

# Configure uma arquitetura Apache Cassandra básica

Bruno Reded Tinoco ([brunocrt@gmail.com](mailto:brunocrt@gmail.com))

08/Fev/2018

Software Architect

GFT

Saiba como configurar e instalar um banco de dados Apache Cassandra distribuído, com alta disponibilidade, com a capacidade de manipular petabytes de dados sem nenhum ponto único de falha e com baixa latência de leitura e gravação. Esses tipos de banco de dados são usados por diversas grandes empresas e instituições que buscam um alto nível de serviço em suas soluções de banco de dados.

Embora você já esteja ouvindo falar sobre big data e bancos de dados NoSQL por um bom tempo, talvez você ainda não tenha tido a oportunidade de começar a trabalhar com eles. Apesar de saber que um banco de dados NoSQL poderia ajudar em alguns de seus projetos, talvez você não tenha confiança suficiente para realizar essa tarefa. Nesse caso, este tutorial está aqui para ajudar.

Vou orientar a instalação de um dos armazenamentos de big data mais interessantes e resilientes disponíveis: o Apache Cassandra. Este é um tutorial prático direcionado a desenvolvedores e administradores de banco de dados com um conhecimento básico de bancos de dados relacionais. Vou abordar os principais aspectos do Cassandra em detalhes e indicar outras fontes para informações adicionais.

Para testar a instalação do ambiente, eu uso os utilitários do Cassandra e um script Python para consumir os dados armazenados no Cassandra. Não se preocupe se você não está familiarizado com essas ferramentas, porque elas não são obrigatórias para a instalação, elas são usadas somente para validar a configuração e funcionam como recursos adicionais.

## Conceitos básicos

Nesta seção, eu explico alguns dos detalhes herdados pelo Cassandra como banco de dados distribuído. Se você já tem algum conhecimento desses conceitos ou se não está interessado na teoria agora, vá diretamente para [Desenvolva o plano](#).

## Teorema de CAP

CAP significa "consistency, availability, and partition tolerance" (consistência, disponibilidade e tolerância de partição). O teorema de CAP, formulado inicialmente por Eric Brewer em 2000, afirma que somente duas dessas propriedades, no máximo, podem ser obtidas em qualquer

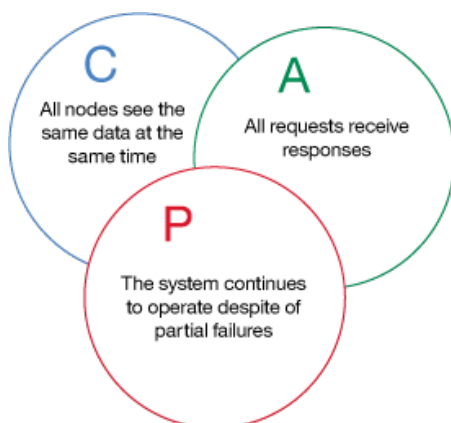
sistema de dados compartilhados. Portanto, deve-se escolher duas, não é possível ter todas. Para saber mais sobre o teorema, consulte "Tópicos relacionados" abaixo, mas eu apresentarei uma visão geral.

É importante entender o teorema de CAP em relação ao Cassandra, porque ele pode fazê-lo chegar à conclusão de que o Cassandra não é o mais adequado para sua solução de banco de dados NoSQL. De qualquer maneira, ele o ajudará a começar a pensar nas restrições da solução em termos de consistência e disponibilidade.

De acordo com o teorema, para qualquer sistema distribuído deve-se escolher as duas garantias mais importantes para o seu sistema (veja a Figura 1). É possível ter todas as três garantias no Cassandra, mas *não ao mesmo tempo*. Portanto, se você deseja um banco de dados altamente disponível sem tempos de inatividade e não deseja que ocorram eventuais falhas de hardware, o Cassandra é o mais adequado para as soluções que se concentram em disponibilidade e tolerância de partição.

Isso se contrasta com as propriedades *ACID* (atomicidade, consistência, isolamento, durabilidade) de um sistema de gerenciamento de banco de dados relacional (RDBMS) tradicional, como MySQL, DB2®, Oracle e Sybase. Eu não quero dizer que no Cassandra não há operações *atômicas* e que os dados do Cassandra não são *isolados* ou *duráveis*. Eu simplesmente quero dizer que esses não são os enfoques principais do Cassandra. O banco de dados foi criado para ser distribuído nativamente e para ser dimensionado facilmente à medida que os dados e as transações de aplicativos aumentam.

## Figura 1. As garantias do teorema de CAP e o Cassandra

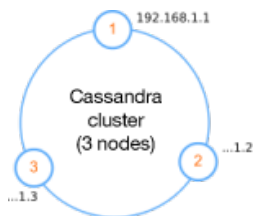


## Banco de dados distribuído

O Cassandra é um banco de dados distribuído por natureza. Isso significa que ele foi desenvolvido para ser executado em uma rede de nós de computação como um servidor com diferentes partes em execução em diferentes máquinas sem nenhum hardware ou software específico para gerenciá-lo ou coordená-lo. Toda a coordenação dos nós e a distribuição de dados ocorrem dentro de sua própria arquitetura. Esse é um dos motivos pelos quais é mais fácil e mais barato dimensionar horizontalmente uma rede do Cassandra do que outros sistemas comuns de banco de dados relacionais.

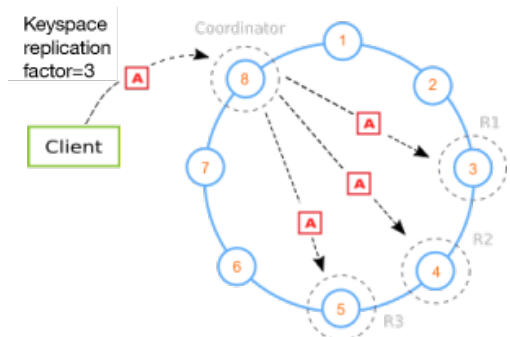
A topologia de rede típica do Cassandra é composta de um cluster de nós, também chamado de anel do Cassandra, em execução em diferentes endereços de rede localizados em diferentes servidores físicos.

**Figura 2. Cluster de nós do Cassandra em diferentes hosts de rede**



Esse recurso aumenta a disponibilidade da rede no caso de falha do nó. Cada nó pode coordenar a solicitação de um cliente sem um nó principal, portanto, não há um ponto único de falha. Isso também permite definir diferentes estratégias de configuração para que os dados fiquem cientes de diferentes locais de nó, aumentando ainda mais a disponibilidade do sistema.

**Figura 3. Cluster de oito nós do Cassandra recebendo uma conexão do cliente gravando dados em um keyspace configurado com um fator de replicação igual a 3**



Todos os dados são distribuídos uniformemente no anel do Cassandra (nós), de acordo com um algoritmo hash para criar o número de cópias necessário, também chamado de *réplicas*. O fator de replicação é um aspecto importante da configuração de cluster. Ele é definido por um keyspace ou uma configuração de esquema.

Todas as informações sobre os dados do cluster, a topologia, a disponibilidade do nó e o desempenho são trocadas entre os nós por meio do *protocolo gossip*, um tipo de protocolo ponto a ponto. Essas informações são importantes para avisar as conexões do cliente sobre qual nó é o melhor para gravar ou ler qualquer dado em um determinado momento.

Os clientes do Cassandra podem se comunicar com o servidor usando dois protocolos: o *protocolo CQL binário* ou um protocolo RPC chamado *thrift*. O protocolo CQL binário é um protocolo mais recente e é preferencial em relação ao *thrift*. Cassandra Query Language (CQL) é uma linguagem semelhante à SQL, usada pelo Cassandra para criar comandos e manipular sua estrutura de esquema e seus dados (DDL e DML).

## Estrutura de dados básica e modelagem

Um aspecto importante e, às vezes, delicado do Cassandra é sua abordagem da modelagem de dados. Primeiro, é necessário entender como seus dados são organizados dentro de sua arquitetura e, em seguida, como modelar a estrutura de dados do aplicativo para aproveitar o máximo de seu desempenho.

No Cassandra, todos os dados são organizados por partições com uma chave primária (chave de linha), que fornece acesso a todas as colunas ou conjuntos de pares chave-valor como é mostrado abaixo.

### Figura 4. Partição da estrutura de dados do Cassandra

Row Key	Column key 1	Column key 2	...	Column key n
	Column value 1	Column value 2		Column value n

A chave primária no Cassandra pode conter duas chaves especiais: a *chave da partição* e (opcionalmente) a *chave de armazenamento em cluster*. O propósito da chave de partição é espalhar os dados uniformemente no cluster. A tarefa da chave de armazenamento em cluster (também chamada de "colunas de armazenamento em cluster") é armazenar em cluster e organizar os dados de uma partição para permitir consultas eficientes. Considere o exemplo a seguir.

Ao criar uma tabela do Cassandra, você usa um comando CQL semelhante ao seguinte:

```
CREATE TABLE movie_catalog (category text, year int, title text,  
PRIMARY KEY (category));
```

A primeira coluna é considerada implicitamente como a chave da partição da tabela `movie_catalog`. Não existe uma chave de armazenamento em cluster. No entanto, suponhamos que você inclua a coluna `year` dentro da chave primária, desta forma:

```
CREATE TABLE movie_catalog (category text, year int, title text,  
PRIMARY KEY (category,year))
```

Agora a `category` continua sendo a chave da partição, enquanto a coluna `year` passa a ser a chave de armazenamento em cluster. As duas colunas fazem parte da chave primária.

Se você achar tudo isso muito confuso, não pense muito! O importante é saber que todas as tabelas do Cassandra precisam ter uma chave primária para localizar o nó no qual os dados estão no cluster. Essa chave é composta de pelo menos uma chave da partição. Como indicado acima, uma chave de armazenamento em cluster, usada para localizar dados dentro do nó (partição), também pode fazer parte da chave primária.

Para modelar suas tabelas, deve-se escolher a chave da partição com atenção para permitir que o Cassandra faça uma boa distribuição de dados entre os nós. Não é interessante colocar todos os dados do aplicativo (linhas) em uma única partição. Pelo mesmo token, também é possível

ter *um grande número* de partições. Portanto, é necessário encontrar um bom equilíbrio no agrupamento dos dados para satisfazer os requisitos do aplicativo.

A técnica mais usada para modelagem no Cassandra é chamada de *Modelagem baseada em consulta*. Essa abordagem requer que você analise quais consultas a interface com o usuário do aplicativo emitirá primeiro. Em seguida, você modela as tabelas com base nessas consultas. Esse tópico é um possível assunto para um próximo tutorial.

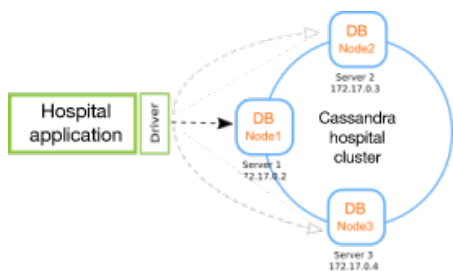
## Desenvolva o plano

Imagine que você precisa desenvolver uma camada de banco de dados para uma arquitetura de aplicativo crítica que armazenará informações de exames de pacientes de um grande hospital. Esse sistema exigirá um tempo de atividade de 24 horas, 7 dias por semana e atenderá muitos usuários. A primeira preocupação é que o banco de dados precisa de alta disponibilidade e precisa ser tolerante a falhas para que não prejudique os usuários e as operações do hospital. Na próxima seção, eu descrevo uma possível solução.

## Visão geral da solução

Você decide configurar inicialmente um cluster de teste básico de três nós para o ambiente de teste (UAT). Depois disso, eles serão implementados em três máquinas de servidores diferentes no datacenter do hospital em produção.

### Figura 5. Aplicativo do hospital acessando a camada de banco de dados dos servidores executando um cluster do Cassandra de três nós



A ideia de ter um cluster de três nós em vez de apenas um nó visa aumentar a disponibilidade do sistema de banco de dados. No caso de falha em um nó, ainda haverá dois nós operando para responder a todas as solicitações do aplicativo. Além disso, é possível balancear a carga de solicitações no caso de aumento de carga, diminuindo a latência do aplicativo durante a leitura e a gravação de dados.

Os drives de clientes do Cassandra podem descobrir automaticamente todos os nós disponíveis e escolher o melhor nó coordenador que será responsável por gravar todas as cópias ou réplicas dos dados. Tudo isso é possível devido à implementação do protocolo gossip do Cassandra que troca informações de funcionamento sobre o nó entre seus nós pares.

## Fator de replicação

A próxima etapa é decidir qual fator de replicação e nível de consistência serão usados para o aplicativo. Depois disso, é possível começar a instalar e configurar o ambiente de tempo de execução do banco de dados e criar esquemas e tabelas para o aplicativo.

Na definição de esquema, você configura um fator de replicação igual a 3. Isso significa que você configura o esquema do banco de dados (keyspace) para criar três cópias dos dados nos três nós. Sempre que o aplicativo se conectar a um nó e inserir um item de dados em uma tabela, ele será replicado para outros dois nós automaticamente. Isso trará mais confiança para armazenar os dados com segurança.

```
CREATE KEYSPACE patient WITH replication = {'class': 'SimpleStrategy',  
      'replication_factor' : 3};
```

## Nível de consistência

Também é necessário definir o nível de consistência a ser usado pelo aplicativo durante a sessão de leitura e gravação quando um cliente for conectado. Isso ajudará a determinar o grau de consistência que as consultas precisam ter em relação ao estado dos dados. Por exemplo, é possível optar por um nível de *consistência QUORUM* para gravar e ler dados, o que significa que você forçará o Cassandra a gravar e ler dados para a maioria dos nós (dois nós) antes de retornar uma solicitação.

O nível de consistência é definido em cada sessão do cliente e pode ser alterado a qualquer momento. Como exemplo, no shell do cliente `cqlsh` mostrado abaixo, é possível testar o nível de consistência a qualquer momento antes das consultas da mesma maneira que os drivers do Cassandra permitem.

```
cqlsh:patient> consistency QUORUM ;  
Consistency level set to QUORUM.  
cqlsh:patient>
```

Existe uma relação entre a consistência e o desempenho de latência. Quanto maior a consistência, maior o tempo das operações de leitura e gravação. Em nossa instalação de cluster básica, isso não impõe nenhuma diferença de tempo significativa, mas é um conceito importante a ser aplicado em grandes clusters do Cassandra. Nesse caso, você estará lidando com grandes quantidades de dados e ainda precisará de um tempo de resposta rápido.

## O que você precisará?

Um banco de dados Cassandra é baseado na plataforma Java™ e pode ser executado nos inúmeros sistemas operacionais com suporte para a tecnologia Java, com espaço em disco pequeno e memória disponível para começar a trabalhar. Para o aplicativo descrito neste tutorial, vou recomendar o seguinte:

- **No mínimo 2 GB de RAM disponível**— Para instalar e executar uma instância do banco de dados Cassandra, eu recomendo uma máquina com pelo menos 4 GB de RAM, com pelo menos 2 GB disponíveis. Uma máquina com 8 GB de RAM seria ainda melhor. Se você decidir executar as instâncias do Cassandra no Docker, cada contêiner deverá ter no mínimo 1 GB de RAM disponível para executar cada nó do Cassandra.
- **Java 8**— Desde o release do Apache Cassandra V3, o Java Standard Edition 8 precisa estar instalado na máquina, porque o Cassandra é executado na Java Virtual Machine (JVM). As

versões mais antigas do Cassandra (como a V2.2) podem ser executadas com o Java 7. É possível verificar sua versão do Java digitando

```
java -version
```

no shell de prompt do sistema operacional.

- **Python 2.7**— A instalação do Python será necessária se você desejar usar a ferramenta de gerenciamento de nó do Cassandra `nodetool` e o utilitário de shell `cqlsh`. Essas ferramentas são úteis para obter informações e gerenciar a instância do Cassandra e seus bancos de dados. É possível verificar qual versão do Python está instalada digitando

```
python --version.
```

- **Docker CE**— Será opcional se você desejar configurar todos os nós do Cassandra em contêineres em execução na mesma máquina. Eu recomendo usá-lo para criar um ambiente em cluster de teste. Não se preocupe se os contêineres do Docker forem novidade para você. Abaixo, apresentarei os comandos necessários para configurar o cluster do Cassandra. Faça download da versão mais recente do Docker CE para sua plataforma no [website do Docker](#).

## Instalação

É possível escolher instalar o Cassandra manualmente por meio do [website do Cassandra](#) ou automaticamente pelos contêineres do Docker. Se você escolher usar os contêineres do Docker para criar o cluster do Cassandra, será possível pular a sessão "Faça download do pacote".

### Faça download do pacote

Se você estiver usando Linux, poderá encontrar um pacote específico para sua instalação, mas na maioria dos casos, você fará o download de um arquivo `tar.gz` compactado da última versão disponível (V3.11, no momento da composição deste artigo).

1. Depois de fazer o download, descompacte o pacote com o utilitário TAR (ou uma ferramenta semelhante):

```
$ tar -xvf apache-cassandra-3.11.0-bin.tar.gz
```

2. Extraia o conteúdo do arquivo para o local desejado. Após a extração, crie um diretório `apache-cassandra-3.11.0` com todos os binários, os arquivos de configuração, os arquivos de documentação, as bibliotecas e as ferramentas utilitárias do Cassandra, como o seguinte:

```
$ ls
CHANGES.txt  LICENSE.txt  NEWS.txt  NOTICE.txt  bin  conf  doc  interface  javadoc  lib  pylib  tools
```

## Configuração

Esta seção cobre a instalação manual do primeiro nó do Cassandra. Mesmo se você estiver usando o Docker, esta seção poderá ser interessante para entender os principais parâmetros de configuração do Cassandra. No entanto, caso deseje, pule para "[Configure um cluster de teste usando o Docker](#)".

Toda a configuração principal do Cassandra está localizada no arquivo `cassandra.yaml` que se encontra no diretório `conf`.

### Parâmetros de configuração

Edite o arquivo `cassandra.yaml` para alterar os seguintes parâmetros básicos:

- `cluster_name`— Este parâmetro identifica o nome do cluster de três nós do Cassandra. É importante que a configuração de todos os nós tenha o mesmo nome.  
`cluster_name: 'Hospital Test Cluster'`
- `seeds`— O endereço de rede de IP ou a lista de nomes do host dos principais nós do cluster. Para seu cluster de teste, você configurará o endereço IP do primeiro nó.  
`seeds: "127.0.0.1"`
- `listen_address`— O nome do host do nó que será usado pelos clientes e outros nós que serão conectados a esse nó. Em vez de usar o host local (como mostrado aqui), configure o nome do host real usado pela máquina em sua rede.  
`listen_address: localhost`
- `native_transport_port`— O número da porta TCP do nó que será usado pelos clientes para se conectar a este nó. Assegure-se de usar uma porta que não esteja bloqueada por firewalls. O padrão é a 9042.  
`native_transport_port: 9042`

Para definir uma configuração básica de autenticação e autorização para essa instância, é necessário alterar estes parâmetros adicionais opcionais:

- `authenticator`— Permite a autenticação do usuário. Também será necessário alterar esse parâmetro para exigir que os usuários informem o nome de usuário e a senha ao se conectarem com o cluster.  
`authenticator: PasswordAuthenticator`
- `authorizer`— Permite a autorização do usuário e limita suas permissões. Se você alterar esse parâmetro, será necessário aumentar o fator de replicação do keyspace `system_auth` para criar outras cópias dos dados de autorização em outros nós no caso de indisponibilidade do nó.  
`authorizer: CassandraRoleManager`

## Inicie o primeiro nó manualmente

Agora que toda a configuração está definida, é possível executar o script do Cassandra localizado dentro do diretório `bin`, como é mostrado abaixo. A opção `-f` emitirá todos os logs de autoinicialização no primeiro plano. Na primeira vez, é interessante verificar se há erros durante a inicialização do Cassandra.

```
$ bin/cassandra -f
```

Se as seguintes informações de log forem exibidas após alguns segundos da inicialização, isso significará que o nó do Cassandra está ativo, em execução e pronto para receber conexões de clientes.

```
INFO [main] 2017-08-20 18:04:58,329 Server.java:156 - Starting listening for CQL
clients on localhost/127.0.0.1:9042 (unencrypted)...
```

Para confirmar novamente o status do nó, é possível usar o utilitário `nodetool`, localizado no diretório `bin`. Isso pode fornecer informações sobre o cluster e os nós do Cassandra. Para verificar o status do cluster, basta emitir o comando a seguir:



```
$ nodetool status
```

O comando a seguir imprime informações sobre o cluster, incluindo o nome do datacenter no qual o cluster está sendo executado (neste caso, a configuração padrão) e o status de cada membro do nó do cluster:

```
Datacenter: datacenter1
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address Load Tokens Owns (effective) Host ID Rack
UN 127.0.0.1 103.67 KiB 256 100.0% 6aae6c1f-cf06-4874-9507-a43025c312d1 rack1
```

As letras **UN** antes do endereço IP significam que o nó está ativo (U) e normal (N). Após a primeira inicialização, os diretórios de dados e logs serão criados para armazenar todos os dados e logs do keyspace e das tabelas, respectivamente.

Se você realmente deseja usar os contêineres do Docker, repita as etapas anteriores para criar outros nós. Lembre-se de usar o endereço IP do primeiro nó como a configuração de valor inicial para os outros nós. Se você realmente deseja usar os contêineres do Docker, siga as próximas etapas.

## Configure um cluster de teste usando o Docker

Em vez de instalar e configurar o Cassandra em máquinas de servidores físicas diferentes, é possível usar os contêineres do Docker para criar um cluster de três nós em execução na mesma máquina de servidor de teste. Verifique se há RAM suficiente para executar três instâncias do Cassandra. Caso contrário, você poderá tentar reduzir o número de nós para dois.

Se o Docker estiver instalado em sua máquina de teste, será possível usar as imagens oficiais disponíveis no hub do Docker. Para fazer o download, instalar e executar o Cassandra 3.11, insira o seguinte comando do Docker:

```
docker run --name node1 -d cassandra:3.11
```

Esse comando procura na Internet uma imagem chamada `cassandra` com a identificação de versão `3.11` no registro do hub do Docker. Em seguida, ele faz o download da imagem e cria e inicia um contêiner chamado `node1`. Esse contêiner já foi definido em uma configuração padrão do Cassandra, semelhante à descrita acima.

É possível verificar se o novo contêiner está funcionando usando o `docker ps` :

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND NAMES	CREATED	STATUS	PORTS
803135731d1a	cassandra:3.11	"/docker-entrypoint.s"	About a minute ago	Up About a minute	7000-7001/tcp, 7199/tcp, 9042/tcp, 9160/tcp node1

Agora é possível iniciar outras instâncias, informando o local do primeiro nó a cada novo nó. Faça isso mudando o endereço IP do nó de valor inicial do novo nó usando o `CASSANDRA_SEEDS` . Esse

comando muda a configuração de valor inicial automaticamente dentro do arquivo `cassandra.yaml` do novo nó criado dentro do contêiner. Para criar e iniciar o segundo contêiner de nó (`node2`), insira o seguinte:

```
$ docker run --name node2 -d -e CASSANDRA_SEEDS="$(docker inspect --format='{{.NetworkSettings.IPAddress }}' node1)" cassandra:3.11
```

Para determinar quantos nós há dentro do cluster, execute o utilitário `nodetool` dentro do contêiner `node1`:

```
$ docker exec -it node1 nodetool status
```

Esse comando exibe o status de cada nó do cluster configurado até o momento, portanto, o resultado esperado é semelhante ao seguinte:

```
Datacenter: datacenter1
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address Load Tokens Owns (effective) Host ID Rack
UN 172.17.0.3 103.25 KiB 256 100.0% f1bbd4d1-8930-45a3-ba43-4a2416617c7f rack1
UN 172.17.0.2 108.64 KiB 256 100.0% bec1b022-a397-4401-bd42-676c60397fe1 rack
```

If `node2` iniciado com sucesso, será possível criar um terceiro nó (`node3`) da mesma maneira. Quando os três nós estiverem em execução, acesse "[Testes](#)" abaixo. Se, por algum motivo, `node2` falhar ao ser iniciado, continue com o procedimento de resolução de problemas aqui.

## Cluster de Resolução de problemas do Docker

Se um dos nós não for iniciado corretamente, verifique os logs do Cassandra:

```
$ docker logs node2
```

Se aparecer uma mensagem de erro como "Unable to gossip with any seeds", será necessário incluir um parâmetro seguindo as instruções abaixo. Caso contrário, acesse "[Testes](#)."

1. Para incluir o parâmetro necessário, primeiro capture o endereço IP do `node1` usando o `inspect` do Docker:  

```
$ docker inspect --format='{{.NetworkSettings.IPAddress }}' node1
```
2. Vamos considerar que o endereço IP retornado seja 172.17.0.2. Execute os comandos a seguir para parar e remover o contêiner `node1` e, em seguida, recrie-o com os parâmetros de endereço de transmissão e porta gossip expostos:  

```
$ docker stop node1
$ docker rm node1
$ docker run --name node1 -d -e CASSANDRA_BROADCAST_ADDRESS=172.17.0.2 -p 7000:7000 cassandra:3.11
```
3. Em seguida, crie o `node2` com um endereço IP de transmissão de 172.17.0.3, reutilizando o endereço `node1` como o nó de valor inicial:  

```
$ docker run --name node2 -d -e CASSANDRA_BROADCAST_ADDRESS=172.17.0.3 -p 7001:7000 -e CASSANDRA_SEEDS=172.17.0.2 cassandra:3.11
```

Esta configuração permitirá que dois nós transmitam as informações do protocolo gossip configuradas na porta 7000 entre si por meio das portas do contêiner 7000 e 7001.

4. Em seguida, use `docker ps` para verificar se os dois processos do Docker estão em execução. Depois, use o utilitário `nodetool` novamente para confirmar o status do cluster:

```
$ docker exec -it node1 nodetool status
Datacenter: datacenter1
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address Load Tokens Owns (effective) Host ID Rack
UN 172.17.0.3 108.29 KiB 256 100.0% fd135375-711a-471a-b4e5-409199bbaaa5 rack1
UN 172.17.0.2 108.66 KiB 256 100.0%
5db97fc3-70e9-48e5-b63b-0be67e35daea rack1
```

## Variáveis de ambiente do contêiner

Para mudar a configuração de nome do cluster padrão do Cassandra dentro do contêiner do Docker, é possível usar variáveis de ambiente do contêiner.

Ao executar uma imagem do Docker do Cassandra, é possível definir configurações específicas do Cassandra passando uma ou mais variáveis de ambiente na linha de comandos do Docker `run` usando `-e`. Essa opção será usada pela imagem para mudar os parâmetros do Cassandra dentro do contêiner. Para obter mais informações, veja o [Documentação da imagem do Cassandra](#) no hub do Docker.

## Testes

A primeira etapa para testar a configuração do cluster é conectar-se a ele usando o utilitário de shell CQL (`cqlsh`). Esse é um script de linha de comandos Python que cria um cliente que consegue se conectar a qualquer host de cluster. Basta emitir o `cqlsh`. Por padrão, o script tentará se conectar a uma instância em execução no host local. É possível mudar o host passando `host`. Consulte a ajuda do `cqlsh` para obter detalhes (`cqlsh --help`).

## Ferramenta shell CQL

Se você estiver usando o Docker, será possível executar `cqlsh` de dentro do contêiner:

```
$ docker exec -it node1 cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.0 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

Este shell permite emitir comandos CQL semelhantes a SQL para criar e definir o keyspace (esquema), as tabelas e manipular dados. É possível encontrar mais informações na [documentação do Cassandra](#).

## Crie o keyspace de teste

Vamos criar o primeiro keyspace para armazenar todas as informações de exames de paciente:

1. Emita o comando CQL CREATE KEYSPACE para criar o esquema do paciente.

```
cqlsh> CREATE KEYSPACE patient WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
```

2. Agora, é possível acessar o keyspace criado e criar a primeira tabela para armazenar dados de exame. O comando CQL para criar uma tabela é semelhante a um comando SQL DDL:

```
CREATE TABLE patient.exam (
  patient_id int,
  id int,
  date timeuuid,
  details text,
  PRIMARY KEY (patient_id, id));
```

Os comandos acima criam uma tabela com uma chave primária composta pelo ID do paciente e o ID do próprio exame.

## Insira dados


1. Agora que a estrutura do keyspace está criada, insira alguns dados de amostra de três pacientes:

```
INSERT INTO exam (patient_id,id,date,details) values (1,1,now(),'first exam patient 1');
INSERT INTO exam (patient_id,id,date,details) values (1,2,now(),'second exam patient 1');
INSERT INTO exam (patient_id,id,date,details) values (2,1,now(),'first exam patient 2');
INSERT INTO exam (patient_id,id,date,details) values (3,1,now(),'first exam patient 3');
```

2. Em seguida, execute uma consulta de teste para capturar todos os exames do paciente 1:

```
cqlsh:patient> select * from exam where patient_id=1;
```

**Figura 6. Executando uma consulta para o paciente 1**



```
cqlsh:patient> select * from exam where patient_id=1 ;
```

patient_id	id	date	details
1	1	c9066760-8c4d-11e7-a246-6d2c86545d91	first exam patient 1
1	2	dfffe770-8c4d-11e7-a246-6d2c86545d91	second exam patient 1

```
(2 rows)
cqlsh:patient> _
```

## Teste o cluster

Este é o momento de testar a disponibilidade do cluster, a consistência e a tolerância da partição. Se houver uma configuração de fator de replicação igual a três para o keyspace do paciente, haverá uma cópia em cada um dos três nós de qualquer dado gravado na tabela de exame.

## Teste de replicação de nó

É possível inserir os dados no `node1` e consultar no `node3`, porque os dados gravados no `node1` serão replicados automaticamente para o `node2` e o `node3`, confirmando as três réplicas dos dados. Para testar a inserção de um paciente no `node1` para determinar se suas informações estão disponíveis no `node3`, insira o seguinte:

```
$ docker exec -it node1 cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.0 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> INSERT INTO patient.exam (patient_id,id,date,details) values (9,1,now(),'first exam patient 9');
cqlsh> quit;
$ docker exec -it node3 cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.0 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> select * from patient.exam where patient_id=9;
  patient_id | id | date                                     | details
-----+-----+-----+-----
          9 |  1 | 9cf570b0-8e9d-11e7-a592-6d2c86545d91 | first exam patient 9
(1 rows)
```

## Teste de falha de nó

Agora, pare o `node2` e o `node3` e, em seguida, insira dados de paciente no `node1`. início `node2` e `node3` novamente para ver se os dados inseridos no `node1` foram replicados para o `node2` e o `node3` indisponíveis.

```
$ docker stop node2
$ docker stop node3
$ docker exec -it node1 cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.0 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> INSERT INTO patient.exam (patient_id,id,date,details) values (10,1,now(),'first exam patient 10');
cqlsh> quit;
$ docker start node2
$ docker start node3
$ docker exec -it node3 cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.0 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> select * from patient.exam where patient_id=10;
  patient_id | id | date                                     | details
-----+-----+-----+-----
         10 |  1 | 76439070-8f04-11e7-a592-6d2c86545d91 | first exam patient 10
(1 rows)
```

## Teste de consistência de nó

Se você desejar uma consistência de leitura forte, o nível de consistência deverá ser configurado para `QUORUM`, para que todas as consultas verifiquem os dados em pelo menos dois nós disponíveis.

```
$ docker stop node1
node1
$ docker stop node2
node2
$ docker exec -it node3 cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.0 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> select * from patient.exam where patient_id=10;
  patient_id | id | date | details
-----+-----+-----+-----
      10 | 1 | 76439070-8f04-11e7-a592-6d2c86545d91 | first exam patient 10
(1 rows)
cqlsh> consistency quorum
Consistency level set to QUORUM.
cqlsh> select * from patient.exam where patient_id=10;
NoHostAvailable:
```

Nesse caso, a maioria dos nós deverá estar funcionando (dois) ou a consulta falhará. Existe uma relação. Se você desejar uma disponibilidade maior com uma consistência mais forte, será necessário haver mais nós no cluster para permitir a continuação da operação em caso de falha de nós. Se você configurar a consistência como ONE, a consulta será bem-sucedida porque há uma cópia local disponível no `node3` no qual você está conectado.

```
cqlsh> consistency one
Consistency level set to ONE.
cqlsh> select * from patient.exam where patient_id=10;
  patient_id | id | date | details
-----+-----+-----+-----
      10 | 1 | 76439070-8f04-11e7-a592-6d2c86545d91 | first exam patient 10
(1 rows)
cqlsh> quit;
```

Se você iniciar o `node1` novamente, você terá a maioria dos nós, assim, uma consistência de QUORUM poderá ser cumprida e a consulta não falhará mais.

```
$ docker start node1
node1
$ docker exec -it node3 cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.0 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> consistency quorum
Consistency level set to QUORUM.
cqlsh> select * from patient.exam where patient_id=10;
  patient_id | id | date | details
-----+-----+-----+-----
      10 | 1 | 76439070-8f04-11e7-a592-6d2c86545d91 | first exam patient 10
(1 rows)
```

Se você verificar o status dos nós, verá que há dois nós ativos (UN) e um nó inativo (DN).

```
$ docker exec -it node3 nodetool status
Datacenter: datacenter1
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address      Load        Tokens      Owns (effective)  Host ID                               Rack
DN  172.17.0.3    306.19 KiB   256         100.0%            fd135375-711a-471a-b4e5-409199bbaaa5  rack1
UN  172.17.0.2    365.72 KiB   256         100.0%            5db97fc3-70e9-48e5-b63b-0be67e35daea  rack1
UN  172.17.0.4    285.42 KiB   256         100.0%            4deb44f8-9253-4bff-b74b-239085e3a912  rack1
```

Também é possível explorar outros cenários de teste, como teste de latência ou teste de balanceamento de carga, com mais nós e confirmar as características descritas anteriormente sobre o cluster distribuído do Cassandra.

## Conclusão

Neste tutorial, meu objetivo foi orientá-lo sobre a instalação e configuração básicas de um cluster do Cassandra e explicar suas características mais importantes. Além disso, eu procurei passar alguns exercícios práticos para ajudá-lo a conhecer melhor esse tipo de banco de dados NoSQL. Talvez agora você esteja preparado para começar seu projeto e usar o Cassandra, caso seus requisitos correspondam aos recursos do Cassandra.

Há outros detalhes sobre técnicas de modelagem para diferentes conjuntos de dados e tarefas de administração e ajuste preciso para melhorar o desempenho que eu não abordei. Além disso, há muitos outros aspectos sobre drivers de clientes relacionados ao desenvolvimento que não fizeram parte deste tutorial. Esses assuntos poderão ser abordados em futuros tutoriais. Enquanto isso, para saber mais sobre o Cassandra, há duas fontes de informações que eu recomendo: o website oficial do Apache Cassandra e o website da documentação do DataStax. Consulte "Tópicos relacionados" abaixo.

Temas relacionados: [Faça o download do Apache Cassandra](#) [Introdução ao Apache Cassandra](#) [Documentação do DataStax](#) [Docker](#) [Documentação da imagem do Cassandra](#) [Teorema de CAP](#) [Prós e contras do banco de dados Apache Cassandra](#) [Apache Cassandra e Kubernetes: bancos de dados seguros, escaláveis e gerenciáveis](#) [Implemente um banco de dados Apache Cassandra escalável no Kubernetes](#)

## Sobre o autor

**Bruno Reded Tinoco**



<http://twitter.com/brunocrt>

© Copyright IBM Corporation 2018. Todos os direitos reservados.

([www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml))

[Marcas Registradas](#)

([www.ibm.com/developerworks/br/ibm/trademarks/](http://www.ibm.com/developerworks/br/ibm/trademarks/))