

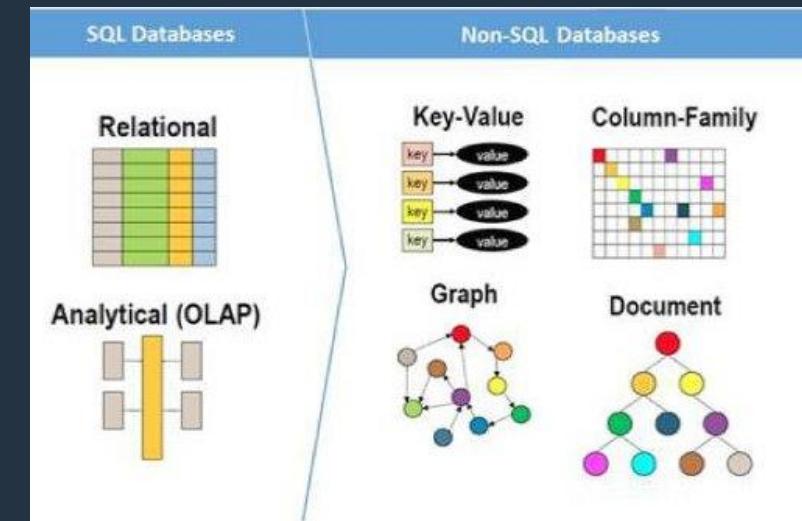
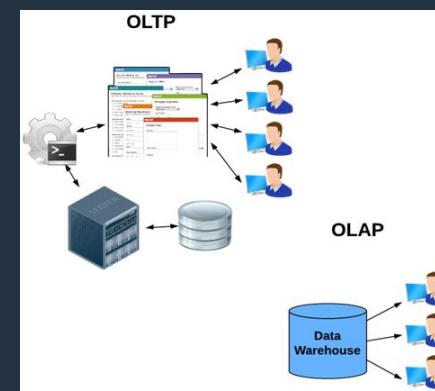


BANCO DE DADOS
NOSQL



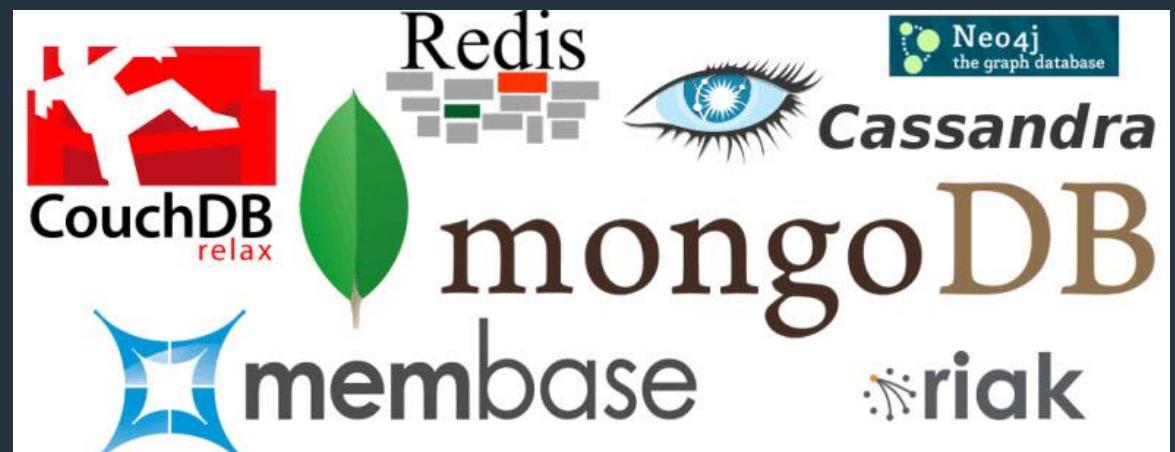
BANCO DE DADOS RELACIONAL

- Os dados são estruturados de acordo com o modelo relacional.
- SQL Server, Oracle, PostgreSQL, MySQL, DB2 e outros.
- Elementos básicos: Relações (tabelas) e Registros (tuplas).
- Características fundamentais:
 - Restrições de integridade
 - PK-primary key, FK-foreign key, UK-unique key, CK-check key, NN-not null.
 - Normalização (formas normais).
 - Linguagem SQL (Structured Query Language)



1. BANCO DE DADOS NOSQL

- NoSQL é um termo genérico que define bancos de dados não-relacionais.
- A tecnologia NoSQL foi iniciada por companhias líderes da internet como Google, Facebook, Amazon, e Linkedin, para superar as limitações de banco de dados relacional para aplicações web modernas.
- Características:
 - Escalabilidade horizontal (add mais nós)
 - Ausência de esquema ou esquema flexível
 - Suporte e Replicação
 - API simples
 - Nem sempre prima pela consistência



PROPRIEDADES DOS BANCOS DE DADOS

- ACID X BASE

Propriedades ACID:

- Atomicidade
- Consistência
- Isolamento
- Durabilidade

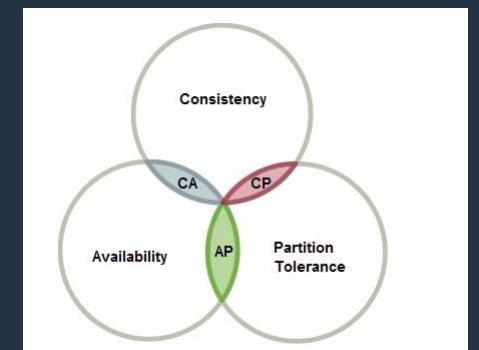
Propriedades BASE:

- Basically Available
- Soft-State
- Eventually Consistent

- Teorema CAP:

- Consistency
- Availability
- Partition Tolerance

De acordo com o teorema CAP, um sistema distribuído de BD somente pode operar com dois desses comportamentos ao mesmo tempo, mas jamais com três simultaneamente.



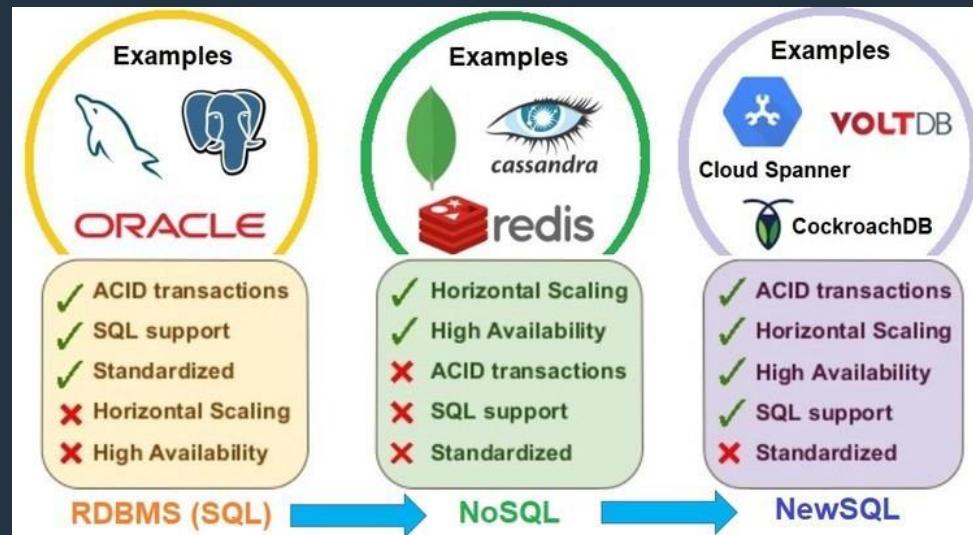
VISÃO GERAL DE NEWSQL

- Definição e visão de NewSQL:

Pode ser definido como uma classe de SGBDs relacionais modernos que buscam fornecer o mesmo desempenho escalonável do NoSQL, para cargas de trabalho OLAP e, simultaneamente, garantir a conformidade ACID para transações como no RDBMS.

Basicamente esses sistemas desejam alcançar a escalabilidade do NOSQL sem ter que descartar o modelo relacional com SQL e suporte a transações do DBMS legado.

- Exemplos de banco de dados NewSQL:



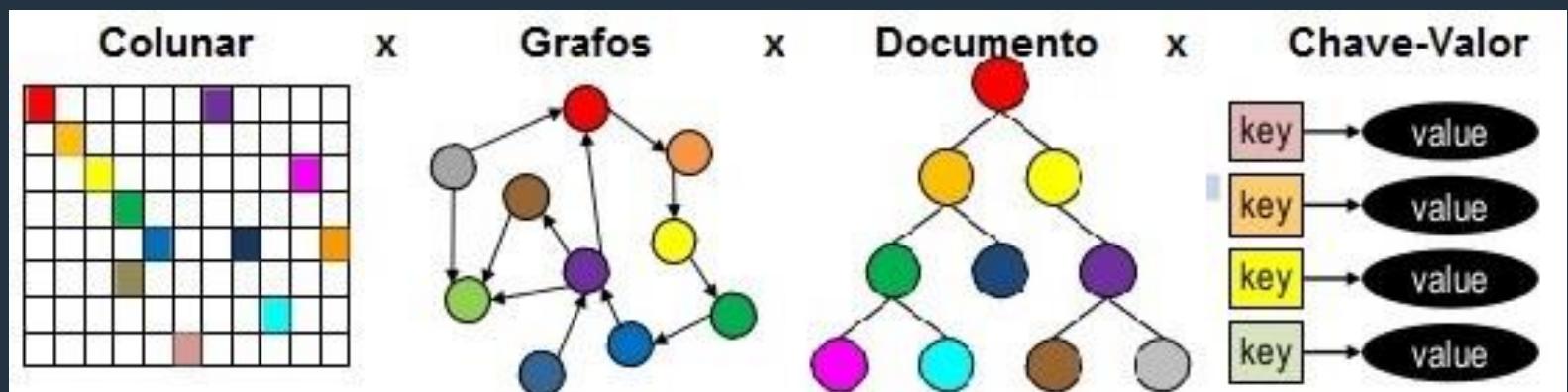
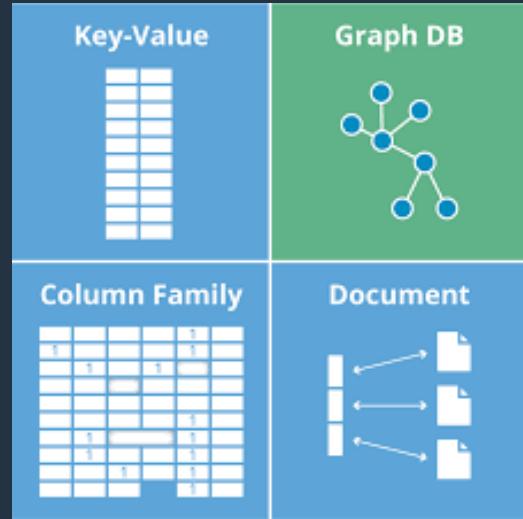
Bancos NewSQL

- MemSQL
- VoltDB
- SQLFire
- MariaDB

	Old SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
SQL	Yes	No	Yes
ACID transactions	Yes	No	Yes
Horizontal scalability	No	Yes	Yes
Performance / big volume	No	Yes	Yes
Schema-less	No	Yes	No

TIPOS DE BANCOS NOSQL

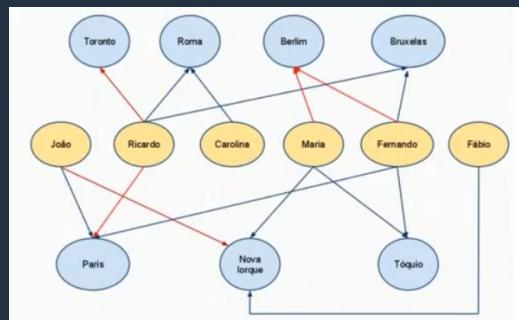
- Temos quatro categorias do NoSQL:
 - Chave-valor (key-Value)
 - Orientado a Grafos
 - Orientado a Coluna (Column Family)
 - Orientado a Documentos



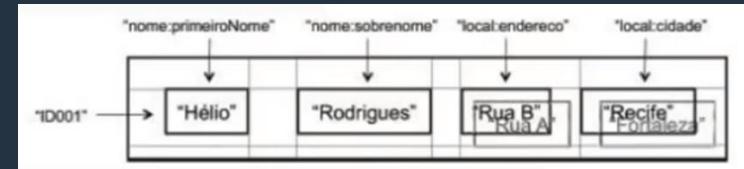
- Chave-valor (Key-Value)

Chave (Campo)	Valor (Instancia)
Nome	Hélio Rodrigues
Idade	45
Sexo	Masculino
Fone	99 99999999

- Orientado a Grafos



- Orientado a Colunas



- Orientado a Documentos

ID: P001 Assunto: "Eu gosto de tomates" Autor: "Mara" Data: 27/01/2011 Tags: ["tomate", "sucu", "plantas"] Mensagem: "Hoje eu estou com vontade de tomar suco de tomate."	ID: P002 Assunto: "Eu gosto de tomates" Autor: "Mara" Data: 15/05/2012 Tags: ["laranjas", "sucu"] Mensagem: "Hoje eu estou com vontade de tomar suco de tomate."
--	---

MOTIVAÇÕES NO USO DE NOSQL

Empresas estão adotando NoSQL para um número crescente de cenários, a escolha que é impulsionada por quatro tendências relacionadas:

- **Big Users:** Um grande número de usuários, combinados com a natureza dinâmica dos padrões de uso está demandando uma tecnologia de BD mais facilmente escalável.
- **Big Data:** é a área do conhecimento que estuda como tratar, analisar e obter informações a partir de conjuntos de dados demasiadamente grandes para serem analisados por sistemas tradicionais. Razões para usar Big Data:
 - Entender padrões
 - Prever situações
 - Criar fronteiras
 - Informar coleções de dados
 - Estimar parâmetros escondidos
 - Calibrar
- **Internet das Coisas:** O termo foi usado pela primeira vez em 1999 para descrever um sistema onde os objetos poderiam ser conectados à internet. 32 bilhões de coisas vão estar conectadas a internet, 10% de todos os dados serão gerados por sistemas embarcados, 21% dos mais valiosos dados serão gerados por sistemas embarcados, e dados de telemetria (semi-estruturados e contínuos) representam um desafio para bancos de dados relacionais. Ex: Cidades Inteligentes: Poluição sonora; otimizar coleta de lixo; controle de tráfego; controle de distribuição de energia elétrica; segurança pública.
- **Cloud Computing:** é um termo coloquial para a disponibilidade sob demanda de recursos do sistema de computador, especialmente armazenamento de dados e capacidade de computação, sem o gerenciamento ativo direto do utilizador.

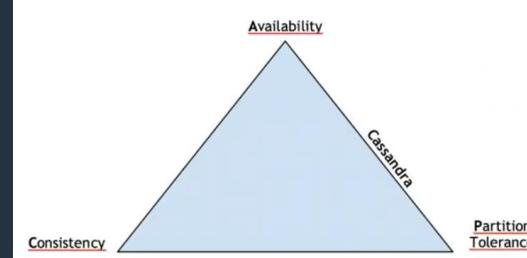


2. PRINCIPAIS BANCO DE DADOS NOSQL



APACHE CASSANDRA

- Orientado a Coluna (Column Family)
- Do Teorema CAP:



Banco de Dados Relacional:

sku_produto	nome_produto	preco_produto
1	Tênis	100
2	Bola	50
3	Camisa	200
4	Bike	900

Banco de Dados Colunar:

sku_produto	id	value
	0	1
	1	2
	2	3
	3	4

nome_produto	id	value
Tênis	0	1
Bola	1	2
Camisa	2	3
Bike	3	4

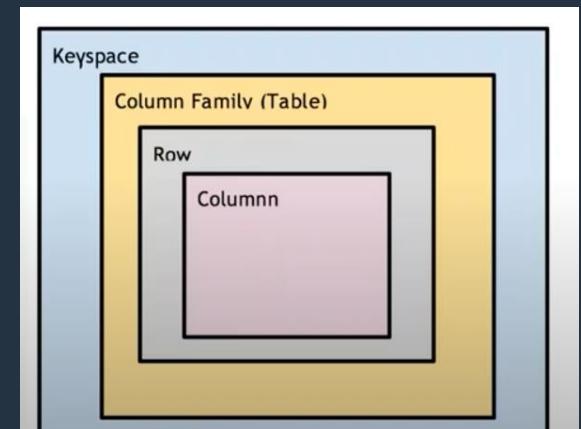
preco_produto	id	value
100	0	100
50	1	50
200	2	200
900	3	900



O Cassandra é um projeto de sistema de BD distribuído altamente escalável de segunda geração, que reúne a arquitetura do DynamoDB da Amazon Web Services e modelo de dados baseado no BigTable do Google.

Arquitetura do Cassandra: é segmentado em:

- Keyspace: corresponde a um **banco de dados** no relacional
- Tables (column families): equivalente à uma **tabela** no mundo relacional
- Rows: São compostas pela Primary key e um conj de columns
- Columns: Composto por Column key, Column value e Timestamp



REDIS – REMOTE DICTIONARY SERVER

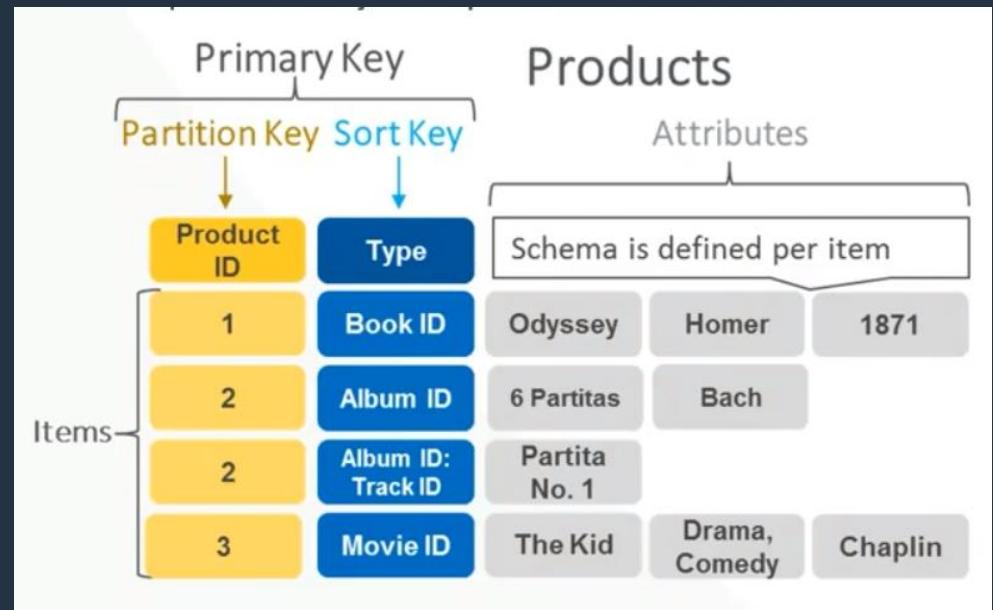


- Chave-valor (key-Value)
- São sistemas distribuídos, também conhecidos como tabelas de hash distribuídas, armazenam objetos indexados por chaves, e possibilitam a busca por esses objetos a partir de suas chaves.
- O Redis é um banco de dados de memória e de código aberto que é usado como cache e como intermediário de mensagens. Também é conhecido como um servidor de dados estruturados.
- Oficialmente o Redis não tem uma distribuição para

Windows, porém é possível encontrar uma distribuição

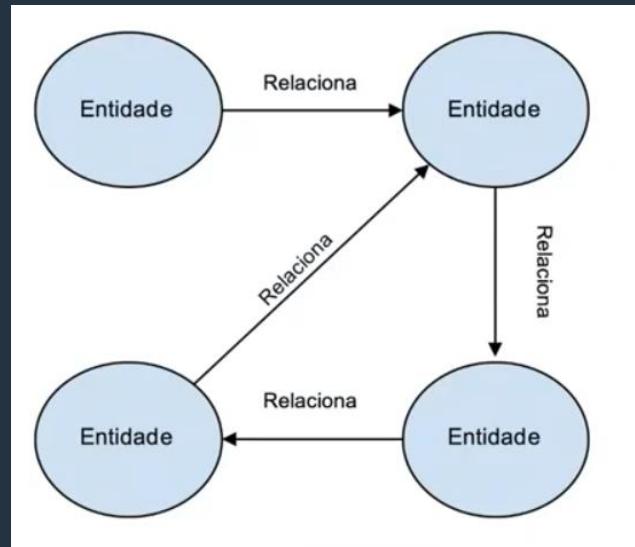
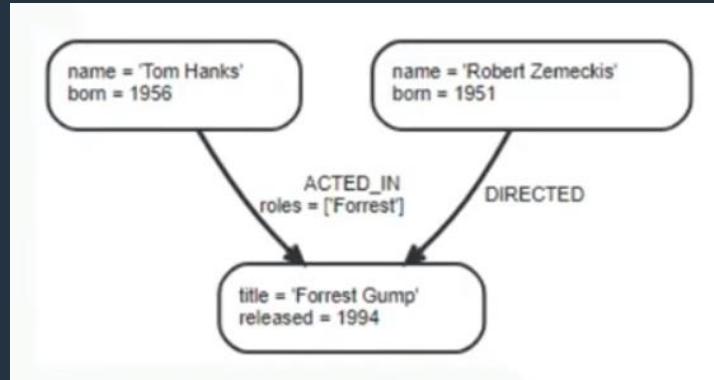
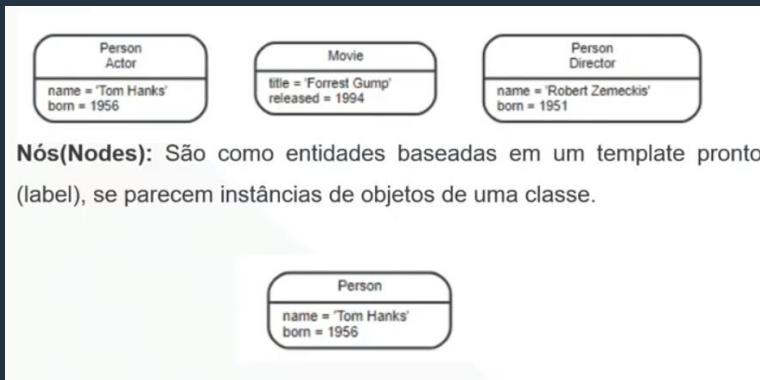
paralela no GitHub. Características:

- Desempenho muito rápido
- Estruturas de dados na memória
- Versatilidade e facilidade de uso
- Replicação e persistência
- Compatibilidade com as linguagens Java, Python, PHP, C, C++, C#, JavaScript, Node.js, Ruby, R, Go e muitas outras.



NEO4J

- Os dados são persistidos em um esquema de grafo, onde um registro “aponta” para o próximo.
- O Neo4j é um sistema SGBD gráfico, descrito como um BD transacional compatível com ACID com armazenamento e processamento de gráfico nativo, está disponível uma “edição da comunidade” de código aberto.
- Implementado em Java e acessível a partir de softwares escritos em outras linguagens usando a linguagem de consulta Cypher Query através de um endpoint HTTP transacional ou através do protocolo binário “bolt”



Palavras chaves do Cypher:

- MATCH
- WHERE
- RETURN
- CREATE
- MERGE

MONGO DB

- Orientado a Documentos
- Armazena chave/valor.
- O valor é um documento estruturado e indexado, com metadados.
- Valor (Documento), pode ser consultado.
- JSON: JavaScript Object Notation.
 - Feito para trocas de dados
 - Mais compacto e legível que XML
- Código aberto e multiplataforma, escrito em C++, escalável, não possui Schema fixo, não tem integridade referencial, permite documentos aninhados.

```
{"clientes": [  
    { "nome": "Fernando", "sobrenome": "Amaral" },  
    { "nome": "Anna", "sobrenome": "Ebling" },  
    { "nome": "Pedro", "sobrenome": "Soares" }  
]
```



mongoDB

NoSQL - JSON

“ Estrutura nome/valor, entre aspas duplas

█ Dados separados por vírgula

█ Chaves separam objetos

»» Vetores entre colchetes

Relacional

Banco de Dados

Tabela

Linha

Coluna

MongoDB

Banco de Dados

Coleção

Documento

Campo

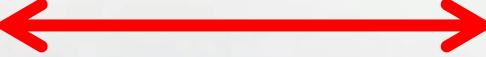
MONGO DB

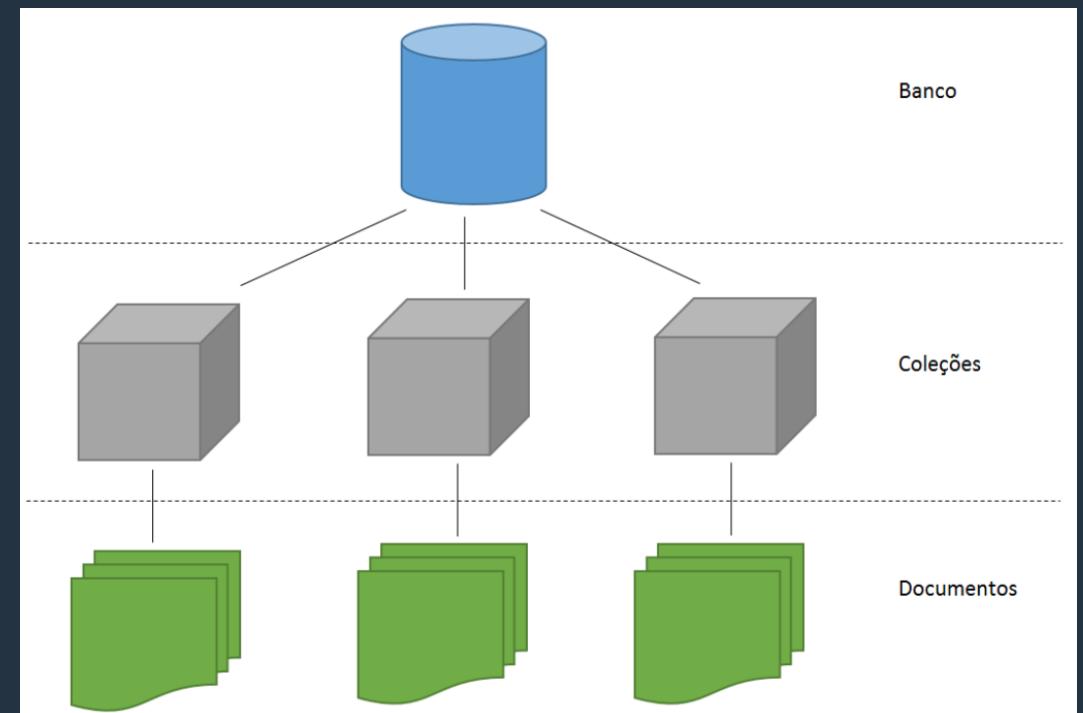
Document

- Um documento é um conjunto de pares de valores-chave.
- Documentos têm esquema dinâmico. Esquema dinâmico significa que os documentos na mesma coleção não precisam ter o mesmo conjunto de campos ou estrutura e campos comuns em documentos de uma coleção podem conter diferentes tipos de dados.

```
db.users.insertOne(  
  {  
    name: "sue",  
    age: 26,  
    status: "pending"  
  })
```

collection
field: value
document

Relacional	MongoDB
Banco de Dados	Banco de Dados
Tabela	Coleção
Linha	
Coluna	Documento
	Campo



MONGO DB – COMANDOS

<https://docs.mongodb.com/manual/reference/command/>

```
db.posts.insert([
  {nome:"André",postagem:"Produtos caros", data:"12-01-2019",idade:25},
  {nome:"Ricardo",postagem:"Produtos caros", data:"14-07-2019", idade:12}])
#idade menor ou igual a 12
> db.posts.find({postagem:"Produtos caros",idade: {$lte: 12}})
{ "_id" : ObjectId("5d0911600ee1100c307004da"), "nome" : "Ricardo", "postagem"
: "Produtos caros", "data" : "14-07-2019", "idade" : 12 }
```

- **\$lt: menor que**
- **\$lte: menor ou igual que**
- **\$ne: diferente de**
- **\$in: contém**
- **\$nin: não contém**

- Comando "use" acessa banco
- Acessar banco inexistente cria o banco
 - É necessário inserir dados para persistir o banco de dados

```
> use dbmidias
switched to db dbmidias
```

- **insert**
 - Insere um único documento na coleção posts
- ```
>db.posts.insert({nome:"José",postagem:"Bons Produtos!", data:"31-06-2019"})
WriteResult({ "nInserted" : 1 })
```
- Acima a coleção é criada implicitamente**
- Para criar a coleção primeiro:
- ```
>db.createCollection("clientes")
```

```
> db.posts.find()
{ "_id" : ObjectId("5d090bc10ee1100c307004d4"),
"nome" : "José", "postagem" : "Bons Produtos!",
"data" : "31-06-2019" }
{ "_id" : ObjectId("5d090cd10ee1100c307004d5"),
"nome" : "Antonio", "postagem" : "Minha bike
quebrou", "data" : "26-05-2019" }
{ "_id" : ObjectId("5d090cd10ee1100c307004d6"),
"nome" : "Maria Silva", "postagem" : "Encontrei
tudo que procurava", "data" : "14-06-2019" }
{ "_id" : ObjectId("5d090cd10ee1100c307004d7"),
"nome" : "Lucas Andrade", "postagem" : "Ótimo
atendimento!", "data" : "12-04-2019" }
```

= (Select *)

3. CRUD NO MONDO DB

Logs

--version: versão de uso do mongod.

--config <filename>, -f <filename>

- Para obter ajuda: db.g<Tab>

- Use setas para cima ou para baixo para acessar o histórico de comandos.
 - CTRL C – copia ou encerra
 - CTRL V – botão direito do mouse
 - Aspas simples, duplas ou ausentes

```
MongoDB Enterprise > db.aula.insert({"nome": "Jose1"})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.aula.insert({"nome": 'Jose1'})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.aula.insert({nome: 'Jose1'})
WriteResult({ "nInserted" : 1 })
```

- Verificar o nível: db.getProfilingLevel()
 - Verificar o status: : db.getProfilingStatus()
 - Alterar o status: : db.getProfilingLevel(level, milisec)
 - Desativar o log: db.setProfilingLevel(0)
 - Fazer uma consulta:
db.system.profile.find({millis : { \$gt:1000}}).sort({ts:-1})

FIELDS

- Tipos Fields (campos) nos documentos →

- ISODate(): YYYY-MM-DD hh:mm:ss"

```
db.ColDate.insert({ aDate: new ISODate('2012-05-01 12:30:00')});
```

```
getDate();
```

```
getMonth();
```

```
getFullYear();
```

- Object Id: Em banco de dados relacionais temos as Primary Keys, que são chaves primárias. É um identificador único da linha da tabela.

No MongoDB temos o ObjectId!

ObjectId.getTimestamp()

ObjectId.toString()

ObjectId.valueOf()

ObjectId usa a data local não do server.

O MongoDB suporta vários tipos de dados:

String	Null	Object
Integer	Symbol	
Boolean	Date	
Double	Object ID	
Min/ Max keys	Binary data	Foto/img
Arrays	Code	
Timestamp	Regular expression	

Exemplo:

Chave - valor	
{ "nome": "Pedro Pereira", "idade": 5, "esportes": ["tênis", "futebol", "videogame"], "estuda": true, "matérias_notas": { "matemática": 9,1, "português": 7,8, "geografia": 8,2 } } WriteResult({ "nInserted" : 1 }) }	// string // number // array // boolean // binary // object // number // number // number

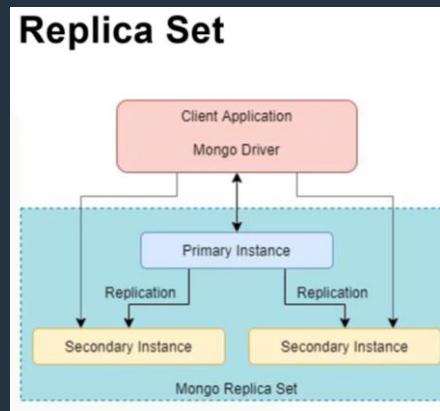
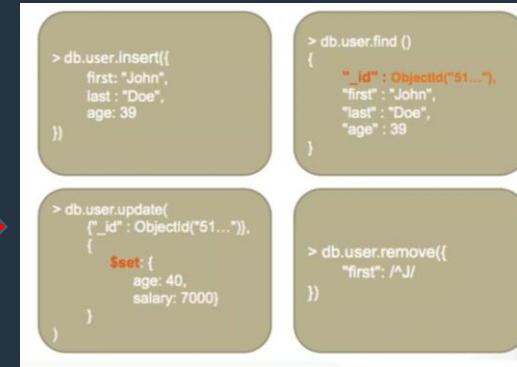
INSERTS

- CRUD
- Inserts

Insert ({})

InsertOne ({})

InsertMany ([{}, {}, {}])



Insert ({})

- ✓ db.crud.insert({field_A: 123})
- ✓ db.crud.insert([{field_A: 123}, {field_B: 223}]) //array com mais de um insert

```
MongoDB Enterprise > db.crud.insert({field_A: 123})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.crud.insert([ {field_A: 123}, {field_B: 223} ])
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 2,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
```

InsertOne ({})

- A partir da versão 3.2 do MongoDB
- ✓ db.crud.insertOne({field_A: 2123})

```
MongoDB Enterprise > db.crud.insertOne([ {field_A: 2123}, {field_B: 2223} ])
uncaught exception: Error: operation passed in cannot be an Array :
addToOperationsList@src/mongo/shell/bulk_api.js:604:23
Bulk/this.insert@src/mongo/shell/bulk_api.js:650:20
DBCollection.prototype.insertOne@src/mongo/shell/crud_api.js:260:5
@(shell):1:1
MongoDB Enterprise > db.crud.insertOne({field_A: 2123})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5fedf44469910c9912e213e4")
}
```

✓ Create Inserção

- db.collection.insert(<document>)
- db.collection.save(<document>)
- db.collection.update(<query>, <update>, { upsert: true })

✓ Read consultas

- db.collection.find(<query>, <projection>)
- db.collection.findOne(<query>, <projection>)

✓ Update alterações

- db.collection.update(<query>, <update>, <options>)

✓ Delete deletar

- db.collection.remove(<query>, <justOne>)

Ordered

```
MongoDB Enterprise > db.crudMany.insertMany([ { _id:2, a: 4123}, { _id:7, b: 4456}], {ordered: false})
uncaught exception: BulkWriteError({
  "writeErrors" : [
    {
      "index" : 0,
      "code" : 11000,
      "errmsg" : "E11000 duplicate key error collection: test.crudMany index: _id_ dup key: { _id: 2.0 }",
      "op" : {
        "_id" : 2,
        "a" : 4123
      }
    }
  ],
  "writeConcernErrors" : [ ],
  "nInserted" : 1,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
```

```
MongoDB Enterprise > db.crudMany.find()
{ "_id" : 1, "a" : 123 }
{ "_id" : 2, "b" : 4456 }
{ "_id" : 3, "c" : 789 }
{ "_id" : 7, "b" : 4456 }
```

QUERY

- Consultas no MongoDB

find()

findOne()

Find()

db.crud2.find()

db.crud2.find

(

{ b: 456 }

)

```
MongoDB Enterprise > db.crud2.find({b:456})
{ "_id" : ObjectId("5fee018169910c9912e21401"), "b" : 456 }
{ "_id" : ObjectId("5fee018169910c9912e21404"), "b" : 456 }
{ "_id" : ObjectId("5fee018169910c9912e21407"), "b" : 456 }
```

Find()

db.collection.find(query, projection)

- ✓ **Query** – opcional: especifica o filtro de seleção usando operadores de consulta. Para retornar todos os documentos em uma coleção, omita este parâmetro ou passe um documento vazio ({}).
- ✓ **Projeção** – opcional: especifica os campos a serem retornados nos documentos que correspondem ao filtro da consulta. Para retornar todos os campos nos documentos correspondentes, omita este parâmetro.

findOne()

db.crud2.findOne()

db.crud2.findOne

(

{ b: 456 }

)

```
MongoDB Enterprise > db.crud2.findOne({b:456})
{ "_id" : ObjectId("5ff1174afe368a5f724ef86c"), "b" : 456 }
```

Filtros que queremos visualizar:

Projection

SELECT nome, endereço

FROM tabx

WHERE

... Seus filtros (nome = "Jose Pereira")

(Projection)

Db.collect.find ({ where-filtros}, { select-projection })

Db.collect.find ({ nome:'Pedro'}, { nome:1, endereço:1 })

Db.collect.find ({ nome:'Pedro'}, { nome:1, endereço:1, _id:0 })

Db.collect.find ({ nome:'Pedro'}, { nome:1, endereço:0, _id:0 })

Projection

```
MongoDB Enterprise > db.teste.findOne({estuda:true}, {nome:1, esportes:1})
{
  "_id" : ObjectId("5fed3897b1446e3c5d4502ae"),
  "nome" : "Pedro Pereira",
  "esportes" : [
    "tênis",
    "futebol",
    "videogame"
  ]
}
MongoDB Enterprise > db.teste.findOne({estuda:true}, {nome:1, esportes:1, _id:0})
{
  "nome" : "Pedro Pereira",
  "esportes" : [
    "tênis",
    "futebol",
    "videogame"
  ]
}
MongoDB Enterprise > db.teste.findOne({estuda:true}, {nome:1, esportes:0, _id:0})
uncaught exception: Error: error: {
  "ok" : 0,
  "errmsg" : "Cannot do exclusion on field esportes in inclusion projection",
```

UPDATES

- Updates no MongoDB:

UpdateOne

UpdateMany

ReplaceOne

UpdateOne

Desde a versão 3.2.

Definição: db.collection.updateOne(filter, update, options)

```
db.collection.updateOne(  
  <filter>,  
  <update>,  
  {  
    upsert: <boolean>, // se não encontrar, vai inserir  
    writeConcern: <document>,  
    collation: <document>,  
    arrayFilters: [ <filterdocument1>, ... ],  
    hint: <document|string> // Available starting in MongoDB 4.2.1  
  }  
)
```

UpdateMany

Desde a versão 3.2.

```
db.funcionarios.updateMany({nome: /a/}, {$set:{salario: 2000}})
```

```
MongoDB Enterprise > db.funcionarios.find({nome: /a/})  
[{"_id": ObjectId("5fee12cae88d88a848a7f5b6"), "nome": "Jose Pereira", "sexo": "M", "escolaridade": "S", "depto": {"deptname": "A", "Local": "X"} }  
{"_id": ObjectId("5fee12d6e88d88a848a7f5b7"), "nome": "Carlos Pereira", "sexo": "M", "escolaridade": "H", "depto": {"deptname": "A", "Local": "Y"} }  
{"_id": ObjectId("5fee12d6e88d88a848a7f5b8"), "nome": "Maria Pereira", "sexo": "F", "escolaridade": "M", "depto": {"deptname": "A", "Local": "Y"} }  
{"_id": ObjectId("5fee12d6e88d88a848a7f5b9"), "nome": "Perla Nascimento", "sexo": "F", "escolaridade": "S", "depto": {"deptname": "B", "Local": "V"}, "salario": 1000 }  
MongoDB Enterprise > db.funcionarios.updateMany({nome: /a/}, {$set:{salario: 2000}})  
{ "acknowledged": true, "matchedCount": 4, "modifiedCount": 4 }  
MongoDB Enterprise > db.funcionarios.find({nome: /a/})  
[{"_id": ObjectId("5fee12cae88d88a848a7f5b6"), "nome": "Jose Pereira", "sexo": "M", "escolaridade": "S", "depto": {"deptname": "A", "Local": "X"}, "salario": 2000 }  
{"_id": ObjectId("5fee12d6e88d88a848a7f5b7"), "nome": "Carlos Pereira", "sexo": "M", "escolaridade": "H", "depto": {"deptname": "A", "Local": "Y"}, "salario": 2000 }  
{"_id": ObjectId("5fee12d6e88d88a848a7f5b8"), "nome": "Maria Pereira", "sexo": "F", "escolaridade": "M", "depto": {"deptname": "A", "Local": "Y"}, "salario": 2000 }  
{"_id": ObjectId("5fee12d6e88d88a848a7f5b9"), "nome": "Perla Nascimento", "sexo": "F", "escolaridade": "S", "depto": {"deptname": "B", "Local": "V"}, "salario": 2000 }
```

ReplaceOne

Desde a versão 3.2.

```
db.funcionarios.replaceOne({nome: /Perla/}, {salario: 5000})
```

```
MongoDB Enterprise > db.funcionarios.find({nome: /Perla/})  
[{"_id": ObjectId("5fee12d6e88d88a848a7f5b9"), "nome": "Perla Nascimento", "sexo": "F", "escolaridade": "S", "depto": {"deptname": "B", "Local": "Y"}, "salario": 2000 }  
MongoDB Enterprise > db.funcionarios.replaceOne({nome: /Perla/}, {salario: 5000})  
{ "acknowledged": true, "matchedCount": 1, "modifiedCount": 1 }
```

DELETES

- Deletes no MongoDB:

DeleteOne

DeleteMany

DeleteOne

Desde a versão 3.2.

Definição: db.collection.deleteOne(filter)

```
const query = { "name": "lego" };
```

```
itemsCollection.deleteOne(query)
```

DeleteMany

db.crud3.deleteMany({a: 123})

```
MongoDB Enterprise > db.crud3.find({a: 123})
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5bd"), "a" : 123 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c0"), "a" : 123 }
MongoDB Enterprise > db.crud3.deleteMany({a: 123})
{ "acknowledged" : true, "deletedCount" : 2 }
MongoDB Enterprise > db.crud3.find({a: 123})
```

DeleteMany

db.crud3.deleteMany({b: {\$exists:true}})

```
MongoDB Enterprise > db.crud3.find({b: {$exists:true}})
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5bb"), "b" : 456 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5be"), "b" : 456 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c1"), "b" : 456 }
MongoDB Enterprise > db.crud3.deleteMany({b: {$exists:true}})
{ "acknowledged" : true, "deletedCount" : 3 }
MongoDB Enterprise > db.crud3.find({b: {$exists:true}})
```

DeleteOne

db.crud3.deleteOne({a: 123})

```
MongoDB Enterprise > db.crud3.insertMany([
... {a: 123},
... {b: 456},
... {c: 789},
... {a: 123},
... {b: 456},
... {c: 789},
... {a: 123},
... {b: 456},
... {c: 789},
... {e: 123},
... {f: 456},
... {g: 789},
... ])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5fee31e2e88d88a848a7f5ba"),
    ObjectId("5fee31e2e88d88a848a7f5bd"),
    ObjectId("5fee31e2e88d88a848a7f5bc"),
    ObjectId("5fee31e2e88d88a848a7f5c0"),
    ObjectId("5fee31e2e88d88a848a7f5be"),
    ObjectId("5fee31e2e88d88a848a7f5bf"),
    ObjectId("5fee31e2e88d88a848a7f5c0"),
    ObjectId("5fee31e2e88d88a848a7f5c1"),
    ObjectId("5fee31e2e88d88a848a7f5c2"),
    ObjectId("5fee31e2e88d88a848a7f5c3"),
    ObjectId("5fee31e2e88d88a848a7f5c4"),
    ObjectId("5fee31e2e88d88a848a7f5c5")
  ]
}
```

```
MongoDB Enterprise > db.crud3.find({a: 123})
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5ba"), "a" : 123 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5bd"), "a" : 123 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c0"), "a" : 123 }
```

```
MongoDB Enterprise > db.crud3.deleteOne({a: 123})
{ "acknowledged" : true, "deletedCount" : 1 }
MongoDB Enterprise > db.crud3.find({a: 123})
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5bd"), "a" : 123 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c0"), "a" : 123 }
```

DeleteMany

Como um delete sem where...

db.crud3.deleteMany({ })

Deleta
TUDO!

```
MongoDB Enterprise > db.crud3.find()
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5bc"), "c" : 789 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5bf"), "c" : 789 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c2"), "c" : 789 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c3"), "e" : 123 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c4"), "f" : 456 }
{ "_id" : ObjectId("5fee31e2e88d88a848a7f5c5"), "g" : 789 }
MongoDB Enterprise > db.crud3.deleteMany({})
{ "acknowledged" : true, "deletedCount" : 6 }
MongoDB Enterprise > db.crud3.find()
```

QUERIES COM OPERADORES

- Consultas mais elaboradas utilizando operadores.

Like

Consulta SQL

```
Select * from funcionário where nome like "%pereira%"
```

MongoDB

```
db.funcionarios.find({nome:/pereira/})
```

não traz nada porque temos "Pereira" e não "pereira"

```
db.funcionarios.find({nome:/ereira/})
```

ou

```
db.getCollection('funcionarios').find({nome:/ereira/})
```

Like

Consulta SQL

```
MongoDB Enterprise > db.funcionarios.find({}, {nome:1, _id:0})
{
  "nome" : "José Pereira"
}
{
  "nome" : "Carlos Pereira"
}
{
  "nome" : "Maria Pereira"
}
{
  "nome" : "Perla Nascimento"
}
MongoDB Enterprise > db.funcionarios.find({nome:/ereira/}, {nome:1, _id:0})
{
  "nome" : "José Pereira"
}
{
  "nome" : "Carlos Pereira"
}
{
  "nome" : "Maria Pereira"
}
```

```
db.funcionarios.find({nome:/ereira/})
```

ou

```
db.getCollection('funcionarios').find({nome:/ereira/})
```

OR

```
{ $or: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }
```

```
db.funcionarios.find({$or: [{escolaridade: 'M', 'escolaridade': 'S'}], _id:false, nome:true, endereço:true})
```

```
db.funcionarios.find({$or: [{escolaridade: 'M'}, {escolaridade: 'S'}]}, {_id:0, nome:1, endereço:1})
```

```
MongoDB Enterprise > db.funcionarios.find({$or: [{escolaridade: 'M'}, {escolaridade: 'S'}]}, {_id:false, nome:true, endereço:true})
{
  "nome" : "José Pereira", "escolaridade" : "M"
}
{
  "nome" : "Carlos Pereira", "escolaridade" : "S"
}
{
  "nome" : "Maria Pereira", "escolaridade" : "M"
}
```

Like

```
db.funcionarios.insertOne(
  { nome: 'José Pereira', sexo: 'M', escolaridade: 'S',
    depto: {deptname: 'A', Local: 'X' } })
```

```
db.funcionarios.insertMany([
  {nome: 'Carlos Pereira', sexo: 'M', escolaridade: 'M',
    depto: {deptname: 'A', Local: 'Y' }},
  {nome: 'Maria Pereira', sexo: 'F', escolaridade: 'M',
    depto: {deptname: 'A', Local: 'Y' }},
  {nome: 'Perla Nascimento', sexo: 'F', escolaridade: 'S',
    depto: {deptname: 'B', Local: 'Y' }}])
```

OR

O operador **\$or** executa uma operação lógica OR em uma matriz de duas ou mais <expressions> e seleciona os documentos que satisfazem pelo menos uma das <expressions>.

```
{ $or: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }
```

Consulta SQL

```
Select * from funcionário where escolaridade = "S" or escolaridade = "M"
```

```
MongoDB Enterprise > db.funcionarios.find({}, { _id:0, nome:1, escolaridade:1 })
{
  "nome" : "José Pereira", "escolaridade" : "S"
}
{
  "nome" : "Carlos Pereira", "escolaridade" : "M"
}
{
  "nome" : "Maria Pereira", "escolaridade" : "M"
}
{
  "nome" : "Perla Nascimento", "escolaridade" : "S" }
```

Operadores

Name	Description
\$eq	Matches values that are equal to a specified value.
\$gt	Matches values that are greater than a specified value.
\$gte	Matches values that are greater than or equal to a specified value.
\$in	Matches any of the values specified in an array.
\$lt	Matches values that are less than a specified value.
\$lte	Matches values that are less than or equal to a specified value.
\$ne	Matches all values that are not equal to a specified value.
\$nin	Matches none of the values specified in an array.

In, Not In

Consulta SQL

```
Select * from funcionário where escolaridade in ("S", "M")
```

MongoDB

```
db.funcionarios.find
```

```
(
```

```
  {escolaridade: {$in: ['M','S']}},
  {_id:false, nome:true, endereço:true}
```

```
)
```

```
db.funcionarios.find( {escolaridade: {$in: ['M','S']}}, {_id:false, nome:true, endereço:true} )
```

```
MongoDB Enterprise > db.funcionarios.find( {escolaridade: {$in: ['M','S']}}, {_id:false, nome:true, endereço:true} )
{
  "nome" : "José Pereira"
}
{
  "nome" : "Carlos Pereira"
}
{
  "nome" : "Maria Pereira"
}
{
  "nome" : "Perla Nascimento"
}
```

AGGREGATION

Aggregate Count

Consulta SQL

```
Select count(*) from funcionário
```

MongoDB

```
db.funcionarios.aggregate(  
  {$group: {'_id': null, 'count': {$sum:1}} }  
)
```

```
MongoDB Enterprise > db.funcionarios.aggregate( {$group: {'_id': null, 'count': {$sum:1}} })  
{ "_id" : null, "count" : 4 }
```

```
db.funcionarios.aggregate(  
  {$group: {'_id': '$salario', 'count': {$sum:1}} }  
)
```

```
MongoDB Enterprise > db.funcionarios.aggregate( {$group: {'_id': '$salario', 'count': {$sum:1}} })  
{ "_id" : 5000, "count" : 1 }  
{ "_id" : 2000, "count" : 3 }
```

Aggregate Avg

```
db.funcionarios.aggregate
```

```
(  
  {$group: {'_id': '$salario', 'media': {$avg:'$salario'}} }  
)
```

```
MongoDB Enterprise > db.funcionarios.aggregate( {$group: {'_id': '$salario', 'media': {$avg:'$salario'}} })  
{ "_id" : 5000, "media" : 5000 }  
{ "_id" : 2000, "media" : 2000 }
```

Aggregate Sum

```
db.funcionarios.aggregate
```

```
(  
  {$group: {'_id': '$salario', 'soma': {$sum:'$salario'}} }  
)
```

```
MongoDB Enterprise > db.funcionarios.aggregate( {$group: {'_id': '$salario', 'soma': {$sum:'$salario'}} })  
{ "_id" : 5000, "soma" : 5000 }  
{ "_id" : 2000, "soma" : 6000 }
```

LOOKUP (AGGREGATION)

\$lookup (aggregation)

Executa uma junção entre collections.

Para cada documento de entrada, o estágio \$ lookup adiciona um novo campo de array cujos elementos são os documentos correspondentes da coleção "unida".

```
{  
  $lookup:  
  {  
    from: <collection to join>,  
    localField: <field from the input documents>,  
    foreignField: <field from the documents of the "from" collection>,  
    as: <output array field>  
  }  
}
```

\$lookup (aggregation)

Essa operação poderia corresponder ao seguinte pseudo-SQL:

```
SELECT *, <output array field>  
FROM collection  
WHERE <output array field> IN (SELECT *  
  FROM <collection to join>  
  WHERE <foreignField>= <collection.localField>);
```

\$lookup - exemplo

```
db.orders.insertMany([  
  { "_id": 1, "item": "almonds", "price": 12, "quantity": 2 },  
  { "_id": 2, "item": "pecans", "price": 20, "quantity": 1 },  
  { "_id": 3 }  
)  
  
db.inventory.insertMany([  
  { "_id": 1, "sku": "almonds", "description": "product 1", "instock": 120 },  
  { "_id": 2, "sku": "bread", "description": "product 2", "instock": 80 },  
  { "_id": 3, "sku": "cashews", "description": "product 3", "instock": 60 },  
  { "_id": 4, "sku": "pecans", "description": "product 4", "instock": 70 },  
  { "_id": 5, "sku": null, "description": "Incomplete" },  
  { "_id": 6 }  
)
```

\$lookup - exemplo

```
MongoDB Enterprise > db.orders.insertMany([  
...  { "_id": 1, "item": "almonds", "price": 12, "quantity": 2 },  
...  { "_id": 2, "item": "pecans", "price": 20, "quantity": 1 },  
...  { "_id": 3 }  
... ])  
{ "acknowledged": true, "insertedIds": [ 1, 2, 3 ] }  
MongoDB Enterprise > db.inventory.insertMany([  
...  { "_id": 1, "sku": "almonds", "description": "product 1", "instock": 120 },  
...  { "_id": 2, "sku": "bread", "description": "product 2", "instock": 80 },  
...  { "_id": 3, "sku": "cashews", "description": "product 3", "instock": 60 },  
...  { "_id": 4, "sku": "pecans", "description": "product 4", "instock": 70 },  
...  { "_id": 5, "sku": null, "description": "Incomplete" },  
...  { "_id": 6 }  
... ])  
{ "acknowledged": true, "insertedIds": [ 1, 2, 3, 4, 5, 6 ] }
```

\$lookup - exemplo

```
db.orders.aggregate([  
  {  
    $lookup:  
    {  
      from: "inventory",  
      localField: "item",  
      foreignField: "sku",  
      as: "inventory_docs"  
    }  
  }  
])
```

\$lookup - exemplo

```
mongod Enterprise > db.orders.aggregate([  
  {  
    $lookup:  
    {  
      from: "inventory",  
      localField: "item",  
      foreignField: "sku",  
      as: "inventory_docs"  
    }  
  }  
]).pretty()  
[  
  {  
    "_id": 1,  
    "item": "almonds",  
    "price": 12,  
    "quantity": 2,  
    "inventory_docs": [  
      {  
        "_id": 1, "sku": "almonds", "description": "product 1", "instock": 120 }  
    ]  
  },  
  {  
    "_id": 2,  
    "item": "pecans",  
    "price": 20,  
    "quantity": 1,  
    "inventory_docs": [  
      {  
        "_id": 4, "sku": null, "description": "Incomplete" }  
    ]  
  }  
]
```

4. MODELAGEM E RELACIONAMENTO MONGO DB

MODELAGEM DE DADOS

- Normalização

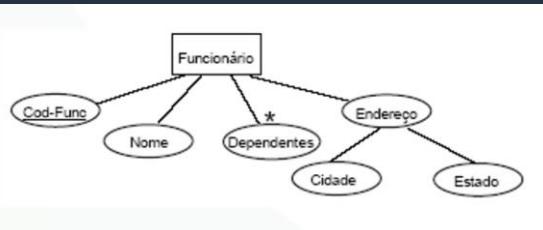
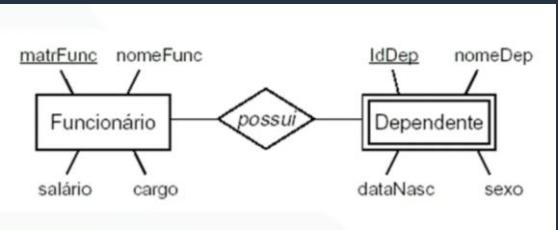
1^a Forma Normal (1FN): toda relação deve ter uma chave primária e deve-se garantir que todo atributo seja atômico. Atributos compostos devem ser separados.

2^a Forma Normal (2FN): toda relação deve estar na 1FN e deve-se eliminar dependências funcionais parciais, ou seja, todo atributo não chave deve ser totalmente dependente da chave primária.

3^a Forma Normal (3FN): toda relação deve estar na 2FN e devem-se eliminar dependências funcionais transitivas, ou seja, todo atributo não chave deve ser mutualmente independente.

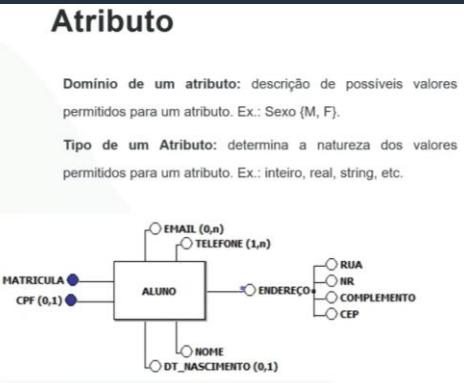
MODELAGEM DE DADOS

- Entidade: É uma representação concreta ou abstrata de um objeto, com características semelhantes, do mundo real. Ex.: Fornecedor, Pessoa, Imóvel, Curso.
- Entidade Forte x Fraca: Entidade Fraca não existe se não estiver relacionada a outra, isto é, ela é logicamente dependente da outra.



Atributo - Classificação

- Atributo Simples
- Atributo Único
- Atributo Opcional
- Atributo Monovalorado
- Atributo Composto
- Atributo Não Único
- Atributo Obrigatório
- Atributo Multivalorado



Atributo

Domínio de um atributo: descrição de possíveis valores permitidos para um atributo. Ex.: Sexo {M, F}.

Tipo de um Atributo: determina a natureza dos valores permitidos para um atributo. Ex.: inteiro, real, string, etc.

Esquema x Instância

Esquema de um Banco de Dados é a especificação da estrutura do Banco de Dados.

Instância é o conjunto de ocorrências dos objetos de dados de um esquema em um dado momento do tempo.

Código	Nome	Sigla	Esquema
1	Tecnologia da Informação	TI	Instância
2	Recursos Humanos	RH	

Modelo Peter Chen

- Identificar
 - Entidades.
 - Relacionamentos.
 - Atributos.
- Desenhar Modelo E-R.
- Mapar DER para o modelo relacional.
- Definir estrutura dos registros.

Relacionamentos

As entidades são conectadasumas às outras através de relacionamentos.

Ex.: As pessoas **Moram** em Apartamentos.

Os apartamentos **Formam** Condomínios.

Os condomínios **Localizam-se** em Ruas ou Avenidas.

As Avenidas e Ruas **Pertencem** a uma Cidade.

Relacionamentos

Cardinalidade do relacionamento (n = vários)

1:N (um para n - uma linha de uma tabela têm apenas um relacionamento com outra linha de outra tabela).

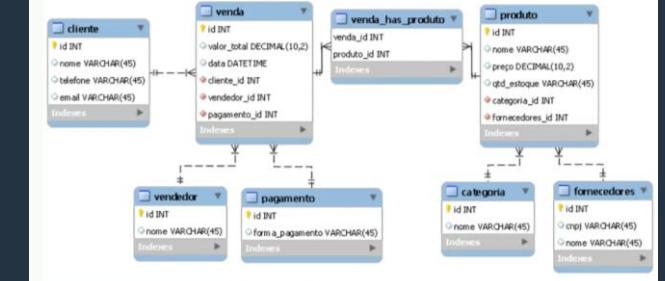
Um aluno mora atualmente em um único endereço).

1:N (um para n - uma linha de uma tabela pode ter "n" relacionamentos com outra tabela - um pai pode ter "n" filhos)

N:1 (idem anterior).

N:N ou N:M (muitos para muitos) - 1 aluno cursa "n" disciplinas e uma disciplina pode conter "n" alunos).

Relacionamentos



MER E DER

- Entidades: é um conjunto de objetos do mundo real sobre a qual deseja-se manter informações no banco de dados. (TABELAS)

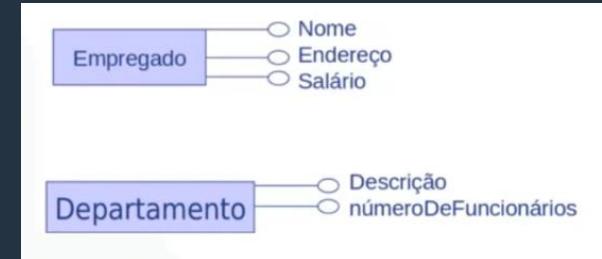
Exemplos: Cliente, Fornecedor, Departamento, Estoque.

- Atributos: informações a respeito de uma Entidade. Atributo chave para identifica-la de forma única. Pode ser um conjunto de atributos (chave-composta)
- Relacionamentos: É uma associação entre Entidades, pode ser representada por um losango.
- Cardinalidade: define a quantidade de elementos de uma Entidade associada com a quantidade de elementos de outra Entidade.

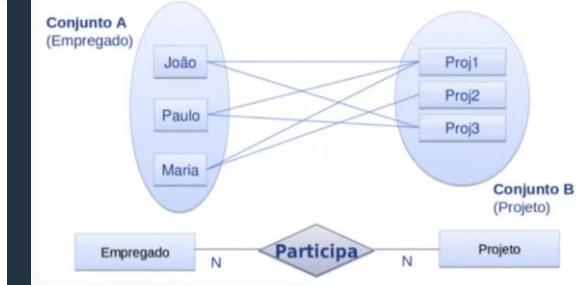
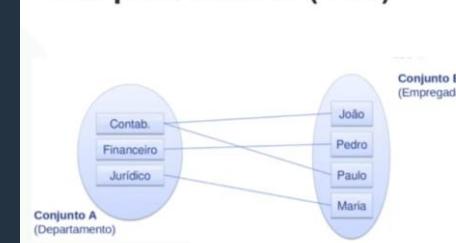
1 : 1 (um para um)

1 : N (um para muitos)

N : N (muitos para muitos)



Um para Muitos (1:N)



Muitos para Muitos (N:N)



CARDINALIDADE NO MONGO DB

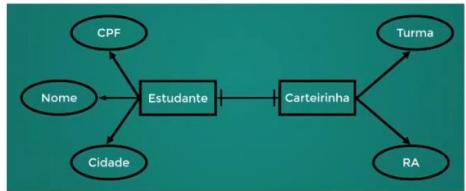
Embedded Documents: GANHO DE PERFORMANCE!
(pois o acesso é em uma única collection)

Um para Um

One to One

Estudante

- [id](#)
- nome
- cpf
- cidade
- [carteirinha_id](#)



- ### # Carteirinha
- [id_carteirinha](#)
 - turma
 - RA

Um para Um

One to One

```
{
  "id": 1,
  "nome": "Pedro Pereira",
  "cpf": 12345645,
  "cidade": "São Paulo",
  "carteirinha_id": 8
}

{
  "id_carteirinha": 8,
  "turma": "210B",
  "RA": 1234
}
```

Um para Um

Embedded documents

```
{
  "id": 1,
  "nome": "Pedro Pereira",
  "cpf": 12345645,
  "cidade": "São Paulo",
  "carteirinha": {
    "_id": 8,
    "turma": "210B",
    "RA": 1234
  }
}
```

Um para Muitos

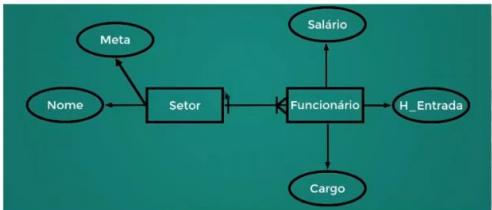
One to Many

Setor

- [id](#)
- meta
- nome

Funcionario

- [id](#)
- salario
- turno
- cargo
- [id_setor](#)



Um para Muitos

One to Many

```
{
  "id": 1,
  "nome": "José Pereira",
  "salario": 1400,
  "turno": "tarde",
  "cargo": "vendedor pleno",
  "id_setor": 3
}

{
  "id": 3,
  "meta": "40000",
  "nome": "depto Vendas"
}
```

Um para Muitos - Embedded Documents

```
{
  "id": 3,
  "meta": "40000",
  "nome": "depto Vendas"
  "funcionarios": [
    {
      "id": 1,
      "nome": "José Pereira",
      "salario": 1400,
      "turno": "tarde",
      "cargo": "vendedor pleno"
    },
    {
      "id": 2,
      "nome": "Ana Pereira",
      "salario": 1100,
      "turno": "tarde",
      "cargo": "vendedor pleno"
    },
    {
      "id": 3,
      "nome": "Carlos Pereira",
      "salario": 2100,
      "turno": "manhã",
      "cargo": "vendedor senior"
    }
  ]
}
```

Muitos para Muitos

Híbrido de muitos para muitos e embedded.

```
// Collection Fornecedor
{
  "_id": "f04",
  "cnpj": "165486984",
  "nome": "Fornecedor 1",
  "cep": "1098465"
}

{
  "_id": "p16",
  "descricao": "Panela",
  "preco": 35.50
}

{
  "_id": "p21",
  "descricao": "Prato",
  "preco": 13
}

{
  "_id": "f07",
  "cnpj": "98498151",
  "nome": "Fornecedor 2",
  "cep": "198498"
}

{
  "_id": "p47",
  "descricao": "Faqueiro",
  "preco": 117.50
}
```

Muitos para Muitos

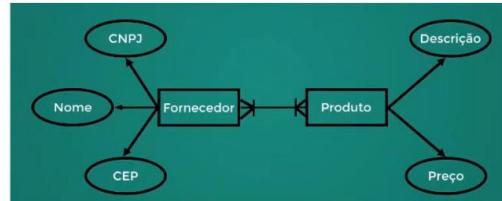
Many to Many

Fornecedor

- id
- cnpj
- nome
- cep

Produto

- id
- descricao
- preco



Muitos para Muitos

// Collection Fornecedores

```
{
  "_id": "f04",
  "cnpj": "165486984",
  "nome": "Fornecedor 1",
  "cep": "1098465"
}

{
  "_id": "p16",
  "descricao": "Panela",
  "preco": 35.50
}

{
  "_id": "p21",
  "descricao": "Prato",
  "preco": 13
}
```

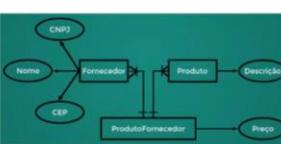
```
{
  "_id": "f07",
  "cnpj": "98498151",
  "nome": "Fornecedor 2",
  "cep": "198498"
}

{
  "_id": "p47",
  "descricao": "Faqueiro",
  "preco": 117.50
}
```

Muitos para Muitos

FornecedorProduto

- fornecedor_id
- produto_id



Muitos para Muitos com Arrays

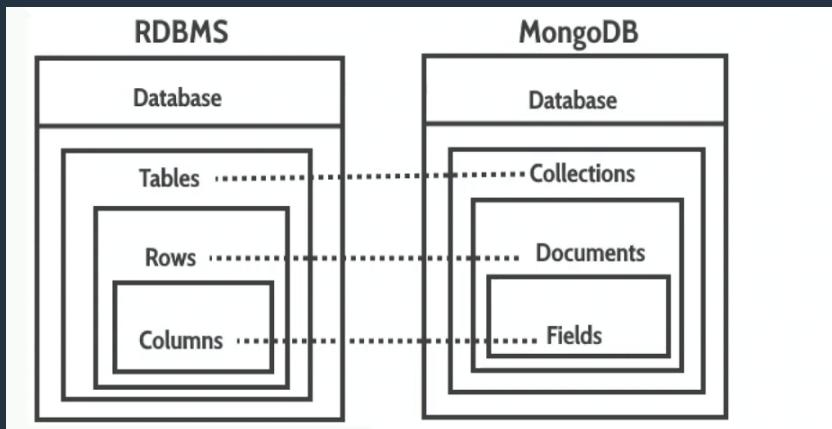
// Collection Fornecedores

```
{
  "_id": "f04",
  "cnpj": "165486984",
  "nome": "Fornecedor 1",
  "cep": "1098465",
  "produto_ids": ["p16", "p21"]
}
```

```
{
  "_id": "f07",
  "cnpj": "98498151",
  "nome": "Fornecedor 2",
  "cep": "198498",
  "produto_ids": ["p21", "p47"]
}
```

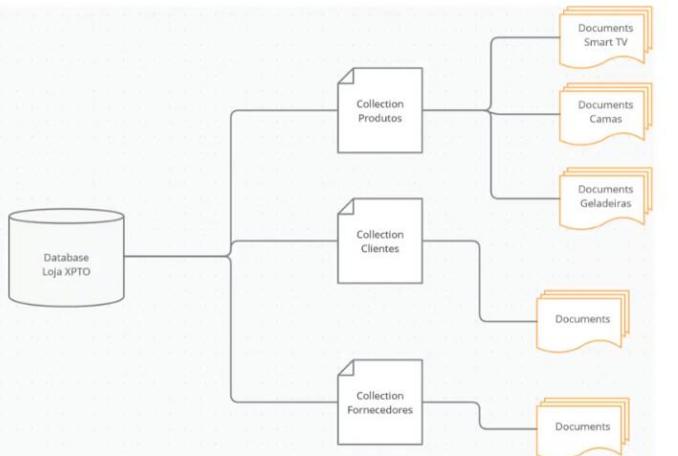
5. SCHEMA E VALIDATION

- *Database*
- *Collections*: no MongoDB é equivalente a *tables* no RDBMS
- *Documents*: no MongoDB é equivalente a *rows* no RDBMS
- *Fields*: no MongoDB é equivalente a *columns* no RDBMS



Database x Collection x Documents

O MongoDB abstrai diversos comandos DDL.
Estruturas são criadas conforme estas se tornam necessárias.



Relational Database

Student_Id	Student_Name	Age	College
1001	Chaitanya	30	Beginnersbook
1002	Steve	29	Beginnersbook
1003	Negan	28	Beginnersbook

MongoDB

```
{  
  "_id": ObjectId("....."),  
  "Student_Id": 1001,  
  "Student_Name": "Chaitanya",  
  "Age": 30,  
  "College": "Beginnersbook"  
}  
{  
  "_id": ObjectId("....."),  
  "Student_Id": 1002,  
  "Student_Name": "Steve",  
  "Age": 29,  
  "College": "Beginnersbook"  
}  
{  
  "_id": ObjectId("....."),  
  "Student_Id": 1003,  
  "Student_Name": "Negan",  
  "Age": 28,  
  "College": "Beginnersbook"  
}
```

DATABASE E COLLECTIONS

- use <database>; criar um database.
- show dbs; mostra os databases existentes.
- db,minhaColletcion.insert({"valor": 123}); comando cria a collection e o database.

Collection x Documents

Para se criar uma coleção, basta inserir um documento nela.

```
db.customers.insert({ nome: "sue", idade: 26, status: "A" })
```

Se quiser verificar quais coleções existem no banco de dados é só executar o comando `show collections` ou `show tables`.

One document

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: ["news", "sports"]  
}
```

One Collection

```
[  
  {  
    name: "sue",  
    age: 26,  
    status: "A",  
    groups: ["news", "sports"]  
  },  
  {  
    name: "al",  
    age: 18,  
    status: "D",  
    groups: ["politics", "news"]  
  }  
]
```

Collection

Para se criar uma coleção, basta inserir um documento nela.

```
db.customers.insert({ nome: "sue", idade: 26, status: "A" })
```

Ou executar o comando:

```
db.createCollections()
```

One Collection

```
[  
  {  
    name: "sue",  
    age: 26,  
    status: "A",  
    groups: ["news", "sports"]  
  },  
  {  
    name: "al",  
    age: 18,  
    status: "D",  
    groups: ["politics", "news"]  
  }  
]
```

config Database

```
MongoDB Enterprise > use admin  
switched to db admin  
MongoDB Enterprise > db.createUser(  
... {  
...   user: "ricardo",  
...   pwd: "igti",  
...   roles: [ "userAdminAnyDatabase",  
             "dbAdminAnyDatabase",  
             "readWriteAnyDatabase" ]  
... }  
)  
Successfully added user: {  
  "user": "ricardo",  
  "roles": [  
    "userAdminAnyDatabase",  
    "dbAdminAnyDatabase",  
    "readWriteAnyDatabase"  
  ]  
}  
MongoDB Enterprise > db.auth("ricardo", "igti")  
1  
MongoDB Enterprise > db.dropUser("ricardo")  
true  
MongoDB Enterprise >
```

admin Database

```
mongod - Copia - Bloco de Notas  
Arquivo Editar Formatar Exibir Ajuda  
storage:  
  dbPath: D:\Program Files\MongoDB\Server\4.4\data  
  journal:  
    enabled: true  
  # engine:  
  # mmapv1:  
  # wiredTiger:  
  
  # where to write logging data.  
  systemLog:  
    destination: file  
    logAppend: true  
    path: D:\Program Files\MongoDB\Server\4.4\log\mongod.log  
  
# network interfaces  
net:  
  port: 27017  
  bindIp: 127.0.0.1  
  
#processManagement:  
#security:  
  authorization: enabled  
#operationProfiling:  
#replication:  
#sharding:
```

local Database

Cada instância do mongod tem seu próprio banco de dados local, que armazena os dados usados no processo de replicação e outros dados específicos da instância. `local.startup_log`

```
MongoDB Enterprise > use local  
switched to db local  
MongoDB Enterprise > show collections  
startup_log  
MongoDB Enterprise > db.startup_log.find().pretty()
```

COLLECTION COM VALIDAÇÃO

Collection - Validator

```
db.createCollection("students", {  
  validator: {  
    $jsonSchema: {  
      bsonType: "object",  
      required: [ "name", "year", "major", "address" ],  
      properties: {  
        name: {  
          bsonType: "string",  
          description: "must be a string and is required"  
        },  
        year: {  
          bsonType: "int",  
          minimum: 2017,  
          maximum: 3017,  
          description: "must be an integer in [ 2017, 3017 ] and is required"  
        }  
      }  
    }  
  }  
})
```

Collection - Validator

```
db.createCollection("carros2", {  
  validator: {  
    $jsonSchema: {  
      bsonType: "object",  
      required: [ "modelo", "ano" ], //campos (fields) obrigatórios  
      properties: {  
        modelo: {  
          bsonType: "string",  
          description: "Modelo é string e é obrigatório."  
          //é mais documental, não mostra em caso de erro  
        },  
        fabricante: {  
          bsonType: "string",  
          minLength: 3,  
          maxLength: 30,  
          description: "Deve ser string."  
        },  
        ano: {  
          bsonType: "int",  
          minimum: 2015,  
          maximum: 2025,  
          description: "Inteiro entre [ 2015, 2025 ] e é obrigatório."  
        }  
      }  
    }  
  }  
})
```

Collection - Validator

```
MongoDB Enterprise > db.carro.insert({modelo: 'HB20', fabricante: 'Hyundai', ano: NumberInt(2020)})  
WriteResult({ "nInserted" : 1 })  
MongoDB Enterprise > db.carro.insert({modelo: 'KA', fabricante: 'Ford', ano: NumberInt(2017)})  
WriteResult({ "nInserted" : 1 })  
MongoDB Enterprise > db.carro.insert({modelo: 'S10', fabricante: 'GM', ano: NumberInt(2017)})  
WriteResult({  
  "nInserted" : 0,  
  "writeError" : {  
    "code" : 121,  
    "errmsg" : "Document failed validation"  
  }  
})
```

```
MongoDB Enterprise > db.carro.insert({modelo: 'Onix', fabricante: 'Chevrolet', ano: '2017'})  
WriteResult({  
  "nInserted" : 0,  
  "writeError" : {  
    "code" : 121,  
    "errmsg" : "Document failed validation"  
  }  
})  
MongoDB Enterprise > db.carro.insert({modelo: 'Onix', fabricante: 'Chevrolet', ano: NumberInt('2017')})  
WriteResult({ "nInserted" : 1 })
```

CAPPED COLLECTION

Capped Collection

Capped: boolean, opcional. Para criar uma coleção limitada, especifique true. Se você especificar true, também deve definir um tamanho máximo no campo size.

Size: number, opcional. Especifique um tamanho máximo em bytes para uma coleção limitada. Quando uma coleção limitada atinge seu tamanho máximo, o MongoDB remove os documentos mais antigos para liberar espaço para os novos documentos. O campo de tamanho é obrigatório para coleções limitadas e ignorado para outras coleções.

Max: number, opcional. O número máximo de documentos permitidos na coleção limitada. *O limite de tamanho tem precedência sobre este limite.* Se uma coleção limitada atingir o limite de tamanho antes de atingir o número máximo de documentos, o MongoDB remove os documentos antigos. Se você preferir usar o limite máximo, certifique-se de que o limite de tamanho, que é necessário para uma coleção limitada, seja suficiente para conter o número máximo de documentos.

Capped Collection

```
db.createCollection( <name>,
{
  capped: <boolean>,
  autoIndexId: <boolean>,
  size: <number>,
  max: <number>,
  storageEngine: <document>,
  validator: <document>,
  validationLevel: <string>,
  validationAction: <string>,
  indexOptionDefaults: <document>,
  viewOn: <string>, // Added in MongoDB 3.4
  pipeline: <pipeline>, // Added in MongoDB 3.4
  collation: <document>, // Added in MongoDB 3.4
  writeConcern: <document>
})
```

Capped Collection

Max: number, opcional. O número máximo de documentos permitidos na coleção limitada. *O limite de tamanho tem precedência sobre este limite.*

```
db.createCollection("log_simulado", {
  capped: true,
  size: 1024, // em bytes
  max: 5
})
```

```
db.log_simulado.insert({n: 1, desc: '1xxxxxx'})
db.log_simulado.insert({n: 2, desc: '2xxxxxx'})
db.log_simulado.insert({n: 3, desc: '3xxxxxx'})
db.log_simulado.insert({n: 4, desc: '4xxxxxx'})
db.log_simulado.insert({n: 5, desc: '5xxxxxx'})
db.log_simulado.insert({n: 6, desc: '6xxxxxx'})
```

OBJECT DOCUMENT MAPPER (ODM)

- ORM – Object Relational Mapper
- ODM – Object Document Mapper

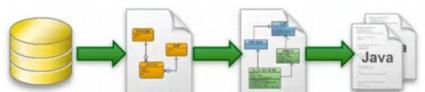
ORM, ODM

Qual o melhor jeito de interagir (conectar) com um banco de dados?

Existem duas abordagens:

- Usando a linguagem de consulta nativa dos bancos de dados (por exemplo, SQL).
- Usando um ORM ou um ODM.

Segundo Krakowiak (2003), em um sistema de computação distribuída, middleware é definido como uma camada de software que reside entre o sistema operacional e as aplicações em cada ponto do sistema.



ORM, ODM



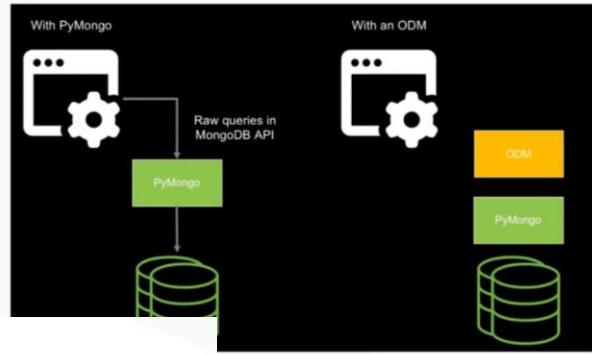
- ORM (Mapeamento Objeto Relacional).
- ODM (Documento Objeto Mapper).

São técnicas de programação para conversão de dados entre sistemas de tipos incompatíveis em bancos de dados e linguagens de programação orientadas a objetos.

Isso cria um "banco de dados de objeto virtual" que pode ser usado a partir de dentro da linguagem de programação.

- ORM - para bancos de dados relacionais.
- ODM - para NoSQL bancos de dados.

Mapper



Soluções ORM, ODM

Existem muitas soluções ODM / ORM disponíveis.

- **Mongoose**: ferramenta de modelagem de objetos MongoDB projetada para funcionar em um ambiente assíncrono.
- **Waterline**: é um ORM extraído da estrutura web **Sails.js** baseada no Express. Ele fornece uma API uniforme para acessar vários bancos de dados diferentes, incluindo Redis, MySQL, LDAP, MongoDB e Postgres.
- **Bookshelf.js**: Apresenta interfaces de retorno de chamada tradicionais e baseadas em promessa, fornecendo suporte a transações, carregamento de relação ansios/aninhado, associações polimórficas e suporte para relações um-para-um, um-para-muitos e muitos-para-muitos. Funciona com PostgreSQL, MySQL e SQLite3.
- **Objection.js**: torna o mais fácil possível o uso de todo o poder do SQL e do mecanismo de banco de dados subjacente (suporta SQLite3, Postgres e MySQL).

ORM

ORM (Object Relational Mapper) significa mapear um objeto com um mundo relacional.

Basicamente converte dados entre tipos incompatíveis em linguagens de programação orientadas a objetos.

ORM envolve os detalhes específicos de implementação de drivers de armazenamento em uma API (interface de programa de aplicativo) e mapeia os campos relacionais para membros de um objeto.

ODM (Object Document Mapper) é um “mapeador” de documentos do MongoDB.

ORM, ODM

- Um melhor desempenho pode ser obtido usando SQL ou qualquer linguagem de consulta suportada pelo banco de dados.
- ORM / ODMs são frequentemente mais lentos porque usam código de tradução para mapear entre objetos e o formato de banco de dados, o que pode não usar as consultas de banco de dados mais eficientes.
- O benefício de usar um ORM / ODM é que os programadores podem continuar a pensar em termos de objetos JavaScript em vez da semântica do banco de dados - isso é particularmente verdadeiro se você precisar trabalhar com bancos de dados diferentes (no mesmo site ou em diferentes sites).
- Eles também fornecem um local centralizado para realizar a validação e verificação de dados.

ORM, ODM

- **Sequelize**: ORM para Node.js e io.js. Ele suporta PostgreSQL, MySQL, MariaDB, SQLite e MSSQL e oferece suporte a transações sólidas, relações, replicação de leitura entre outros
- **Node ORM2**: é um ORM para NodeJS. Ele suporta MySQL, SQLite e Progress, ajudando a trabalhar com o banco de dados usando uma abordagem orientada a objetos.
- **JugglingDB**: é um ORM de banco de dados cruzado para NodeJS, fornecendo uma interface comum para acessar os formatos de banco de dados mais populares. Atualmente com suporte a MySQL, SQLite3, Postgres, MongoDB, Redis e js-memory-storage (mecanismo de escrita própria apenas para teste).

O Mongoose é o mais popular ODM no momento e é uma escolha razoável se você estiver usando o MongoDB para seu banco de dados.

Mongoose

- O Mongoose fornece um método simples e baseado em solução de esquema para modelar seus dados no aplicativo. Inclui seleção de tipo, validação, construção de consulta, lógica de negócios.
- O Mongoose é instalado no seu projeto (package.json) assim como outra dependência qualquer — usando NPM. Para instalá-lo, use a seguinte linha de comando dentro da pasta do seu projeto: **npm install mongoose**.
- A instalação do Mongoose adiciona todas as suas dependências, incluindo o driver de banco de dados MongoDB, mas não instala o MongoDB propriamente dito.

FERRAMENTAS DE GERENCIAMENTO

- Existem diversas ferramentas disponíveis no mercado para administrar o banco de dados MongoDB. Elas melhoram a produtividade em tarefas de desenvolvimento e administração.

NoSQLBooster (antigo MongoBooster)

O NoSQLBooster (anteriormente MongoBooster) é uma ferramenta popular para administrar seu banco de dados. Disponível em diversas plataformas (Mac, Linux e Windows) ela é centrada na linha de comando mongo e fornece recursos de monitoramento, construtor de consultas com autocomplete, consultas SQL e suporte à sintaxe ES2017.

Principais recursos

- Autocomplete inteligente.
- Consultas em SQL.
- Construção de consultas encadeadas.
- Análise de Schemas.
- Monitoramento de servidor.

NoSQL Manager

Essa ferramenta de administração mescla a interface amigável e o poder da linha de comando. Oferece alto desempenho com suporte para todos os recursos mais recentes do MongoDB e MongoDB Enterprise.

Principais recursos

- Interface gráfica para linha de comando do MongoDB com preenchimento automático de código.
- Três modos de exibição: Árvore, Tabela e JSON.
- Importação de tabelas de bancos de dados MySQL e SQL Server.
- Exportação de documentos para os formatos CSV, XML, XLSX e JSON.

✓ NoSQL Booster

✓ NoSQL Manager

✓ MongoDB Compass

✓ Robo3T

✓ Studio3T

MongoDB Compass

O MongoDB Compass é desenvolvido pela própria equipe do MongoDB. Ele fornece aos usuários uma visualização gráfica de seu esquema MongoDB sem a necessidade da linguagem de consulta por linha de comando.

Principais recursos

- O MongoDB Compass analisa documentos e exibe estruturas chaves em uma coleção usando consultas ad-hoc em segundos.
- Oferece suporte a informações rápidas sobre o status do servidor e o desempenho da consulta.
- Permite visualizar o desempenho de consultas.
- Ajuda os usuários a tomar decisões sobre indexação, validação de documentos e muito mais.
- Não há necessidade de escrever na linha de comando.

Robo 3T (antigo Robomongo)

Robo 3T (anteriormente Robomongo) é uma das mais populares interfaces gráficas e gratuita para os amantes do MongoDB. Leve e de código aberto, oferece suporte a várias plataformas (Mac, Linux e Windows) e incorpora também a linha de comando mongo em sua aplicação para fornecer interação para os usuários mais avançados. É desenvolvido pela 3T Software, a equipe por trás da IDE Studio 3T.

Principais recursos

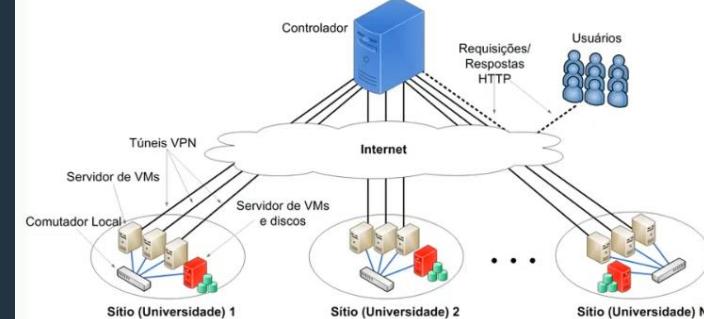
- Linha de comando incorporada.
- Interface assíncrona e sem bloqueios.
- Suporte para MongoDB 4.0+.

6. ARMAZENAMENTO DE DADOS EM NUVEM E SISTEMAS DE ARQUIVOS

INTRODUÇÃO A BANCO DE DADOS EM NUVEM

1. Modelos em Nuvem: SaaS, PaaS, IaaS
2. DBaaS
3. Tipos de Nuvem
4. Principais Fornecedores

- Provisionamento dinâmico de recursos sob demanda
- Escalabilidade
- Cobrança baseada no uso do recurso ao invés de uma taxa fixa
- Distribuição geográfica dos recursos



Modelos em Nuvem

SaaS é caracterizado principalmente pela não-aquisição de licenças de software.

PaaS envolve um ambiente virtual para criação, hospedagem e controle de softwares e bancos de dados.

IaaS apenas abstraí aspectos relacionados à parte física de servidores e redes.

MODELOS EM NUVEM

IaaS — Infrastructure as a Service (Infraestrutura como Serviço)

Nesse exemplo dos modelos de nuvem, *a empresa contrata uma capacidade de hardware que corresponde a memória, armazenamento, processamento, etc.* Podem entrar nesse pacote de contratações os servidores, roteadores, racks, entre outros.

Dependendo do fornecedor e do modelo escolhido, a sua empresa pode ser tarifada, por exemplo, pelo número de servidores utilizados e pela quantidade de dados armazenados ou trafegados. *Em geral, tudo é fornecido por meio de um data center com servidores virtuais, em que você paga somente por aquilo que usar.*

On-Premises

Um servidor *on-premises* é aquele em que a própria empresa tem a responsabilidade de processar suas aplicações de hardware e software.

Em outras palavras, toda a infraestrutura, customização, configuração e atualização é feita internamente.

PaaS — Platform as a Service (Plataforma como Serviço)

Nesse cenário, o PaaS surge como o ideal porque é, como o próprio nome diz, *uma plataforma que pode criar, hospedar e gerir esse aplicativo.*

Nesse modelo de nuvem, contrata-se um ambiente completo de desenvolvimento, no qual é possível criar, modificar e otimizar softwares e aplicações.

Aqui, a grande vantagem é que *a equipe de desenvolvimento só precisa se preocupar com a programação do software, pois o gerenciamento, manutenção e atualização da infraestrutura ficam a cargo do fornecedor e as várias ferramentas de desenvolvimento de software são oferecidas na plataforma.*

SaaS — Software as a Service (Software como Serviço)

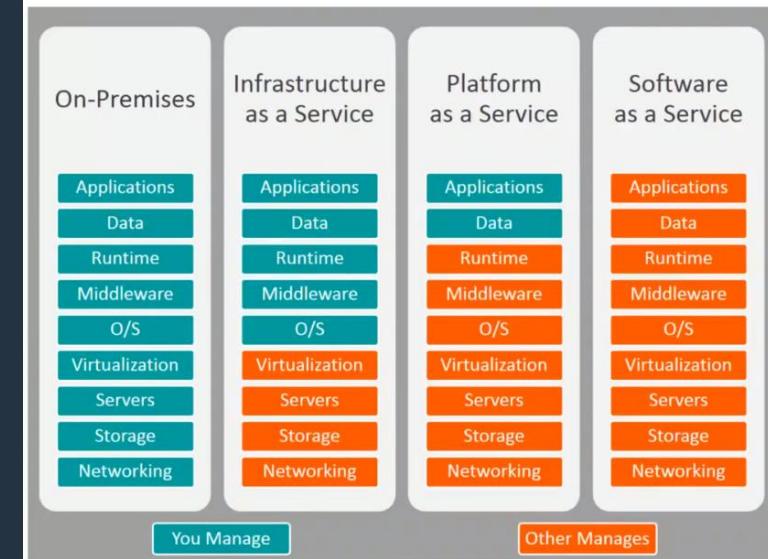
Nesse modelo de nuvem *você pode ter acesso a um software sem comprar a sua licença, utilizando-o a partir da Cloud Computing.*

Muitos CRMs ou ERPs trabalham no sistema SaaS.

Assim, o acesso a esses softwares é feito usando a internet. Os dados, contatos e demais informações podem ser acessados de qualquer dispositivo, dando mais mobilidade à equipe.

O Google Docs e o Office 365 funcionam dessa maneira.

Computação em Nuvem



DBaaS – BANCO DE DADOS COMO SERVIÇO

- **Virtualização:** permite que o banco de dados seja instalado em uma máquina virtual.
- **Dbaas:** fornece uma Plataforma flexível escalável, sob demanda que está orientada para o autoserviço e gerenciamento fácil, particularmente em termos de provisionamento de um negócio no próprio ambiente.

DBaaS oferece a funcionalidade de um banco de dados *semelhante ao que é encontrado em sistemas de gerenciamento de banco de dados relacionais (RDBMS), como o SQL Server, MySQL e Oracle.*

Sendo baseado em nuvem, por outro lado o **DBaaS fornece uma plataforma flexível escalável**, sob demanda que está orientada para o autoserviço e gerenciamento fácil, particularmente em termos de provisionamento de um negócio no próprio ambiente.

Produtos DBaaS normalmente fornecem capacidades de monitoramento suficientes para acompanhar o desempenho e o consumo e para alertar os usuários sobre possíveis problemas.

As *desvantagens* para o modelo DBaaS *incluem a falta de controle sobre os problemas de desempenho de rede, tais como falhas de latência e de aplicação inaceitáveis.*

Além disso, *alguns produtos DBaaS não suportam capacidades dos RDBMS típicos, tais como compressão de dados e partições de tabela.*

Antes de contratar um DBaaS, é essencial que a empresa avalie suas necessidades específicas e garanta que elas sejam satisfatoriamente resolvidas.

Principais Fornecedores

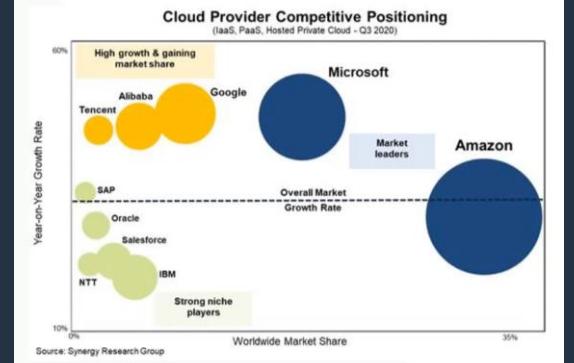


Como se diferenciam:

	Nuvem pública	Nuvem privada	Servidores dedicados	Nuvem híbrida
Descrição	Ambiente multi-inquilino com escalabilidade de pagamento pelo uso.	Escalabilidade, mais a segurança e o controle aprimorados de um ambiente de inquilino único	Para cargas de trabalho prévisíveis que exigem segurança e controle aprimorados	Conecte a nuvem pública à sua nuvem privada ou a servidores dedicados — até mesmo em seu próprio data center
Hardware físico	Compartilhado	Dedicada	Dedicada	Compartilhado + dedicado
Melhor para	Operações não sigilosas, voltadas para o público, e tráfego imprevisível	Operações sigilosas críticas e requisitos exigentes de desempenho, segurança e conformidade		

Como se diferenciam (Rackspace)

Descrição	Nuvem pública		Nuvem privada		Servidores dedicados		Nuvem híbrida	
	Expansível	Custo baixo, cobrança como serviços públicos	Ambiente multi-inquilino com escalabilidade de pagamento pelo uso.	Escalabilidade, mais a segurança e o controle aprimorados de um ambiente de inquilino único	Para cargas de trabalho prévisíveis que exigem segurança e controle aprimorados	Conecte a nuvem pública à sua nuvem privada ou a servidores dedicados — até mesmo em seu próprio data center		
Flexibilidade	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
Personalizável	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
Alto desempenho	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
Segurança e controle aprimorados	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
Custo previstível	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		



INTRODUÇÃO A AWS

<https://aws.amazon.com/pt/getting-started/>

Serviços em destaque

- Análises
- Integração de aplicações
- AR e VR
- Gerenciamento de custos da AWS
- Blockchain
- Aplicações empresariais
- Computação
- Contêineres
- Envolvimento de clientes
- Banco de dados
- Ferramentas de desenvolvedor
- Computação de usuário final
- Web e plataforma móvel front-end

Serviços em destaque

- Amazon EC2** Servidores virtuais na nuvem
- Amazon Simple Storage Service (S3)** Armazenamento escalável na nuvem
- Amazon Aurora** Banco de dados relacional gerenciado de alta performance
- Amazon DynamoDB** Banco de dados NoSQL gerenciado
- Amazon RDS** Serviço gerenciado de banco de dados relacional para MySQL, PostgreSQL, Oracle, SQL Server e MariaDB
- AWS Lambda** Execute código sem se preocupar com servidores

AWS – EC2

O Amazon **Elastic Compute Cloud** (Amazon EC2) oferece uma capacidade de computação escalável na Nuvem da Amazon Web Services (AWS).

O uso do Amazon EC2 **elimina a necessidade de investir em hardware inicialmente**, portanto, você pode desenvolver e implantar aplicativos com mais rapidez.

Você **pode usar o Amazon EC2 para executar o número de servidores virtuais que precisar**, configurar a segurança e a rede, e gerenciar o armazenamento.

O Amazon EC2 também permite a **expansão ou a redução para gerenciar as alterações de requisitos ou picos de popularidade**, reduzindo, assim, a sua necessidade de prever o tráfego do servidor.



<https://aws.amazon.com/pt/ec2/instance-types/>

A1	T4g	T3	T3a	T2	M6g	M5	M5a	M5n	M4
As instâncias A1 do Amazon EC2 oferecem economia substancial e são ideais para cargas de trabalho com escalabilidade horizontal baseadas em Arm com suporte ao ecossistema do Arm. As instâncias A1 são as primeiras instâncias do EC2 baseadas em processadores AWS Graviton, que oferecem núcleos Arm Neoverse de 64 bits e silício personalizado projetado pela AWS.									
Recursos:									
<ul style="list-style-type: none"> Processador AWS Graviton Custom personalizado com núcleos Arm Neoverse de 64 bits Supoorte a redes avançadas com até 10 Gbps de largura de banda de rede Otimizado para EBS por padrão Oferecidas pelo AWS Nitro System, uma combinação de hardware dedicado e hipervisor leve 									
Instância	vCPU	Mem (GiB)	Armazenamento	Performance de rede (Gbps)					
a1.medium	1	2	Somente EBS	Até 10					
a1.large	2	4	Somente EBS	Até 10					
a1.xlarge	4	8	Somente EBS	Até 10					
a1.2xlarge	8	16	Somente EBS	Até 10					
a1.4xlarge	16	32	Somente EBS	Até 10					
a1.meta	16*	32	Somente EBS	Até 10					

<https://aws.amazon.com/pt/ec2/instance-types/>

A1	T4g	T3	T3a	T2	M6g	M5	M5a	M5n	M4
As instâncias M4 oferecem recursos equilibrados de computação, memória e rede e são uma boa opção para muitos aplicativos.									
Recursos:									
<ul style="list-style-type: none"> Processadores Intel Xeon® E5-2686 v4 (Broadwell) de 2,3 GHz ou processadores Intel Xeon® E5-2676 v3 (Haswell) de 2,4 GHz Otimizado para EBS por padrão, sem custo adicional Supoorte a redes avançadas Equilíbrio entre recursos de computação, memória e rede 									
Instância	vCPU*	Mem (GiB)	Armazenamento	Largura de banda dedicada do EBS (Mbps)	Performance de rede				
m4.large	2	8	Somente EBS	450	Moderada				
m4.xlarge	4	16	Somente EBS	750	Alta				
m4.2xlarge	8	32	Somente EBS	1.000	Alta				
m4.4xlarge	16	64	Somente EBS	2.000	Alta				
m4.10xlarge	40	160	Somente EBS	4.000	10 Gigabit				
m4.16xlarge	64	256	Somente EBS	10.000	25 Gigabit				

<https://aws.amazon.com/pt/ec2/instance-types/>

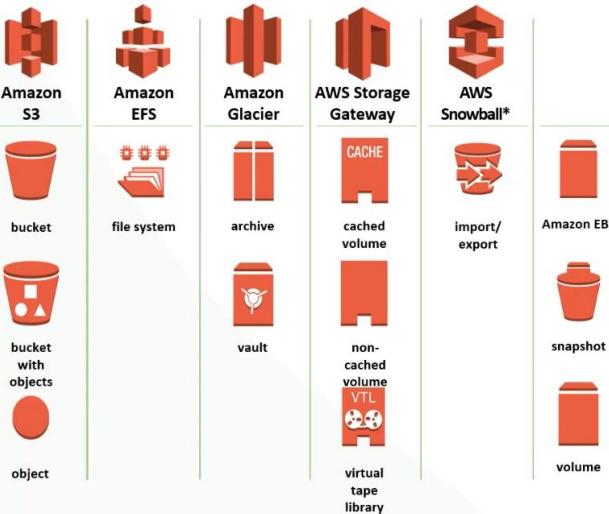
Instance	EC2 Compute Units	Memory (GiB)	Storage (GB)	Price per Hour
t2.small	variable	2.0	EBS	\$0.023
m5a.large	4	8.0	EBS	\$0.086
m5a.xlarge	8	16.0	EBS	\$0.172
c5.large	9	4.0	EBS	\$0.085
c5.xlarge	17	8.0	EBS	\$0.170

EC2 - Instâncias

General Purpose	Compute Optimised	Memory Optimised	Accelerated Computing	Storage Optimised
A1 ARM based core and custom silicon	C4 Compute - CPU intensive apps and DB's	R4 RAM - Memory intensive apps and DB's	P2 Processing optimised - Machine Learning	H1 High Disk Throughput - Big data clusters
T2 Tiny - Web servers and small DBs	X1 Extreme RAM - For SAP/Spark	G3 Graphics Intensive - Video and streaming	I3 IOPS - NoSQL DBs	
M4 Main - App servers and general purpose	z1d High Compute and High Memory - Gaming	F1 Field Programmable - Hardware acceleration	D2 Dense Storage - Data Warehousing	

INTRODUÇÃO A AWS

Storage



AWS – S3

O Amazon **Simple Storage Service** é armazenamento para a Internet. Ele foi projetado para facilitar a computação de escala na web para os desenvolvedores.

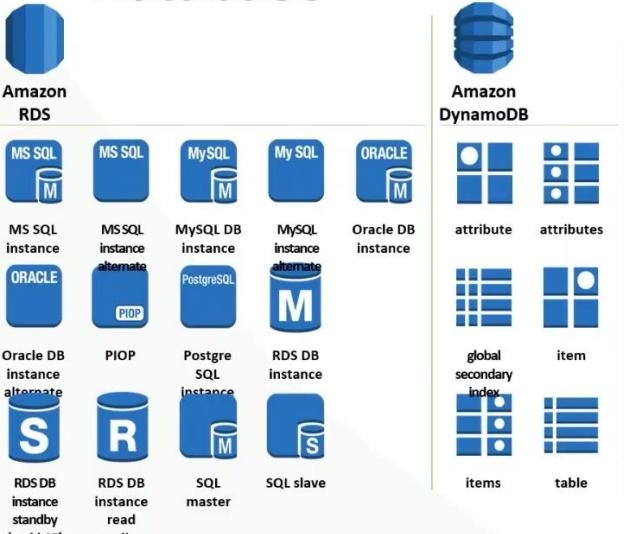
O Amazon S3 tem uma interface simples de serviços da web que você pode usar para armazenar e recuperar qualquer quantidade de dados, a qualquer momento, em qualquer lugar da web.

Elá concede acesso a todos os desenvolvedores para a mesma infraestrutura altamente dimensionável, confiável, segura, rápida e econômica que a Amazon utiliza para rodar a sua própria rede global de sites da web.

O serviço visa maximizar os benefícios de escala e poder passar esses benefícios para os desenvolvedores.

	S3 Standard	S3 Intelligent-Tiering*	S3 Standard-IA	S3 One Zone-IA†	S3 Glacier	S3 Glacier Deep Archive**
Projetado para resiliência	99,999999999% (11 9s)					
Projetado para disponibilidade	99,99%	99,9%	99,9%	99,9%	99,99%	99,99%
Acordo de nível de serviço de disponibilidade	99,9%	99%	99%	99%	99,9%	99,9%
Zonas de disponibilidade	≥3	≥3	≥3	1	≥3	≥3
Cobrança mínima de capacidade por objeto	N/D	N/D	128 KB	128 KB	40 KB	40 KB
Cobrança mínima de duração de armazenamento	N/D	30 dias	30 dias	30 dias	90 dias	180 dias
Taxa de recuperação	N/D	N/D	por GB recuperado	por GB recuperado	por GB recuperado	por GB recuperado
Latência de primeiro byte	milissegundos	milissegundos	milissegundos	milissegundos	milissegundos	milissegundos
Tipo de armazenamento	Objeto	Objeto	Objeto	Objeto	Objeto	Objeto
Transições de ciclo de vida	Sim	Sim	Sim	Sim	Sim	Sim

Database



AWS – RDS

O Amazon **Relational Database Service** (Amazon RDS) **facilita a configuração, a operação e a escalabilidade de bancos de dados relacionais na nuvem**.

O serviço **oferece capacidade econômica e redimensionável** e **automatiza tarefas demoradas de administração, como provisionamento de hardware, configuração de bancos de dados, aplicação de patches e backups**.

Dessa forma, você pode se concentrar na performance rápida, alta disponibilidade, segurança e conformidade que os aplicativos precisam.

COMPUTE	STORAGE	DATABASE
Free Tier 12 MONTHS FREE Amazon EC2 750 Hours per month	Free Tier 12 MONTHS FREE Amazon S3 5 GB of cold-store storage	Free Tier 12 MONTHS FREE Amazon RDS 750 Hours per month of auto scaling storage
Resizable compute capacity in the Cloud. 1000 hours equivalent of CPU, RAM, or GPU	Secure, durable, and scalable object storage infrastructure. 5 TB of storage	Managed Relational Database Service for MySQL, PostgreSQL, MariaDB, Oracle MySQL, or SQL Server
DATABASE	STORAGE	COMPUTE
Free Tier ALWAYS FREE Amazon DynamoDB 25 GB of storage	Free Tier 12 MONTHS FREE Amazon S3 5 GB of cold-store storage	Free Tier ALWAYS FREE AWS Lambda 1 Million free minutes per month
Fast and Flexible NoSQL database with seamless scalability. 100 TB of storage	Secure, durable, and scalable object storage infrastructure. 5 TB of storage	Compute service that runs your code in response to events and automatically manages the compute resources.
MACHINE LEARNING	COMPUTE	
Free Tier FREE TRIAL Amazon SageMaker 250 Hours per month of 12 hours maximum usage for the first two months	Free Tier ALWAYS FREE Amazon Lambda 1 Million free minutes per month	
Fully managed platform to build, train, and deploy machine learning models.		

INTRODUÇÃO AO GOOGLE CLOUD

Google Cloud Platform

Google Cloud Platform é uma suíte de computação em nuvem oferecida pelo Google, funcionando na mesma infraestrutura que a empresa usa para seus produtos dirigidos aos usuários, dentre eles o Buscador Google e o Youtube.

Juntamente com um conjunto de ferramentas de gerenciamento modulares, fornecem uma série de serviços **incluindo, computação, armazenamento de dados, análise de dados e aprendizagem de máquina.**

Compute Engine

Criação de máquinas virtuais

O Google Compute Engine é o componente **Infraestrutura como serviço do Google Cloud Platform**, construído sobre a infraestrutura global que executa o mecanismo de pesquisa do Google, Gmail, YouTube e outros serviços.

O Google Compute Engine **permite que os usuários iniciem máquinas virtuais sob demanda**. Uma máquina virtual é um computador emulado em outra máquina. Nesse caso, o servidor em nuvem executa vários sistemas operacionais em diversas outras máquinas, em vez de cada computador ter um sistema operacional próprio.

Compute Engine

O Google Compute Engine **oferece máquinas virtuais em execução nos centros de dados inovadores do Google e na rede mundial de fibra**. O suporte a ferramentas e fluxo de trabalho do Compute Engine permite dimensionar de instâncias únicas para computação em nuvem balanceada de carga global. **As VMs do Compute Engine são inicializadas rapidamente, vêm com opções de disco persistentes e locais de alto desempenho, oferecem desempenho consistente.**

- Instâncias.
- Configurações.
- Deploy de aplicação e Jobs.
- Alertas automáticos.

Storage & Databases

Amplio armazenamento de dados em nuvem.

A ideia de Plataforma como um Serviço (PaaS) já é bem difundida em muitos setores, especialmente no que diz respeito à gestão e ao armazenamento de dados.

É similar ao Google Drive, porém, em escala bem maior, o que permite a você armazenar e organizar todos os arquivos da empresa e acessá-los a partir de qualquer máquina com permissão de acesso. É ótimo para evitar a perda de documentos e para minimizar o uso do espaço físico em seu negócio.

Storage & Databases

Armazenamento de produtos escaláveis, resilientes e de alto desempenho. Bancos de dados para suas aplicações.

- Armazenamento de arquivos.
- Gerenciamento de buckets.
- Bucket Locations.
- Multi Regional.
- Regional.
- NearLine.
- CodeLine.
- Linha de comando, API, Console.

App Engine

Plataforma para criação de aplicativos Web escaláveis e backends para dispositivos móveis. O App Engine fornece serviços integrados e APIs, como datastores NoSQL, memcache e uma API de autenticação de usuário, comum à maioria dos aplicativos.

- Construir aplicações de desenvolvimentos escaláveis.
- Instâncias automáticas.
- Integrações.
- Deploy automatizado.

INTRODUÇÃO AO GOOGLE CLOUD

Big Data

Quando precisamos tomar uma decisão importante para a empresa, é importante ter dados que deem suporte a essa escolha. Ainda mais, atualmente, em um mercado cada vez mais complexo e volátil.

Totalmente gerenciado data warehousing, lote e processamento de fluxo, exploração de dados, Hadoop / Spark, e mensagens confiáveis.

- Análise de dados em bancos NoSQL.
- Construir bancos via: linha de comando, console e API's.
- Importar dados: csv, txt, integrações.

Machine Learning

Serviços de ML rápidos, escaláveis e fáceis de usar. Use modelos pré-treinados ou treine modelos personalizados em seus dados.

- Máquina de conhecimento, como funciona.
- Exemplo Case Google: E-mail e Spam.
- Google Cloud Vision API.
- Google Translate API.
- Google Speech API.

The screenshot shows the Google Cloud Platform Pricing Calculator. At the top, there's a navigation bar with links like Compute Engine, App Engine, Kubernetes Engine, Cloud Run, Vmware Engine, Cloud Storage, Networking, Cloud Load Balancer, and Bigtable. Below the navigation is a search bar with placeholder text "Search for a product you are interested in." Underneath the search bar, there are dropdown menus for "Instances", "Number of instances", "Operating System / Software", "Machine Class", and "Machine Family". The main area is titled "Google Cloud Pricing Calculator" and contains a table with columns for "Service", "Type", "Compute Engine", and "AWS EC2". The table lists various instance types and their equivalents across both providers.

Google Cloud vs AWS

O Google Cloud e o AWS são bem semelhantes. Para fazermos uma comparação seria necessário observar as categorias abaixo:

- Compute.
- Armazenamento em Blocos.
- Rede.
- Faturamento e Preços.
- Suporte e tempo de atividade.
- Segurança.

Google Cloud vs AWS

Tipo de Máquina/Instância	Google Compute Engine	AWS EC2
Compartilhado	f1-micro g1-small	t2.nano – t2.2xlarge
Padrão	n1-standard-1 – n1-standard-96 (beta)	m3.medium – m3.xlarge m4.large – m4.16xlarge
Alta memória	n1-highmem-2 – n1-highmem-96 (beta)	r3.large – r3.8xlarge r4.large – r4.16xlarge x1.16xlarge – x1.32xlarge
Alto CPU	n1-highcpu-2 – n1-highcpu-96 (beta)	c3.large – c3.8xlarge c4.large – c4.8xlarge
GPU	You can add GPUs to machine types	g2.2xlarge g2.8xlarge
Armazenamento SSD	n1-standard-1 – n1-standard-32 n1-highmem-2 – n1-highmem-32 n1-highcpu-2 – n1-highcpu-32	i2.xlarge – i2.8xlarge
Armazenamento denso	N/A	d2.xlarge – d2.8xlarge

Google Cloud vs AWS

Armazenamento em bloco	Google Cloud Platform	AWS
Serviço	SSD	300 IOPS genérico e provisionado
Tamanhos	1 GB até 64 TB	1 GB até 16 TB IOPS provisionados de 4 GB a 16 TB
IOPS máximos por volume	40.000 ler, 30.000 escrever	10.000 (20.000 para IOPS provisionados) IOPS máximo de 75.000 / instância
Rendimento Máximo por Volume (MB/s)	800 ler, 400 escrever	160 (320 para IOPS provisionadas)
Replicação	Padrão com incorporada	RAID-1
Redundância de snapshots	Múltiplas localizações	Múltiplas localizações
Encriptação	SEE 256-bit AES	SEE 256-bit AES
Encriptação	SEE 256-bit AES	SEE 256-bit AES
Preço Magnético (por GB / mês)	\$0.040 (desconto padrão)	\$0.045
Preço de SSD (por GB/mês)	\$0.170	\$0.10
Preços de SSD POP3 (por GB/mês)	N/A	\$0.125

INTRODUÇÃO AO MICROSOFT AZURE

Microsoft Azure

O Microsoft Azure é uma plataforma destinada à execução de aplicativos e serviços, baseada nos conceitos da computação em nuvem.

A apresentação do serviço foi feita no dia 27 de outubro de 2008 durante a Professional Developers Conference, em Los Angeles e lançado em 1 de Fevereiro de 2010 como Windows Azure, para então ser renomeado como Microsoft Azure em 25 de Março de 2014.

Funcionamento

Sua computação em nuvem é definida como uma combinação de software como serviço (SaaS) com computação em grid.

A computação em grid dá o poder de computação e alta escalabilidade oferecida para as aplicações, através de milhares de máquinas (hardware) disponíveis em centros de processamento de dados de última geração. De software como serviço se tem a capacidade de contratar um serviço e pagar somente pelo uso, permitindo a redução de custos operacionais, com uma configuração de infraestrutura realmente mais aderente às necessidades.

Recursos

Além dos recursos de computação, armazenamento e administração oferecidos pelo Microsoft Azure, a plataforma também disponibiliza uma série de serviços para a construção de aplicações distribuídas, além da total integração com a solução on-premise (local) baseada em plataforma .NET.

Entre os principais serviços da plataforma Windows Azure há o SQL Azure Database, Azure AppFabric Platform e uma API de gerenciamento e monitoramento para aplicações colocadas na nuvem.

Microsoft Azure x AWS

Alguns pontos em comum entre a AWS e Azure:

- Autonomia e provisionamento
- Escalabilidade instantânea
- Segurança
- Conformidade
- Gerenciamento de identidade

AWS

As formas de cobrança da AWS podem ser:

On-demand: O cálculo é feito em cima de horas ou segundos utilizados (no mínimo 60 segundos) e somente as instâncias EC2 que forem utilizadas;

Instâncias reservadas (RIs): O preço por hora é fixo, independentemente do uso, e existe um prazo pré-determinado de contratação. Essa forma de pagamento tem o benefício de obter desconto, uma vez que o cliente tem o compromisso de um a três anos;

Instâncias spot: Por serem instâncias extras, ou seja, instâncias de capacidade extra na Nuvem AWS, o preço é muito mais atrativo. Por outro lado, se o EC2 precisar de capacidade, o cliente que utiliza Instância Spot será notificado 2 minutos antes que suas instâncias serão interrompidas.



Microsoft Azure

As formas de cobrança da Azure podem ser:

On-demand: Seus custos são realizados em cima dos minutos utilizados. Neste modelo, não é necessário compromisso de tempo mínimo de contratação. Como o pagamento é feito de acordo com a utilização, é possível aumentar e diminuir recursos sem limite.

Contrato pré-definido: Como um determinado tempo de utilização é acordado, o custo é reduzido.

Acordo empresarial: Nessa modalidade, o pagamento é realizado antecipadamente, por esse motivo, há benefícios e desconto. O uso adicional é pago de forma separada, mas com desconto nas taxas.

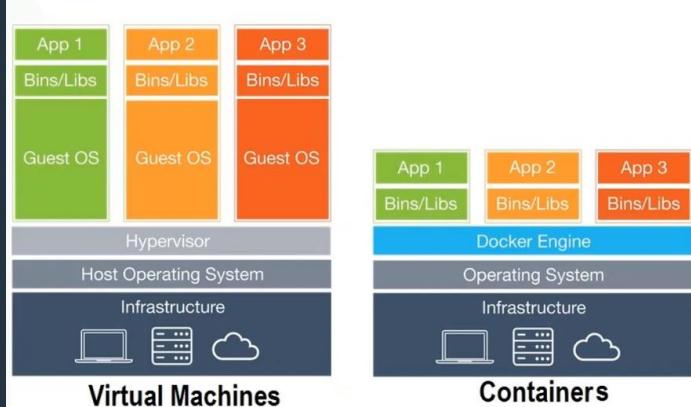
ARQUITETURA DE MICROSERVIÇOS

- Arquitetura Monolítica
- Arquitetura Microserviços
- Containers
- Orquestradores de Container

Container

Em vez de usar um sistema operacional para cada estrutura, como na virtualização, os Containers são blocos de espaços divididos pelo Docker em um servidor, o que possibilita a implementação de estruturas de Microserviços que compartilham o mesmo sistema operacional. Porém, de forma limitada (conforme a demanda por capacidade).

O fato de os Containers não terem seus próprios sistemas operacionais, permite que eles consumam menos recursos e, com isso, sejam mais leves.



Arquiteturas Monolíticas

Com as arquiteturas monolíticas, todos os processos são altamente acoplados e executam como um único serviço.

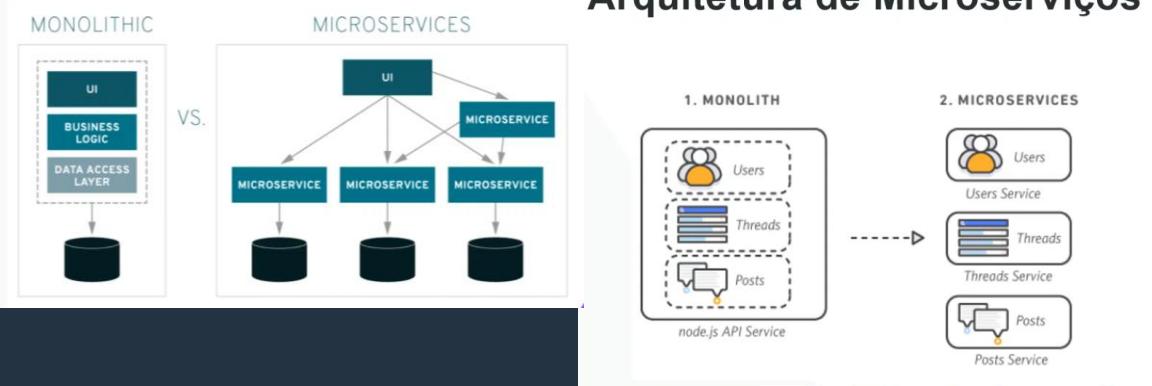
Isso significa que se um processo do aplicativo apresentar um pico de demanda, toda a arquitetura deverá ser escalada. A complexidade da adição ou do aprimoramento de recursos de aplicativos monolíticos aumenta com o crescimento da base de código. Essa complexidade limita a experimentação e dificulta a implementação de novas ideias.

As arquiteturas monolíticas aumentam o risco de disponibilidade de aplicativos, pois muitos processos dependentes e altamente acoplados aumentam o impacto da falha de um único processo.

Arquitetura de Microserviços

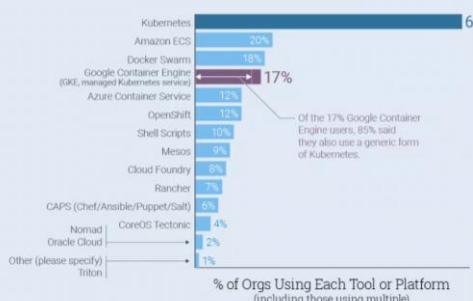
Escalabilidade flexível

Os microserviços permitem que cada serviço seja escalado de forma independente para atender à demanda do recurso de aplicativo oferecido por esse serviço. Isso permite que as equipes dimensionem corretamente as necessidades de infraestrutura, meçam com precisão o custo de um recurso e mantenham a disponibilidade quando um serviço experimenta um pico de demanda.



Orquestradores de Containers

Kubernetes Manages Containers at 69% of Organizations Surveyed

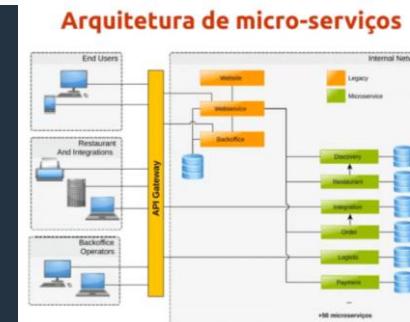


Ferramentas de Orquestração de Containers

As ferramentas de orquestração de containers são aplicações em nuvem que permitem fazer o gerenciamento de múltiplos contêineres.

Seus principais objetivos são:

- Cuidar do ciclo de vida dos containers de forma autônoma, subindo e distribuindo, conforme nossas especificações ou demandas;
- Gerenciar volumes e rede, que podem ser local ou no cloud provider de sua preferência.



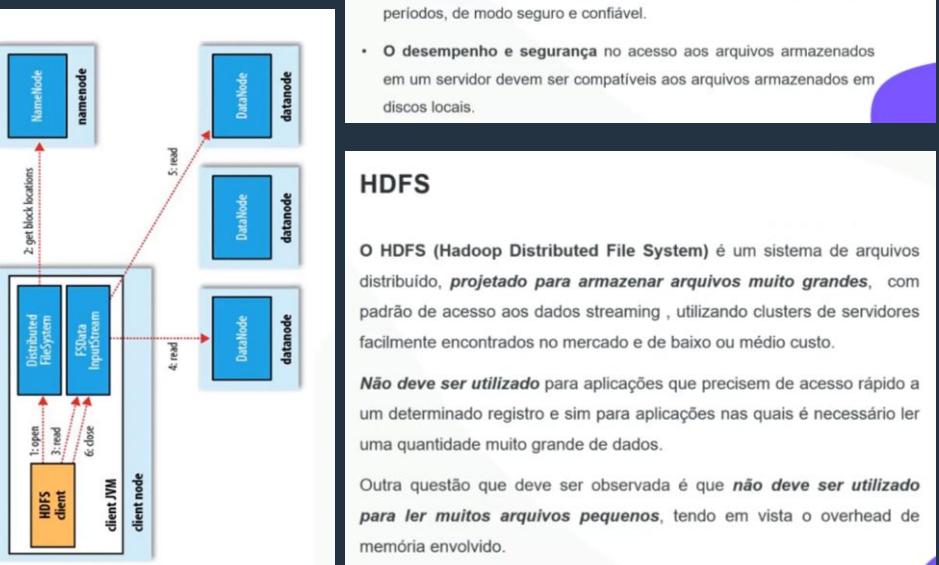
Arquitetura de Microserviços

Com uma arquitetura de microserviços, um aplicativo é criado como componentes independentes que executam cada processo do aplicativo como um serviço.

Esses serviços se comunicam por meio de uma interface bem definida usando APIs leves. Os serviços são criados para recursos empresariais e cada serviço realiza uma única função. Como são executados de forma independente, cada serviço pode ser atualizado, implantado e escalado para atender a demanda de funções específicas de um aplicativo.

SISTEMA DE ARQUIVOS DISTRIBUÍDOS

- Sistemas de Arquivo
- Sistemas de Arquivo Distribuído
- Apache Hadoop
- HDFS



HDFS

O HDFS (Hadoop Distributed File System) é um sistema de arquivos distribuído, *projeto para armazenar arquivos muito grandes*, com padrão de acesso aos dados streaming, utilizando clusters de servidores facilmente encontrados no mercado e de baixo ou médio custo.

Não deve ser utilizado para aplicações que precisem de acesso rápido a um determinado registro e sim para aplicações nas quais é necessário ler uma quantidade muito grande de dados.

Outra questão que deve ser observada é que *não deve ser utilizado para ler muitos arquivos pequenos*, tendo em vista o overhead de memória envolvido.

Sistemas de Arquivos

- Foram originalmente desenvolvidos como um *recurso do S.O.* que fornece uma interface de programação conveniente para armazenamento em disco.
- São responsáveis pela organização, armazenamento, recuperação, atribuição de nomes, compartilhamento e proteção de arquivos.
- Projetados para *armazenar e gerenciar um grande número de arquivos*, com recursos para criação, atribuição de nomes e exclusão de arquivos.

Sistemas de Arquivos Distribuídos (DFS ou SAD)

Um sistema de arquivos distribuídos *permite aos programas armazenarem e acessarem arquivos remotos exatamente como se fossem locais*, possibilitando que os usuários accedam a arquivos a partir de qualquer computador em uma rede." (COULOURIS, et. al, p. 284).

- **Objetivo:** permitir que os programas armazenem e accedam a arquivos remotos exatamente como se fossem locais.
- Permite que vários processos compartilhem dados por longos períodos, de modo seguro e confiável.
- **O desempenho e segurança** no acesso aos arquivos armazenados em um servidor devem ser compatíveis aos arquivos armazenados em discos locais.

Sistemas de Arquivos Distribuídos (DFS ou SAD)

Funções de um Sistema de Arquivos Distribuído:

- **Armazenar e compartilhar programas e dados**
 - Funções idênticas às de um sistema centralizado (local).
- **Ênfase na disponibilidade, confiabilidade e segurança.**
- **Desempenho**
 - Questão importante porque acessos remotos podem ser significativamente mais lentos que os locais.
 - Não se pretende em geral que o SAD seja mais rápido que um SA local, mas sim que a degradação seja aceitável.

Benefícios do Apache Hadoop

- **Flexibilidade** – ao contrário de sistemas de gerenciamento de banco de dados relacionais tradicionais, você não tem que esquemas estruturados criados antes de armazenar dados. *Você pode armazenar dados em qualquer formato, incluindo formatos semi-estruturados ou não estruturados*, e em seguida, analisar e aplicar esquema para os dados quando ler.
- **Baixo custo** – ao contrário de software proprietário, o *Hadoop é open source* e é executado em hardware commodity de baixo custo.

Arquitetura do SAD

- Modelo abstrato de arquitetura que serve para o *Network File System (NFS)* e *Andrew File System (AFS)*.
- Divisão de responsabilidades entre três módulos.
 - Cliente.
 - Serviço de arquivos planos.
 - Serviço de diretórios.
- Design aberto
 - Diferentes módulos cliente podem ser utilizados para implementar diferentes interfaces.
 - Simulação de operações de arquivos de diferentes S.O.
 - Otimização de performance para diferentes configurações de hardware de clientes e servidores.



Apache Hadoop

Hadoop é uma *plataforma de software de código aberto para o armazenamento e processamento distribuído de grandes conjuntos de dados, utilizando clusters de computadores com hardware commodity*.

Os serviços do Hadoop fornecem armazenamento, processamento, acesso, governança, segurança e operações de Dados.

Benefícios do Apache Hadoop

Algumas das razões para se usar Hadoop é a sua "capacidade de armazenar, gerenciar e analisar grandes quantidades de dados estruturados e não estruturados de forma rápida, confiável, flexível e de baixo custo.

- **Escalabilidade e desempenho** – distribuídos tratamento de dados local para cada nó em um cluster Hadoop permite armazenar, gerenciar, processar e analisar dados em escala petabyte.
- **Confiabilidade** – clusters de computação de grande porte são propensos a falhas de nós individuais no cluster. Hadoop é fundamentalmente resistente – quando um nó falha de processamento é redirecionado para os nós restantes no cluster e os dados são automaticamente re-replicados em preparação para falhas de nó futuras.

MONGO DB NA NUVEM

- Configurando mLab
- Conectando ao mLab

MongoDB Gratuito Hospedado na Nuvem

Conhecido um tempo atrás como MongoLab, o mLab é um serviço de banco de dados gerenciável que hospeda na nuvem um banco de dados MongoDB e é executado em provedores como a Amazon Web Services (AWS), Google Cloud e Microsoft Azure.

A parte mais interessante é que o serviço tem um plano gratuito que oferece 0,5 Gb para armazenamento de dados.



Criando Clusters

Your cluster is being created
New clusters take between 1-3 minutes to provision

RECORDS (0) - 0.00\$ (0.00) + PROJECTS

Clusters

Find a cluster

SANDBOX Cluster0 Version 4.2.10

CONNECT METRICS COLLECTIONS

CLUSTER TIER My Sandbox (General)

REGION Amazon N. Virginia (us-east-1)

TYPE Replicaset - 3 nodes

LINKED REALM APP None Linked

Guia

Connect to Atlas

Follow this checklist to get started.

40%

- ✓ Build your first cluster
- ✓ Create your first database user
- Whitelist your IP address
- Load Sample Data (Optional)
- Connect to your cluster

Primeiro Acesso

Dos planos, também temos outras três opções:

Choose a path. Adjust anytime.

Available as a fully managed service across 60+ regions on AWS, Azure, and Google Cloud

Dedicated Multi-Region Clusters

For teams developing world-class applications that require multi-region resiliency or ultra-low latency.

- ✓ Includes all features from Shared and Dedicated Clusters
- ✓ Replicate data across multiple regions
- ✓ Globally distributed read and write operations
- ✓ Control data residency at the document level

Create a cluster

Starting at \$0.13/hr*
Estimated cost \$66.95/month

Dedicated Clusters

For teams building applications that need advanced development and production environments.

- ✓ Includes all features from Shared Clusters
- ✓ Auto-scaling
- ✓ Network isolation
- ✓ Realtime performance metrics

Create a cluster

Starting at \$0.08/hr*
Estimated cost \$56.95/month

Shared Clusters

For teams learning MongoDB or developing small applications.

- ✓ Highly available auto-healing cluster
- ✓ End-to-end encryption
- ✓ Role-based access control

Create a cluster

Starting at FREE

Primeiro Acesso

Na próxima tela você terá duas escolhas a fazer: o provedor e o plano. Dos provedores, temos três possibilidades:

- Amazon Web Services (AWS)
- Google Cloud Platform
- Microsoft Azure

Feito isso, você terá a opção de escolher a região. Recomendo escolher US East (Virginia) (us-east-1) para obter uma resposta mais rápida dos servidores do mLab.

Primeiro Acesso

Na próxima tela você terá duas escolhas a fazer: o provedor e o plano. Dos provedores, temos três possibilidades:

- Amazon Web Services (AWS)
- Google Cloud Platform
- Microsoft Azure

Cloud Provider & Region



AWS, N. Virginia (us-east-1) ▾

★ Recommended region

ASIA

Singapore (ap-southeast-1) ★

NORTH AMERICA

N. Virginia (us-east-1) ★

EUROPE

Frankfurt (eu-central-1) ★

Mumbai (ap-south-1)

Oregon (us-west-2) ★

Ireland (eu-west-1) ★

END



mongoDB®