Acropolis Apt",

"pincode": 456789,

"city": "Chicago",

"state": "Illinois"

}]

}

Referenced (Manual References)

Cần 2 truy vấn

{

"\_id":ObjectId("52ffc33cd85242f436000001"),

"contact": "987654321",

"dob": "01-01-1991",

"name": "Tom Benzamin",

"address\_ids": [

ObjectId("52ffc4a5d85242602e000000"),

ObjectId("52ffc4a5d85242602e000001")

]

}

**DBRefs**

Tuy nhiên, trong các trường hợp mà một Document chứa nhiều tham chiếu từ các Collection khác nhau, thì chúng ta sử dụng **DBRefs.**

**$ref:** Trường này xác định Collection của Document được tham chiếu.

**$id:** Trường này xác định trường \_id của Document được tham chiếu.

**$db:** Trường này là một trường tùy ý, chứa tên của Database mà Document được tham chiếu ở trong đó.

{

"\_id":ObjectId("53402597d852426020000002"),

"address": {

"$ref": "address\_home",

"$id": ObjectId("534009e4d852427820000002"),

"$db": "tutorialspoint"},

"contact": "987654321",

"dob": "01-01-1991",

"name": "Tom Benzamin"

}

**Covered Query**

* All fields was index.
* All return fields belong to same index.

Lastly, remember that an index cannot cover a query if −

* Any of the indexed fields is an array
* Any of the indexed fields is a subdocument

[**https://www.tutorialspoint.com/mongodb/mongodb\_covered\_queries.htm**](https://www.tutorialspoint.com/mongodb/mongodb_covered_queries.htm)

# Index, [text](https://docs.mongodb.com/manual/core/index-text/) index

## List

db.people.getIndexes()

## Create

db.collection.createIndex(

{

content: "text",

"users.comments": "text",

"users.profiles": "text"

},

{

name: "MyTextIndex"

}

)

## ReIndex

db.collection.reIndex()

This operation drops all indexes, including the [\_id index](https://docs.mongodb.com/manual/indexes/#index-type-id), and then rebuilds all indexes.

### Usage

Normally, MongoDB compacts indexes during routine updates. For most users, the db.collection.reIndex() is unnecessary. However, it may be worth running if the collection size has changed significantly or if the indexes are consuming a disproportionate amount of disk space.

#### Modify an Index

To modify an existing index, you need to drop and recreate the index with the exception of m TTL indexes.

#### Rebuild Indexes

For replica sets, [db.collection.reIndex()](https://docs.mongodb.com/manual/reference/method/db.collection.reIndex/#db.collection.reIndex) will not propagate from the [primary](https://docs.mongodb.com/manual/reference/glossary/#term-primary) to [secondaries](https://docs.mongodb.com/manual/reference/glossary/#term-secondary). [db.collection.reIndex()](https://docs.mongodb.com/manual/reference/method/db.collection.reIndex/#db.collection.reIndex) will only affect a single [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) instance.

[db.collection.reIndex()](https://docs.mongodb.com/manual/reference/method/db.collection.reIndex/#db.collection.reIndex) will rebuild indexes in the [background](https://docs.mongodb.com/manual/core/index-creation/#index-creation-background) if the index was originally specified with this option. However, [db.collection.reIndex()](https://docs.mongodb.com/manual/reference/method/db.collection.reIndex/#db.collection.reIndex) will rebuild the \_id index in the foreground, which takes the database’s write lock.

## Drop

[db.collection.dropIndexes()](https://docs.mongodb.com/manual/reference/method/db.collection.dropIndexes/#db.collection.dropIndexes)

Remove all indexes, except for the [\_id index](https://docs.mongodb.com/manual/indexes/#index-type-id) from a collection.

[db.collection.dropIndex()](https://docs.mongodb.com/manual/reference/method/db.collection.dropIndex/#db.collection.dropIndex)

# Case insensitive

findOne(), $regex, $text, spring-data

# MongoTemplate and MongoRepository?

<https://stackoverflow.com/questions/17008947/whats-the-difference-between-spring-datas-mongotemplate-and-mongorepository>

"Convenient" and "powerful to use" are contradicting goals to some degree. Repositories are by far more convenient that the templates but the latter of course give you more fine-grained control over what to execute.

As the repository programming model is available for multiple Spring Data modules you find more in-depth documentation of it in the general section of the Spring Data MongoDB [reference docs](http://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#repositories).

**TL;DR**

We generally recommend the following approach:

1. Start with the repository abstract and just declare simple queries using the query derivation mechanism or manually defined queries.
2. For more complex queries, add manually implemented methods to the repository (as documented here). For the implementation use MongoTemplate.

**Details**

1. For your example this would look something like this:

Define an interface for your custom code:

interface CustomUserRepository {

List<User> yourCustomMethod();

}

1. Add an implementation for this class and follow the naming convention to make sure we can find the class.

class UserRepositoryImpl implements CustomUserRepository {

private final MongoOperations operations;

@Autowired

public UserRepositoryImpl(MongoOperations operations) {

Assert.notNull(operations, "MongoOperations must not be null!");

this.operations = operations;

}

public List<User> yourCustomMethod() {

// custom implementation here

}

}

1. Now let your base repository interface extend the custom one and the infrastructure will automatically use your custom implementation:

interface UserRepository extends CrudRepository<User, Long>, CustomUserRepository {

}

This way you essentially get the choice: everything that just easy to declare goes into UserRepository, everything that's better implemented manually goes into CustomUserRepository. The customization options are documented [here](http://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#repositories.custom-implementations).

|  |  |  |
| --- | --- | --- |
| |  | | --- | |  | | The most common mistake is to name the implementation class wrong: if your base repo interface is called YourRepository, the implementation class has to be named YourRepositoryImpl. Is that the case? If so I'm happy to take a look at a sample project on GitHub or the like… |

db.isMaster()

mongo DbName --port 27017 --username asdf --password asdf --authenticationDatabase admin

Db.collection.distinct('NetworkID')

<https://docs.mongodb.com/manual/reference/method/db.collection.insert/>

<https://testlio.com/blog/handy-tips-for-mongodb-shell-queries/>

# db.getSiblingDB()

users = db.getSiblingDB('users')

records = db.getSiblingDB('records')

users.active.count()

users.active.findOne()

records.requests.count()

records.requests.findOne()