

NewSQL: Suporte à Replicação e Propriedades ACID

Processamento Massivo de Dados

Prof. Dr. Sahudy Montenegro González

Bruno Bevilacqua Rizzi Giovanni Alvarenga

brunorizzi@estudante.ufscar.br giovanni.alvarenga@estudante.ufscar.br

05/01/2021

Sumário

1. Introdução

SQL e ACID

NoSQL / Replicação e *Sharding*

NewSQL

2. Ambiente

Estudo de Caso - Northwind

Docker

SingleStore

CockroachDB

Versões dos Softwares

3. Instalação dos Softwares

Requisitos Recomendados

Docker

SingleStore

CockroachDB

4. Propriedades NewSQL

Replicação e Sharding

Propriedades ACID

5. Benchmark

6. Conclusão

7. Referências Bibliográficas

Introdução

SQL / ACID

Um Banco de Dados Relacional é um tipo de banco de dados que armazena e fornece acesso a pontos de dados que estão relacionados entre si. Os bancos de dados relacionais são baseados no modelo relacional, uma maneira intuitiva e direta de representar dados em tabelas. Esse tipo de banco de dados é implementado por um Sistema Gerenciador de Banco de Dados Relacional (SGBDR) que utiliza da linguagem SQL no seu manuseio.

SQL / ACID

Um dos atrativos desse tipo de sistema são as propriedades ACID[1] que são um conjunto de propriedades destinadas a garantir a validade dos dados mesmo na ocorrência de erros, falhas de energia e outros contratempos. Essas propriedades são:

- Atomicidade - Uma transação será executada por completo ou não será executada.
- Consistência - A transação cria um novo estado válido dos dados ou em caso de falha retorna todos os dados ao seu estado antes que a transação foi iniciada.
- Isolamento - Uma transação em andamento mas ainda não validada deve permanecer isolada de qualquer outra operação.
- Durabilidade - Dados validados são registrados pelo sistema de tal forma que mesmo no caso de uma falha e/ou reinício do sistema, os dados estão disponíveis em seu último estado correto.

NoSQL / Replicação e *Sharding*

Um banco de dados NoSQL fornece um mecanismo para armazenamento e recuperação de dados que são modelados diferente das relações tabulares usadas em bancos de dados relacionais. Os bancos de dados NoSQL são cada vez mais usados em aplicativos da web de Big Data. Muitos sistemas NoSQL comprometem a consistência (no sentido do teorema CAP) em favor da disponibilidade, tolerância de partição e velocidade.

NoSQL / Replicação e *Sharding*

A replicação se refere a uma configuração de banco de dados na qual várias cópias do mesmo conjunto de dados são hospedadas em máquinas separadas. O principal motivo para haver replicação é a redundância. Se uma única máquina host do banco de dados ficar inativa, a recuperação será rápida, pois uma das outras máquinas que hospeda uma réplica do mesmo banco de dados pode assumir o controle.

O *Sharding*, também frequentemente chamado de particionamento, envolve a divisão de grandes dados com base em *keys*. Isso aumenta o desempenho porque reduz o impacto sobre cada um dos recursos individuais, permitindo que eles compartilhem a carga em vez de ter todos os dados e carga concentrados em um só lugar.

NewSQL

NewSQL é uma classe de sistemas de gerenciamento de banco de dados relacional que busca fornecer a escalabilidade de sistemas NoSQL enquanto mantém as propriedades ACID de um sistema de banco de dados tradicional. Muitos sistemas empresariais que lidam com dados *high-profile* (por exemplo, sistemas financeiros e de processamento de pedidos) são muito grandes para bancos de dados relacionais convencionais, mas têm requisitos transacionais e de consistência que não são práticos para sistemas NoSQL.

O objetivo desse tutorial é apresentar um passo-a-passo de como podemos criar um banco de dados em dois sistemas NewSQL diferentes e como eles disponibilizam e atendem as propriedades explicadas acima.

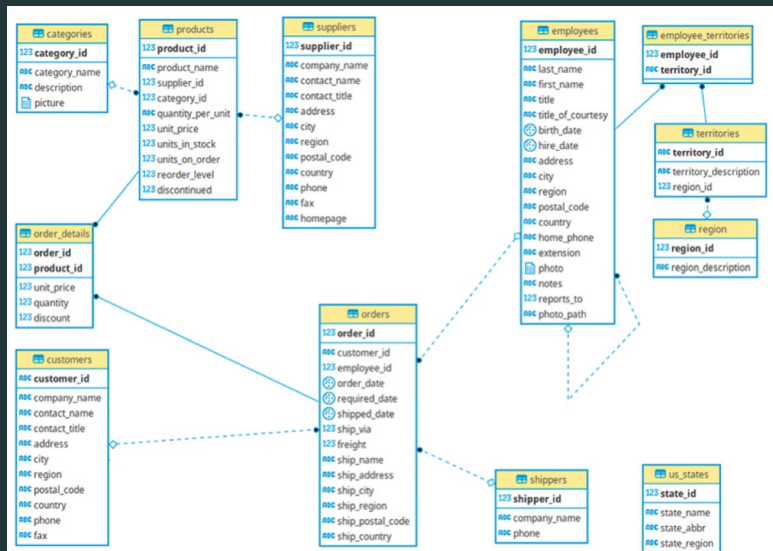
Ambiente

Northwind

O banco de dados Northwind[2] é um banco de dados amostral que foi originalmente criado pela Microsoft e usado como base para seus tutoriais em uma variedade de produtos de banco de dados por décadas. O banco de dados Northwind contém os dados de vendas de uma empresa fictícia chamada “Northwind Traders” , que importa e exporta alimentos especiais de todo o mundo. O banco de dados Northwind é um excelente esquema tutorial para um ERP de pequenas empresas, com clientes, pedidos, estoque, compras, fornecedores, remessas, funcionários e contabilidade de entrada única. O banco de dados Northwind desde então foi transferido para uma variedade de bancos de dados não Microsoft, incluindo PostgreSQL e MySQL.

Esquema do Northwind

A imagem abaixo representa o esquema do Northwind utilizado.



Considerações sobre o Northwind

- Para esse tutorial foi utilizado uma variação do esquema original, não contendo as tabelas `customer_customer_demo` e `customer_demographics`.
- Tipos de atributos que não são suportados pelos sistemas utilizados foram trocados pelo tipo `VARCHAR`.
- Outro detalhe importante é que, para o sistema `SingleStore`, os atributos com a *constraint* `FOREIGN KEY` tiveram que ser retirados, pois o sistema não suporta tal restrição.

Docker

Na elaboração deste tutorial foi utilizado o Docker[3] para a criação de *containers* com os sistemas escolhidos, isso foi feito para facilitar as demonstrações que irão ocorrer sobre os temas abordados.

Docker é um serviço que utiliza da virtualização de de sistemaa operacionais para entregar *containers*. Os *containers* são isolados uns dos outros e agrupam seus próprios softwares, bibliotecas e arquivos de configuração. Eles podem se comunicar uns com os outros por meio de caminhos definidos na sua inicialização. Todos os *containers* são executados por um único kernel do sistema operacional e, portanto, usam menos recursos do que as máquinas virtuais.

SingleStore

O SingleStore é um sistema de gerenciamento de banco de dados SQL (RDBMS) distribuído, relacional, que oferece suporte a ANSI SQL e é conhecido pela velocidade na inserção de dados, processamento de transações e processamento de consultas.

SingleStore era conhecido anteriormente como MemSQL.

O SingleStore armazena principalmente dados relacionais, embora também possa armazenar dados JSON, dados de gráfico e dados de série temporal. Para consultas, ele compila SQL em código de máquina.

A arquitetura do SingleStore tem muitas semelhanças com os bancos de dados NoSQL existentes, além de ter uma estrutura de banco de dados relacional e suporte a SQL. Ele também obedece as propriedades ACID.

CockroachDB

O CockroachDB é um sistema de gerenciamento de banco de dados SQL (RDBMS) distribuído, relacional, com as seguintes características:

- Dados Distribuídos, replicados e com armazenamento chave-valor transacional.
- Dados armazenados em *Key-Spaces* Monolíticos.
- Replicação *Raft* de intervalos de 64MB.
- Dados considerados para distribuição das réplicas: Espaço, Diversidade, Carregamento (*Load*) e Latência.
- As transações re-organizam o armazenamento dos dados.
- Mapeamento de dados SQL para armazenamento *Key-Value*.
- Execução SQL distribuída.

Versões

- Os sistemas operacionais utilizados foram o Ubuntu 20.04 e o MacOS Catalina v.10.15.7
- As versões utilizadas do Docker foram 20.10.1 e 20.10.0, para cada sistema operacional, respectivamente.
- A versão utilizada do SingleStore no container foi a 7.3.2 no sistema Ubuntu.
- A versão utilizada do CockroachDB no container foi a 20.2.3 no sistema MacOS.

Instalação dos Softwares

Requisitos Recomendados

SingleStore

- 8 vCPU por máquina host.
- Pelo menos 4 GB por núcleo, mínimo de 32 GB por nó folha.
- Forneça um sistema de armazenamento para cada nó com pelo menos 3 vezes a capacidade da memória principal. O armazenamento SSD é recomendado.

CockroachDB

- 4 GiB de RAM por vCPU
- 150 GiB de armazenamento por vCPU
- 500 IOPS por vCPU (I/O do disco)
- 30 MB/s por vCPU (I/O do disco)

Docker

Para não estender esse tutorial não será ensinado o passo-a-passo para a instalação do Docker, todavia o link a seguir contém um passo-a-passo oficial de instalação para os principais sistemas operacionais.

<https://docs.docker.com/get-docker/>

SingleStore

Para a instalação do SingleStore você, primeiramente, precisará se registrar no link abaixo e obter uma *License Key* para poder utilizar um mês do sistema gratuitamente.

<https://www.singlestore.com/try-free/>

Após ter a licença em mãos, defina a mesma e a senha do seu usuário *root* que será criado no banco de dados.

1

```
1 export LICENSE_KEY="sua licença"
2 export ROOT_PASSWORD="sua senha"
```

Feito isso já podemos criar o *container*. Na primeira criação ele irá pegar a imagem necessária para a instalação do repositório do Docker.

2

```
1 sudo docker run -i --init --name memsql-pmd -e LICENSE_KEY=\$LICENSE_KEY -e ROOT_PASSWORD=\$ROOT_PASSWORD  
2 -p 3306:3306 -p 8080:8080 memsql/cluster-in-a-box
```

Criado o *container* podemos inicializá-lo.

3

```
1 sudo docker start memsql-pmd
```

Pronto! Seu *container* já está criado.

Podemos agora carregar o banco de dados no nosso *container*. Há várias maneiras de fazer isso, ensinaremos uma delas.

Inseriremos o DLL e os dados da base de dados no nosso banco. Para isso copiaremos os arquivos `.sql` para dentro do *container*.

4

```
1 sudo docker cp \${HOME}/Northwind_DLL_MemSQL.sql memsql-pmd:/
2 sudo docker cp \${HOME}/Northwind_Data_MemSQL.sql memsql-pmd:/
```

Feito isso, entraremos no terminal do *container*, criaremos o banco de dados Northwind e carregaremos o esquema e os dados.

5

```
1 sudo docker exec -it memsql-pmd /bin/bash
```

6

```
1 memsql --password="sua senha"
```

7

```
1 CREATE DATABASE northwind;  
2 EXIT;
```

8

```
1 #Navegue ate a pasta aonde você copiou os arquivos .sql no seu container  
2 memsql -p northwind < Northwind_DLL_MemSQL.sql  
3 memsql -p northwind < Northwind_Data_MemSQL.sql
```

■ Pronto! O conjunto de dados foi carregado.

CockroachDB

Para a instalação[4] do CockroachDB você, primeiramente, precisará baixar a versão oficial do banco de dados no site do *cockroachlabs*.

<https://www.cockroachlabs.com/docs/v20.2/install-cockroachdb-mac.html>

Para criar o cluster, utilizaremos 5 containers que representarão 5 máquinas nas quais nosso banco de dados estará armazenado. Os containers terão a seguinte estrutura:

- Roach1: Master;
- Roach2: Folha;
- Roach3: Folha;
- Roach4: Folha;
- Roach5: Folha;

Para a comunicação entre os containers, utilizaremos uma rede interna do Docker, que será criada com o seguinte comando:

1

```
1 docker network create -d bridge roachnet
```

Este comando irá criar nossa rede interna, chamada roachnet.

Para mais informações sobre as Redes Bridge do docker, acesse o link abaixo.

<https://docs.docker.com/network/bridge/>

■ Criamos então os volumes Docker para cada um de nossos *containers*.

2

```
1 docker volume create roach1
2 docker volume create roach2
3 docker volume create roach3
4 docker volume create roach4
5 docker volume create roach5
```

■ Inicializamos o primeiro *container*, que trabalhará como *Master-Aggregator*, com o seguinte comando.

3

```
1 docker run -d --name=roach1 --hostname=roach1 --net=roachnet -p 26257:26257 -p 8080:8080
2 -v "roach1:/cockroach/cockroach-data" cockroachdb/cockroach:v20.2.3 start --insecure
3 --join=roach1,roach2,roach3,roach4,roach5
```

- **docker run:** Comando Docker para criar um novo container.
- **-d:** Inicializa o container em background.
- **-name:** Nome do container para referencias futuras, opcional.
- **-hostname:** Nome do host para o container.
- **-net:** The bridge network for the container to join. See step 1 for more details.
- **-p 26257:26257 -p 8080:8080:** Mapeamento da porta padrão para a comunicação e da porta 8080 para as requests do DB, possibilitando assim acesso ao DB Console pelo navegador.
- **-v "roach1:/cockroach/cockroach-data":** Esta flag monta um diretório no host como um volume de dados. Os dados e logs deste nó serão armazenados no host, no volume de dados roach1 e persistirão mesmo após a deleção do container. Ver o tópico de volume na documentação do Docker para mais informações.
- **cockroachdb/cockroach:v20.2.3 start -insecure -join:** O comando no CockroachDB para inicializar um nó no modo inseguro (sem segurança). A flag **-join** especifica o nome do host de cada nó que inicialmente ira compor seu cluster.

Agora, com o primeiro nó ativo, podemos prosseguir com a inicialização dos outros dois nós que serão utilizados nesta primeira etapa. Para isso, precisamos rodar os codigos abaixo em nosso terminal.

Inicializar o *container* roach2:

4

```
1 docker run -d --name=roach2 --hostname=roach2 --net=roachnet -v "roach2:/cockroach/cockroach-data"
2 cockroachdb/cockroach:v20.2.3 start --insecure --join=roach1,roach2,roach3,roach4,roach5
```

Inicializar o *container* roach3:

5

```
1 docker run -d --name=roach3 --hostname=roach3 --net=roachnet -v "roach3:/cockroach/cockroach-data"
2 cockroachdb/cockroach:v20.2.3 start --insecure --join=roach1,roach2,roach3,roach4,roach5
```

■ Agora, inicializamos o cluster que contém nossos *containers*:

6

```
1 docker exec -it roach1 ./cockroach init --insecure
```

■ Temos então nosso cluster roach1 funcionando, com 3 *containers*, sendo o roach1 mestre e roach2 e roach3 folhas.

A inserção dos dados no banco pode ser feita de forma parecida com a mostrada no SingleStore, mudando apenas o nome do *container* para entrar no bash do mesmo. Para acessar o bash do CockroachDB dentro do *container* utilizamos agora:

7

```
1 ./cockroach sql --insecure
```

■ Ao invés de:

8

```
1 memsql --password="sua senha"
```

Propriedades NewSQL

Replicação e Sharding - SingleStore

Nesta sessão, iremos testar a replicação e o *sharding* do banco de dados SingleStore, que já contém 2 nós (um master e outro folha) e adicionaremos mais 3 nós folha para verificar a redistribuição.

Com nosso container ligado podemos entrar na interface gráfica fornecida pelo SingleStore no link abaixo.

<http://localhost:8080/connect>

Entraremos no *LocalHost* com o usuário *root* e a senha que você definiu na etapa de instalação.

Podemos ver nas duas imagens abaixo a quantidade de nós do nosso banco e o particionamento das tabelas do Northwind.

Por padrão o SingleStore cria um nó *Master Aggregator* e um nó folha para guardar as réplicas do banco.

Na imagem abaixo podemos perceber que temos apenas o nó *Master Aggregator* e um único nó folha, na qual está as réplicas da partições.

SingleStore Studio

localhost:8080/cluster/localhost/nodes

STUDIO

Localhost root

OVERVIEW

- Dashboard
- Events
- Hosts
- Nodes
- Databases

QUERY

- SQL Editor
- Activity Resource Usage
- Active Processes
- Visual Explain

INGEST

- Pipelines

Nodes

Last updated: 10:24:03 PM

Node Address	State	Partition Instance Status	Role	CPU Usage	Memory Usage	Disk Usage
127.0.0.1:3306	Online	Partitions Online	Master Aggregator	0.67%	2.51% 276 MB/10.7 GB	76.02% 22 GB/28.9 GB
127.0.0.1:3307	Online	Partitions Online	Leaf	1.16%	3.23% 356 MB/10.7 GB	76.02% 22 GB/28.9 GB

Console Connection Log

?

Já nesta imagem podemos ver quantas partições o singleStore criou do nosso banco e que todas elas estão no nó de porta 3307, ou seja, nosso único nó folha.

SingleStore Studio

localhost:8080/cluster/localhost/databases/northwind/partitions

DATABASE INFO

- Memory Usage: 10 MB
- Disk Usage: 0 B
- Partition Count: 8
- Table Count: 12
- Replication Type: Synchronous
- DR Replica: Database is not a secondary replica
- Nodes Used: 1 / 1

Partitions

Partition	Health Status	Total Partition Instances	Partition Type	Master Partition Instance Node	Replica Partition Instance Node
northwind	Online	2	Reference Partition	127.0.0.1:3306	All Leaf Nodes
northwind:0	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind:1	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind:2	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind:3	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind:4	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind:5	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind:6	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind:7	Online	1	Data Partition	127.0.0.1:3307	N/A

Console Connection Log

Podemos observar que o SingleStore cria réplicas do banco e particiona as tabelas automaticamente sem a necessidade de requisitar explicitamente.

Porém quando mais nós são adicionados precisaremos rodar um comando para balancear as partições entre os novos nós. Demonstraremos isso a seguir.

■ Adicionaremos mais 3 nós com os códigos a seguir, nas portas 3008, 3009 e 3010.

1

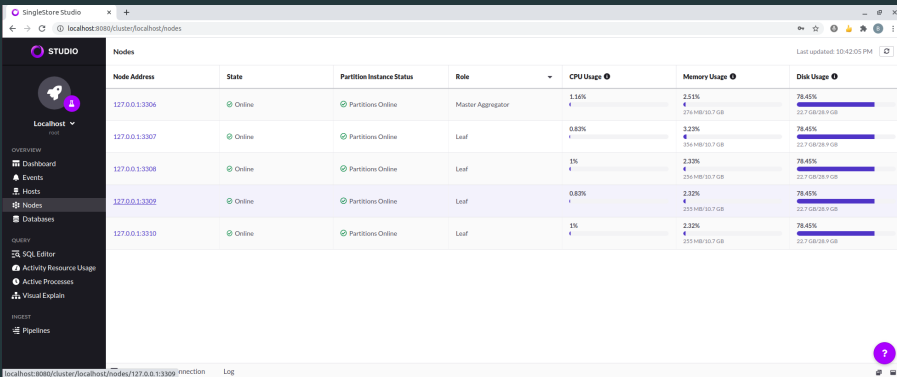
```
1 sudo docker exec -it memsql-pmd memsqlctl create-node --port 3308 --password=pmd
2 sudo docker exec -it memsql-pmd memsqlctl create-node --port 3309 --password=pmd
3 sudo docker exec -it memsql-pmd memsqlctl create-node --port 3310 --password=pmd
```

■ Com os nós criados, transformaremos eles em folha para serem reconhecidos pelo banco.

2

```
1 sudo docker exec -it memsql-pmd memsqlctl add-leaf --host 127.0.0.1 --port 3308 --password=pmd
2 sudo docker exec -it memsql-pmd memsqlctl add-leaf --host 127.0.0.1 --port 3309 --password=pmd
3 sudo docker exec -it memsql-pmd memsqlctl add-leaf --host 127.0.0.1 --port 3310 --password=pmd
```

Podemos ver nas imagens abaixo que os nós foram criados porém o nó de porta 3307 continua sendo o único com as partições.



SingleStore Studio

localhost:8080/cluster/localhost/nodes

STUDIO

Localhost

OVERVIEW

- Dashboard
- Events
- Hosts
- Nodes
- Databases

QUERY

- SQL Editor
- Activity Resource Usage
- Active Processes
- Visual Explain

INVEST

- Pipelines

Nodes

Last updated: 10:42:05 PM

Node Address	State	Partition Instance Status	Role	CPU Usage	Memory Usage	Disk Usage
127.0.0.1:3306	Online	Partitions Online	Master Aggregator	1.14%	2.51%	78.45%
127.0.0.1:3307	Online	Partitions Online	Leaf	0.83%	3.22%	78.45%
127.0.0.1:3308	Online	Partitions Online	Leaf	1%	2.33%	78.45%
127.0.0.1:3309	Online	Partitions Online	Leaf	0.83%	2.32%	78.45%
127.0.0.1:3310	Online	Partitions Online	Leaf	1%	2.32%	78.45%

localhost:8080/cluster/localhost/nodes/127.0.0.1:3309/rnnection Log

Porém, como podemos ver na imagem abaixo, o SingleStore não rebalanceia automaticamente as partições nos novos nós, assim para que as partições sejam distribuídas nos nós folha criados teremos que explicitamente pedir para o sistema realizar o rebalanceamento.

SingleStore Studio

localhost:8080/cluster/localhost/databases/northwind/partitions

Databases > northwind / Partitions

Last updated: 10:12:38 PM

DATABASE INFO

- Memory Usage: 10 MB
- Disk Usage: 0 B
- Partition Count: 8
- Table Count: 12
- Replication Type: Synchronous
- DR Replica: Database is not a secondary replica
- Nodes Used: 1 / 4

Partition	Health Status	Total Partition Instances	Partition Type	Master Partition Instance Node	Replica Partition Instance Node
northwind	Online	5	Reference Partition	127.0.0.1:3306	All Leaf Nodes
northwind.0	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind.1	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind.2	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind.3	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind.4	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind.5	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind.6	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind.7	Online	1	Data Partition	127.0.0.1:3307	N/A

Console Connection Log

Nos conectaremos novamente ao bash do *container* e a linha de comando do SingleStore, e executaremos o comando a seguir.

3

```
1 REBALANCE PARTITIONS ON northwind;
```

Pronto! As partições foram distribuídas entre os novos nós.

Databases > northwind / Partitions

Last updated: 10:59:33 PM

DATABASE INFO

- Memory Usage: 10 MB
- Disk Usage: 0 B
- Partition Count: 8
- Table Count: 12
- Replication Type: Synchronous
- DR Replica: Database is not a secondary replica
- Nodes Used: 4 / 4

Schema | **Partitions**

Partition	Health Status	Total Partition Instances	Partition Type	Master Partition Instance Node	Replica Partition Instance Node
northwind	Online	5	Reference Partition	127.0.0.1:3306	All Leaf Nodes
northwind:0	Online	1	Data Partition	127.0.0.1:3308	N/A
northwind:1	Online	1	Data Partition	127.0.0.1:3309	N/A
northwind:2	Online	1	Data Partition	127.0.0.1:3310	N/A
northwind:3	Online	1	Data Partition	127.0.0.1:3308	N/A
northwind:4	Online	1	Data Partition	127.0.0.1:3309	N/A
northwind:5	Online	1	Data Partition	127.0.0.1:3310	N/A
northwind:6	Online	1	Data Partition	127.0.0.1:3307	N/A
northwind:7	Online	1	Data Partition	127.0.0.1:3307	N/A

Console | Connection | Log

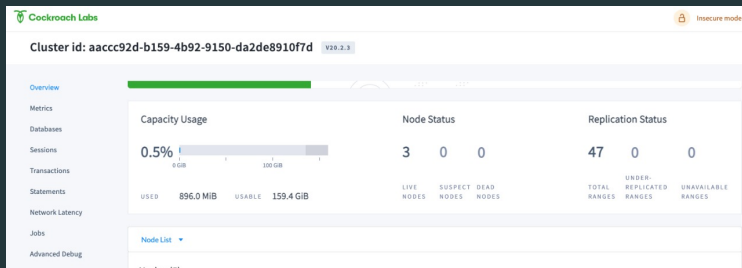
Replicação e Sharding - CockroachDB

Nesta sessão, iremos testar a replicação e o Sharding do banco de dados Cockroach, criando inicialmente 3 nós (um master e 2 folha) e depois adicionaremos mais 2 nós folha para verificar a redistribuição.

Da mesma forma que o SingleStore podemos acessar a interface gráfica do CockroachDB no link abaixo.

<http://localhost:8080/connect>

Na imagem a seguir podemos verificar a quantidade de clusters e o número de réplicas em cada um deles.



Replicação e Sharding - CockroachDB

Para validarmos o funcionamento da replicação e do *sharding*, primeiramente precisamos entender como o CockroachDB realiza estas tarefas.

Replicação: por padrão, o CockroachDB cria 3 réplicas de cada um dos *shards* para poder armazená-los em nós diferentes, no nosso caso, *containers* diferentes.

Sharding: por padrão, o Cockroach DB armazena os dados em espaços de 64MB, utilizando um algoritmo de eleição de líder que segue o protocolo Raft (para mais informações acesse o link abaixo).

https:

[//www.cockroachlabs.com/docs/stable/architecture/replication-layer.html](https://www.cockroachlabs.com/docs/stable/architecture/replication-layer.html)

O CockroachDB realiza automaticamente o balanceamento e redistribuição dos dados, conforme mais nós são adicionados ao banco.

Vamos criar dois novos containers para nosso banco de dados, executando os códigos abaixo.

Roach 4:

1

```
1 docker run -d --name=roach4 --hostname=roach4 --net=roachnet -v "roach4:/cockroach/cockroach-data"
2 cockroachdb/cockroach:v20.2.3 start --insecure --join=roach1,roach2,roach3,roach4,roach5
```

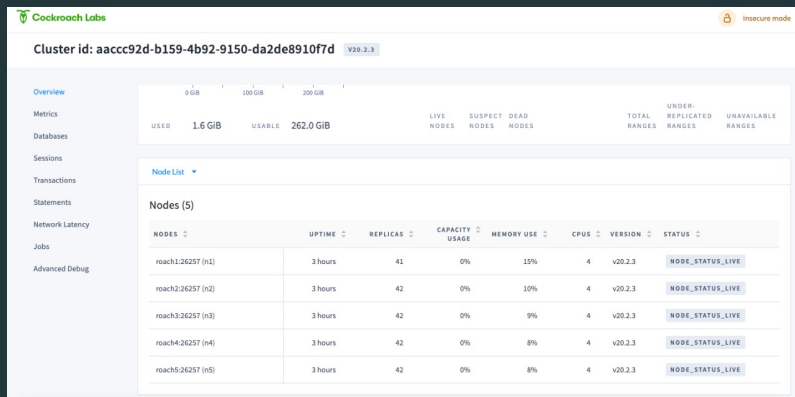
Roach 5:

2

```
1 docker run -d --name=roach5 --hostname=roach5 --net=roachnet -v "roach5:/cockroach/cockroach-data"
2 cockroachdb/cockroach:v20.2.3 start --insecure --join=roach1,roach2,roach3,roach4,roach5
```

Replicação e Sharding - CockroachDB

Podemos verificar as alterações nos containers e em seu número de réplicas na imagem abaixo.



Propriedades ACID

Devido a limitações técnicas não foi possível trazer exemplos das propriedades ACID[1] nos sistemas. Contudo a documentação dos dois sistemas garantem o cumprimento dessas propriedades [5, 6, 7].

Mais informações podem ser encontradas nos links abaixo.

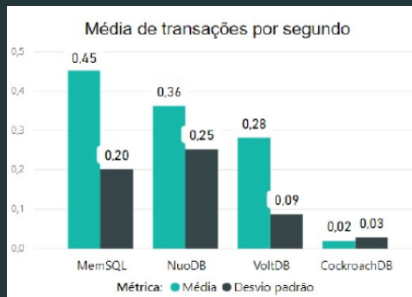
<https://www.singlestore.com/blog/how-memsql-works/>

[https:](https://www.cockroachlabs.com/blog/how-cockroachdb-distributes-atomic-transactions)

[/www.cockroachlabs.com/blog/how-cockroachdb-distributes-atomic-transactions](https://www.cockroachlabs.com/blog/how-cockroachdb-distributes-atomic-transactions)

Benchmark

O artigo *Uma Análise de Soluções NewSQL* nos proporciona uma análise completa sobre a performance de vários sistemas NewSQL, realizando testes com 2 Benchmarks, YCSB e Voter. O resultado destes testes apontou o MemSQL como o banco de dados com os melhores resultados de transações e latência, enquanto que o CockroachDB ficou em última posição.



https://www.researchgate.net/publication/336867128_Uma_Analise_de_Solucoes_NewSQL

Conclusão

Este tutorial teve como objetivo apresentar o conceito de NewSQL com uma abordagem prática, utilizando dois SGBDs (Cockroach e SingleStore) para criar e gerir o conjunto de dados Northwind, a fim de demonstrar as propriedades de replicação e sharding nestes bancos.

O tutorial apresenta o passo-a-passo da instalação até testes básicos de escalabilidade dos sistemas.

Com este tutorial, identificamos que o NewSQL tem um grande potencial de escalabilidade e garante as propriedades ACID, sendo um paradigma promissor para os próximos anos. No entanto, por ser muito recente, ainda há poucas opções viáveis e a maior parte das que existem não possuem uma boa documentação, dificultando desde sua instalação até o real uso.

Referências Bibliográficas



Propriedades acid.

<https://medium.com/opensanca/o-que-e-acid-59b11a81e2c6>.

Accessed: 12-01-2021.



Northwind.

<https://docs.yugabyte.com/latest/sample-data/northwind/>.

Accessed: 12-01-2021.



Docker documentation.

<https://docs.docker.com/>.

Accessed: 12-01-2021.



Cockroachdb - local cluster.

<https://www.cockroachlabs.com/docs/stable/start-a-local-cluster-in-docker-mac.html>.

Accessed: 12-01-2021.



How singlestore works.

<https://www.singlestore.com/blog/how-memsql-works/>.

Accessed: 12-01-2021.



How singlestore works - 2.

<https://docs.singlestore.com/v7.1/introduction/how-memsql-works/>.

Accessed: 12-01-2021.



Cockroachdb - acid.

[https://www.cockroachlabs.com/blog/
how-cockroachdb-distributes-atomic-transactions/](https://www.cockroachlabs.com/blog/how-cockroachdb-distributes-atomic-transactions/).

Accessed: 12-01-2021.