Apresentação

O que é MongoDB?

- Banco de dados NoSQL orientado a documentos.
- Utiliza JSON/BSON para armazenar dados.
- Altamente escalável e flexível.

Por que usar Clusters?

- Alta disponibilidade
- Recuperação automática de falhas
- Escalabilidade horizontal

📌 O que é um Cluster?

- Um **cluster** é um conjunto de computadores (ou instâncias/contêineres) que trabalham juntos como se fossem um único sistema. Em bancos de dados, isso serve para garantir **alta disponibilidade**, **tolerância a falhas** e **escalabilidade**.
- No contexto do MongoDB, um cluster pode ser configurado para que o sistema continue funcionando mesmo que um dos servidores falhe — isso é chamado de **Replica Set**.

Conjunto de Réplicas (Replica Set)
O que é?

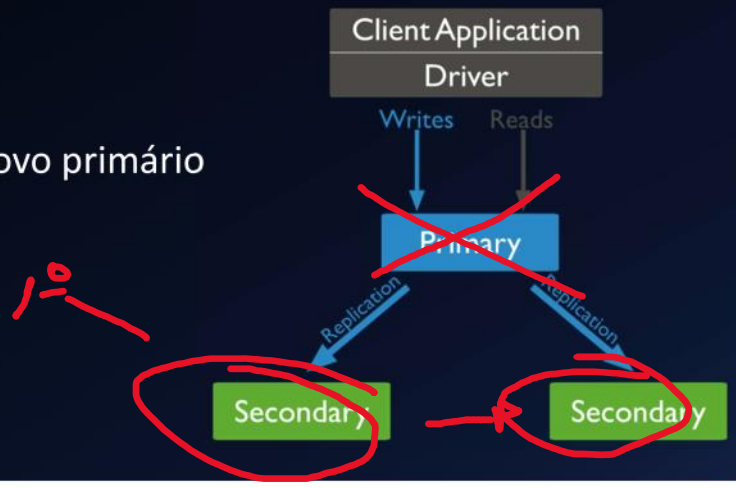
- Grupo de nós MongoDB que mantêm cópias idênticas dos dados.Utiliza JSON/BSON para armazenar dados.

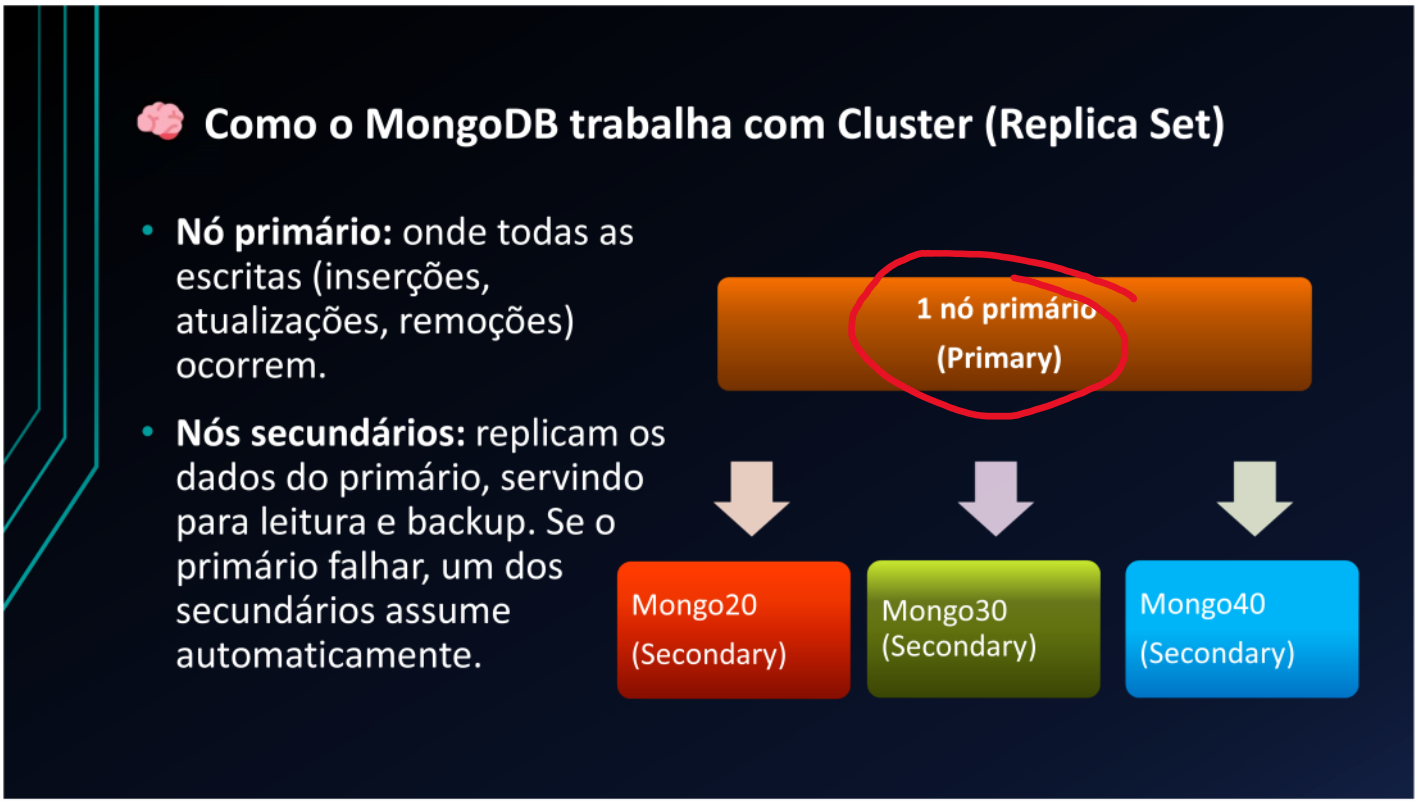
Benefícios:

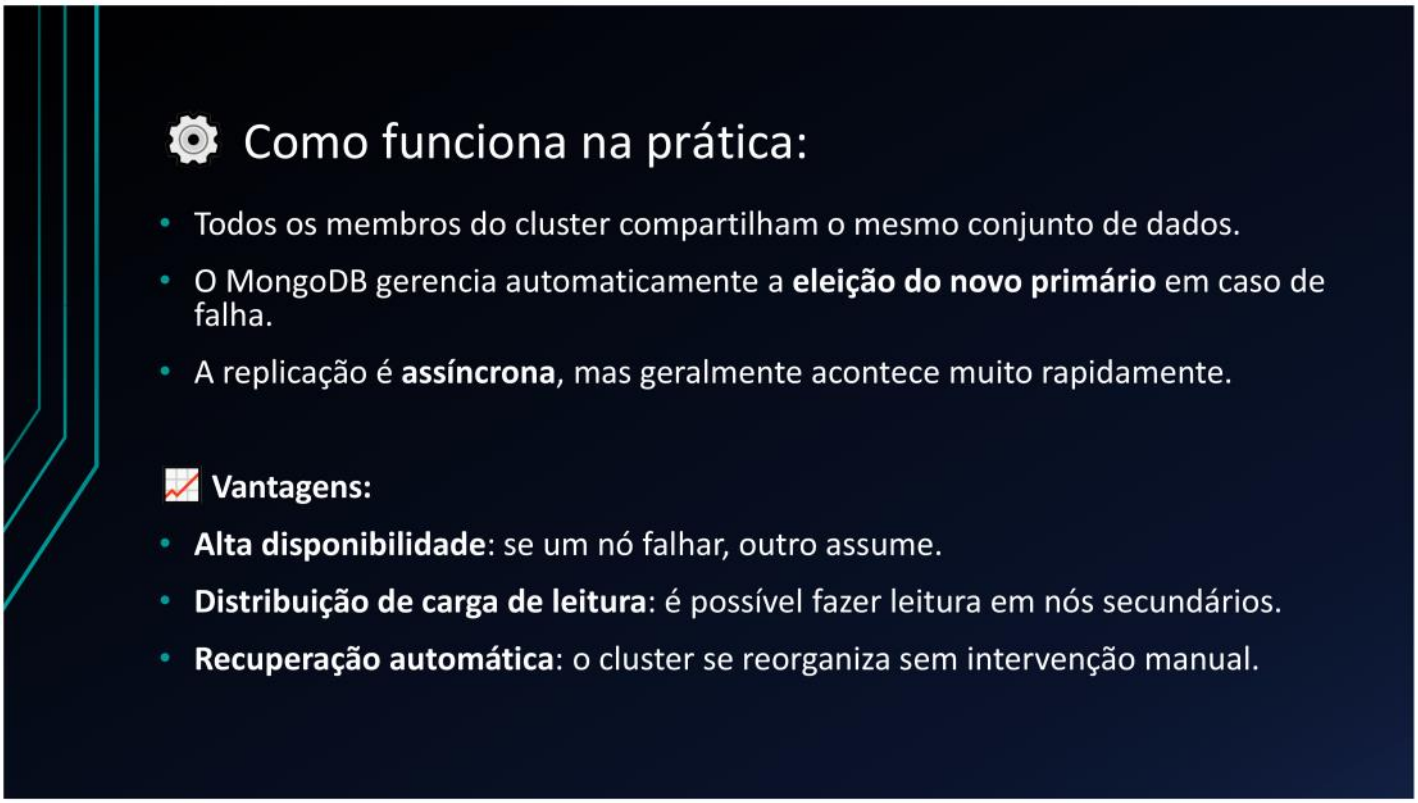
- Redundância
- Eleição automática de um novo primário

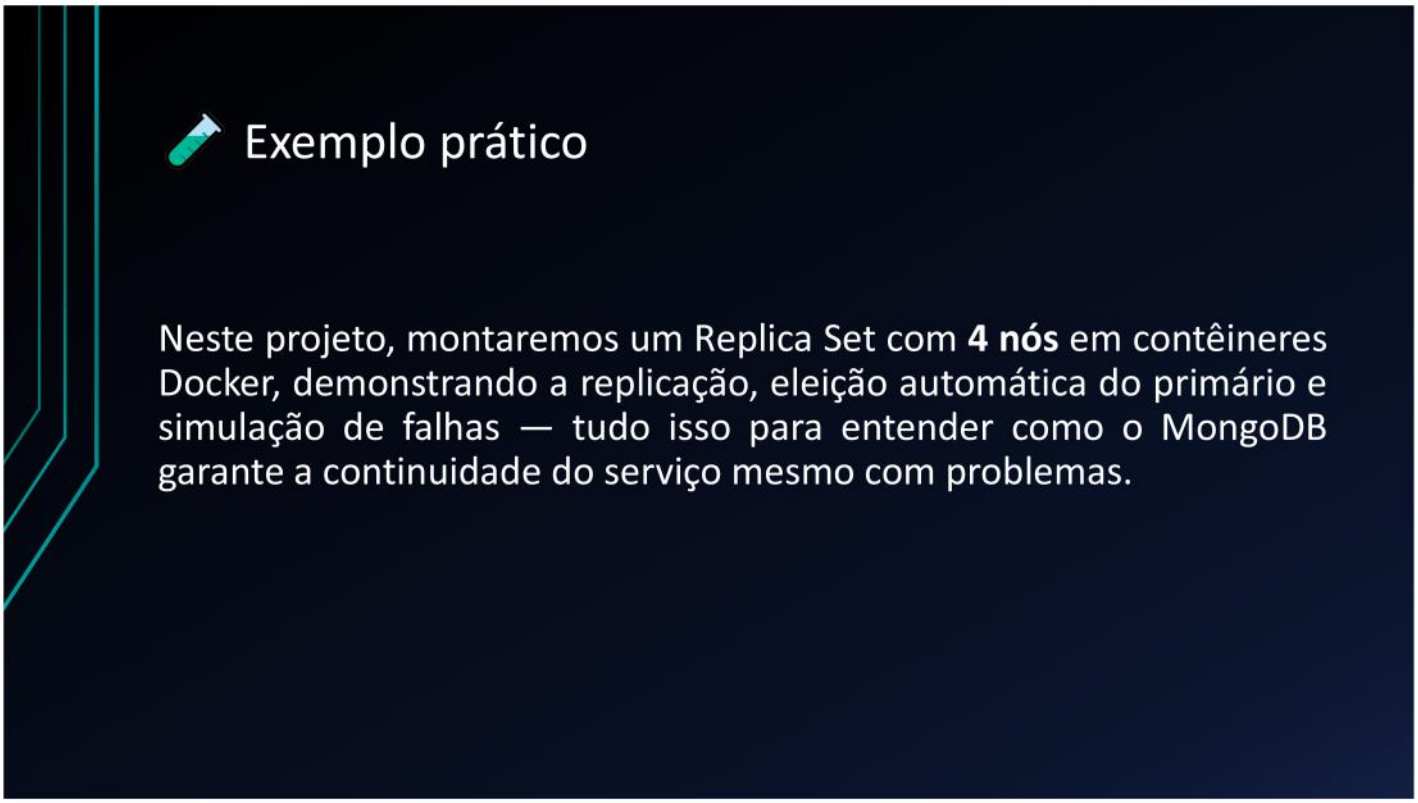
em caso de falha;

- Leitura escalável.




Como o MongoDB trabalha com Cluster (Replica Set)
Nó primário: onde todas as escritas (inserções, atualizações, remoções) ocorrem.
Nós secundários: replicam os dados do primário, servindo para leitura e backup. Se o primário falhar, um dos secundários assume automaticamente.
1 nó primário
(Primary)
Mongo20
(Secondary)
Mongo30
(Secondary)
Mongo40
(Secondary)


⚙ Como funciona na prática:
• Todos os membros do cluster compartilham o mesmo conjunto de dados.
• O MongoDB gerencia automaticamente a **eleição do novo primário** em caso de falha.
• A replicação é **assíncrona**, mas geralmente acontece muito rapidamente.
📈 **Vantagens:**
• **Alta disponibilidade**: se um nó falhar, outro assume.
• **Distribuição de carga de leitura**: é possível fazer leitura em nós secundários.
• **Recuperação automática**: o cluster se reorganiza sem intervenção manual.


Exemplo prático
Neste projeto, montaremos um Replica Set com **4 nós** em contêineres Docker, demonstrando a replicação, eleição automática do primário e simulação de falhas — tudo isso para entender como o MongoDB garante a continuidade do serviço mesmo com problemas.

Criando a rede Docker

Primeiro, precisamos criar uma rede Docker para garantir que os contêineres possam se comunicar entre si.

- [ ] `docker network create` **`mongoCluster`**

Explicação:

- O comando docker network create cria uma rede Docker chamada mongoCluster.
- Esse passo é necessário para que os contêineres possam se comunicar entre si. Ao executar esse comando, todos os nós do MongoDB serão capazes de se "ver" na rede e trabalhar juntos como parte do **Replica Set**.

Subindo os Contêineres MongoDB (4 Nós)

Agora, vamos iniciar as 4 instâncias do MongoDB em contêineres. Usamos o comando docker run para iniciar cada contêiner, passando as opções para configurá-los corretamente.

- [x] `docker run -d --rm -p 27017:27017 --name mongo10 --network mongoCluster mongodb/mongodb-community-server:latest --replSet myReplicaSet --bind_ip localhost,mongo10`
- [x] `docker run -d --rm -p 27018:27017 --name mongo20 --network mongoCluster mongodb/mongodb-community-server:latest --replSet myReplicaSet --bind_ip localhost,mongo20`
- [ ] `docker run -d --rm -p 27019:27017 --name mongo30 --network mongoCluster mongodb/mongodb-community-server:latest --replSet myReplicaSet --bind_ip localhost,mongo30`
- [ ] `docker run -d --rm -p 27020:27017 --name mongo40 --network mongoCluster mongodb/mongodb-community-server:latest --replSet myReplicaSet --bind_ip localhost,mongo40`

`docker run -d`	Executa o contêiner em segundo plano.
`--rm`	Remove o contêiner quando ele for parado.
`-p 27017:27017`	Mapeia a porta 27017 do contêiner para a porta 27017 da máquina host (para comunicação com o MongoDB).
`--name mongo10`	Nome do contêiner. Este nome é importante para que o MongoDB saiba como se conectar com o contêiner
`--network mongoCluster`	Conecta o contêiner à rede Docker mongoCluster, que permite que os contêineres se comuniquem entre si.
`mongodb/mongodb-community-server:latest`	Usa a imagem do MongoDB Community Server mais recente.
`--replSet myReplicaSet`	Inicia o MongoDB com o **Replica Set** chamado myReplicaSet.
`--bind_ip localhost,mongo10`	(para o mongo10): Define os IPs nos quais o MongoDB pode se conectar (localhost e o nome do contêiner mongo10). Para os outros nós, o nome será alterado para mongo20, mongo30, e mongo40.

Importante: O MongoDB precisa de um replSet para funcionar em um cluster de replica set, e o parâmetro `--bind_ip` permite que ele se conecte corretamente aos outros contêineres.

Inicializando o Replica Set com os 4 Nós

Depois de criar os contêineres, o próximo passo é inicializar o Replica Set. Vamos fazer isso acessando um dos contêineres com o **MongoDB Shell** (mongosh) e executando o comando `rs.initiate()`.

Primeiro, entraremos no contêiner do mongo10:

- [ ] `docker exec -it mongo10 mongosh`

- [ ] `Copiar o "Connecting to" no notepad, iremos usar em breve...`

Verifique se o container docker conectado está executando, com o comando

- [ ] `db.runCommand ({hello:1})`

Agora, dentro do mongosh, executaremos o comando de inicialização do Replica Set:

- [ ]
```
   rs.initiate({
  _id: "myReplicaSet",
  members: [
    {_id: 0, host: "mongo10"},
    {_id: 1, host: "mongo20"},
    {_id: 2, host: "mongo30"},
    {_id: 3, host: "mongo40"}
  ]
})
```

Explicação:

- `rs.initiate()`: Comando para inicializar um Replica Set no MongoDB.
- `_id: "myReplicaSet"`: Define o nome do Replica Set como myReplicaSet.
- `members`: Lista de membros do Replica Set. Cada nó tem um identificador único (`_id`), e o nome do host é o nome do contêiner.

Dê o comando exit para sair do container shell

- `exit`

Importante: Os contêineres devem estar rodando corretamente antes de executar o `rs.initiate()`.

Testando o Status do Replica Set

Agora, podemos testar o status do Replica Set e verificar se todos os nós foram corretamente adicionados.

☑ `docker exec -it mongo10 mongosh --eval "rs.status()"`

Explicação:

- `rs.status()`: Exibe o status atual do Replica Set, mostrando detalhes como o estado dos membros e qual nó é o primário.

📦 Acesso via MongoDB Compass

Depois de abrir o Compass e colar a **string de conexão do replica set**, aquela que haviámos copiado no notepad.

- Ver que está conectado a um **replica set (myReplicaSet)**;
- Observar se o nó atual é primário ou secundário;
- Usar a aba **“Collections”** para navegar entre bancos e coleções.

Saber em qual nó estamos conectado

Com o shell aberto no MongoDB Compass, devemos rodar:

- [ ] `rs.isMaster().primary`

Esse comando mostra qual dos nós está como **primário** no momento.

Inserção de Dados

Você só pode **inserir dados no nó primário**. Tente executar:

```
use CorporeSystem

db.cliente.insertOne({codigo: 1, nome: "Matheus"});
```

Depois, liste os dados:

```
db.cliente.find()
```

Se estivermos conectado ao **primário**, o dado será inserido com sucesso.

🔍 Testando a Tolerância a Falhas (Failover)

Vamos simular a queda do nó primário:

- [ ] `docker stop mongo10`

O cluster irá detectar a falha e **elegerá um novo nó primário** automaticamente.

Agora verificaremos o status do cluster após a queda, utilizaremos o comando:

```
docker exec -it mongo20 mongosh --eval "rs.status()"
```

Veremos que um novo primário foi eleito.

🗀Conectando diretamente ao novo nó primário

Usaremos a string de conexão da instância que se tornou primária.

Agora podemos continuar inserindo dados normalmente:

- `db.cliente.insertOne({codigo: 3, nome: "Eduardo"})`

Reiniciando o nó que caiu

Se quiser **trazer de volta** o mongo10, use:

- [ ] `docker run -d --rm -p 27017:27017 --name mongo10 --network mongoCluster mongodb/mongodb-community-server:latest --replSet myReplicaSet --bind_ip localhost,mongo10`

Depois verifique novamente o status do cluster:

- [ ] `docker exec -it mongo20 mongosh --eval "rs.status()"`

Agora o mongo10 voltará como **secundário**, mantendo a alta disponibilidade do cluster.

🛠️Adicionar um ARBITER (Opcional)

O que é um Arbiter?

Um **arbiter** é um nó do MongoDB que **não armazena dados**, mas participa das eleições para ajudar a manter **um número ímpar de votos**, o que **evita empates** na hora de eleger o primário.

Por que adicionar?

- Nosso cluster atualmente tem 4 nós (número par).
- Isso **pode causar empate** em uma eleição (2x2).
- Com um arbiter, você tem 5 votos → sempre haverá um vencedor.

Como adicionar um arbiter:

1. Subir o container:

```
docker run -d --rm --name arbiter --network mongoCluster mongodb/mongodb-community-server:latest --replSet myReplicaSet --bind_ip localhost,arbiter
```

2. Adicionar no cluster:

Conecta em qualquer nó :

```
docker exec -it mongo20 mongosh
```

Adiciona o arbiter:

```
rs.addArb("arbiter")
```

Verifica se foi adicionado:

rs.status()

Vai aparecer como:

```
{
  "_id" : 4,
  "name" : "arbiter",
  "arbiterOnly" : true
}
```