



# **Análise de Dados em Tempo Real**

DataCentric

2022 / 2023

**1191507 – Bárbara Diana da Cruz Pereira Pinto**

**ISEP** INSTITUTO SUPERIOR  
DE ENGENHARIA DO PORTO



# **Análise de Dados em Tempo Real**

DataCentric

2022 / 2023

1191507 – Bárbara Diana da Cruz Pereira Pinto



## **Licenciatura em Engenharia Informática**

Junho de 2023

Orientador ISEP: **Ricardo Almeida**

Supervisor Externo: **Paulo Faria**

# Resumo

Com a evolução das infraestruturas digitais, a captação de grandes quantidades de dados tem vindo a ficar mais fácil e mais valiosa para os negócios.

No contexto do *e-commerce*, a captação e processamento de dados em tempo real surge como uma vantagem competitiva, permitindo responder rapidamente às mudanças no mercado e melhorar a experiência do cliente enquanto navega.

Neste relatório é abordado o desenvolvimento de uma arquitetura de análise de dados em tempo real, com aplicação a um sistema de recomendação para uma plataforma de *e-commerce*. O objetivo principal do trabalho consiste em explorar as vantagens da captação e processamento de dados em tempo real, bem como o seu grau de adequação à aplicação selecionada.

O presente relatório apresenta, numa primeira instância, uma análise do problema e do domínio envolvente, seguida do desenho da solução e da arquitetura adotada, terminando com a descrição do processo de implementação e a avaliação da solução.

Com o trabalho realizado, conclui-se que o sistema de recomendação em tempo real desenvolvido apresenta o desempenho e escalabilidade necessária para ser integrado e contribuir para a melhor experiência de utilizador.

**Palavras-chave (Tema):** Engenharia de dados, análise em tempo real, sistemas de recomendação, *e-commerce*.

**Palavras-chave (Tecnologias):** Python, Apache Kafka, Apache Spark, MySQL, Docker, Azure.

# Índice

<b>1</b>	<b><i>Introdução</i></b>	<b>1</b>
1.1	Enquadramento/Contexto	1
1.2	Descrição do Problema	2
1.2.1	Objetivos	3
1.2.2	Abordagem	3
1.2.3	Contributos	4
1.2.4	Planeamento do trabalho	5
1.3	Estrutura do relatório	6
<b>2</b>	<b><i>Estado da arte</i></b>	<b>8</b>
2.1	Engenharia de dados	8
2.1.1	Armazenamento de dados	9
2.1.2	Extração	11
2.1.3	Transformação	13
2.2	Sistemas de recomendação	14
2.2.1	Filtragem de conteúdo	15
2.2.2	Filtragem colaborativa	15
2.3	Trabalhos relacionados	16
2.4	Tecnologias existentes	18
2.4.1	Ferramentas e frameworks	18
2.4.2	Linguagens de programação	25
<b>3</b>	<b><i>Análise e desenho da solução</i></b>	<b>27</b>
3.1	Partes envolvidas do negócio	27
3.2	Domínio do problema	28
3.3	Requisitos funcionais	30
3.4	Requisitos não funcionais	35
3.5	Desenho	36
3.5.1	Arquitetura global	37
3.5.2	Arquiteturas alternativas	44
<b>4</b>	<b><i>Implementação da solução</i></b>	<b>49</b>
4.1	Tecnologias utilizadas	49
4.2	Funcionalidades implementadas	51

4.3	Avaliação da solução .....	61
5	Conclusões.....	63
5.1	Objetivos concretizados .....	63
5.2	Limitações e trabalho futuro .....	64
5.3	Apreciação final .....	65
	Referências.....	67
	Anexo A – Lista de testes de qualidade efetuados .....	75

# Índice de Figuras

Figura 1 - Planeamento do trabalho (Diagrama de Gantt) .....	6
Figura 2 - Exemplo de matriz de correlação .....	16
Figura 3 - Resumo das ferramentas analisadas e respetiva fase do processo .....	19
Figura 4 - Modelo de domínio .....	29
Figura 5 - Diagrama de casos de uso .....	35
Figura 6 - Arquitetura global do projeto .....	37
Figura 7 - Arquitetura geral de processamento e armazenamento .....	38
Figura 8 - Arquitetura detalhada de processamento e armazenamento.....	39
Figura 9 - Modelo de dados da estrutura local .....	40
Figura 10 - Modelo de dados da estrutura staging .....	42
Figura 11 - Modelo de dados da estrutura dw .....	44
Figura 12 - Arquitetura alternativa de processamento em tempo real utilizando streaming e micro-batches.....	45
Figura 13 - Arquitetura alternativa de processamento em tempo real utilizando Azure .....	47
Figura 14 - Componentes executadas em containers locais no Docker.....	52
Figura 15 - Componentes executadas em ambiente cloud na plataforma Azure.....	53
Figura 16 - Captação de eventos operacionais em tempo real (Diagrama de sequência).....	54
Figura 17 - Armazenamento dos eventos em armazém de dados (Diagrama de sequência).	56
Figura 18 - Desenvolvimento de sistema de recomendação (Diagrama de sequência) .....	58
Figura 19 - Consulta utilizada para obter recomendações a partir da tabela cf_similarity_matrix .....	59
Figura 20 - Visualização da latência de processamento para os registos da tabela fac_order_items .....	61

# Índice de Tabelas

Tabela 1 - Planeamento do trabalho .....	5
Tabela 2 - Diferenças entre DW e DL.....	10
Tabela 3 - Diferenças entre event-processing e micro-batching.....	11
Tabela 4 - Diferenças entre on-premise computing e cloud computing.....	13
Tabela 5 - Comparação de Apache Spark, Apache Kafka e Azure Event Hubs .....	21
Tabela 6 - Comparação de Apache Hadoop HBase e Azure Data Lake Storage Gen 2.....	22
Tabela 7 - Comparação de Hadoop MapReduce, Apache Spark e Apache Ignite .....	24
Tabela 8 - Descrição do caso de uso/user story DCRA-1 .....	30
Tabela 9 - Descrição do caso de uso/user story DCRA-2 .....	31
Tabela 10 - Descrição do caso de uso/user story DCRA-3 .....	32
Tabela 11 - Descrição do caso de uso/user story DCRA-4 .....	33
Tabela 12 - Descrição do caso de uso/user story DCRA-5 .....	34
Tabela 13 - Requisitos não funcionais .....	36
Tabela 14 - Resumo das principais tecnologias utilizadas.....	50
Tabela 15 - Lista resumida de testes efetuados (US DCRA-3) .....	55
Tabela 16 - Lista resumida de testes efetuados (US DCRA-4) .....	57
Tabela 17 - Lista resumida de testes efetuados (US DCRA-5) .....	60
Tabela 18 - Lista de milestones e respetivo grau de realização .....	63



# Notação e Glossário

<b>CDC</b>	<i>Change Data Capture</i>
<b>CTR</b>	<i>Click Through Rate</i>
<b>DBMS</b>	<i>Database Management System</i>
<b>DL</b>	<i>Data Lake</i>
<b>DW</b>	<i>Data Warehouse</i>
<b>ESG</b>	<i>Enterprise Research Group</i>
<b>ETL</b>	<i>Extract, Transform, Load</i>
<b>HDFS</b>	<i>Hadoop Distributed File System</i>
<b>IoT</b>	<i>Internet of Things</i>
<b>OLAP</b>	<i>Online Analytical Processing</i>
<b>OLTP</b>	<i>Online Transaction Processing</i>
<b>QA</b>	<i>Quality Assurance</i>
<b>RDD</b>	<i>Resilient Distributed Dataset</i>
<b>SaaS</b>	<i>Software-as-a-Service</i>
<b>SCD</b>	<i>Slowly Changing Dimension</i>
<b>UC</b>	<i>Use Case</i>
<b>US</b>	<i>User Story</i>

# 1 Introdução

Este capítulo visa contextualizar o leitor acerca do projeto desenvolvido, da área em que se insere e da sua relevância quer para a empresa onde foi desenvolvido como para a comunidade científica. Será feita a apresentação do enquadramento da engenharia de dados e da análise de dados em tempo real nas empresas, a descrição do problema e solução específicos ao projeto, bem como os respetivos objetivos, abordagem e planeamento.

## 1.1 Enquadramento/Contexto

À medida que as infraestruturas digitais continuam a crescer, as empresas são capazes de captar cada vez maiores quantidades de dados. Estes são recursos valiosos uma vez que permitem obter informações sobre as suas operações, clientes e mercados, consequentemente tomando decisões informadas, desenvolvendo estratégias eficazes e otimizando o negócio[1].

No entanto, para poder utilizar esse volume de dados torna-se necessário agregá-los a partir de várias fontes, armazená-los, processá-los e analisá-los. A engenharia de dados desempenha um papel importante por isso mesmo, fornecendo as ferramentas, tecnologias e processos necessários para a construção de pipelines capazes de captar, armazenar, transformar e carregar dados em ferramentas analíticas, como modelos de *machine learning* e *dashboards*.

Especificamente, a captação e o processamento de dados em tempo real estão a tornar-se cada vez mais relevantes, dada a facilidade crescente de acumular grandes volumes de dados e extrair *insights*. A aceleração do processo agrega valor por meio de[2]:

- Tomada de decisões oportuna – acedendo a dados em tempo real para responder rapidamente às mudanças no mercado e obter vantagem competitiva;
- Análise preditiva - identificando padrões e tendências que podem ser usados para prever futuros resultados;
- Melhor experiência do cliente - captando dados comportamentais em tempo real para identificar problemas e fornecer soluções, bem como ofertas e recomendações relevantes instantaneamente;
- Maior eficiência operacional - captando dados sobre processos em tempo real para resolver *bottlenecks* e ineficiências rapidamente, melhorando a eficiência geral e reduzindo custos.

Essas características são especialmente relevantes quando se consideram setores de mercado altamente competitivos e voláteis, como o retalho, que exigem uma resposta rápida às mudanças no contexto por forma a ficar à frente dos concorrentes. Os retalhistas procuram redesenhar as suas infraestruturas de dados, afastando-se de arquiteturas rígidas e lentas e apostando na migração para novas, de forma a responder ao ritmo cada vez mais acelerado das expectativas e requisitos de retenção dos clientes.

Com base neste enquadramento, a DataCentric<sup>1</sup>, como empresa de consultoria na área de análise de dados e cujos principais clientes são grandes empresas retalhistas nacionais, demonstra particular interesse na análise de dados em tempo real.

A empresa pretende desenvolver à priori conhecimento relacionado com técnicas de processamento de dados em tempo real bem como aplicações analíticas para esses mesmos dados. A exploração desta área visa antecipar futuras propostas de projetos por parte dos clientes, que já implementam geração de dados em tempo real (isto é, já fornecem fontes de dados com escrita em tempo real, como é o exemplo dos sistemas que registam transações de caixa de supermercado).

## 1.2 Descrição do Problema

O presente problema surge como consequência da adoção crescente por parte dos clientes da DataCentric de sistemas de registo de transações em tempo real, que são mais frequentemente utilizados como fonte de dados nos processos de engenharia de dados.

As arquiteturas utilizadas atualmente nos projetos liderados pela empresa não permitem responder aos problemas e exigências que uma aplicação de análise de dados integralmente em tempo real produziria. As tecnologias existentes nas arquiteturas de *big data* da DataCentric, como são exemplo as frameworks Hadoop e Spark, foram desenvolvidas para lidar com replicação de dados distribuída por várias unidades de processamento.

No entanto, essas tecnologias são ainda inadequadas para enfrentar os desafios relacionados com a redundância de dados, qualidade, inconsistência e custo de armazenamento. Além disso, carecem de uma estrutura (*schema*) para minimizar a redundância e não são capazes o suficiente de armazenar enormes quantidades de dados[3].

Assim, torna-se essencial adaptar as arquiteturas para utilizar tecnologias capazes de abordar e minimizar estes problemas visando atender às necessidades presentes e futuras.

---

<sup>1</sup> Empresa acolhedora do estágio.

Deste modo, pretende-se desenvolver uma arquitetura de captação, processamento e análise de dados em tempo real, capaz de responder aos requisitos de tempo de processamento, custo de manutenção, qualidade de análise e potencial valor acrescentado ao negócio dos clientes da DataCentric. O modelo desenvolvido poderá ser utilizado como guia para projetos futuros da empresa que exijam a execução em tempo real.

### **1.2.1 Objetivos**

Este projeto apresenta como principal objetivo a geração com sucesso de dados analíticos em tempo real, suportada pela concretização das seguintes tarefas:

- Modelação de uma arquitetura robusta e escalável, com considerações relativas a reduzidos tempos de processamento e armazenamento de grandes quantidades de dados;
- Desenvolvimento de processos para capturar eventos operacionais em tempo real;
- Desenvolvimento de processos de deteção e tratamento automático de erros e inconsistência de dados;
- Análise e desenvolvimento de um sistema de recomendação baseado em filtragem colaborativa;
- Desenho de um modelo de dados analíticos para armazenamento e posterior apresentação de dados processados;
- Desenvolvimento de processos de armazenamento em armazém de dados;
- Desenvolvimento de processos de agendamento e execução automática da aplicação.

### **1.2.2 Abordagem**

A organização e método de trabalho transversal a todo o projeto baseou-se no *Scrum*, uma abordagem iterativa e incremental, delimitada por *sprints* em que as fases tradicionais do ciclo de desenvolvimento - análise de requisitos, desenho/arquitetura, desenvolvimento e testes – são revisitadas a cada iteração[4].

A escolha desta abordagem para o presente projeto de estágio passou pelo conjunto de benefícios que podem ser aplicados, entre eles:

- Envolvimento desde cedo do cliente[4], que neste caso sendo a própria DataCentric é especialmente importante para a obtenção de feedback regular e acompanhamento sistemático por parte do supervisor;

- Desenvolvimento iterativo[4], que permite aperfeiçoar e corrigir erros de iterações passadas à medida que novo conhecimento é obtido, sem prejuízo para a integridade ou pontualidade do projeto;
- Equipas auto-organizadas[4], cuja estrutura promove o poder de tomar decisões autónomas, a comunicação entre a equipa e a partilha de ideias;
- Adaptação à mudança[4], cuja importância para além dos potenciais novos requisitos propostos pelo cliente surge também, por exemplo, no caso de certas tecnologias estudadas no âmbito do projeto se provarem inadequadas.

A nível mais detalhado da conceção e implementação do processamento em tempo real de dados, seguiu-se a abordagem *Extract, Transform, Load (ETL)* que consiste em três fases de desenvolvimento:

1. Extração, que inclui a análise das fontes de dados existentes, estabelecimento de ligações e credenciais de acesso aos respetivos repositórios, bem como validação, limpeza e formatação dos dados para uso analítico[5];
2. Transformação, envolvendo a alteração da disposição dos dados, a adição ou cálculo de informação adicional ou a remoção de dados não relevantes, com o objetivo de produzir um modelo correspondente às necessidades técnicas e de negócio[5];
3. Carregamento, que armazena o resultado dos passos anteriores no sistema-alvo, podendo incluir o controlo histórico dos dados já aí existentes e a criação de tabelas de auditoria[5].

### **1.2.3 Contributos**

O presente projeto conta como aspetos inovadores a execução em tempo real, um modelo ainda pouco utilizado no tecido empresarial que compõe os clientes da DataCentric. Este modelo contrasta com a mais disseminada execução em *batch*, e enfrenta um conjunto de desafios, tais como latência de processamento e a disponibilidade/equilíbrio de carga de recursos de processamento.

Pretende-se com o desenvolvimento deste modelo em tempo real abrir uma nova área de *expertise* para a DataCentric, possibilitando a oferta de integrações valiosas junto dos seus clientes. Tais integrações contribuem para a retenção de clientes e a eficiência operacional, tais como funcionalidades de recomendações em tempo real, notificações automáticas para aprovisionamento de stocks, entre outros.

Junto dos consumidores em geral, muitas vezes o público-alvo destas inovações, prevê-se uma contribuição positiva na satisfação com a experiência de compra, pela maior relevância

temporal de recomendações e pela prevenção de quebras de stock, como consequência dos exemplos acima.

Para a comunidade científica e a área de engenharia de dados, pretende-se ainda chegar a conclusões relevantes acerca de métodos de otimização de recursos de computação, otimização de custos e casos de uso em que se justifica aplicar modelos em tempo real, fomentando inovação e pesquisa acerca de possíveis implementações da tecnologia nessas áreas.

#### 1.2.4 Planeamento do trabalho

Com base nos objetivos definidos para este projeto, definiram-se as principais fases do projeto, com a respetiva calendarização de acordo com a Tabela 1 e a Figura 1.

*Tabela 1 - Planeamento do trabalho*

<b>Milestone</b>	<b>Data de início</b>	<b>Data de fim</b>
1- Análise de estado da arte	27/02/2023	24/03/2023
2- Modelação de uma arquitetura robusta e escalável, com considerações relativas a reduzidos tempos de processamento e armazenamento de grandes quantidades de dados	27/03/2023	21/04/2023
3 - Desenvolvimento de processos para capturar eventos operacionais em tempo real	03/04/2023	21/04/2023
4 - Desenvolvimento de processos de armazenamento em armazém de dados	10/04/2023	28/04/2023
5 - Desenho de um modelo de dados analíticos para armazenamento e posterior apresentação de dados processados	17/04/2023	12/05/2023
6 - Análise e desenvolvimento de um sistema de recomendação baseado em filtragem colaborativa	01/05/2023	26/05/2023
7 - Desenvolvimento de processos de deteção e tratamento automático de erros e inconsistência de dados	29/05/2023	16/06/2023
8 - Desenvolvimento de processos de agendamento e execução automática da aplicação	19/06/2023	23/06/2023
9 – Escrita do relatório final	27/02/2023	26/06/2023

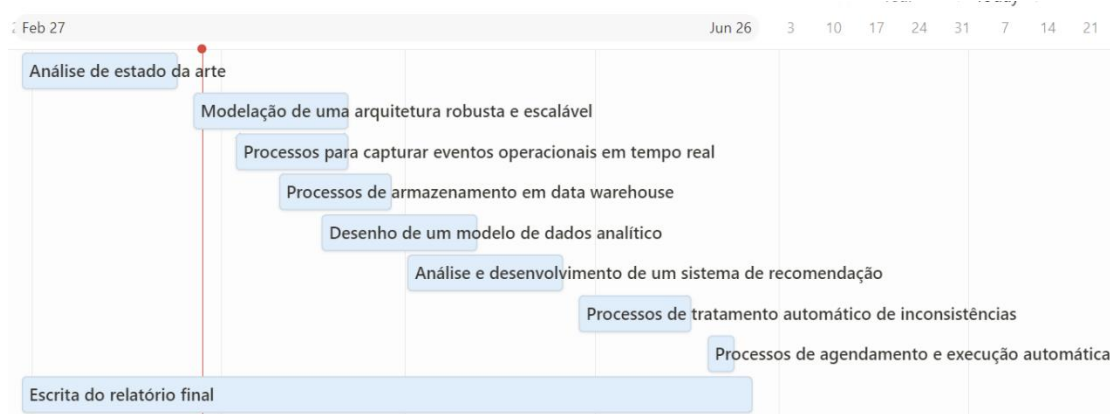


Figura 1 - Planeamento do trabalho (Diagrama de Gantt)

De notar que o plano concebido envolve uma fase inicial com bastante sobreposição de *milestones*, de modo a garantir que quaisquer requisitos ou limitações tecnológicas que surjam no início do desenvolvimento da fase 2 e 3 possam ser refletidas no modelo de arquitetura definido na fase 1. Igualmente, o desenho do modelo de dados analíticos sobrepõe-se tanto à fase anterior como à seguinte por forma a configurar *a priori* um armazenamento adequado às necessidades do modelo – fase 3 – e a formatação do *output* do sistema de recomendação também já de acordo com o modelo – fase 5.

### 1.3 Estrutura do relatório

O presente relatório divide-se em 5 capítulos, nomeadamente Introdução, Estado da Arte, Análise e desenho da solução, Implementação e Conclusões.

O presente capítulo, Introdução, pretende facultar uma visão geral do problema em estudo, os seus objetivos e a abordagem adotada.

O segundo capítulo, Estado da Arte, consiste numa introdução teórica aos conceitos de engenharia de dados e sistemas de recomendação, seguido de uma análise bibliográfica de trabalhos relacionados com relevância para os objetivos do projeto a ser desenvolvido. É ainda feita uma recolha das tecnologias de referência no desenvolvimento de projetos de análise de dados em tempo real.

O terceiro capítulo, Análise e desenho da solução, descreve as partes interessadas e o domínio do projeto como base para a construção dos requisitos funcionais e não funcionais, bem como a arquitetura e modelos de dados, também explorados no capítulo.

O quarto capítulo, Implementação da solução, detalha o processo de desenvolvimento de cada um dos requisitos funcionais, completo com o respetivo plano de testes e avaliação da solução.

Por fim, o quinto capítulo, Conclusões, pretende sistematizar os resultados obtidos, relacionando-os com os objetivos estabelecidos anteriormente. É também neste capítulo que é feita a reflexão acerca de limitações e pontos a melhorar, incluindo recomendações de futuras extensões e modificações.



## 2 Estado da arte

Neste capítulo será feita uma breve introdução teórica de alguns conceitos da área de engenharia de dados e sistemas de recomendação importantes para o entendimento técnico do presente projeto. Serão ainda conduzidas comparações de técnicas e tecnologias (plataformas, ferramentas e linguagens) relevantes, procurando identificar as vantagens e desvantagens de cada uma.

Finalmente, serão apresentados alguns casos de estudo onde a análise de dados em tempo real é aplicada atualmente.

### 2.1 Engenharia de dados

A engenharia de dados, designada em inglês por *data engineering*, consiste na construção de sistemas para a captação, processamento e limpeza de dados[6]. Muitas vezes o conceito associa-se com os de análise de dados (*data analysis*) e de ciência de dados (*data science*), apesar de apenas tratar da preparação dos dados para análise e do seu armazenamento em modelos adequados para o efeito[7].

Tipicamente, o processo de engenharia de dados consiste numa sequência de fases que transformam os dados brutos em dados adequados para análise. Este processo designa-se *Extract, Transform, Load* (ETL).

A fase de extração (*Extract*) visa combinar dados de uma ou várias fontes num único sistema. Esta captação pode ser feita de forma programada - uma vez por dia, por exemplo - ou através de *triggers* - sempre que há inserção de dados na fonte. O primeiro método é conhecido como processamento agendado (*timetable scheduling*), enquanto o segundo consiste em processamento por eventos (*event processing*). Estes conceitos são explicados com mais detalhes na secção 2.1.2.

A fase de transformação (*Transform*) é responsável pelo tratamento dos dados, eliminando erros e duplicados, normalizando os dados e convertendo-os nos formatos-alvo[6]. Esta é a fase mais intensiva em termos de processamento e, por isso, onde mais importa distinguir entre processamento a nível local (*on-premise computing*) ou na nuvem (*cloud computing*), explorados na secção 2.1.3.

A fase de carregamento (*Loading*) disponibiliza os dados num repositório central para uso analítico, quer por *dashboards*, plataformas de *business intelligence* ou equipas de *data science*[6].

Ao longo de todo o processo é ainda feito o armazenamento faseado dos dados. O resultado de cada uma das fases acima descritas é persistido num repositório de acordo com o seu nível de processamento (*raw*, *transformed* ou *processed*). Os repositórios mais utilizados para armazenar *big data* desta maneira são os armazéns de dados e os *data lakes*[8].

### 2.1.1 Armazenamento de dados

O armazenamento de dados é uma funcionalidade transversal ao processo de engenharia de dados. Existem atualmente sistemas estabelecidos, com diferentes arquiteturas, que fornecem um repositório central para os dados.

Os sistemas de gestão de bases de dados (*Database management systems* ou DBMSs), implementam geralmente um de dois ambientes estruturados: *Online analytical processing* (OLAP) e *Online transaction processing* (OLTP).

O sistema OLTP utiliza a lógica tradicional de bases de dados transacionais, normalmente utilizados para suportar as necessidades diárias do negócio. Neste sistema, os dados são armazenados de forma atómica, isto é, cada valor é armazenado numa linha. O sistema OLAP, por sua vez, suporta uma lógica de análise, agregando e sumarizando dados[9].

Num processo típico de engenharia de dados, ambos os sistemas são integrados. O ambiente OLTP é orientado para o cliente[10], integrado ao nível da produção e constituindo geralmente a fonte do processo. Já o ambiente OLAP é orientado para o mercado[10] e integrado ao nível da análise de dados. É utilizado em todo o processo após a fase de transformação e é aquele que mais será abordado no presente projeto.

Dentro deste ambiente, distinguem-se dois principais modelos de dados: armazém de dados e *data lake*.

Um armazém de dados (*data warehouse* ou DW) armazena dados de forma estruturada, filtrada e processada de acordo com um propósito específico[8]. Para um dado problema de negócio poderão existir vários armazéns de dados para analisar cada ângulo do problema.

Um *data lake* (DL) armazena dados no seu formato original, categorizados através de metadados[8]. Este é um modelo emergente, que tem vindo a ser mais utilizado devido à capacidade de reter informação mais abrangente e remover o *overhead* relacionado com a transformação de dados.

Uma análise detalhada das diferenças entre armazéns de dados e *data lakes* é apresentada na Tabela 2.

Tabela 2 - Diferenças entre DW e DL[8]

Parâmetro	Armazém de dados	Data lake
Dados	Focado em armazenar dados relevantes para uma dada análise	Armazena todos os dados do negócio
Tipos de dados	Dados estruturados (partilham o mesmo <i>schema</i> )	Dados estruturados, semi-estruturados e não estruturados
Processamento	Dados processados	Dados geralmente não processados
Propósito	Otimizado para consultas ( <i>queries</i> ) especializadas	Otimizado para consultas <i>ad hoc</i>
Flexibilidade	Configuração fixa, pouco flexível	Pode ser reconfigurada conforme necessário
Performance	Otimizado para eficiência no retorno de consultas	Otimizado para eficiência na ingestão de dados
Custos	Armazenamento de custo elevado, pela necessidade de recursos computacionais para transformação de dados	Armazenamento de custo reduzido
Granularidade	Dados sumarizados ou agregados	Dados detalhados
Utilização	Análise e reporte de dados, <i>business intelligence</i>	Filtragem e visualização de dados, <i>machine learning</i>

Conclui-se que os *data lakes* são ideais para a rápida ingestão e armazenamento geral de dados para análise posterior. Já os armazéns de dados são construídos para fornecer *insights* específicos para um problema de negócio predefinido, através de consultas rápidas e especializadas. Apesar de cada um ter as suas aplicações particulares, é possível combinar as suas forças, utilizando-os em diferentes fases de um processo ETL para o otimizar.

Como é explicado nos próximos capítulos “Arquitetura global”(secção 3.5.1) e “Tecnologias utilizadas”(secção 4.1), optou-se por utilizar uma *lakehouse* ao longo do processo ETL implementado. Uma *lakehouse* é uma arquitetura que combina as características de *data lake* e armazém de dados[11], podendo ser utilizada para qualquer um destes propósitos.

No início do processo ETL, o modelo *data lake* foi utilizado para permitir a ingestão rápida dos dados não estruturados que eram gerados na fonte. Ao longo do processo, esses dados foram sucessivamente filtrados e refinados até atingir uma lógica de armazém de dados, para potencializar as consultas em tempo real do lado do motor do sistema de recomendação.

### 2.1.2 Extração

A captação de dados para um processo de engenharia de dados é geralmente feita de forma contínua, com o objetivo de produzir análises atualizadas e úteis para a tomada de decisão. Existem atualmente duas principais abordagens para a captação de dados:

- Processamento agendado ou *timetable scheduling*, que consiste na captação de dados em intervalos pré-determinados. Este é o método mais tradicional, utilizado na ingestão de dados em *batch*[12].
- Processamento por eventos ou *event processing*, que inicia o processo de captação de dados sempre que uma ação ou *trigger* é despoletado, exigindo mais recursos computacionais pela necessidade de existirem *listeners* a monitorar quaisquer alterações[13].

No caso de aplicações em tempo real, é comum utilizar o processamento por eventos ou um caso particular do processamento agendado – o *micro-batching*, que utiliza intervalos de tempo muito curtos. Uma análise detalhada das diferenças destas duas alternativas é apresentada na Tabela 3.

Tabela 3 - Diferenças entre *event-processing* e *micro-batching*[14]–[16]

	<b><i>Micro-batching</i></b>	<b><i>Event processing</i></b>
<b>Dados</b>	Processa dados entre intervalos de tempo curtos	Processa dados assim que são gerados
<b>Tamanho dos dados</b>	Pequenos <i>batches</i> de dados	Fluxo contínuo de dados singulares
<b>Processamento</b>	Processamento em paralelo	Processamento em série
<b>Performance</b>	Maior latência	Latência baixa e alta taxa de transmissão
<b>Velocidade</b>	Próximo de tempo-real ( <i>sub-minute</i> )	Tempo-real ( <i>sub-second</i> )

	<i>Micro-batching</i>	<i>Event processing</i>
<b>Alocação de recursos</b>	Utilização eficiente e estável dos recursos	Alta alocação de recursos para implementar <i>listeners</i> e lidar com picos de volume de dados
<b>Escalabilidade</b>	Alta	Baixa
<b>Custo</b>	Baixo	Alto
<b>Operações</b>	Computações complexas ou operações <i>stateful</i> <sup>2</sup>	Transformações simples, agregações ou filtragem de dados; operações <i>stateless</i> <sup>2</sup>
<b>Utilização</b>	<i>Web analytics</i> , comportamento de utilizador	Publicidade em tempo-real, <i>online machine learning</i> , deteção de fraude

Um caso particular do *event processing* é o *change data capture* (CDC), que regista atividade numa base de dados quando tabelas ou linhas são modificadas. Com o CDC, operações de INSERT, UPDATE ou DELETE podem ser detetadas em tempo real e replicadas noutra plataforma, como por exemplo um armazém de dados. Este mecanismo é geralmente implementado usando *database triggers* ou *binary logs*[17].

Os *database triggers* consistem em procedimentos que são automaticamente executados em resposta a operações numa dada tabela numa base de dados[18].

Os pontos fortes dos *database triggers* passam por [17]:

- Facilidade de implementação, uma vez que já vêm implementados em muitos dos DBMSs existentes;
- Facilidade de gerir o evento na fonte, inserindo metadados que serão úteis no restante processo.

No entanto, a utilização de *triggers* aumenta a complexidade operacional da base de dados e afeta a sua *performance* [17]:

- Aumentam o tempo de execução das operações originais;

<sup>2</sup> Entende-se por *stateful* uma operação que incorpora o estado de elementos anteriores quando processa novos elementos, tal como *queries* que incluam comandos DISTINCT ou SORTED. Uma operação *stateless* constitui o oposto[83].

- Por serem internos à base de dados, exigem a execução contínua de consultas por parte de plataformas externas para obter os eventos;
- São dependentes do *schema* e sujeitos a alteração manual sempre que existe uma alteração da estrutura das tabelas que monitorizam.

Por sua vez, os *binary logs* são ficheiros internos da base de dados onde todas as operações são registadas, incluindo criação/remoção de tabelas, alterações ao seu *schema* e alterações seu ao conteúdo.

Ao contrário do que acontece com os *triggers*, esta implementação não afeta a *performance* da base de dados, lendo os eventos diretamente do sistema de ficheiros em vez de executar consultas. Além disso, não é dependente do *schema*, captando alterações à estrutura das tabelas sem necessidade de qualquer *input* manual[17].

### 2.1.3 Transformação

A fase de transformação na *pipeline* ETL é geralmente a mais intensiva em recursos computacionais, envolvendo filtrar dados, fazer mapeamento de campos, detetar duplicações, ordenar valores ou executar junções de tabelas. Neste contexto, a forma como a alocação de recursos é feita tem relevância para o *workflow*, sendo possível para o efeito recorrer a recursos locais ou na *cloud*.

A principal diferença entre a definição de computação na *cloud* e computação local é simplesmente a localização do software.

Na computação local ou *on-premise computing* o software é armazenado e executado na própria infraestrutura de hardware da empresa, localizada nas instalações físicas da mesma[19]. Já na computação na *cloud* ou *cloud computing* o software é armazenado e executado nos servidores do provedor do serviço, e é acessível através de um navegador Web ou aplicação proprietária[19].

A Tabela 4 apresenta as principais características diferenciadoras de ambos os métodos de computação.

Tabela 4 - Diferenças entre *on-premise computing* e *cloud computing*[19]–[22]

	On-premise computing	Cloud computing
<b>Custo</b>	Exige suportar todos os custos de manutenção, como custos de	Apenas são pagos os recursos utilizados e o custo de cada recurso é dependente do nível de consumo.

	On-premise computing	Cloud computing
	<i>hardware</i> de servidores, consumo de energia e espaço físico.	
<b>Tipo de custo</b>	Despesa de capital. Requer um alto investimento inicial, com custos esporádicos de manutenção.	Despesa operacional. Requer pagamentos regulares durante todo o período de utilização.
<b>Controlo</b>	Retém controlo total de todos os dados.	Os dados são detidos pelo provedor do serviço.
<b>Segurança</b>	A segurança e privacidade dos dados pode ser totalmente garantida pelo proprietário.	Sujeito a fugas de dados e violação de privacidade que estão fora do controlo do proprietário.
<b>Conectividade</b>	Pode ser utilizado online e offline. Não depende de fatores externos.	Requer acesso à internet e depende da disponibilidade do serviço.
<b>Escalabilidade</b>	Dispendiosa, devido a limitações de espaço e hardware.	Alta e de baixo custo.
<b>Personalização</b>	Total controlo sobre a configuração da infraestrutura.	Pouco controlo sobre a configuração da infraestrutura.

A escolha entre a computação local e na *cloud* depende, em última análise, das necessidades e preferências únicas de cada negócio. Embora a computação local ofereça controlo completo e segurança, requer investimentos em *hardware* e infraestrutura. Por outro lado, a computação na nuvem oferece flexibilidade e escalabilidade, mas tem custos de assinatura contínuos e depende do acesso à internet.

## 2.2 Sistemas de recomendação

Os sistemas de recomendação utilizam as experiências, opiniões e comportamentos dos utilizadores como base para seleccionar os produtos que estes considerarão mais relevantes entre os resultados possíveis[23]. Estes sistemas são aplicados em diversos tipos de *e-commerce*, por exemplo, na recomendação de filmes, séries, livros, notícias e produtos eletrónicos.

O processamento, modelação e análise de dados são fundamentais para o funcionamento destes sistemas, sendo responsáveis pela extração e transformação dos dados de vendas e de utilização que alimentam os algoritmos de recomendação. A aplicação da análise de dados em tempo real neste contexto tem ganho popularidade, sendo atualmente aplicada por grandes *players* tecnológicos como o YouTube[24], Instagram[25] e Netflix[26].

Apesar da abordagem em tempo real ser mais exigente e dispendiosa a nível computacional, permite gerar recomendações em resposta às interações de utilizadores, algo que não é possível com a abordagem *batch*, que é baseada em janelas temporais ao invés de eventos (secção 2.1.2).

Os algoritmos aplicados aos sistemas de recomendação e que definem os produtos a recomendar seguem 2 principais abordagens: filtragem de conteúdo e filtragem colaborativa[27].

### 2.2.1 Filtragem de conteúdo

A filtragem de conteúdo ou *content-based filtering* é uma abordagem *item-to-item* que explora as características de produtos que o utilizador gostou/adquiriu no passado e procura sugerir produtos com características semelhantes[27]. Por exemplo, se um utilizador adquire um livro de um do género “Policial”, serão recomendados outros livros do género “Policial”.

### 2.2.2 Filtragem colaborativa

A filtragem colaborativa ou *collaborative filtering* é uma abordagem *user-to-user*, em que são analisadas as preferências de utilizadores com padrões de consumo semelhantes. Por exemplo, se o utilizador X e o utilizador Y viram os mesmos 5 filmes no último mês, serão recomendados a X os filmes que o utilizador Y viu que não foram já vistos por X[27].

A lógica do processo de filtragem colaborativa consiste geralmente em [28]:

- Analisar o consumo/avaliação de cada produto por cada utilizador (através de histórico de compras ou histórico de navegação);
- Calcular a similaridade entre todos os utilizadores baseado nos seus consumos/avaliações de produtos;
- Selecionar os *Top-N* utilizadores com maior similaridade ao utilizador-alvo;
- Recomendar os produtos com melhores avaliações/mais consumidos pelos *Top-N* utilizadores que o utilizador-alvo não tenha ainda adquirido.

Para o efeito, é utilizada uma matriz de correlação utilizador/produto, como exemplificado na Figura 2.



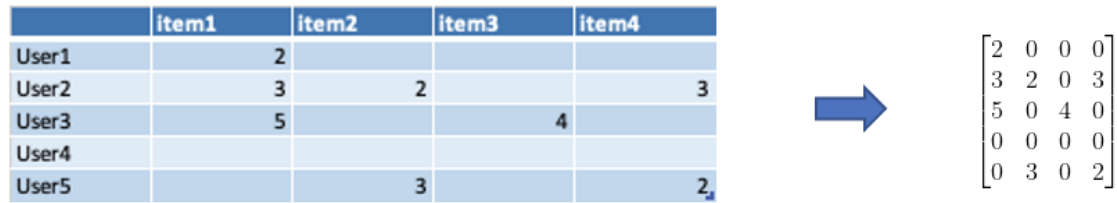


Figura 2 - Exemplo de matriz de correlação

Fonte: Medium[28]

Com a matriz como input, a similaridade pode ser calculada utilizando dois métodos matemáticos: a similaridade por cosseno e a correlação de Pearson[29].

Em termos de complexidade, note-se que os algoritmos de filtragem por conteúdo são relativamente simples e a sua escalabilidade depende apenas do crescimento de número de produtos existentes. Por contraste, os algoritmos de filtragem colaborativa escalam com o número de utilizadores, que excedem e crescem mais rapidamente que o número de produtos. Uma vez que implica também a utilização de matrizes e o cálculo de todos os pares possíveis de linhas, a complexidade é exponencialmente maior.

## 2.3 Trabalhos relacionados

A pesquisa e o desenvolvimento de sistemas de recomendação têm sido um campo de interesse na comunidade científica há mais de uma década, tendo produzido métodos comprovados para computação “consciente de preferências”[30].

Apesar disso, vários trabalhos têm contribuído para implementar sistemas de recomendação híbridos, que combinam métodos clássicos. Estes sistemas visam alavancar as vantagens da filtragem colaborativa e de conteúdo, evitando certas limitações, como o problema de *cold-start* (respeitante a novos utilizadores e novos itens) e o problema de esparsidade de dados.

Stream Rec[31] implementou um modelo de recomendação de filtragem colaborativa híbrido baseado num sistema de processamento em *streaming*. Utiliza uma matriz fatorizada em duas partes: correlação de utilizadores e correlação de itens. Aplica ainda um algoritmo de K-vizinhos mais próximos, de modo a comparar um utilizador a um grupo de utilizadores em vez de a cada outro utilizador individualmente.

No entanto, requer dados de feedback explícitos, ou seja, avaliações de itens, que geralmente não estão disponíveis em situações práticas.

TencentRec[32] propõe uma arquitetura generalista de recomendação para feedback implícito. Implementa uma série de algoritmos clássicos (filtragem de conteúdo, filtragem colaborativa, regra de associação e *click through rate* (CTR) situacional). A nível de arquitetura, utiliza Apache Storm e um sistema de comunicação *publish-subscribe*.

Lida, no entanto, com algumas limitações de automação no que toca ao paralelismo e *load-balancing*.

Fundamentado nestes dois trabalhos, o presente projeto explora também algoritmos de recomendação híbridos, combinando a filtragem de conteúdo (itens) e colaborativa (utilizadores) para colmatar o problema de *cold-start* e de esparsidade de dados. O fator de diferenciação do projeto, relativamente ao Stream Rec, passa por tentar contornar as limitações de feedback explícito através da conversão do comportamento implícito de utilizadores em fatores binários (p. ex. repetiu ou não a compra do produto X). Já em relação ao TencentRec, o projeto pretende introduzir melhorias no paralelismo e *load-balancing* através da utilização do Apache Spark a nível de arquitetura. A tecnologia Spark destaca-se pelo seu foco em computação distribuída e otimização para paralelização de tarefas. Na secção 2.4.1 são explorados os ganhos de desempenho desta tecnologia emergente, face ao Apache Storm, por exemplo.

Fora do âmbito de sistemas de recomendação, é possível encontrar aplicações de sistemas de análise de dados em tempo real em outras áreas, capazes de fornecer *insights* valiosas no presente projeto.

Na área de serviços de saúde, Ta e Liu[33] apresentam uma solução em tempo real com aplicação na previsão de propagação de doenças como a gripe. O grupo de investigadores concebeu uma arquitetura genérica utilizando ferramentas de código aberto, nomeadamente Apache Kafka (consumo de dados a partir de várias fontes, através de sistemas *publish-subscribe*), Apache Storm (computação) e NoSQL Cassandra (armazenamento).

Na área de Internet of Things (IoT), observa-se uma arquitetura semelhante, nomeadamente a sugerida por Malek et al[34], que desenvolveu uma plataforma para processamento em *streaming* de dados provenientes de vários sensores. A arquitetura consiste em Apache Kafka, para ingestão, Apache Storm para computação e NoSQL MongoDB para armazenamento. Note-se que esta aplicação é relevante em outras áreas que fazem utilização intensiva de sensores, como gestão de Energia (redes elétricas inteligentes) e implementação de Veículos Autónomos.

Por último, na área Financeira nota-se o uso de análise de dados tempo real em aplicações de detecção de fraude. Em [35] e [36] são exploradas arquiteturas que utilizam Apache Kafka para ingestão de dados, Hadoop MapReduce para computação e armazenamento em bases de dados SQL. A nível algorítmico, é também aplicado o algoritmo de K-vizinhos mais próximos para agrupar padrões de comportamento.

Estes exemplos de aplicações em áreas alternativas fundamentam as escolhas na arquitetura do presente projeto, particularmente com a utilização do sistema *publish-subscribe* Apache Kafka e o armazenamento de dados em formato não estruturado (NoSQL). Os trabalhos expostos demonstram a aplicação destas técnicas de análise de dados em tempo real, com resultados consistentes e bom desempenho.

É importante ressaltar que o presente projeto se destaca pela sua combinação específica de técnicas e tecnologias, adaptadas às necessidades de um sistema de recomendação em tempo real para *e-commerce*.

## **2.4 Tecnologias existentes**

Nesta secção irão ser apresentadas as ferramentas e linguagens de programação mais utilizadas atualmente em projetos de engenharia de dados em tempo real.

Durante a revisão literária para este tópico, constatou-se que existe uma *pool* de comparação e análise na comunidade científica de um conjunto de ferramentas *open-source* da Apache Foundation – principalmente Kafka, Flume, Storm e Hadoop –, com a maioria dos artigos publicados entre 2018 e 2020 [37]–[39].

No entanto, fontes corporativas/organizacionais indicam uma mudança na indústria nos últimos 3 anos, mantendo-se a utilização de algumas ferramentas, mas surgindo outras tecnologias Apache bem como tecnologias proprietárias da Amazon, Google e Microsoft[40], [41].

Desta forma, o estado da indústria apresentado é relativo ao período 2021 – 2023 e tem como base estudos de mercado de marcas na área[40], [41]. Também foi tida em consideração a própria experiência da DataCentric na escolha das tecnologias a selecionar para análise.

### **2.4.1 Ferramentas e frameworks**

Uma vez que o processo de engenharia de dados é composto por fases distintas, como exposto na secção 2.1, também as ferramentas utilizadas em cada fase são especializadas e

distintas. Por esta razão, apresentar-se-á de seguida uma análise separada pelos seguintes tópicos: Ingestão, Armazenamento e Processamento de dados, como resumido na Figura 3.

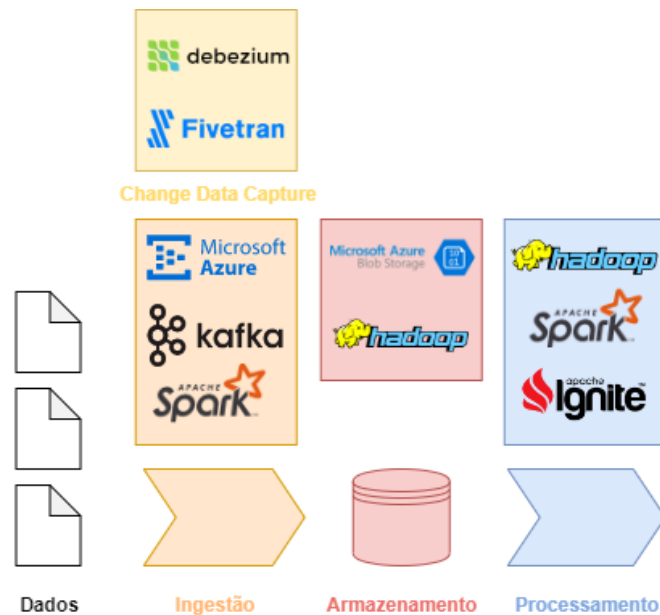


Figura 3 - Resumo das ferramentas analisadas e respetiva fase do processo

## Ingestão

A tecnologia utilizada na ingestão de dados no contexto do presente projeto divide-se em duas categorias: a deteção de alterações e a propagação dessas alterações.

A deteção de alterações é feita por meio de tecnologias de CDC, das quais se destacam Fivetran – a alternativa paga mais utilizada atualmente[40], [41] – e Debezium – a alternativa *open-source* mais utilizada[42]. Ambas as tecnologias são *log-based*, ou seja, recorrem à leitura de *binary logs*, tal como a maioria das restantes tecnologias no mercado. A principal diferença entre ambas prende-se com o número de fontes de dados suportadas, facilidade de uso e pelos custos de operação.

Fivetran é um software como serviço (SaaS) que permite às empresas mover dados isolados para armazenamento acessível, como armazéns de dados na *cloud*, sem criar *pipelines*[42]. Por ser uma plataforma *no-code*, é de fácil utilização. Disponibiliza uma lista extensa de conectores a fontes de dados, totalizando 318 à data de escrita do presente relatório[43].

Debezium é uma ferramenta *open-source* para CDC baseada no Apache Kafka. É por defeito um *software* local, de elevada configurabilidade. Possui atualmente 6 conectores, com 4 adicionais em fase de desenvolvimento[42].

Por comparação ao Fivetran, este último requer mais conhecimento técnico e documentação para implementar e exige a montagem de uma *pipeline* com ferramentas adicionais para alimentar e extrair *outputs* para posterior tratamento.

A nível de custos operacionais, sendo o Fivetran uma ferramenta em ambiente *cloud*, apenas são pagos os recursos efetivamente usados, ainda que a um preço definido pelo provedor e vantajoso para o mesmo. Já o Debezium é executado localmente, dependendo assim o seu custo dos recursos já existentes e da gestão de utilização dos mesmos pela empresa.

Relativamente à propagação de alterações, nota-se a utilização de ferramentas Apache – Kafka, Spark, Pulsar e Flink – bem como os serviços cloud Amazon Kinesis, Google Pub/Sub e Azure Event Hubs[40], [41]. Pela sua relevância para o presente projeto, foram selecionadas para análise as seguintes ferramentas:

- Apache Spark Streaming, um componente do projeto Spark concebido exclusivamente para paralelização automática. O Spark Streaming recebe dados em tempo real e divide os dados em *batches*, que são depois processados pelo *engine* Spark[44] (explorado em mais detalhe na p.23).
- Apache Kafka, um sistema de mensagens distribuído num ambiente *publish - subscribe*. A arquitetura do Kafka consiste em um ou mais produtores, que enviam mensagens para certos tópicos. Por sua vez os consumidores inscrevem um dado tópico. Cada tópico é dividido em partições e cada mensagem é identificada por um *key-value* dentro da partição. Os intermediários (*brokers*) alojam os tópicos, gerem as operações de leitura e escrita e são responsáveis pela replicação das partições entre si[45].
- Azure Event Hubs, uma plataforma de streaming de *big data* em ambiente *cloud*, oferecida pela Microsoft. A arquitetura do Event Hubs é semelhante à do Kafka, com a distinção de estar centralizada num único intermediário (*broker*), em oposição à lógica de intermediários *peer-to-peer* do Kafka[46].

Na Tabela 5 apresenta-se uma comparação das principais características dos três sistemas de ingestão.

Tabela 5 - Comparação de Apache Spark, Apache Kafka e Azure Event Hubs[46]–[51]

	Spark	Kakfa	Event Hubs
<b>Tipo de ingestão</b>	<i>Micro-batching</i>	<i>Event processing</i>	Suporta ambas
<b>Garantia de entrega<sup>3</sup></b>	<i>Exactly-once</i>	<i>Exactly-once</i>	<i>Atleast-once</i>
<b>Desempenho</b>	Sem valor de referência. Em geral, a latência é sub-minuto e consegue atingir valores mínimos de 1ms.	Latência: ~2ms Taxa de transmissão: ~30K msg/sec	Latência: ~10ms Taxa de transmissão: ~20K msg/sec
<b>Fontes de dados suportadas</b>	Ficheiros parquet, orc, json, csv, txt, avro e binários; tabelas Hive; suporta conexão JDBC a outras bases de dados; Kafka	ActiveMQ, AMPS, Azure, Data Diode, Datagen, FTPS, GitHub, Google Cloud, HDFS, IBM MQ, InfluxDB, JDBC, Jira, JMS, Kudu, Marketo, MQTT, Oracle CDC, RabbitMQ, Salesforce, ServiceNow, SFTP, SNMP, Splunk, Syslog, Terradata, TIBCO, Zendesk <sup>4</sup>	AMQP, Kafka
<b>Linguagens suportadas</b>	Java, Scala, R, Python	Java, Scala	Go, JavaScript, Python, Java, .NET,

O Kafka destaca-se como a solução mais flexível, pela grande quantidade de fontes de dados que disponibilizam conectores, e mais rápida, pela mais baixa latência de entre todas as alternativas. De destacar que as restantes ferramentas, no entanto, oferecem uma solução

<sup>3</sup> As metodologias utilizadas são *atleast-once* (garante a captação dos dados com possibilidade de duplicados), *atmost-once* (não admite duplicados mas pode não captar os dados em caso de falha) e *exactly-once* (capta todos os dados sem duplicação).

<sup>4</sup> Os conectores listados são oferecidos por terceiros e não vêm *bundled* com o software.

mais abrangente. O Spark é um sistema capaz não apenas de ingestão, mas também de processamento dos dados, eliminando potenciais *lags* que acontecem quando estas fases são feitas separadamente. O Event Hubs, por sua vez, tem a vantagem de ser um serviço de *cloud*, com utilização mais eficiente de recursos computacionais e consequentemente mais barato. Também oferece conexões fiáveis com outros produtos da Azure, mantidas pelo provedor.

### Armazenamento de dados

O armazenamento de dados é a categoria com maior diversidade de tecnologias utilizadas[41] e onde as diferenças de performance nativa são, em geral, pequenas. Por essa razão, selecionou-se para análise uma amostra de duas ferramentas populares e frequentemente em uso em projetos da DataCentric, abrangendo ambos os ambientes local e *cloud*:

- Hadoop Distributed File System (HDFS), um sistema *open-source* de armazenamento de dados que suporta até centenas de nós em cluster. Lida com dados estruturados e não estruturados e armazena grandes volumes (ou seja, os arquivos armazenados podem ser maiores que um terabyte). Apesar do HDFS ser desenhado para operações *batch* de alta latência, surgiu em 2008 o projeto HBase, construído com o HDFS como base, que oferece uma base de dados não relacional concebida para operações de baixa latência[52].
- Azure Blob Storage, uma solução em ambiente *cloud* fornecida pela Microsoft para armazenamento de grandes volumes de objetos não estruturados. Um *blob* pode consistir em dados binários ou qualquer ficheiro de texto[53]. Tal como no caso acima, foi criado um projeto – Azure Data Lake Storage Gen 2- construído com o Blob Storage como base, que oferece um *data lake* concebido para consultas de alto desempenho[54].

Segue na Tabela 6 um resumo comparativo de ambas as tecnologias.

Tabela 6 - Comparação de Apache Hadoop HBase e Azure Data Lake Storage Gen 2 [55], [56]

	HBase	Data Lake Storage
<b>Tipo de dados</b>	Estruturado e não estruturado	Não estruturado
<b>Tipo de armazenamento</b>	<i>Data lake</i>	<i>Data lake</i>

	HBase	Data Lake Storage
<b>Desempenho de pesquisa<sup>5</sup></b>	Pequenos sets de dados: ~1.5s	Pequenos sets de dados: ~35.7s
	Grandes sets de dados (GBs): ~41.9s	Grandes sets de dados (GBs): ~42.0s
<b>Custo</b>	--	3x mais barato do que executar HDFS na Azure <i>cloud</i>
<b>Linguagens suportadas</b>	Shell, REST, Java, Avro, Thrift	Shell, REST, SFTP, .NET, Java, Node.js, Python, Go

As principais diferenças entre HBase e Data Lake Storage são assim o custo e velocidade de uma solução *on-premise* contra uma solução *cloud*. Apesar de o armazenamento HBase ser objetivamente mais rápido, a solução *cloud* da Azure consegue lidar com grandes volumes de dados com uma performance muito semelhante. Beneficia ainda de custos mais baixos, pelo pagamento apenas do que é consumido. Um estudo pelo Enterprise Research Group (ESG) indica inclusive que empresas que migraram de um sistema HDFS local para armazenamento *cloud* tipicamente têm um custo total 57% mais baixo[57].

## Processamento

As tecnologias utilizadas no processamento de dados em tempo real têm estado sujeitas a uma maior variação nos últimos anos, comparativamente com outras fases do processo. Enquanto que na comunidade científica o Apache Storm tem sido a alternativa mais adotada[37]–[39], na indústria notou-se a popularização do Hadoop, agora já em fase de declínio, e o surgimento do Apache Spark como o novo *standard*[40], [41]. De modo a obter uma boa visão temporal, selecionaram-se estas tecnologias, bem como uma mais recente e com potencial de adoção futura (solicitada pela empresa acolhedora do estágio):

- Hadoop MapReduce, um dos primeiros e mais conhecidos *frameworks* de processamento em cluster. MapReduce segue um modelo de programação funcional e realiza sincronização explícita entre estados computacionais. A *framework* expõe uma API de programação simples, com as funções *map()* e *reduce()*[58].

<sup>5</sup> Dados obtidos em clusters de teste com 3 worker nodes de 8 cores, 56GB de RAM e 512GB de disco[56].



- Apache Spark, também um *framework* de computação em cluster para conjuntos grandes de dados. O fundamento principal da *framework* são os conjuntos de dados distribuídos resilientes (RDDs, doravante referidos apenas como *dataframes*) que implementam estruturas de dados em memória, usadas para armazenar dados intermediários na *cache* num conjunto de nós. Como os RDDs podem ser mantidos em memória, os algoritmos podem iterar sobre os dados RDD muitas vezes com muita eficiência[58].
- Apache Ignite, um DBMS distribuído para computação de alta performance, com capacidade de armazenamento e processamento de dados em *cache* numa rede de *clusters* interligados. O Ignite segue uma lógica *server-client*, em que os servidores são unidades computacionais e de armazenamento enquanto os clientes são pontos de ligação à base de dados partilhada pelo *cluster* de nós de servidores[59].

Tabela 7 - Comparação de Hadoop MapReduce, Apache Spark e Apache Ignite

	MapReduce	Spark	Ignite
<b>Tipo de computação</b>	Computação <i>disk-based</i>	Computação <i>RAM-based</i>	Computação <i>RAM-based</i>
<b>Tolerância a falha</b>	Replicação	Resilient Distributed Datasets (RDD)	Transferência automática de <i>jobs</i> entre nós
<b>Velocidade<sup>6</sup></b>	MapReduce é ~1.5x mais lento que Spark	--	Ignite é 1.5x a 3x mais lento que Spark
<b>Linguagens suportadas</b>	Java, Ruby, Python, C++	Java, Scala, Python, R	Java, .NET/C#, C++

Como se constata na Tabela 7, o Spark destaca-se como sendo a ferramenta de processamento mais rápida e com boa fiabilidade devido ao uso de RDDs. No entanto, é discutida a possibilidade de potenciar ainda mais a velocidade do Spark integrando-a com o Ignite, como tecnologia semelhante[60]. A solução combinada passa por utilizar Ignite como auxiliar ao Spark para compartilhar estados diretamente em memória, sem precisar de armazená-los no disco.

<sup>6</sup> Tempo de execução de um algoritmo K-Means com tamanho variável do ficheiro de input[58], [60].

### 2.4.2 Linguagens de programação

Em geral, as aplicações modernas de engenharia de dados em tempo real requerem linguagem de programação com:

- Capacidades de *multi-threading* e processamento em paralelo;
- Capacidade de integrar com *frameworks* existentes de pré e pós-processamento, comunicação e de sistema.

No universo de ferramentas de análise de dados são tipicamente suportadas três principais linguagens: Java, Python e R[61].

Estas são consideradas linguagens de produtividade[61], pela abstração que oferecem em relação aos detalhes complexos de algoritmos de processamento distribuído.

Java é uma linguagem de programação de propósito geral que oferece suporte a programação orientada a objetos. É utilizada para o desenvolvimento de aplicações em contexto empresarial e possui uma grande comunidade de desenvolvedores[62].

Python é uma linguagem de programação interpretada de alto nível conhecida pela sua simplicidade e legibilidade. É utilizada em ciência de dados e análise de dados devido à sua rica coleção de bibliotecas e facilidade de uso[62].

R é uma linguagem de programação e ambiente de software usado principalmente para análise estatística e visualização de dados. É adotada na comunidade acadêmica devido ao conjunto de poderosas funções integradas que oferece[62].

De entre as três linguagens, destaca-se o Python, como a linguagem mais popular segundo o Índice TIOBE[63] e PYPL[64], sendo capaz de cumprir com os requisitos enumerados acima.

Embora o Java também seja uma linguagem amplamente utilizada na área de engenharia de dados, tende a aumentar a complexidade de desenvolvimento. A necessidade de maior quantidade de código para realizar tarefas semelhantes e a curva de aprendizagem mais acentuada pode ter impacto negativo no tempo de desenvolvimento. De notar, no entanto, que é dotada de capacidade de processamento superior, com *threads* e suporte a concorrência[62].

Por sua vez, o R apesar de ser uma linguagem popular para análise estatística, pode não ser tão eficiente quanto o Python para tarefas de processamento em tempo real e integração com *frameworks* específicos. O R oferece uma extensa coleção de pacotes estatísticos que facilitam a análise avançada, no entanto é menos adequada para processamento em tempo

real devido à falta de suporte nativo para processamento distribuído e acentuada curva de aprendizagem[62].

Python diferencia-se uma vez que consegue executar qualquer tarefa na área de engenharia de dados, graças ao grande número de bibliotecas disponíveis, entre as quais o NumPy (funções matemáticas), pandas (manipulação de bases de dados) e Matplotlib (visualização de dados). Python também apresenta uma boa integração com as principais *frameworks* utilizadas atualmente em engenharia de dados, sob a forma de PySpark e PyKafka. Apesar de ter um desempenho relativamente inferior face a linguagens compiladas e otimizadas para escalabilidade, como Java, apresenta um bom equilíbrio em termos de desenvolvimento e manutenção do código.

De notar que para a implementação do presente projeto – na *framework* Apache Spark – estão consideradas duas linguagens de programação. Foi selecionado Python como linguagem de desenvolvimento, pelos motivos já explicados, no entanto está pressuposta a utilização de Java como código-fonte da *framework* em si. Tal permite aproveitar as vantagens de paralelismo do Java, embutidas no Apache Spark, e as vantagens de fácil desenvolvimento com Python ao longo de toda a implementação da lógica de tratamento de dados.

## 3 Análise e desenho da solução

O desenho da solução implementada preocupou-se em responder às necessidades das partes envolvidas e acrescentar valor ao negócio. Para esse fim, foi conduzida uma seleção e análise do domínio abordar, entendendo os conceitos e relações existentes, bem como recolhidos um conjunto de requisitos pretendidos. Foi também feita a recolha de informação acerca dos *stakeholders* e dos utilizadores finais da solução.

Relembra-se, nesta fase, que o presente projeto visa conceber uma arquitetura que responda à exigência de recursos computacionais e de tempos de processamento para permitir a análise de dados em tempo real. O desenvolvimento desta solução irá ter especial foco em requisitos não funcionais, como confiabilidade, desempenho e restrições físicas.

Os modelos utilizados para descrever os requisitos recolhidos no processo de desenho da solução serão casos de uso e *user stories*, para os requisitos funcionais, e FURPS+, para os requisitos não funcionais.

### 3.1 Partes envolvidas do negócio

A primeira fase da análise do problema e conceção da solução foi conduzida em diálogo com o cliente, tendo-se desde logo estabelecido os *stakeholders* (partes interessadas), os utilizadores da solução e o público-alvo para futuras integrações.

Assim, a solução foi desenhada tendo em conta os seguintes atores:

- Gestor de projeto – pessoa associada à entidade que acolhe o desenvolvimento do presente projeto (DataCentric). Consiste no principal avaliador da solução. Pretende que o sistema seja capaz de demonstrar as capacidades do processamento em tempo real a futuros clientes. Pretende ainda a qualidade de documentação de análise e desenvolvimento para uso futuro.
- Engenheiro de dados – pessoa responsável pela gestão dos processos de ingestão e tratamento de dados. Consiste no utilizador final da solução, desempenhando a atividade de monitorizar o fluxo de dados. Pretende que o sistema seja de fácil manutenção.
- Retalhista – entidade que comercializa produtos de retalho. Não interage com o sistema, no entanto é considerado no desenho da solução a nível de futura integração. Pretende que o *output* do sistema possa ser consultado pelos seus analistas de negócio ou ser integrado na sua plataforma de *e-commerce*.

- Consumidor – pessoa que adquire produtos a retalho através da plataforma do retalhista. Não interage com o sistema, no entanto é considerado no desenho da solução a nível de futura integração. Pretende que o *output* do sistema seja disponibilizado de forma rápida e que produza resultados relevantes.

### 3.2 Domínio do problema

O problema de análise de dados em tempo real pode existir em vários domínios. Sendo o projeto uma prova de conceito para a mencionada infraestrutura, verificou-se alguma liberdade na seleção do domínio.

Considerou-se, assim, abordar o domínio de *e-commerce*, utilizando para o efeito um *dataset* público de uma plataforma de venda a retalho online brasileira - referida ao longo deste relatório como “*olist*” - como fonte de dados.

A escolha deste domínio em específico deveu-se à potencial relevância do contexto de *e-commerce* para os clientes da DataCentric, visto que se nota uma maior aposta destas partes em plataformas de *marketplace* onde vários vendedores põem à disposição artigos com a plataforma como intermediário.

De notar que não foram utilizados dados reais da empresa acolhedora do estágio devido a restrições de acessos e permissões que significativamente impactariam o tempo de desenvolvimento do projeto. Também se decidiu evitar fontes de dados excessivamente complexas e/ou com grande número de particularidades, como é o caso dos dados destas empresas, de modo a focar a maior parte do tempo de desenvolvimento na arquitetura ao invés da pré-análise e pré-processamento de dados.

Os conceitos principais a abordar no domínio de *e-commerce* serão Cliente, Vendedor, Encomenda e Produto, os quais são detalhados no modelo de domínio presente na Figura 4.

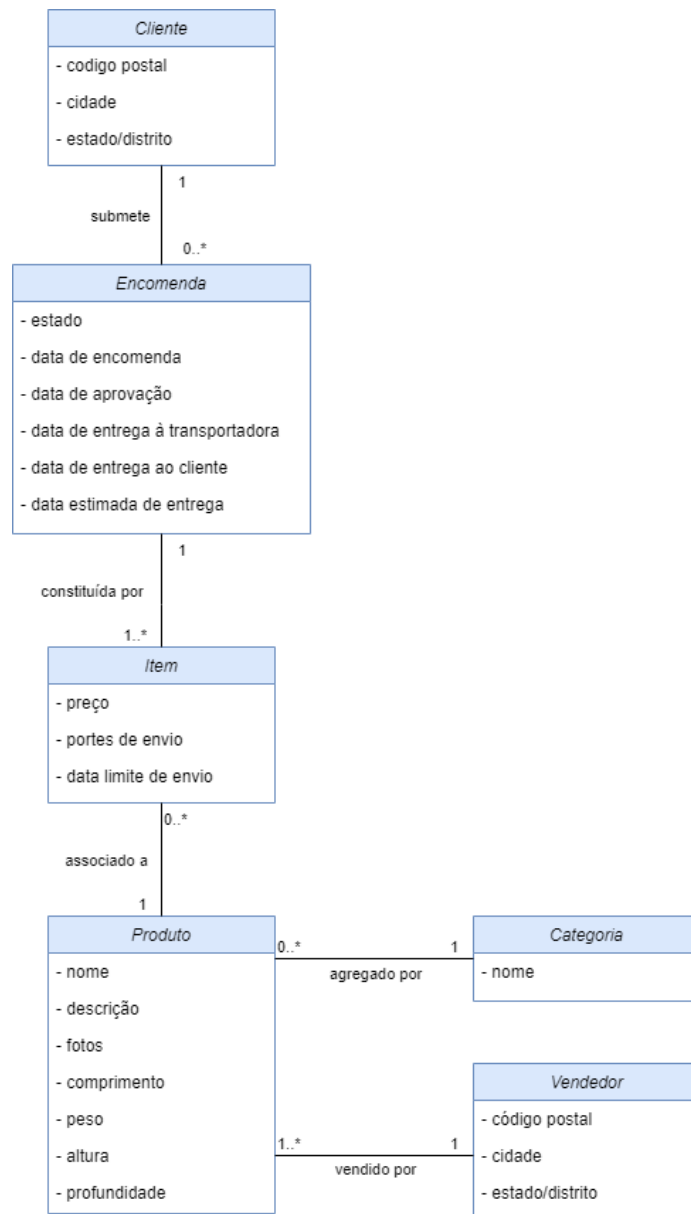


Figura 4 - Modelo de domínio

Estão ainda representados alguns conceitos secundários, tais como Item e Categoria, que representam especificidades para o negócio.

Para efeitos da solução a desenvolver, os conceitos de Cliente e Produto serão os pontos centrais, uma vez que serão geradas recomendações, sob a forma de produtos relevantes para cada cliente.

Para base estatística, é relevante o conceito Encomenda e o conceito secundário Item, para determinar as preferências e as tendências de acordo com a frequência de compras efetuadas.

Finalmente, acrescenta-se o conceito de Categoria e Vendedor, como fatores de agregação das recomendações.

### 3.3 Requisitos funcionais

Um requisito funcional descreve as funções do *software*, isto é, as tarefas que efetivamente executa. Uma função é descrita como um conjunto de entradas, seu comportamento e as saídas[65].

Para a análise destes requisitos, utilizou-se o modelo de casos de uso para descrever a forma como os utilizadores interagem com o sistema para concretizar um dado objetivo[66].

De seguida, irão ser descritos individualmente os casos de usos recolhidos em reuniões com o cliente. No âmbito deste projeto, escolheu-se associar uma *user story* a cada caso de uso, cada uma incluindo a sua descrição, critérios de aceitação e dependências. É ainda incluído um diagrama de casos de uso no final desta análise, com o objetivo de resumir a lista de requisitos e fornecer uma visão global.

Tabela 8 - Descrição do caso de uso/user story DCRA-1

UC/US DCRA-1 – Analisar o estado da arte e selecionar artefactos
<b>Descrição do caso de uso</b>
O gestor de projeto reúne e apresenta ao cliente a base tecnológica do projeto e a fonte de dados para validação.
<b>Descrição da <i>user story</i></b>
Como gestor de projeto, pretendo ter uma análise dos algoritmos de recomendação a implementar e das tecnologias de tempo real a utilizar. Pretende-se ainda a seleção de um conjunto de dados de <i>e-commerce</i> a utilizar como fonte.
<b>Critérios de aceitação</b>
AC1: O <i>dataset</i> deverá ter informação acerca das vendas com granularidade por <i>client_id</i> e <i>product_id</i> .
AC2: O <i>dataset</i> deverá ter informação acerca das categorias a que corresponde cada <i>product_id</i> .
AC3: O <i>dataset</i> deverá ter informação acerca das <i>reviews</i> com granularidade ( <i>client_id</i> , <i>product_id</i> ).

**UC/US DCRA-1 – Analisar o estado da arte e selecionar artefactos**

AC4: O *dataset* deverá ter informação temporal acerca de cada venda (data e hora).

AC5: Devem ser documentadas pelos menos 2 alternativas de algoritmo de recomendação, com complexidades distintas.

AC6: O algoritmo de recomendação deverá ser implementável apenas com código SQL, sem recurso a técnicas de *machine learning*.

AC7: A tecnologia de tempo real deve ser implementável em ambiente *cloud*.

Tabela 9 - Descrição do caso de uso/user story DCRA-2

<b>UC/US DCRA-2 – Modelar a arquitetura e o esquema de base de dados</b>
<b>Descrição do caso de uso</b>
O gestor de projeto reúne e apresenta ao cliente a documentação de análise do projeto e dos dados a ser processados pelo sistema.
<b>Descrição da user story</b>
Como gestor de projeto, pretendo que sejam elaborados modelos da estrutura do projeto para acelerar o desenvolvimento de casos de uso futuros.
<b>Critérios de aceitação</b>
AC1: Deve ser considerado o domínio do problema, através de um modelo de domínio.
AC2: Devem ser consideradas várias fases de tratamento de dados, incluindo pelo menos os estados <i>raw</i> , <i>transformed</i> e <i>processed</i> . Para cada estado deve ser documentado um modelo de base de dados.
AC3: O modelo de base de dados <i>processed</i> deve estar otimizado para executar consultas sobre os itens de encomenda (tabela <i>order items</i> ).
AC3: Deve ser elaborado um diagrama de arquitetura que descreva as tecnologias a utilizar, com a sua sequência e respetivas dependências.
AC4: Deve ser elaborado um modelo analítico de cálculo de tempos de execução ao longo do processo.
<b>Dependências</b>



**UC/US DCRA-2 – Modelar a arquitetura e o esquema de base de dados**

UC DCRA-1:

- Os modelos de base de dados a elaborar baseiam-se na estrutura de tabelas e colunas do *dataset* selecionado.

Tabela 10 - Descrição do caso de uso/user story DCRA-3

UC/US DCRA-3 – Captar eventos operacionais em tempo real
Descrição do caso de uso
O engenheiro de dados insere, altera ou elimina um registo na base de dados. O sistema capta em tempo real e apresenta temporariamente esse registo.
Descrição da <i>user story</i>
Como engenheiro de dados, pretendo visualizar, em tempo real, a transferência dos dados que são gerados localmente para um ambiente <i>cloud</i> .
Critérios de aceitação
AC1: Deve ser criada uma base de dados local “olist-dataset” e inserido o conteúdo do dataset selecionado.
AC2: Deve ser criada uma base de dados local “olist” com a mesma estrutura da “olist-dataset”.
AC3: Deve ser criado um script para inserir registos da base de dados “olist-dataset” na base de dados “olist”, de modo a simular a criação de linhas em tempo real.
AC4: Deve ser criada uma instância na <i>cloud</i> com subscrição a serviço de armazenamento em tabela (Data Lake), serviço de processamento de dados (Databricks) e serviço de ingestão por eventos (EventHubs).
AC5: Deve ser criada uma instância Kafka para gerir a escrita e leitura de eventos provenientes da base de dados.
AC6: Deve ser criado um conector Debezium à base de dados “olist”, capaz de gerar eventos Kafka através de CDC.

UC/US DCRA-3 – Captar eventos operacionais em tempo real
AC6: Deve ser criado um notebook Databricks para cada tabela, que consiga captar os eventos Kafka gerados pelo CDC, com diferenciação entre operações INSERT, UPDATE e DELETE.
Dados de entrada e saída
<b>Entrada:</b> Inserção de linha numa tabela da base de dados local “olist” <b>Saída:</b> Inserção da mesma linha num dataframe Spark em execução na cloud
Dependências
UC DCRA-1: <ul style="list-style-type: none"> <li>O conteúdo a ser inserido na base de dados “olist-dataset” corresponde ao dataset selecionado.</li> </ul>

Tabela 11 - Descrição do caso de uso/user story DCRA-4

UC/US DCRA-4 – Armazenar os eventos em armazém de dados
Descrição do caso de uso
O engenheiro de dados insere, altera ou elimina um registo na base de dados. O sistema guarda em tempo real esse registo para consultas futuras.
Descrição da <i>user story</i>
Como engenheiro de dados, pretendo que os dados recebidos na <i>cloud</i> sejam persistidos em bases de dados de acordo com os modelos desenvolvidos em DCRA-2.
CrITÉRIOS de aceitação
AC1: Deve ser criado um notebook Databricks para a persistência de cada tabela, com as transformações necessárias para a conformidade com os modelos de base de dados. AC2: Deve ser feita a validação de tipo e exclusividade de cada coluna.
Dados de entrada e saída
<b>Entrada:</b> Inserção de linha numa tabela da base de dados local “olist” <b>Saída:</b> Inserção da mesma linha numa tabela de Data Lake Storage

UC/US DCRA-4 – Armazenar os eventos em armazém de dados
Dependências
<p>UC DCRA-2:</p> <ul style="list-style-type: none"> <li>A estrutura das tabelas de Data Lake Storage corresponde aos modelos de dados definidos.</li> </ul> <p>UC DCRA-3:</p> <ul style="list-style-type: none"> <li>O conteúdo a ser inserido nas tabelas de Data Lake Storage corresponde ao fluxo de dados captado em tempo real.</li> </ul>

Tabela 12 - Descrição do caso de uso/user story DCRA-5

UC/US DCRA-5 – Desenvolver sistema de recomendação
Descrição do caso de uso
O sistema gera em tempo real recomendações com os dados de cada utilizador. O engenheiro de dados consulta a tabela de recomendações para validação.
Descrição da <i>user story</i>
Como engenheiro de dados, pretendo que o sistema gere e persista tabelas de recomendação para cada utilizador.
Critérios de aceitação
AC1: Deve ser criada uma tabela intermédia para o algoritmo de filtragem de conteúdo.
AC2: Deve ser criada uma tabela intermédia para o algoritmo de filtragem colaborativa.
Dados de entrada e saída
<p><b>Entrada:</b> Tabelas do <i>schema processed</i></p> <p><b>Saída:</b> Tabelas de recomendação</p>
Dependências
<p>DCRA-4:</p> <ul style="list-style-type: none"> <li>A fonte das tabelas de recomendação consiste nas tabelas armazenadas em Data Lake Storage.</li> </ul>

Na Figura 5, apresenta-se um diagrama resumido dos casos de uso mencionados e dos respetivos atores.

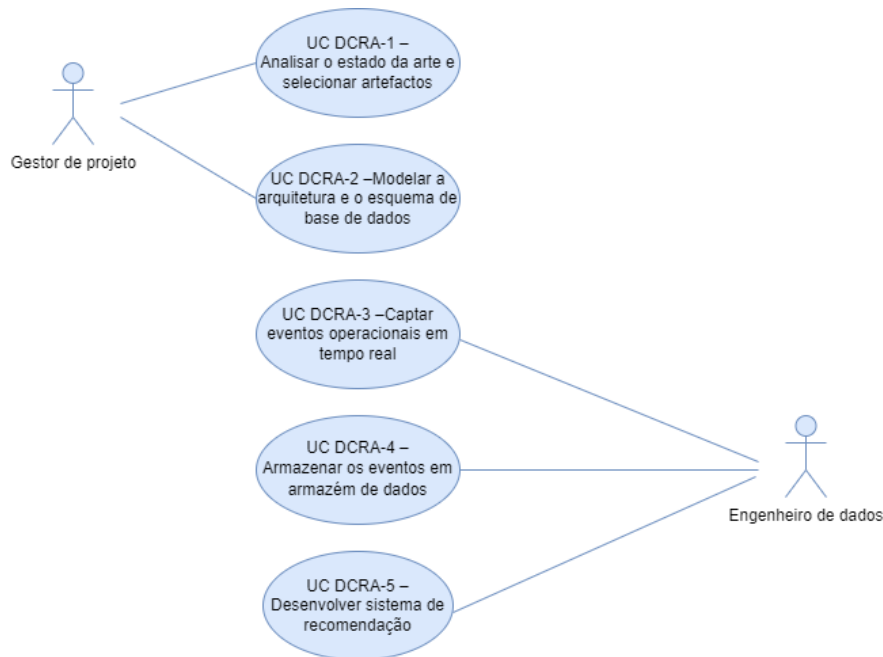


Figura 5 - Diagrama de casos de uso

### 3.4 Requisitos não funcionais

A análise de requisitos não funcionais passa pelo conjunto de exigências que o *software* deve cumprir e que não são consideradas funcionalidades. Tal como referido na secção 3, a velocidade de execução é um ponto crucial de um sistema em tempo real e será o principal requisito analisado e testado.

Para descrever estes requisitos, foi escolhido o modelo FURPS+, de modo a claramente categorizá-los. Este é um modelo de qualidade de software definido por 5 características: Funcionalidade, Usabilidade, Confiabilidade, Desempenho e Suporte. Acrescenta-se ainda a vertente '+', que representa outros aspetos como implementação, interface, operações, empacotamento e legal[67].

Não serão exploradas todas as categorias do modelo, uma vez que, por uma questão de síntese e relevância, se pretende abordar os critérios mais críticos para o sucesso do projeto. Por exemplo, a categoria de Usabilidade não será focada uma vez que o desenvolvimento do sistema não inclui um *front end*, nem se pretende que seja utilizado por um cliente final, mas por equipas técnicas de engenharia de dados. Não quer isto dizer que a Usabilidade será negligenciada, mas que será desenvolvida a um nível básico.

Assim sendo, é possível categorizar os requisitos mais importantes da seguinte forma:

Tabela 13 - Requisitos não funcionais

Categoria	Sub-categoria	Descrição
Funcionalidade	Auditoria	O processo deve captar dados adicionais de execução para efeitos de análise. Devem ser registados carimbos data/hora no final da execução de uma dada fase do processo (criação, ingestão, transformação e processamento dos dados).
	Reporte	Deve ser possível gerar gráficos demonstrativos dos tempos de execução e latência do processo.
	Atualização	O sistema deve ser capaz de atualizar as recomendações automaticamente e em tempo real.
Confiabilidade	Tolerância a falhas	O processo deve usar uma lógica de <i>checkpointing</i> para recuperar de falhas na fase de ingestão (leitura do tópico Kafka) e de persistência (escrita para as tabelas em <i>cloud</i> ).
	Concorrência	O sistema deve suportar consultas e manipulações concorrentes de dados.
	Replicação	O sistema deve manter registo de todo o histórico de dados consumidos antes de qualquer processamento (em estado <i>raw</i> ) para garantir integridade em caso de falhas ou erros nos processos subsequentes.
Desempenho	Tempo real	A latência de uma consulta ao sistema de recomendação deve ser de 1 segundo ou menos[68]. Já a latência total do processo deve ser de 5 segundos ou menos.

### 3.5 Desenho

Tendo por base a análise do problema efetuada no capítulo anterior, será apresentado neste capítulo o desenho da solução implementada neste projeto.

Em primeiro lugar, é apresentado o desenho da arquitetura global, começando com uma visão geral e progressivamente detalhando a solução. De seguida, serão justificadas duas

arquiteturas alternativas que não foram implementadas neste projeto, mas que foram também consideradas.

### 3.5.1 Arquitetura global

A Figura 6 fornece uma visão geral da arquitetura do sistema, representada por um diagrama de fluxo.

Optou-se por utilizar um diagrama de fluxo em vez do modelo tradicional C4 devido ao uso comum deste primeiro em arquiteturas de dados, presente particularmente em arquiteturas promovidas por líderes da indústria como a AWS e Azure. A decisão de usar um diagrama de fluxo permite também uma comparação mais fácil com arquiteturas alternativas que serão discutidas mais adiante nesta secção.

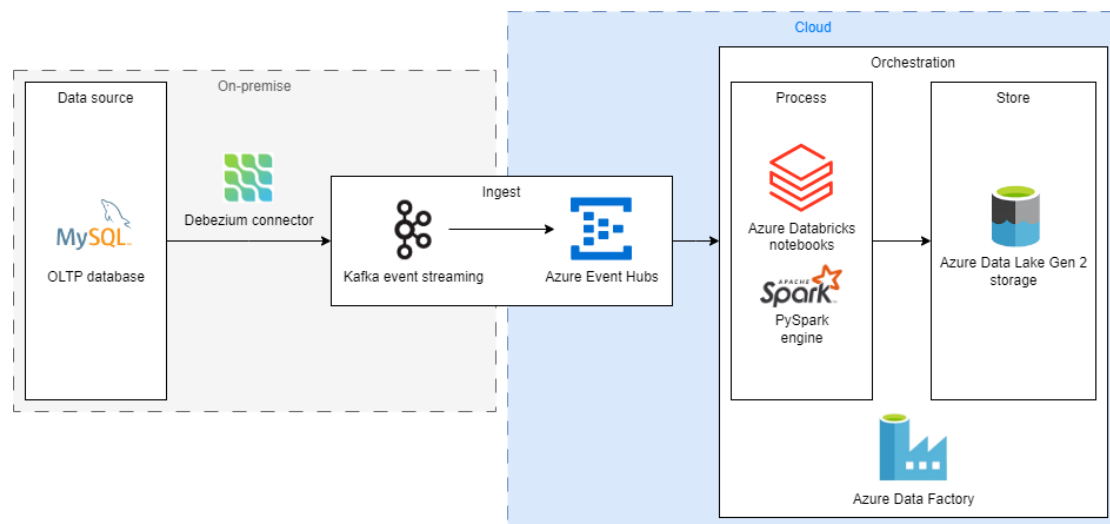


Figura 6 - Arquitetura global do projeto

Como é possível observar, a arquitetura implementada consiste numa componente local e numa componente *cloud*.

Na componente local, os dados fonte são armazenados numa base de dados MySQL. Dois *schemas* são considerados: um para a disponibilização do *dataset* completo e outro para a simulação da criação de registos em tempo real, através da cópia de linhas do *dataset* completo em intervalos agendados ou por input manual.

A base de dados local é configurada com um conector de CDC Debezium que expõe as alterações captadas nos *binary logs*, sob a forma de eventos, ao Kafka. O Kafka por sua vez armazena o *changelog* e providencia um ponto de ligação à *cloud*.

Na componente *cloud*, o Event Hubs consome o *stream* proveniente de Kafka e fornece uma interface para a conexão do motor de processamento Spark, executado em *notebooks* Databricks.

A função destes *notebooks* é de ler os dados provenientes de cada tópico, fazer o processamento necessário e armazenar em bases de dados na *cloud*. Na Figura 7, encontra-se mais detalhada esta interação.

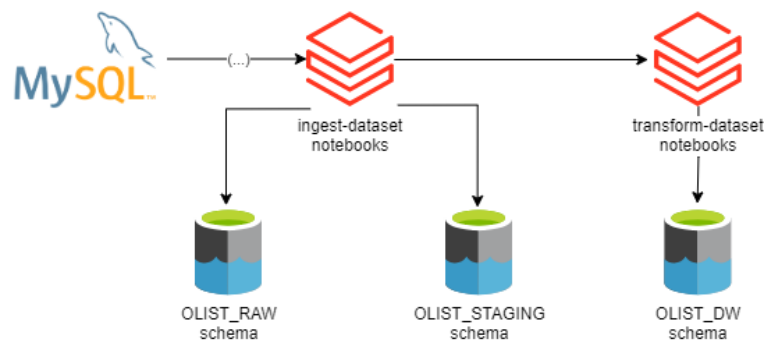


Figura 7 - Arquitetura geral de processamento e armazenamento

Os *notebooks* de ingestão (*ingest-dataset-notebooks*) consomem os tópicos relacionados com cada tabela da fonte e persistem os dados recebidos.

A persistência de dados segue a lógica da arquitetura de medalhão, um padrão de desenho de dados cujo objetivo é melhorar incrementalmente e progressivamente a estrutura e qualidade dos dados, à medida que atravessa cada camada da arquitetura[69]. As camadas são denominadas *bronze*, *silver* e *gold* e correspondem a:

- Bronze – dados “raw”, tal como são integrados a partir da fonte;
- Silver – dados filtrados, sujeitos a validações, formatações e limpeza;
- Gold – dados agregados de acordo com as necessidades de negócio.

Assim, numa primeira fase a persistência é feita em formato *raw*, apenas com o processamento necessário para extrair as colunas, e de seguida em formato *staging* (*silver*), que segue uma lógica de preparação dos dados para a fase de transformação.

Os *notebooks* de transformação (*transform-dataset-notebooks*) partem do esquema de dados em *staging* e conduzem as agregações e filtrações necessárias para chegar ao formato *data warehouse* (*gold*).

De notar que esta é uma representação simplificada do processo, sendo que o sistema está desenhado com a consideração de um *notebook* de ingestão e transformação para cada

tabela, bem como a existência de várias tabelas em cada *schema*. Esta complexidade está explicita na Figura 8.

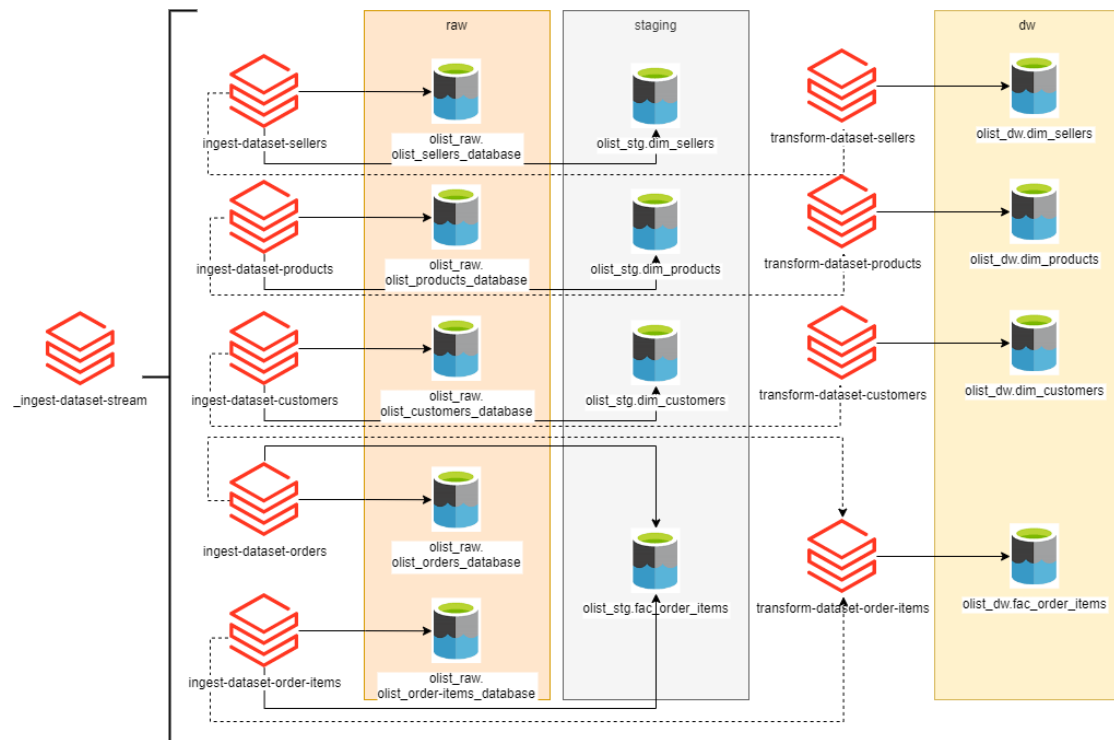


Figura 8 - Arquitetura detalhada de processamento e armazenamento

### Modelo de dados

Em relação ao armazenamento, foi ainda definido um modelo de dados para cada uma das bases de dados e *schemas* utilizados. Todos os modelos criados tiveram por base a estrutura pré-existente do *dataset* selecionado como fonte. Este modelo de dados consiste num modelo relacional, que está representado na Figura 9.





Figura 9 - Modelo de dados da estrutura local

O modelo é constituído por 5 tabelas, com informação acerca de clientes, encomendas, itens de encomenda, vendedores e produtos.

De notar que o requisito funcional DCRA-1 listado na secção 3.3 inclui o seguinte critério de aceitação: “O *dataset* deverá ter informação acerca das *reviews* com granularidade (client\_id, product\_id)”, que não está representado no modelo. Isto acontece uma vez que o *dataset* selecionado, apesar de ter uma tabela com informação acerca de *reviews*, não apresentava um sistema de classificação suficientemente consistente, sendo por isso a tabela removida de análise. Em alternativa, as *reviews* a considerar para o desenvolvimento do sistema de recomendação serão binárias: 1 caso o cliente tenha comprado o produto, 0 caso contrário.

Com base neste modelo, escolheu-se desenvolver os modelos de persistência com base também na arquitetura de medalhão, em que as camadas foram nomeadas “olist\_raw”, “olist\_stg” e “olist\_dw”.

### Camada “olist\_raw”

Para o esquema *raw*, o modelo considera os atributos dos dados tal como são formatados pelas ferramentas de CDC e *publish-subscribe*. Este esquema é também relacional e todas as tabelas possuem exatamente os mesmos atributos, destacando-se a colunas:

- *schema* – a estrutura dos dados capturados pelo CDC;
- *before* – o *snapshot* dos dados antes da alteração capturada pelo CDC;
- *after* – o *snapshot* dos dados após a alteração capturada pelo CDC;
- *source\_ts\_ms* – o carimbo data/hora em que se deu a alteração na base de dados local;
- *connector\_ts\_ms* – o carimbo data/hora em que o evento deu entrada no conector local CDC;
- *enqueued\_time* – o carimbo data/hora em que o evento deu entrada no conector *cloud*;
- *offset* – identificador único e sequencial atribuído pelo Kafka para cada mensagem que é publicada (evento).

### Camada “olist\_stg”

O modelo de dados da fase de *staging*, representado na **Error! Reference source not found.**, conta já com a extração dos dados de negócio codificados nos campos *schema*, *before* e *after* do modelo anterior.

O desenho deste modelo pretende servir como transição entre o esquema relacional dos modelos anteriores e o esquema em estrela que se pretende aplicar ao modelo final *dw*.

O esquema em estrela consiste numa tabela central, denominada tabela factual e num conjunto de tabelas denominadas dimensões. A tabela factual contém as medidas a ser agregadas para análise, bem como as chaves estrangeiras que fazem a ligação a cada dimensão, enquanto as dimensões contêm dados qualitativos auxiliares para filtragem[70].

Assim, as tabelas no modelo de *staging* já recebem a denominação de dimensão (dim) ou factual (fac) de acordo com o seu papel na análise pretendida.

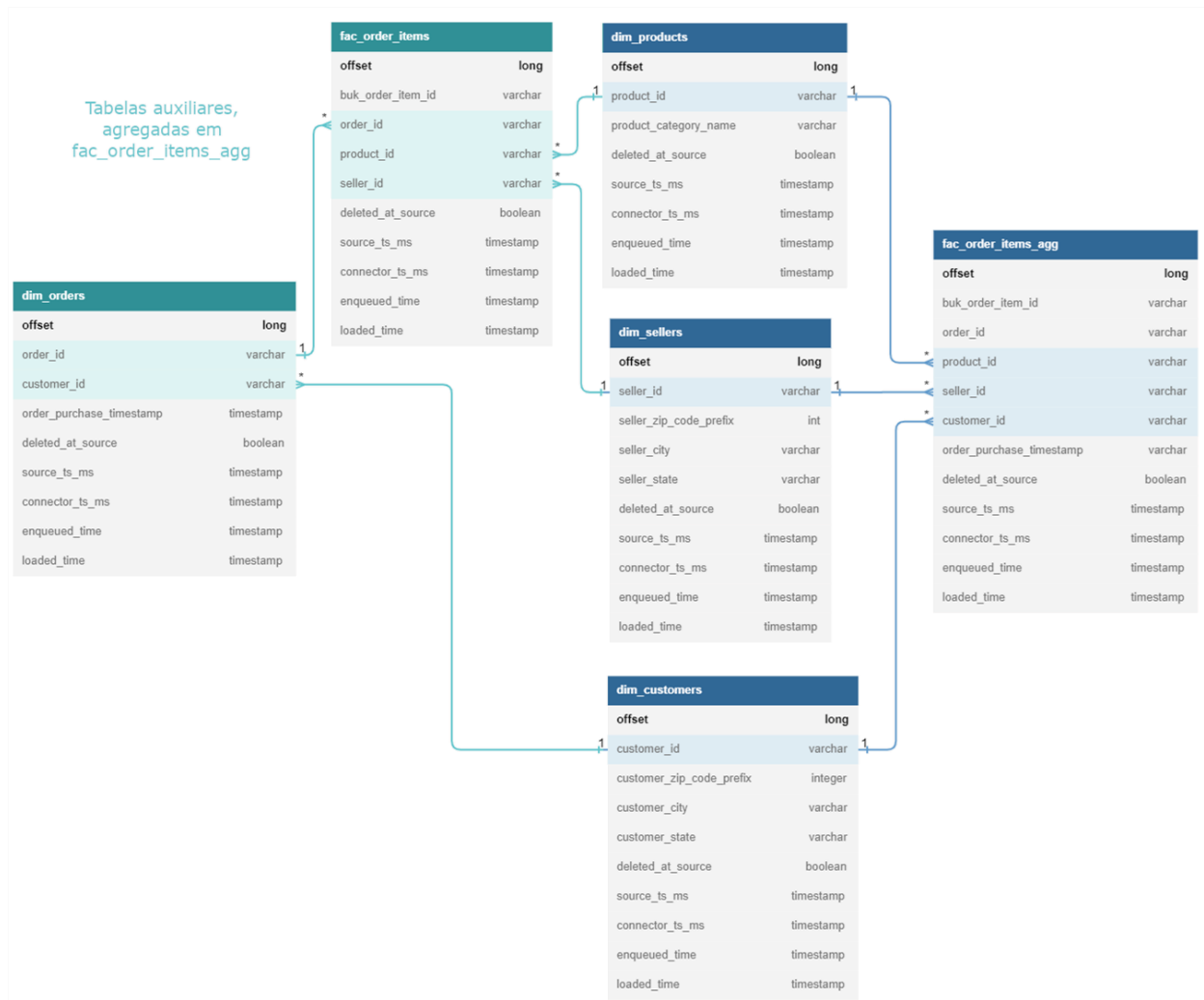


Figura 10 - Modelo de dados da estrutura staging

Numa primeira fase, são consideradas as tabelas tal como na estrutura fonte (*orders*, *order items*, *products*, *customers*, *sellers*) no entanto para sintetização dos dados para análise é feita a desnormalização deste esquema, com o objetivo de minimizar o tempo de execução de consultas no contexto do presente caso de uso. Para isso, é feita a agregação da tabela *orders* e *orders items*, eliminando a relação entre duas dimensões (*orders* – *customers*) e centralizando todas as chaves na tabela factual agregada *order items*.

Note-se que já acontece neste nível a filtragem dos campos relevantes para análise. São ainda mantidos alguns metadados para efeitos de avaliação de performance, como *source\_ts\_ms*, *conector\_ts\_ms*, *enqueued\_time* e *loaded\_time* (calculado à data/hora de inserção na base de dados *cloud*), ou para efeitos de rastreabilidade e identificação única, como o *offset*.

**Camada “olist\_dw”**

Por fim, o modelo de dados da fase *dw* consiste na transformação de registos duplicados nas tabelas de *staging*, adicionando o campo *created\_at* e *updated\_at* para acompanhar aquilo que são registos novos ou atualizações a registos existentes.

Adicionou-se o campo *units\_sold* como fator de agregação e que será a coluna alvo das consultas de recomendação.

Este modelo implementa o esquema em estrela e escolheu-se seguir a lógica de Dimensão de Alteração Lenta (*slowly changing dimensions* ou SCD) de tipo 1.

Uma dimensão de alteração lenta é uma dimensão que, tal como neste modelo, armazena tantos dados atuais como históricos. Existem 3 tipos de SCDs[71]:

- Tipo 1 – acontece quando dados novos reescrevem registos já existentes, não havendo lugar à manutenção de um histórico dos dados existentes. Cada registo mantém apenas campos de data de criação e data de atualização;
- Tipo 2 – acontece quando dados novos dão lugar à criação de novos registos. O registo com os dados existentes é dado como fechado, com data de criação e data de expiração para identificar o período em que esteve ativo. Dão lugar à manutenção do histórico inteiro dos dados existentes;
- Tipo 3 – é um intermédio dos tipos 1 e 2. Cada registo guarda duas colunas para um dado atributo, uma com o seu valor atual e outra com o último valor registado. Dão lugar a uma manutenção parcial do histórico de dados, preservando apenas o *backup* mais recente.

A razão para a escolha do Tipo 1 para aplicação ao modelo de dados *dw* foi a sua simplicidade de implementação e, como é referido na secção 3.5.2 “Arquiteturas alternativas”, a viabilização de uma arquitetura inteiramente desenhada em *streaming*.

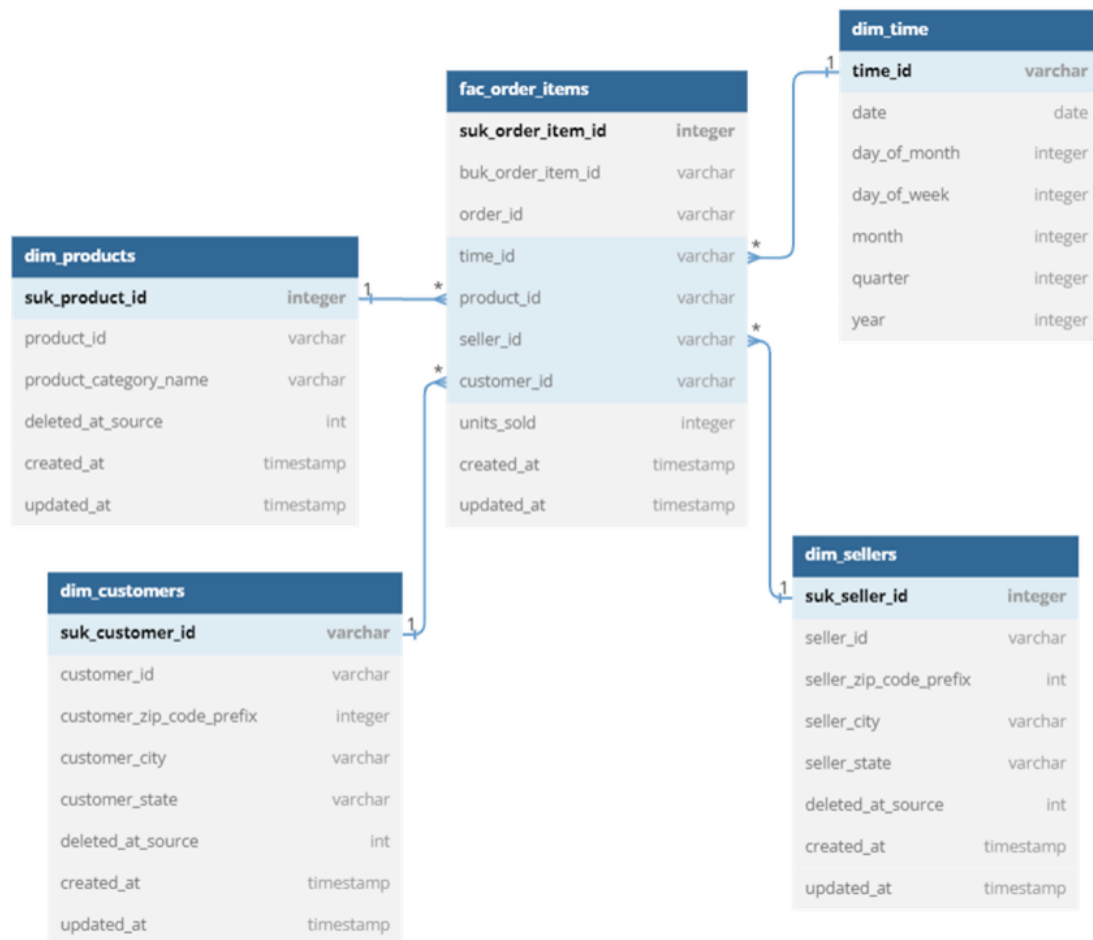


Figura 11 - Modelo de dados da estrutura dw

Note-se que, para além das tabelas *dim\_products*, *dim\_customers*, *dim\_sellers* e *fac\_order\_items* que foram extraídas do *schema staging*, foi criada uma tabela adicional *dim\_time*. O propósito desta tabela é de agregar registos temporais de forma mais rápida e eficiente.

### 3.5.2 Arquiteturas alternativas

Foram consideradas arquiteturas alternativas ao longo da fase de desenho do projeto, que, pelas tecnologias usadas ou lógica implementadas, apresentavam desvantagens face à arquitetura implementada.

Segue-se a descrição de duas alternativas que foram sugeridas, bem como as razões da sua não implementação.

#### Processamento separado em *batches*

Enquanto a arquitetura implementada seguiu a lógica de uma *pipeline* contínua em *streaming*, existem arquiteturas alternativas que separam o processo em duas ou mais partes: uma em

*streaming* e outra(s) em *batches* programados em intervalos curtos ou de execução cíclica (a iniciar logo após o término do *batch* anterior).

A porção em *streaming* corresponde a todo o processo de ingestão de dados e o processamento inicial de *staging*, enquanto a porção em *batch* se reserva para o restante processamento.

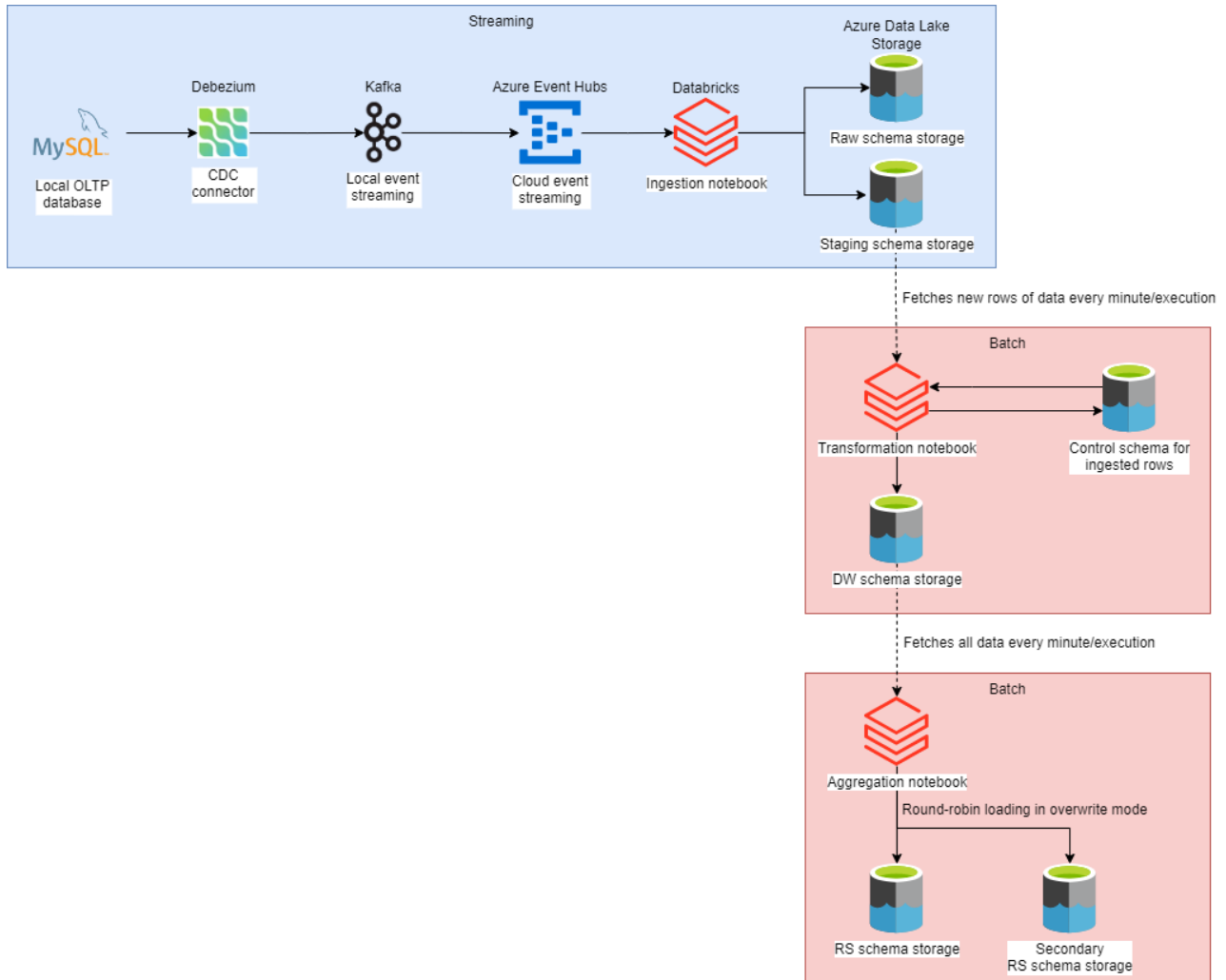


Figura 12 - Arquitetura alternativa de processamento em tempo real utilizando streaming e micro-batches

Esta arquitetura – representada na Figura 12 – foi pensada para possibilitar a execução de operações de transformação e agregação mais complexas.

Apesar da maior velocidade de um processo inteiramente implementado em *streaming*, este apresenta um conjunto limitado de operações que são suportadas.

Um dos exemplos destas limitações passa pelo *Spark Structured Streaming* não suportar subconsultas correlacionadas, necessárias para a criação de tabelas com fonte em outras tabelas e com lógicas complexas.

Na arquitetura implementada, a construção da tabela factual *fac\_order\_items*, representada na Figura 11, foi desenhada com uma lógica simples e as suas dimensões com um SCD de tipo 1 apenas. Isto acontece porque a implementação de um SCD de tipo 2 exigiria acrescentar uma chave artificial a cada tabela para efeitos de manutenção de histórico (isto é, uma chave para distinguir linhas com a mesma chave de negócio). Por sua vez, a criação da tabela factual exigiria elaborar uma subconsulta no momento da operação de *insert* às dimensões, com base nos registos em entrada, para obter a chave artificial correta.

Também seria possível implementar um SCD de tipo 2 com recurso a operações de *join* ao invés da subconsulta, no entanto este tipo de operação tem um grande impacto a nível de desempenho.

No final, esta arquitetura não foi adotada uma vez que se pretendeu manter o objetivo primário do presente projeto de produzir um processo o mais próximo possível de tempo real, ainda que mais simples.

#### **Utilização mais intensiva de serviços *cloud***

A utilização da *cloud* foi uma preocupação sempre presente no desenho do projeto, uma vez que representa uma mais-valia para a base de clientes da DataCentric, muitos dos quais se encontram em processo de migração de processos alojados em servidores locais para serviços em *cloud*.

A apresentação de uma prova de conceito que possua as duas componentes e prove que a tecnologia em tempo real pode ser aplicada nesses mesmos processos em fase de migração (com fonte *on-premise* mas com produção de resultados na *cloud*) teria uma aplicação imediata.

Por esta razão, pretendeu-se encontrar um equilíbrio para este tipo de empresa cuja utilização da *cloud* está ainda na fase inicial, tendo sido para o efeito analisadas arquiteturas com uso mais ou menos intensivo de serviços *cloud*.

Um exemplo que alavanca mais as funcionalidades oferecidas pelo Microsoft Azure é a arquitetura representada na Figura 13.

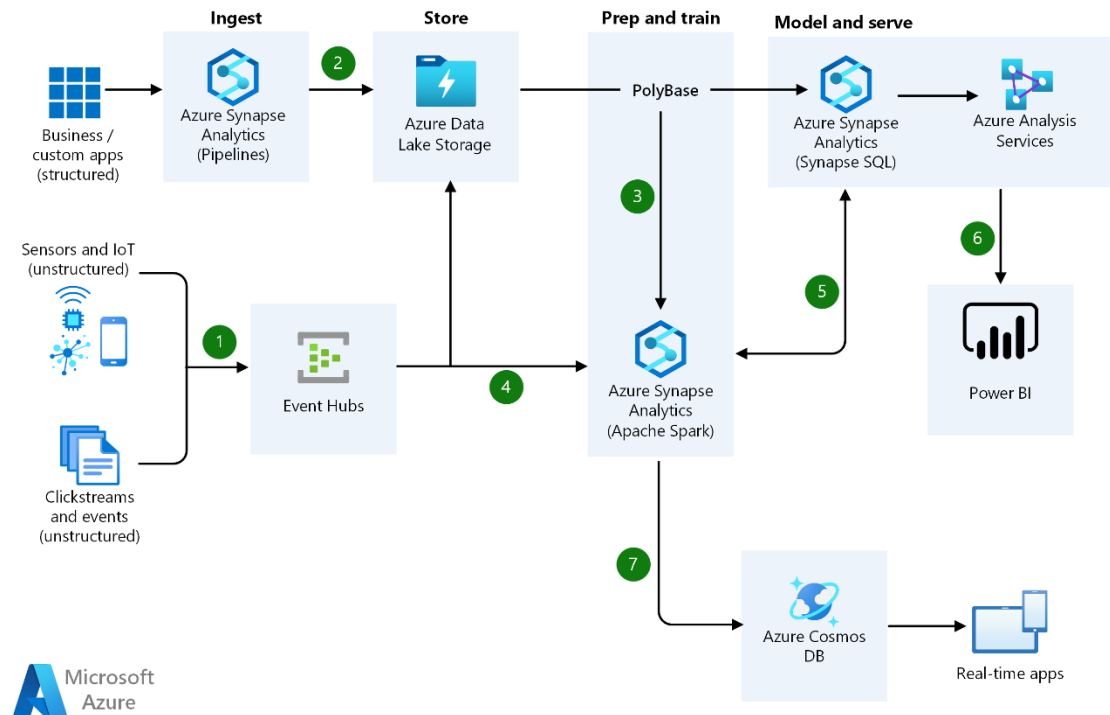


Figura 13 - Arquitetura alternativa de processamento em tempo real utilizando Azure[72]

Neste caso, os eventos gerados nas fontes de dados são diretamente ingeridos pelo Event Hubs ou pelas pipelines criadas em Synapse Analytics.

A Synapse Analytics é uma ferramenta de integração, análise e armazém de dados, com uma interface fácil de usar e sem configuração necessária[72]. Esta constitui uma alternativa ao Databricks, utilizado na arquitetura implementada. A principal desvantagem desde último é que exige maior conhecimento técnico das ferramentas *open-source* Apache, da gestão de *clusters* e das suas configurações. No entanto, o Databricks possui uma versão otimizada do Spark com melhor desempenho e combina funcionalidades de armazenamento em *data lake* e armazém de dados para melhor suportar as necessidades do processo ao longo da *pipeline*. Por esta razão a arquitetura implementada focou o uso de Databricks em detrimento do Synapse Analytics.

O método de armazenamento intermédio utilizado nesta arquitetura é o Data Lake Storage, tal como na implementada. No entanto, para efeitos de disponibilização ao *end-point* da aplicação é utilizada a Cosmos DB, uma base de dados NoSQL distribuída. A principal diferença passa pela menor latência da Cosmos DB comparativamente à Data Lake Storage. Esta última está construída sobre o sistema de ficheiros *Blob storage* – tal com mencionado na secção 2.4.1 –, apresentando por isso algumas limitações, como por exemplo, maior latência em operações de *update* (que exigem a reescrita do ficheiro inteiro) e a restrição a um único



modelo de armazenamento em *key-value* – não suporta outros modelos como documento, gráfico ou família de colunas[73].

Note-se ainda que a limitação de recursos disponíveis representa também um fator na escolha de uma arquitetura menos intensiva em serviços *cloud*. Uma arquitetura deste género pode ser eficiente em aplicações reais, com grande volume de dados para processar e, consequentemente, necessidade de aquisição de servidores ou processamento adicional. No entanto, no presente caso, o consumo de recursos locais é reduzido e não comporta custos adicionais. Além disso, uma vez que os serviços *cloud* são do tipo *pay-as-you-go*, exigem um investimento constante, que ultrapassaria o orçamento disponível para o projeto caso fosse implementada a arquitetura alternativa. À data de escrita do presente relatório, a implementação da arquitetura escolhida acumula um custo de subscrição de 200€.

## 4 Implementação da solução

Neste capítulo irá ser apresentado o processo de implementação das funcionalidades do projeto, de acordo com os requisitos funcionais identificados e a arquitetura desenhada para o efeito.

Numa primeira instância, serão identificadas as tecnologias e ambientes de desenvolvimento utilizados durante o desenvolvimento da solução, com base na análise efetuada das tecnologias existentes e trabalhos relacionados.

De seguida, será feita a apresentação das funcionalidades implementadas, com a respetiva explicação da lógica a nível de algoritmos e componente utilizados. Associada a cada funcionalidade está ainda a descrição dos testes efetuados para validação do cumprimento dos requisitos.

Por fim, será feita uma avaliação da solução baseada em métricas recolhidas ao longo da sua execução.

### 4.1 Tecnologias utilizadas

A implementação da solução exigiu a utilização de várias ferramentas, pela natureza distinta de cada fase do processo.

A fase de *setup* e ingestão envolveu principalmente a utilização de ferramentas de código aberto e de reduzido peso em termos computacionais e de memória, como MySQL, Debezium, Docker e ferramentas Apache. Tal deveu-se não só às limitações da máquina utilizada no desenvolvimento como a considerações relativas ao baixo consumo de recursos nos servidores da empresa em caso de implantação da solução, visto que a natureza de ambiente local desta fase torna a escalabilidade um aspeto crítico.

Na transição para o ambiente *cloud* foi ainda utilizada a ferramenta Event Hubs da provedora Azure, que facilitou bastante o processo de conexão e autenticação ao motor de processamento, também ele em execução na *cloud*. De notar, no entanto, que a utilização desta ferramenta não era estritamente necessária e que comportou a maior fatia de custos durante o desenvolvimento, pela necessidade de estar permanentemente em execução.

Na fase de processamento utilizou-se o motor Apache Spark, integrado no ambiente Databricks. Esta ferramenta foi escolhida por oferecer uma versão otimizada do Spark, que já

sendo o motor com maior velocidade de processamento entre as alternativas exploradas no capítulo Tecnologias existentes (secção 2.4), torna-se assim ainda mais vantajoso.

A Tabela 14 apresenta um resumo destas tecnologias utilizadas no desenvolvimento da solução, segregada pelas fases do processo.

Tabela 14 - Resumo das principais tecnologias utilizadas

Fase	Categoria	Tecnologia	Breve descrição
Simulação de fluxo de dados	Base de dados	MySQL	Sistema de gestão de base de dados relacionais que utiliza a linguagem SQL como interface[74].
Ingestão	Comunicação entre aplicações	Debezium	Plataforma distribuída que captura alterações ao nível de linha em bases de dados[75].
	Comunicação entre aplicações	Apache Kafka	Plataforma de captura de eventos em tempo real que utiliza um sistema de <i>publish-subscribe</i> para importar e exportar dados[76].
	Ambiente de desenvolvimento	Docker	Plataforma de desenvolvimento, entrega e execução de aplicações que utiliza virtualização de sistema operacional para criar ambientes independentes ( <i>containers</i> )[77].
	Orquestração	Apache ZooKeeper	Serviço centralizado para manutenção de configurações, nomenclaturas e sincronização em aplicações distribuídas[78].
	Comunicação entre aplicações	Azure Event Hubs	Plataforma em <i>cloud</i> de ingestão de eventos em tempo real, que oferece ligação a serviços Azure e Microsoft como fonte de dados[79].
Ingestão/ Processamento	Ambiente de desenvolvimento	Azure Databricks	Plataforma em <i>cloud</i> de análise de dados, com integração Spark, que

Fase	Categoria	Tecnologia	Breve descrição
			permite desenvolver código em <i>notebooks</i> e armazenar dados em <i>lakehouses</i> (combina <i>data warehouse</i> e <i>data lake</i> )[80].
	Framework de desenvolvimento	Apache Spark	Framework de computação distribuída, que utiliza <i>clusters</i> e paralelismo para executar análises de dados de forma rápida[81].
	Linguagem de programação	Python	Linguagem de programação com elevado número de bibliotecas de manipulação de dados, que neste caso serve como interface para utilizar o motor Spark.
Armazenamento	Base de dados	Azure Delta Lake	Camada de armazenamento otimizada para Azure Databricks, que guarda tabelas e dados em <i>lakehouse</i> . Utiliza a expansão de ficheiros <i>parquet</i> com registo de transações ACID[82].
	Base de dados	Azure Data Lake Storage	Plataforma de armazenamento de <i>big data</i> em lógica de <i>data lake</i> . Serve como tecnologia subjacente ao Delta Lake.

## 4.2 Funcionalidades implementadas

A implementação das funcionalidades seguiu a análise e desenho das *user stories* abordadas no capítulo Requisitos funcionais (secção 3.3). Neste capítulo será feita uma breve descrição do método de implementação usado para cada *user story* e apresentados os diagramas de sequência correspondentes.

Uma vez que as *user stories* DCRA-1 e DCRA-2 são destinadas a análise e modelação, não serão abordadas nesta secção. A implementação de funcionalidades é conduzida nas *user stories* DCRA-3, DCRA-4 e DCRA-5, cujo procedimento será descrito de seguida.

Adicionalmente, como parte da implementação de cada funcionalidade foram conduzidos testes de qualidade para efeitos de validação, seguindo o modelo utilizado pela equipa de Quality Assurance (QA) da DataCentric. Incluiu-se nesta secção apenas a descrição dos testes mais relevantes para cada funcionalidade. A lista completa de testes conduzida pode ser consultada no Anexo A.

### US DCRA-3 – Captação de eventos operacionais em tempo real

Esta funcionalidade consiste na visualização, em tempo real e em ambiente *cloud*, dos dados que são gerados localmente.

Para o efeito, o engenheiro de dados insere, atualiza ou elimina registos na base de dados local “olist”, podendo de seguida aceder ao *notebook* de ingestão Databricks em execução e visualizar o *dataframe* a ser preenchido instantaneamente.

Esta interação está detalhada no diagrama de sequência representado na Figura 16.

Neste diagrama, para além das componentes do Databricks estão também representadas as componentes MySQL, Debezium e Kafka, que se encontram configuradas para correr localmente em *containers* de Docker, como se pode visualizar na Figura 14 **Error! Reference source not found..**





















Name	Image	Status	Port(s)	Last started	Actions
 <b>connect</b> 075f4d370efc 	<a href="https://quay.io/debezium/connect:2.1">quay.io/debezium/connect:2.1</a>	Running	<a href="#">8083:8083</a> 	7 seconds ago	 
 <b>mysql</b> 7fa91426b363 	<a href="https://quay.io/debezium/example-mysql:2.1">quay.io/debezium/example-mysql:2.1</a>	Running	<a href="#">3306:3306</a> 	2 minutes ago	 
 <b>kafka</b> ec42d0da9bf7 	<a href="https://quay.io/debezium/kafka:2.1">quay.io/debezium/kafka:2.1</a>	Running	<a href="#">9092:9092</a> 	1 minute ago	 
 <b>zookeeper</b> d18ffe6e7f33 	<a href="https://quay.io/debezium/zookeeper:2.1">quay.io/debezium/zookeeper:2.1</a>	Running	<a href="#">2181:2181</a>  <a href="#">Show all ports (3)</a>	2 minutes ago	 

Figura 14 - Componentes executadas em containers locais no Docker

Está ainda representada a componente de ligação à *cloud* Event Hub, cuja configuração é feita diretamente na plataforma web da Microsoft Azure, juntamente com os outros serviços, tal como apresentado na Figura 15Figura 15.

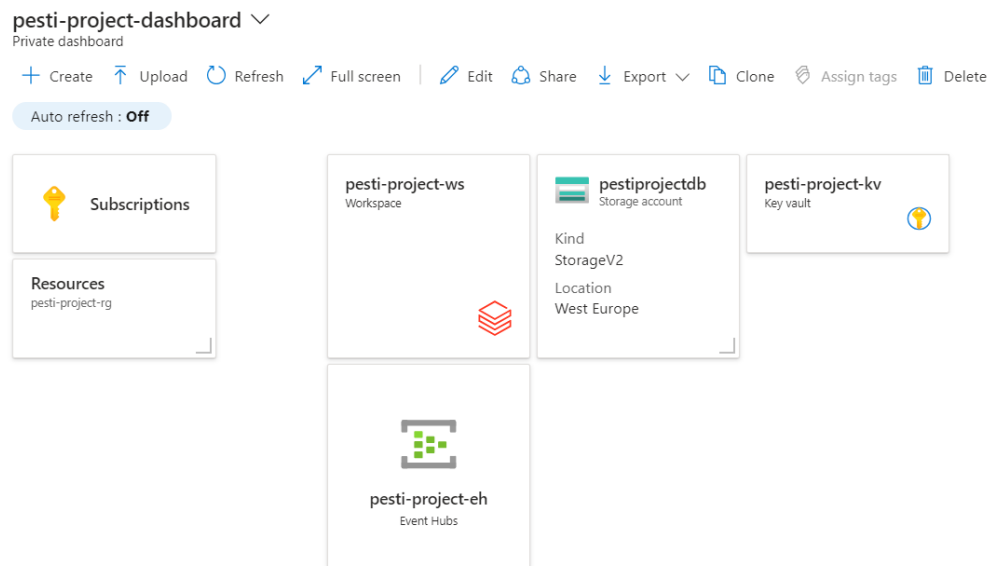


Figura 15 - Componentes executadas em ambiente cloud na plataforma Azure

As ligações preliminares entre estas componentes são feitas nos passos 1 a 3 do diagrama, com o Event Hub a consumir o tópico Kafka e o conector Debezium, produtor do mesmo tópico Kafka, a monitorizar mudanças na base de dados MySQL.

A partir do passo 4, a interação é feita ao nível do Databricks, começando por estabelecer conexão e autenticação ao Event Hub, que por sua vez lê e retorna o conteúdo do tópico Kafka. Este primeiro ponto de contacto retorna os dados a ser persistidos no esquema *raw*.

Nos passos 10 a 12, é feita uma transformação aos dados *raw*, procedendo-se à sua formatação, validação e à criação de novas colunas de controlo.

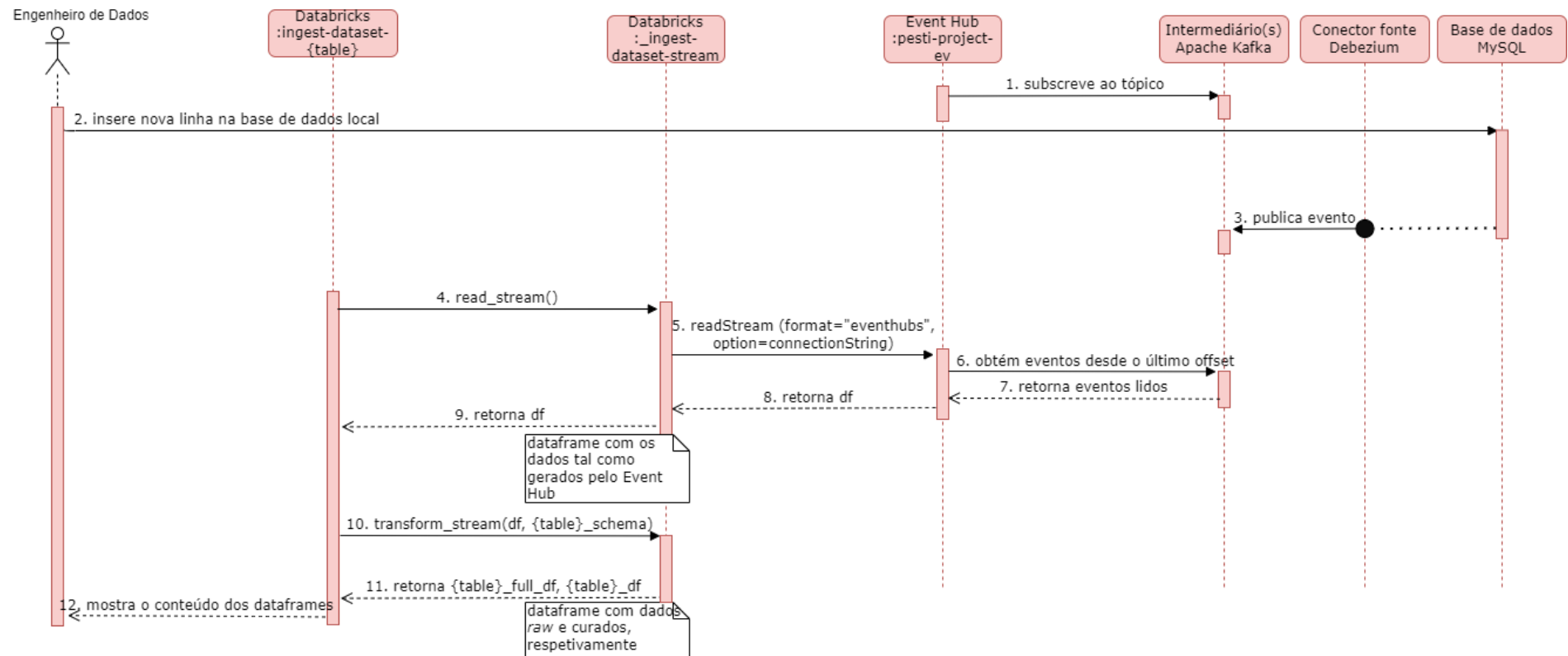


Figura 16 - Captação de eventos operacionais em tempo real (Diagrama de sequência)

Os testes conduzidos a esta funcionalidade consistiram em:

- Validar a criação das tabelas locais na base de dados MySQL;
- Validar a criação dos tópicos Kafka;
- Validar a execução dos *notebooks* Databricks e visualização de dados.

Na Tabela 15 encontra-se a tabela resumida da lista de testes efetuados.

*Tabela 15 - Lista resumida de testes efetuados (US DCRA-3)*

#CASO DE USO	CENÁRIO	#CASO DE TESTE	OBJETO DE TESTE	RESULTADO ESPERADO	RESULTADO TESTE
DCRA-3	Criação do esquema "olist-dataset", "olist" e dos scripts de carregamento de dados	#1	Validar se as tabelas "sellers", "customers", "orders", "order_items" e "products" existem em ambos os esquemas e os dados foram inseridos com sucesso	Tabelas existem e possuem dados	PASS
DCRA-3	Criação de ligação da base de dados MySQL ao Kafka	#2	Validar se os tópicos Kafka correspondentes às tabelas "sellers", "customers", "orders", "order_items" e "products" existem e estão populados	Tópicos existem e possuem dados	PASS
DCRA-3	Criação de ligação do Kafka à cloud	#3	Validar se os tópicos Kafka correspondentes às tabelas "sellers", "customers", "orders", "order_items" e "products" estão presentes na interface do Event Hubs	Tópicos são visíveis na interface cloud	PASS
DCRA-3	Execução do notebook de ingestão Databricks	#4	Validar se a execução do notebook correu com sucesso e gerou a visualização dos dataframes de ingestão, com os dados da tabela MySQL	Dataframes possuem dados inseridos localmente no momento da execução	PASS

#### US DCRA-4 – Armazenamento dos eventos em armazém de dados

Esta funcionalidade consiste em persistir, em base de dados *cloud*, os dados à medida que são recebidos a partir da fonte. A persistência dos dados é feita de acordo com o esquema de armazém de dados definido na secção 3.5.1.

Para o efeito, e tal como na *user story* anterior, o engenheiro de dados insere, atualiza ou elimina registos na base de dados local "olist", podendo de seguida aceder à Data Lake Storage e efetuar uma consulta pelos novos dados.



Esta interação está detalhada no diagrama de sequência representado na Figura 16.

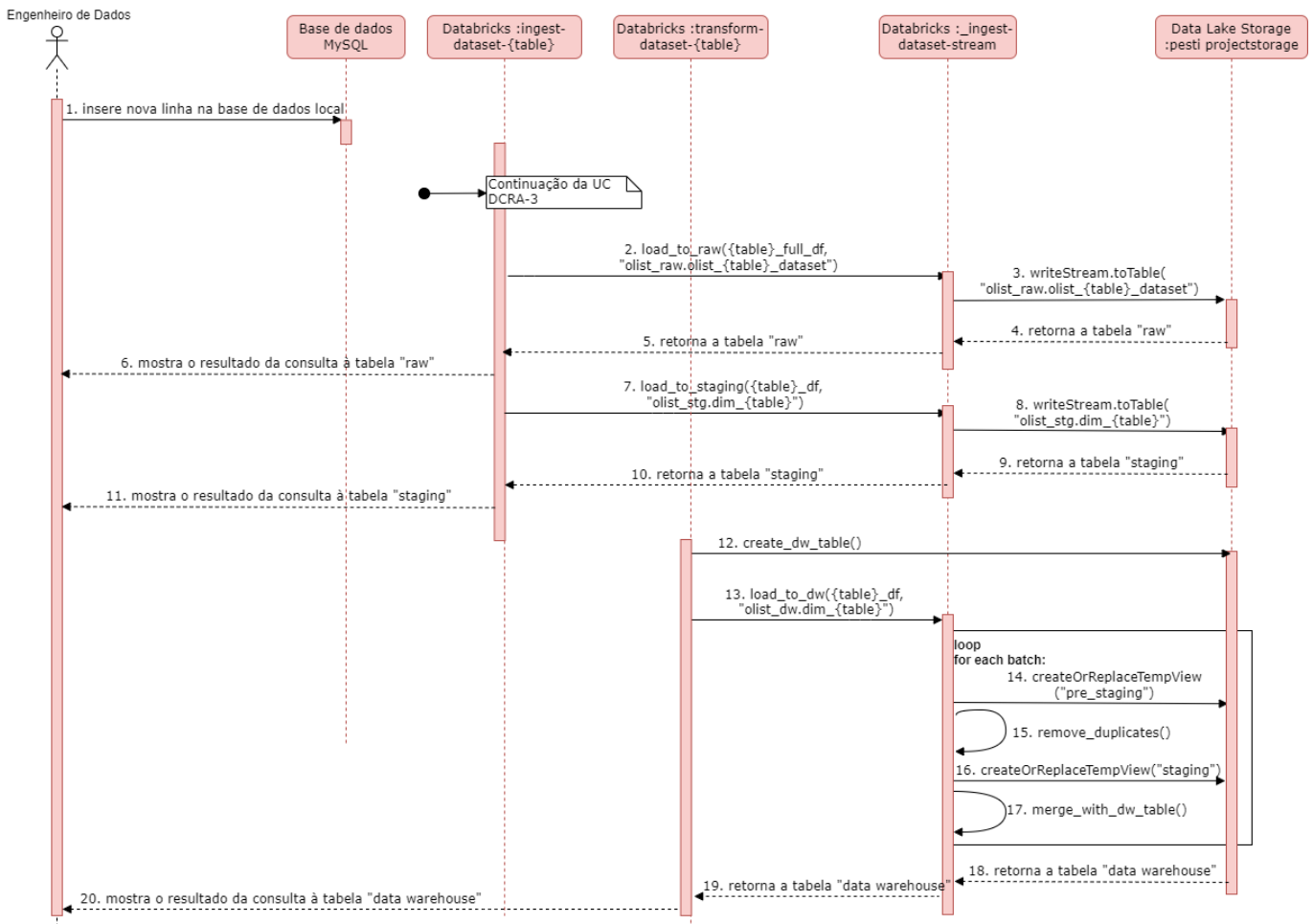


Figura 17 - Armazenamento dos eventos em armazém de dados (Diagrama de sequência)

Esta *user story* é um seguimento direto da anterior, partindo dos *dataframes* de ingestão resultantes desta. O seu funcionamento passa por escrever os dados do *dataframe* na Data Lake Storage de acordo com os modelos *raw*, *staging* e *data warehouse* já definidos.

Nos passo 2 até 10, é feita a escrita dos dados nos esquemas *raw* e *staging* diretamente durante o processo de ingestão, uma vez que envolvem transformações muito simples.

De seguida, é feita a criação da tabela em esquema *data warehouse*. No passo 13, o *dataframe* é escrito nessa tabela em *micro-batches*, de modo a permitir a execução de transformações mais complexas. Ao contrário dos outros esquemas, em que a existência de duplicados não é problemática, neste esquema é necessário fazer uma limpeza de eventuais registos captados mais do que uma vez para que as consultas não sejam afetadas, o que acontece no passo 15.

Por fim, no passo 17 procede-se ao *merge* de cada *micro-batch*. A operação *merge* permite efetuar uma operação de *update* ou *insert* conforme o registo já exista ou não na tabela alvo, seguindo assim a lógica de SCD tipo 1 mencionada na secção 3.5.1.

A realização de testes para esta funcionalidade envolveu validar:

- A criação das tabelas em cada esquema;
- As colunas e tipo de dados de cada tabela segundo o respetivo modelo de dados;
- A consistência dos mesmos dados nos 3 esquemas.

Na Tabela 16 encontram-se alguns exemplos de testes efetuados.

*Tabela 16 - Lista resumida de testes efetuados (US DCRA-4)*

#CASO DE USO	CENÁRIO	#CASO DE TESTE	OBJETO DE TESTE	RESULTADO ESPERADO	RESULTADO TESTE
DCRA-4	Criação do esquema "olist-dw" e inserção dos dados provenientes da base de dados local	#9	Validar se as tabelas "dim_sellers", "dim_customers", "fac_order_items" e "dim_products" existem no esquema e os dados foram inseridos com sucesso	Tabelas existem e possuem dados	PASS
DCRA-4	Modelação do esquema "olist_dw" de acordo com o respetivo modelo de dados	#10	Validar se as tabelas possuem as colunas e tipos de dados definidos pelo modelo de dados "olist_dw"	Colunas existem e têm o tipo de dados correto	PASS
DCRA-4	Inserção de registos nos esquemas "olist_raw", "olist_stg" e "olist_dw"	#11	Validar se o mesmo registo apresenta dados consistentes nos esquemas "olist_raw", "olist_stg" e "olist_dw"	Dados existem e são equivalente em todos os esquemas	PASS

### US DCRA-5 – Desenvolver sistema de recomendação

Esta funcionalidade consiste em criar uma tabela de recomendação baseada num algoritmo de filtragem colaborativa, que pode ser consultada de modo a obter a recomendação de produtos para um dado utilizador.

Na Figura 18, está representada a sequência de criação da tabela de recomendação.

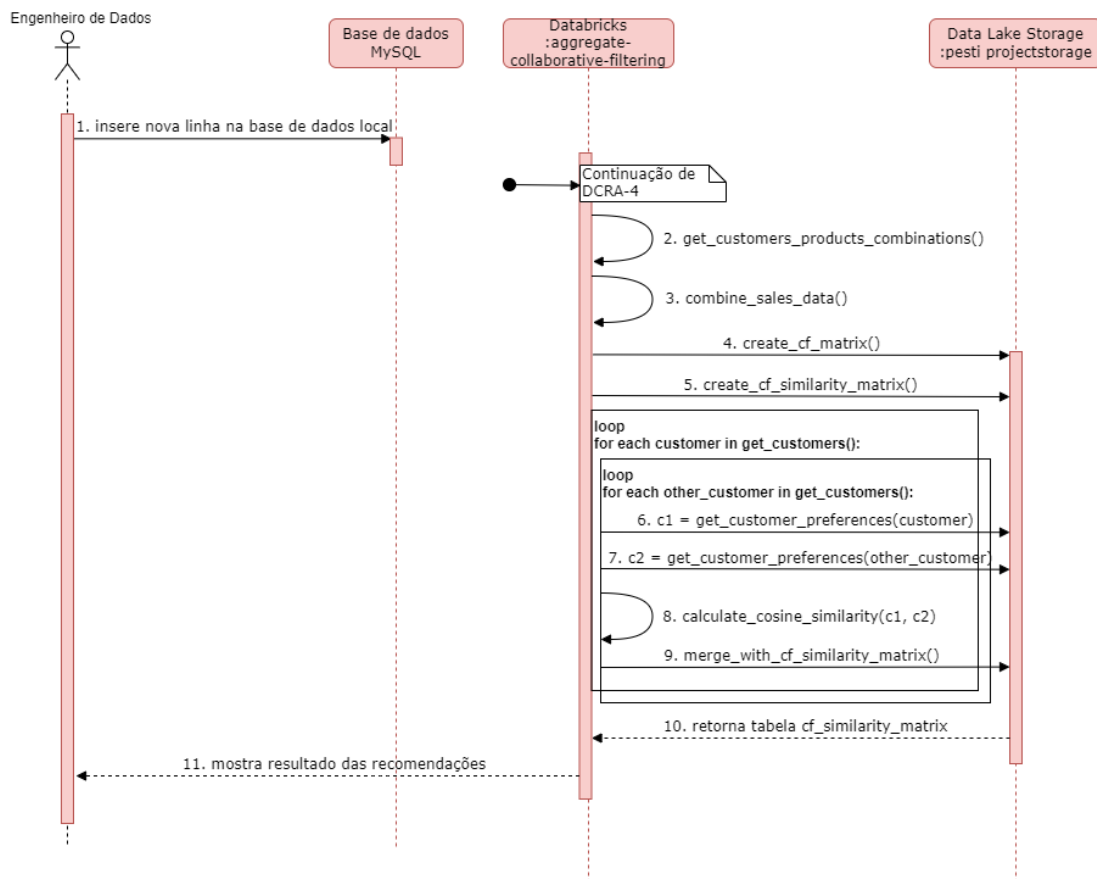


Figura 18 - Desenvolvimento de sistema de recomendação (Diagrama de sequência)

Em primeiro lugar, nos passos 2 a 4, é criada a tabela auxiliar “cf\_matrix”, que contém todas as combinações possíveis de clientes e produtos, juntamente com as unidades vendidas para cada uma destas combinações.

De seguida, nos passos 5 a 9, são calculados os coeficientes de similaridade (cuja fórmula é explicada na secção 2.2.2) para cada par de clientes existentes na tabela “cf\_matrix” e posteriormente guardados na tabela final “cf\_similarity\_matrix”.

Os produtos recomendados para um dado cliente podem depois ser obtidos através de uma consulta que faz o cruzamento desta nova tabela com a tabela factual do armazém de dados. Esta consulta está descrita na Figura 19, em que os parâmetros correspondem a:

- *customer* – id do cliente para o qual se pretende obter recomendações;
- *sim\_threshold* – coeficiente de similaridade mínimo a considerar;
- *n\_users* – número máximo de outros clientes a usar como referência;
- *n\_products* – número máximo de produtos a recomendar.

```

SELECT DISTINCT rs.product_id
FROM olist_dw.fac_order_items rs
LEFT JOIN (
    SELECT product_id
    FROM olist_dw.fac_order_items
    WHERE customer_id = {customer}
) excl ON rs.product_id = excl.product_id
WHERE rs.customer_id IN (
    SELECT other_customer_id
    FROM olist_rs.cf_similarity_matrix
    WHERE customer_id = {customer}
    AND cos_similarity > {sim_threshold}
    ORDER BY cos_similarity DESC
    LIMIT {n_users}
)
AND excl.product_id IS NULL
LIMIT {n_products};

```

Figura 19 - Consulta utilizada para obter recomendações a partir da tabela *cf\_similarity\_matrix*

De notar que no diagrama da Figura 18 está representada a lógica de criação inicial da tabela de recomendação, que é mais exigente em termos de processamento devido à existência de *for loops* encadeados.

De modo a reduzir o tempo de execução do processo em geral, escolheu-se correr este algoritmo apenas uma vez, no início do processo, e efetuar um reprocessamento parcial quando um registo novo é inserido. Isto significa que quando um novo produto ou novo cliente são adicionados ao sistema, despoletam a adição de uma coluna à tabela “*cf\_matrix*”. Já quando é registada uma nova venda, o valor existente para o produto/cliente especificados é alterado na “*cf\_matrix*” também. Assim, a “*cf\_similarity\_matrix*” é recalculada apenas para esse cliente e respetivos pares.

A nível de testes a esta funcionalidade, procedeu-se à validação de:

- Criação das tabelas “*cf\_matrix*” e “*cf\_similarity\_matrix*”;
- Valor matemático resultante do coeficiente de similaridade para certos casos unitários;
- Adição de colunas à tabela “*cf\_matrix*” quando novos produtos/clientes são registados;
- Alteração da quantidade na tabela “*cf\_matrix*” e do coeficiente de similaridade na tabela “*cf\_similarity\_matrix*” quando uma nova venda é registada.

Apresenta-se um excerto do registo destes testes na Tabela 17.

Tabela 17 - Lista resumida de testes efetuados (US DCRA-5)

#CASO DE USO	CENÁRIO	#CASO DE TESTE	OBJETO DE TESTE	RESULTADO ESPERADO	RESULTADO TESTE
DCRA-5	Criação do esquema "olist_rs" e inserção dos dados correspondentes a todos os clientes e produtos existentes.	#12	Validar se as tabelas "cf_matrix" e "cf_similarity_matrix" existem no esquema e os dados foram inseridos com sucesso	Tabelas existem e possuem dados	PASS
DCRA-5	Alteração do esquema "olist_rs" quando novos produtos, clientes ou vendas são registados.	#13	Validar se as colunas da tabela "cf_matrix" correspondem a todos os clientes e produtos registados.	Colunas existem e têm os dados corretos	PASS
DCRA-5	Alteração do esquema "olist_rs" quando novos produtos, clientes ou vendas são registados.	#14	Validar se os valores da tabela "cf_matrix" correspondem a todas as unidades vendidas para cada par cliente/produto.	Valores existem e corresponde m à soma correta	PASS
DCRA-5	Cálculo do coeficiente de similaridade na tabela "cf_similarity_matrix"	#15	Validar se os valores da tabela "cf_similarity_matrix" correspondem ao coeficiente de similaridade correto para cada par cliente/cliente.	Valores existem e corresponde m ao cálculo correto	PASS

### 4.3 Avaliação da solução

Para além da avaliação feita através da concordância com os testes de qualidade mencionados na secção anterior, foi criado um modelo para avaliação do desempenho da solução em termos de velocidade de execução e latência.

Através da inserção de metadados nas tabelas persistidas em memória, tornou-se possível obter carimbos data/hora para cada operação realizada sobre os dados, desde a sua criação no ambiente local até à sua disponibilização em formato agregado no ambiente *cloud*.

Como exemplo, na Figura 20 estão representados graficamente os tempos de execução das operações sobre a tabela *fac\_order\_items*, em que:

- A *source\_latency* representa a quantidade de tempo (em segundos) desde a criação do registo na base de dados local e a sua assimilação pelo conector de fonte – neste caso, o Debezium;
- A *sink\_latency* representa a quantidade de tempo (em segundos) entre a passagem dos dados desde o conector de fonte e o conector de destino – neste caso, o Event Hub;
- A *processing\_latency* representa a quantidade de tempo (em segundos) desde a receção do evento no Event Hub e o seu carregamento para a tabela de *staging*;
- A *total\_latency* representa a soma de todos os intervalos de tempo acima, logo o tempo total que o processo demora a executar para cada registo.

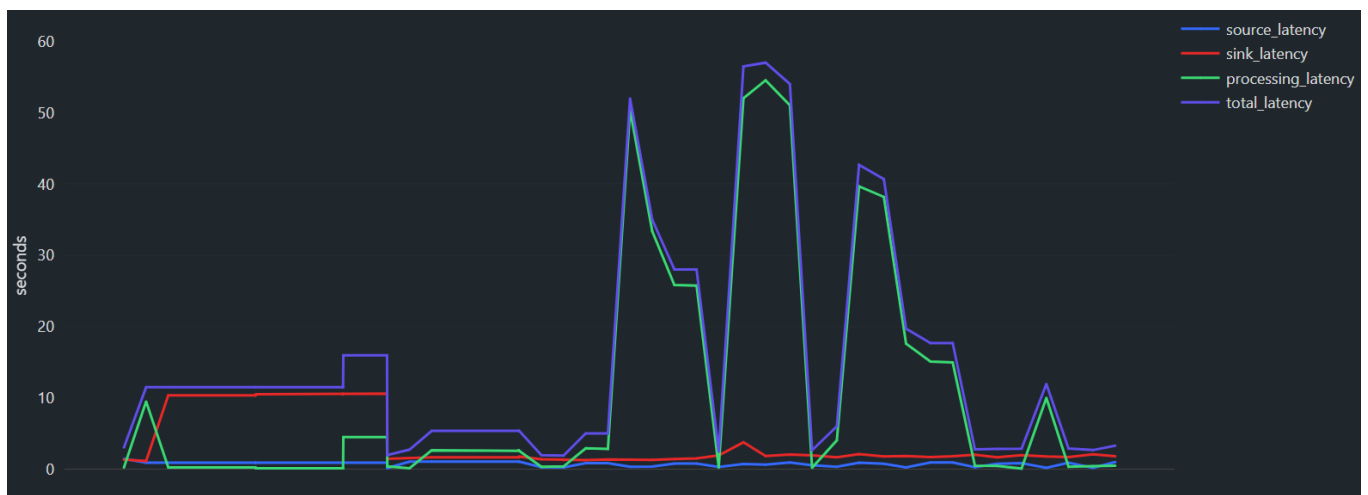


Figura 20 - Visualização da latência de processamento para os registos da tabela *fac\_order\_items*

Uma das principais conclusões a tirar deste exemplo passa pela verificação de uma latência muito baixa do processo na fonte (*source*) e no destino (*sink*), indicando que as tecnologias selecionadas para construir a *pipeline* de ingestão foram eficazes.

Observa-se também que a fase de processamento é alvo de maior latência, como seria expectável, uma vez que exige maior poder computacional. De notar que este exemplo foi executado num *cluster* de capacidades mínimas – 1 único nó, 14 GB de memória e 4 cores – para efeitos de poupança de custos.

Acrescenta-se ainda que é avaliada a latência média e mediana para o conjunto de registos como critério adicional de avaliação. Neste exemplo, a latência média total foi de 14.63 segundos e a latência mediana total de apenas 5.35 segundos. Este último valor aproxima-se do alvo de 5 segundos definido nos Requisitos não funcionais mas diverge significativamente do valor médio. Tal revela que o processo apresenta alguns valores de latência extremos, o que se pode dever a quebras de rede, indisponibilidade dos serviços a montante ou sobrecarga da unidade de processamento causada por picos no influxo de dados.

## 5 Conclusões

Neste capítulo será efetuada uma reflexão acerca do trabalho realizado, avaliando a concretização dos objetivos estabelecidos no capítulo introdutório. Esta reflexão incluirá ainda uma análise crítica das limitações da solução obtida, identificando potenciais melhorias futuras.

Por fim, será apresentada uma apreciação final sobre o projeto, tendo em conta os contributos para a DataCentric e a comunidade científica. Será ainda considerada uma reflexão acerca da experiência profissional proporcionada pelo estágio curricular.

### 5.1 Objetivos concretizados

Tal como foi referido na secção 1.2.1, o principal objetivo do projeto desenvolvido foi a geração com sucesso de dados analíticos em tempo real.

Na Tabela 18 são apresentadas as *milestones* definidas no início do projeto e o respetivo grau de realização após a sua conclusão.

*Tabela 18 - Lista de milestones e respetivo grau de realização*

Objetivo	Grau de realização
1- Modelação de uma arquitetura robusta e escalável, com considerações relativas a reduzidos tempos de processamento e armazenamento de grandes quantidades de dados	Maioritariamente realizado
2- Desenvolvimento de processos para capturar eventos operacionais em tempo real	Realizado
3- Desenvolvimento de processos de deteção e tratamento automático de erros e inconsistência de dados	Parcialmente realizado
4- Análise e desenvolvimento de um sistema de recomendação baseado em filtragem colaborativa	Realizado
5- Desenho de um modelo de dados analíticos para armazenamento e posterior apresentação de dados processados	Realizado
6- Desenvolvimento de processos de armazenamento em armazém de dados	Realizado



Objetivo	Grau de realização
7- Desenvolvimento de processos de agendamento e execução automática da aplicação	Realizado

De notar que, para além destas *milestones* atingidas, todos os requisitos funcionais e não funcionais descritos nas secções 3.3 e 3.4 foram implementados e concluídos com sucesso, com algumas limitações apenas ao que diz respeito ao requisito não funcional de Concorrência.

Na secção seguinte serão explicadas as limitações que impediram a realização completa dos *milestones* e requisitos mencionados.

## 5.2 Limitações e trabalho futuro

Devido a restrições de tempo, orçamento ou recursos técnicos, nem todos os aspetos previstos para o sistema foram possíveis de implementar de acordo com o nível de qualidade pretendido.

Uma das primeiras limitações enfrentadas foi o orçamento disponível para o projeto, que afetou a plena validação da 1ª *milestone*, descrita como “modelação de uma arquitetura robusta e escalável, com considerações relativas a reduzidos tempos de processamento e armazenamento de grandes quantidades de dados”.

Neste contexto, a execução de serviços *cloud* foi limitada, tendo sido estudadas arquiteturas alternativas (secção 3.5.2) com potenciais vantagens em relação à implementada. A utilização da arquitetura alternativa poderia proporcionar maior escalabilidade e reduzir o tempo de desenvolvimento do projeto, permitindo a implementação de funcionalidades adicionais, tais como um algoritmo de recomendação mais complexo ou a criação de uma interface gráfica para interagir dinamicamente com o sistema.

Esta limitação de orçamento foi um impedimento também para validar mais a fundo a escalabilidade do sistema implementado, impossibilitando o aumento da capacidade de processamento e a exploração de CPUs de diferentes especificações, devido ao esgotamento de créditos disponíveis.

Dito isto, é importante salientar que o balanço de desempenho/custo obtido foi satisfatório para o volume de dados utilizados.

Outra limitação encontrada passou pela detecção de erros, incluída na 3ª *milestone*, descrita como “desenvolvimento de processos de detecção e tratamento automático de erros e inconsistência de dados”.

Apesar de ter sido implementada com sucesso a validação de tipo e estrutura de dados, admite-se que esta validação superficial não seria suficiente num contexto real e poderia ter sido mais desenvolvida. Trabalhos futuros deverão fornecer maior foco na detecção de inconsistências de dados, como por exemplo a receção de uma encomenda com um produto que não existe na respetiva tabela de produtos.

Por fim, para responder ao requisito não funcional de concorrência, que exige que o sistema suporte consultas e manipulações concorrentes de dados, salientam-se as limitações de testagem detetadas.

Devido à natureza de prova de conceito do projeto, o volume de dados utilizado foi reduzido (cerca de 100.000 registos) face ao volume real de dados dos negócios a retalho. Este fator, combinado com a inexistência de utilizadores para testagem, impossibilitou a validação da carga suportada pelo sistema, nomeadamente em relação a consultas concorrentes. Desenvolvimentos futuros passariam por priorizar testes de carga ao sistema, alimentando-o com eventos concorrentes na fonte, testando resposta a picos de utilização, carregando maiores volumes de dados e testando consultas simultâneas por parte de diferentes utilizadores.

### **5.3 Apreciação final**

Em retrospectiva, o desenvolvimento deste projeto decorreu conforme o planeado, tendo sido atingidos, em grande parte, os objetivos propostos e detetadas limitações e extensões do sistema desenvolvido que deverão ser avaliadas em desenvolvimentos futuros.

Relativamente aos contributos do projeto para a DataCentric, verificou-se já interesse por parte de um cliente em utilizar a tecnologia Spark Streaming e Apache Kafka para uma aplicação capaz de melhorar a visibilidade de inventário em tempo real. Neste projeto já fui envolvida para fornecer apoio na fase de análise de requisitos e de desenvolvimento inicial, pelo conhecimento que obtive nestas tecnologias através do projeto de estágio.

Prevê-se que no futuro este tipo de projetos sejam solicitados por outros clientes da empresa. Neste contexto, a existência do presente estudo será uma mais valia como base tecnológica para os desenvolvedores da empresa utilizarem como referência.

Já para a comunidade científica, foi referido na secção 2.4.1 do Estado da Arte que os artigos académicos existentes em relação às tecnologias de processamento de dados abordam ferramentas já em desuso no contexto empresarial. Como resposta a esse problema, espera-se que o presente relatório possa contribuir para a atualização do conhecimento e discurso científico nesta área.

A nível pessoal, o conhecimento que obti com o desenvolvimento deste projeto abriu várias oportunidades dentro da própria empresa de estágio. Foi colocado o desafio de continuar o desenvolvimento de um projeto existente relativo a um algoritmo de previsão de vendas, elaborado inteiramente em Spark. A familiaridade de já tinha com o Spark Streaming, muito semelhante em termos de utilização e sintaxe, permitiu a minha rápida integração no projeto e resultou num produto finalizado, que à data de escrita deste relatório está já em fase de testes para entrada em produção.

Após esta experiência acredito ter obtido as competências necessárias para uma entrada com sucesso no mercado de trabalho como engenheira informática. A nível técnico, obtive familiaridade com conceitos de engenharia de dados, que me eram desconhecidos à data de início da escrita do relatório, e a nível social obtive experiência valiosa relativa aos métodos de organização do trabalho e às interações com colegas e clientes.

## Referências

- [1] F. Provost e T. Fawcett, «Data Science and its Relationship to Big Data and Data-Driven Decision Making», *Big Data*, vol. 1, n. 1, pp. 51–59, Mar. 2013, doi: 10.1089/BIG.2013.1508/ASSET/IMAGES/LARGE/FIGURE1.JPEG.
- [2] Farmer Donald, «6 Top Business Benefits of Real-Time Data Analytics», *TechTarget*, 11 de Janeiro de 2021. <https://www.techtarget.com/searchbusinessanalytics/feature/6-top-business-benefits-of-real-time-data-analytics> (acedido 28 de Março de 2023).
- [3] R. Habeeb, F. Nasaruddin, e A. Gani, «Real-time big data processing for anomaly detection: A survey», *Int J Inf Manage*, vol. 45, pp. 289–307, 2019, Acedido: 13 de Março de 2023. [Em linha]. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0268401218301658>
- [4] Y. Beng Leau, W. Khong Loo, W. Yip Tham, e S. Fun Tan, «Software Development Life Cycle AGILE vs Traditional Approaches».
- [5] «What is ETL Pipeline? Process, Considerations, and Examples», *ProjectPro*, 28 de Fevereiro de 2023. [https://www.projectpro.io/article/how-to-build-etl-pipeline-example/526#mcetoc\\_1ghj67rdj1v](https://www.projectpro.io/article/how-to-build-etl-pipeline-example/526#mcetoc_1ghj67rdj1v) (acedido 15 de Março de 2023).
- [6] «Data Engineering: Data Warehouse, Data Pipeline and Data Engineer Role | AltexSoft», *Blog Altexsoft*, 13 de Março de 2023. <https://www.altexsoft.com/blog/datascience/what-is-data-engineering-explaining-data-pipeline-data-warehouse-and-data-engineer-role/> (acedido 28 de Março de 2023).
- [7] J. Reis e M. Housley, «Fundamentals of Data Engineering».
- [8] A. Nambiar, D. M.-B. D. and C. Computing, e undefined 2022, «An Overview of Data Warehouse and Data Lake in Modern Enterprise Data Management», *mdpi.com*, 2022, doi: 10.3390/bdcc6040132.
- [9] S. S. Conn, «OLTP and OLAP data integration: A review of feasible implementation methods and architectures for real time data analysis», *Proceedings. IEEE SoutheastCon*, 2005, doi: 10.1109/SECON.2005.1423297.
- [10] G. Satyanarayana Reddy, R. Srinivasu, M. Poorna, C. Rao, e S. R. Rikkula, «DATA WAREHOUSING, DATA MINING, OLAP AND OLTP TECHNOLOGIES ARE ESSENTIAL ELEMENTS TO SUPPORT DECISION-MAKING PROCESS IN INDUSTRIES», *IJCSE*

- International Journal on Computer Science and Engineering*, vol. 02, n. 09, pp. 2865–2873, 2010, Acedido: 6 de Abril de 2023. [Em linha]. Disponível em: <http://pwp.starnetinc.com/larryg/articles.html>
- [11] «What Is a Lakehouse? | Databricks Blog». <https://www.databricks.com/blog/2020/01/30/what-is-a-data-lakehouse.html> (acedido 26 de Junho de 2023).
- [12] «Event-driven Design vs. Timetable Scheduling: What’s the Difference?», *CloverDX*, 18 de Abril de 2017. <https://www.cloverdx.com/blog/event-driven-design-vs-timetable-scheduling> (acedido 7 de Abril de 2023).
- [13] Frempong Clifford, «Data Engineering: Scheduling Data», *Medium*, 9 de Julho de 2021. <https://clifflool.medium.com/data-engineering-scheduling-data-dfef321731f6> (acedido 7 de Abril de 2023).
- [14] Levy Eran, «Batch vs Stream vs Microbatch Processing: A Cheat Sheet | Upsolver», *upsolver*, 21 de Janeiro de 2021. <https://www.upsolver.com/blog/batch-stream-a-cheat-sheet> (acedido 8 de Abril de 2023).
- [15] «What are the benefits and challenges of using micro-batching over event-driven processing?», *LinkedIn*. <https://www.linkedin.com/advice/0/what-benefits-challenges-using-micro-batching-over> (acedido 8 de Abril de 2023).
- [16] A. M. Garcia, D. Griebler, C. Schepke, e L. G. Fernandes, «Micro-batch and data frequency for stream processing on multi-cores», *Journal of Supercomputing*, 2023, doi: 10.1007/S11227-022-05024-Y.
- [17] Tanner Matt, «MySQL Change Data Capture ( CDC ) : The Definitive Guide», *Arcion*, 2 de Dezembro de 2022. <https://www.arcion.io/learn/mysql-cdc> (acedido 7 de Abril de 2023).
- [18] «The Comprehensive Guide to SQL Triggers», *SQL Tutorial*. <https://www.sqltutorial.org/sql-triggers/> (acedido 7 de Abril de 2023).
- [19] «On Premise vs. Cloud: Key Differences, Benefits and Risks | Cleo». <https://www.cleo.com/blog/knowledge-base-on-premise-vs-cloud> (acedido 8 de Abril de 2023).
- [20] Siwiecki Stefan, «The battle of costs – On-premises vs Cloud-based solutions», *Columbus*, 13 de Janeiro de 2021. <https://www.columbusglobal.com/en-us/blog/on-premises-vs-cloud-based-solutions> (acedido 8 de Abril de 2023).

- [21] «Cloud Vs On Premise Software: Which is Best For Your Business?», *Xperience*. <https://www.xperience-group.com/news-item/cloud-vs-on-premise-software/> (acedido 8 de Abril de 2023).
- [22] L. Qian, Z. Luo, Y. Du, e L. Guo, «Cloud computing: An overview», *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5931 LNCS, pp. 626–631, 2009, doi: 10.1007/978-3-642-10665-1\_63.
- [23] P. Alamdari, N. Navimipour, ... M. H.-I., e undefined 2020, «A systematic study on the recommender systems in the E-commerce», *ieeexplore.ieee.org*, Acedido: 26 de Junho de 2023. [Em linha]. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9118884/>
- [24] P. Covington, J. Adams, e E. Sargin, «Deep neural networks for youtube recommendations», *RecSys 2016 - Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 191–198, Set. 2016, doi: 10.1145/2959100.2959190.
- [25] «Powered by AI: Instagram’s Explore recommender system», *Meta AI*, 25 de Novembro de 2019. <https://ai.facebook.com/blog/powered-by-ai-instagrams-explore-recommender-system/> (acedido 14 de Abril de 2023).
- [26] C. Y. Wu, C. V. Alvino, A. J. Smola, e J. Basilico, «Using navigation to improve recommendations in real-time», *RecSys 2016 - Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 341–348, Set. 2016, doi: 10.1145/2959100.2959174.
- [27] A. Felfernig, M. Jeran, G. Ninaus, F. Reinfrank, e S. Reiterer, «Toward the Next Generation of Recommender Systems: Applications and Research Challenges», *Multimedia Services in Intelligent Environments: Recommendation Services*, vol. 8767, pp. 81–98, 2013, Acedido: 14 de Abril de 2023. [Em linha]. Disponível em: [www.springer.com](http://www.springer.com).
- [28] «Overview of collaborative filtering algorithms», *Medium*, 14 de Abril de 2021. <https://medium.com/analytics-vidhya/overview-of-collaborative-filtering-algorithms-9a76d2eb861b> (acedido 14 de Abril de 2023).
- [29] R. H. Singh, S. Maurya, T. Tripathi, T. Narula, e G. Srivastav, «Movie Recommendation System using Cosine Similarity and KNN», *Int J Eng Adv Technol*, 2020, doi: 10.35940/ijeat.E9666.069520.

- 
- [30] B. Chandramouli, J. J. Levandoski, § Ahmed Eldawy, e M. F. Mokbel, «StreamRec: A Real-Time Recommender System», Acedido: 25 de Abril de 2023. [Em linha]. Disponível em: <http://www.netflix.com>.
- [31] B. Chandramouli, J. J. Levandoski, § Ahmed Eldawy, e M. F. Mokbel, «StreamRec: A Real-Time Recommender System», Acedido: 30 de Abril de 2023. [Em linha]. Disponível em: <http://www.netflix.com>.
- [32] Y. Huang, B. Cui, W. Zhang, J. Jiang, e Y. Xu, «Tencentrec: Real-time stream recommendation in practice», *Proceedings of the ACM SIGMOD International Conference on Management of Data*, vol. 2015-May, pp. 227–238, Mai. 2015, doi: 10.1145/2723372.2742785.
- [33] V. Ta, C. Liu, G. N.-2016 I. international, e undefined 2016, «Big data stream computing in healthcare real-time analytics», *ieeexplore.ieee.org*, Acedido: 30 de Abril de 2023. [Em linha]. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7529531/>
- [34] Y. N. Malek *et al.*, «On the use of IoT and Big Data Technologies for Real-time Monitoring and Data Processing», *Procedia Comput Sci*, vol. 113, pp. 429–434, Jan. 2017, doi: 10.1016/J.PROCS.2017.08.281.
- [35] Y. Abakarim, M. Lahby, e A. Attioui, «An Efficient Real Time Model For Credit Card Fraud Detection Based On Deep Learning», vol. 18, 2018, doi: 10.1145/3289402.
- [36] S. Patil, V. Nemade, e P. K. Soni, «Predictive Modelling For Credit Card Fraud Detection Using Data Analytics», *Procedia Comput Sci*, vol. 132, pp. 385–395, Jan. 2018, doi: 10.1016/J.PROCS.2018.05.199.
- [37] A. Mătăcuță e C. Popa, «Big Data Analytics: Analysis of Features and Performance of Big Data Ingestion Tools», *Informatica Economică*, vol. 22, n. 2, 2018, doi: 10.12948/issn14531305/22.2.2018.03.
- [38] J. Alwidian, S. A. Rahman, M. Gnaim, F. Al-Taharwah, e K. Abdullah, «Big Data Ingestion and Preparation Tools», *Mod Appl Sci*, vol. 14, n. 9, 2020, doi: 10.5539/mas.v14n9p12.
- [39] F. Gurcan, M. Berigel, e F. Gürcan, «Real-Time Processing of Big Data Streams: Lifecycle, Tools, Tasks, and Challenges», doi: 10.1109/ISMSIT.2018.8567061.
- [40] Molander Oliver, «The rise and future of data engineering — what’s it all about?», *Validio*, 26 de Outubro de 2021. <https://validio.io/blog/the-rise-and-future-of-data-engineering--whats-it> (acedido 21 de Abril de 2023).

- 
- [41] Orr Einat, «The State of Data Engineering 2022», *lakeFS*, 20 de Junho de 2022. <https://lakefs.io/blog/the-state-of-data-engineering-2022/> (acedido 17 de Abril de 2023).
- [42] Tanner Matt, «8 Best CDC Tools of 2023», *Arcion*, 11 de Janeiro de 2023. <https://www.arcion.io/blog/cdc-tools#other-change-data-capture-tools-> (acedido 22 de Abril de 2023).
- [43] «Data Sources | Connector Directory | Fivetran». <https://www.fivetran.com/connectors> (acedido 22 de Abril de 2023).
- [44] A. C. Ikegwu, H. F. Nweke, C. V. Anikwe, U. R. Alo, e O. R. Okonkwo, «Big data analytics for data-driven industry: a review of data sources, tools, challenges, solutions, and research directions», *Cluster Comput*, vol. 25, n. 5, pp. 3343–3387, Out. 2022, doi: 10.1007/S10586-022-03568-5.
- [45] Berglund Tim, «Apache Kafka®: Basic Concepts, Architecture, and Examples», *Confluent*. <https://developer.confluent.io/learn-kafka/apache-kafka/events/> (acedido 21 de Abril de 2023).
- [46] V. Jain, P. Jana, R. Kumar, e M. Sagarkar, «A Study of Distributed Event Streaming & Publish-Subscribe Systems», 2021, doi: 10.13140/RG.2.2.28795.80160.
- [47] «Kafka Connectors - Confluent Documentation». [https://docs.confluent.io/kafka-connectors/self-managed/kafka\\_connectors.html](https://docs.confluent.io/kafka-connectors/self-managed/kafka_connectors.html) (acedido 22 de Abril de 2023).
- [48] H. Isah, T. Abughofa, S. Mahfuz, ... D. A.-I., e undefined 2019, «A survey of distributed data stream processing frameworks», *ieeexplore.ieee.org*, Acedido: 21 de Abril de 2023. [Em linha]. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8864052/>
- [49] Deshpande Shruti, «Apache Kafka Vs Apache Spark: What are the differences?», *Knowledge Hut*, 17 de Fevereiro de 2023. <https://www.knowledgehut.com/blog/big-data/kafka-vs-spark> (acedido 21 de Abril de 2023).
- [50] Prakash Chandan, «Spark Streaming vs Flink vs Storm vs Kafka Streams vs Samza : Choose Your Stream Processing Framework», *Medium*, 1 de Maio de 2018. <https://medium.com/@chandanbaranwal/spark-streaming-vs-flink-vs-storm-vs-kafka-streams-vs-samza-choose-your-stream-processing-91ea3f04675b> (acedido 21 de Abril de 2023).



- 
- [51] Indrasiri Kasun, «Benchmarking Azure Event Hubs Premium for Kafka and AMQP workloads», *Microsoft Community Hub*, 24 de Março de 2022. <https://techcommunity.microsoft.com/t5/messaging-on-azure-blog/benchmarking-azure-event-hubs-premium-for-kafka-and-amqp/ba-p/3377483> (acedido 22 de Abril de 2023).
- [52] A. Oussous, F. Z. Benjelloun, A. Ait Lahcen, e S. Belfkih, «Big Data technologies: A survey», *Journal of King Saud University - Computer and Information Sciences*, vol. 30, n. 4, pp. 431–448, Out. 2018, doi: 10.1016/J.JKSUCI.2017.06.001.
- [53] «Introduction to Blob (object) Storage - Azure Storage | Microsoft Learn». <https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction> (acedido 23 de Abril de 2023).
- [54] «Azure Data Lake Storage Gen2 Introduction - Azure Storage | Microsoft Learn». <https://learn.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-introduction#built-on-azure-blob-storage> (acedido 23 de Abril de 2023).
- [55] A. Oussous, F. Z. Benjelloun, A. Ait Lahcen, e S. Belfkih, «Big Data technologies: A survey», *Journal of King Saud University - Computer and Information Sciences*, vol. 30, n. 4, pp. 431–448, Out. 2018, doi: 10.1016/J.JKSUCI.2017.06.001.
- [56] «Performance testing Impala and Spark on Azure Data Lake Store vs. HDFS», *Cazena*. <https://cazena.com/performance-testing-impala-and-spark-azure-data-lake-store-vs-hdfs/> (acedido 23 de Abril de 2023).
- [57] «HDFS vs. Cloud Storage: Pros, cons and migration tips | Google Cloud Blog». <https://cloud.google.com/blog/products/storage-data-transfer/hdfs-vs-cloud-storage-pros-cons-and-migration-tips> (acedido 23 de Abril de 2023).
- [58] J. Shi *et al.*, «Clash of the Titans: MapReduce vs. Spark for Large Scale Data Analytics», 2150, Acedido: 23 de Abril de 2023. [Em linha]. Disponível em: <http://aws.amazon.com/ec2/instance-types/>
- [59] «Apache Ignite Documentation», *Apache Ignite*. <https://ignite.apache.org/docs/latest/> (acedido 3 de Maio de 2023).
- [60] C. S. Stan, A. E. Pandelica, V. A. Zamfir, R. G. Stan, e C. Negru, «Apache spark and apache ignite performance analysis», *Proceedings - 2019 22nd International Conference on Control Systems and Computer Science, CSCS 2019*, pp. 726–733, Mai. 2019, doi: 10.1109/CSCS.2019.00129.

- 
- [61] C. Widanage *et al.*, «High Performance Data Engineering Everywhere», Acedido: 25 de Abril de 2023. [Em linha]. Disponível em: <https://github.com/cylondata/cylon>
  - [62] B. Johnson e A. S. Chandran, «COMPARISON BETWEEN PYTHON, JAVA AND R PROGRAMMING LANGUAGE IN MACHINE LEARNING», *www.irjmets.com @International Research Journal of Modernization in Engineering*, vol. 3288, Acedido: 27 de Junho de 2023. [Em linha]. Disponível em: [www.irjmets.com](http://www.irjmets.com)
  - [63] «TIOBE Index - TIOBE». <https://www.tiobe.com/tiobe-index/> (acedido 25 de Abril de 2023).
  - [64] «PYPL PopularitY of Programming Language index». <https://pypl.github.io/PYPL.html> (acedido 25 de Abril de 2023).
  - [65] C. Vazquez e G. Simões, «Engenharia de Requisitos: software orientado ao negócio», 2016, Acedido: 18 de Junho de 2023. [Em linha]. Disponível em: [https://books.google.com/books?hl=pt-PT&lr=&id=gA7kDAAAQBAJ&oi=fnd&pg=PA74&dq=Vazquez,+Carlos%3B+Sim%C3%B5es,+Guilherme+\(2016\).+Engenharia+de+Requisitos:+Software+Orientado+ao+Neg%C3%B3cio.+%5BS.I.%5D:+Brasport&ots=sOes4OV\\_xQ&sig=sl7YSy3uqZrExb7rvKX8lhRdKXI](https://books.google.com/books?hl=pt-PT&lr=&id=gA7kDAAAQBAJ&oi=fnd&pg=PA74&dq=Vazquez,+Carlos%3B+Sim%C3%B5es,+Guilherme+(2016).+Engenharia+de+Requisitos:+Software+Orientado+ao+Neg%C3%B3cio.+%5BS.I.%5D:+Brasport&ots=sOes4OV_xQ&sig=sl7YSy3uqZrExb7rvKX8lhRdKXI)
  - [66] «Use-Case Model - Javatpoint». <https://www.javatpoint.com/use-case-model> (acedido 14 de Junho de 2023).
  - [67] P. Maio, «Requirements Engineering».
  - [68] P. Pu, L. Chen, R. Hu, P. Pu, R. Hu, e L. Chen, «Evaluating recommender systems from the user's perspective: survey of the state of the art», *doc.rero.ch*, vol. 22, pp. 317–355, 2012, doi: 10.1007/s11257-011-9115-7.
  - [69] «What is a Medallion Architecture?» <https://www.databricks.com/glossary/medallion-architecture> (acedido 18 de Junho de 2023).
  - [70] M. Zafar Iqbal Karmani, G. Mustafa, N. Sarwar, e S. Hamza Wajid, «A Review of Star Schema and Snowflakes Schema», doi: 10.1007/978-981-15-5232-8\_12.
  - [71] «Slowly Changing Dimensions», *Oracle*. [https://www.oracle.com/webfolder/technetwork/tutorials/obe/db/10g/r2/owb/owb10gr2\\_gs/owb/lesson3/slowlychangingdimensions.htm](https://www.oracle.com/webfolder/technetwork/tutorials/obe/db/10g/r2/owb/owb10gr2_gs/owb/lesson3/slowlychangingdimensions.htm) (acedido 18 de Junho de 2023).

- 
- [72] «Near real-time lakehouse data processing - Azure Architecture Center | Microsoft Learn». <https://learn.microsoft.com/en-us/azure/architecture/example-scenario/data/real-time-lakehouse-data-processing> (acedido 25 de Maio de 2023).
- [73] «Choose a data storage technology - Azure Architecture Center | Microsoft Learn». <https://learn.microsoft.com/en-us/azure/architecture/data-guide/technology-choices/data-storage> (acedido 25 de Maio de 2023).
- [74] «MySQL :: MySQL 8.0 Reference Manual :: 1.2.1 What is MySQL?». <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html> (acedido 30 de Maio de 2023).
- [75] «Reference Documentation». <https://debezium.io/documentation/> (acedido 30 de Maio de 2023).
- [76] «Apache Kafka». <https://kafka.apache.org/documentation/#gettingStarted> (acedido 30 de Maio de 2023).
- [77] «Get Docker | Docker Documentation». <https://docs.docker.com/get-docker/> (acedido 30 de Maio de 2023).
- [78] «Apache ZooKeeper». <https://zookeeper.apache.org/> (acedido 30 de Maio de 2023).
- [79] «What is Azure Event Hubs? - a Big Data ingestion service - Azure Event Hubs | Microsoft Learn». <https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-about> (acedido 30 de Maio de 2023).
- [80] «What is Azure Databricks? - Azure Databricks | Microsoft Learn». <https://learn.microsoft.com/en-us/azure/databricks/introduction/> (acedido 30 de Maio de 2023).
- [81] «Apache Spark™ - Unified Engine for large-scale data analytics». <https://spark.apache.org/> (acedido 30 de Maio de 2023).
- [82] «O que é o Delta Lake? - Azure Databricks | Microsoft Learn». <https://learn.microsoft.com/pt-pt/azure/databricks/delta/> (acedido 30 de Maio de 2023).
- [83] «Java Platform SE 8 Documentation», *Oracle*. <https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html> (acedido 8 de Abril de 2023).

## Anexo A – Lista de testes de qualidade efetuados

#CASO DE USO	CENÁRIO	#CASO DE TESTE	OBJETO DE TESTE	CONDIÇÕES/FILTROS	ORIGEM	DESTINO	AMBIENTE ORIGEM	AMBIENTE DESTINO	RESULTADO ESPERADO	RESULTADO TESTE	OBSERVAÇÕES
DCRA-3	Criação do esquema "olist-dataset", "olist" e dos scripts de carregamento de dados	#1	Validar se as tabelas "sellers", "customers", "orders", "order_items" e "products" existem em ambos os esquemas e os dados foram inseridos com sucesso	--	MySQL	MySQL	Docker	Docker	Tabelas existem e possuem dados	PASS	--
DCRA-3	Criação de ligação da base de dados MySQL ao Kafka	#2	Validar se os tópicos Kafka correspondentes às tabelas "sellers", "customers", "orders", "order_items" e "products" existem e estão populados	--	MySQL	Kafka	Docker	Docker	Tópicos existem e possuem dados	PASS	--
DCRA-3	Criação de ligação do Kafka à cloud	#3	Validar se os tópicos Kafka correspondentes às tabelas "sellers", "customers", "orders", "order_items" e "products" estão presentes na interface do Event Hubs	--	Kafka	Event Hub	Docker	Azure	Tópicos são visíveis na interface cloud	PASS	--

#CASO DE USO	CENÁRIO	#CASO DE TESTE	OBJETO DE TESTE	CONDIÇÕES/FILTROS	ORIGEM	DESTINO	AMBIENTE ORIGEM	AMBIENTE DESTINO	RESULTADO ESPERADO	RESULTADO TESTE	OBSERVAÇÕES
DCRA-3	Execução do notebook de ingestão Databricks	#4	Validar se a execução do notebook correu com sucesso e gerou a visualização dos dataframes de ingestão, com os dados da tabela MySQL	--	MySQL	Databricks	Docker	Azure	Dataframes possuem dados inseridos localmente no momento da execução	PASS	--
DCRA-4	Criação do esquema "olist-raw" e inserção dos dados provenientes da base de dados local	#5	Validar se as tabelas "sellers", "customers", "orders", "order_items" e "products" existem no esquema e os dados foram inseridos com sucesso	--	MySQL	Data Lake Storage	Docker	Azure	Tabelas existem e possuem dados	PASS	--
DCRA-4	Modelação do esquema "olist_raw" de acordo com o respectivo modelo de dados	#6	Validar se as tabelas possuem as colunas e tipos de dados definidos pelo modelo de dados "olist_raw"	--	MySQL	Data Lake Storage	Docker	Azure	Colunas existem e têm o tipo de dados correto	PASS	--
DCRA-4	Criação do esquema "olist-stg" e inserção dos dados provenientes da base de dados local	#7	Validar se as tabelas "dim_sellers", "dim_customers", "dim_orders", "fac_order_items", "dim_products" e "fac_order_items_agg" existem no esquema e os dados foram inseridos com sucesso	--	MySQL	Data Lake Storage	Docker	Azure	Tabelas existem e possuem dados	PASS	--

#CASO DE USO	CENÁRIO	#CASO DE TESTE	OBJETO DE TESTE	CONDIÇÕES/FILTROS	ORIGEM	DESTINO	AMBIENTE ORIGEM	AMBIENTE DESTINO	RESULTADO ESPERADO	RESULTADO TESTE	OBSERVAÇÕES
DCRA-4	Modelação do esquema "olist_stg" de acordo com o respetivo modelo de dados	#8	Validar se as tabelas possuem as colunas e tipos de dados definidos pelo modelo de dados "olist_stg"	--	MySQL	Data Lake Storage	Docker	Azure	Colunas existem e têm o tipo de dados correto	PASS	--
DCRA-4	Criação do esquema "olist-dw" e inserção dos dados provenientes da base de dados local	#9	Validar se as tabelas "dim_sellers", "dim_customers", "fac_order_items" e "dim_products" existem no esquema e os dados foram inseridos com sucesso	--	MySQL	Data Lake Storage	Docker	Azure	Tabelas existem e possuem dados	PASS	--
DCRA-4	Modelação do esquema "olist_dw" de acordo com o respetivo modelo de dados	#10	Validar se as tabelas possuem as colunas e tipos de dados definidos pelo modelo de dados "olist_dw"	--	MySQL	Data Lake Storage	Docker	Azure	Colunas existem e têm o tipo de dados correto	PASS	--
DCRA-4	Inserção de registos nos esquemas "olist_raw", "olist_stg" e "olist_dw"	#11	Validar se o mesmo registo apresenta dados consistentes nos esquemas "olist_raw", "olist_stg" e "olist_dw"	--	MySQL	Data Lake Storage	Docker	Azure	Dados existem e são equivalente em todos os esquemas	PASS	--

#CASO DE USO	CENÁRIO	#CASO DE TESTE	OBJETO DE TESTE	CONDIÇÕES/FILTROS	ORIGEM	DESTINO	AMBIENTE ORIGEM	AMBIENTE DESTINO	RESULTADO ESPERADO	RESULTADO TESTE	OBSERVAÇÕES
DCRA-5	Criação do esquema "olist_rs" e inserção dos dados correspondentes a todos os clientes e produtos existentes.	#12	Validar se as tabelas "cf_matrix" e "cf_similarity_matrix" existem no esquema e os dados foram inseridos com sucesso	--	Data Lake Storage	Data Lake Storage	Azure	Azure	Tabelas existem e possuem dados	PASS	--
DCRA-5	Alteração do esquema "olist_rs" quando novos produtos, clientes ou vendas são registrados.	#13	Validar se as colunas da tabela "cf_matrix" correspondem a todos os clientes e produtos registrados.	--	MySQL	Data Lake Storage	Docker	Azure	Colunas existem e têm os dados corretos	PASS	--
DCRA-5	Alteração do esquema "olist_rs" quando novos produtos, clientes ou vendas são registrados.	#14	Validar se os valores da tabela "cf_matrix" correspondem a todas as unidades vendidas para cada par cliente/produto.	--	MySQL	Data Lake Storage	Docker	Azure	Valores existem e correspondem à soma correta	PASS	--
DCRA-5	Cálculo do coeficiente de similaridade na tabela "cf_similarity_matrix"	#15	Validar se os valores da tabela "cf_similarity_matrix" correspondem ao coeficiente de similaridade correto para cada par cliente/cliente.	--	Data Lake Storage	Data Lake Storage	Azure	Azure	Valores existem e correspondem ao cálculo correto	PASS	--

