Projeto Laboratorial

Beatriz Santos - 50473, Inês Santos - 49436, Manoela Azevedo - 50034 e Rodrigo Paiva - 49442

Abstract—Este relatório documenta a implementação de um sistema distribuído utilizando Hadoop para processamento de dados em larga escala, com integração via APIs REST desenvolvidas em Node.js. O objetivo principal é demonstrar o uso de funções de mapeamento e redução em um fluxo de dados reais.

Index Terms—Hadoop, Supabase, BigData, API, REST, Computação na Nuvem, Máquinas Virtuais.

1 Introdução

O presente projeto propõe o desenvolvimento de uma aplicação em ambiente na nuvem para a gestão e análise de um catálogo de livros da coleção digital do Projeto Gutenberg.

O objetivo principal é fornecer um sistema para organizar, catalogar e analisar informações sobre os livros em grande escala de dados. A aplicação utiliza tecnologias como a *framework* Hadoop para o processamento em larga escala com a aplicação da função MapReduce. Utiliza também a plataforma de base de dados em nuvem Supabase para armazenamento e gestão de dados.

O sistema permite, também, a inserção, atualização e consulta de livros. A abordagem em ambiente nuvem garante escalabilidade e flexibilidade no processamento e armazenamento de grandes volumes de dados, permitindo que o catálogo de livros seja constantemente atualizado e analisado de forma eficaz.

1.1 Divisão do trabalho

A divisão de trabalho neste projeto foi realizada de maneira colaborativa, com todos os membros da equipa contribuindo ativamente nas etapas do desenvolvimento. Promovendo assim um ambiente de trabalho conjunto e eficiente.

Cada integrante participou da concepção do sistema, incluindo a pesquisa e escolha das tecnologias utilizadas, como a Supabase. Tendo sido o integrante do grupo, Rodrigo Calado, a realizar a configuração da base de dados e a conecção com a mesma. Já as integrantes, Beatriz Santos e Manoela Azevedo implementaram as APIs REST, garantindo assim que a funcionalidade dos endpoints atendessem aos objetivos propostos. E por fim, a integrante, Inês Santos, focou-se mais no apoio nas tarefas dos outros integrantes e a realização do relatório do projeto.

2 Estrutura/Desenvolvimento

2.1 Arquitetura

O nosso trabalho seguiu uma estrutura simples com diferentes componentes interconectados e em diferentes ambientes.

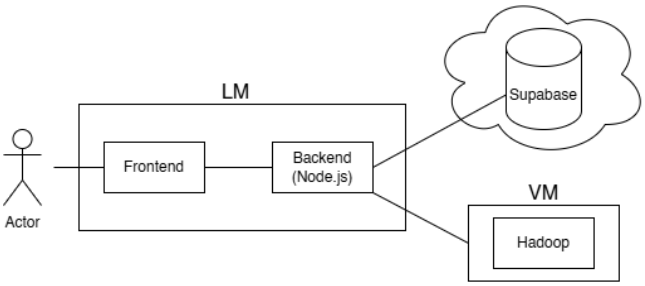

Fig. 1. Arquitetura Simples do Projeto

2.1.1 *Máquina Local do Cliente*

- O sistema apresenta um **frontend** que exibe a interface ao utilizador, permitindo interações.
- O sistema também possui um **backend**, que é a camada lógica responsável por processar as solicitações do utilizador e gerenciar a comunicação com outros componentes.

2.1.2 *Máquina Virtual*

- Esta máquina executa o **Hadoop**, uma framework de processamento de grandes volumes de dados, conectando-se ao backend para enviar e receber dados.

2.1.3 *Infraestrutura na Nuvem*

- A **Supabase**, um banco de dados armazenado na nuvem, está conectada ao backend, permitindo o envio e recebimento de informações.

2.2 Algoritmos

Previamente, serão apresentadas as explicações sobre as funções de mapeamento e redução.

Função de Mapeamento

Lógica: A Função de Mapeamento é responsável por processar cada linha de entrada e extrair o ano de emissão do livro. Essa informação é emitida no formato chave-valor, permitindo que os dados sejam agrupados por ano na próxima etapa.

```python
#!/usr/bin/env python3
import sys

for line in sys.stdin:
  try:
    parts = line.strip().split(',')

    if parts[0] != "ID":
      year = parts[1][:4]

      if year.isdigit() and len(year) == 4:
        print(f"{year}\t1")
  except IndexError:
    continue
```

Funcionamento: Esta função recolhe o ano de publicação dos registos e expressa pares chave-valor no formato `{year}\t1` omitindo cabeçalhos e entradas inválidas.

Função de Redução

Lógica: A Função de Redução tem o intuito de contar o número de livros publicados em cada ano e agrupar e consolidar os pares chave-valor gerados pela função de mapeamento.

```python
#!/usr/bin/env python3
import sys
from collections import defaultdict

counts = defaultdict(int)

for line in sys.stdin:
  try:
    year, count = line.strip().split('\t')
    counts[year] += int(count)
  except ValueError:
    continue

for year, total in sorted(counts.items()):
  print(f"{year}\t{total}")
```

Funcionamento: A função de redução agrupa e soma os pares chave-valor que foram recebidos no formato `{year}\t1`, analisando o íntegro de eventualidades para cada ano. Depois de processar os dados, expressa o resultado ordenado no formato `{year}\t{total}`.

2.3 Métodos

Na secção do Backend, utilizamos Node.js e Express.js para estruturar a aplicação. Desenvolvemos uma API REST que facilita a comunicação entre o cliente e o servidor. Cada operação está vinculada a um método HTTP específico: GET, POST e DELETE, para respetivamente, aceder, criar e eliminar livros da base de dados do Supabase.

Método POST

Lógica: A Função do POST, é responsável por receber dados de um livro (como data, título, idioma e autor) e atribuí-los a um objeto chamado "data". Dependendo do endpoint, esse objeto é criado como um novo livro, ou atualiza os valores de um livro já existente na base de dados.

Função POST para criar um livro:

```javascript
api.post('/insertBook/', async(req, res) => {
    const date = (req.query && req.query.date ?
        req.query.date : null);
    const title = (req.query && req.query.title ?
         req.query.title : null);
    const language = (req.query && req.query.
        language ? req.query.language : null);
    const authors = (req.query && req.query.
        authors ? req.query.authors : null);
    const data = {date: date, title: title,
        language: language, authors: authors}
    const bookInfo = await books.insertBook(data)
        ;
    return res.status(200).json(JSON.parse(JSON.
        stringify(bookInfo)));
});
```

Funcionamento: A função POST, definida em `api.post('/insertBook/')`, é responsável por inserir um novo livro na base de dados. Quando é chamada, recebe dados do livro e caso um dos dados não seja recebido, é definido como null. Os dados são guardados ao objeto "data" que é enviado para a função insertBook(), é responsável por inseri-lo na base de dados.

Função POST para atualizar um livro:

```javascript
api.post('/updateBook/', async(req, res) => {
    const id = (req.query && req.query.id ? req.
        query.id : null);
    const date = (req.query && req.query.date ?
        req.query.date : null);
    const title = (req.query && req.query.title ?
         req.query.title : null);
    const language = (req.query && req.query.
        language ? req.query.language : null);
    const authors = (req.query && req.query.
        authors ? req.query.authors : null);
    if (!id || id === null) { return res.status
        (400).json({message: "No ID Defined." })
        }
    const data = {date: date, title: title,
        language: language, authors: authors}
    const bookInfo = await books.updateBookByID(
        id, data);
    return res.status(200).json(JSON.parse(JSON.
        stringify(bookInfo)));
});
```

Funcionamento: A função POST para atualizar um livro, definida em `api.post('/updateBook/')`, é utilizada para modificar as informações de um livro existente na base de dados. Quando é chamada, verifica se o ID do livro foi fornecido na query string. Caso o ID não seja fornecido, a função retorna uma mensagem de erro com o código de status 400, indicando que o ID é obrigatório.

Após a validação do ID, os dados do livro são extraídos (caso um não seja fornecido, é definido como null) e organizados no objeto que juntamente com o ID, são enviados à função updateBookById(), que é responsável por atualizar o livro correspondente ao ID na base de dados. Se a atualização for bem-sucedida, as informações do livro já atualizado são retornadas ao cliente no formato JSON.

Método GET

Lógica: A função do GET é responsável por buscar dados dos livros armazenados na base de dados. Dependendo do endpoint, ela pode devolver a lista completa de livros ou os detalhes de um livro específico por um ID.

Função GET para ver todos livros:

```
api.get('/showBooks/', async(req, res) => {
   const bookList = await books.mostrarLivros();
   return res.status(200).json(bookList);
});
```

Funcionamento: A função GET, definida em `api.get('/showBooks/')` é responsável por devolver todos os livros armazenados na base de dados.

Função GET para ver um livro:

```
api.get('/showBook/', async(req, res) => {
   const id = (req.query && req.query.id ? req.
        query.id : null);
   if (!id || id === null) { return res.status
        (400).json({message: "No ID Defined." })
        }
   const bookInfo = await books.showBookByID(id)
        ;
   return res.status(200).json(JSON.parse(JSON.
        stringify(bookInfo)));
});
```

Funcionamento: A função GET, definida em `api.get('/showBook/')` é utilizada para obter os detalhes de um livro específico de acordo com o ID fornecido. Se o ID for válido, a funcção chama o `showBookByID()` que consulta a base de dados e retorna o livro correspondente.

Método DELETE

Lógica: A Função do DELETE é responsável por remover um livro da base de dados de acordo com o ID fornecido. O ID é essencial para identificar o livro a ser eliminado, e caso não seja enviado, a operação não é realizada.

Função DELETE para alterar um livro:

```
api.delete('/deleteBook/', async(req,res) => {
   const id = (req.query && req.query.id ? req.
        query.id : null);
   if (!id || id === null) { return res.status
        (400).json({message: "No ID Defined." })
        }
   const bookInfo = await books.deleteBookByID(
        id);
   return res.status(200).json(JSON.parse(JSON.
        stringify(bookInfo)));
});
```

Funcionamento: A função DELETE, definida em `api.delete('/deleteBook/')` é utilizada para remover um livro da base de dados. Quando é chamada, verifica se o ID do livro foi fornecido na query string. Este então, é enviado para a função `deleteBookByID()`, que realiza a operação de exclusão na base de dados.

2.4 Protocolos

O protocolo HTTP é utilizado para comunicação entre o cliente e o servidor no sistema REST. Permite a troca de mensagens utilizando métodos fundamentais para a interação com os serviços web.

TABLE 1
Endpoints Implementados

Endpoint	Método	Descrição
`/insertAllBooks/`	POST	Adiciona todos os livros à base de dados.
`/insertBook/`	POST	Adiciona um novo livro. Requer campos como `id`, `year`, `title`, `language`, `authors`.
`/updateBook/`	POST	Atualiza um livro. Requer campos como `id` e os que quer atualizar.
`/deleteBook/`	DELETE	Remove um livro existente com base no `id`.
`/showBooks/`	GET	Lista todos os livros disponíveis na Base de Dados.
`/showBook/`	GET	Apresenta os dados do livro existente na Base de Dados com base no `id`.

Método CURL

Lógica: O middleware, implementado com comandos CURL, faz uso do protocolo HTTP para enviar dados ao SupaBase via as funções de nuvem. Por exemplo, o comando abaixo envia uma nova entrada para o catálogo:

Funções CURL - GET:

```
ShowBooks
- curl -X GET "http://localhost:5000/api/
     showBooks" -H "Content-Type: application/
     json"

ShowBook (ID)
- curl -X GET "http://localhost:5000/api/
     showBook/?id=1" -H "Content-Type:
     application/json"
```

Funções CURL - POST:

```
UpdateBook (ID, DATE, TITLE, LANGUAGE, AUTHORS):
- curl -X POST "http://localhost:5000/api/
     updateBook/?id=1&date=12-12-2024&title=
     Titulo%20Bonito&language=en&authors=Eu" -H
     "Content-Type: application/json"

InsertBook (DATE, TITLE, LANGUAGE, AUTHORS):
- curl -X POST "http://localhost:5000/api/
     insertBook/?date=12-12-2024&title=Titulo%20
     Bonito&language=en&authors=Eu" -H "Content-
     Type: application/json"
```

Funções CURL - DELETE:

```
DeleteBook (ID)
- curl -X DELETE "http://localhost:5000/api/
     deleteBook/?id=1" -H "Content-Type:
     application/json"
```

Funcionamento: Este domínio em CURL é utilizado para encaminhar uma requisição HTTP do tipo DELETE a uma API. Este remove um livro específico constatado pelo parâmetro `id=1` da base de dados do servidor estabelecido em `http://localhost:5000/api/deleteBook/`

e apresentando o cabeçalho `Content-Type: application/json`.

2.5 Funções

Função UpdateFileFromInfo():

Funcionamento: A função UpdateFileFromInfo() busca dados da tabela books no banco de dados Supabase, organiza-os em formato CSV e salva em um ficheiro chamado catalog.csv, substituindo o anterior se existir. Em seguida, chama a função InsertCatalog e processHDFSFile. Se não houver dados ou ocorrer um erro, mensagens são exibidas no console.

```
async function UpdateFileFromInfo() {
 try {
    const { data, error } = await supabase.from(
        'books').select().order('ID', {
        ascending: true});

    if (error) {
       throw new Error(`> Error Trying to Fetch
            Data: ${error.message}`);
    }

    if (data && data.length >0) {
       const headers = Object.keys(data[0]);

       const csvData = [headers.join(','), ...
           data.map(row => headers.map(header
           => row[header]).join(','))].join('\
           n');

       fs.writeFileSync('./catalog.csv', csvData
           , 'utf8');

       console.log(`> All Data Exported
           Successfully.`);

       await InsertCatalog();
       console.log('> Catalog Inserted
           Successfully and Now Processing
           Output File.')
       await hadoopCommands();
       await processHDFSFile();
    } else {
       console.log('> Any Data Recieved From
           the Table.');
    }
 } catch (error) {
    console.error(`> An Error Ocurred: ${error.
        message}`);
 }
}
```

Função processHDFSFile():

Funcionamento: A função processHDFSFile() lê um ficheiro de saída chamado output_hadoop.txt e processa seus dados linha por linha. Cada linha é dividida em ano (year) e quantidade (amount). Para cada linha válida, ela remove entradas correspondentes no banco de dados Supabase e, em seguida, adiciona ou atualiza os dados processados.

Ao final, a função exibe uma mensagem indicando a conclusão bem-sucedida ou relata qualquer erro encontrado durante o processo.

```
async function processHDFSFile() {
 try {
    const { data, error } = await supabase.from("
        books_info").select();
    for (const dataInfo of data) {
     const deleteResponse = await supabase.from("
         books_info").delete().eq('YEAR',
         dataInfo.YEAR);
     if (deleteResponse.error) {
      console.error(`> Error When Trying to
          Delete Year ${year}: ${deleteResponse
          .error.message}`);
     }
    }
   } catch(error) {
    console.error(`> Error Processing Database: $
        {error.message}`);
   }

   try {
    const fileData = fs.readFileSync('./output_
        hadoop.txt', 'utf8');

    const lines = fileData.trim().split('\n');
    const data = [];

    for (const line of lines) {
       const [year, amount] = line.split('\t');

       if (!year || !amount) {
          console.error(`> Error Processing Line:
              ${line}`);
          continue;
       }

       console.log(`> Processing Line: YEAR=${
           year}, AMOUNT=${amount}`);
       data.push({ YEAR: parseInt(year), AMOUNT:
           parseInt(amount) });
    }

    for (const row of data) {
       const { error } = await supabase.from("
           books_info").upsert(row);
       if (error) {
          console.error(`> Error Trying to Insert
              Line (${row.YEAR}, ${row.AMOUNT})
             : ${error.message}`);
       }
    }

    console.log('> Process Finished Successfully.
        ');
   } catch (error) {
     console.error(`> Error Processing Database:
         ${error.message}`);
   }
}
```

Função hadoopCommands():

Funcionamento: A função hadoopCommands() executa uma série de comandos Hadoop através do SSH, organizados para processar e guardar dados. Primeiro, ela remove antigos ficheiros no HDFS, carrega um novo ficheiro CSV para o diretório de entrada do Hadoop, e executa uma tarefa Hadoop de modo a empregar uma função Mapper e Reducer. Logo depois, os dados gerados e processados são lidos do HDFS e são atribuídos em uma variável. No final, esses dados são guardados localmente em um ficheiro de texto. Se ocorrer algum erro, ele é registado, e a função então rejeitará a promessa.

```
async function hadoopCommands() {
 return new Promise((resolve, reject) => {
   const outputFile = './output_hadoop.txt';
   let outputData = '';
```

```
    const ssh = new SSH({
      host: '10.0.0.128',
      user: 'hadoop',
      pass: '1234'
    });

    ssh.exec(`/home/hadoop/hadoop/bin/hdfs dfs -
         rm -r -f ${hadoopOutputPath}`, {
          out: console.log.bind(console),
          err: console.error.bind(console),
        })
        .exec(`/home/hadoop/hadoop/bin/hdfs dfs -
             rm -r ${hadoopInputPath}/catalog.csv`
             , {
          out: console.log.bind(console),
          err: console.error.bind(console),
        })
        .exec(`/home/hadoop/hadoop/bin/hdfs dfs -
             put -f ${remoteCsvFilePath} ${
             hadoopInputPath}`, {
          out: console.log.bind(console),
          err: console.error.bind(console),
        })
        .exec(`/home/hadoop/hadoop/bin/hdfs dfs -
             test -e ${hadoopInputPath}/catalog.
             csv`, {
          out: (data) => {
            console.log('File exists:', data);
          },
          err: console.error.bind(console),
        })
        .exec(`/home/hadoop/hadoop/bin/hadoop jar
             /home/hadoop/hadoop/share/hadoop/
             tools/lib/hadoop-streaming-3.3.4.jar
             -mapper /home/hadoop/mapper.py -
             reducer /home/hadoop/reducer.py -
             input ${hadoopInputPath}/catalog.csv
             -output ${hadoopOutputPath}`, {
          out: console.log.bind(console),
          err: console.error.bind(console),
        })
        .exec(`/home/hadoop/hadoop/bin/hdfs dfs -
             cat ${hadoopOutputPath}/*`, {
          out: (data) => {
            outputData += data;
          },
          err: console.error.bind(console),
        })
        .exec(`/home/hadoop/hadoop/bin/hdfs dfs -
             rm -r -f ${hadoopOutputPath}`, {
          out: console.log.bind(console),
          err: console.error.bind(console),
        })
        .on('end', () => {
          if (outputData !== '') {
            fs.writeFile(outputFile, outputData, (
                 err) => {
              if (err) {
                console.error('Erro ao tentar
                     gravar o arquivo:', err);
                reject(err);
              }
              console.log('Todos os dados foram
                   salvos em:', outputFile);
              resolve();
            });
          } else {
            console.error('> Any Object as Been
                 Saved in The Cat.');
            resolve();
          }

          ssh.end();
        })
        .on('error', (err) => {
          console.error('> Error Occurred During
               SSH:', err);
          reject(err);
        })
        .start();
    });
  }
```

Função InsertCatalog():

Funcionamento: A função InsertCatalog() conecta-se a um servidor remoto via SSH, faz o upload do arquivo catalog.csv, e executa comandos Hadoop para processá-lo. Caso o upload ou os comandos falhem, a função encerra a conexão e retorna um erro. Ela é usada para automatizar a integração e o processamento dos livros num ambiente Hadoop.

```
async function InsertCatalog() {
  return new Promise((resolve, reject) => {
    Client({
       host: '10.0.0.128',
       port: 22,
       username: 'hadoop',
       password: '1234',
    }).then(async (client) => {
      client.uploadFile('./catalog.csv', `${
           remoteCsvFilePath}`)
      .then(async (response) => {
        console.log('> File Inserted and Now
             Running Hadoop');
        client.close();
        resolve();
      }).catch((error) => {
         console.error('> File Upload Error: ',
              error);
        client.close();
        reject(error);
      });
    })
    .catch((error) => {
       console.error('> SSH Connection Error: ',
            error);
      reject(error);
    });
  });
}
```

2.6 Exemplo Ilustrado

Exemplo - Pedido CURL showBook

```
curl -X GET "http://localhost:5000/api/showBook
     /?id=1" -H "Content-Type: application/json"
{"ID":1,"YEAR":"1971-12-01","TITLE":"The
     Declaration of Independence of the United
     States of America","LANGUAGE":"en","AUTHORS
     ":"Jefferson, Thomas, 1743-1826"}
```

Exemplo - Pedido CURL updateBook

```
curl -X POST "http://localhost:5000/api/
     updateBook/?id=1&date=12-12-2024&title=
     Titulo%20Bonito&language=en&authors=Eu" -H
     "Content-Type: application/json"
{"ID":1,"YEAR":"1971-12-01","TITLE":"The
     Declaration of Independence of the United
     States of America","LANGUAGE":"en","AUTHORS
     ":"Jefferson, Thomas, 1743-1826"}
```

Exemplo - Pedido CURL insertBook

```
curl -X POST "http://localhost:5000/api/
     insertBook/?date=12-12-2024&title=Titulo%20
     Bonito&language=en&authors=Eu" -H "Content-
     Type: application/json"
{"ID":1,"YEAR":"1971-12-01","TITLE":"The
     Declaration of Independence of the United
     States of America","LANGUAGE":"en","AUTHORS
```

```
":"Jefferson, Thomas, 1743-1826"}
```

Exemplo - Pedido CURL deleteBook

```
curl -X DELETE "http://localhost:5000/api/
    deleteBook/?id=1" -H "Content-Type:
    application/json"
{"ID":1,"YEAR":"1971-12-01","TITLE":"The
    Declaration of Independence of the United
    States of America","LANGUAGE":"en","AUTHORS
    ":"Jefferson, Thomas, 1743-1826"}
```

3 Configuração Experimental

3.1 Arquitetura do sistema implementado

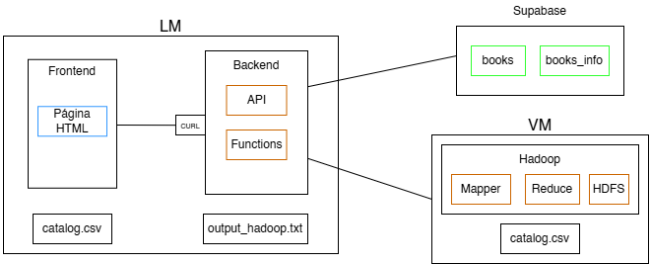

Fig. 2. Arquitetura Simples do Projeto

O utilizador interage com a interface HTML ou utiliza o CURL para inserir um novo livro, como descrito anteriormente. O backend, ao receber esses dados, realiza uma validação para garantir que todas as informações estejam corretas e completas.

Após a validação, o backend comunica-se com o Supabase, para registar o livro na tabela "books". Isso é feito através da função insertBook, que insere os dados validados. Em seguida, a função updateFileFromInfo() é chamada para extrair os dados da tabela "books" e convertê-los em um ficheiro CSV atualizado, chamado catalog.csv. Este ficheiro contém uma representação estruturada de todos os livros registados até o momento, que será usado para o processamento.

Com o ficheiro CSV gerado, o backend chama a função InsertCatalog, que se conecta ao servidor Hadoop, realizando o upload do ficheiro catalog.csv. Uma vez o arquivo carregado, a função `hadoopCommands()` executa os comandos necessários no Hadoop, incluindo a execução do MapReduce, que processa os dados de maneira distribuída e paralela. Após a execução do MapReduce, os dados processados são guardados no ficheiro output_hadoop.txt.

A função processHDFSFile() entra em ação em seguida, lendo o conteúdo do ficheiro output_hadoop.txt, que contém os dados agregados do processamento realizado no Hadoop. Essa função analisa o conteúdo e organiza os dados de maneira estruturada para posterior armazenamento. Finalmente, os dados processados, com a informação do ano e quantidade de livros são enviados de volta ao Supabase, onde são atualizados na tabela book_info, garantindo que a informação final seja armazenada corretamente.

Em cada uma dessas etapas, o sistema inclui verificações de erro para garantir que qualquer falha, como problemas na validação de dados, falhas na comunicação com o Supabase, ou erros durante a execução dos comandos no Hadoop, seja identificada e relatada ao utilizador ou ao administrador.

3.2 Benchmarks

TABLE 2
Benchmarks

Benchmark	Resultado	Observações
Insert All Books	N/A	Não foi possível devido a limitações de rede.
Update/Insert/Delete	14.52s	Dependendo da capacidade da máquina virtual e do hospedeiro, neste caso foi utilizado um `Ryzen 7 5800H` a `4.0GHz`, 16GB de memória `DDR4` a `3200MHz` mas para o `Hadoop` foram usados `3vCores` e apenas `2GB` de memória.
Show Book	1.24s	Dependendo da conexão, é rápida a obtenção da informação do livro em específico, em que basta enviar o `ID` do livro correspondente.
Hadoop Book Map	12.09s	Esta fase é a mais demorada, é necessário processar todos os 72.000 livros e igualmente escrever no seu output.
Update Catalog File	2.03s	Mais uma vez, dependendo da máquina usada e da conexão, mas permitido um processo bastante rápido, unicamente fazer o pedido à base de dados e extrair para um novo ficheiro catalog.csv.
Update Book Info	5.27s	Novamente, dependendo da máquina usada e da conexão, mas também a quantidade de dados, permite um processo bastante rápido, unicamente fazer o pedido à base de dados e inserir cada ano e a quantidade de livros.

3.3 Detalhes de instalação e configuração

3.3.1 *Criar uma Conta e um Projeto no Supabase*

- Acessar https://supabase.com e criar uma conta.
- Fazer login e criar um novo projeto em "New Project".
- Definir um nome para o projeto.
- Configurar a base de dados.
- Selecionar a região do servidor mais adequada para o projeto.

3.3.2 *Configuração Inicial do Supabase*

- Acessar a aba "Database" no painel do Supabase. Utilizar a interface para criar uma nova tabela que represente os catálogos de livros. Por exemplo:

Tabela: books

Colunas:
- ID (primary key, unique, integer)
- YEAR (integer)
- TITLE (text)
- LANGUAGE (text)
- AUTHORS (text)

- Novamente, utilizamos a interface para criar uma outra nova tabela que represente a quantidade de livros pelo ano correspondente. Por exemplo:

Tabela: books_info

Colunas:
- YEAR (primary key, unique, integer)
- AMOUNT (integer)

- Ativar a API REST para esta tabela (a API do Supabase é ativada por padrão).

3.3.3 Instalar o Supabase no Ambiente de Desenvolvimento

- Certificar de que o Node.js está instalado no sistema.
- Para verificar, é necessário executar no terminal: "node -v".
- Instalar a biblioteca Supabase no backend Node.js: "npm install @supabase/supabase-js"

3.3.4 Configurar a Ligação com a Supabase

- Aceder à aba "API" e copiar a URL da API e a chave pública (anon key).

Ficheiro de Configuração

Configurar a ligação ao Supabase no ficheiro de configuração do backend Node.js.

```
const { createClient } = require('
    @supabase/supabase-js');

const supabaseURL = "URL";
const supabaseKey = "ANON_KEY";
const supabase = createClient(supabaseURL
    , supabaseKey);
```

3.3.5 Inserir Dados na Tabela

- Carregar o catálogo inicial (catalog.csv).

3.3.6 Testar as Funcionalidades

- Testar os serviços REST utilizando um cliente como Insomnia/curl.

3.4 Conjunto de dados utilizados

O conjunto de dados utilizado neste projeto é o ficheiro `books.csv`, extraído do Projeto Gutenberg. Este arquivo contém informações detalhadas sobre aproximadamente 72.000 livros de domínio público. As principais características dos dados incluem:

- **Tamanho**: O ficheiro ocupa cerca de 50 MB em disco, variando conforme o formato de exportação.
- **Formato**: Ficheiro em formato CSV, estruturado em colunas delimitadas por vírgulas.
- **Características**: Cada linha do arquivo representa um livro, com informações como `ID`, `title`, `authors`, `language`, e `year`.

4 Resultados Experimentais

Nesta categoria decidimos realizar um vídeo demonstrativo de como os objetivos do trabalho foram concluidos, demonstrando o uso do Curl para realizar pedidos, como o caso dos usados no vídeo, InsertBook e DeleteBook, sendo também possível o UpdateBook e o ShowBook como anteriormente apresentado no relatório. Igualmente, o processamento completo do ficheiro CSV, atualizando o mesmo e processando toda a informação com o Hadoop, acabando por gerar um ficheiro de output com todos os anos e quantidades de livros correspondentes, para de seguida, ser enviado para a base de dados.

Vídeo Demonstrativo

5 Conclusão

Em conclusão, este projeto proporciona uma experiência inicial valiosa no desenvolvimento de aplicações em ambiente na nuvem, oferecendo desafios e oportunidades de aprendizagem. Com uma abordagem inovadora, o projeto propõe uma solução interessante para a gestão e análise de catálogos de livros digitais, utilizando tecnologias avançadas e integradas, como Supabase e Hadoop.

A experiência adquirida ao longo deste desenvolvimento foi fundamental para a compreensão das complexidades e das vantagens da computação em nuvem, preparando-nos para enfrentar desafios mais complexos em futuros projetos tecnológicos.

References

[1] Tiago M. C. Simões. link. Acedido: 19/12/2024.