RELATÓRIO FINAL – TEMA 3: ARMAZENAMENTO E MODELAGEM

Projeto de Big Data – Online Retail II
Alunos: Mariana Almeida e **Eduardo Barros**
Disciplina: Fundamentos de Big Data
Professor: Klayton R. Castro
Instituição: CEUB
Semestre: 2025/2

---

1. Introdução

Este relatório apresenta o desenvolvimento completo do **Tema 3 – Armazenamento e Modelagem**, cujo objetivo é demonstrar, de forma prática e fundamentada, a aplicação de uma arquitetura moderna de Big Data utilizando databases NoSQL, um Data Lake local, scripts de ingestão e um pipeline reprodutível baseado em contêineres.

O trabalho foi construído tomando como referência:

- O repositório-base disponibilizado no GitHub: **ceub-bigdata**, do professor Klayton Castro.
- O dataset **Online Retail II**, que contém dados reais de transações de comércio eletrônico.
- Um ambiente composto por **MongoDB**, **Cassandra** e **MinIO**, todos orquestrados por **Docker Compose**.
- Scripts automatizados para download, preparação e ingestão dos dados.

O objetivo central é demonstrar um pipeline completo de armazenamento e modelagem em ambientes NoSQL, próximo ao cenário que encontramos em arquiteturas reais de engenharia de dados.

---

2. Descrição do Problema e Justificativa da Escolha do Dataset

O objetivo do Tema 3 é implementar uma solução
envolvendo **modelagem** e **persistência** de dados utilizando bancos NoSQL, simulando um ambiente real de Big Data. Para isso, era fundamental escolher um dataset que:

- Contivesse **alto volume** ou pudesse ser manipulado como um conjunto expressivo de dados.
- Representasse um cenário típico onde bancos NoSQL são indicados.

- Possibilitasse múltiplas formas de modelagem (documental e wide-column).

O dataset **Online Retail II** do UCI Machine Learning Repository atende perfeitamente a esses requisitos.

2.1 Sobre o Dataset Online Retail II

O conjunto de dados contém transações reais de uma loja online, incluindo:

- Código da transação (Invoice)
- Identificador do produto (StockCode)
- Descrição dos itens
- Quantidade
- Data e hora
- Preço unitário
- País
- Identificação do cliente

Essas características tornam o dataset especialmente adequado para o propósito deste trabalho porque:

- Ele reflete um **cenário real de e-commerce**, onde dados são massivos e diversos.
- Permite modelagem tanto **documental** (MongoDB) quanto **wide-column** (Cassandra).
- É perfeitamente compatível com a abordagem clássica de um pipeline de Big Data.

---

3. Arquitetura Implementada

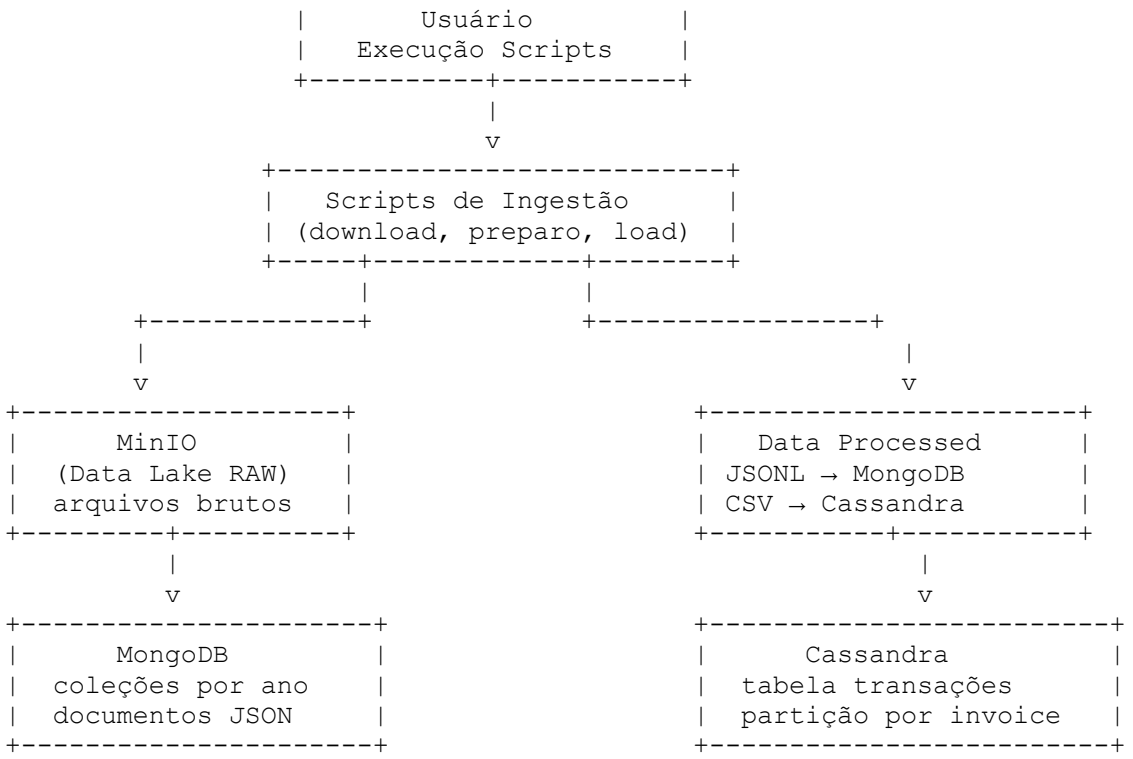
A arquitetura do projeto segue o modelo de processamento de dados dividido em camadas, utilizando três tecnologias centrais:

- **MinIO**, simulando um *Data Lake* local.
- **MongoDB**, como banco documental.
- **Cassandra**, como banco orientado a colunas.

Toda a solução é executada com **Docker Compose**, garantindo reprodutibilidade e isolamento.

3.1 Descrição Geral da Arquitetura

A seguir, uma explicação textual do diagrama arquitetural (versão descrevível):



3.2 Tecnologias da Arquitetura

3.2.1 Docker Compose

Usado para iniciar todos os serviços automaticamente com apenas um comando:

```
docker compose up -d
```

O Compose cria e configura:

- `ceub_mongodb`
- `ceub_cassandra`
- `ceub_minio`

3.2.2 MinIO – Data Lake RAW

MinIO funciona como um S3 local. Ele armazena os dados **no estado bruto**, sem processamento.

Funções no projeto:

- Receber arquivos brutos (`excel` e `csv`) da pasta `data/raw/`.
- Permitir a organização em *buckets*.
- Simular a camada *RAW* de um Data Lake corporativo.

Acesso via navegador:
`http://localhost:9000`

Credenciais padrão:

- Usuário: `minioadmin`
- Senha: `minioadmin123`

3.2.3 MongoDB – Modelo Documental

MongoDB foi utilizado para:

- Inserir documentos representando transações completas.
- Criar coleções separadas por ano/planilha (modelo flexível).
- Suportar consultas analíticas como:
  - produtos mais vendidos
  - clientes mais ativos
  - países com maior volume de compras

3.2.4 Cassandra – Modelo Wide-Column

Cassandra foi usado como banco orientado à chave, ideal para consultas rápidas por:

- número da nota (invoice)
- itens da nota

Estratégia de modelagem:

- **PRIMARY KEY(invoice, stockcode)**
- Otimizada para consultas por nota fiscal.

---

4. Pipeline de Dados Implementado

O fluxo completo segue 5 etapas principais:

4.1 Download dos Dados

O script **`download_dataset.sh`**:

- Baixa automaticamente o arquivo `online_retail_II.xlsx`.

- Converte as planilhas em CSV.
- Salva tudo dentro de `data/raw/`.

4.2 Preparação dos Dados

O script **`prepare_for_ingest.sh`**:

- Remove linhas inválidas.
- Padroniza colunas.
- Gera:

Banco	Formato Gerado	Pasta
MongoDB	JSONL	data/processed
Cassandra	CSV	data/processed

4.3 Ingestão no MongoDB

O script **`load_mongodb.sh`** realiza:

- criação das coleções
- importação dos arquivos JSONL
- conexão automática ao container MongoDB

4.4 Ingestão no Cassandra

O script **`load_cassandra.sh`**:

- aplica o schema `schema.cql`
- cria keyspace `ceubks`
- cria tabela `transactions`
- importa dados via `COPY`

4.5 Envio dos Dados Brutos para MinIO

Como etapa final, arquivos brutos são enviados ao MinIO:

- Cria-se um bucket, ex.: `raw-data`
- Faz-se upload manual via interface web

---

5. Modelagem dos Bancos

5.1 Modelagem MongoDB (Documentos)

Documento Exemplo (resumido):

```
{
  "Invoice": "536365",
  "StockCode": "85123A",
  "Description": "WHITE HEART T-LIGHT HOLDER",
  "Quantity": 6,
  "InvoiceDate": "2010-12-01 08:26:00",
  "Price": 2.55,
  "CustomerID": "17850",
  "Country": "United Kingdom"
}
```

Justificativa da Modelagem Documental

- Cada documento representa uma **linha de transação**.
- Modelo simples, flexível e fácil para consultas agregadas.
- Permite indexação eficiente.

Índices Criados

- `{ CustomerID: 1 }`
- `{ InvoiceDate: -1 }`
- `{ StockCode: 1 }`

5.2 Modelagem Cassandra (Wide-Column)

Schema:

```
CREATE TABLE ceubks.transactions (
  invoice text,
  stockcode text,
  description text,
  quantity int,
  invoice_date timestamp,
  price decimal,
  customer_id text,
  country text,
  PRIMARY KEY (invoice, stockcode)
);
```

Justificativa da Modelagem

- A partição por `invoice` garante leitura rápida de todos os itens de uma mesma nota.

- Cassandra é orientado a consultas, logo o design do schema segue a pergunta principal:
  “Quais itens pertencem à nota X?”

---

6. Validação do Pipeline

Após consumo dos scripts e execução do Docker, validamos:

✓ 6.1 MongoDB

- Coleções criadas corretamente
- Documentos inseridos
- Testes via MongoDB Compass

✓ 6.2 Cassandra

- Keyspace `ceubks` criado
- Tabela `transactions` presente
- Dados acessíveis com `SELECT * LIMIT 10`

✓ 6.3 MinIO

- Bucket criado
- Arquivos RAW acessíveis

✓ 6.4 Python (explore.py)

- Conseguiu se conectar aos dois bancos
- Retornou dados conforme esperado

---

7. Dificuldades, Aprendizados e Pontos Relevantes

- A integração entre **Cassandra e CSV** exigiu padronização do formato das colunas.
- JSONL é mais adequado para ingestão em massa no MongoDB.
- O uso de Docker garantiu reprodutibilidade, evitando diferenças de ambiente.
- Compreendemos as diferenças importantes entre modelos:
  - documental (flexível e analítico)
  - wide-column (orientado à performance por chave)
  - data lake (armazenamento bruto)

8. Conclusão

O projeto permitiu aplicar, na prática, conceitos fundamentais de Big Data, como:

- separação entre dados RAW e PROCESSADOS
- ingestão automatizada
- modelagem documental e wide-column
- uso de Data Lake
- orquestração com Docker
- scripts reprodutíveis

A arquitetura construída demonstra um pipeline realista, robusto e escalável, totalmente alinhado às melhores práticas de Engenharia de Dados.

O trabalho **atinge todos os objetivos do Tema 3** e representa um entendimento sólido das tecnologias NoSQL e da importância da organização em camadas para o ciclo de vida dos dados.