**Tutorial: Configuração do NetBeans, Docker e MySQL com Replicação**

Este tutorial orienta a configuração do **NetBeans** para trabalhar com **Docker**, a criação de containers **MySQL Master-Slave** com replicação e a implementação de um **sistema Java para cadastro de pessoas**.

**1. Configurar o NetBeans para Trabalhar com Docker**

**1.1 Habilitando o suporte ao Docker no NetBeans**

1. Abra o **NetBeans**.
2. Vá em **Tools → Plugins**.
3. Na aba **Available Plugins**, procure por **Docker** e instale-o.
4. Reinicie o NetBeans após a instalação.
5. Vá em **Tools → Docker** e adicione uma nova instância do Docker.

**1.2 Configurar o Docker no Windows para aceitar conexões TCP**

1. Abra o **Docker Desktop** e vá para **Settings**.
2. Vá até a aba **General** e ative a opção **"Expose daemon on tcp://localhost:2375 without TLS"**.
3. Clique em **Apply & Restart**.

**1.3 Configurar a Conexão no NetBeans**

1. Vá em **Tools → Docker**.
2. Clique em **Add Docker Instance**.
3. Escolha **TCP Connection**.
4. No campo **URL**, insira:

http://127.0.0.1:2375

1. Deixe **Certificates Path** vazio.
2. Clique em **Test Connection** e, se tudo estiver correto, clique em **Finish**.

Agora o NetBeans está configurado para rodar containers Docker.

**2. Criar Containers MySQL com Replicação**

**2.1 Criar o arquivo docker-compose.yml**

Crie um diretório para o projeto e dentro dele um arquivo chamado docker-compose.yml:

version: '3.8'

services:

mysql-master:

image: mysql:8.0

container\_name: mysql-master

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: root

MYSQL\_DATABASE: cadastro

MYSQL\_USER: usuario

MYSQL\_PASSWORD: senha

ports:

- "3307:3306"

command: --default-authentication-plugin=mysql\_native\_password --server-id=1 --log-bin=mysql-bin

volumes:

- mysql\_master\_data:/var/lib/mysql

- ./master.cnf:/etc/mysql/conf.d/master.cnf

mysql-slave:

image: mysql:8.0

container\_name: mysql-slave

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: root

MYSQL\_DATABASE: cadastro

MYSQL\_USER: usuario

MYSQL\_PASSWORD: senha

ports:

- "3308:3306"

command: --default-authentication-plugin=mysql\_native\_password --server-id=2 --relay-log=mysql-relay-bin

volumes:

- mysql\_slave\_data:/var/lib/mysql

- ./slave.cnf:/etc/mysql/conf.d/slave.cnf

volumes:

mysql\_master\_data:

mysql\_slave\_data:

**2.2 Criar os arquivos de configuração para replicação**

**2.2.1 Arquivo master.cnf**

[mysqld]

server-id=1

log-bin=mysql-bin

binlog-do-db=cadastro

**2.2.2 Arquivo slave.cnf**

[mysqld]

server-id=2

relay-log=mysql-relay-bin

log-bin=mysql-bin

binlog-do-db=cadastro

read\_only=1

**2.3 Subir os containers**

Abra o terminal, entre no diretório do projeto e execute o seguinte comando:

docker-compose up -d

**3. O que é Replicação MySQL?**

A replicação do **MySQL** é um processo em que **um servidor (Master)** copia seus dados para **outro servidor (Slave)**. O Slave recebe todas as alterações do Master e as aplica em tempo real.

Existem três tipos principais de replicação:

1. **Replicação baseada em logs binários** (*Statement-Based Replication - SBR* e *Row-Based Replication - RBR*).
2. **Replicação Multi-Source** (um Slave recebe dados de vários Masters).
3. **Replicação em Grupo** (*Group Replication* para alta disponibilidade).

No nosso caso, estamos usando a **replicação baseada em logs binários**, onde o Slave lê os arquivos de log do Master e aplica as mudanças.

**3.1. Configuração Passo a Passo**

Aqui está uma explicação detalhada das configurações:

**Passo 1: Configurar o Master (master.cnf)**

O **arquivo master.cnf** define configurações essenciais para que o **MySQL Master** registre todas as mudanças feitas no banco de dados em um **log binário (log-bin=mysql-bin)**.

[mysqld]

server-id=1 # ID único do servidor Master

log-bin=mysql-bin # Ativa o log binário para replicação

binlog-do-db=cadastro # Define o banco que será replicado

* O server-id=1 define que este servidor é o **Master**.
* O log-bin=mysql-bin ativa a **gravação de logs** das operações do banco de dados.
* O binlog-do-db=cadastro indica que **somente o banco "cadastro" será replicado**.

**Passo 2: Configurar o Slave (slave.cnf)**

O **arquivo slave.cnf** configura o servidor Slave para atuar como **somente leitura** e aplicar as mudanças recebidas do Master.

[mysqld]

server-id=2 # ID único do Slave

relay-log=mysql-relay-bin # Define um log intermediário (relay log)

log-bin=mysql-bin # Também grava logs binários (opcional)

binlog-do-db=cadastro # Replica apenas o banco "cadastro"

read\_only=1 # Impede escrita direta no Slave

* server-id=2: Define que este servidor será o **Slave**.
* relay-log=mysql-relay-bin: Armazena logs intermediários antes de aplicá-los.
* read\_only=1: Garante que **o Slave não aceite comandos de escrita** (exceto do Master).

**Passo 3: Criar e Configurar a Replicação no Master**

Após subir os containers, entre no **Master**:

docker exec -it mysql-master mysql -u root -p

Crie um usuário **exclusivo para a replicação**:

CREATE USER 'repl'@'%' IDENTIFIED WITH mysql\_native\_password BY 'replpass';

GRANT REPLICATION SLAVE ON \*.\* TO 'repl'@'%';

Agora, veja o estado atual do Master:

SHOW MASTER STATUS;

Isso retornará algo como:

+---------------+----------+

| File | Position |

+---------------+----------+

| mysql-bin.000003 | 660 |

+---------------+----------+

🚨 **Anote os valores de File e Position**, pois serão usados no Slave!

**Passo 4: Configurar o Slave**

Agora, entre no **Slave**:

docker exec -it mysql-slave mysql -u root -p

Conecte o Slave ao Master, substituindo <FILE> e <POSITION> pelos valores obtidos no **SHOW MASTER STATUS**:

CHANGE MASTER TO

MASTER\_HOST='mysql-master',

MASTER\_USER='repl',

MASTER\_PASSWORD='replpass',

MASTER\_LOG\_FILE='**mysql-bin.000005**',

MASTER\_LOG\_POS=**157**;

Agora, **inicie a replicação**:

START SLAVE;

E **verifique se está funcionando**:

SHOW SLAVE STATUS\G;

Se tudo estiver correto, **Slave\_IO\_Running e Slave\_SQL\_Running** devem estar como **YES**:

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

A linha **Seconds\_Behind\_Master: 0** indica que o Slave está sincronizado com o Master.

**3.2 Como Testar a Replicação**

Agora que o Slave está configurado, vamos testar:

**Criando uma tabela e inserindo dados no Master**

docker exec -it mysql-master mysql -u root -p

USE cadastro;

CREATE TABLE pessoa (

id INT AUTO\_INCREMENT PRIMARY KEY,

nome VARCHAR(100) NOT NULL

);

INSERT INTO pessoa (nome) VALUES ('Joao');

**Verificando no Slave**

Agora, entre no **Slave** e execute:

docker exec -it mysql-slave mysql -u root -p

SELECT \* FROM cadastro.pessoa;

Se o registro apareceu, a replicação **está funcionando!** 🎉

**4. Testar a Conexão com o Banco MySQL no NetBeans**

**Passo 1: Abrir a Ferramenta de Bancos de Dados**

1. Vá para **Window → Services**.
2. Expanda **Databases**.
3. Clique com o botão direito em **Drivers → MySQL (Connector/J)** e selecione **Connect Using...**.

**Passo 2: Configurar a Conexão**

**Para conectar ao mysql-master:**

jdbc:mysql://localhost:3307/cadastro

**Para conectar ao mysql-slave:**

jdbc:mysql://localhost:3308/cadastro

* **User Name:** usuario
* **Password:** senha
* Clique em **Test Connection**.

Se a conexão for bem-sucedida, o NetBeans já pode acessar o MySQL no Docker! 🎉

**5. Adicionar a Dependência do MySQL no pom.xml**

Caso esteja usando **Maven**, adicione a seguinte dependência no arquivo pom.xml:

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.33</version>

</dependency>

Se não estiver usando Maven, baixe o **MySQL Connector/J** e adicione ao classpath do projeto.

**6. Criar um Sistema Java para Cadastro**

**Criar a classe DatabaseConnection.java**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DatabaseConnection {

private static final String URL = "jdbc:mysql://localhost:3307/cadastro";

private static final String USER = "usuario";

private static final String PASSWORD = "senha";

public static Connection getConnection() {

try {

Class.forName("com.mysql.cj.jdbc.Driver");

return DriverManager.getConnection(URL, USER, PASSWORD);

} catch (Exception e) {

throw new RuntimeException("Erro ao conectar ao banco de dados", e);

}

}

}

**Criar a classe Pessoa.java**

public class Pessoa {

private String nome;

public Pessoa(String nome) { this.nome = nome; }

public String getNome() { return nome; }

}

**Criar a classe PessoaDAO.java**

import java.sql.Connection;

import java.sql.PreparedStatement;

public class PessoaDAO {

public void cadastrarPessoa(Pessoa pessoa) {

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement("INSERT INTO pessoa (nome) VALUES (?)")) {

stmt.setString(1, pessoa.getNome());

stmt.executeUpdate();

}

}

}

**Criar o Main.java**

public class Main {

public static void main(String[] args) {

new PessoaDAO().cadastrarPessoa(new Pessoa("Maria"));

}

}

Agora execute o programa e verifique a replicação no banco slave! 🚀

Estamos utilizando **dois mecanismos de redundância** no nosso sistema:

1. **Redundância no Banco de Dados (Replicação MySQL Master-Slave)**
   * A replicação **garante que os dados do banco master sejam automaticamente copiados para o banco slave**.
   * Se o servidor master falhar, o slave pode ser promovido para assumir o papel de master, garantindo disponibilidade.
   * A replicação também melhora o desempenho, pois consultas de leitura podem ser distribuídas para o slave.
2. **Redundância no Docker (Containers Isolados e Recriação Automática)**
   * Utilizamos **containers Docker para encapsular os servidores MySQL** (master e slave), garantindo que o ambiente esteja isolado e reproduzível.
   * Se um container parar ou falhar, o Docker pode **recriá-lo automaticamente** (restart: always no docker-compose.yml).
   * Com **volumes persistentes**, os dados não são perdidos quando o container é reiniciado.

Esse modelo fornece **alta disponibilidade** e **resiliência** tanto no nível da aplicação (dados replicados) quanto no nível da infraestrutura (containers gerenciáveis). 🚀