

Treinamento DBA MongoDB

Instrutor: Adriano Bonacin



Apresentação

Quem sou

Graduado e mestre em Física, pós banco de dados Oracle

Banco de dados há 17 anos, DBA há 12 anos

OCP DBA e OCP PL/SQL

AWS Assoc (Dev, Archit, SysOps), Specialty (Bigdata, Database)

MongoDB DB Admin Assoc

Datastax Cassandra Admin, Dev, operations on K8S

Recentemente fui nomeado Cassandra Catalyst, MongoDB Comm Creator



Yadax

- Desde 2007
- Bancos de dados relacionais MySQL e Oracle
- NoSQL: Cassandra e MongoDB
- Administração, performance e tuning
- Políticas de backup e recuperação de desastres
- Instalação, patches, upgrades
- Observabilidade e monitoramento com acionamento automático
- Clusters, escalabilidade e alta disponibilidade



Como são os cursos

- Para DBAs de outras tecnologias
- Usamos VirtualBox ou AWS
- Distro populares e recentes
- Pontos básicos de Linux
- Pontos básicos de banco de dados
- Pontos avançados de Linux
- Pontos avançados de banco de dados
- Tudo ao vivo
- Grupo de apoio no WhatsApp
- Materiais são livres e licenciados, download no nosso site.



Licença do Material

Este material está licenciado sob a licença [Attribution-NoDerivatives 4.0 International](https://creativecommons.org/licenses/by-nd/4.0/)
<https://creativecommons.org/licenses/by-nd/4.0/>

Produzido por Adriano Bonacin

email: abonacin@yadax.com.br

Isso significa que:

- Você pode usar e distribuir em qualquer meio e formato, até comercialmente
- Você deve sempre atribuir os créditos ao proprietário
- Você não pode fazer alterações



Agenda

Módulo 1

- **Conceitos básicos NoSQL**
- **Instalando/Gerenciando o Serviço MongoDB**
- **Dados: CRUD**
- **Autenticação/Autorização:
Administrando Roles / Users**
- **Arquitetura MongoDB**
- **ReplicaSet - Alta disponibilidade**
- **Backup/Restore**

Módulo 2

- **ReplicaSet - Falhas Generalizadas**
- **Administrando Queries**
- **Shards - Escalabilidade**
- **Upgrade - ReplicaSet e Shards**

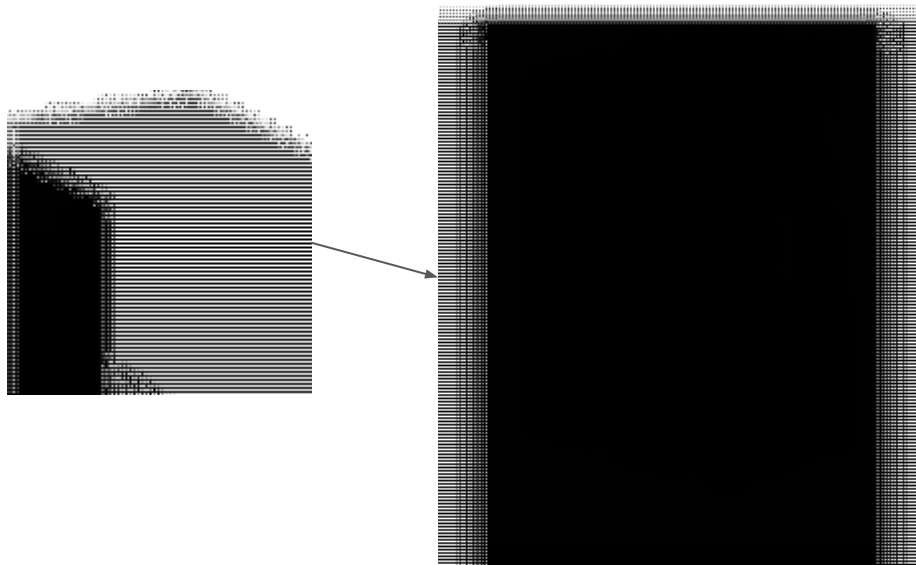


Introdução a banco de dados NoSQL

O que é escalar

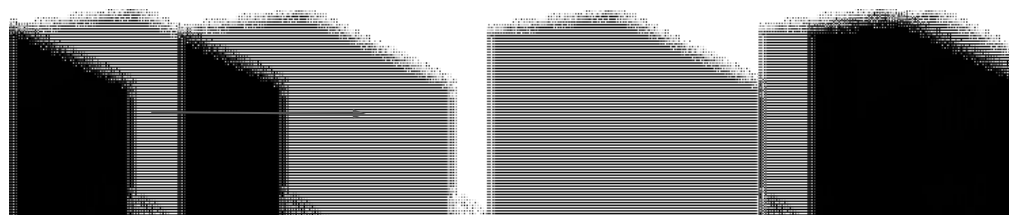
- Escalar Vertical

- Host mais poderoso
- Geralmente mais caro
- Geralmente um aumento significativo



- Escalar horizontal

- Mais hosts
- Geralmente mais barato
- Mais gradual



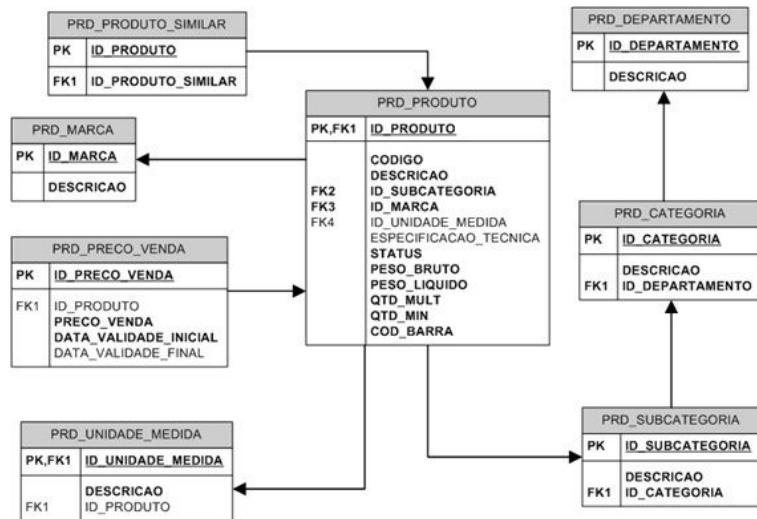
O que é alta disponibilidade

- Capacidade de operar mesmo em caso de falha de algum componente
- Atualmente é um requisito praticamente obrigatório para se adotar uma tecnologia
- A inversão de papéis pode ser manual ou automática. Qual você prefere?



O que é relacionamento/normalização

- Garante integridade dos dados
 - Registro pai deve existir
 - Não é possível excluir pai com filhos
- Dados não duplicados/espalhados
 - Um cadastro de produto
 - Nas outras tabelas: id_produto



Bancos Relacionais

- Por muitos anos foi a única solução
- Atendia praticamente todas demandas
- Escalar geralmente é caro, vertical
- Consistente, ACID
- Normalizado, Joins
- Integridade referencial



PostgreSQL



ACID

- Atomic – Não existe meia transação
- Consistent – Todos veem o DB da mesma forma em dado momento
- Isolation – Privacidade nos registros alterados até o commit.
- Durable – Se o DB respondeu OK para seu commit, está OK mesmo em crash da instance.

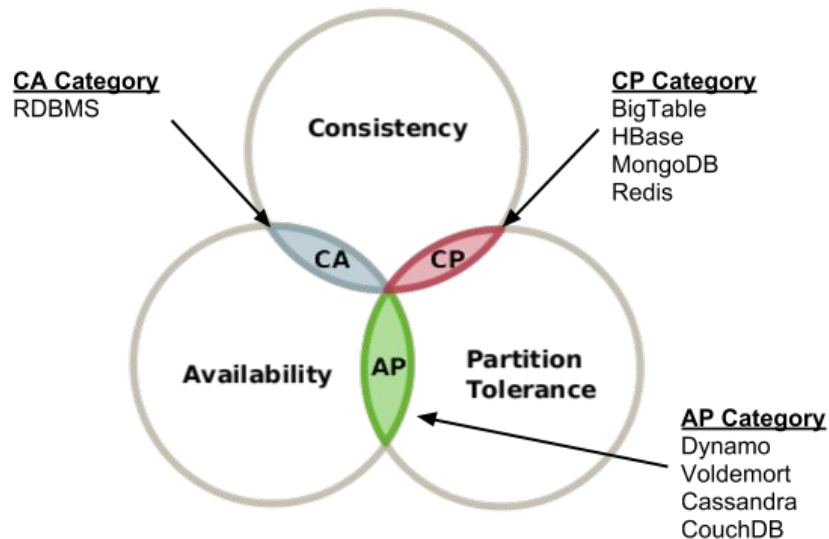


Teorema CAP: Escolha dois

Consistency: Todo mundo vê os dados da mesma forma

Availability: Clientes conseguem ler e escrever mesmo em caso de falha total de um componente

Partition Tolerance: Vai continuar funcionando mesmo que tenha uma falha de comunicação entre os nodes



NoSQL

- Alta disponibilidade
- Escalabilidade
- Distribuídos
- Alto volume de dados
- Schema!?
- Normalização!?
- Relacionamento!?
- Not Only SQL



Por quê NoSQL?

- Volume de dados, TB/host com vários hosts
- Tempo de resposta baixo
- Alta disponibilidade e escalabilidade (cluster) mais simples.



Tipos de NoSQL

Tipos de NoSQL

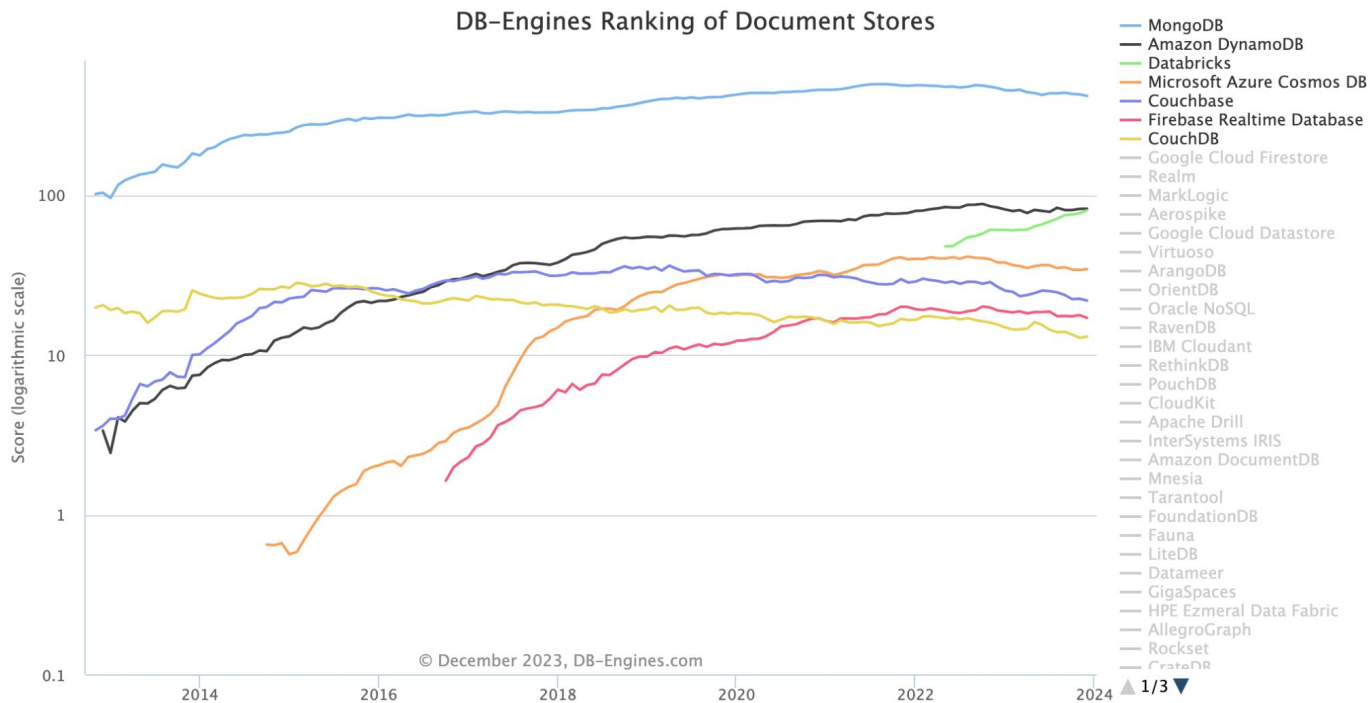


Document

- Armazena docs
 - JSON
 - XML
 - YAML
 - ...
- Permite busca pelos atributos
- Permite doc dentro de doc
- Schema flexível
- Ex.: MongoDB, DynamoDB, CosmosDB

```
{ "name": "Chris" , "age": 23, "city": "New York" }  
{ "name": "Emily" , "age": 19, "city": "Atlanta" }  
{ "name": "Joe" , "age": 32, "city": "New York" }  
{ "name": "Kevin" , "age": 19, "city": "Atlanta" }  
{ "name": "Michelle" , "age": 27, "city": "Los Angeles" }  
{ "name": "Robert" , "age": 45, "city": "Manhattan" }  
{ "name": "Sarah" , "age": 31, "city": "New York" }
```

Document

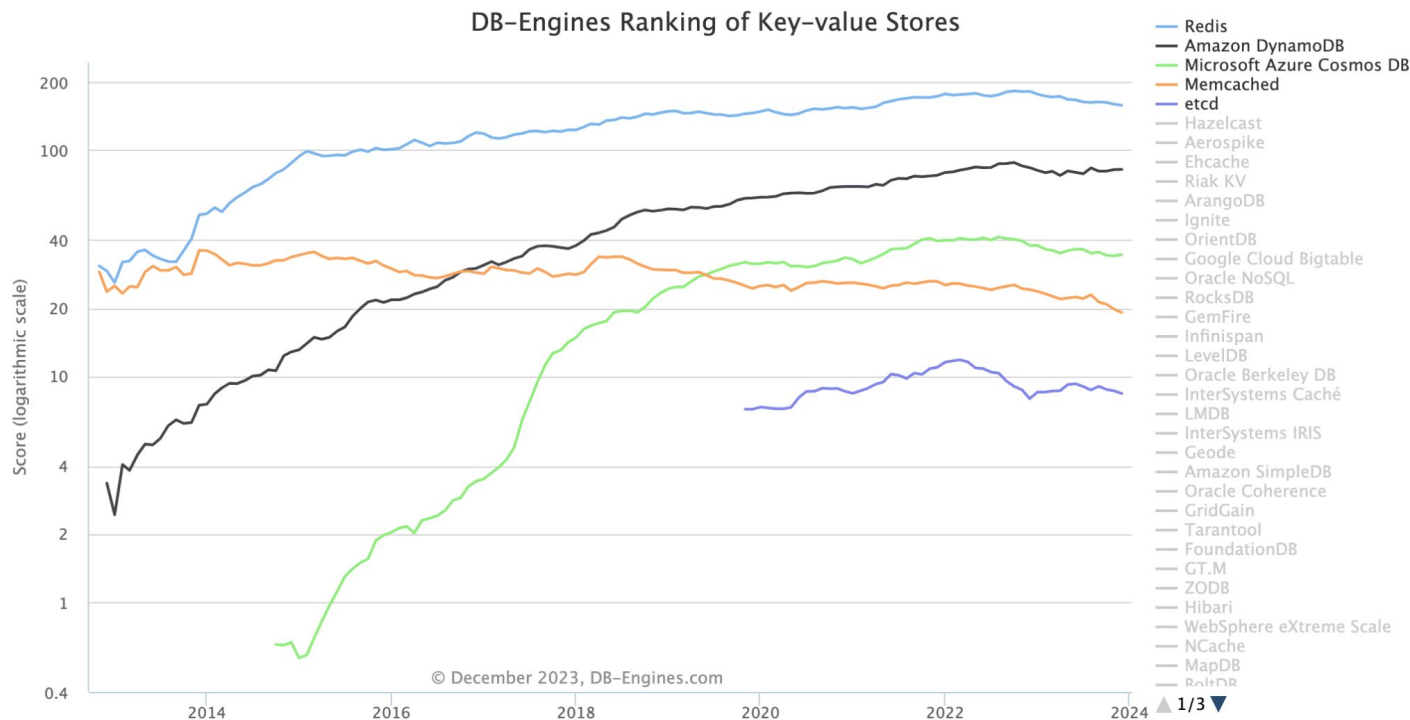


KeyValue

- Modelo mais simples
 - Chave: Valor
- Maior taxa R/W entre os NoSQL
- Geralmente a chave é tipo numérica
- Valor pode assumir uma variedade de datatypes
- Ex.: MemCached, Redis, DynamoDb

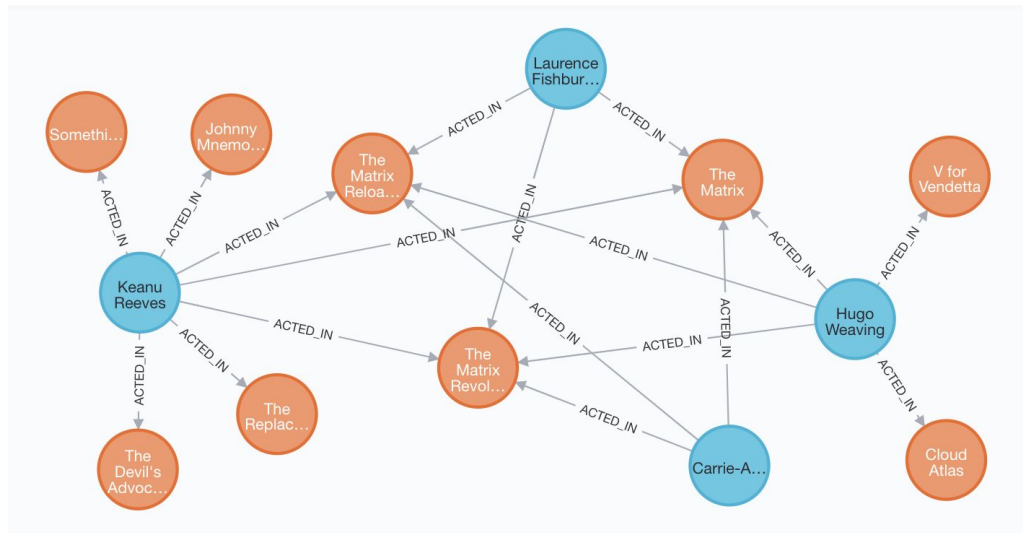
Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Key Value

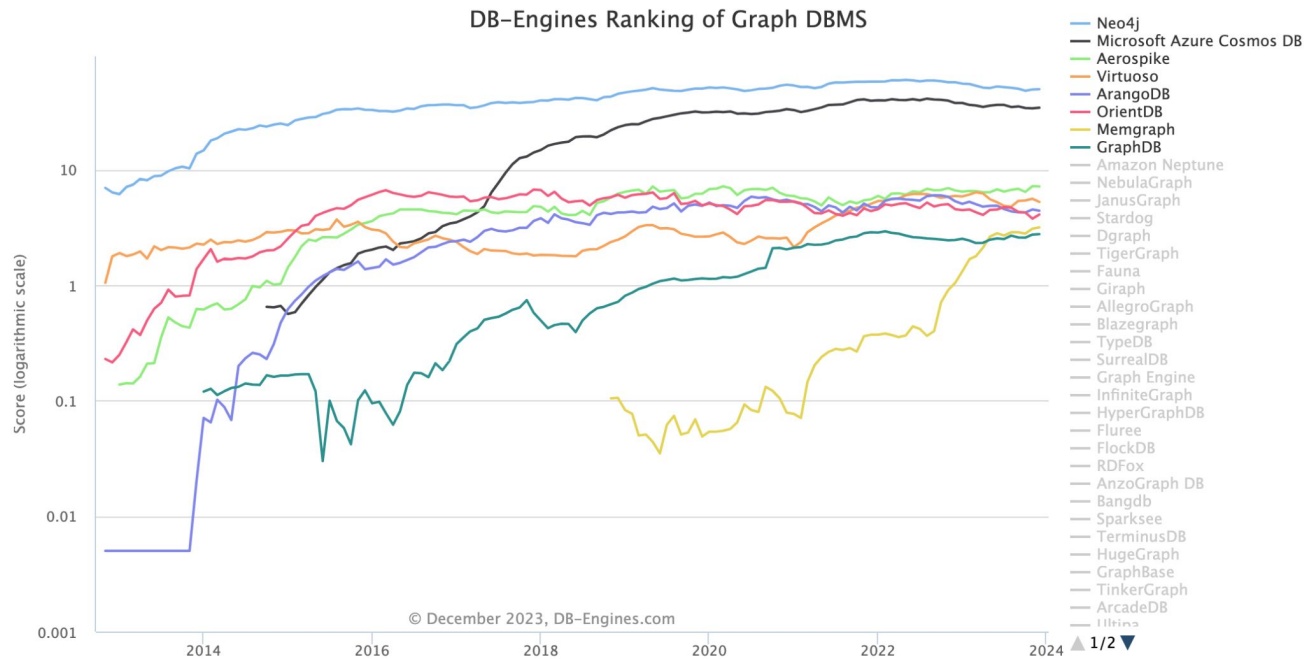


Graph

- Armazenar elementos/relações
- Nós/Vértices: Elementos
- Arestas: Relacionamento
- Atributos: vértices e arestas
- Ex.: Neo4J, CosmosDB



Graph

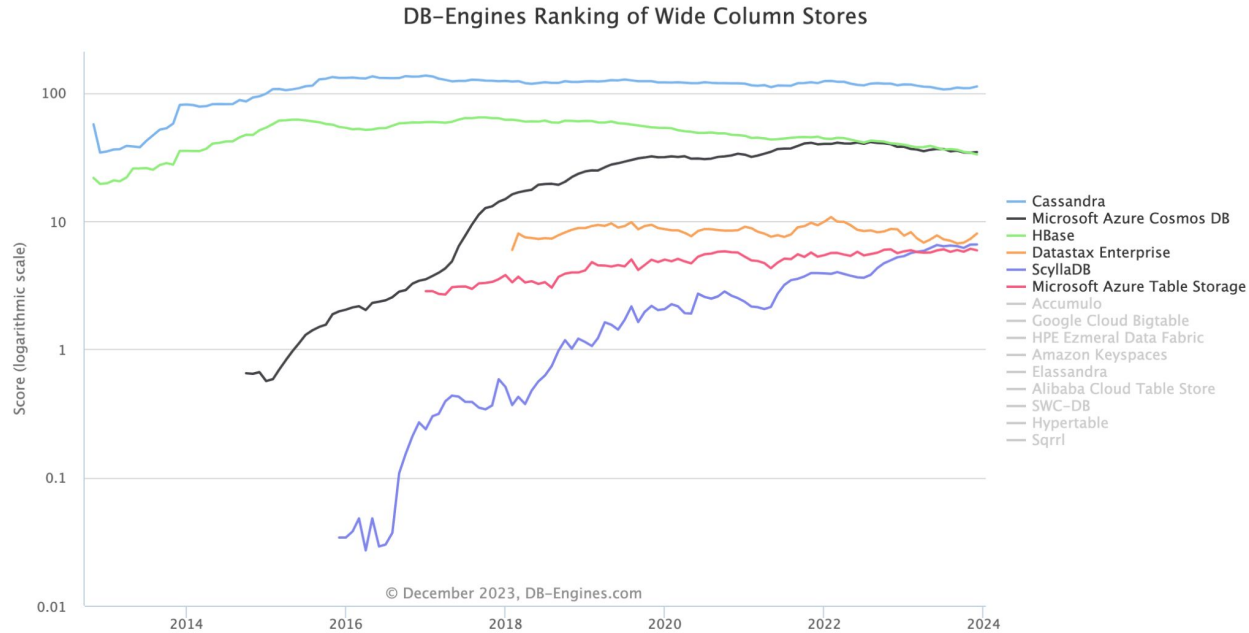


Wide column store / Colunar

- Baseados BigTable do Google
- Para cada linha, um conjunto de colunas (atributos)
- Suporta grande volume de requisições R/W
- VLDB
- Dados replicados
- Ex.: Cassandra, Hbase, Azure CosmosDB



Wide column store / Colunar



Introdução ao MongoDB

Agenda

Introdução ao MongoDB

Instalando/Gerenciando o Serviço MongoDB

Camada de Dados - CRUD

Cursores



MongoDB

- Construído para ser fácil de usar (não de administrar)
- Orientado a documento - JSON
- Não relacional
- Escalável horizontalmente e verticalmente
- Cluster ou não, transparente para a app
- Driver do MongoDB envia requisições para o lugar correto



MongoDB

- Não utiliza “ROW”
- Documentos encadeados e arrays
- Modelo Flexível e ajustável
- Modelo facilmente testável
- Regra geral: Dados que se lê junto, são guardados juntos

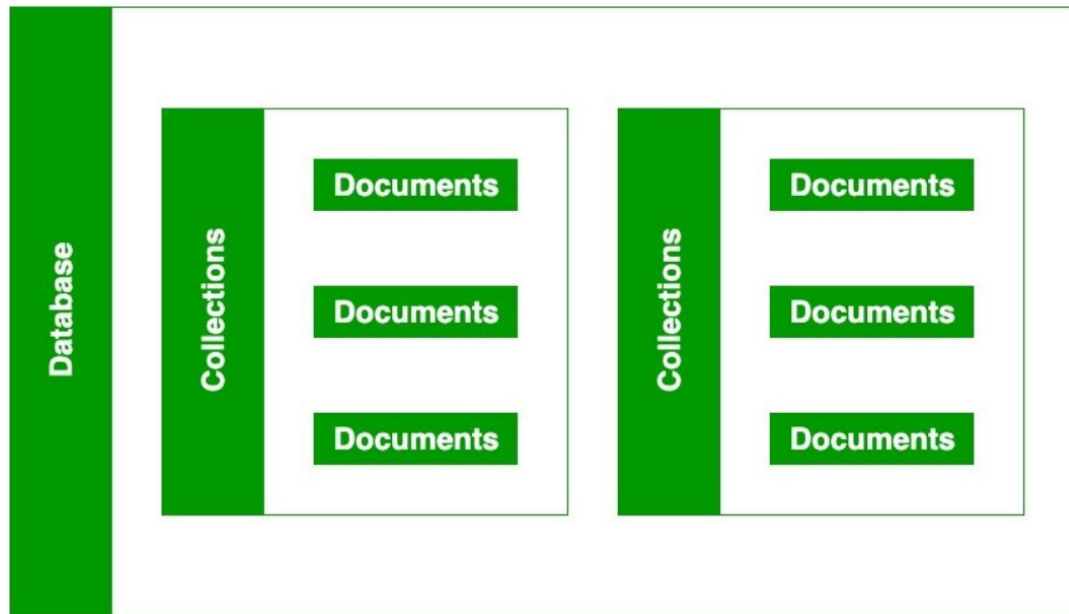
```
1 {  
2   "count": 7,  
3   "items": ["socks", "pants", "shirts", "hats"],  
4   "manufacturer": {  
5     "name": "Molly's Seamstress Shop",  
6     "id": 39233,  
7     "location": {  
8       "address": "123 Pickleton Dr.",  
9       "city": "Tucson",  
10      "state": "AZ",  
11      "zip": 85705  
12    }  
13  },  
14  "total_price": "$393.23",  
15  "purchase_date": "2022-05-30",  
16  "country": "USA"  
17 }
```

MongoDB Features

- Index
 - Normal
 - Unique
 - Compound
 - GeoSpatial
 - Full Text
 - Nested Docs e Arrays
 - Partial
- Aggregation - Pipeline
- TTL (Time to Live)
- Capped Collections (tamanho fixo)
- Joins (com cautela)
- Locks – multi users
- Cache - RAM
- Cust based plans
- Plan Cache
- ReplicaSet
- Shards

Estruturas

- Document
- Collection
- Database



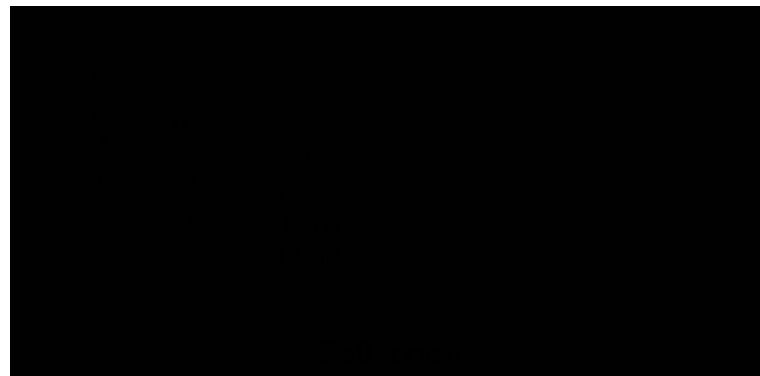
Estruturas – Document JSON

- Menor unidade – Objeto / JSON
- Conjunto de chave (str) /valor
- Equivale a uma linha em um DB relacional
- No MongoDB sempre haverá um field “_id”, único (PK)
- Começa/termina: { ..., ... }
- Campos separados por vírgula

```
{  
  "id": 1,  
  "name": "Foo",  
  "price": 123,  
  "tags": [  
    "Bar",  
    "Eek"  
  ],  
  "stock": {  
    "warehouse": 300,  
    "retail": 20  
  }  
}
```

Estruturas - Collection

- Conjunto de Docs
- Não precisa ser criado explicitamente
- Basta inserir um Doc que a collection é criada automaticamente
- Os docs não precisam ter os mesmos fields, nem os mesmos tipos



Estrutura - Database

- Conjunto de collections
- Isolamento lógico. Ex.:
 - RH
 - Vendas
 - Financeiro



RDBMS x MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field

Restrições para Documents

- Fields são sempre STRING
- No MongoDB, chave e valor são type e case-sensitive
 - `{"views": 3} != {"views": "3"} != {"Views": 3}`
- Não são permitidos chaves(keys) duplicadas
 - `{"Nome": "Adriano", "Nome": "Bonacin" }`
- Tamanho máximo de um documento é 16MB

"chave": valor

Restrições para collections

- Possuem schemas livre
 - {"nome": "Adriano"}
 - {"nome": "Adriano", "idade": 40}
 - {"Nome": "Adriano", "email": "abonacin@yadax.com.br"}
- Mas evite fazer isso, use esse coringa quando REALMENTE for necessário.
- Organização facilita criação de índices
- Facilita a manipulação com tipos definidos e não inferidos.

Restrições para collections

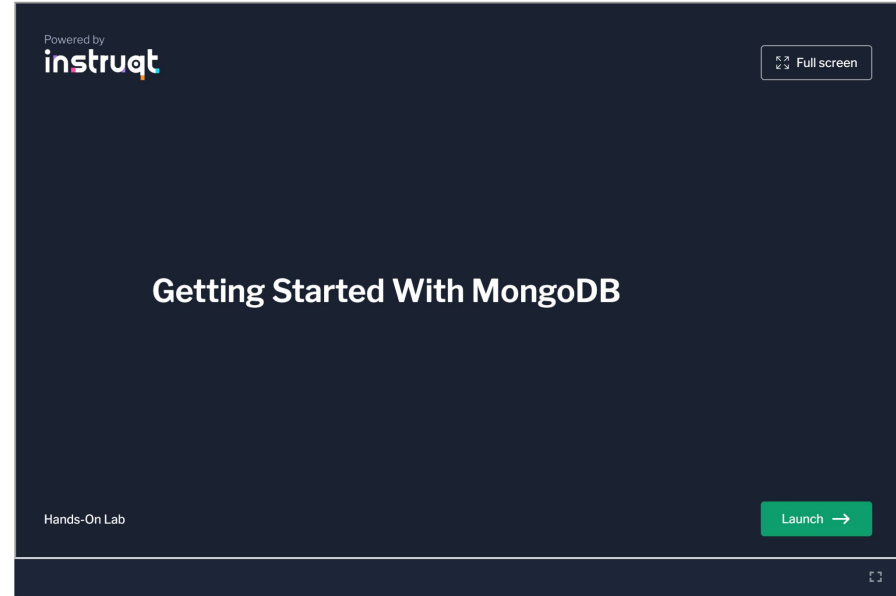
- Nomes das collections são strings UTF8
- Não podem começar com "system."
- Não usar \$
- Pode ser utilizado . (ponto) para separar o nome (subcollection):
 - cliente.endereço
 - cliente.compras
- Ainda assim, no exemplo anterior, as collections não possuem relação a nível do banco de dados, serve apenas para organização

Restrições para databases

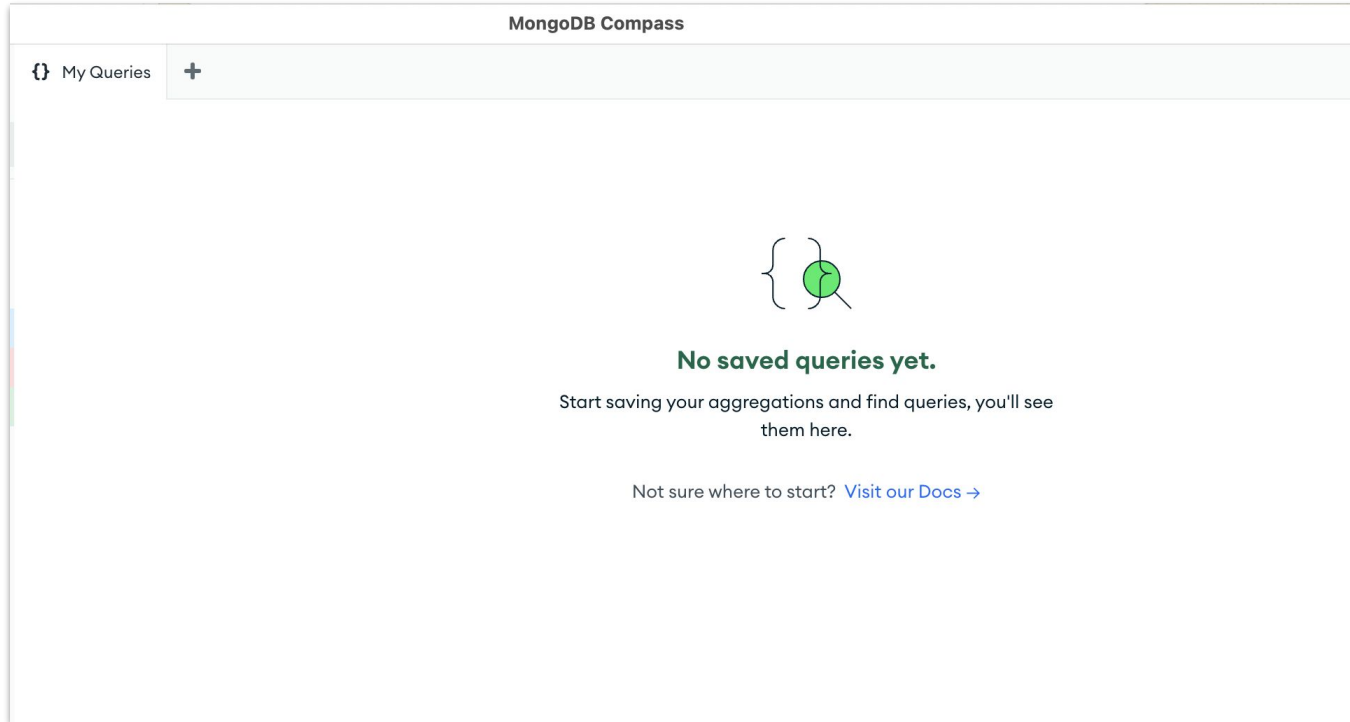
- Uma instance de MongoDB pode ter vários DBs
- Utilize um DB para cada app, a fim de organização
- Os nomes são strings UTF8, mas com restrições:
 - / \ . " * < > : | ? \$ (space)
- Os nomes de DBs são case insensitive
- Max 64 bytes
- DBs reservados:
 - admin, local, config
- `db.collection[.subcollection]` é o fqn = namespace (max 120 B)

Mongo Shell -> Client

- Faz parte do pacote do MongoDB Server
- Interface para administração e manipulação de dados
- Interpretador Javascript
- Roda em Windows, Mac e Linux



MongoDB Compass



Versões do MongoDB

- 2015 (Early) — MongoDB 3.0
- 2015 (Late) — MongoDB 3.2
- 2016 — MongoDB 3.4
- 2017 — MongoDB 3.6
- 2018 — MongoDB 4.0
- 2019 — MongoDB 4.2
- 2020 — MongoDB 4.4
- 2021-2022 — MongoDB 5.0 and Rapid Releases
- 2022 — MongoDB 6.0
- 2023 — MongoDB 7.0
- 2024 — MongoDB 8.0



Edições do MongoDB

MongoDB Community

MongoDB Enterprise

MongoDB Atlas (Serviço na nuvem)

Amazon DocumentDB*

Instalação MongoDB

Instalando o MongoDB

<https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-red-hat/>

<https://www.mongodb.com/docs/manual/administration/install-on-linux/>

Conecte no host

Com Putty ou MobaXterm

```
ssh dba@adriano0.dev.yadax.com.br
```

```
sudo su -
```

```
Pass: Email
```



Instalando o MongoDB

Criar um repo yum.

```
$ sudo su -
```

```
# cat /etc/yum.repos.d/mongodb-org-8.0.repo
```

```
[mongodb-org-8.0]
```

```
name=MongoDB Repository
```

```
baseurl=https://repo.mongodb.org/yum/redhat/8/mongodb-org/8.0/x86_64/
```

```
gpgcheck=1
```

```
enabled=1
```

```
gpgkey=https://pgp.mongodb.com/server-8.0.asc
```



Instalando o MongoDB

Instale o mongodb-org

```
# yum install -y mongodb-org
```

Instalando o MongoDB

Dependencies Resolved

Package	Arch	Version	Repository	Size
Installing:				
mongodb-org	x86_64	5.0.9-1.el7	mongodb-org-5.0	6.1 k
Installing for dependencies:				
cyrus-sasl	x86_64	2.1.26-24.el7_9	updates	88 k
cyrus-sasl-gssapi	x86_64	2.1.26-24.el7_9	updates	41 k
cyrus-sasl-plain	x86_64	2.1.26-24.el7_9	updates	39 k
mongodb-database-tools	x86_64	100.5.3-1	mongodb-org-5.0	50 M
mongodb-mongosh	x86_64	1.5.0-1.el8	mongodb-org-5.0	41 M
mongodb-org-database	x86_64	5.0.9-1.el7	mongodb-org-5.0	6.2 k
mongodb-org-database-tools-extra	x86_64	5.0.9-1.el7	mongodb-org-5.0	11 k
mongodb-org-mongos	x86_64	5.0.9-1.el7	mongodb-org-5.0	20 M
mongodb-org-server	x86_64	5.0.9-1.el7	mongodb-org-5.0	28 M
mongodb-org-shell	x86_64	5.0.9-1.el7	mongodb-org-5.0	15 M
mongodb-org-tools	x86_64	5.0.9-1.el7	mongodb-org-5.0	6.1 k
Updating for dependencies:				
cyrus-sasl-lib	x86_64	2.1.26-24.el7_9	updates	156 k
Transaction Summary				
=====				
Install 1 Package (+11 Dependent packages)				
Upgrade (1 Dependent package)				
Total download size: 154 M				



mongod.service

```
# systemctl status mongod
```

- mongod.service - MongoDB Database Server

```
Loaded: loaded (/usr/lib/systemd/system/mongod.service;  
enabled; vendor preset: disabled)
```

```
Active: inactive (dead)
```

```
Docs: https://docs.mongodb.org/manual
```



mongod.service

```
# cat /usr/lib/systemd/system/mongod.service
[Unit]
Description=MongoDB Database Server
Documentation=https://docs.mongodb.org/manual
After=network-online.target
Wants=network-online.target

[Service]
User=mongod
Group=mongod
Environment="OPTIONS=-f /etc/mongod.conf"
EnvironmentFile=-/etc/sysconfig/mongod
ExecStart=/usr/bin/mongod $OPTIONS
ExecStartPre=/usr/bin/mkdir -p /var/run/mongodb
ExecStartPre=/usr/bin/chown mongod:mongod /var/run/mongodb
ExecStartPre=/usr/bin/chmod 0755 /var/run/mongodb
PermissionsStartOnly=true
PIDFile=/var/run/mongodb/mongod.pid
```



Iniciando o MongoDB

Desabilite o SELINUX

```
# setenforce 0
```

```
# vi /etc/selinux/config
```

```
SELINUX=disabled
```

```
{ "t": { "$date": "2022-06-21T13:10:50.225+00:00" }, "s": "I", "c": "NETWORK", "id": 23015, "ctx": "listener",  
  "msg": "Listening on", "attr": { "address": "/tmp/mongodb-27017.sock" } }  
{ "t": { "$date": "2022-06-21T13:10:50.225+00:00" }, "s": "I", "c": "NETWORK", "id": 23015, "ctx": "listener",  
  "msg": "Listening on", "attr": { "address": "127.0.0.1" } }  
{ "t": { "$date": "2022-06-21T13:10:50.225+00:00" }, "s": "I", "c": "NETWORK", "id": 23016, "ctx": "listener",  
  "msg": "Waiting for connections", "attr": { "port": 27017, "ssl": "off" } }
```



Iniciando o MongoDB

```
# systemctl start mongod
```

```
# cat /etc/mongod.conf
```

```
# tail -f /var/log/mongodb/mongod.log
```

Mongo Shell

- Quando instalamos o mongodb-org, o Mongo Shell vem no pacote (com versão independente)
 - “Linha de comandos” / Tela preta
 - Administra tudo relativo ao MongoDB
 - Fala Javascript
 - Interage com o MongoDB
-
- `mongod >>` processo daemon que gerencia o MongoDB
 - `mongo >>` Mongo Shell (deprecated) - Presente até o 5.0
 - `mongosh >>` Novo Mongo Shell (MongoDB 5+)



Novo Mongo Shell

(em outra janela)

```
[root@ip-20-0-1-55 ~]# mongosh
```

```
Current Mongosh Log ID: 62b1c4f81458822b46f4174a
```

```
Connecting to:
```

```
mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.5.0
```

```
Using MongoDB: 5.0.9
```

```
Using Mongosh: 1.5.0
```

```
...
```

```
test>
```



Mongo Shell

É uma console que fala javascript

```
test> i = 1
```

```
1
```

```
test> i + 1
```

```
2
```

```
test> "Hello, World!".replace ( "World" , "Adriano" );
```

```
Hello, Adriano!
```

```
test>
```



Mongo Shell - Mostrando o DB atual

- Embora funcione com javascript, mongodb é o que interessa.
- A instrução "db" retorna o DB que você está conectado, embora no novo mongosh ele já mostre isso por default.
- Vamos explorar algumas funcionalidades básicas do Mongo Shell

```
test> db
```

```
test
```

```
test>
```

Mongo Shell - Listando todos os DBs

```
test> show dbs;

admin      40.00 KiB
config    108.00 KiB
local      40.00 KiB

test> show databases;

admin      40.00 KiB
config    108.00 KiB
local      40.00 KiB

test>
```

Cadê o DB Test?



Mongo Shell - Acessando um DB específico

Com o comando "use" é possível alterar o DB em uso.

```
test> use admin
```

```
switched to db admin
```

```
admin>
```

Mongo Shell - Listando as collections de um DB

Com o comando "show collections" listamos as collections do DB atual

```
admin> show collections;
```

```
system.version
```

```
admin>
```



Mongo Shell

Mongo Shell tem muitos truques, muita coisa para ser explorada.

Uma pausa no Mongo Shell para começar algumas operações no MongoDB

Em seguida voltaremos explorar mais o Mongo Shell



CRUD

Collections - CRUD

- As collections expõem métodos para manipular dados
 - Create – insert, insertOne, insertMany
 - Read – find, findOne, findAndModify, findOneAndDelete, findOneAndReplace, findOneAndUpdate
 - Update – update, updateOne, updateMany, findOneAndUpdate, findOneAndReplace
 - Delete – deleteOne, deleteMany, findOneAndDelete
- Vamos explorá-los

Mongo Shell - Métodos

```
admin> db.system.version.find() <<< EXECUTA  
[ { _id: 'featureCompatibilityVersion', version: '6.0' } ]
```

```
admin> db.system.version.find <<< DESCRIBE  
[Function: find] AsyncFunction {  
  returnsPromise: true,  
  apiVersions: [ 1, Infinity ],  
  returnType: 'Cursor',  
  ...  
  help: [Function (anonymous)] Help  
}  
admin>
```



Novo database: loja

- O MongoDB é schemaless. Não é necessário criar o DB explicitamente.

```
admin> use loja;
```

```
switched to db loja
```

```
loja> show dbs;
```

```
admin      40.00 KiB
```

```
config    108.00 KiB
```

```
local      40.00 KiB
```

```
loja>
```

Se você criar uma collection, cria-se o DB.



Nova collection: vendas

Se você inserir algum dado, cria-se a collection

```
loja> show collections; << Vazia
```

```
loja> db.vendas.find() << A collection nem existe, mas podemos consultar
```

```
loja>
```

Novo document (+ collection, + DB)

Se você inserir algum dado, cria-se a collection e o DB

```
loja> db.clients.insertOne({"name":"Adriano"})
```

```
{ acknowledged: true,   insertedIds: { '0':  
ObjectId("62b1ccb2ce9e22a741213147")  }}
```

```
loja>
```

```
loja> show collections;
```



Novo document (+ collection, + DB)

```
{ "t": { "$date": "2023-12-06T19:16:00.490+00:00" }, "s": "I", "c": "STORAGE",  
  "id": 20320,  
  "ctx": "conn8", "msg": "createCollection", "attr": { "namespace": "loja.teste", "u  
uidDisposition": "generated", "uuid": { "uuid": { "$uuid": "3a53802d-735e-43b0-8d  
63-a850ef160bb8" } } }, "options": { } } }
```

```
{ "t": { "$date": "2023-12-06T19:16:00.503+00:00" }, "s": "I", "c": "INDEX",  
  "id": 20345, "ctx": "conn8", "msg": "Index build: done  
building", "attr": { "buildUUID": null, "collectionUUID": { "uuid": { "$uuid": "3a53  
802d-735e-43b0-8d63-a850ef160bb8" } } }, "namespace": "loja.teste", "index": "_id_  
", "ident": "index-8--6898233672849508994", "collectionIdent": "collection-7--  
6898233672849508994", "commitTimestamp": null } }
```



Doc versus array

Este é um doc:

```
{ "key": "value" }
```

Este é um array:

```
[ { "key": "value1" }, { "key": "value2" } ]
```

insertOne (doc)

```
loja> db.vendas.insertOne({"cliente":"Bonacin"})  
  
{  
  acknowledged: true,  
  insertedId: ObjectId("62b44b102664d3b97077875b")  
}
```

insertOne - usando variavel (javascript object)

```
loja> minha_venda = {"cliente": "Adriano", "valor": 100,  
"cidade": "São Paulo"}
```

```
{ cliente: 'Adriano', valor: 100, cidade: 'São Paulo' }
```

```
loja> db.vendas.insertOne(minha_venda)
```

```
{  
  acknowledged: true,  
  insertedId: ObjectId("62b44e302664d3b97077875e")  
}
```



insertMany (array)

```
loja> db.vendas.insertMany([ {"cliente":"Raul Seixas"}, {"cliente":"Elvis Presley"} ] )

{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("62b44b462664d3b97077875c"),
    '1': ObjectId("62b44b462664d3b97077875d")
  }
}
```

Doc versus array

Doc: {}

```
{"cliente": "Bonacin"}
```

Array: [{}, {}, ..., {}]

```
[{"cliente": "Raul Seixas"}, {"cliente": "Elvis Presley"}]
```

find

```
loja> db.vendas.find()
[
  { _id: ObjectId("62b44b102664d3b97077875b"), cliente: 'Bonacin' },
  { _id: ObjectId("62b44b462664d3b97077875c"), cliente: 'Raul Seixas' },
  {
    _id: ObjectId("62b44b462664d3b97077875d"),
    cliente: 'Elvis Presley'
  }
]
```

Qual o retorno no método .find()?

É importante para quando formos trabalhar com scripts / Cursores

O retorno do .find() é um Cursor

O retorno do .findOne() é um doc

find

```
loja> db.vendas.find()
```

```
[  
  { _id: ObjectId("62b44b102664d3b97077875b"), cliente: 'Bonacin' },  
  { _id: ObjectId("62b44b462664d3b97077875c"), cliente: 'Raul Seixas' },  
  {  
    _id: ObjectId("62b44b462664d3b97077875d"),  
    cliente: 'Elvis Presley'  
  }  
]
```



findOne

```
loja> db.vendas.findOne()
```

```
{ _id: ObjectId("62b448a42664d3b97077875a"), cliente:  
"Bonacin" }
```

```
loja>
```

ObjectId?

- Todo documento possui obrigatoriamente o campo `_id`
- É primary key, com índice criado no momento da criação da collection
- Você pode usar um identificador externo (como CPF ou CNPJ) como `_id`, mas essa não é uma boa prática

```
loja> db.vendas.find()
```

```
[
```

```
  { _id: ObjectId("62b448a42664d3b97077875a"), cliente: 'Bonacin' },
```

```
  { _id: ObjectId("62b44b102664d3b97077875b"), ...
```



Datatypes

Pausa no CRUD por um tempo e vamos aprofundar nos datatypes

Para que servem datatypes?

- Organização
- Otimização espaço e operações
- Compor info (ano+mes+dia)

Datatypes Básicos

- Null: {"inactive_date": null}
- Boolean: {"is_active": true}
- Number: {"pi": 3.14}
 - Default: 64bits floating
 - NumberInt – 4 bytes
 - NumberLong – 8 bytes
- String: {"nome": "Adriano"}
 - UTF-8
- Date: {agora = new Date(); data = new Date("2011-10-10")}

Datatypes Básicos

- Array: `{frutas : ["uva", "laranja", "banana"]}`
- Docs encadeados:

```
{ "endereco" :  
  { "logradouro" : "Av Paulista",  
    "numero" : "1000" }  
}
```
- ObjectId: `{"_id" : ObjectId("5f36af41b6e4233255a33350")}`
 - Gerado automaticamente, depende do timestamp atual + uma combinação randômica.

Insert informando o _id

- `_id`: identificador único de um documento
- Se não informado, o MongoDB gera automaticamente

```
loja> db.cursos.insertOne({"_id": 0, "nome": "DBA MongoDB"});
```

```
...
```

```
{ acknowledged: true, insertedIds: { '0': 0 } }
```

Insert – PK(_id)

```
yadax> db.cursos.insert({"_id": 0, "nome": "DBA Cassandra"});
```

Uncaught:

MongoBulkWriteError: E11000 duplicate key error ... cursos index: _id_ dup key:

```
{ _id: 0 }
```

```
Result: BulkWriteResult {
```

```
  result: {
```

```
    ok: 1,
```

```
    writeErrors: [
```

```
      WriteError {
```

```
        err: {
```

```
          index: 0,
```

```
          code: 11000,
```

```
          errmsg: 'E11000 duplicate key error ... index: _id_ dup key: { _id: 0
```

```
}',
```



insertOne

```
loja> document = {"_id": 1, "nome": "DBA ClickHouse"}  
  
{ _id: 1, nome: 'DBA ClickHouse' }  
  
loja> db.cursos.insertOne(document)  
  
{ acknowledged: true, insertedId: 1 }  
  
loja>
```

insertMany

```
loja> array_docs = [  
...     {"_id": 2, "nome": "DBA MariaDB"},  
...     {"_id": 3, "nome": "DBA Cassandra"},  
...     {"_id": 4, "nome": "DBA Oracle"}]  
[  
  { _id: 2, nome: 'DBA MariaDB' },  
  { _id: 3, nome: 'DBA Cassandra' },  
  { _id: 4, nome: 'DBA Oracle' }  
]  
loja> db.cursos.insertMany(array_docs)  
{ acknowledged: true, insertedIds: { '0': 2, '1': 3, '2': 4  
} }
```



insertMany

- Mais eficiente
- Evita idas e vindas Client $\longleftrightarrow$ Server
- Múltiplos docs em uma collection
- **Por default, insere na ordem do array**
- Para importar dados de arquivo texto, talvez o mongoimport seja mais eficiente
- Se a cláusula order for false, o MongoDB reorganiza os docs e tentar otimizar a escrita, geralmente performa mais
- A app não pode depender da ordem de inserção

insertMany

```
db.cursos.insertMany([  
    {"_id": 6, "nome": "DBA ScyllaDB"},  
    {"_id": 6, "nome": "DBA DynamoDB"},  
    {"_id": 7, "nome": "DBA Redshift"}  
])
```

insertMany

```
db.cursos.insertMany([  
  {"_id": 6, "nome": "DBA ScyllaDB"},  
  {"_id": 6, "nome": "DBA DynamoDB"},  
  {"_id": 7, "nome": "DBA Redshift"}  
])
```

O QUE HOUE? Quantos registros foram inseridos?



insertMany

```
db.cursos.insertMany([  
  { "_id": 8, "nome": "DBA Postgres" },  
  { "_id": 8, "nome": "DBA MSSQL" },  
  { "_id": 9, "nome": "DBA Redis" }  
], { ordered: false })
```

insertMany

```
db.cursos.insertMany([  
  { "_id": 8, "nome": "DBA Postgres" },  
  { "_id": 8, "nome": "DBA MSSQL" },  
  { "_id": 9, "nome": "DBA Redis" }  
, { ordered: false }])
```

O QUE HOUE? Quantos registros foram inseridos?



Query

- Vamos aprofundar um pouco nas capacidades de query do MongoDB
 - Ranges
 - Operadores
 - Inigualdades
 - Condicionais
 - Cursores
 - ...

Query - find({par1},{par2})

- Find: Método utilizado para buscar documentos
- O primeiro parâmetro determina **quais documentos** serão selecionados
- {} -> default é sem restrições, retorna todos os docs
- db.mycoll.find() -> db.mycoll.find({})

Query - find({par1},{par2})

- Find: Método utilizado para buscar documentos
- O segundo parâmetro determina **quais campos** serão exibidos.
 - 1 -> Exibe
 - 0 -> Oculta
- {} -> default é sem restrições, retorna todos os campos
- db.mycoll.find() -> db.mycoll.find({},{})

Vamos exercitar

Crie dois docs

```
db.clientes.insertMany([  
  {"nome": "Fulano", "logradouro": "Rua Paraná", "numero":  
"123"},  
  {"nome": "Beltrano", "logradouro": "Rua São Paulo",  
"numero": 123}])
```



Vamos exercitar

Consulte o doc com nome Fulano.

```
db.clientes.find({"nome": "Fulano"})  
  
{  
  "_id" : ObjectId("60ec275003140fa2eb0c57e8"),  
  "nome" : "Fulano",  
  "logradouro" : "Rua Paraná",  
  "numero" : "123"  
}
```



Vamos exercitar

Consulte o doc com logradouro igual a Rua São Paulo. Perceba que todos os campos foram exibidos.

```
db.clientes.find({"logradouro": "Rua São Paulo"})
```

```
yadax> db.clientes.find({logradouro: 'Rua São Paulo'})  
[  
  {  
    _id: ObjectId('6571239258c73c12ac2f669a'),  
    nome: 'Beltrano',  
    logradouro: 'Rua São Paulo',  
    numero: 123  
  }  
]
```



Vamos exercitar

Consulte o doc com nome Fulano. Mostre apenas o nome e logradouro.

```
db.clientes.find({"nome": "Fulano"}, {"nome": 1, "logradouro": 1})
```

```
db.clientes.find({"nome": "Fulano"}, {"nome": 1, "logradouro": 0})
```

<< O QUE HOUE?

```
db.clientes.find({"nome": "Fulano"}, {"nome": 1}) << O QUE HOUE?
```

```
db.clientes.find({nome: 'Fulano'}, {"nome": 1, "_id": 0})
```



Vamos exercitar

Consulte o doc com nome Fulano. Mostre apenas o nome e logradouro.

```
db.clientes.find({"nome": "Fulano"}, {"numero": 0})
```

É importante existir um padrão nos fields.

Vamos exercitar

Por default, o `_id` sempre aparece. Use o 0 (zero) para ocultá-lo

```
db.clientes.find({"nome": "Fulano"}, {"_id": 0})
```

Vamos exercitar

Consulte todos docs. Mostre apenas o nome e logradouro.

```
db.clientes.find({}, {"_id":0, "numero":0})
```

Vamos exercitar

O datatype faz diferença

```
db.foo.find({"idade": 30}) <<< NUMERO
```

```
db.foo.find({"idade": "30"}) <<< STRING
```

Vamos exercitar

Perceba a diferença:

```
db.clientes.find({"numero": 123})
```

```
db.clientes.find({"numero": "123"})
```

```
i = 123
```

```
db.clientes.find({"numero": i.toString()})
```

```
j = "123"
```

```
db.clientes.find({"numero": parseInt(j)})
```



Vamos exercitar

```
# cat /root/cars.txt
```

```
use yadax
```

```
db.cars.insertMany([
  {"marca": "HONDA", "modelo": "CIVIC", "ano_fabricacao": 2015, "ano_modelo": 2016, "cor": "Prata", "preco": 60000},
  {"marca": "HONDA", "modelo": "Civic", "ano_fabricacao": 2018, "ano_modelo": 2018, "cor": "Branco", "preco": 80000},
  {"marca": "HONDA", "modelo": "FIT", "ano_fabricacao": 2019, "ano_modelo": 2019, "cor": "Branco", "preco": 50000},
  {"marca": "FIAT", "modelo": "UNO", "ano_fabricacao": 2000, "ano_modelo": 2001, "cor": "Preto", "preco": 10000},
  {"marca": "FIAT", "modelo": "PALIO", "ano_fabricacao": 2006, "ano_modelo": 2007, "cor": "Prata", "preco": 11000},
  {"marca": "FIAT", "modelo": "STRADA", "ano_fabricacao": 2018, "ano_modelo": 2018, "cor": "Branco", "preco": 40000},
  {"marca": "FIAT", "modelo": "TORO", "ano_fabricacao": 2020, "ano_modelo": 2020, "cor": "Preto", "preco": 90000},
  {"marca": "FIAT", "modelo": "TORO", "ano_fabricacao": 2019, "ano_modelo": 2020, "cor": "Prata", "preco": 80000},
  {"marca": "VW", "modelo": "GOL", "ano_fabricacao": 2012, "ano_modelo": 2012, "cor": "Preto", "preco": 20000},
  {"marca": "VW", "modelo": "POLO", "ano_fabricacao": 2020, "ano_modelo": 2020, "cor": "Branco", "preco": 60000},
  {"marca": "VW", "modelo": "GOLF", "ano_fabricacao": 2019, "ano_modelo": 2019, "cor": "Prata", "preco": 90000},
  {"marca": "VW", "modelo": "GOL", "ano_fabricacao": 2003, "ano_modelo": 2003, "cor": "Branco", "preco": 10000},
  {"marca": "FORD", "modelo": "FIESTA", "ano_fabricacao": 2005, "ano_modelo": 2006, "cor": "Prata", "preco": 11000},
  {"marca": "FORD", "modelo": "FIESTA", "ano_fabricacao": 2007, "ano_modelo": 2007, "cor": "Prata"},
  {"marca": "FORD", "modelo": "PAMPA", "ano_fabricacao": 1989, "ano_modelo": 1989, "cor": "Prata", "preco": null}
])
```



Operador AND

Por default, o operador será AND

```
db.cars.find({"marca": "HONDA", "modelo": "FIT"})
```

```
select *  
from cars  
where marca = "HONDA"  
and modelo = "FIT"
```

Operadores: >, >=, <, <=

- No mongodb são respectivamente:
 - \$gt
 - \$gte
 - \$lt
 - \$lte

```
db.cars.find({"field":{"operator": value}})
```

```
db.cars.find({"preco": {"$lt": 20000}}, {"_id":0})
```

Guarde esse padrão

```
db.collection.find({"field":{"operator": value}})
```



Operadores: >, >=, <, <=

Múltiplas condições: AND

```
db.cars.find({"field":{"operator": value}})
```

```
db.cars.find({"preco": {"$gte": 50000, "$lt": 80000}},  
{"_id":0})
```

Operadores: <> (not equal)

Queries podem buscar por diferente ou "ne" – not equal

```
db.cars.find({"field":{"operator": value}})
```

```
db.cars.find({"marca": {"$ne": "HONDA"}})
```

Operadores: IN

Operações com múltiplos valores, podem ser realizadas com \$in, que segue um padrão ligeiramente diferente.

```
db.cars.find({"field": {"$in": [ARRAY]}})
```

```
db.cars.find({"ano_modelo": {"$in": [2018,2019]}}, {"_id":0})
```



Operadores: IN

Operações com múltiplos valores, podem ser realizadas com \$in

```
db.cars.find({"field": {"$in": [1, "UM"]}})
```

O ARRAY não precisa ter todos os elementos do mesmo datatype.

O ARRAY pode ter apenas um elemento.

Operadores: NIN - Not in

O oposto de \$in é **\$nin – NOT IN**

```
db.cars.find({"field": {"$nin": [ARRAY]}})
```

```
db.cars.find({"marca": {"$nin": ["HONDA", "VW"]}}, {"_id": 0})
```



Operadores: OR

Quando precisamos comparar dois campos distintos, precisamos usar o **\$or**

```
db.cars.find({"$or": [{cond1}, {cond2}]})
```

```
db.cars.find({"$or": [{"marca":  
"HONDA"}, {"modelo": "GOLF"}]}, {"_id": 0})
```



Operadores: OR

A condição pode ser um dos operadores que já vimos

```
db.cars.find({"$or": [{cond1}, {cond2}]})
```

```
db.cars.find({"$or": [{"marca": "HONDA"}, {"ano_fabricacao":  
{"$gt": 2018}]}], {"_id": 0})
```



Field NULL

NULL – Diferente de bancos relacionais, o mongodb entende "igual a null"

```
db.cars.find({"preco": null}, {"_id": 0} )
```

Field NULL

Porém, se um campo não existe o mongodb entende que ele é null.

```
db.cars.find({"km_rodados": null}, {"_id": 0} )
```

Operadores: Exists e Equal

Porém, para filtrar um doc que o campo existe e é null, precisamos utilizar o operador \$exists

\$eq -> equal

```
db.cars.find( {"preco": null} , {"_id": 0} )
```

```
db.cars.find( {"preco": {"$eq": null, "$exists": true}} , {"_id": 0})
```



Updates

Como nos bancos relacionais, precisamos "achar" o registro a ser alterado e dizer o que muda.

```
update table_xpto  
set     coluna_x = novo_valor  
where  id        = 10;
```

```
db.collection.updateOne({where doc}, {set})
```



updateOne({where},{set})

```
yadox> db.cars.findOne({"marca": "VW",  
"modelo": "POLO"}, {_id: 0})  
{  
  marca: 'VW',  
  modelo: 'POLO',  
  ano_fabricacao: 2020,  
  ano_modelo: 2020,  
  cor: 'Branco',  
  preco: 60000          <<<<< PRECO vai mudar para 80000  
}
```

updateOne({where},{set})

```
db.cars.updateOne({"marca": "VW", "modelo": "POLO"}, {$set: {"preco": 80000}})

{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

updateOne({where},{set})

```
db.cars.findOne( {"marca": "VW", "modelo": "POLO"}, { _id: 0 } )
{
  marca: 'VW',
  modelo: 'POLO',
  ano_fabricacao: 2020,
  ano_modelo: 2020,
  cor: 'Branco',
  preco: 80000
}
```

updateMany({where},{set})

```
db.cars.find({marca: 'VW'}, {marca:1, modelo:1, _id:0})  
  
[  
  { marca: 'VW', modelo: 'GOL' },  
  { marca: 'VW', modelo: 'POLO' },  
  { marca: 'VW', modelo: 'GOLF' },  
  { marca: 'VW', modelo: 'GOL' }  
]
```

updateMany({where},{set})

```
db.cars.updateMany({marca: 'VW'},{$set: {marca: 'VOLKSWAGEN'}})

{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
```

updateMany({where},{set})

```
yadax> db.cars.find({marca: 'VW'}, {marca:1, modelo:1, _id:0})
```

```
yadax> db.cars.find({marca: 'VOLKSWAGEN'}, {marca:1, modelo:1, _id:0})
```

```
[  
  { marca: 'VOLKSWAGEN', modelo: 'GOL' },  
  { marca: 'VOLKSWAGEN', modelo: 'POLO' },  
  { marca: 'VOLKSWAGEN', modelo: 'GOLF' },  
  { marca: 'VOLKSWAGEN', modelo: 'GOL' }  
]
```



Delete

Para o delete, também precisamos informar quem será apagado.

```
delete table_xpto  
where id = 10;
```

```
db.collection.deleteOne({where})
```

```
db.collection.deleteMany({where})
```



deleteOne ({where})

```
db.cars.find({}, {marca:1, modelo:1, _id:0})
```

```
[
```

```
  { marca: 'HONDA', modelo: 'CIVIC' },
```

```
  { marca: 'HONDA', modelo: 'Civic' },
```

<<<<<<<< VENDIDO

```
  { marca: 'HONDA', modelo: 'FIT' },
```

```
  { marca: 'FIAT', modelo: 'UNO' },
```

```
  { marca: 'FIAT', modelo: 'PALIO' },
```

```
  { marca: 'FIAT', modelo: 'STRADA' },
```

```
  { marca: 'FIAT', modelo: 'TORO' },
```

```
  { marca: 'FIAT', modelo: 'TORO' },
```

```
  { marca: 'VOLKSWAGEN', modelo: 'GOL' },
```

```
  { marca: 'VOLKSWAGEN', modelo: 'POLO' },
```

```
  { marca: 'VOLKSWAGEN', modelo: 'GOLF' },
```

```
  { marca: 'VOLKSWAGEN', modelo: 'GOL' },
```

```
  { marca: 'FORD', modelo: 'FIESTA' },
```

```
  { marca: 'FORD', modelo: 'FIESTA' },
```

```
  { marca: 'FORD', modelo: 'PAMPA' }
```

```
]
```



deleteOne ({where})

```
db.cars.deleteOne()
```

MongoshInvalidInputError: [COMMON-10001] Missing required argument at position 0 (Collection.deleteOne)

Obrigatório informar o where, mesmo que não tenha restrição alguma.

```
db.cars.deleteOne({})
```

```
{ acknowledged: true, deletedCount: 1 }
```



deleteOne ({where})

```
db.cars.deleteOne({marca: 'HONDA', modelo: 'Civic'})  
{ acknowledged: true, deletedCount: 1 }
```

Resultado

```
db.cars.find({}, {marca:1, modelo:1, _id:0})  
[  
  { marca: 'HONDA', modelo: 'FIT' },  
  { marca: 'FIAT', modelo: 'UNO' },  
  { marca: 'FIAT', modelo: 'PALIO' },  
  { marca: 'FIAT', modelo: 'STRADA' },  
  { marca: 'FIAT', modelo: 'TORO' },  
  { marca: 'FIAT', modelo: 'TORO' },  
  { marca: 'VOLKSWAGEN', modelo: 'GOL' },  
  { marca: 'VOLKSWAGEN', modelo: 'POLO' },  
  { marca: 'VOLKSWAGEN', modelo: 'GOLF' },  
  { marca: 'VOLKSWAGEN', modelo: 'GOL' },  
  { marca: 'FORD', modelo: 'FIESTA' },  
  { marca: 'FORD', modelo: 'FIESTA' },  
  { marca: 'FORD', modelo: 'PAMPA' }  
]
```

deleteMany ({where})

```
db.cars.deleteMany({marca: 'FORD'})  
{ acknowledged: true, deletedCount: 3 }
```

```
db.cars.find({}, {marca:1,modelo:1,_id:0})  
[  
  { marca: 'HONDA', modelo: 'FIT' },  
  { marca: 'FIAT', modelo: 'UNO' },  
  { marca: 'FIAT', modelo: 'PALIO' },  
  { marca: 'FIAT', modelo: 'STRADA' },  
  { marca: 'FIAT', modelo: 'TORO' },  
  { marca: 'FIAT', modelo: 'TORO' },  
  { marca: 'VOLKSWAGEN', modelo: 'GOL' },  
  { marca: 'VOLKSWAGEN', modelo: 'POLO' },  
  { marca: 'VOLKSWAGEN', modelo: 'GOLF' },  
  { marca: 'VOLKSWAGEN', modelo: 'GOL' }  
]
```



deleteMany ({where})

```
db.cars.deleteMany({})      <<<<< BOM E VELHO DELETE SEM WHERE  
  
{ acknowledged: true, deletedCount: 10 }
```

Vamos exercitar

```
# cat /root/cars.txt
use yadax
db.cars.insertMany([
  {"marca": "HONDA", "modelo": "CIVIC", "ano_fabricacao": 2015, "ano_modelo": 2016, "cor": "Prata", "preco": 60000},
  {"marca": "HONDA", "modelo": "Civic", "ano_fabricacao": 2018, "ano_modelo": 2018, "cor": "Branco", "preco": 80000},
  {"marca": "HONDA", "modelo": "FIT", "ano_fabricacao": 2019, "ano_modelo": 2019, "cor": "Branco", "preco": 50000},
  {"marca": "FIAT", "modelo": "UNO", "ano_fabricacao": 2000, "ano_modelo": 2001, "cor": "Preto", "preco": 10000},
  {"marca": "FIAT", "modelo": "PALIO", "ano_fabricacao": 2006, "ano_modelo": 2007, "cor": "Prata", "preco": 11000},
  {"marca": "FIAT", "modelo": "STRADA", "ano_fabricacao": 2018, "ano_modelo": 2018, "cor": "Branco", "preco": 40000},
  {"marca": "FIAT", "modelo": "TORO", "ano_fabricacao": 2020, "ano_modelo": 2020, "cor": "Preto", "preco": 90000},
  {"marca": "FIAT", "modelo": "TORO", "ano_fabricacao": 2019, "ano_modelo": 2020, "cor": "Prata", "preco": 80000},
  {"marca": "VW", "modelo": "GOL", "ano_fabricacao": 2012, "ano_modelo": 2012, "cor": "Preto", "preco": 20000},
  {"marca": "VW", "modelo": "POLO", "ano_fabricacao": 2020, "ano_modelo": 2020, "cor": "Branco", "preco": 60000},
  {"marca": "VW", "modelo": "GOLF", "ano_fabricacao": 2019, "ano_modelo": 2019, "cor": "Prata", "preco": 90000},
  {"marca": "VW", "modelo": "GOL", "ano_fabricacao": 2003, "ano_modelo": 2003, "cor": "Branco", "preco": 10000},
  {"marca": "FORD", "modelo": "FIESTA", "ano_fabricacao": 2005, "ano_modelo": 2006, "cor": "Prata", "preco": 11000},
  {"marca": "FORD", "modelo": "FIESTA", "ano_fabricacao": 2007, "ano_modelo": 2007, "cor": "Prata"},
  {"marca": "FORD", "modelo": "PAMPA", "ano_fabricacao": 1989, "ano_modelo": 1989, "cor": "Prata", "preco": null}
])
```



Vamos exercitar

- Liste os carros da FIAT
- Liste os carros da HONDA fabricados antes de 2017
- Liste os carros fabricados em 2015, 2018 ou 2020
- Liste os carros fabricados em 2015, 2016 ou 2020, mais caros que 70000
- Altere Civic para CIVIC
- Liste todos os modelos CIVIC ou marca VW
- Delete o FIAT STRADA
- Delete todos os FORDS
- Delete todos os registros

Cursores

Vamos exercitar

```
# cat /root/cars.txt
use yadax
db.cars.insertMany([
{"marca":"HONDA", "modelo":"CIVIC", "ano_fabricacao": 2015 , "ano_modelo": 2016, "cor": "Prata", "preco": 60000},
{"marca":"HONDA", "modelo":"Civic", "ano_fabricacao": 2018 , "ano_modelo": 2018, "cor": "Branco", "preco": 80000},
{"marca":"HONDA", "modelo":"FIT", "ano_fabricacao": 2019 , "ano_modelo": 2019, "cor": "Branco", "preco": 50000},
{"marca":"FIAT", "modelo":"UNO", "ano_fabricacao": 2000 , "ano_modelo": 2001, "cor": "Preto", "preco": 10000},
{"marca":"FIAT", "modelo":"PALIO", "ano_fabricacao": 2006 , "ano_modelo": 2007, "cor": "Prata", "preco": 11000},
{"marca":"FIAT", "modelo":"STRADA", "ano_fabricacao": 2018 , "ano_modelo": 2018, "cor": "Branco", "preco": 40000},
{"marca":"FIAT", "modelo":"TORO", "ano_fabricacao": 2020 , "ano_modelo": 2020, "cor": "Preto", "preco": 90000},
{"marca":"FIAT", "modelo":"TORO", "ano_fabricacao": 2019 , "ano_modelo": 2020, "cor": "Prata", "preco": 80000},
{"marca":"VW", "modelo":"GOL", "ano_fabricacao": 2012 , "ano_modelo": 2012, "cor": "Preto", "preco": 20000},
{"marca":"VW", "modelo":"POLO", "ano_fabricacao": 2020 , "ano_modelo": 2020, "cor": "Branco", "preco": 60000},
{"marca":"VW", "modelo":"GOLF", "ano_fabricacao": 2019 , "ano_modelo": 2019, "cor": "Prata", "preco": 90000},
{"marca":"VW", "modelo":"GOL", "ano_fabricacao": 2003 , "ano_modelo": 2003, "cor": "Branco", "preco": 10000},
{"marca":"FORD", "modelo":"FIESTA", "ano_fabricacao": 2005 , "ano_modelo": 2006, "cor": "Prata", "preco": 11000},
{"marca":"FORD", "modelo":"FIESTA", "ano_fabricacao": 2007 , "ano_modelo": 2007, "cor": "Prata"},
{"marca":"FORD", "modelo":"PAMPA", "ano_fabricacao": 1989, "ano_modelo": 1989, "cor": "Prata", "preco": null}
])
```



Cursores - WHILE

hasNext -> verifica se tem mais docs

next -> traz o próximo doc

```
var cCarros = db.cars.find()
while (cCarros.hasNext()) {
    x = cCarros.next();
    print(x.marca + "/" + x.modelo + " por: " + x.preco)
}
```



Cursores - FOR EACH

Aqui usamos o conceito de função anônima, do javascript. É uma função que só existe em um escopo específico e nem precisa de nome

x vai receber o valor da linha.

```
var cCarros = db.cars.find()

cCarros.forEach(function(x) {

    //print(x)

    print(x.marca + "/" + x.modelo + " por: " + x.preco)

})
```



Cursorres

- Se você tentar executar duas vezes uma operação sobre o mesmo cursor, verá que a segunda vez não traz registros.
 - Por que?
- Outro ponto importante é que enquanto você não pedir explicitamente por um registro, a query não é enviada ao DB.
- Somente quando pedimos `.hasNext()`, `.next()` ou algo assim, é que a query será submetida ao DB

Mongod - Arquivo de configuração

Doc

<https://www.mongodb.com/docs/manual/reference/configuration-options/>

Mongod

- (Re) Starte o serviço do mongo:

```
# cat /etc/mongod.conf
```

```
# systemctl restart mongod
```

Mongod - conf

where to write logging data.

systemLog:

destination: file

logAppend: true

path: /var/log/mongodb/mongod.log

```
systemLog:
  verbosity: <int>
  quiet: <boolean>
  traceAllExceptions: <boolean>
  syslogFacility: <string>
  path: <string>
  logAppend: <boolean>
  logRotate: <string>
  destination: <string>
  timeStampFormat: <string>
  component:
    accessControl:
      verbosity: <int>
    command:
      verbosity: <int>
```

O log vai ser uma das coisas mais importante na hora do troubleshooting



Mongod - conf

Where and how to store data.

storage:

dbPath: /var/lib/mongo

journal:

enabled: true

directoryPerDB: true <<<< Não default

Normalmente temos um filesystem dedicado para o dbPath.

```
storage:
  dbPath: <string>
  journal:
    commitIntervalMs: <num>
  directoryPerDB: <boolean>
  syncPeriodSecs: <int>
  engine: <string>
  wiredTiger:
    engineConfig:
      cacheSizeGB: <number>
      journalCompressor: <string>
      directoryForIndexes: <boolean>
      maxCacheOverflowFileSizeGB: <number>
    collectionConfig:
      blockCompressor: <string>
    indexConfig:
      prefixCompression: <boolean>
  inMemory:
    engineConfig:
      inMemorySizeGB: <number>
    oplogMinRetentionHours: <double>
```



Mongod - conf

how the process runs

processManagement:

fork: true # fork and run in background

pidFilePath: /var/run/mongodb/mongod.pid

timezoneInfo: /usr/share/zoneinfo

```
processManagement:
  fork: <boolean>
  pidFilePath: <string>
  timeZoneInfo: <string>
```



Mongod - conf

network interfaces

net:

port: 27017

bindIp: 127.0.0.1

127.0.0.1 -> Somente conexões locais (**lo**)

172.10.10.NN -> Somente pela interface **eth0**

0.0.0.0 -> Ambos, loopback e **eth0**

```
net:
  port: <int>
  bindIp: <string>
  bindIpAll: <boolean>
  maxIncomingConnections: <int>
  wireObjectCheck: <boolean>
  ipv6: <boolean>
  unixDomainSocket:
    enabled: <boolean>
    pathPrefix: <string>
    filePermissions: <int>
  tls:
    certificateSelector: <string>
    clusterCertificateSelector: <string>
    mode: <string>
    certificateKeyFile: <string>
    certificateKeyFilePassword: <string>
    clusterFile: <string>
    clusterPassword: <string>
    CAFile: <string>
    clusterCAFile: <string>
```



Autenticação/Autorização

Autenticação / Autorização

- Autenticação: Garantir que o user é quem ele está dizendo que é.
 - Salted Challenge Response Authentication Mechanism (SCRAM) - User/Pass
 - x.509 Certificate Authentication
 - Kerberos Authentication
 - LDAP Proxy Authentication
- Autorização: Garantir que o user faz o que ele tem permissão para fazer
 - Role-Based Access Control
 - Built-In Roles
 - User-Defined Roles
 - LDAP Authorization

Criar usuário - DBA (root)

```
yadax> use admin
switched to db admin
admin> db.createUser(
{
  user: "dba",
  pwd: passwordPrompt(),
  roles: [
    { role: "root", db: "admin" }
  ]
}
```

```
Enter password
*****{ ok: 1 }
```



Habilitar Autenticação/Autorização

```
vi /etc/mongod.conf
```

```
...
```

```
security:
```

```
    authorization: enabled
```

```
security:
  keyFile: <string>
  clusterAuthMode: <string>
  authorization: <string>
  transitionToAuth: <boolean>
  javascriptEnabled: <boolean>
  redactClientLogData: <boolean>
  clusterIpSourceAllowlist:
    - <string>
  sasl:
    hostname: <string>
    serviceName: <string>
    saslauthdSocketPath: <string>
  enableEncryption: <boolean>
  encryptionCipherMode: <string>
  encryptionKeyFile: <string>
  kmip:
    keyIdentifier: <string>
    rotateMasterKey: <boolean>
```



Habilitar Autenticação/Autorização

Restarte o mongod

```
[root@ip-20-0-1-35 ~]# systemctl status mongod
```

- mongod.service - MongoDB Database Server

```
    Loaded: loaded (/usr/lib/systemd/system/mongod.service; enabled; vendor preset: disabled)
```

```
... mongod -f /etc/mongod.conf
```

```
... Started MongoDB Database Server.
```

```
[root@ip-20-0-1-35 ~]# systemctl stop mongod
```

```
[root@ip-20-0-1-35 ~]# systemctl start mongod
```



Conecte no mongosh

```
[root@ip-20-0-1-35 ~]# mongosh
```

```
...
```

```
Using MongoDB:      5.0.9
```

```
Using Mongosh:      1.5.0
```

```
For mongosh info see: https://docs.mongodb.com/mongodb-shell/
```

```
test> show dbs;
```

```
MongoServerError: command listDatabases requires authentication
```

```
test>
```



Conecte no mongosh

```
test> db.auth("dba","mudar123")  
MongoServerError: Authentication failed.
```

```
test> use admin  
switched to db admin
```

```
admin> db.auth("dba","mudar123")  
{ ok: 1 }
```

```
admin> show dbs;  
admin      132.00 KiB  
config     36.00 KiB  
local      72.00 KiB  
yadax      48.00 KiB
```



Localhost Exception

Se habilitar a autorização antes de criar um user no DB, você poderá fazer isso apenas uma vez conectando no localhost.

Connections using the localhost exception have access to create *only* the **first user or role**.



Conecte no mongosh

```
--file [arg]  
--host [arg]  
--port [arg]  
--username [arg]  
--password [arg]  
--authenticationDatabase [arg]
```

DB Address Examples:

```
foo  
192.168.0.5/foo  
192.168.0.5:9999/foo  
mongodb://192.168.0.5:9999/foo
```



Conecte no mongosh

```
[root@ip-20-0-1-35 ~]# mongosh --username userAdmin --password mudar123
Current Mongosh Log ID: 62b74950ed8f33a6eb8d1d41
Connecting to:
mongodb://<credentials>@127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=200
0&appName=mongosh+1.5.0
Using MongoDB:      5.0.9
Using Mongosh:      1.5.0
```

For mongosh info see: <https://docs.mongodb.com/mongodbs-shell/>

```
test> show dbs
admin      132.00 KiB
config     36.00 KiB
local      72.00 KiB
yadax      48.00 KiB
```



Vamos criar o DB yadax

```
test> use yadax;
switched to db yadax
yadax> db.cars.insertMany([
... {"marca":"HONDA", "modelo":"CITY", "ano_fabricacao": 2018 , "ano_modelo":
2018, "cor": "Prata", "preco": 71000},
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("62b75194941607ddb9f0a68"),...
```



Vamos criar o DB yadax2

```
test> use yadax2;
switched to db yadax2
yadax2> db.cars.insertMany([
... {"marca":"HONDA", "modelo":"CITY", "ano_fabricacao": 2018 , "ano_modelo":
2018, "cor": "Prata", "preco": 71000},
...
])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("62b75194941607ddb9f0a68"),...
```



Criar userApp1

```
admin> use admin
already on db admin
admin> db.createUser(
...   {
.....   user: "userApp1",
.....   pwd: 'dba2001',
.....   roles: [
.....     { role: "read", db: "yadax" },
.....   ]
..... }
... )
Enter password
*****{ ok: 1 }
```



Criar userApp2

```
admin> db.createUser(  
...  {  
.....    user: "userApp2",  
.....    pwd: 'dba2001',  
.....    roles: [  
.....        { role: "read", db: "yadax2" },  
.....    ]  
..... }  
... )  
Enter password  
*****{ ok: 1 }  
admin>
```



Consulta yadax / userApp1

```
[root@ip-20-0-1-35 ~]# mongosh --username userApp1 --password dba2001
```

```
...
```

```
test> use yadax;
```

```
switched to db yadax
```

```
yadax> db.cars.find({}, {_id:0,marca:1,modelo:1})
```

```
[
```

```
  { marca: 'HONDA', modelo: 'CIVIC' },
```

```
  { marca: 'HONDA', modelo: 'Civic' },
```

```
  { marca: 'HONDA', modelo: 'FIT' },
```

```
...
```



Consulta yadax2 / userApp1

```
yadax> use yadax2;  
switched to db yadax2
```

```
yadax2> db.cars.find({}, {_id:0,marca:1,modelo:1})
```

```
MongoServerError: not authorized on yadax2 to execute  
command { find: "cars", filter: {}, projection: { _id: 0,  
marca: 1, modelo: 1 }, lsid: { id:  
UUID("7d0a5372-af8a-4543-b9cd-cdcd365236a6") }, $db:  
"yadax2" }
```



Consulta yadax* / userApp2

Repita o mesmo para o userApp2

```
test> use yadax2;  
switched to db yadax2
```

```
yadax2> db.cars.find({}, {_id:0,marca:1,modelo:1})
```

O que houve?



Roles

- Conjunto de privilégios
- Built-in
 - Prontas para o uso
 - Normalmente é suficiente
 - read, readWrite, dbOwner, admin, userAdminAnyDatabase
- User defined
 - Necessidades específicas, mais restritivas

Roles Built-in

Conecte com o userAdmin

```
admin> db.runCommand({ rolesInfo: 1, showBuiltinRoles: true })
```

```
admin> db.getRoles(  
  {  
    rolesInfo: 1,  
    showPrivileges:false,  
    showBuiltinRoles: true  
  }  
)
```



Roles Built-in

```
db.runCommand({ rolesInfo: 1, showPrivileges:true  
,showBuiltinRoles: true })
```

```
db.getRoles(  
  {  
    rolesInfo: 1,  
    showPrivileges:true,  
    showBuiltinRoles: true  
  }  
)
```

read

```
'changeStream',  
'collStats',  
'dbHash',  
'dbStats',  
'find',  
'killCursors',  
'listCollections',  
'listIndexes',  
'planCacheRead'
```

User defined roles - create role

```
use admin
db.createRole(
  {
    role: "readYadaxCars",
    privileges: [
      { resource: { db: "yadax", collection: "cars" }, actions: [ "find" ] }
    ],
    roles: []
  }
)
```



User defined roles - create user

```
db.createUser(  
  {  
    user: "userCars",  
    pwd: passwordPrompt(),  
    roles: [  
      { role: "readYadaxCars", db: "admin" },  
    ]  
  }  
)
```

db -> DB onde a role está armazenada



User defined roles - find cars

```
[root@ip-20-0-1-35 ~]# mongosh --username userCars --password mudar123
```

```
...
```

```
test> use yadax;
```

```
switched to db yadax
```

```
yadax> db.cars.find({}, {_id:0,marca:1,modelo:1})
```

```
[
```

```
  { marca: 'HONDA', modelo: 'CIVIC' },
```

```
  { marca: 'HONDA', modelo: 'Civic' },
```

```
  { marca: 'HONDA', modelo: 'FIT' },
```

```
  { marca: 'FIAT', modelo: 'UNO' },
```

```
...
```



User defined roles - outra collection

```
yadox> db.cars2.find({}, {_id:0,marca:1,modelo:1})
```

```
MongoServerError: not authorized on yadox to execute command { find:
"cars2", filter: {}, projection: { _id: 0, marca: 1, modelo: 1 }, lsid: {
id: UUID("d71b48e9-cb6b-45df-a462-64882c6a4b57") }, $db: "yadox" }
```

```
yadox> db.cars.insertOne({nome: 'Adriano'})
```

```
MongoServerError: not authorized on yadox to execute command { insert:
"cars", documents: [ { nome: "Adriano", _id:
ObjectId('62b75f3b93ad41227a411a29') } ], ordered: true, lsid: { id:
UUID("d71b48e9-cb6b-45df-a462-64882c6a4b57") }, $db: "yadox" }
```



Listar roles

```
db.auth('userAdmin','mudar123')  
  
db.getRoles(  
  {  
    rolesInfo: 1,  
    showPrivileges: false,  
    showBuiltinRoles: false  
  }  
)
```

Listar uma role

```
db.getRole('readYadaxCars')    <<< AQUI NÃO É UM DOC

{
  _id: 'admin.readYadaxCars',
  role: 'readYadaxCars',
  db: 'admin',
  roles: [],
...
}
```



Listar uma role e seus privilégios

```
admin> db.getRole('readYadaxCars',{showPrivileges: true})
```

```
{
  _id: 'admin.readYadaxCars',
  role: 'readYadaxCars',
  db: 'admin',
  privileges: [
    {
      resource: { db: 'yadax', collection: 'cars' },
      actions: [ 'find' ]
    }
  ],
  ...
}
```



Listar users

```
db.getUsers()  
{  
  users: [  
    {  
      _id: 'admin.userAdmin',  
      userId: UUID("b465661e-476d-4303-b541-1f56e7cdaa0c"),  
      user: 'userAdmin',  
      db: 'admin',  
      roles: [  
        { role: 'userAdminAnyDatabase', db: 'admin' },  
        { role: 'readWriteAnyDatabase', db: 'admin' }  
      ],  
      mechanisms: [ 'SCRAM-SHA-1', 'SCRAM-SHA-256' ]  
    }, ...
```



Listar user específico

```
admin> db.getUser('userCars')
{
  _id: 'admin.userCars',
  userId: UUID("dd927559-da8a-4699-8648-ed53964bb318"),
  user: 'userCars',
  db: 'admin',
  roles: [ { role: 'readYadaxCars', db: 'admin' } ],
  mechanisms: [ 'SCRAM-SHA-1', 'SCRAM-SHA-256' ]
}
```



Listar user específico

```
admin> db.getUser('userCars', {showPrivileges: true})
{
  _id: 'admin.userCars',
  userId: UUID("dd927559-da8a-4699-8648-ed53964bb318"),
  user: 'userCars',
  db: 'admin',
  mechanisms: [ 'SCRAM-SHA-1', 'SCRAM-SHA-256' ],
  roles: [ { role: 'readYadaxCars', db: 'admin' } ],
  inheritedRoles: [ { role: 'readYadaxCars', db: 'admin' }
],
}
```



Arquitetura do MongoDB

Json vs Bson

JSON: JavaScript Object Notation

BSON: Binary JSON

```
{"hello": "world"} →  
\x16\x00\x00\x00      // total document size  
\x02                  // 0x02 = type String  
hello\x00             // field name  
\x06\x00\x00\x00world\x00 // field value  
\x00                  // 0x00 = type E00 ('end of object')
```

Datafile

```
[root@ip-20-0-1-116 mongo]# strings collection-9--5458125161921291952.wt
```

marca

HONDA

modelo

CITY

ano_fabricacao

ano_modelo

Prata

preco

marca

HONDA

modelo

CITY

```
[root@ip-20-0-1-116 mongo]# ll
```

total 456

```
-rw-----, 1 mongod mongod 20480 Jun 26 13:36 collection-0--5458125161921291952.wt
-rw-----, 1 mongod mongod 20480 Jun 26 13:36 collection-2--5458125161921291952.wt
-rw-----, 1 mongod mongod 24576 Jun 26 13:50 collection-4--5458125161921291952.wt
-rw-----, 1 mongod mongod 20480 Jun 26 13:37 collection-7--5458125161921291952.wt
-rw-----, 1 mongod mongod 20480 Jun 26 13:50 collection-9--5458125161921291952.wt
drwx-----, 2 mongod mongod    48 Jun 26 13:50 diagnostic.data
-rw-----, 1 mongod mongod 20480 Jun 26 13:50 index-10--5458125161921291952.wt
-rw-----, 1 mongod mongod 20480 Jun 26 13:36 index-1--5458125161921291952.wt
-rw-----, 1 mongod mongod 20480 Jun 26 13:36 index-3--5458125161921291952.wt
-rw-----, 1 mongod mongod 24576 Jun 26 13:50 index-5--5458125161921291952.wt
-rw-----, 1 mongod mongod 24576 Jun 26 13:50 index-6--5458125161921291952.wt
-rw-----, 1 mongod mongod 20480 Jun 26 13:37 index-8--5458125161921291952.wt
drwx-----, 2 mongod mongod   110 Jun 26 13:35 journal
-rw-----, 1 mongod mongod 36864 Jun 26 13:50 _mdb_catalog.wt
-rw-----, 1 mongod mongod     5 Jun 26 13:35 mongod.lock
-rw-----, 1 mongod mongod 36864 Jun 26 13:46 sizeStorer.wt
-rw-----, 1 mongod mongod   114 Jun 26 13:35 storage.bson
-rw-----, 1 mongod mongod    50 Jun 26 13:35 WiredTiger
-rw-----, 1 mongod mongod  4096 Jun 26 13:35 WiredTigerHS.wt
-rw-----, 1 mongod mongod    21 Jun 26 13:35 WiredTiger.lock
```



Datafile / directoryPerDB

O mongo permite separar DBs em diretórios diferentes.

Journal

- Estrutura relacionada a durabilidade de uma transação

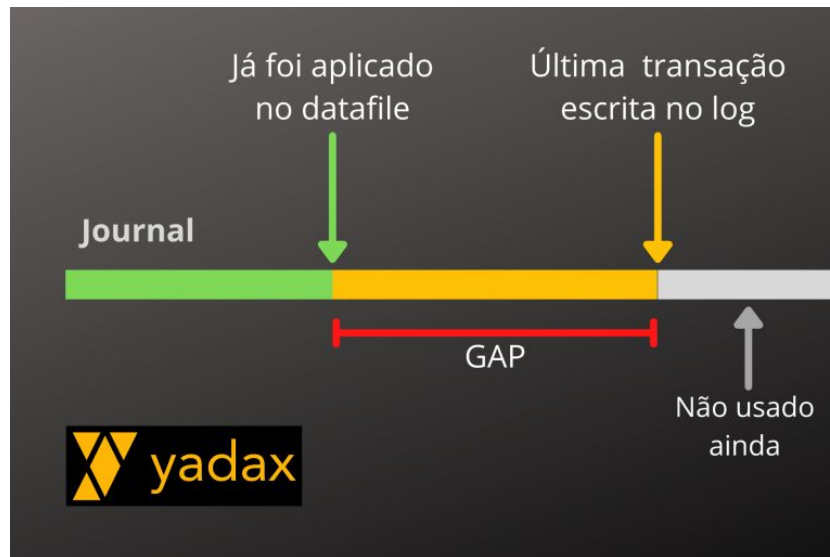
- REDO
- WAL
- BINLOG
- COMMITLOG
- TRANSACTLOG

```
[root@ip-20-0-1-116 mongo]# cd journal/  
[root@ip-20-0-1-116 journal]# ll  
total 307200  
-rw-----, 1 mongod mongod 104857600 Jun 26 14:12 WiredTigerLog.0000000001  
-rw-----, 1 mongod mongod 104857600 Jun 26 13:35 WiredTigerPrelog.0000000001  
-rw-----, 1 mongod mongod 104857600 Jun 26 13:35 WiredTigerPrelog.0000000002  
[root@ip-20-0-1-116 journal]# █
```

- É possível desabilitar para standalone (No Archive Log / Recovery Mode Simple)
 - storage.journal.enabled: false

Checkpoint

- Processo que aplica mudanças do Journal nos datafiles
- Ocorre a cada 60s



OpLog

É uma capped collection no DB

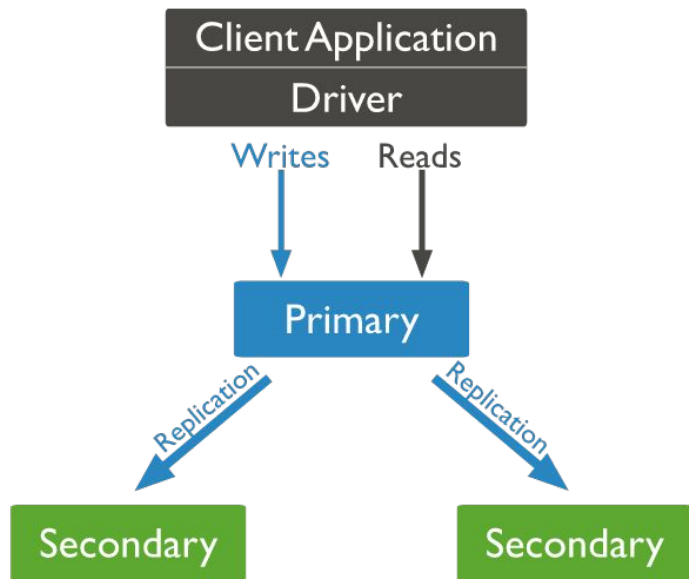
Guarda informação da transação para aplicar em secondaries

MongoDB usa uma técnica chamada de tailable cursor (tail -f)

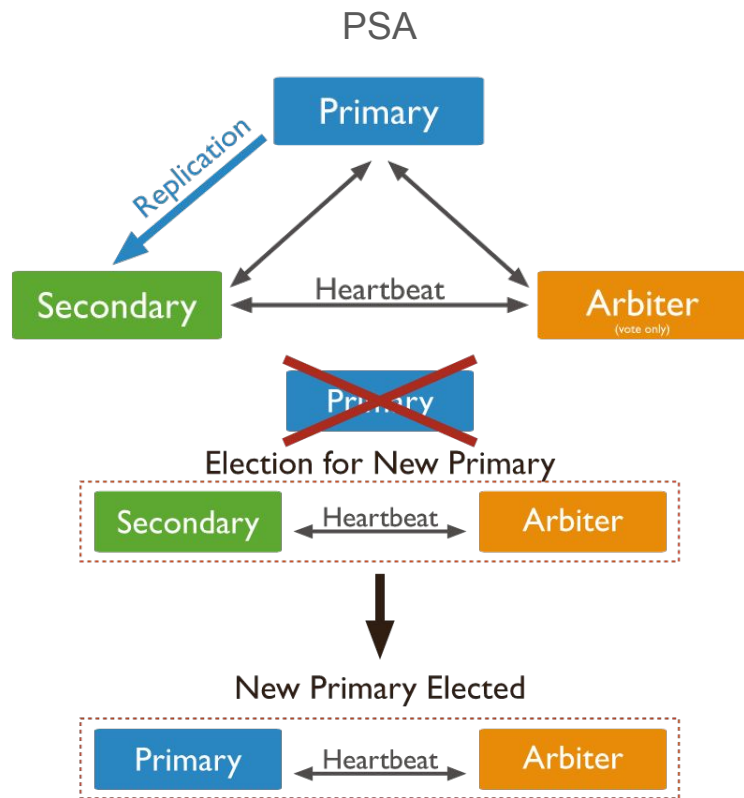
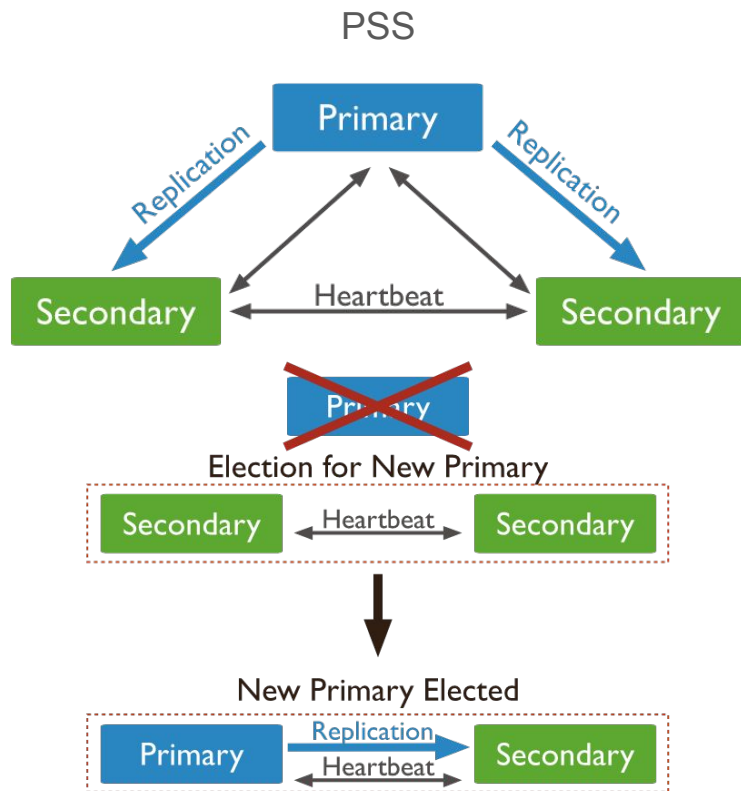
FIFO

ReplicaSet

- Definição: um conjunto de processos mongod que gerenciam o mesmo dataset.
- Redundância
- Alta disponibilidade
- Primary: Read / Write
- Secondary: Read only
- # ímpar de membros



ReplicaSet



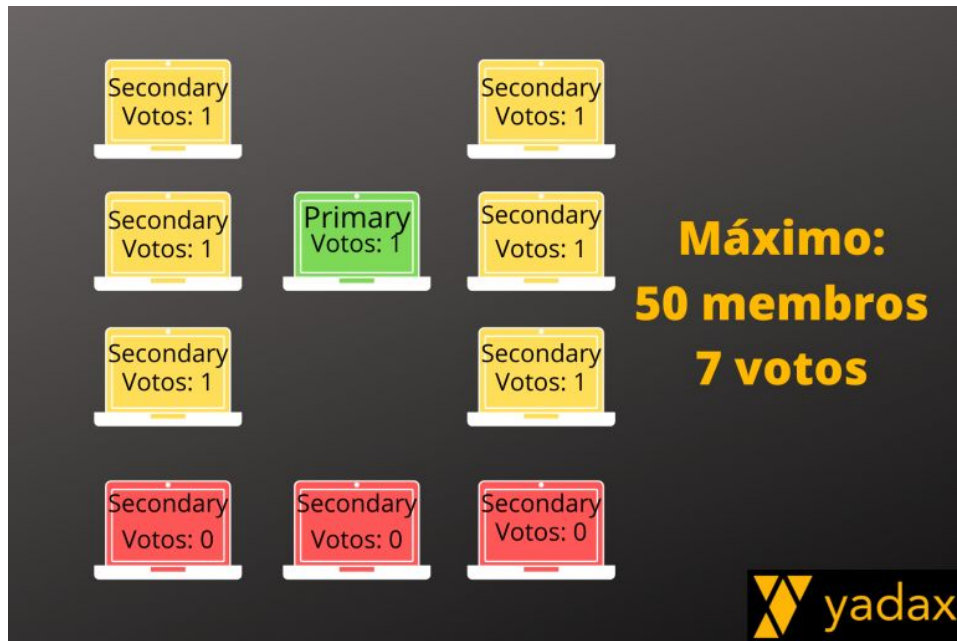
Eleição

Podemos ter 7 votos

Acontece sempre que perdemos o primary

Se desejar que um membro não vote, é possível configurar zero votos

Os drivers mais recentes percebem a indisponibilidade e possuem retry de escrita



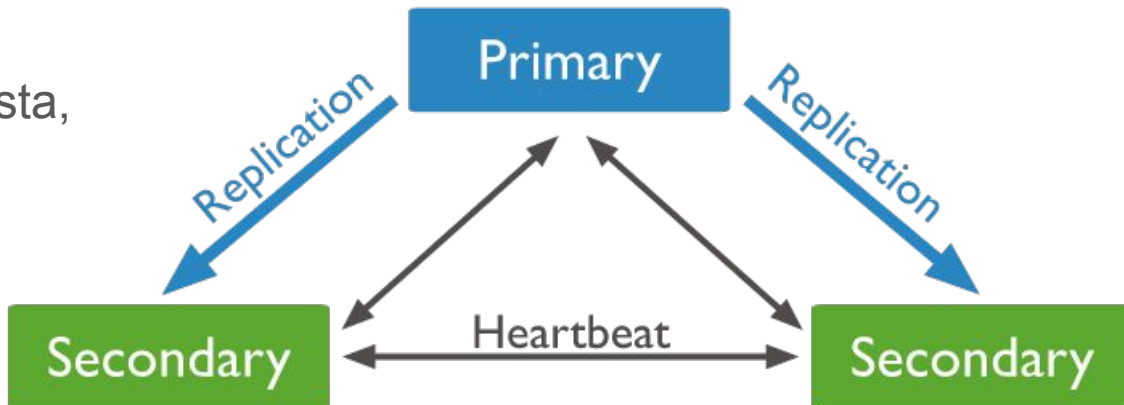
Eleição - Priority

- É possível configurar a prioridade dos membros, assim quanto maior a prioridade, maior a chance do membro virar primary.
- Membro com priority igual 0 não podem assumir o papel de primary
- Membros não votantes devem ter prioridade 0
- Membros com prioridade > 0 não podem ter 0 votos.

Heartbeat

Os membros testam a conexão a outros membros a cada 2s.

Se em 10 s não houver resposta, inicia uma eleição



ReplicaSet

Instalar o mongoDB nas tres maquinas

- Criar o arquivo de repo
 - `/etc/yum.repos.d/mongodb-org-8.0.repo`
- Instalar o MongoDB
 - `yum install -y mongodb-org`
- Configurar o `/etc/mongod.conf`

Vamos configurar nosso primeiro ReplicaSet

```
net:  
  bindIp: 0.0.0.0  
    ○ Por default, usamos o localhost (127.0.0.1)  
    ○ O que isso significa?  
    ○ O que devemos usar?  
      ■ 0.0.0.0
```

```
replication:  
  replSetName: rsYadax
```

```
security:  
  authorization: disabled
```



Vamos configurar nosso primeiro ReplicaSet

```
storage:  
  dbPath: ...  
  journal:  
    enabled: true  
  directoryPerDB: true
```

Restarte o serviço do mongod:

- `systemctl restart mongod`

Configurar nosso primeiro ReplicaSet

Assista a mágica acontecer no log.

```
$ mongosh
test> rs.initiate()
{
  info2: 'no configuration specified. Using a default configuration for the
set',
  me: 'ip-20-0-1-254.ec2.internal:27017',
  ok: 1
}
rsYadax [direct: other] test>

rsYadax [direct: primary] test>
```



Quando não temos DNS

```
# vi /etc/hosts
```

```
192.168.0.NNM mongodb1
```

```
192.168.0.NNN mongodb2
```

```
192.168.0.NNO mongodb3
```

Configurar nosso primeiro ReplicaSet

Vamos explorar a configuração. Veja estas duas opções:

```
rs.status()
```

```
rs.conf()
```

Adicionando membros

- Instalar o mongodb na mesma versão
 - `# vi /etc/yum.repos.d/mongodb-8.0.repo`
 - `# yum install -y mongodb-org`
- Copiar o arquivo de conf da maquina 1 para as maquinas 2 e 3.

Adicionando membros

- Certifique que o novo membro seja alcançável na porta do mongo (27017)
 - yum install telnet
- O novo membro deve ter o mesmo replSetName
- O novo membro deve estar UP
 - "error": "NotYetInitialized: Cannot use non-local read concern until replica set is finished initializing."
- Você vai precisar do fqdn (hostname -f)
 - Nas versões mais novas do MongoDB, não podemos criar o ReplicaSet baseado no IP. Precisamos de um nome "resolvível" (DNS ou /etc/hosts)
-

Testando a conexão

```
[root@ip-20-0-1-254 ~]# telnet ip-20-0-1-129.ec2.internal  
27017
```

```
Trying 20.0.1.129...
```

```
Connected to ip-20-0-1-129.ec2.internal.
```

```
Escape character is '^]'.
```



Adicionando apenas um Secondary

```
rsYadax [direct: primary] test>
```

```
rs.add( { host: "ip-10-0-66-196.sa-east-1.compute.internal:27017" } )
```

```
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1656337760, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"),
0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1656337760, i: 1 })
}
rsYadax [direct: primary] test>
```



Status ReplicaSet

Vamos explorar a configuração.

```
rs.status()
```

```
rs.status().members
```

Status ReplicaSet

```
rsYadax [direct: primary] test>  
rs.status().members.forEach(function(x) {  
    print(x.name)  
    })
```

```
ip-20-0-1-254.ec2.internal:27017
```

```
ip-20-0-1-129.ec2.internal:27017
```



Status ReplicaSet

```
rsYadax [direct: primary] test>  
rs.status().members.forEach(function(x) {  
    print(x.name + '/' + x.stateStr + '/' + x.health )  
    })
```

```
ip-20-0-1-254.ec2.internal:27017/PRIMARY/1
```

```
ip-20-0-1-129.ec2.internal:27017/SECONDARY/1
```



Stop Secondary

Pare o mongod no secondary

- `systemctl stop mongod`

O que aconteceu com o primary?

Espera uns 15s desde o stop e:

```
rs.status()
```

Stop Secondary

"Failed to check socket connectivity"

"Dropping unhealthy pooled connection"

"Heartbeat failed after max retries"

"Member is now in state RS_DOWN"

"Error connecting to ip-20-0-1-129.ec2.internal:27017 (20.0.1.129:27017) ::
caused by :: Connection refused"

"Can't see a majority of the set, relinquishing primary"

"Stepping down from primary in response to heartbeat"



Stop Secondary

"Starting to kill user operations"

"Stopped killing user operations"

"Invalidating sessions for stepdown."

"Replica set state

transition", "attr": {"newState": "SECONDARY", "oldState": "PRIMARY"}

"Interrupting PrimaryOnlyService due to stepDown"

Starte o antigo secondary

O que aconteceu?

Adicionando terceiro node

- Instalar o mongodb na mesma versão
 - `# vi /etc/yum.repos.d/mongodb-8.0.repo`
 - `# yum install -y mongodb-org`
- Copiar o arquivo de conf da maquina 1 para as maquinas 2 e 3.

Adicionando membros

- Certifique que o novo membro seja alcançável na porta do mongo (27017)
 - yum install telnet
- O novo membro deve ter o mesmo replSetName
- O novo membro deve estar UP
 - "error": "NotYetInitialized: Cannot use non-local read concern until replica set is finished initializing."
- Você vai precisar do fqdn (hostname -f)
 - Nas versões mais novas do MongoDB, não podemos criar o ReplicaSet baseado no IP. Precisamos de um nome "resolvível" (DNS ou /etc/hosts)
-

Vamos adicionar mais um membro

```
rsYadax [direct: primary] test> rs.add({host: 'ip-20-0-1-193.ec2.internal:27017'})

{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1656420569, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("00000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1656420569, i: 1 })
}
rsYadax [direct: primary] test> rs.status().members
```



Configurar todos os nodes do ReplicaSet

Você também pode informar todos os membros no momento de iniciar o ReplicaSet

```
rs.initiate( {  
  _id : "rs0",  
  members: [  
    { _id: 0, host: "mongodb0.yadax.com.br:27017" },  
    { _id: 1, host: "mongodb1.yadax.com.br:27017" },  
    { _id: 2, host: "mongodb2.yadax.com.br:27017" }  
  ]  
})
```



Verifique o status

```
rsYadax [direct: primary] test>  
rs.status().members.forEach(function(x) {  
    print(x.name + '/' + x.stateStr + '/' + x.health )  
    })  
ip-20-0-1-139.ec2.internal:27017/PRIMARY/1  
ip-20-0-1-53.ec2.internal:27017/SECONDARY/1  
ip-20-0-1-193.ec2.internal:27017/SECONDARY/1
```



Pare um dos secondaries e verifique o status

```
rsYadax [direct: primary] test>  
rs.status().members.forEach(function (x) { print(x.name +  
  '/' + x.stateStr + '/' + x.health); })  
  
ip-20-0-1-139.ec2.internal:27017/PRIMARY/1  
ip-20-0-1-53.ec2.internal:27017/SECONDARY/1  
ip-20-0-1-193.ec2.internal:27017/(not reachable/healthy)/0
```



Starte o secondary e verifique o status

```
rsYadax [direct: primary] test>  
rs.status().members.forEach(function (x) { print(x.name +  
'/' + x.stateStr + '/' + x.health); })  
ip-20-0-1-139.ec2.internal:27017/PRIMARY/1  
ip-20-0-1-53.ec2.internal:27017/SECONDARY/1  
ip-20-0-1-193.ec2.internal:27017/SECONDARY/1
```



Pare o primary e verifique o status

```
rsYadax [direct: primary] test>  
rs.status().members.forEach(function (x) { print(x.name +  
'/' + x.stateStr + '/' + x.health); })  
ip-20-0-1-139.ec2.internal:27017/(not_reachable/healthy)/0  
ip-20-0-1-53.ec2.internal:27017/PRIMARY/1  
ip-20-0-1-193.ec2.internal:27017/SECONDARY/1
```



Starte o antigo primary e verifique o status

```
rsYadax [direct: primary] test>  
rs.status().members.forEach(function (x) { print(x.name +  
  '/' + x.stateStr + '/' + x.health); })  
ip-20-0-1-139.ec2.internal:27017/SECONDARY/1  
ip-20-0-1-53.ec2.internal:27017/PRIMARY/1  
ip-20-0-1-193.ec2.internal:27017/SECONDARY/1
```



Switchover

No atual primary:

```
rsYadax [direct: primary] test> rs.stepDown()
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1656422458, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"),
0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1656422458, i: 1 })
}
```



Switchover

Verifique o status

```
rsYadax [direct: secondary] test>  
rs.status().members.forEach(function (x) { print(x.name +  
'/' + x.stateStr + '/' + x.health); })  
ip-20-0-1-139.ec2.internal:27017/PRIMARY/1  
ip-20-0-1-53.ec2.internal:27017/SECONDARY/1  
ip-20-0-1-193.ec2.internal:27017/SECONDARY/1
```



Habilitando autenticação no ReplicaSet

Entre membros:

- Internal authentication (using keyfile)
- Clients: User Access Control

Habilitando autenticação no ReplicaSet

Criar um keyfile:

```
# echo "qualquercoisabase64" > /etc/mongo.keyfile
```

ou

```
# openssl rand -base64 756 > /etc/mongo.keyfile
```

```
# chown mongod:mongod /etc/mongo.keyfile
```

```
# chmod 400 /etc/mongo.keyfile
```



Habilitando autenticação no ReplicaSet

Criar um user Administrativo:

```
> use admin
> db.createUser(
  {
    user: "userAdmin",
    pwd: passwordPrompt(),
    roles: [
      { role: "root", db: "admin" },
    ]
  }
)
```

Habilitando autenticação no ReplicaSet

Criar um keyfile nos 3 nodes:

```
echo "qualquercoisabase64" > /etc/mongo.keyfile
```

ou

```
openssl rand -base64 756 > /etc/mongo.keyfile
```

```
chown mongod:mongod /etc/mongo.keyfile
```

```
chmod 400 /etc/mongo.keyfile
```



Habilitando autenticação no ReplicaSet

Alterar o mongod.conf

```
security:
```

```
  authorization: enabled
```

```
  keyFile: /etc/mongo.keyfile
```



Habilitando autenticação no ReplicaSet

Parar todos os membros, começando pelos secondaries

Startar todos os membros

```
rsYadax [direct: primary] test> rs.status().members.forEach(function  
(x) { print(x.name + '/' + x.stateStr + '/' + x.health); })
```

```
ip-20-0-1-139.ec2.internal:27017/SECONDARY/1
```

```
ip-20-0-1-53.ec2.internal:27017/PRIMARY/1
```

```
ip-20-0-1-193.ec2.internal:27017/SECONDARY/1
```



Status da replicação

- rsYadax [direct: primary] test> **rs.printReplicationInfo()**
- rsYadax [direct: primary] test> **rs.printSecondaryReplicationInfo()**



Habilitando autenticação no ReplicaSet

Teste a autenticação

Alterando a prioridade de um membro

ip-20-0-1-207.ec2.internal:27017/PRIMARY/1 << prioridade 5

ip-20-0-1-116.ec2.internal:27017/SECONDARY/1 << prioridade 4

ip-20-0-1-137.ec2.internal:27017/SECONDARY/1 << prioridade 1



Alterando a prioridade de um membro

Verifique o retorno do:

```
rs.conf()
```

```
rs.conf().members
```

Guarde o retorno do `rs.conf()` em uma variable `cfg`

```
cfg = rs.conf()
```

```
cfg.members[0]
```

```
cfg.members[0].priority
```

Alterando a prioridade de um membro

```
rs.reconfig( configuration, { options } )
```

Reconfigura um replicaset sobrescrevendo as configurações anteriores

configuration -> é o retorno do rs.conf()

Podemos manipular o cfg e reconfigurar com uma a nova versão



Alterando a prioridade de um membro

```
rsYadax [direct: primary] test> cfg = rs.conf()

rsYadax [direct: primary] test> cfg.members[0]

rsYadax [direct: primary] test> cfg.members[0].priority
1

rsYadax [direct: primary] test> cfg.members[0].priority = 5
5
rsYadax [direct: primary] test> cfg.members[1].priority = 4
4
rsYadax [direct: primary] test> rs.reconfig(cfg)
{
  ok: 1,
  '$clusterTime': {
...
}
```



Alterando a prioridade de um membro

Verifique novamente o rs.conf()

```
rsYadax [direct: primary] test> rs.conf().members
[
  {
    _id: 0,
    host: 'ip-20-0-1-207.ec2.internal:27017',
    priority: 5,
  },
  {
    _id: 1,
    host: 'ip-20-0-1-116.ec2.internal:27017',
    priority: 4,
  },
  {
    _id: 2,
    host: 'ip-20-0-1-137.ec2.internal:27017',
    priority: 1,
  }
]
```



Alterando a prioridade de um membro

Quem é o primary?

Faça um `rs.stepDown()`. Quem assume?

Alterando os votos de um membro

O procedimento anterior é o mesmo para alterar o número de votos.

Mude para:

ip-20-0-1-207.ec2.internal:27017/PRIMARY/1 << voto 1

ip-20-0-1-116.ec2.internal:27017/SECONDARY/1 << voto 1

ip-20-0-1-137.ec2.internal:27017/SECONDARY/1 << voto 0



Remover membro do ReplicaSet

Pare o mongod do membro a ser removido

Conecte no primary

```
rsYadax [direct: primary] test> rs.status().members.forEach(function(x)
{print(x.name + '/' + x.stateStr + '/' + x.health )})
```

```
ip-20-0-1-7.ec2.internal:27017/PRIMARY/1
```

```
ip-20-0-1-8.ec2.internal:27017/SECONDARY/1
```

```
ip-20-0-1-12.ec2.internal:27017/(not reachable/healthy)/0
```



Remover membro do ReplicaSet

Conecte no primary

```
rsYadax [direct: primary] test> rs.remove("ip-20-0-1-12.ec2.internal:27017")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1656499768, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("f73cb45f86a2033e72c44b2c16fd07e2972abe0b", "hex"),
0),
      keyId: Long("7114608012949454854")
    }
  },
  operationTime: Timestamp({ t: 1656499768, i: 1 })
}
```

Limpe o conteúdo em /var/lib/mongo no node removido:

```
# systemctl stop mongod
# rm -rf /var/lib/mongo/*
```



Remover membro do ReplicaSet

Consulte agora o status

```
rsYadax [direct: primary] test>  
rs.status().members.forEach(function(x) {print(x.name + '/'  
+ x.stateStr + '/' + x.health )})
```

```
ip-20-0-1-7.ec2.internal:27017/PRIMARY/1
```

```
ip-20-0-1-8.ec2.internal:27017/SECONDARY/1
```



Adicionar Árbitro no ReplicaSet

Adicione novamente o membro, agora como árbitro.

```
db.adminCommand( { getDefaultRWConcern : 1 } )

db.adminCommand({
  "setDefaultRWConcern" : 1,
  "defaultWriteConcern" : {
    "w" : 1
  }
})
```



Adicionar Árbitro no ReplicaSet

Adicione novamente o membro, agora como arbitro.

Certifique que limpou os dados anteriores

```
rsYadax [direct: primary] test> rs.addArb("ip-20-0-1-127.ec2.internal:27017")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1656671509, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("549e20c0366a0fc3072e0c40d22749e4693b24e8", "hex"),
0),
      keyId: Long("7115346876173385734")
    }
  },
  operationTime: Timestamp({ t: 1656671509, i: 1 })
}
rsYadax [direct: primary] test>
```



Adicionar Árbitro no ReplicaSet

Consulte novamente o status

```
rsYadax [direct: primary] test>  
rs.status().members.forEach(function(x) {print(x.name + '/' +  
x.stateStr + '/' + x.health )})
```

```
ip-20-0-1-136.ec2.internal:27017/PRIMARY/1
```

```
ip-20-0-1-7.ec2.internal:27017/SECONDARY/1
```

```
ip-20-0-1-127.ec2.internal:27017/ARBITER/1
```



Derrube o Primary e o Árbitro

Stop o mongod em dois hosts.

O que houve?

```
rsYadax [direct: secondary] test>  
rs.status().members.forEach(function (x) { print(x.name +  
'/' + x.stateStr + '/' + x.health); })
```



Derrube o Primary e o Arbitro

Como recuperar?

Derrube o Primary e o Arbitro

```
rs.reconfig(  
<configuration>,  
{  
  "force" : <boolean>,  
  "maxTimeMS" : <int>  
}  
)
```

```
rs.reconfig(  
  cfg,  
  {"force" : true}  
)
```

Collections relacionadas

```
rsYadax [direct: primary] local> show collections;  
oplog.rs  
replset.election  
replset.initialSyncId  
replset.minvalid  
replset.oplogTruncateAfterPoint  
startup_log  
system.replset  
system.rollback.id  
system.tenantMigration.oplogView [view]  
system.views
```



Backup / Restore

Tipos de backup (Lógicos)

Exporta JSON

Texto

Mais lento

- mongoexport
- mongoimport

Exporta BSON

Binário

Mais rápido

- mongodump
- mongorestore

Backups Físicos

Não há ferramenta disponível nativa para backup de MongoDB

Percona Backup for MongoDB está em Preview

É possível utilizar LVM Snapshot / Block Volume (EBS) Snapshot



Vamos importar alguns registros

```
use yadax;
```

```
db.cars.insertMany([
  {"marca": "HONDA", "modelo": "CIVIC", "ano_fabricacao": 2015, "ano_modelo": 2016, "cor": "Prata", "preco": 60000},
  {"marca": "HONDA", "modelo": "Civic", "ano_fabricacao": 2018, "ano_modelo": 2018, "cor": "Branco", "preco": 80000},
  {"marca": "HONDA", "modelo": "FIT", "ano_fabricacao": 2019, "ano_modelo": 2019, "cor": "Branco", "preco": 50000},
  {"marca": "FIAT", "modelo": "UNO", "ano_fabricacao": 2000, "ano_modelo": 2001, "cor": "Preto", "preco": 10000},
  {"marca": "FIAT", "modelo": "PALIO", "ano_fabricacao": 2006, "ano_modelo": 2007, "cor": "Prata", "preco": 11000},
  {"marca": "FIAT", "modelo": "STRADA", "ano_fabricacao": 2018, "ano_modelo": 2018, "cor": "Branco", "preco": 40000},
  {"marca": "FIAT", "modelo": "TORO", "ano_fabricacao": 2020, "ano_modelo": 2020, "cor": "Preto", "preco": 90000},
  {"marca": "FIAT", "modelo": "TORO", "ano_fabricacao": 2019, "ano_modelo": 2020, "cor": "Prata", "preco": 80000},
  {"marca": "VW", "modelo": "GOL", "ano_fabricacao": 2012, "ano_modelo": 2012, "cor": "Preto", "preco": 20000},
  {"marca": "VW", "modelo": "POLO", "ano_fabricacao": 2020, "ano_modelo": 2020, "cor": "Branco", "preco": 60000},
  {"marca": "VW", "modelo": "GOLF", "ano_fabricacao": 2019, "ano_modelo": 2019, "cor": "Prata", "preco": 90000},
  {"marca": "VW", "modelo": "GOL", "ano_fabricacao": 2003, "ano_modelo": 2003, "cor": "Branco", "preco": 10000},
  {"marca": "FORD", "modelo": "FIESTA", "ano_fabricacao": 2005, "ano_modelo": 2006, "cor": "Prata", "preco": 11000},
  {"marca": "FORD", "modelo": "FIESTA", "ano_fabricacao": 2007, "ano_modelo": 2007, "cor": "Prata"},
  {"marca": "FORD", "modelo": "PAMPA", "ano_fabricacao": 1989, "ano_modelo": 1989, "cor": "Prata", "preco": null}
])
```



mongoexport

-h, --host=<hostname>

--port=<port>

-u, --username=<username>

-p, --password=<password>

--authenticationDatabase=<database-name>

-d, --db=<database-name>

-c, --collection=<collection-name>

--uri=mongodb-uri

-f, --fields=e.g. -f "name,age"

--type=json or csv

-o, --out=<filename>

-q, --query=<json>

--skip=<count>

--limit=<count>



mongoexport

Dump full to stdout

```
mongoexport --username=admin --password=password2  
--authenticationDatabase=admin --db=yadax --collection=cars
```

```
{ "_id": {"$oid": "630d2c8e0853c6be1fa9764e"}, "marca": "HONDA", "modelo": "CIVIC", "ano_fabricacao": 2015, "ano_modelo": 2016, "cor": "Prata", "preco": 60000 }  
{ "_id": {"$oid": "630d2c8e0853c6be1fa9764f"}, "marca": "HONDA", "modelo": "Civic", "ano_fabricacao": 2018, "ano_modelo": 2018, "cor": "Branco", "preco": 80000 }  
{ "_id": {"$oid": "630d2c8e0853c6be1fa97650"}, "marca": "HONDA", "modelo": "FIT", "ano_fabricacao": 2019, "ano_modelo": 2019, "cor": "Branco", "preco": 50000 }  
{ "_id": {"$oid": "630d2c8e0853c6be1fa97651"}, "marca": "FIAT", "modelo": "UNO", "ano_fabricacao": 2000, "ano_modelo": 2001, "cor": "Preto", "preco": 10000 }  
{ "_id": {"$oid": "630d2c8e0853c6be1fa97652"}, "marca": "FIAT", "modelo": "PALIO", "ano_fabricacao": 2006, "ano_modelo": 2007, "cor": "Prata", "preco": 11000 }  
{ "_id": {"$oid": "630d2c8e0853c6be1fa97653"}, "marca": "FIAT", "modelo": "STRADA", "ano_fabricacao": 2018, "ano_modelo": 2018, "cor": "Branco", "preco": 40000 }  
{ "_id": {"$oid": "630d2c8e0853c6be1fa97654"}, "marca": "FIAT", "modelo": "TORO", "ano_fabricacao": 2020, "ano_modelo": 2020, "cor": "Preto", "preco": 90000 }
```



mongoexport

Escolhendo os campos desejados

```
mongoexport --username=admin --password=password2  
--authenticationDatabase=admin --db=yadax --collection=cars  
--fields="marca,modelo,preco"
```

```
{"_id":{"_id":"630d2c8e0853c6be1fa9764e"},"marca":"HONDA","modelo":"CIVIC","preco":60000}  
{"_id":{"_id":"630d2c8e0853c6be1fa9764f"},"marca":"HONDA","modelo":"Civic","preco":80000}  
{"_id":{"_id":"630d2c8e0853c6be1fa97650"},"marca":"HONDA","modelo":"FIT","preco":50000}  
{"_id":{"_id":"630d2c8e0853c6be1fa97651"},"marca":"FIAT","modelo":"UNO","preco":10000}  
{"_id":{"_id":"630d2c8e0853c6be1fa97652"},"marca":"FIAT","modelo":"PALIO","preco":11000}  
{"_id":{"_id":"630d2c8e0853c6be1fa97653"},"marca":"FIAT","modelo":"STRADA","preco":40000}
```



mongoexport

Type CSV

```
mongoexport --username=admin --password=password2  
--authenticationDatabase=admin --db=yadax --collection=cars  
--fields="marca,modelo,preco" --type=csv
```

```
marca,modelo,preco  
HONDA,CIVIC,60000  
HONDA,Civic,80000  
HONDA,FIT,50000  
FIAT,UNO,10000  
FIAT,PALIO,11000
```



mongoexport

Query

```
# mongoexport --username=userAdmin --password=password2  
--authenticationDatabase=admin --db=yadax --collection=cars  
--fields="marca,modelo,preco" --type=csv  
--query='{ "marca": "FIAT" }'
```

```
marca,modelo,preco  
FIAT,UNO,10000  
FIAT,PALIO,11000  
FIAT,STRADA,40000  
FIAT,TORO,90000  
FIAT,TORO,80000
```



mongoexport

Pulando 1

```
mongoexport --username=admin --password=password2  
--authenticationDatabase=admin --db=yadax --collection=cars  
--fields="marca,modelo,preco" --type=csv --query  
'{"marca":"FIAT"}' --skip=1
```



mongoexport

Limitando a 2

```
mongoexport --username=admin --password=password2  
--authenticationDatabase=admin --db=yadax --collection=cars  
--fields="marca,modelo,preco" --type=csv --query  
'{"marca":"FIAT"}' --skip=1 --limit=2
```



mongoexport

Jogando em um arquivo

```
mongoexport --username=admin --password=password2  
--authenticationDatabase=admin --db=yadax --collection=cars  
--fields="marca,modelo,preco" --type=csv --query  
'{"marca":"FIAT"}' --skip=1 --limit=2 --out=cars.csv
```

```
$ cat cars.csv
```



Drop a collection cars

```
rsYadax1 [direct: primary] test> use yadax;
```

```
switched to db yadax
```

```
rsYadax1 [direct: primary] yadax> db.cars.drop()
```

```
true
```



mongoimport

--headerline (CSV, TSV)

--drop

--maintainInsertionOrder

--ignoreBlanks

--numInsertionWorkers

--stopOnError

--mode=[insert|upsert|merge|delete]

--writeConcern

--db=<database-name>

--collection=<collection-name>

--fields="name,age"

--file=<filename>

--type=json, csv, or tsv



mongoimport

```
# mongoimport --username=admin --password=password2  
--authenticationDatabase=admin --db=yadax --collection=cars  
--type=csv --file=cars.csv --headerline
```

```
2 document(s) imported successfully. 0 document(s) failed to  
import.
```



Query collection

```
[root@ip-20-0-1-154 ~]# cat cars.csv  
marca,modelo,preco  
FIAT,PALIO,11000  
FIAT,STRADA,40000
```

```
rsYadax1 [direct: primary] test> use yadax;  
switched to db yadax  
rsYadax1 [direct: primary] yadax> db.cars.find()  
[  
  {  
    _id: ObjectId("630d37b8e303d928abe89d10"),  
    marca: 'FIAT',  
    modelo: 'STRADA',  
    preco: 40000  
  },  
  {  
    _id: ObjectId("630d37b8e303d928abe89d11"),  
    marca: 'FIAT',  
    modelo: 'PALIO',  
    preco: 11000  
  }  
]  
rsYadax1 [direct: primary] yadax> █
```



Vamos importar alguns dados

```
# yum install -y git
```

```
# git clone https://github.com/mcampo2/mongodb-sample-databases.git
```

```
# cd mongodb-sample-databases
```

```
# mongoimport --drop --host localhost --port 27017 \  
--username dba --password mudar123 \  
--authenticationDatabase 'admin' \  
--db sample --collection airbnb \  
--file sample_airbnb/listingsAndReviews.json
```



mongodump

-h, --host=<hostname>

--port=<port>

-u, --username=<username>

-p, --password=<password>

--authenticationDatabase=<database-name>

-d, --db=<database-name>

-c, --collection=<collection-name>

-q, --query=

-o, --out=<directory-path>

--gzip

--archive=<file-path>

--dumpDbUsersAndRoles

--excludeCollection=<collection-name>

--excludeCollectionsWithPrefix=<collection-prefix>



LAB

Faça um backup da `sample.airbnb`

LAB

Faça um backup da sample.airbnb

```
mongodump --username=admin --password=password2  
--authenticationDatabase=admin --db=sample  
--collection=airbnb --archive=airbnb.dump
```

```
writing sample.airbnb to archive 'airbnb.dump'
```

```
done dumping sample.airbnb (5555 documents)
```



mongorestore

d, --db=<database-name>

-c, --collection=<collection-name>

--excludeCollection=<collection-name> DEPR

--excludeCollectionsWithPrefix DEPR

--nsExclude=<namespace-pattern>

--nsInclude=<namespace-pattern>

--nsFrom=<namespace-pattern>

--nsTo=<namespace-pattern>

--archive=<filename>

--restoreDbUsersAndRoles

--dir=<directory-name>

--gzip

--drop

--dryRun

--noOptionsRestore

--maintainInsertionOrder



LAB

Drope a sample.airbnb

Restaure o backup

LAB

Drope a sample.airbnb

```
rsYadax1 [direct: primary] yadax>  
use sample
```

```
switched to db sample
```

```
rsYadax1 [direct: primary] sample>  
db.airbnb.drop()
```

```
true
```

Restaure o backup

```
mongorestore --username=admin  
--password=password2  
--authenticationDatabase=admin  
--db=sample --collection=airbnb  
--archive=airbnb.dump
```

```
finished restoring sample.airbnb (5555  
documents, 0 failures)
```

```
no indexes to restore for collection  
sample.airbnb
```

```
5555 document(s) restored successfully. 0  
document(s) failed to restore.
```



LAB

Drope a sample.airbnb

Restaure o backup na collection imoveis

LAB

Drope a sample.airbnb

```
rsYadax1 [direct: primary] yadax>  
use sample
```

switched to db sample

```
rsYadax1 [direct: primary] sample>  
db.airbnb.drop()
```

true

Restaure o backup

```
mongorestore --username=admin  
--password=password2  
--authenticationDatabase=admin  
--nsInclude=sample.airbnb --archive=airbnb.dump  
--nsFrom=sample.airbnb --nsTo=sample.imoveis
```

finished restoring sample.imoveis (5555
documents, 0 failures)

no indexes to restore for collection
sample.imoveis

5555 document(s) restored successfully. 0
document(s) failed to restore.



Percona Backup for MongoDB (PBM)

PBM

Open Source

Baixo Impacto

Distribuído

Shards e ReplicaSet

Backup Físico* ou Lógico

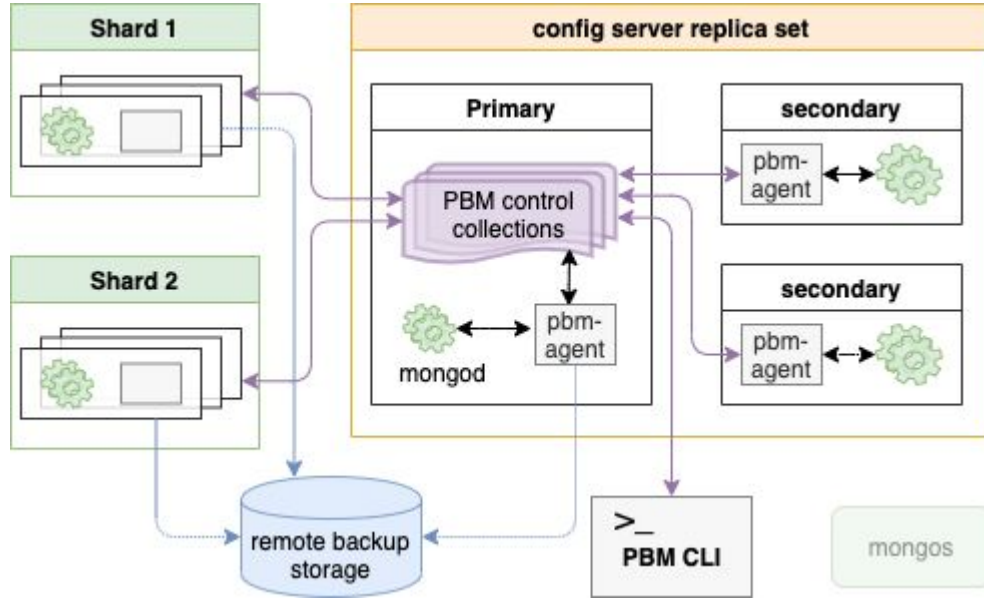
PIT Recovery para backups lógicos

CLI simples

Destino precisa ser Shared: S3 compat, Azure, NFS



PBM - Componentes



rsYadax1 [direct:
pbmAgents
pbmBackups
pbmCmd
pbmConfig
pbmLock
pbmLockOp
pbmLog
pbmOpLog
pbmPITRChunks
pbmRestores

Vamos criar um ReplicaSet

```
# /etc/mongod.conf

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# Where and how to store data.
storage:
  dbPath: /var/lib/mongod
  directoryPerDB: true
  journal:
    enabled: true

# how the process runs
processManagement:
  fork: true # fork and run in background
  pidFilePath: /var/run/mongodb/mongod.pid
  timeZoneInfo: /usr/share/zoneinfo

# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0

security:
  authorization: enabled
  keyFile: /etc/keyfile.mongo

replication:
  replSetName: rsYadax1
```

```
[Unit]
Description=MongoDB Database Shard Server
Documentation=https://docs.mongodb.org/manual
After=network-online.target
Wants=network-online.target

[Service]
User=mongod
Group=mongod
Environment="OPTIONS=-f /etc/mongod.conf"
EnvironmentFile=-/etc/sysconfig/mongod
ExecStart=/usr/bin/mongod $OPTIONS
ExecStartPre=/usr/bin/mkdir -p /var/run/mongodb
ExecStartPre=/usr/bin/chown mongod:mongod /var/run/mongodb
ExecStartPre=/usr/bin/chmod 0755 /var/run/mongodb
PermissionsStartOnly=true
PIDFile=/var/run/mongodb/mongod.pid
Type=forking
LimitFSIZE=infinity
LimitCPU=infinity
LimitAS=infinity
LimitNOFILE=64000
LimitNPROC=64000
LimitMEMLOCK=infinity
TasksMax=infinity
TasksAccounting=false

[Install]
WantedBy=multi-user.target
```



Vamos criar um ReplicaSet

```
# echo "qualquercoisabase64" > /etc/keyfile.mongo  
# chown mongod: /etc/keyfile.mongo  
# chmod 400 /etc/keyfile.mongo
```

- Stop mongod (hosts 2 e 3)
- rm -rf /var/lib/mongo/* (hosts 2 e 3)
- Start mongod (host 2 e 3)

Vamos criar um ReplicaSet

```
> rs.initiate() (done)
> use admin
> db.createUser(
  {
    user: 'admin',
    pwd: 'password',
    roles: [ { role: 'root', db: 'admin' } ]
  }
);
> exit;
```

```
# mongosh --username userRoot --password AdminP4ss --authenticationDatabase
admin
```

```
> rs.add('ip-20-0-1- 20.ec2.internal:27017')
> rs.add('ip-20-0-1- 190.ec2.internal:27017')
```



Install percona-release - Todos os hosts

```
# sudo yum install -y
```

```
https://repo.percona.com/yum/percona-release-latest.noarch.rpm
```

```
# sudo percona-release enable pbm release
```



Install pbm - Todos os hosts

```
sudo yum install percona-backup-mongodb
```



Create PBM role - No Primary

```
> db.getSiblingDB("admin").createRole({ "role": "pbmAnyAction",  
    "privileges": [  
        { "resource": { "anyResource": true },  
          "actions": [ "anyAction" ]  
        }  
    ],  
    "roles": []  
});
```



Create PBM user - No Primary

```
db.getSiblingDB("admin").createUser({user: "pbmuser",  
  "pwd": "password",  
  "roles" : [  
    { "db" : "admin", "role" : "readWrite", "collection": "" },  
    { "db" : "admin", "role" : "backup" },  
    { "db" : "admin", "role" : "clusterMonitor" },  
    { "db" : "admin", "role" : "restore" },  
    { "db" : "admin", "role" : "pbmAnyAction" }  
  ]  
});
```



Verifique o service pbm-agent

systemctl status pbm-agent

```
[root@ip-20-0-1-154 ~]# cat /usr/lib/systemd/system/pbm-agent.service
[Unit]
Description=pbm-agent
After=time-sync.target network.target

[Service]
EnvironmentFile=-/etc/sysconfig/pbm-agent
Type=simple
User=mongod
Group=mongod
PermissionsStartOnly=true
ExecStart=/usr/bin/pbm-agent

[Install]
WantedBy=multi-user.target
[root@ip-20-0-1-154 ~]# █
```



Mongosh + connection string (a partir de um SEC)

```
[root@ip-10-0-65-42 ~]# mongosh  
"mongodb://userAdmin:AdmlnP4ss@localhost:27017/?authSource=admin"
```

```
Current Mongosh Log ID: 658b58d334cd751894979114
```

```
Connecting to:
```

```
mongodb://<credentials>@localhost:27017/?authSource=admin&directConnectio  
n=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.1
```

```
...
```

```
rsYadax [direct: secondary] test>
```



Mongosh + connection string

```
[root@ip-10-0-65-42 ~]# mongosh  
"mongodb://userAdmin:AdmlnP4ss@localhost:27017/?authSource=admin& replicaSet=r  
sYadax"
```

```
Current Mongosh Log ID: 658b58d334cd751894979114
```

```
Connecting to:
```

```
mongodb://<credentials>@localhost:27017/?authSource=admin&directConnection=t  
rue&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.1
```

```
...
```

```
rsYadax [primary] test>
```



Vamos exportar a connection string - **Agent** - Todos

Isso é importante para o agent rodar corretamente

```
[root@ip-20-0-1-154 ~]# cat /etc/sysconfig/pbm-agent
```

```
PBM_MONGODB_URI="mongodb://pbmuser:password@localhost:27017/  
?authSource=admin&replicaSet=rsYadax"
```



Vamos exportar a connection string - Client - Todos

Isso é importante sempre que você for chamar o client. Por isso pode exportar em um `.bash_profile` ou algo assim

```
[root@ip-20-0-1-154 ~]# export  
PBM_MONGODB_URI="mongodb://pbmuser:password@localhost:27017/  
?authSource=admin&replicaSetName=rsYadax"
```



Arquivo de config - Em qualquer node

Se for um EC2 que possui Instance Role Profile:

```
[root@ip-20-0-1-154 ~]# vi pbm_config.yaml
storage:
  type: s3
  s3:
    region: us-east-1
    bucket: medusa-backup
    prefix: data/pbm/backup/<seunome>
```

Se não:

```
[root@ip-20-0-1-154 ~]# vi pbm_config.yaml
storage:
  type: s3
  s3:
    region: us-east-1
    bucket: medusa-backup
    prefix: data/pbm/backup/seunome
    credentials:
      access-key-id: <your-access-key-id-here>
      secret-access-key:
        <your-secret-key-here>
```



Upload Arquivo de config - Em qualquer node

```
[root@ip-20-0-1-154 ~]# pbm config --file pbm_config.yaml
```

```
> use admin
```

```
> db.pbmConfig.find()
```



Start agent - Todos os nodes

```
sudo systemctl start pbm-agent
```

```
sudo systemctl status pbm-agent
```

CLI Options

```
export PBM_MONGODB_URI="mongodb://pbmuser:password@localhost:27017/?authSource=admin&replSetName=rsYadax"
```

```
[root@ip-20-0-1-154 ~]# pbm config --list
```

```
pitr:
```

```
  enabled: false
  oplogSpanMin: 5
  compression: s2
```

```
storage:
```

```
  type: s3
  s3:
    provider: aws
    region: us-east-1
    bucket: medusa-backup
    prefix: data/pbm/backup
    credentials: {}
    maxUploadParts: 10000
    storageClass: STANDARD
    insecureSkipTLSVerify: false
```



CLI Options - Backup

```
pbm backup
```

```
--type: physical, logical (default)
```

```
--compression: gzip, snappy, lz4, s2 (default), pgzip, zstd.
```

```
--compression-level: 0 a 10
```

```
--wait
```

```
pbm cancel-backup (running backup)
```



CLI Options - Restore

```
pbm restore
```

```
--time=
```

```
-w
```

```
--base-snapshot
```

```
--replset-remapping
```

CLI Options - List

```
pbm list
```

```
--restore
```

```
--size=N
```

```
--unbacked
```

CLI Options - Delete Backup

```
pbm delete-backup  
--older-than=TIMESTAMP  
%Y-%M-%DT%H:%M:%S (2020-04-20T13:13:20)  
%Y-%M-%D ( 2020-04-20)  
--force
```

```
pbm delete-pitr  
-a, --all  
--older-than=TIMESTAMP  
--force
```



CLI Options - Version

```
pbm version
```

```
--short
```

```
--commit
```

CLI Options - Status

```
pbm status
```

```
-s, --sections=SECTIONS
```

```
Supported values: cluster, pitr, running, backups
```

CLI Options - Logs

```
pbm logs
-t, --tail=20
-e, --event=EVENT
    - backup
    - restore
    - resyncBcpList
    - pitr
    - pitrestore
    - delete
-n, --node=NODE
-s, --severity=I - D - Debug, I - Info (default), W - Warning, E - Error, F
  - Fatal.
-i, --opid=OPID
-x, --extra
```



Importar alguns dados - No Primary

```
# yum install -y git

# git clone https://github.com/mcampo2/mongodb-sample-databases.git

# cd mongodb-sample-databases

# mongoimport --drop --host localhost --port 27017 \
--username 'admin' --password 'password' \
--authenticationDatabase 'admin' \
--db sample --collection airbnb \
--file sample_airbnb/listingsAndReviews.json
```



LAB

Faça um backup:

```
pbm backup
```

LAB

Liste os backups:

```
pbm list
```

LAB

Habilite o PITR:

```
pbm config --set pitr.enabled=true
```

LAB

Configure o intervalo de coleta do PITR para 1 min:

```
pbm config --set pitr.oplogSpanMin=1
```

LAB

Liste novamente os backups:

```
pbm list
```

LAB

Se o PITR estiver habilitado, drope o sample.airbnb.

```
> use sample  
> db.airbnb.drop()
```

Restaure o backup para um ponto anterior ao drop dentro do range do PBM.

```
pbm config --set pitr.enabled=false  
pbm restore --time="2022-09-20T13:20:00"
```

Volte o PITR e faça um novo backup

```
pbm config --set pitr.enabled=true
```

```
pbm backup
```

Performance de queries

Index

- O que são?
- Para que servem?
- Quando usar?

Index

A

addictions 35, 39
the Arctic 68, 69

B

the bar 48, 68, 69, 75, 81, 89, 90, 132, 149
bush tips 36, 38, 52, 57, 58, 65, 66, 79, 129
business 34, 39, 48, 76, 77, 116, 128, 132, 145, 156

C

city life 41, 54, 72, 74, 98, 108, 116, 135, 146, 147

D

dog's life 12, 16, 21, 22, 24, 25, 32, 40, 43, 53, 57, 61, 62, 66, 71, 75, 86, 87, 89, 91, 92, 95, 96, 98, 100, 101, 102, 103, 112, 115, 125, 126, 129, 144, 152, 160

E

the environment 26, 33, 62, 77, 78, 82, 85, 121, 124, 128, 134, 136, 137, 138, 139, 147, 148, 149, 150, 151, 152, 160, 163
explorers 17, 28, 29, 77, 114

F

fall 13, 83
fishing 6, 9, 10, 19, 46, 78, 127, 157, 159

flying 22, 23

G

government 12, 17, 25, 43, 49, 57, 63, 68, 70, 76, 79, 80, 99, 114, 126, 127, 133, 140, 143, 146, 151, 153, 155

grouse 24, 27, 163

guiding 17, 61, 86, 122, 128, 136, 158

H

health 30

hippies 15, 35, 36, 37, 88

horses 47, 122, 144

hunting 14, 31, 32, 46, 61, 67, 70, 72, 85, 86, 106, 107, 124, 127, 141, 151, 161, 162

K

kids 24, 30, 58, 59, 74, 123, 137

M

marriage 23, 33, 36, 49, 52, 56, 60, 66, 68, 71, 75, 76, 89, 90, 96, 97, 100, 102, 124, 141, 154, 157

mechanics 18, 66, 87, 145

money 32

mushing 1, 2, 3, 4, 5, 9, 10, 18, 20, 21, 34, 37, 43, 53, 55, 73, 74, 75, 89, 94, 98, 109, 112, 150

O

oldtimers 43, 93, 107, 126, 156, 158

P

philosophy 94, 102, 108, 109, 113, 115, 116, 121, 123, 130, 132, 134, 142, 144, 146, 152, 154, 155, 156, 157, 159, 161, 162, 164

police 11, 13, 27, 34, 44

politics 6, 44, 56, 67, 71, 73, 125, 139, 140, 142

prospecting 18, 29, 31, 64, 104, 123, 130

punks 16, 19, 47, 108, 130

R

ravens 21, 30, 47, 67, 71, 95, 104, 124, 134, 137, 138, 139, 140, 165

religion 7, 38, 45, 58

romance 4, 5, 16, 23, 35, 42, 50, 52, 55, 56, 71, 114, 145

S

scientists 11, 25, 28, 38, 63, 65, 67, 90, 120, 122, 127, 135, 137, 138, 139, 140, 150, 155, 163, 164
snowmachines 20, 149

southerners 49, 60, 62, 63, 72, 74, 76, 80, 81, 82, 99, 101, 103, 113, 120, 122, 131, 133, 136, 142, 146, 152, 153, 154, 158, 159, 160
summer 10, 27, 45, 64, 157

T

tourism 8, 21, 26, 33, 41, 44, 45, 46, 64, 80, 81, 83, 120, 125, 133, 135, 143, 158

tradition 84, 98

trapping 14, 19, 32, 56, 60, 72, 92, 94, 97, 107, 109, 112, 113, 128, 153, 165

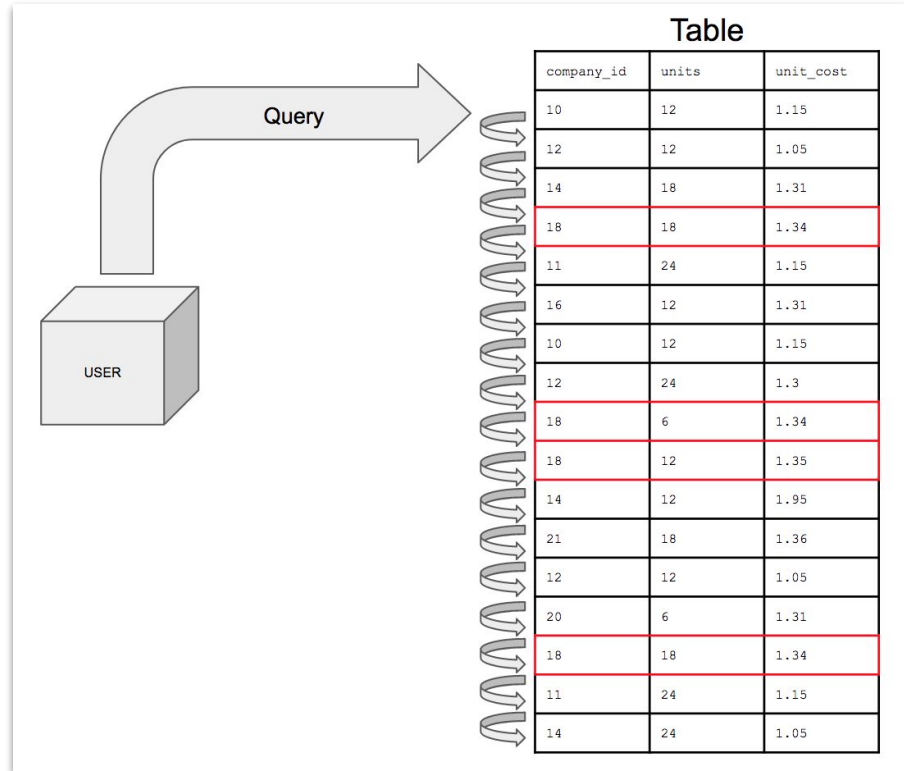
W

weather 20, 50, 93, 101, 115
winter 50, 51, 107

Collection Scan (COLLSCAN)

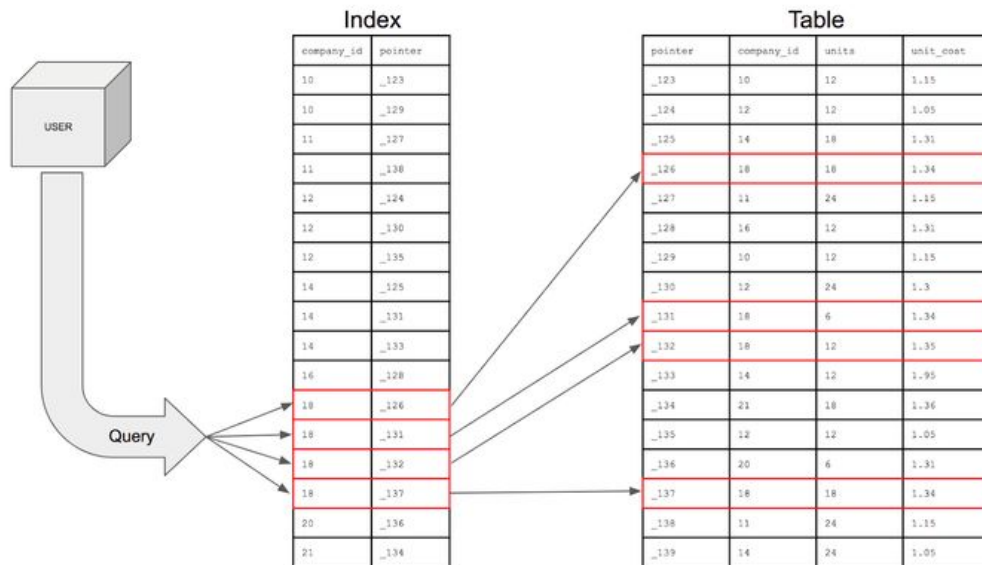
- Similar a Full Table Scan
- Varre todos os documentos
- Deve ser evitado para collections grandes

```
db.coll.find({company_id: 18})
```



Index

- O índice é ordenado, b-tree
- Cada entrada aponta para a posição do doc na collection
- Collections podem ter mais de um índice em diferentes fiels
- `_id` é indexado por default
- Não crie mais índices do que você necessita
- Há impacto na escrita



Index

- `db.collection.createIndex({field: 1})`
- field pode ser um dentro de um subdocumento (`endereco.cidade`)
- 1 -> ASC
- -1 -> DESC
- Single doc scan
- Range doc scan
- Múltiplos docs (IN)

Index

- in background mongodb 4.2-
- unique
- partial
- expireAfterSeconds (TTL)
- hidden

Importar alguns dados

```
# yum install -y git

# git clone https://github.com/mcampo2/mongodb-sample-databases.git

# cd mongodb-sample-databases

# mongoimport --drop --host localhost --port 27017 --username 'admin' \
--password 'password' --authenticationDatabase 'admin' \
--db sample --collection airbnb --file sample_airbnb/listingsAndReviews.json

--db sample --collection companies --file sample_training/companies.json
--db sample --collection grades --file sample_training/grades.json
```



Criar alguns dados

```
> use sample
> for (i = 0; i < 10000000; i++) {
  db.users.insertOne(
    {
      "userid": i,
      "username": "user"+i,
      "age": Math.floor(Math.random()*120),
      "created": new Date()
    });
}
```

Execution Plan

Sequência de passos que o DB irá executar para retornar o resultado

Uma query pode ter mais de um Plano (Winner / Rejecteds)

Normalmente DBs mantêm cache de Plano

Normalmente DBs mostram o plano que a query será executada

Query Planner

Execution Stats:

```
db.collection.find().explain('executionStats')
```

nReturned

totalKeysExamined

totalDocsExamined

executionStages: stage

Explain Methods:

- queryPlanner (sem executar)
- **executionStats (executa a query)**
- allPlansExecution (mais verboso)



Slow Queries

```
> use sample;
```

```
> db.users.find({"username": "user101"})
```

```
> db.companies.find({email_address: 'info@wetpaint.com'})
```

Query Planner

```
db.users.find({"username":  
"user101"}).explain("executionStats")
```

```
db.companies.find({email_address:  
'info@wetpaint.com'}).explain('executionStats')
```



Slow Queries - Profiles

```
use sample
```

```
db.setProfilingLevel(1, { slowms: 50 })
```

- Level 0 - The profiler is off and does not collect any data. This is the default profiler level.
- Level 1 - The profiler collects data for operations that take longer than the value of slowms.
- Level 2 - The profiler collects data for all operations.



Slow Queries

```
> use sample
```

```
db.users.find({"username": "user1401"})
```

```
db.companies.find({email_address: 'info@wetpaint.com' })
```

```
db.system.profile.find({ millis: { $gt: 60 }, "op": "query" })
```

Busca os docs com mais de 20 ms de execução



Query Planner

```
db.users.find({"username":  
"user101"}) .explain("executionStats")
```

```
db.companies.find({email_address:  
'info@wetpaint.com'}) .explain('executionStats')
```



Create Index

```
>use sample
```

```
>db.users.createIndex({"username": 1})
```

```
>db.companies.createIndex({"email_address":1})
```

```
>db.users.getIndexes()
```

```
>db.companies.getIndexes()
```



Query Planner

```
db.users.find({"username":  
"user101"}).explain("executionStats")
```

```
db.companies.find({email_address:  
'info@wetpaint.com'}).explain('executionStats')
```



Compound index

```
db.airbnb.find({"host.host_name": 'Ynaie', "address.country":  
"Brazil"}).explain("executionStats")
```

```
db.airbnb.createIndex({ 'host.host_name': 1, "address.country": 1})
```

```
db.airbnb.find({"host.host_name": 'Ynaie'}).explain("executionStats")
```

```
db.airbnb.find({"address.country": "Brazil"}).explain("executionStats")
```

```
db.airbnb.find({"host.host_name": 'Ynaie', "address.country":  
"Brazil"}).explain("executionStats")
```



Shards

Scale

- ReplicaSet pode suportar um grande volume de dados
- Mas é apenas uma máquina primary e as outras são cópias
- Se o volume de dados cresce podemos:
 - Aumentar o disco
 - Aumentar a RAM
 - Aumentar CPU
- Escalar Vertical

Scale

- Mas e se ainda assim um host não for suficiente para suportar o volume de dados?
- Precisamos escalar horizontal
- É aí que entra o sharding, distribuímos os dados em várias máquinas
- A arquitetura fica um pouco mais complexa, mas o crescimento é ilimitado

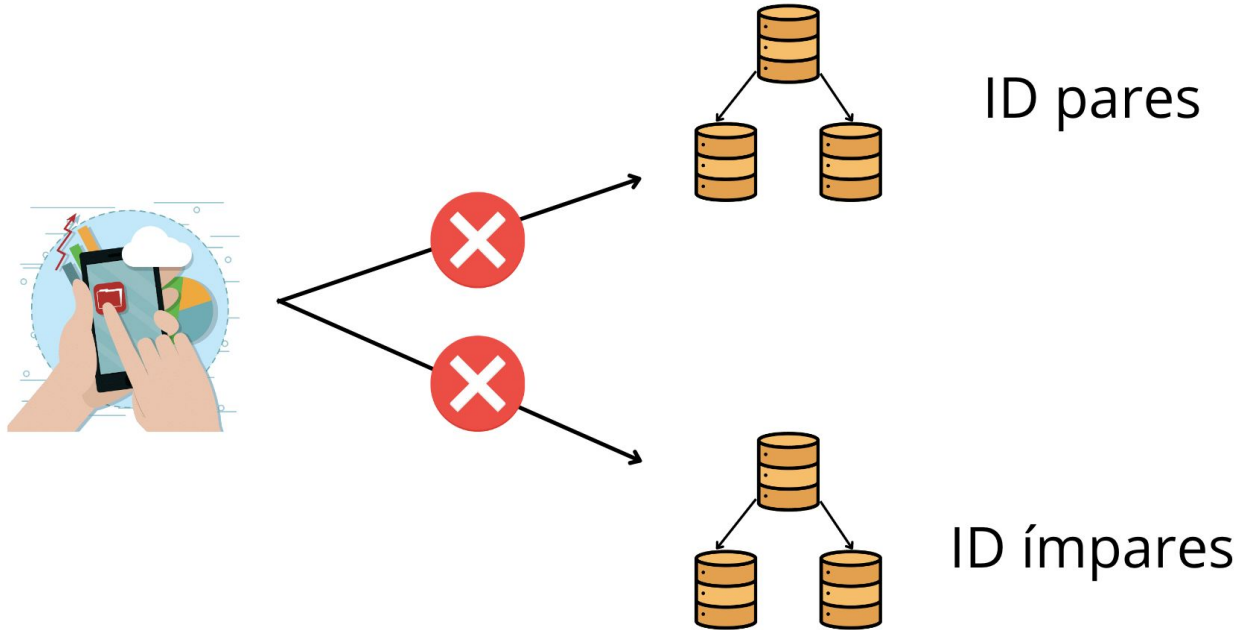


Scale - ReplicaSet (Shard)

- Cada shard será um ReplicaSet, para manter a alta disponibilidade
- Cada shard será responsável por parte dos dados
 - Para simplificar, imagine um shard responsável por ID pares e outro pelos ímpares
- Como a app sabe onde buscar o dado?
 - Aqui entra uma camada de router que recebe a query e encaminha para o shard correto



Scale - Para onde enviar a requisição

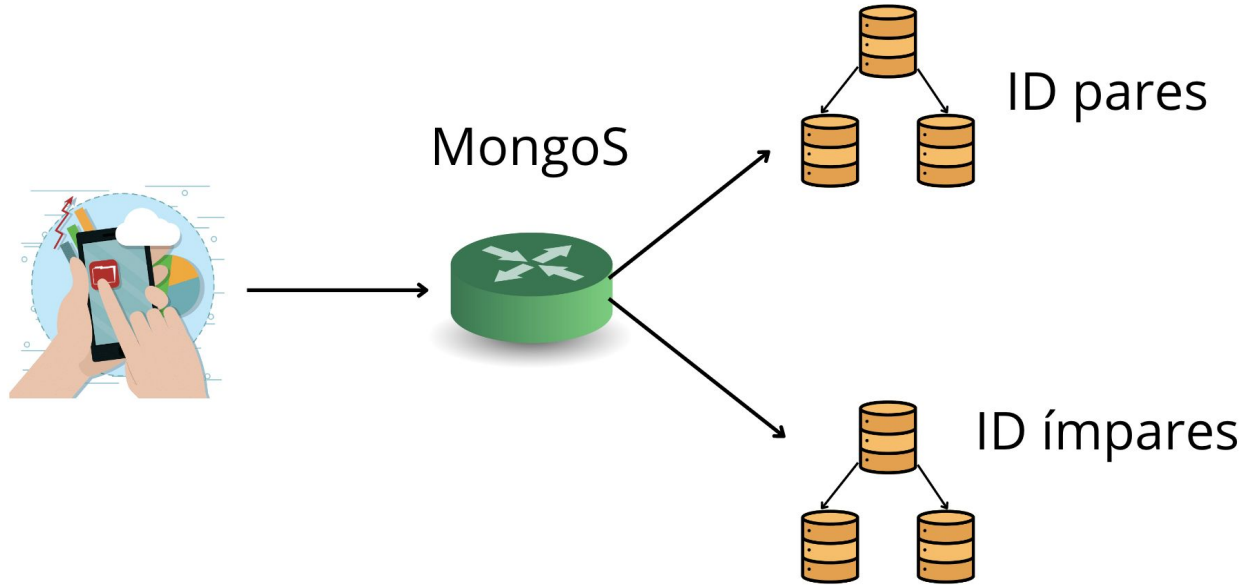


Scale - MongoS (ou router)

- Esse router é o MongoS
- Você pode ter MongoS do lado do DB ou junto a App, quantos quiser.
- O MongoS não armazena dados
- O ConfigServer fornece os metadados para o MongoS
- Assim o MongoS sabe para onde enviar as requisições



Scale - Papel do MongoS



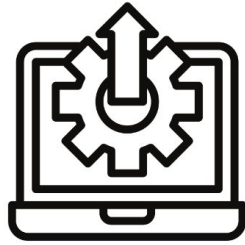
Scale - ConfigServer

- O ConfigServer também é um ReplicaSet, para manter a alta disponibilidade
- O ConfigServer não pode ter árbitros
- O ConfigServer não é acessado pela App diretamente
- No ConsigServer teremos armazenados as chaves e os respectivos shards

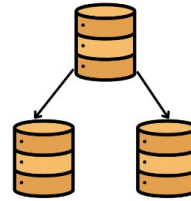


Scale - ConfigServer

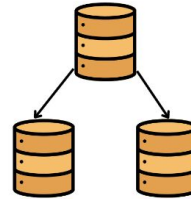
ConfigServer



Shard	Dados
Shard #1	Pares
Shard #2	Ímpares



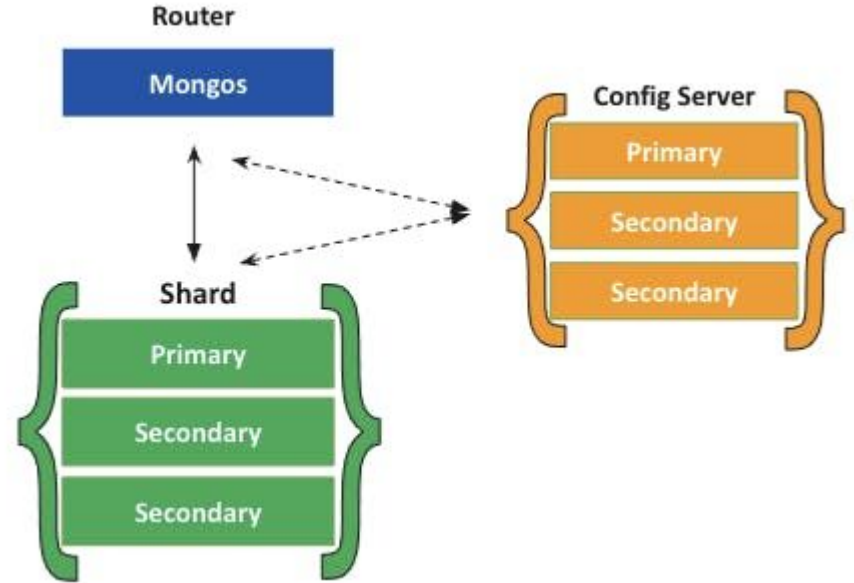
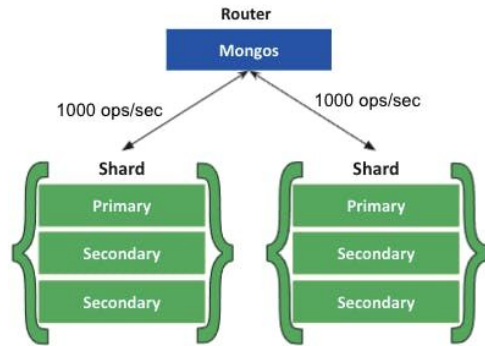
ID pares



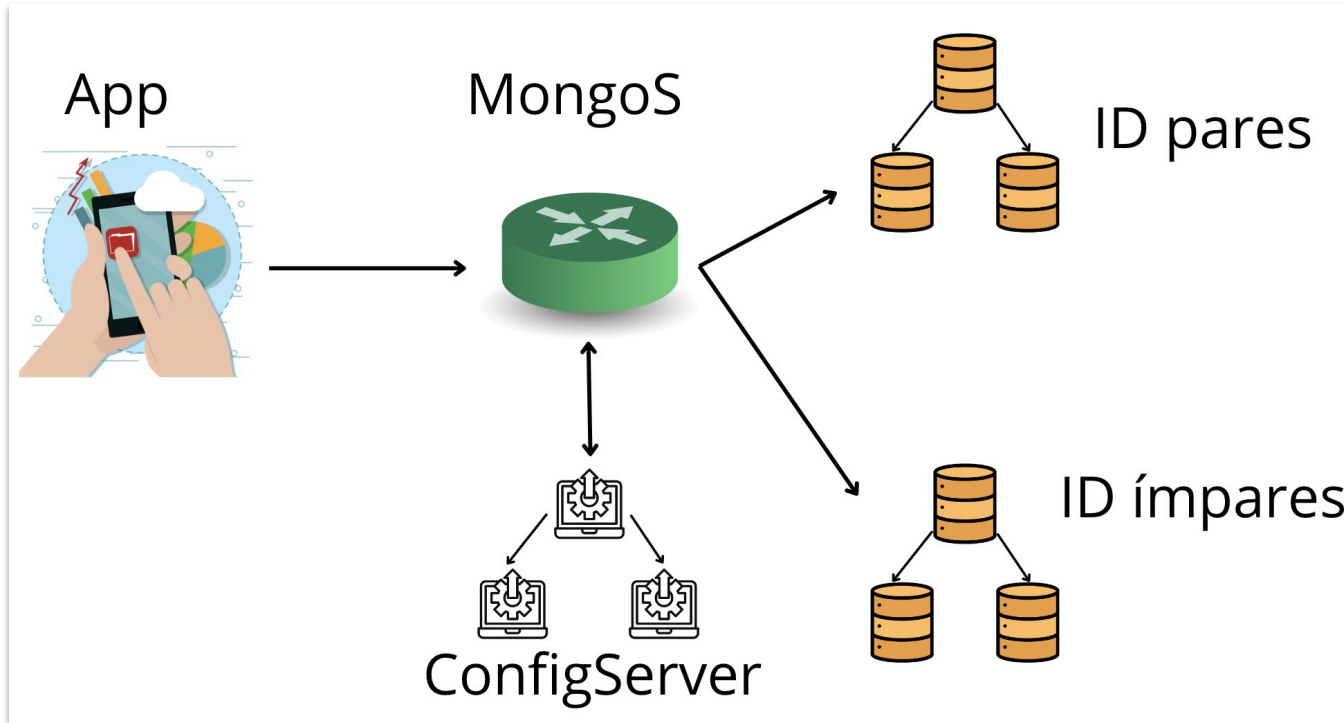
ID ímpares

Scale

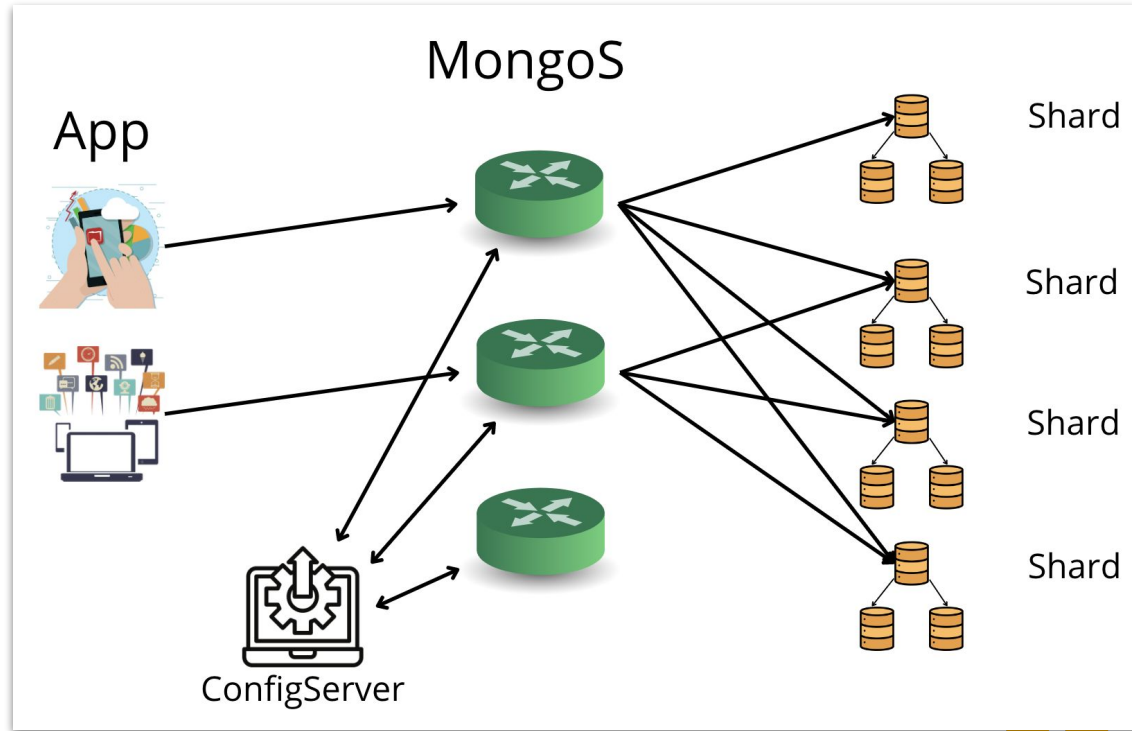
- Shard: Dados distribuídos e replicados
- ConfigServer: Metadados centralizado
- MongoS: serve a app direcionando as requisições para os shards corretos



Arquitetura Sharding



Arquitetura Sharding

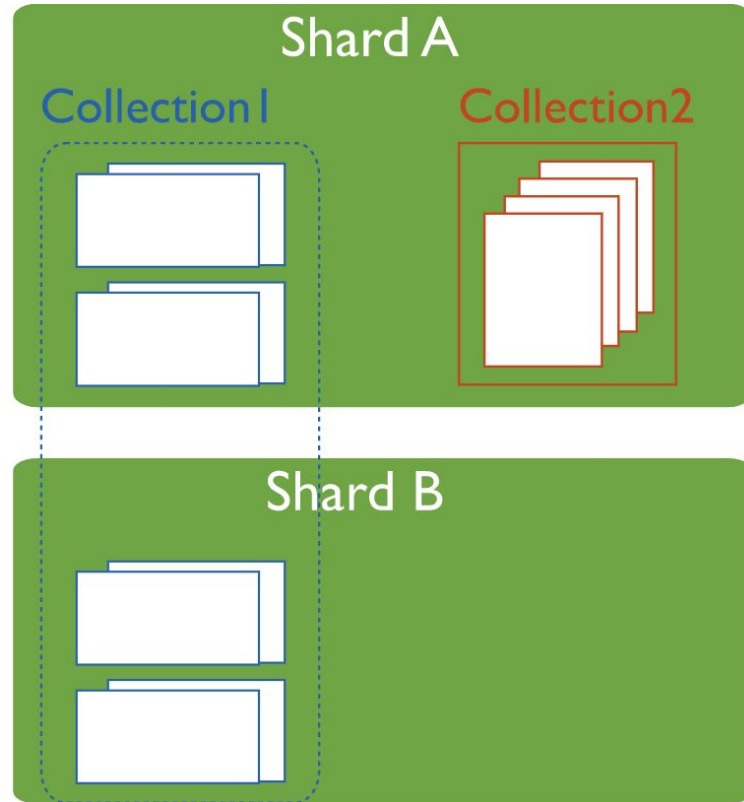


Scale

- Principais indicadores que precisamos de shard:
 - Volume
 - Throughput
 - Performance
 - Tempo Backup/Restore

Primary Shard

- O DB é shardeado
- Nem toda collection do DB precisa ser shardeada
- As collections não shardeadas residem no Primary Shard
- É possível mudar o primary shard
- DBs diferentes podem ter diferentes Primary Shard



Host	Papel	Porta
host0	cfg	27017
host1	cfg	27017
host2	cfg	27017
host3	shard1	27017
host4	shard1	27017
host5	shard1	27017
host6	shard2	27017
host7	shard2	27017
host8	shard2	27017
host9	mongos	27017

LAB: Criar o ConfigServer



Se seu mongo está rodando

Nos hosts 0, 1 e 2:

```
systemctl stop mongod
```

```
rm -rf /var/lib/mongo/*
```

Edite seu arquivo de conf:

```
vi /etc/mongod.conf
```



Instalar o MongoDB

```
# vi /etc/yum.repos.d/mongodb.repo
[mongodb-org-5.0]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mong
odb-org/5.0/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-5.0.asc

# sudo yum install -y mongodb-org
# setenforce 0
```



Setup Sharding Cluster - ConfigServer

- O primeiro passo será criar nosso ConfigServer
- Ele é um ReplicaSet, mas precisamos informar ao MongoDB que se trata de um RS especial
- Vamos incluir o parâmetro "sharding" no arquivo de config
- A role será configsvr

host0, host1 e host2

```
cp /root/mongod_cfg.conf /etc/mongod.conf
```

```
[root@ip-20-0-1-65 ~]# cat /etc/mongod.conf | egrep -v '^$|^$'
```

```
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log
storage:
  dbPath: /var/lib/mongo
  journal:
    enabled: true
processManagement:
  timeZoneInfo: /usr/share/zoneinfo
net:
  port: 27017
  bindIp: 0.0.0.0 # Enter 0.0.0.0,:: to bind to all IPv4 and IPv6 addresses or, alternatively, use the net.bindIpAll setting.
security:
  authorization: enabled
  keyFile: /etc/keyfile.mongo
replication:
  replSetName: rsConfigServer
sharding:
  clusterRole: configsvr
[root@ip-20-0-1-65 ~]#
```



ConfigSvr config file

```
root # cat /etc/mongod.conf
...
# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# Where and how to store data.
storage:
  dbPath: /var/lib/mongo
  journal:
    enabled: true

# how the process runs
processManagement:
  fork: true # fork and run in
background
  pidFilePath:
/var/run/mongodb/mongod.pid
  timeZoneInfo: /usr/share/zoneinfo
```

```
# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0

security:
  authorization: enabled
  keyFile: /etc/keyfile.mongo

sharding:
  clusterRole: configsvr

replication:
  replSetName: rsConfigServer
```

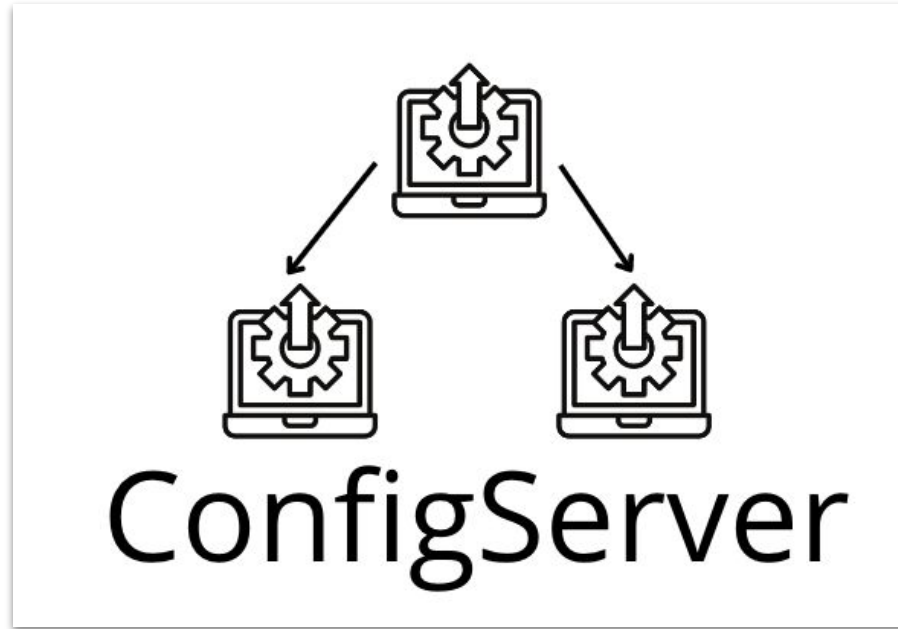


Criar diretório, arquivo de autenticação, arquivo service

```
# echo "qualquercoisabase64" >> /etc/keyfile.mongo  
# chown mongod: /etc/keyfile.mongo  
# chmod 400 /etc/keyfile.mongo  
# setenforce 0
```



Repita isso nos 3 hosts



Iniciar o ReplicaSet

O ConfigServer é um ReplicaSet

Starte o mongod nos três hosts

Vamos até o primeiro node e iniciá-lo

Em seguida, criar um user admin e se autenticar

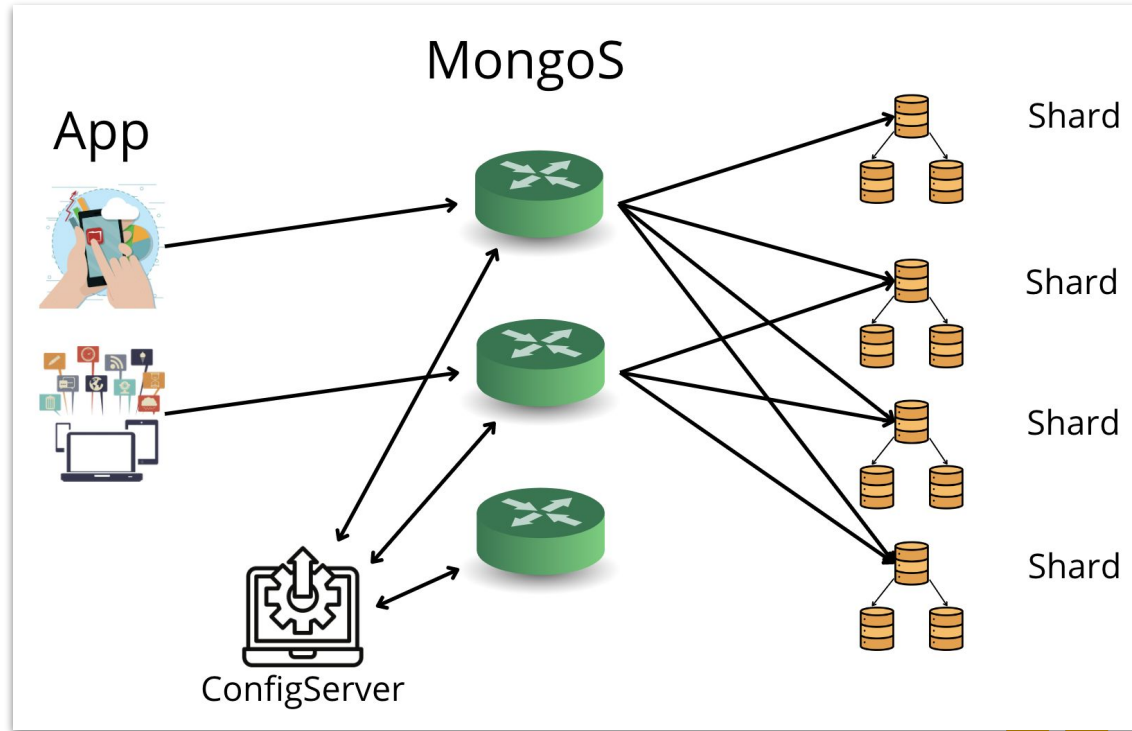
Vamos adicionar os outros membros no ReplicaSet

Confira o status do ReplicaSet e garanta que todos os nodes estejam UP



LAB: Criar o MongoS

MongoS



MongoS

O MongoS tem a estrutura mais simples, não armazena dados, não é responsável por autenticação

Depende do ConfigServer, a connection string é informada no .conf

Pode ficar do lado do DB ou do lado da APP

O binário que starta o MongoS é o /usr/bin/mongo**s**

systemctl status mongod

Neste arquivo /usr/lib/systemd/system/mongod.service

ExecStart=/usr/bin/mongos** \$OPTIONS**

systemctl daemon-reload



MongoS - Conf

```
# cp /root/mongod_router.conf /etc/mongod.conf
```

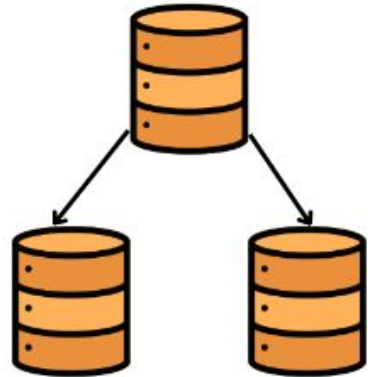
```
[root@ip-20-0-1-20 ~]# cat /etc/mongod.conf | egrep -v '^$|^#'
```

```
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log
processManagement:
  timeZoneInfo: /usr/share/zoneinfo
net:
  port: 27017
  bindIp: 0.0.0.0 # Enter 0.0.0.0,:: to bind to all IPv4 and IPv6 addresses o
r, alternatively, use the net.bindIpAll setting.
security:
  keyFile: /etc/keyfile.mongo
sharding:
  configDB: rsConfigServer/ip-20-0-1-65.ec2.internal:27017,ip-20-0-1-31.ec2.in
ternal:27017,ip-20-0-1-51.ec2.internal:27017
[root@ip-20-0-1-20 ~]# █
```

Vamos fazer o mesmo que o
ConfigServer para os 3 nodes do
shard1

Configurar o shard1, nos hosts 3, 4 e 5

- Arquivo `/etc/mongod.conf`
- Arquivo de autenticação `keyfile.mongo`
- `setenforce 0`
- `systemctl start mongod`
- ReplicaSet: Initiate + create user + Add nodes



Configurar o shard

```
# cp /root/mongo_sh1.conf /etc/mongod.conf
```

```
# setenforce 0
```

```
# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# Where and how to store data.
storage:
  dbPath: /var/lib/mongo
  directoryPerDB: true
  journal:
    enabled: true

# how the process runs
processManagement:
  timeZoneInfo: /usr/share/zoneinfo

# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0

security:
  authorization: enabled
  keyFile: /etc/keyfile.mongo

replication:
  replSetName: rsYadax1

sharding:
  clusterRole: shardsvr
```

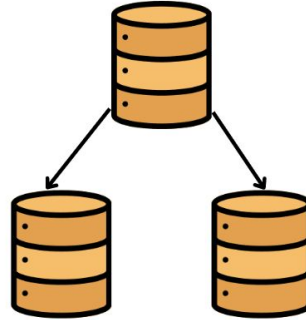
Adicionar o shard na arquitetura

Adicionar shard

MongoS

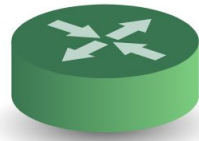


ConfigServer

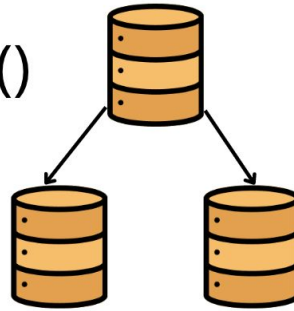
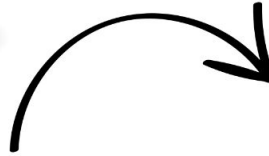


Adicionar shard

MongoS



addShard()



ConfigServer

Adicionar shard

Conectado no MongoS

```
sh.status()
```

```
sh.addShard( "rsYadax1/ip-20-0-1-XX.ec2.internal:27017,...:27017,...")
```

Confira o status do shard:

```
sh.status()
```



Database config

use config

db.databases.find()

db.collections.find()

db.shards.find()

db.chunks.find()

db.mongos.find()

Chunks

Estratégia utilizada pelo MongoDB para distribuir dados

Os dados são distribuídos em pedaços chamados chunks

Os chunks podem ser movidos de um shard para outro

O tamanho default de um chunk é 64M (5.0), 128M (6.0)

Pode ser de 1MB a 1024MB



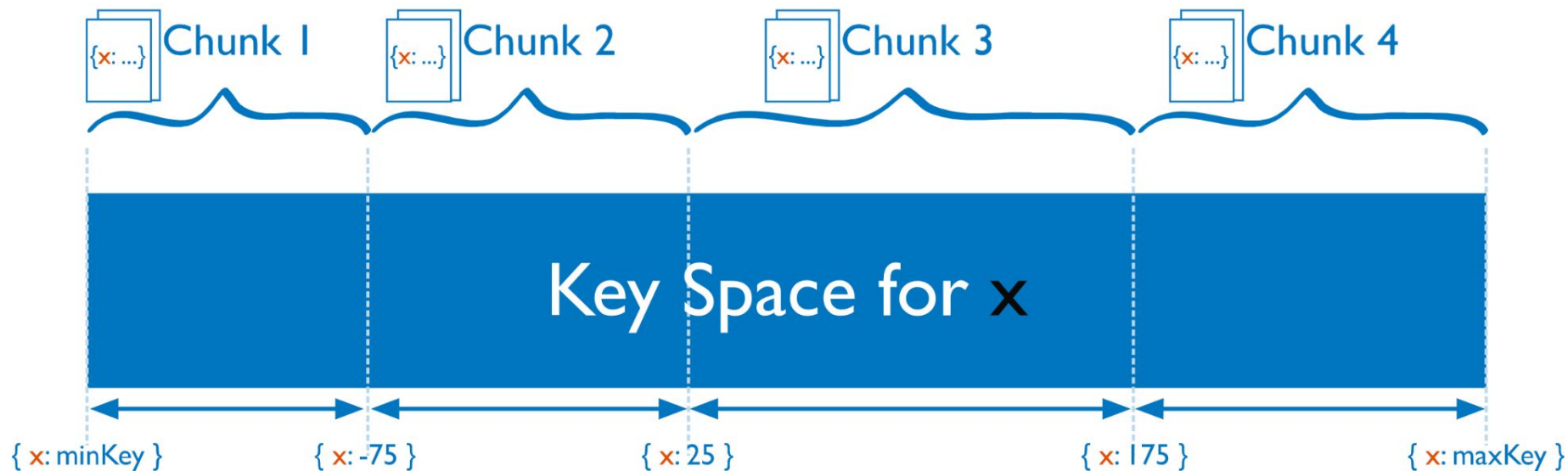
Chunks

Se diminuir, eles serão "splitados"

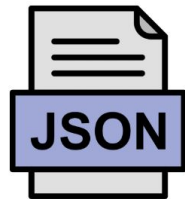
Se aumentar, vão crescer naturalmente

Podem ocorrer splits naturalmente durante insert/update

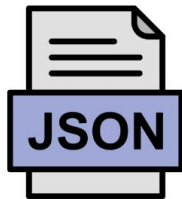
Chunks



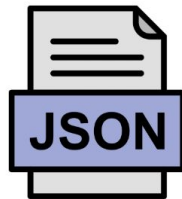
Sem
shard



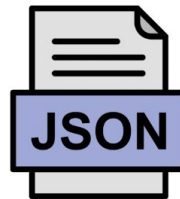
{id: 1}



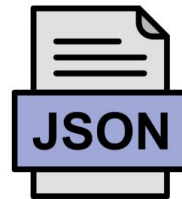
{id: 2}



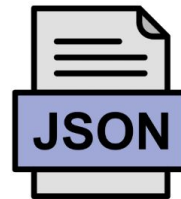
{id: 3}



{id: 4}



{id: 5}



{id: 6}

Com
shard



{id: 1}



{id: 2}

$-\infty < id < 3$



{id: 3}



{id: 4}

$3 \leq id < 5$



{id: 5}



{id: 6}

$5 \leq id < +\infty$

Shard Key

É importante que a shard key esteja presente nas queries para que o MongoS direcione as requisições para o shard correto.

Ela sempre vai ser indexada, antes de ser a shard key

Não use valores com alta frequência ou que sempre aumente

Até o mongo 4.2 não era possível mudar a shard key

A partir do MongoDB 4.4, você podia acrescentar campos a chave

A partir do MongoDB 5.0 você pode mudar a chave.



Vamos importar alguns dados

```
yum install -y git
```

```
git clone https://github.com/mcampo2/mongodb-sample-databases.git
```

```
cd mongodb-sample-databases
```

No **MongoS** (host9)

```
mongoimport --drop --authenticationDatabase=admin --username userRoot  
--password Adm1nP4ss --db sample --collection airbnb --file  
sample_airbnb/listingsAndReviews.json
```



Habilitar shard no DB

Conectado no MongoS

```
sh.status()
```

```
sh.enableSharding('sample')
```

```
sh.status()
```

Não vai habilitar shard nas collections. Elas devem ser habilitadas uma a uma



Database config

use config

db.databases.find()

db.collections.find()

db.shards.find()

db.chunks.find()

db.mongos.find()

Habilitar shard na Collection

Se a shard key for em cima do `_id`, já existe index.

Se não for, precisamos criar um index.

Um collection não pode ser unsharded

É possível usar shard key hashed (apenas um field)

```
use sample
```

```
db.airbnb.createIndex({"field": 1})
```

```
ou db.airbnb.createIndex({"field": "hashed"})
```

```
db.airbnb.getIndexes()
```

```
sh.shardCollection("sample.airbnb", {_id: 1})
```

```
sh.status()
```



Database config

sh.status()

use config

db.databases.find()

db.collections.find()

db.shards.find()

db.chunks.find()

db.mongos.find()



Primary Shard

Como buscar o primary shard?

```
mongos> use config
```

```
mongos> db.databases.find()
```

Balancer

Roda no primary do ConfigServer

Percebe que o número de chunks está desbalanceado e balanceia

O Balancer pode splitar chunks se ele precisar

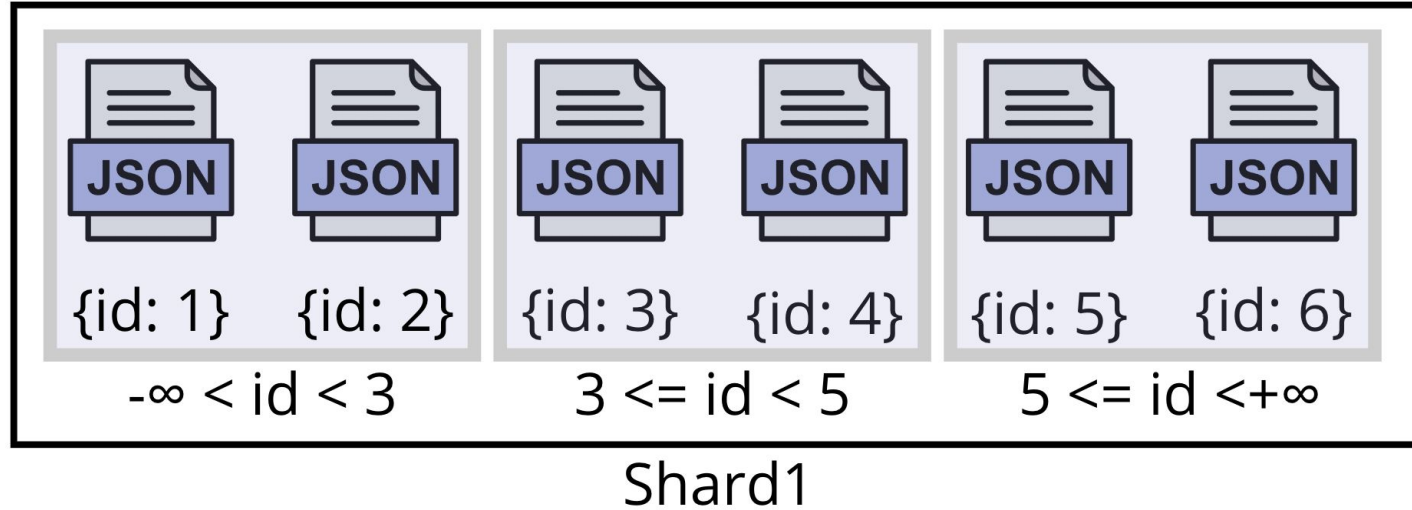
`sh.startBalancer`

`sh.stopBalancer`

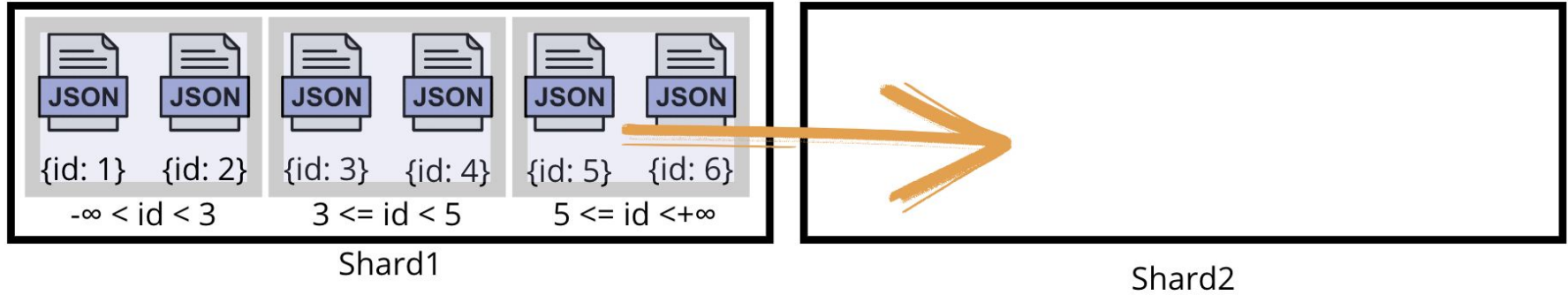
`sh.setBalancerState`

Adicionar Shard no Cluster

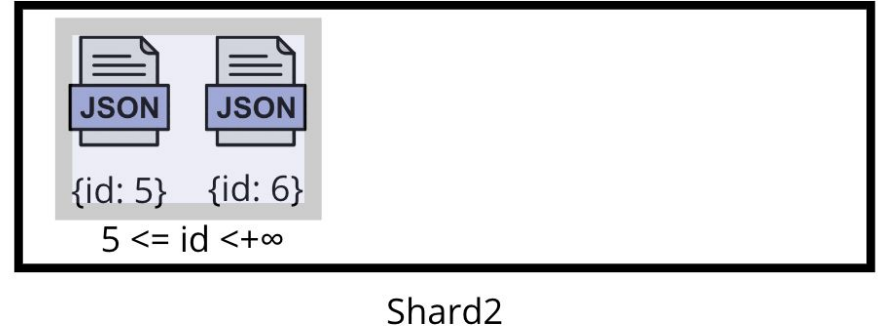
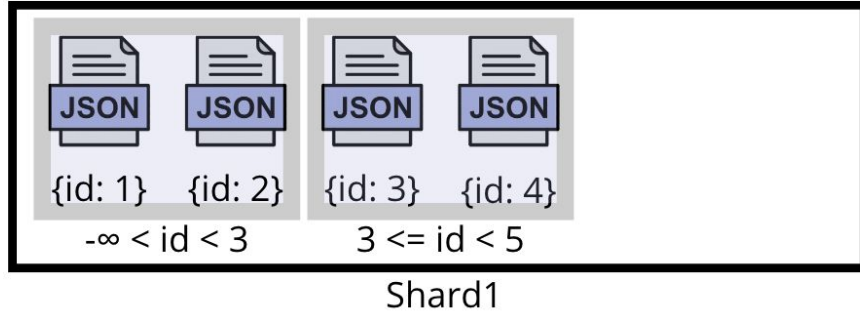
Add shard no cluster



Add shard no cluster



Add shard no cluster



Add shard no cluster

`hosts: 6, 7 e 8`

- Copiar Arquivo .conf /root/mongo_sh2.conf
- Criar keyfile.mongo
- setenforce 0
- Start .service
- ReplicaSet: Initiate + create user + Add nodes

Add shard no cluster

/etc/mongod.conf

```
# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# Where and how to store data.
storage:
  dbPath: /var/lib/mongo
  directoryPerDB: true
  journal:
    enabled: true

# how the process runs
processManagement:
  timeZoneInfo: /usr/share/zoneinfo

# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0

security:
  authorization: enabled
  keyFile: /etc/keyfile.mongo

replication:
  replSetName: rsYadax2

sharding:
  clusterRole: shardsvr
```



Add shard no cluster

```
sh.status()
```

```
sh.addShard(  
"rsYadax2/ip-20-0-1-?? .ec2.internal:27017,ip-20-0-1-?? .ec2.i  
nternal:27017,ip-20-0-1-?? .ec2.internal:27017")
```

```
sh.status()
```

```
wait...
```

```
sh.status()    //O que houve?
```



Reinicie as 10 máquinas

reboot



Remover Shard do Cluster

Remover Shard do Cluster

Processo contrário, mas não é tão online quando add

O shard a ser removido, pode* ser o principal

Precisamos mover o shard principal para outro shard

*pode ser feito antes ou depois, mas não é de fato removido até você movê-lo



Remover shard do cluster

```
db.adminCommand( { listShards: 1 } )
```

```
[direct: mongos] test> db.adminCommand( { removeShard: "rsYadax1" } )  
{  
  msg: 'draining started successfully',  
  state: 'started',  
  ...
```

```
[direct: mongos] test> db.adminCommand( { removeShard: "rsYadax1" } )  
{  
  msg: 'draining ongoing',  
  state: 'ongoing',  
  ...
```



Remover shard do cluster

```
[direct: mongos] test> db.adminCommand( { removeShard: "rsYadax1" } )
{
  msg: 'draining ongoing',
  state: 'ongoing',
  remaining: { chunks: Long("548"), dbs: Long("1"), jumboChunks: Long("0") },
  note: 'you need to drop or movePrimary these databases',
  dbsToMove: [ 'sample' ],
```

O MongoDB lista todos os DBs que precisam ser movidos. Então, vamos movê-lo:

```
[direct: mongos] test> db.adminCommand( { movePrimary: "sample", to: "rsYadax2" })
{
  ok: 1,
  ...
```



Remover shard do cluster

```
[direct: mongos] test> db.adminCommand( { removeShard: "rsYadax1" } )
{
  msg: 'draining ongoing',
  state: 'ongoing',
  remaining: { chunks: Long("548"), dbs: Long("1"), jumboChunks: Long("0")
},
  note: 'you need to drop or movePrimary these databases',
  dbsToMove: [ 'sample' ],
```

```
[direct: mongos] test> db.adminCommand( { movePrimary: "sample", to:
"rsYadax2" })
{
  ok: 1,
  ...
```



Remover shard do cluster

```
sh.status()  
...  
databases  
[  
  {  
    database: { _id: 'config', primary: 'config', partitioned: true },  
    collections: {  
      'config.system.sessions': {  
        shardKey: { _id: 1 },  
        unique: false,  
        balancing: true,  
        chunkMetadata: [  
          { shard: 'rsYadax1', nChunks: 346 },  
          { shard: 'rsYadax2', nChunks: 678 }  
        ],  
      }  
    }  
  }  
]
```



Remover shard do cluster

```
[direct: mongos] test> db.adminCommand( { removeShard:
"rsYadax1" } )
{
  msg: 'removeshard completed successfully',
  state: 'completed',
  shard: 'rsYadax1',
  ok: 1,

...
sh.status()
```



Upgrade

Pré requisitos - Versão

Treine BACKUP e RESTORE

Para upgrade para a versão 6.0, é necessário que o MongoDB esteja na 5.0

Caso esteja em versões anteriores, para o upgrade:

- 3.x -> 4.x
- 4.x -> 5.0

Check compatibility changes

- <https://www.mongodb.com/docs/upcoming/release-notes/6.0-compatibility/>



Pré requisitos - Driver

- Consulte a compatibilidade do driver que você utiliza
- <https://www.mongodb.com/docs/drivers/java/sync/current/compatibility/>

Java Driver Version	MongoDB 6.0	MongoDB 5.0	MongoDB 4.4	MongoDB 4.2	MongoDB 4.0	MongoDB 3.6	MongoDB 3.4	MongoDB 3.2	MongoDB 3.0
4.7	✓	✓	✓	✓	✓	✓	✓	✓	✓
4.6	⊗	✓	✓	✓	✓	✓	✓	✓	✓
4.5	⊗	✓	✓	✓	✓	✓	✓	✓	✓
4.4	⊗	✓	✓	✓	✓	✓	✓	✓	✓

Pré Requisitos - Rollback

Esteja preparado

Simule o rollback para a versão 5.0

Não é possível fazer rollback depois de mudar o compatible



Pré Requisitos - Compatible

```
rsYadax1 [direct: primary] test> db.adminCommand( {  
getParameter: 1, featureCompatibilityVersion: 1 } )
```

...

```
featureCompatibilityVersion: { version: '5.0' }
```

Caso não seja 5.0:

```
db.adminCommand( { setFeatureCompatibilityVersion: "5.0" } )
```



Upgrade MongoDB 5.0 para 6.0

Upgrade

- Alterar o binário não vai impactar os dados
- Matemos os dados e fazemos o upgrade do binário.
- Usamos yum/apt (gerenciador de pacotes) ou tarball

Standalone

Verificar compatible

```
test> db.adminCommand( { getParameter: 1, featureCompatibilityVersion: 1 } )

{ featureCompatibilityVersion: { version: '5.0' }, ok: 1 }
```

Upgrade

- Verificar se o Compatible está em 5.0
- Fazer backup dos arquivos de conf
- Stop mongod
- Remover binário antigo
- Instalar binário novo
- Startar o mongod (usando o config file antigo)

Ligar test connection

```
from time import sleep
from pymongo import MongoClient
from datetime import datetime
from pymongo.errors import ServerSelectionTimeoutError

time_up = 0
time_down = 0
time_sleep = 10
while True:
    try:
        client = MongoClient('20.0.1.223',serverSelectionTimeoutMS=500) //standalone
        mydb = client["yadax"]
        mycoll = mydb["testConn"]
        mydoc = {"now": datetime.now().strftime("%d/%m/%Y, %H:%M:%S")}
        x = mycoll.insert_one(mydoc)
        time_up = time_up + time_sleep
        time_down = 0
        client.close()
        print("Up: " + str(time_up) + " segs")
    except ServerSelectionTimeoutError:
        time_up = 0
        time_down = time_down + time_sleep
        print("Down: " + str(time_down) + " segs")
        sleep(time_sleep)
```



ReplicaSet

Upgrade ReplicaSet

- Online
- Mesmo procedimento, mas começando pelos secondaries
- Verificar se o Compatible está em 5.0
- Fazer backup dos arquivos de conf
- Stop mongod
- Remover binário antigo
- Instalar binário novo
- Startar o mongod (usando o config file antigo)
- Quando finalizar os SECs, no primary faça um `rs.stepDown()`

Vamos criar um MongoDB ReplicaSet

```
# mongosh
rsYadax1 [direct: primary] test> use admin

rsYadax1 [direct: primary] test> db.createUser(
  {
    user: 'admin',
    pwd: 'password',
    roles: [ { role: 'root', db: 'admin' } ]
  }
);
exit;
```



Vamos criar um MongoDB ReplicaSet

```
# mongosh --port 27017 --username admin --password password --authenticationDatabase admin
```

```
rsYadax1 [direct: primary] test> rs.add('ip-20-0-1-20.ec2.internal:27017')
{
  ok: 1,
  ...
}
rsYadax1 [direct: primary] test> rs.add('ip-20-0-1-190.ec2.internal:27017')
{
  ok: 1,
  ...
}
rsYadax1 [direct: primary] test> rs.status()
```



Ligar test connection

```
from time import sleep
from pymongo import MongoClient
from datetime import datetime
from pymongo.errors import ServerSelectionTimeoutError

time_up = 0
time_down = 0
time_sleep = 10
while True:
    try:
        client = MongoClient('mongodb://20.0.1.223:27017,20.0.1.20:27017,20.0.1.190:27017/?replicaSet=rsYadax1',
                               username='admin',
                               password='password',
                               serverSelectionTimeoutMS=500)

        mydb = client["yadax"]
        mycoll = mydb["testConn"]
        mydoc = {"now": datetime.now().strftime("%d/%m/%Y, %H:%M:%S")}
        x = mycoll.insert_one(mydoc)
        time_up = time_up + time_sleep
        time_down = 0
        client.close()
        print("Up: " + str(time_up) + " segs")
    except ServerSelectionTimeoutError:
        time_up = 0
        time_down = time_down + time_sleep
        print("Down: " + str(time_down) + " segs")
        sleep(time_sleep)
```



Fazer o upgrade

Check o compatible

Comece pelos Secondaries

Fazer o upgrade

```
> db.adminCommand( { shutdown: 1 } )
```

```
# systemctl stop mongod
```

```
# atualizar o binario
```

```
# systemctl start mongod
```

```
> rs.status()
```



Fazer o upgrade

Em seguida, faça o switchover

```
> rs.stepDown()
```

E faça o upgrade no último host

No novo primary

```
> db.adminCommand( { setFeatureCompatibilityVersion: "6.0" } )
```



Shard

Vamos criar um MongoDB Shard

ConfigServer

MongoS

Shard

addShard

enableSharding (DB) - sample

shardCollection (Collection) - sample.airbnb



Host	Papel	Porta
host0	cfg	27017
host1	cfg	27017
host2	cfg	27017
host3	shard1	27017
host4	shard1	27017
host5	shard1	27017
host6	shard2	27017
host7	shard2	27017
host8	shard2	27017
host9	mongos	27017

Pre Upgrade

Pare somente o balancer

No mongos (host9)

```
sh.stopBalancer()
```

Vamos ao upgrade - ConfigServer

Comece com o ConfigServer - SECONDARY (host0, 1 e 2)

```
systemctl stop mongod
```

```
yum remove mongodb*
```

```
alterar o /etc/yum.repos.d/mongodb.repo
```

```
yum install -y mongodb-org
```

```
cp /etc/mongod.conf.rpmsave /etc/mongod.conf
```

```
systemctl start mongod
```



/etc/yum.repos.d/mongodb.repo

```
[mongodb-org-6.0]
```

```
name=MongoDB Repository
```

```
baseurl=https://repo.mongodb.org/yum/redhat/8/mongodb-org/6.0/x86_64/
```

```
gpgcheck=1
```

```
enabled=1
```

```
gpgkey=https://www.mongodb.org/static/pgp/server-6.0.asc
```



Vamos ao upgrade - ConfigServer

No PRIMARY: StepDown -> Secondary

```
mongosh> rs.stepDown()
```

```
#systemctl stop mongod
```

```
#yum remove mongodb*
```

```
alterar o /etc/yum.repo.d/mongodb.repo
```

```
#yum install -y mongodb-org
```

```
#cp /etc/mongod.conf.rpmsave /etc/mongod.conf
```

```
#systemctl start mongod
```



Vamos ao upgrade - ConfigServer

Veja o valor do compatible

```
mongosh> db.adminCommand( { getParameter: 1,  
featureCompatibilityVersion: 1 } )
```



Vamos ao upgrade - Shard2 (host6, 7 e 8)

Comece com os Secondaries

```
systemctl stop mongod
```

```
yum remove mongod*
```

```
alterar o /etc/yum.repos.d/mongodb.repo
```

```
yum install -y mongodb-org
```

```
cp /etc/mongod.conf.rpmsave /etc/mongod.conf
```

```
systemctl start mongod
```



Vamos ao upgrade - Shard

No Primary

```
rs.stepDown()
```

```
systemctl stop mongod
```

```
yum remove mongodb*
```

```
alterar o /etc/yum.repos.d/mongodb.repo
```

```
yum install -y mongodb-org
```

```
cp /etc/mongod.conf.rpmsave /etc/mongod.conf
```

```
systemctl start mongod
```



Vamos ao upgrade - MongoDB

Em cada um dos mongoS

```
systemctl stop mongod
```

```
yum remove mongodb*
```

```
alterar o /etc/yum.repos.d/mongodb.repo
```

```
yum install -y mongodb-org
```

```
cp /etc/mongod.conf.rpmsave /etc/mongod.conf
```

Edite o `/usr/lib/systemd/system/mongod.service`

```
ExecStart /usr/bin/mongod -> /usr/bin/mongos
```

```
systemctl daemon-reload
```

```
systemctl start mongod
```



Vamos ao upgrade - MongoDB

```
sh.status()  
  
active mongoses  
  
[ { '5.0.11': 1 } ]
```

Faça o upgrade dos MongoDB, um a um.

```
active mongoses  
  
[ { '6.0.12': 1 } ]
```



Pós upgrade

Habilitar o Balancer

```
[direct: mongos] test> sh.startBalancer()
```

```
[direct: mongos] test> use admin
```

```
[direct: mongos] admin> db.adminCommand( {  
  setFeatureCompatibilityVersion: "6.0" } )  
{  
  ok: 1,  
  ...  
}
```

Obs.: Vai alterar o compatible em cascata



Fim!

Espero que tenha gostado do nosso curso!

Seu feedback verdadeiro é sempre importante.

Siga-nos nas redes sociais.

<https://www.facebook.com/yadaxbr>

<https://www.linkedin.com/company/yadax>

<https://www.instagram.com/yadaxbr/>

Sugira conteúdos de seu interesse



Fim!