UNIVERSIDADE BEIRA INTERIOR
Engenharia

Camada de Data Warehouse num sistema de Big Data da Critical TechWorks

João Duarte Baptista Marques

Relatório do projeto de estágio
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor João Carlos Raposo Neves
Co-orientador: Prof. Doutor Pedro Ricardo Morais Inácio
Co-orientador: David Miguel Varrasquinho Guerreiro
Co-orientador: João Carlos Cordeiro Santos Gonçalves

Covilhã, junho de 2021

Agradecimentos

A elaboração deste relatório não seria possível sem o apoio das pessoas que me acompanham. Assim sendo, pretendo agradecer a todos os que sempre me apoiaram e contribuíram para a realização e concretização desta etapa final na minha formação académica, o Mestrado em Engenharia Informática na Universidade da Beira Interior.

Em primeiro lugar, quero agradecer à minha família, em especial aos meus pais e avós, por todo o apoio, esforço e dedicação que me deram ao longo de toda a minha vida e especialmente nesta etapa da minha vida académica.

Quero agradecer ao orientador, por toda a orientação, simpatia e disponibilidade no desenvolvimento deste projeto de estágio, manifestando sempre as suas opiniões pertinentes e enriquecedoras. Aos meus mentores e amigos da Critical TechWorks, David Guerreiro e João Gonçalves pela orientação, atenção, compreensão e simpatia. Sem esquecer todo o pessoal da Critical TechWorks pelo suporte e auxilio.

Por fim, quero agradecer aos meus colegas e amigos que me acompanharam neste percurso, por todo o companheirismo, amizade, entreajuda e união.

Resumo

Este trabalho visa estudar e documentar a análise de serviços de *Data Warehouse* para a sua integração num sistema de *Big Data* em desenvolvimento pela *Critical TechWorks* (CTW), com o objetivo de ser implementado no sistema financeiro da *Bayerische Motoren Werke* (BMW) no Reino Unido, dando suporte às decisões de negócio, personalizando e melhorando assim o serviço prestado pela BMW.

Palavras-chave

Big data, *Data Warehouse*, CTW, BMW, *Amazon Web Services* (AWS) Athena, AWS Aurora, AWS RedShift.

Abstract

This work aims to study and report an analysis of Data Warehouse services to be intgrated on a Big Data system in development by CTW. This system has the main objective of being integrated on the BMW financial system on United Kingdom, giving support to business decisions, giving better and custom service to BMW clients.

Keywords

Big Data,*Data Warehouse*,CTW,BMW,AWS Athena,AWS Aurora, AWS RedShift.

Conteúdo

1 Introdução **1**
1.1 Caracterização da empresa . . . 1
1.2 Descrição do Problema . . . 3
1.3 Organização do Documento . . . 5

2 Estado da Arte **7**
2.1 Data Warehouse . . . 7
2.2 AWS Athena . . . 9
2.2.1 Integração . . . 9
2.2.2 Desempenho . . . 9
2.2.3 Segurança . . . 9
2.2.4 Custos . . . 10
2.3 AWS Aurora . . . 10
2.3.1 Integração . . . 11
2.3.2 Desempenho . . . 11
2.3.3 Segurança . . . 11
2.3.4 Custos . . . 11
2.4 AWS RedShift . . . 12
2.4.1 Integração . . . 12
2.4.2 Desempenho . . . 13
2.4.3 Segurança . . . 13
2.4.4 Custos . . . 13
2.5 Google BigQuery . . . 14
2.5.1 Integração . . . 14
2.5.2 Desempenho . . . 15
2.5.3 Segurança . . . 15
2.5.4 Custos . . . 15
2.6 Microsoft Azure Synapse Analytics . . . 15
2.6.1 Integração . . . 16
2.6.2 Desempenho . . . 16
2.6.3 Segurança . . . 17
2.6.4 Custos . . . 17
2.7 Conclusão . . . 18

3 Método Proposto **21**
3.1 Abordagem Proposta . . . 21
3.1.1 SCRUM . . . 21
3.2 Fases de desenvolvimento . . . 23
3.3 Planificação do Trabalho . . . 24

4 Tecnologias e Ferramentas Usadas **27**
4.1 AWS *Simple Storage Services* (S3) . . . 27
4.2 AWS *Identity and Access Management* (IAM) . . . 29
4.3 AWS Glue . . . 29
4.3.1 Python3 e PySpark . . . 31
4.3.2 Workflow . . . 34
4.4 AWS Kinesis . . . 34
4.5 AWS Lambda . . . 39
4.6 Terraform . . . 41
4.7 Bitbucket . . . 42
4.8 AWS CodeBuild . . . 42
4.9 AWS CodePipeline . . . 43

5 Sistema *Big Data* **45**
5.1 Fontes de dados . . . 45
5.2 Modelação de dados . . . 46
5.2.1 *Online Transactional Processing* (OLTP) e *Online Analytical Processing* (OLAP) . . . 46
5.2.2 Tipos de tabelas em OLAP . . . 47
5.2.3 Esquemas OLAP . . . 47
5.3 Processo *Extract Transform Load* (ETL) . . . 49
5.3.1 Primeira fase . . . 49
5.3.2 Segunda fase . . . 49
5.4 *Continuous Integration* (CI)/*Continuous Deployment* (CD) . . . 50

6 Comparação e análise dos serviços de *Data Warehouse* escolhidos **55**
6.1 Integração e configuração . . . 55
6.1.1 AWS Athena . . . 55
6.1.2 AWS Aurora *serverless* . . . 56
6.1.3 AWS RedShift e Spectrum . . . 57
6.2 Desempenho . . . 58
6.2.1 Desempenho entre configurações e serviços . . . 58
6.2.2 *Cache* . . . 62
6.2.3 Ficheiros Parquet . . . 64
6.3 Propriedades de bases de dados . . . 64
6.4 Funcionalidades adicionais . . . 66
6.5 Conclusão . . . 67

7 Conclusão **69**
7.1 Conclusões Principais . . . 69
7.2 Trabalho Futuro . . . 70

Bibliografia **71**

Lista de Figuras

1.1 Apresentação de o novo sistema iDrive presente no painel panorâmico dos novos automóveis iX, desenhada e desenvolvido na CTW [BMW21a]. . . . 3
1.2 Diagrama representativo de um sistema de *Big Data* completo com a camada de *Data Warehouse* assinalada pela caixa vermelha. 4

2.1 Gráfico com uma estimativa de volume de dados existentes ao longo dos anos em *Zettabyte* (ZB) [JRRR16]. 7
2.2 Gráfico do *Magic Quadrant* representativo da relação entre a visão e qualidade geral do produto *cloud* [Gar20]. 18

3.1 Gráfico representativo das etapas de desenvolvimento *software* utilizando a *framework* Scrum. 23
3.2 Mapa de Gantt representativo da planificação das fases do projeto de estágio com os seus respetivos períodos de tempo. 25

4.1 Diagrama representativo da relação entre as componentes de Glue no processo de ingestão de dados [Ama21a]. 32
4.2 Exemplo de um Workflow de Glue onde é apresentado o grafo com a cadeia de execução das componentes do Glue no processo ETL [Ama21b] 34
4.3 Exemplo de uma *pipeline* criada através de CodePipeline para o processo CD [AWS21]. 43

5.1 Página web com a seleção do modelo BMW a escolher pelo cliente na compra do automóvel [BMW21b]. 46
5.2 Representação da relação de tabelas factuais e de dimensão, num modelo OLAP com um esquema em estrela. 48
5.3 Representação da relação de tabelas factuais e de dimensão, num modelo OLAP com um esquema em floco de neve. 48
5.4 Diagrama de extração de dados das fontes através da execução de *jobs* em AWS Glue. 49
5.5 Diagrama de extração de dados das fontes em tempo real através da execução de uma *stream* em AWS Kinesis. 50
5.6 Diagrama do processo de transformação dos dados (ETL) usando AWS Glue. 50
5.7 Processo de configuração dos ambientes de desenvolvimento através de código Terraform e CodeBuild. 51
5.8 Processo de integração continua de código Python. 52
5.9 Exemplo de uma *pipeline* no serviço CodeBuild para o processo de CI/CD. 53

6.1 Gráfico comparativo de desempenho entre execução de *queries* em RedShift ou através da utilização de Spectrum. 59

6.2 Gráfico comparativo de desempenho entre execução de *queries* na *interface* Athena ou num *cluster* Redshift. 60
6.3 Gráfico comparativo de desempenho entre várias configurações de Aurora *serverless*. 62
6.4 Gráfico comparativo de desempenho entre os sistemas de *cache* dos serviços em análise. 63
6.5 Gráfico de comparação de desempenho uma *query* executada em Athena em diferentes formatos de ficheiro. 65
6.6 Resultado visual produzido automáticamente por RedShift após a execução de uma *query* com uma contagem de agregação da coluna ”Column_1”. . . 67

Lista de Tabelas

2.1 Tabela de comparação das características chave entre os diferentes serviços de *Data Warehouse* 19

6.1 Tempos de execução para a comparação de desempenho entre os serviços Spectrum e RedShift nativo. 59
6.2 Tempos de execução para a comparação de desempenho entre os serviços Spectrum e RedShift nativo. 60
6.3 Tempos de execução para a comparação de desempenho entre várias configurações de Aurora *serverless*. 61
6.4 Tempos de execução para a comparação de desempenho entre os serviços Aurora e Athena. 62
6.5 Tempos de execução consecutiva de uma *query* para a testagem do desempenho do sistema de *cache* de cada serviço. 63
6.6 Tempo de execução e quantidade de dados utilizados para a comparação de entre a utilização de vários formatos de dados. 64

Lista de Acrónimos

BI *Business intelligence*

CD *Continuous Deployment*

CI *Continuous Integration*

GB *Gigabyte*

IP *Internet Protocol*

kB *Quilobyte*

MB *Megabyte*

PB *Petabyte*

S3 *Simple Storage Services*

TB *Terabyte*

ZB *Zettabyte*

ACL *Access Control List*

AES *Advanced Encryption Standard*

API *Application Programming Interface*

APN *Amazon Partner Network*

AWS *Amazon Web Services*

BMW *Bayerische Motoren Werke*

CKM *Cloud Key Management*

CLI *Command-Line Interface*

CSV *Comma-separated Values*

CTW *Critical TechWorks*

DDL *Data Definition Language*

DNS *Domain Name System*

EC2 *Elastic Compute Cloud*

ETL *Extract Transform Load*

GCP *Google Cloud Platform*

HDD *Hard Disk Drive*

IaC *Infrastructure as Code*

IAM *Identity and Access Management*

IoT *Internet of Things*

KMS *Key Management Service*

KPI *Key Performance Indicator*

LZO *Lempel-Ziv-Oberhumer*

OCI *Oracle Cloud Infrastructure*

ORC *Optimized Row Columnar*

PHP *Hypertext Preprocessor*

RAM *Random Access Memory*

RDS *Relational Database Service*

SDK *Software Development Kit*

SNS *Simple Notification Service*

SSD *Solid State Drive*

SSL *Secure Sockets Layer*

SQL *Structured Query Language*

TCP *Transmission Control Protocol*

TLS *Transport Layer Security*

TSV *Tab-separated values*

UBI *Universidade da Beira Interior*

URI *Uniform Resource Identifier*

VPC *Virtual Private Cloud*

XML *Extensible Markup Language*

ACID *Atomicidade, Consistência, Isolamento e Durabilidade*

AEAD *Authenticated Encryption with Associated Data*

DDoS *Distributed Denial of Service*

HTTP *Hypertext Transfer Protocol*

JDBC *Java Database Connectivity*

JSON *JavaScript Object Notation*

ODBC *Open Database Connectivity*

OLAP *Online Analytical Processing*

OLTP *Online Transactional Processing*

REST *Representational State Transfer*

RTSP *Real Time Streaming Protocol*

SSMS Structured Query Language *(SQL) Server Management Studio*

SaaS *Software as a Service*

T-SQL *Transact - Structured Query Language*

vCPU *Virtual Central Processing Unit*

Capítulo 1

Introdução

Este documento específica o trabalho realizado na unidade curricular de projeto de estágio em Engenharia Informática inserido no segundo ano do 2º ciclo em Engenharia Informática na *Universidade da Beira Interior* (UBI). Este projeto consiste na análise e documentação de vários serviços de *Data Warehouse* para aferir qual é o mais adequado a um sistema sustentável de *Big Data* para suportar as decisões de negócio num sistema de informação, em desenvolvimento na CTW. Este sistema enquadra-se na área financeira da BMW no Reino Unido, uma região de elevada importância ao nível de volume de vendas nos vários serviços prestados.

Existem inúmeras fontes de informações úteis sobre os vários serviços e negócios do grupo BMW, como por exemplo, venda de automóveis do grupo BMW e o sistema bancário BMW. O processamento e armazenamento destas informações/dados é imperativo para auxiliar nas tomadas de decisões, permitindo à BMW oferecer um melhor serviço nas suas mais variadas áreas de negócio.

As seguintes subsecções apresentam melhor o que é pretendido, com uma caracterização da empresa onde se insere este projeto, a descrição do problema a resolver e a organização do documento onde irá apresentar o que é tratado em cada capítulo.

1.1 Caracterização da empresa

Este projeto de estágio em Engenharia Informática é realizado em colaboração com a CTW, empresa focada no desenvolvimento de soluções de software para o ramo automóvel, detida pela BMW e pela Critical Software. A CTW surge da fusão da responsabilidade e tradição no sector automóvel do grupo BMW com a excelência e experiência da Critical Software no desenvolvimento de soluções recorrendo às melhores práticas de engenharia.

O início da história do grupo BMW remonta a 1913 e conta com uma vasta experiência no desenho e fabrico de automóveis e motociclos. Atualmente a marca BMW é uma das mais reconhecidas e premiadas, contando com modelos icónicos de carros como BMW E30 M3, BMW 2002 Turbo e mais recentemente BMW i8. Esta história teve alguns marcos ao longo dos anos, dos mais importantes destacam-se:

- **1913**: nasce a fabricante de motores de avião Rapp Motorenwerke fundada por Karl Rapp;
- **1917**: saída de Karl Rapp e a empresa muda o nome para Bayerische Motoren Werke, grande parte da produção desviada para apoiar o exército alemão na Primeira Guerra Mundial;

- **1919**: a criação do Tratado de Versalhes impede a BMW de produzir motores de avião e foca a sua produção em máquinas agrícolas e travões para comboios;
- **1921**: início da fabricação de motores de motociclos para outras empresas, mais tarde esta fabricação passou para o desenvolvimento de motociclos BMW com a introdução da BMW R 32;
- **1928**: compra da empresa Automobilwerk Eisenach que permite à BMW iniciar a produção de automóveis, produzindo o primeiro automóvel da marca BMW designado BMW 3/15;
- **1939**: abertura de uma fábrica em Munique para a fabricação de motores de aviões, esta produção serve para apoiar o exército alemão na Segunda Guerra Mundial;
- **1945**: fim da Segunda Guerra Mundial com todas as fábricas destruídas pela união soviética, inicia a proibição da BMW produzir carros, motociclos e aviões.
- **1952**: BMW volta a iniciar a produção de automóveis e ciclomotores, e inicia um percurso de sucesso;
- **1994**: aquisição do grupo Rover, porém só mantém a marca Mini;
- **1998**: aquisição prestigiada marca Rolls-Royce;
- **2013**: alteração do foco na produção de automóveis com carburação de combustíveis fosseis para automóveis elétricos.

Com a necessidade por parte da BMW de melhorar a sua produção de *software* e melhorar os seus processos através da tecnologia, nasceu a Critical TechWorks. A CTW é uma *joint venture* entre a BMW e a Critical Software iniciada em 2018, para liderar o futuro das máquinas em movimento [Cri21]. A CTW foi exclusivamente criada para apoiar a BMW na construção de software para apoiar as suas mais variadas áreas de negócio como a construção de soluções modernas no suporte a futuras máquinas e na modernização de todos os processos BMW.

A produção de *software* na CTW, foca-se tanto na componente *onboard* como *offboard*. Para a componente *onboard* trata todos os sistemas que existem no interior dos veículos, tais como: painéis de navegação tal como o apresentado na figura 1.1; e aplicações de entretenimento. Por outro lado, a componente de *offboard* trata todos os sistemas fora dos veículos, como sistemas para áreas financeiras e para fábricas de produção de veículos.

Através desta colaboração, a CTW pretende contribuir para a formação de quadros especializados, dinamizando colaborações Indústria/Universidade essenciais para o crescimento de ambas as partes.

Figura 1.1: Apresentação de o novo sistema iDrive presente no painel panorâmico dos novos automóveis iX, desenhada e desenvolvido na CTW [BMW21a].

1.2 Descrição do Problema

Em grandes sistemas de informação pode existir um aumento exponencial de informação gerada, através de inúmeras fontes com um enorme fluxo de dados em formatos diferentes. Este aumento torna-se um problema e desafio para que as empresas conseguirem ingerir, armazenar e analisar estes dados em tempo razoável e com eficiência [Cig19]. A análise destes dados permite a identificação de tendências ou irregularidades, sendo fulcral e crítica para muitas empresas, permitindo inovar, realizar o planeamento de *marketing* e orçamento, ajudando na tomada de decisões.

Para solucionar este problema, surgiu a área de *Big Data* que visa tratar, analisar e obter informações a partir de conjuntos de dados de grande dimensão que os sistemas tradicionais não conseguem processar. Esta área deu origem a algoritmos e ferramentas que conseguem processar diferentes quantidades de dados (desde os *Megabyte* (MB) até aos *Petabyte* (PB)) com diversas velocidades (desde a receção de dados em tempo real até à receção periódica de dados em *batch*) com os mais variados tipos e formatos de dados (como texto e imagens) [SAS21]. Contextualizando, um sistema completo de *Big Data* é composto por três camadas:

- Camada de Ingestão ou *Data Ingestion*: permite receber os dados gerados pelas várias fontes, processa esses dados, transformando-os;
- Camada de *Data Warehouse*: possibilita o armazenamento de dados processados na camada de ingestão, num serviço de base de dados na *cloud*, altamente disponível, confiável;

- Camada de *Reporting*: executa *queries* nos dados presentes na camada de *Data Warehoure* para a sua análise, por meio de algoritmos de visualização e *machine learning*.

O funcionamento de um sistema de *Big Data* inicia com a receção e transformação de dados para um formato definido usando ferramentas de ingestão (camada de ingestão). O armazenamento desta informação é efetuada automaticamente em serviços de base de dados na *cloud* designadas de *Data Warehouses*. Finalmente, estes dados são analisados em tempo útil na camada de *Reporting* utilizando ferramentas de *Business Intelligence* e *Data Analytics*. O diagrama 1.2 representa um exemplo da estruturação de um sistema de *Big Data* completo com as várias camadas, onde é visível o fluxo de dados por meio das várias camadas.

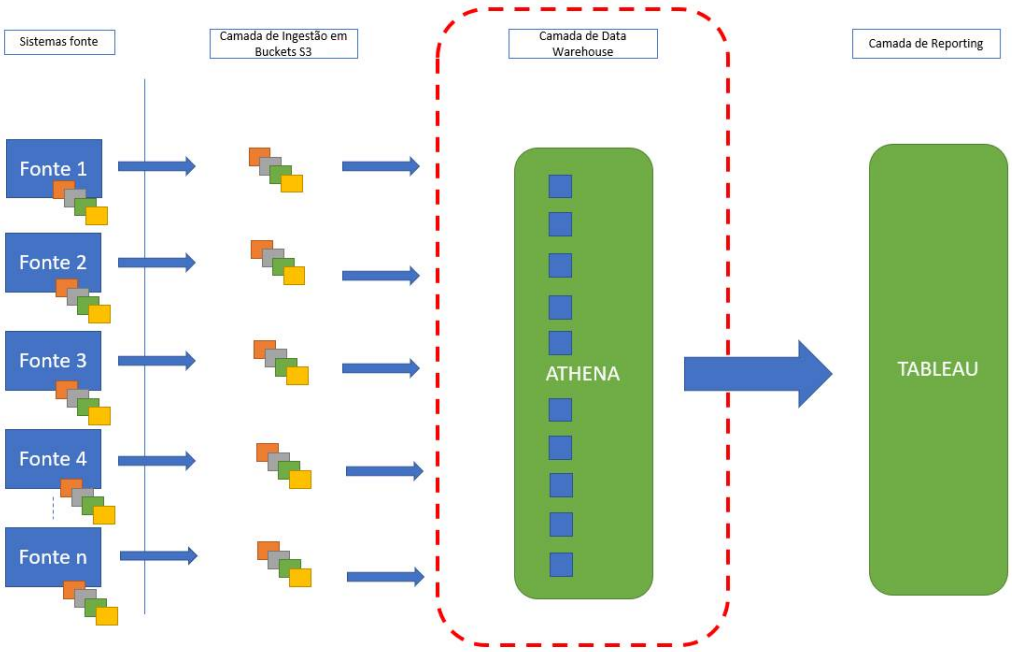

Figura 1.2: Diagrama representativo de um sistema de *Big Data* completo com a camada de *Data Warehouse* assinalada pela caixa vermelha.

Existe um sistema de *Big Data* atualmente em desenvolvimento pela CTW. O foco atual está na integração e desenvolvimento das duas primeiras camadas (*Data Ingestion* e de *Data Warehouse*). Este sistema começa por receber dados de vendas automóveis BMW de várias fontes, transformando esses dados brutos em informação útil. De seguida, armazena-os para que possam ser usados por ferramentas de análise com *Dashboards*, dentro da temática de *Business Inteligence*. Por conseguinte, é facilitada a tomada de decisões nas mais variadas áreas de negócio da BMW, auxiliando esta na continuação da sua liderança de mercado.

O principal objetivo deste projeto é a analise de ferramentas *Data Warehouse* para a integração no sistema de *Big Data*, com o objetivo de aferir qual a que mais se adequa ao sistema. As ferramentas atualmente utilizadas no sistema são AWS Glue na camada de Ingestão e Tableu (Python3) na camada de *Reporting*, por isso a ferramenta de *Data*

Warehouse selecionada terá obrigatoriamente compatibilidade com estas tecnologias.

A camada de *Data Warehouse* é uma componente central de um sistema de *Big Data*, tendo o seu desempenho uma elevada importância para o bom funcionamento e eficiência de todo o sistema. Por isso, tem de apresentar algumas funcionalidades chave como segurança, escalabilidade, mecanismos de recuperação e desempenho para permitir um rápido acesso aos dados.

A realização deste projeto faz proveito do mecanismo de trabalho da CTW que utiliza uma filosofia Agile numa *framework* Scrum.

1.3 Organização do Documento

Este documento encontra-se estruturado da seguinte forma:

1. O primeiro capítulo – **Introdução** – apresenta o projeto, contextualiza o mesmo, apresentando uma caracterização da empresa onde este projeto se insere. Descreve o problema a solucionar e os seus objetivos. Apresentando, por fim a respetiva organização do documento.

2. O segundo capítulo – **Estado da arte** – descreve as potenciais ferramentas de *Data Warehouse* que podem ser integradas no sistema de *Big Data* da CTW.

3. O terceiro capítulo – **Método Proposto** – expõe o método proposto através da sua abordagem e da respetiva planificação.

4. O quarto capítulo – **Tecnologias e Ferramentas Usadas** – descreve as tecnologias e ferramentas principais usadas na construção de um sistema *Big Data*.

5. O quinto capítulo – **Sistema *Big Data*** – apresenta o funcionamento e o propósito do sistema *Big Data* CTW onde este projeto se insere.

6. O sexto capítulo – **Comparação e análise dos serviços de *Data Warehouse* escolhidos** – compara detalhadamente os serviços de *Data Warehouse* selecionados em contexto de sistema *Big Data* apresentado no quinto capítulo.

7. O sétimo capítulo – **Conclusão** – faz uma revisão de todo o documento e reiterando as principais conclusões retiradas da análise feita sobre problema a solucionar, bem como a apresentação de futuras implementações.

Capítulo 2

Estado da Arte

Este capítulo apresenta e analisa diversas tecnologias *Data Warehouse* atualmente disponíveis no mercado nomeadamente: AWS Athena, AWS Aurora, AWS RedShift, Google BigQuery e Microsoft Azure Synapse Analytics. Esta análise tem o intuito de perceber e selecionar quais as que mais se adequam a um sistema de *Big Data* em desenvolvimento na CTW.

As tecnologias de *Data Warehouse* têm de ser capazes de armazenar e mover uma grande quantidade de dados para que estes sejam analisados para o apoio decisões de negócio. Devido ao fluxo constante de dados, estas tecnologias têm de ser capazes de gerir este fluxo sem falhas, com um bom desempenho, segurança e não exibindo custos exorbitantes.

De seguida, é apresentada uma breve introdução sobre o conceito de *Data Warehouse*, seguido de várias apresentações e análises a diferentes tecnologias de *Data Warehouse*. Cada tecnologia inicia com uma introdução, descrevendo e realizando uma análise das suas características em cada subsecção. No final é apresentada uma breve conclusão.

2.1 Data Warehouse

A quantidade de dados no mundo está a crescer a um ritmo exponencial, não havendo sinais deste crescimento abrandar, já que, cada vez mais há um acréscimo de população com acesso à Internet. Este crescimento exponencial é visível na imagem 2.1 onde é apresentado a quantidade de dados a cada ano.

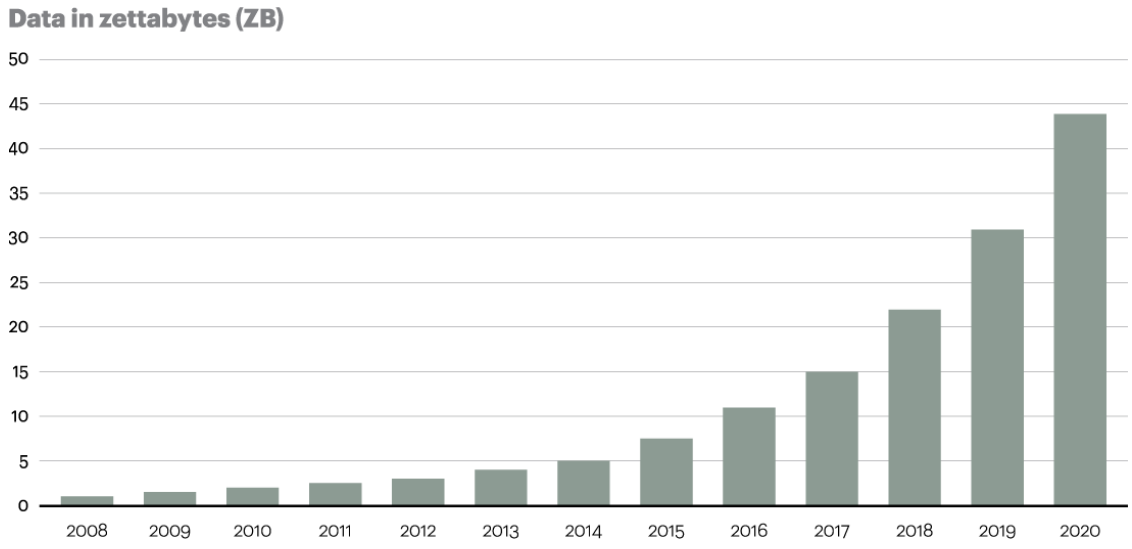

Figura 2.1: Gráfico com uma estimativa de volume de dados existentes ao longo dos anos em ZB [JRRR16].

Camada de Data Warehouse num sistema de Big Data da Critical TechWorks

Existe uma enorme diversidade de atividades com uma grande variedade de dados envolvidos para completar estas atividades. Para o cliente, é espectável que estas transações de dados sejam feitas de formas praticamente instantânea, já para um negócio, qualquer atividade como *clicks*, *likes* ou *swipes* é informação útil sobre os clientes e o que estes querem. Por exemplo, em termos estatísticos no ano de 2020 a cada minuto houve em média 6659 encomendas realizadas no *website* da Amazon, 138889 *clicks* em publicidades no Instagram e 208333 utilizadores em chamada pela aplicação Zoom [DOM21]. É importante que a análise destes dados gerados seja feitos de forma rápida, porque se houver um processo moroso, os dados presentes na análise podem já estar desactualizados. Desta forma, a escolha do serviço de Data Warehouse a utilizar em cada problema e cenário é essencial.

Escolher o melhor serviço ou tecnologia para um certo cenário ou problema envolve perceber as soluções disponíveis no mercado, percebendo quais são os seus pontos fortes e fracos. A escolha da tecnologia mais adequada melhora drasticamente o desempenho da solução e reduz o tempo de manutenção da solução, assim como a sua segurança e robustez. Para realizar esta escolha é necessário ter uma percepção de cada tecnologia, como por exemplo, que ferramentas são compatíveis com o serviço ou a possível existência de limites na quantidade de dados.

A decisão sobre qual o serviço de Data Warehouse mais apropriado, normalmente envolve um estudo sobre certos aspectos chave. Estes aspectos e a sua importância, são nomeadamente:

- Integração: para um serviço ser inserido num sistema é importante perceber como este se pode integrar com os outros componentes do sistema através de conexões ou *Application Programming Interface* (API);
- Desempenho: para algumas indústrias, o atraso em milissegundos de uma *query* pode afetar o funcionamento de uma aplicação. Podem haver adicionalmente requisitos em termos de quanto tempo demora o serviço a recuperar através de um *backup*;
- Segurança: para uma grande maioria dos sistemas ou soluções, a segurança dos dados é uma prioridade absoluta. Por isso, é fulcral escolher o serviço que tenha um nível de segurança do qual a solução necessita. Por exemplo, alguns serviços realizam tarefas de encriptação por definição, outros necessitam de configuração para permitir essas tarefas ou de uma integração com um serviço externo. Adicionalmente, cada serviço oferece diferentes opções nas políticas de controlo de acesso;
- Custos: é sempre uma das prioridades de cada projeto visto que nunca há um orçamento infinito. Perceber os custos para os diferentes componentes de cada serviço e como estes mudam dependendo da sua utilização e configuração, é essencial para o desenho geral da solução.

De seguida, são analisados vários serviços de Data Warehouse com base nas características e aspectos chave.

2.2 AWS Athena

AWS Athena é um serviço de consultas sem servidor que permite realizar consultas iterativas para facilitar a análise de dados presentes em AWS S3, não sendo necessário a presença de nenhuma infraestrutura de computação para usufruir deste serviço, e por isso não requer nenhuma configuração e manutenção [Ama16a].

O motor que o AWS Athena utiliza para realizar as consultas em SQL é o software *open-source* Presto. É um motor de consultas SQL distribuído, otimizado para baixa latência e desenhado para consultas analíticas rápidas em dados de qualquer tamanho. Devido a este motor, é garantindo o desempenho mesmo em solicitações SQL mais complexas como funções de janela (como LAG e LEAD), grandes junções e agregações.

O Athena consegue processar tipos de dados estruturados e não estruturados, sendo compatível com diferentes formatos de ficheiros como *Comma-separated Values* (CSV), *JavaScript Object Notation* (JSON), *Optimized Row Columnar* (ORC), Parquet e Avro. Apresenta um vasto suporte a formatos de ficheiros compactados como Snappy, Zlib, *Lempel-Ziv-Oberhumer* (LZO) e gzip.

2.2.1 Integração

Existem várias formas de aceder ao serviço do Athena, nomeadamente por meio de AWS Management Console, de uma API ou de uma conexão *Java Database Connectivity* (JDBC) driver. De seguida, apenas é necessário definir o esquema e começar a executar consultas SQL nos dados armazenados em S3.

Athena oferece uma facilidade extra na sua integração com outros serviços do portfólio da AWS como o AWS QuickSight para a visualização de dados ou com o AWS Glue para uma catalogação de dados mais eficaz. A sua principal ligação de integração é com o AWS S3 que funciona diretamente como um armazenamento de dados subjacente fornecendo redundância de dados. Athena usa catálogos de dados para armazenar informações e esquemas relacionados às pesquisas realizadas nos dados armazenados em AWS S3.

2.2.2 Desempenho

Em relação ao desempenho, Athena é especialmente otimizado para uma performance rápida com o serviço S3, que é atualmente utilizado pela equipa CTW. Não há necessidade de carregar ou agregar dados S3 no AWS Athena, trabalhando diretamente com os dados armazenados no S3, o que torna mais fácil e rápido a análise destes dados.

2.2.3 Segurança

Um administrador pode gerir e monitorizar os acessos ao Athena através das políticas de identidade e de gestão de acessos do AWS. Isto é realizado por intermédio da apresentação de listas de controlo de acessos e das políticas dos *buckets* AWS S3. Permite também o acesso entre contas aos *buckets* S3 de outros utilizadores.

O serviço Athena beneficia do AWS *Key Management Service* (KMS), permitindo aos utilizadores executarem *queries* cifradas e receber os resultados das *queries* através de chaves geradas pelo KMS.

2.2.4 Custos

O custo dos serviços do AWS Athena é feito mediante a quantidade de dados verificados por cada consulta através do número de *bytes*. O tamanho destas consultas são arredondados por excesso em *bytes* para o *Megabyte* mais próximo. Apenas as consultas com um mínimo de 10 MB são cobradas. As *queries* de *Data Definition Language* (DDL) como CREATE, ALTER e DROP TABLE não possuem qualquer custo adicional [Ama16b].

Existem vários métodos para reduzir os custos, sendo estes: compactação dos dados para que o serviço contabilize uma menor quantidade de dados, organização de dados em formatos colunares que permitem realizar consultas seletivas apenas às colunas necessárias para processar os dados e particionar estes para restringir a quantidade de dados analisados.

O preço por cada *Terabyte* (TB) de dados consultados é de 4,11 € (euros), porém é expectável que através dos métodos de poupança mencionados anteriormente haja uma poupança de entre 30% a 90% no valor.

2.3 AWS Aurora

O AWS Aurora é um mecanismo de base de dados relacional na *cloud*. É desenhado para oferecer o melhor de ambas as soluções comerciais e dos serviços *open-source*. Combinando assim a velocidade e a confiabilidade dos serviços de base de dados comerciais com o baixo custo e versatilidade dos serviços *open-source*. Pelo facto do AWS Aurora ser um serviço *cloud* sem servidor, é possível simplificar os acessos e configurações do serviço através da interface de Amazon *Relational Database Service* (RDS), permitindo uma maior velocidade no desenvolvimento de *software* [Ama14a].

As tarefas de gestão de um serviço de base de dados do AWS Aurora são feitas através do AWS RDS que fica responsável pela automação de tarefas administrativas. Estas tarefas são: o provisionamento de hardware, aplicação de atualizações de *software*, execução de *backups*, deteção de falhas e recuperações.

A escalabilidade é feita automaticamente, sendo que há a possibilidade de aumentar ou reduzir facilmente a escala de cada implantação e usar instâncias menores ou maiores conforme a evolução das necessidades. O Aurora aumenta automaticamente o armazenamento de cada uma destas instâncias até 128 TB que atuam como destino de *failover*, em que a sua recuperação leva menos de 30 segundos e sem perda de dados, havendo assim um máximo de 15 réplicas.

2.3.1 Integração

É um serviço compatível com MySQL e PostgreSQL, oferecendo uma grande facilidade na migração de dados destes serviços. O Aurora pode utilizar uma destas duas ferramentas como motores, disponibilizando uma versão *cloud* melhorada destas ferramentas a que designamos de Aurora MySQL e Aurora PostgreSQL. Estas duas versões melhoradas pelo Aurora não necessitam de qualquer instalação das suas versões básicas para funcionar diretamente nas aplicações, sendo tudo executado através da *cloud*.

Para embutir o serviço Aurora num sistema, é automaticamente associado ao serviço um *Internet Protocol* (IP) ou *Transmission Control Protocol* (TCP), havendo necessidade de definir o número da porta para que as aplicações desenvolvidas possam comunicar com a base de dados do serviço Aurora.

2.3.2 Desempenho

O Amazon Aurora MySQL fornece uma *performance* até cinco vezes maior que o MySQL em *throughput*, enquanto o Amazon Aurora PostgreSQL fornece um desempenho até três vezes maior que o PostgreSQL. Estes dados de *performance* são relativos à velocidade de execução de *queries* SQL de ”SELECT”e ”UPDATE”. Estas melhorias são possíveis através de uma série de otimizações realizadas pela Amazon tais como a presença de uma camada de armazenamento virtualizada. O código desenvolvido para qualquer aplicação que utilize MySQL ou PostgreSQL não necessita de nenhuma alteração para ser intregado com o seu correspondente no serviço da Amazon.

2.3.3 Segurança

Há vários níveis de segurança que o AWS Aurora oferece em relação à base de dados. À semelhança do AWS Athena, o Aurora utiliza os serviços do AWS KMS para a geração de chaves de encriptação para os dados da base de dados, incluindo os que estão presentes em *backups* e em réplicas. Aurora faz uso de AWS *Virtual Private Cloud* (VPC) para o isolamento de redes e *Secure Sockets Layer* (SSL) para proteger a conexão entre a base de dados e as aplicações. Por fim, utiliza *Advanced Encryption Standard* (AES) de 256 *bits* para cifrar os dados que não estão em uso.

2.3.4 Custos

Em relação aos custos, o Aurora tem uma taxa de armazenamento de dados no valor de 0,098 €(euros)/*Gigabyte* (GB) por mês. Nos custos das consultas é faturada uma taxa de 0,18 €(euros) por cada 1 milhão de solicitações em dados de entrada/saída. Em relação ao armazenamento de *backup* e *snapshots* o preço é de cerca de 0,019 €(euros)/GB por mês [Ama14b].

Existe uma opção de voltar atrás no tempo que permite restaurar uma base de dados para um momento anterior sem a necessidade de recorrer a um *backup*, porém esta funcionalidade só está disponível atualmente para Aurora MySQL. Este recurso serve para

recuperar rapidamente erros dos utilizadores como por exemplo a eliminação de tabelas ou linhas erradas, sendo necessário especificar há quanto tempo se encontra a versão da base de dados pretendida. O Aurora guarda os ficheiros *log* e é cobrada uma taxa por hora para armazenar estes registos de cerca de 0,012 € (euros) por cada 1 milhão de registos. Por fim, é possível a execução de *snapshots* que são cobrados pelo seu tamanho em GB tendo um preço de 0,0091 € (euros) por GB.

2.4 AWS RedShift

AWS RedShift é um produto de computação na *cloud* de *Data Warehouse* oferecendo uma computação paralela para conseguir processar ou migrar grandes quantidades de dados estruturados ou semiestruturados organizados em formato colunar, baseado em PostgreSQL. Esta computação combinada com a compressão de dados permite ao RedShift uma velocidade elevada, desenhada para processar conjuntos de dados numa escala PB de tamanho, permitindo executar milhões de operações em linhas de uma tabela na base de dados ao mesmo tempo [Ama12a].

Existem vários formatos de ficheiros compatíveis com o RedShift como o Parquet, ORC, JSON, Avro, CSV, entre outros. Ficheiros com estes formatos que estejam inseridos em AWS S3 podem ser diretamente consultados através de SQL. Para exportar estes dados novamente para o AWS S3, o Redshift formata e move novamente para o AWS S3 de forma automática. Estes formatos de ficheiros também permitem uma maior facilidade na consulta e gravação de dados em *data lakes*, havendo um comando UNLOAD em SQL que exporta diretamente dados para um *data lake*, após a escolha de um formato de ficheiro compatível. A funcionalidade que permite a ligação direta entre RedShift e S3 é chamada de RedShift Spectrum.

O *backup* dos dados é executado automaticamente, transferindo os dados para o AWS S3. Os *snapshots* podem ser replicados em várias regiões de disponibilidade do serviço Amazon para a prevenção de recuperação após desastres. O RedShift monitoriza continuamente a integridade dos dados, replica-os de forma automática as unidades em falha e substitui os componentes defeituosos oferecendo assim tolerância a falhas.

2.4.1 Integração

O RedShift é simples de configurar e de manter. A configuração é rápida e facilitada, por meio da criação automática da infraestrutura e automatização da maioria das tarefas administrativas como *backups* e replicação. Mas também permite personalizar o serviço por um administrador, caso este queira assumir o controlo.

RedShift apresenta uma integração nativa natural com os outros serviços do ecossistema da AWS, como o AWS Glue, Amazon QuickSight e o Amazon SageMaker. O primeiro, extrai e transforma dados, e de seguida carrega estes diretamente no RedShift. O segundo, serve para a visualização dos dados, e por fim, o Amazon SageMaker para executar cargas de trabalho de *machine learning*.

Existem muitos parceiros da Amazon com ferramentas compatíveis com o RedShift com o objetivo de facilitar a integração deste serviço, apenas sendo necessário conexões *Open Database Connectivity* (ODBC) ou JDBC padrão.

2.4.2 Desempenho

Os altos níveis de *throughput* e desempenho, mesmo com cargas de trabalho variáveis e atividades simultâneas de utilizadores, são garantidas através da utilização de algoritmos sofisticados de *machine learning*. Os algoritmos permitem prever e classificar as consultas recebidas com base nos seus tempos de execução e requisitos de recursos para gerir dinamicamente o desempenho e a simultaneidade, para além de ajudar a priorizar as cargas de trabalho empresariais críticas. Através destas previsões é possível gerir dinamicamente a memória e a simultaneidade garantindo assim a maximização do *throughput* das consultas. O escalonamento da memória tem um limite de 8 PB. Adicionalmente, são armazenados os resultados de consultas em *cache* para que consultas repetidas tenham tempos de resposta inferiores a um segundo.

2.4.3 Segurança

No que se refere à segurança da RedShift é utilizado SSL na proteção de dados em trânsito e criptografia AES de 256 *bits* para os dados inoperantes, incluindo os *backups*. O serviço RedShift gere automaticamente as chaves sem recorrer a outro serviço, no entanto, existe a possibilidade de gerar chaves noutro serviço externo.

É disponibilizado o serviço AWS CloudTrail para possibilitar a auditoria de todas as chamadas à API do Redshift. O Redshift regista em ficheiros *log* todas as operações de SQL, inclusive tentativas de conexão, consultas e alterações. Oferece ainda a possibilidade de restringir os dados aos quais os utilizadores têm acesso, podendo bloquear linhas e colunas.

Finalmente, é possível executar o Redshift dentro da AWS VPC para obter um isolamento da rede.

2.4.4 Custos

Os custos do AWS RedShift são cobrados através da definição do serviço *on demand* (pagamento à medida que os recursos são utilizados) ou através de uma instância reservada. Nos pagamentos de serviço *on demand*, paga-se uma taxa por hora dependendo do poder computacional utilizado que varia entre o número de *Virtual Central Processing Unit* (vCPU) disponíveis (entre 2 e 48), da memória a utilizar (entre 15GB e 384GB), do tipo e capacidade do armazenamento (entre 0,16 TB e 64 TB) e da velocidade de E/S (entre 0,60GB por segundo e 8,00GB por segundo), variando no preço entre 0,27€ (euros) e 12,9€ (euros) por hora. No caso de uma instância reservada os recursos são iguais apenas tendo uma duração fixa do serviço e o tipo de pagamento a efetuar [Ama12b].

Existe uma taxa de 0,021 €(euros) por mês a cada GB de dados armazenados no RedShift. Já o RedShift Spectrum tem uma taxa de 4,14€ (euros) por TB de dados exa-

minados para *queries* com, no mínimo, 10 MB sem haver custos para instruções de DDL sendo possível economizar este valor.

2.5 Google BigQuery

Google BigQuery é um serviço *cloud* sem servidor de *Data Warehouse* da Google Cloud Plataform, destinado ao processamento de grandes conjuntos de dados na escala de PB usando SQL. É desenhado especificamente para a temática de *Business Inteligence* sendo otimizado para a leitura de dados. Pode processar diretamente dados de fontes externas, como dados provenientes de outros serviços *cloud*, formatos de ficheiros como Parquet, ORC e ficheiros presentes no Google Drive [Goo11a].

A Google oferece uma série de ferramentas inovadoras que melhoram o serviço do BigQuery, como por exemplo, Data Q&A que permite o processamento de linguagem natural, transformando linguagem humana em código SQL. Esta facilita o acesso dos dados a qualquer pessoa, mesmo não tendo conhecimentos de SQL. Pode também pode ser integrada em aplicações *software* para vários fins, como *chatbots*, interfaces personalizadas e no auxílio da análise de dados.

O BigQuery fornece recursos automaticamente escaláveis sem restrições e sem necessidade de gerir uma infraestrutura. Os dados são replicados automaticamente em várias regiões de disponibilidade dos serviços *cloud* da Google. Estas réplicas podem servir de *backups*, havendo um histórico de alterações feitas nos últimos 7 dias facilitando a restauração em caso de falha e comparar dados presentes em tempos distintos. A replicação dos dados para certas localizações geográficas também pode ser requisitada manualmente. É disponibilizado um conjunto de recursos avançados de monitorização, geração de ficheiros *log* e alertas por através de um serviço complementar da Google, o Cloud Logging.

2.5.1 Integração

Os dados podem ser também diretamente carregados através de Google Dataflow que é o serviço de ingestão de dados da Google e outros serviços da Google como Google Ads, YouTube e Google Marketing Platform. Pode adicionalmente receber dados de serviços externos como AWS S3 ou Teradata.

É fornecida uma API *Representational State Transfer* (REST) otimizada para processar fluxos de dados constantes de *websites* com o objetivo de disponibilizar rapidamente os dados para análise [Ber15]. Existe uma API Storage que permite a escrita ou leitura de dados sem a sua replicação para serviços como TensorFlow, Apache Spark e Apache Beam. Estão disponíveis bibliotecas de cliente em Java, Python, Node.js, C#, Go, Ruby e *Hypertext Preprocessor* (PHP).

Por fim, BigQuery possui a disponibilidade de *drivers* ODBC e JDBC para a integração em aplicações já desenvolvidas.

2.5.2 Desempenho

Não é disponibilizada uma métrica que justifique a *performance* de BigQuery. A Google apenas apresenta uma série de recomendações para maximizar a velocidade do serviço. Esta lista inclui diversas recomendações nomeadamente: não usar código SQL para a transformação de dados, evitar usar funções de JavaScript criadas pelo utilizador (funções não nativas), usar as funções de agregação mais aproximadas possível (como ”APPROX_COUNT_DISTINCT()”em vez de ”COUNT(DISTINCT))”e usar tabelas com partições.

BigQuery oferece a integração de um serviço de armazenamento de consultas em *cache* chamado BigQuery Materialized Views para melhorar o desempenho e reduzir custos.

2.5.3 Segurança

A nível de segurança, BigQuery oferece vários recursos como o controlo de acessos ao sistema com uma gestão de permissões e funções. Estas podem ser personalizadas granularmente especificando exatamente que serviços cada utilizador tem acesso e quais as suas permissões nestes serviços.

BigQuery codifica automaticamente todos os dados antes destes serem armazenados e são automaticamente descodificados caso um utilizador autorizado pretenda ler estes dados. A geração de chaves criptográficas é feita automaticamente através do serviço Google *Cloud Key Management* (CKM), utilizando AES para a encriptação de dados estáticos e *Transport Layer Security* (TLS) para dados em transações. Caso um administrador não queira usar o serviço Google CKM, há a possibilidade de rejeitar este serviço inserindo e fazendo a gestão de chaves criptográficas manualmente.

Finalmente, existe a possibilidade de encriptar valores individuais numa tabela com o objetivo de apenas alguns utilizadores obterem acesso a estes dados através de funções *Authenticated Encryption with Associated Data* (AEAD) que são baseadas em AES.

2.5.4 Custos

Em relação aos custos, BigQuery oferece gratuitamente operações de DDL. O armazenamento de dados apresenta uma taxa mensal de 0,016€ (euros) por GB. Na consulta e processamento de dados é faturada uma taxa de 4,11 € (euros) por cada TB [Goo11b]. Estes valores podem ser reduzidos por meio de uma lista de ”melhores práticas”disponibilizada pela Google onde é demonstrado práticas em certos cenários para reduzir o custo, servindo de guia. Tais práticas incluem: como restringir o número de *bytes* nas consultas, evitar instruções de SQL como ”SELECT *”e fazer partição de dados por datas.

2.6 Microsoft Azure Synapse Analytics

Microsoft Azure Synapse Analytics é um serviço versátil e unificado que consegue constituir um sistema inteiro de *Big Data*, contendo individualmente as três camadas de Ingestão, *Data Warehouse* e *Business Inteligence*. Cada uma destas camadas pode ser utilizada

individualmente, e esta análise foca-se apenas na componente de *Data Warehouse*. Esta oferece um serviço de base de dados relacional em *cloud* pertencente à família de serviços na *cloud* Azure, oferecendo soluções de computação sem servidor assim como *hardware* dedicado para as mais variadas necessidades de cada caso empresarial. Ambas as soluções funcionam com dados estruturados e com dados não estruturados, podendo ser utilizadas com motores de MySQL e PostgreSQL [Mic11a].

Em relação à compatibilidade de formatos de ficheiros, o Azure Synapse Analytics é compatível com uma variedade de formatos tais como Parquet, CSV, *Tab-separated values* (TSV) e JSON.

Para a execução de *queries* é utilizada uma linguagem proprietária da Microsoft, a *Transact - Structured Query Language* (T-SQL) que é uma extensão da linguagem SQL com algumas alterações para melhorar o desempenho e o custo do serviço de *Data Warehouse*. Desta forma, oferece uma integração com modelos de *machine learning* utilizando a função ”PREDICT”. Esta linguagem permite um adicional controlo de erros através da implementação de uma instrução de ”TRY CATCH”.

Devido ao facto de Azure Synapse Analytics ser um serviço sem servidor, não é necessário configurar nem fazer a manutenção de uma infraestrutura já que a disponibilização e gestão da infraestrutura é feita de forma automática e com total abstração. A gestão automática desta infraestrutura contém serviços como a gestão da disponibilidade dos dados, gestão de *backups* e escalabilidade numa escala de PB.

2.6.1 Integração

A integração deste serviço de *Data Warehouse* com os outros serviços complementares do Microsoft Azure Synapse Analytics é feita de forma natural, sendo disponibilizada uma ferramenta de ingestão de dados usando o motor Apache Spark e uma ferramenta de análise de dados.

O desenvolvimento de *software* com a integração deste serviço é facilitada com outros *software* da Microsoft com o Microsoft Visual Studio Code, que é um ambiente de desenvolvimento oferecendo por definição uma opção que facilita a integração do software desenvolvido com o serviço.

Para a utilização deste serviço estão disponíveis conexões ODBC e JDBC e compatibilidade com as linguagens Python, Java, C#, PHP, Scala e .Net que suportam vários sistemas operativos como macOS, Ubuntu e Windows. Para cada caso a Microsoft oferece documentação com exemplos de código para facilitar o desenvolvimento de *software*. Ainda é disponibilizada adicionalmente uma API REST para o desenvolvimento de *websites*.

2.6.2 Desempenho

Para controlar o desempenho, o Azure Synapse Analytics permite ao administrador escolher o número de vCPU máximo e mínimo para oferecer um desempenho personalizado. Os limites de memória e velocidade de *input-output* são sempre geridos e escalados automaticamente através de algoritmos de inteligência artificial.

Adicionalmente são utilizadas técnicas de computação de dados paralela para a execução mais rápida de *queries* e técnicas de armazenamento de dados das últimas em cache para melhorar o desempenho de *queries* repetidas.

A utilização de T-SQL favorece o desempenho porque oferece melhorias do mesmo em certas características do SQL. Consistindo estas, na melhoria no desempenho de instruções de ”DELETE”e ”UPDATE”, funções de suporte a certos tipos de dados como *strings* e *bits*, e declaração de variáveis locais com instruções de ”DECLARE”e ”SET”.

2.6.3 Segurança

Para manter um sistema seguro, as autenticações são feitas com múltiplos fatores, oferecendo uma interface chamada Azure Security Center. Aqui é monitorizada, reportada e gerida toda a atividade de cada conta autenticada nos últimos 30 dias e permite dar alertas de atividades consideradas anómalas ou suspeitas através de algoritmos de *machine learning*. Esta interface tem a capacidade de gerir permissões e privilégios para cada contas/identidades, e podem ser registados vários dispositivos para se ter um acesso exclusivo ao serviço.

Todos os dados estáticos são encriptados de forma transparente usando AES e são utilizados algoritmos de *machine learning* que detetam e reportam potenciais riscos. Estes dados podem ser cifrados a nível colunar utilizando diferentes chaves de encriptação para cada coluna de uma tabela, e a nível celular onde pode haver encriptação adicional de certas células numa tabela. Os dados em trânsito e as comunicações são cifradas utilizando SSL ou TLS. Todas as chaves são geradas e memorizadas pelo serviço Azure por predefinição podendo ser feita de forma manual.

Adicionalmente, é oferecida um proteção base extra de perímetro através de *firewalls* e proteção para ataques de *brute force* e *Distributed Denial of Service* (DDoS). Esta proteção pode ser melhorada com um custo adicional para a proteção de ataques de protocolo e ataques ao nível da aplicação.

2.6.4 Custos

A nível de custos é cobrada uma taxa de 5,271€ (euros) por cada TB de dados processados através de T-SQL sendo que as consultas abaixo de 10 MB e as declarações de DDL não são cobradas. As cópias de segurança podem ser armazenadas com o objetivo de poder restaurar a base de dados para um estado anterior não superior a 35 dias. O custo destas cópias depende do tamanho da base de dados sendo taxado o valor de 0,1116€ (euros) por GB a cada mês [Mic11b].

Para economizar, o serviço permite, de uma forma inteligente, determinar e prever os custos, assim como, a possibilidade de demilitar um valor de TB ou de € (euros) num contexto diário, semanal ou mensal para controlar os custos. A Microsoft também disponibiliza uma lista com exemplos e recomendações para reduzir os custos, como por exemplo, o uso de ficheiros CSV e Parquet, e evitar a execução de certas *queries*.

2.7 Conclusão

Este capítulo aborda os serviços de *Data Warehouse* disponíveis no mercado, visando a análise e a posterior integração de um destes num sistema de *Big Data*. A escolha destes serviços foi baseada no Magic Quadrant, representado na figura 2.2, desenvolvido pela Gartner acerca dos serviços *cloud*. Aqui encontram-se analisados os vários serviços *cloud* disponíveis atualmente no mercado através de uma série de características como a qualidade geral do serviço *cloud* e a visão do serviço através da inovação [Gar20].

O Magic Quadrant, publicado no mês de agosto de 2020, apresenta e classifica vários serviços de *cloud*, identificando os líderes de mercado que, por ordem consistem em: Amazon (AWS), Microsoft (Azure) e Google. Por isso, a escolha dos serviços foi: AWS Athena, AWS Aurora, AWS RedShift, Google BigQuery e Microsoft Azure Synapse Analytics.

Figura 2.2: Gráfico do *Magic Quadrant* representativo da relação entre a visão e qualidade geral do produto *cloud* [Gar20].

Com o levantamento das características chave dos serviços de *Data Warehouse* selecionados, obtém-se a tabela 2.1. Todos estes serviços oferecem soluções vantajosas assegurando níveis de desempenho que são fulcrais para a análise de dados, integração e compatibilidade com as tecnologias atualmente utilizadas pela CTW, segurança para dados críticos e sensíveis, e finalmente de que forma estas soluções se refletem nos custos.

Apesar de todos estes serviços serem interessantes numa perspetiva de negócio, apenas foram selecionados os serviços do ecossistema AWS para a análise final, isto é, Athena, Aurora e RedShift . Isto deve-se ao facto de Microsoft Azure Synapse Analytics apresentar

Tabela 2.1: Tabela de comparação das características chave entre os diferentes serviços de *Data Warehouse*

	Desempenho	Integração	Segurança	Custos de consultas
AWS Athena	Otimizado para S3	AWS Management Console, API e Conexão JDBC	AWS KMS	4,11€(euros) por TB
AWS Aurora	3 vezes mais rápido que PostgreSQL e 5 vezes mais rápido que MySQL	TCP ou IP	AWS KMS, VPC, SSL e AES	0,18€(euros) por 1 milhão de solicitações
AWS RedShift	Algoritmos de controlo para melhorar desempenho	API na *cloud*, Conexões JDBC e ODBC	SSL, VPC e CloudTrail	4,14€(euros) por TB
Google BigQuery	Lista de recomendações para melhorar desempenho	API REST, API Storage, Conexões ODBC e JDBC	Google CKM, AES, TLS e AEAD	4,11€(euros) por TB
Microsoft Azure Synapse Analytics	Algoritmos de inteligência artificial, T-SQL para melhorar o desempenho	API REST e conexões JDBC e ODBC	AES e TLS ou SSL	5,271€(euros) por TB

custos elevados e uma complexidade superior ao que o atual projeto da BMW requer. A exclusão do serviço Google BigQuery deve-se ao facto de não apresentar argumentos que superem os serviços escolhidos, mostrando-se menos vantajoso e apelativo. Por fim, a BMW tem um contrato estratégico com a Amazon para a utilização do ecossistema AWS no qual os serviços Athena, Aurora e RedShift estão inseridos [AWS20].

Capítulo 3

Método Proposto

Este projeto consiste no tratamento de dados através de um sistema sustentável de *Big Data* que visa suportar as decisões de negócio da BMW. Desta forma, o desenvolvimento deste projeto encontra-se dividido em diferentes fases que são realizadas em simultâneo com a empresa.

O presente capítulo apresenta a abordagem proposta onde é relatada a filosofia Agile de desenvolvimento de *software* utilizada pela CTW, designada por *framework* Scrum. Esta *framework* vai ser utilizada para a análise dos serviços de *Data Warehouse*.

Por fim, são exibidas as fases de desenvolvimento do processo de estágio nomeadamente: a fase de levantamento, de experiências, conclusão, revisão e por fim realização do relatório. Estas fases de desenvolvimento são apresentadas esquematicamente através de um diagrama de Gantt.

3.1 Abordagem Proposta

A CTW adota uma filosofia de desenvolvimento de *software* Agile que oferece um guia de como escolher metodologias e procedimentos que se enquadram melhor para cada equipa. Os valores e princípios de Agile não ordenam ou prescrevem a forma como a equipa trabalha, mas procuram ajudar uma equipa a pensar e interagir de forma a conseguir agilidade. Agilidade é o processo de adaptar e melhorar a forma de trabalhar. Devido a esta capacidade de adaptação, Agile não é considerada uma metodologia. Assim, em vez de declarar o que exatamente cada equipa deve fazer e como, Agile oferece um conjunto de valores e princípios que cada equipa pode adotar [Atl20].

Existem várias *frameworks* que ajudam as equipas a seguir esta filosofia, a equipa CTW a integrar utiliza especificamente a *framework* Scrum.

De seguida, é apresentada uma subsecção acerca da *framework* Scrum descrevendo o seu funcionamento e como é vantajosa para o desenvolvimento de *software*. Após a apresentação de Scrum são apresentadas as várias fases constituintes deste projeto de análise a estas plataformas.

3.1.1 SCRUM

Scrum é uma *framework* simples Agile para ajudar na colaboração de elementos de uma equipa na produção, entrega e manutenção de produtos complexos, com especial ênfase no desenvolvimento de *software* [Sar17].

Numa equipa de Scrum existem vários cargos: o *Product Owner*, o Scrum Master e a restante equipa de desenvolvimento. O *Product Owner* é responsável por definir e sequenciar os requisitos do *software* desenvolvido, enquanto que o Scrum Master organiza

a equipa de desenvolvimento que produz o *software*. O processo de desenvolvimento está dividido em fases/eventos:

- **Planeamento do *sprint***: reunião em que a equipa em conjunto define um objetivo, que será uma componente do sistema a desenvolver no próximo *sprint*, validada pelo *Product Owner*;

- ***Sprint***: período de tempo definido, normalmente entre 2 semanas a 4 semanas, em que equipa de desenvolvimento tenta fazer a componente definida no Planeamento do *sprint*, no período de tempo definido. Caso a equipa falhe com a meta proposta pode passar as tarefas que faltam para o próximo *sprint*, definindo novamente um objetivo;

- ***Daily***: reunião de equipa diária com a duração de cerca de 15 minutos onde a equipa se coordena para alcançar o objetivo definido no *sprint*, isto faz com que os elementos da equipa sejam transparentes e se consigam adaptar a novas alterações e informações;

- **Avaliação do *sprint***: reunião após o *sprint* em que a equipa apresenta ao cliente o que conseguiu desenvolver durante o *sprint* e receber algum *feedback* para melhorar e adaptar futuros *sprints*;

- **Retrospetiva do *sprint***: reunião onde a equipa reflete e debate como correu o *sprint* para haver uma adaptação a futuros *sprints* de maneira a que o trabalho seja mais eficaz.

O desenvolvimento do sistema completo também está dividida em 3 componentes: o *backlog* do produto, o *backlog* do *sprint* e o incremento. O *backlog* do produto é uma lista de funcionalidades que o *Product Owner* quer no sistema, ordenada pela importância destas funcionalidades. Por outro lado, o *Backlog* do *sprint* é um conjunto de funcionalidades presentes no backlog do produto, escolhidas pela equipa no planeamento do *sprint* para que essas funcionalidades sejam desenvolvidas durante esse *sprint*. Já o incremento é o resultado de um *sprint* onde as funcionalidades são integradas no sistema alvo/final.

O processo completo do desenvolvimento de *software* em Scrum, visível na figura 3.1, passa pelas fases de levantamento inicial do *backlog* do produto que de seguida é acompanhado de ciclos. Estes começam com o planeamento do *sprint* onde são definidas as funcionalidades que são desenvolvidas no *sprint*, constituindo assim o *backlog* do *sprint*. Seguidamente, inicia-se o *sprint* onde são produzidas as funcionalidades presentes no *backlog* do *sprint*. No decorrer do *sprit*, há reuniões diárias as *daily* entre a equipa de desenvolvimento. No final, ao obter o incremento, há a avaliação do *sprint* em que há uma validação do incremento pelo cliente. Caso o incremento seja aceite, é atualizado o *backlog* do produto. Por fim, ocorre a retrospetiva do *sprint* onde a equipa reflete acerca deste e decide melhorias a aplicar no próximo planeamento.

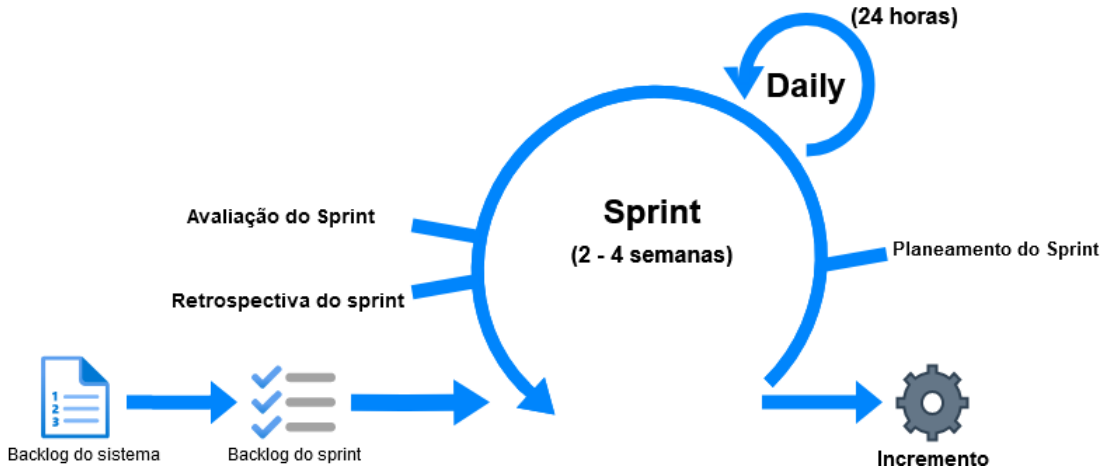

Figura 3.1: Gráfico representativo das etapas de desenvolvimento *software* utilizando a *framework* Scrum.

3.2 Fases de desenvolvimento

O desenvolvimento deste projeto de estágio inserido na CTW está divido em fases, o resultado do desenvolvimento e experiências nestas fases irá ser apresentado e documentado nos seguintes capítulos, estas fases são constituidas por:

1. Fase 1 - **Levantamento** – Participação no *onboarding* e na *data academy* da CTW, seguido da integração na equipa de desenvolvimento da plataforma de *big data*, com o entendimento teórico de toda a solução, e familiarização de componentes importantes do projeto;

2. Fase 2 - **Experiências** – Corresponde ao período prático, iniciado com o levantamento e integração no sistema *Big Data* com o objetivo de perceber o seu funcionamento. A fim de analisar e comparar AWS Athena, AWS Aurora e AWS RedShift no contexto deste sistema.

3. Fase 3 - **Conclusões** – Resulta na comparação dos resultados obtidos na análise dos serviços de *Data Warehouse*, seguido da análise e tratamento dos resultados obtidos originando o levantamento de conclusões;

4. Fase 4 - **Revisão** – Revisão da solução encontrada, com a possível integração do serviço de *Data Warehouse* no sistema de *big data* e últimos retoques;

5. Fase 5 - **Realização do relatório** - Esta etapa corresponde à escrita na íntegra de todos os procedimentos realizados ao longo do estágio. Esta fase tem uma duração contínua ao longo do desenvolvimento do estágio, sendo que nas últimas duas semanas terá um foco acentuado para a conclusão da versão final.

3.3 Planificação do Trabalho

O estágio na CTW realiza-se ao longo do segundo semestre do ano letivo de 2020/2021 que decorre entre 18 de janeiro a 14 de junho de 2021. Ao distribuir as fases descritas na secção anterior (3.2) pelo tempo de estágio, a planificação do mesmo resultou no mapa de Gantt presente na figura 3.2.

O estágio inicia com 2 semanas de *onboarding* onde são apresentadas as várias componentes da CTW através de palestras e *workshops* de forma a expor a visão, propósito e serviços da CTW. De seguida, há um período de aprendizagem onde a CTW fornece formação sobre manipulação e modelação de dados, serviços *cloud* e *frameworks*. Após a formação há um período de 1 semana para a integração na equipa de desenvolvimento.

Existe um período de cerca de 3 semanas, onde decorre a integração na equipa de desenvolvimento, analisando o estado atual do sistema e procede-se à aprendizagem das tecnologias atualmente em uso neste sistema de *Big Data*.

O período da fase 2 é o mais critico deste projeto de estágio consoante as dificuldades na implementação de testes pertinentes, potenciais dificuldades de acesso ou erros encontrados. Esta fase tem a duração de 6 semanas, distribuídas por 3 semanas de integração no sistema *Big Data* analisando o estado atual deste e procedendo à aprendizagem das tecnologias atualmente em uso neste sistema. Esta fase termina com execução de análises em AWS Athena, AWS Aurora e AWS RedShift em 3 semanas (1 semana para cada).

As fases 3 e 4 são as finais do projeto de estágio, ocupando 3 semanas, para a conclusão e revisão do que foi desenvolvido e efetuado ao longo do estágio.

A fase 5 relativa à realização deste documento é a fase mais extensa. Devido ao facto de ser necessário incrementar e atualizar este documento no decorrer do estágio, sempre que seja pertinente e existam novas informações relevantes, à medida que este é desenvolvido.

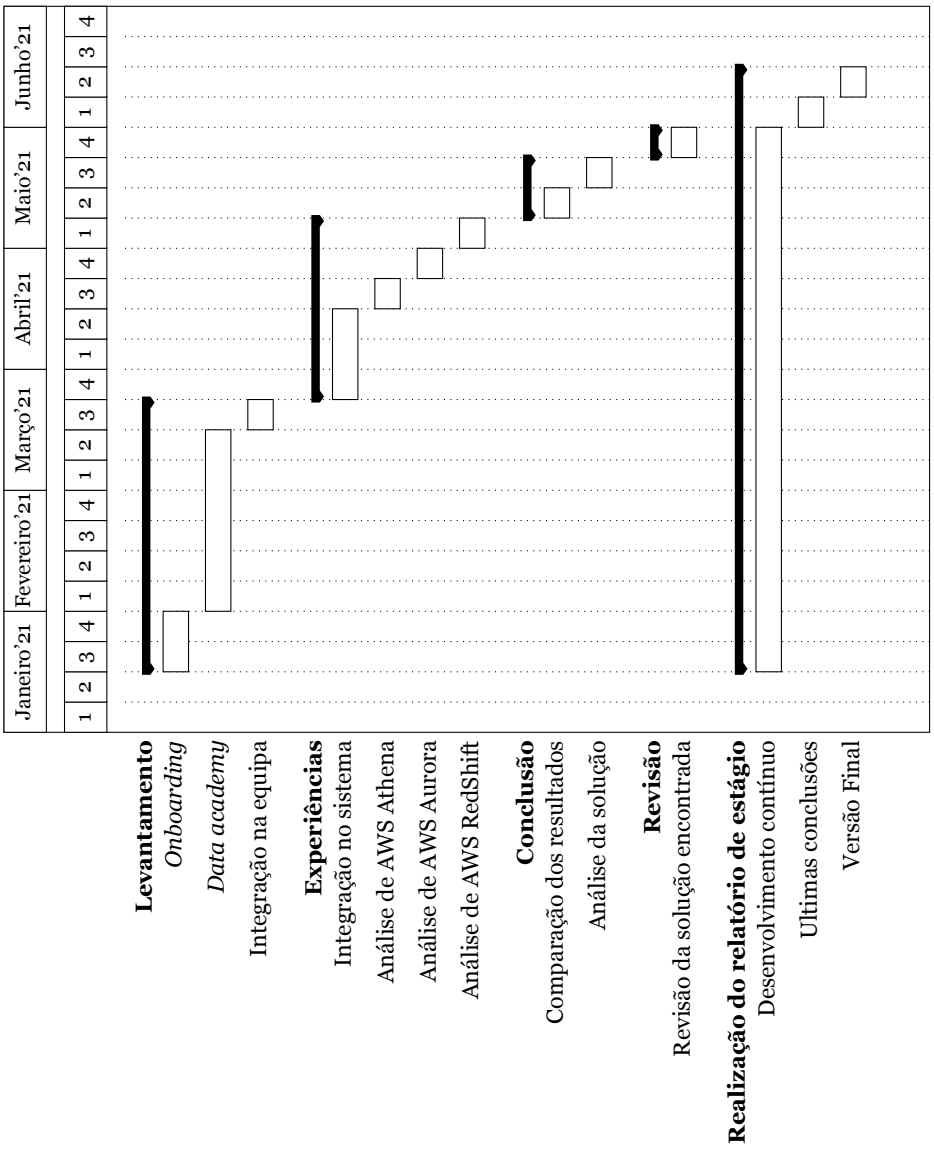

Figura 3.2: Mapa de Gantt representativo da planificação das fases do projeto de estágio com os seus respetivos períodos de tempo.

Capítulo 4

Tecnologias e Ferramentas Usadas

A implementação de um sistema de *Big Data* envolve uma vasta série de serviços e tecnologias constituintes. Assim, para se poder entender um sistema tal como o da CTW, é necessário conhecer e perceber cada componente desse sistema e como estes se relacionam.

Atualmente uma grande parte dos serviços e tecnologias utilizados fazem parte do ecossistema AWS devido à qualidade dos seus serviços e ferramentas. Há ainda uma série de serviços e ferramentas externas que é necessário analisar e entender.

Este capítulo descreve e apresenta as tecnologias utilizadas no sistema de *Big Data* da CTW com o intuito de perceber cada componente deste sistema granularmente.

4.1 AWS S3

Os sistemas de *Big Data* são centrados em dados, e por isso, é imperativo conter um serviço que consiga de forma simples e segura armazenar os dados numa grande escala. Isto porque, os dados gerados pelas fontes necessitam de um local que permita a constante cópia, servindo como repositório onde podem ser executados os trabalhos de ingestão e até mesmo como *backup*. Para isso, é utilizado o serviço armazenamento de objetos AWS S3.

O serviço AWS S3 é especialmente desenhado para oferecer escalabilidade, disponibilidade de dados, segurança e desempenho no armazenamento destes dados. São oferecidos recursos e ferramentas para que de forma facilitada se possa configurar e gerir os acessos aos dados. Devido a estas características o serviço pode ser utilizado nos mais variados casos de uso como *Data Lakes*, *websites*, aplicações para dispositivos móveis, dispositivos *Internet of Things* (IoT) e sistemas *Big Data*.

A escalabilidade dos recursos necessários para atender às constantes alterações de exigência no armazenamento de dados é feita de forma automática. Em relação à durabilidade, todos os dados armazenados no serviço AWS S3 são copiados e replicados em vários *Data Centers* dispersos por diversas regiões de disponibilidade da Amazon. É oferecida uma opção que permite guardar várias versões de cada objeto após cada alteração, havendo uma rápida e fácil recuperação a erros e falhas, porém esta funcionalidade apresenta um custo adicional. Assim, este serviço garante uma elevada disponibilidade e proteção contra falhas, erros e ameaças.

Há uma vasta rede de parceiros, tecnologias e ferramentas compatíveis com o AWS S3 oferecendo soluções integradas como armazenamento principal, *backup*, restauração e recuperação de desastres. Esta rede de parcerias é designada por *Amazon Partner*

Network (APN) e inclui empresas como a Gekko, Transact Technology Solutions e Hitachi Vantara Storage Solutions.

Existe a possibilidade de executar consultas diretamente em dados presentes em AWS S3, através de serviços como AWS Athena e AWS RedShift Spectrum. Estes serviços têm a capacidade de executar *queries* de SQL padrão diretamente, estabelecendo assim um serviço de Data Warehouse em conjunto com os recursos de AWS S3.

Os dados são armazenados como objetos em recursos chamados *buckets* e um único objeto pode ter no máximo 5 TB. Um *bucket* é um *container* de objetos, enquanto que um objeto é um ficheiro e qualquer *metadata* associada a esse ficheiro. O acesso aos objetos é feito através de S3 *Access Points* ou pelo *hostname* do *bucket*.

AWS S3 oferece uma panóplia de opções para a proteção de dados, por meio de ferramentas de gestão e controlo de acessos e por recursos de criptografia. Para o controlo de acesso há a possibilidade de filtrar o acesso público através de *Access Control List* (ACL) ou com pontos de acesso. Estes pontos de acesso permitem o controlo de acesso entre várias contas AWS a cada *bucket* ou objeto.

Os recursos de criptografia podem ser aplicados individualmente a cada bucket ou objeto por uma ferramenta de S3 em que gera chaves AES de 256 *bits* e cifra os dados automaticamente. Há adicionalmente a opção de integração com AWS KMS para que todas as tarefas de criptografia sejam executadas num serviço dedicado.

A utilização deste serviço é muito simples e intuitivo através da *interface* presente no *website* da AWS, havendo ainda a integração com a AWS *Command-Line Interface* (CLI) onde facilmente se pode gerir os recursos de S3.

Para criar um *bucket* através de AWS CLI basta realizar os seguinte comando:

```
$ aws s3 mb s3://name-of-the-bucket
```

Este comando cria um *bucket* na região onde o utilizador está inserido com a sua configuração pré-definida, ou seja, com acesso ao público completamente bloqueado e sem controlo de versões e criptografia. Para criar uma directoria dentro do *bucket* e de seguida inserir um ficheiro dentro dessa directoria, são executados os seguintes comandos:

```
$ aws s3api put-object --bucket name-of-the-bucket --key folder1/
$ aws s3 cp file.txt s3://name-of-the-bucket/folder1/
```

Finalmente para apagar um objeto e de seguida o bucket, são executados os seguintes comandos:

```
$ aws s3 rm s3://s3://name-of-the-bucket/folder1/file.txt
$ aws s3 rb s3://name-of-the-bucket --force
```

4.2 AWS IAM

Para gerir e controlar o acesso de utilizadores aos serviços e recursos de AWS é utilizado o serviço AWS IAM. Através deste serviço é possível criar utilizadores e agrupá-los, associando permissões a estes grupos para conceder ou negar o acesso individual de cada recurso AWS. Adicionalmente, permite controlar o acesso e a partilha de recursos entre recursos. Por exemplo, para um *job* de Glue poder ler e alterar um objeto no serviço S3, necessita de uma IAM *role* com a seguinte política:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name/*"
            ]
        }
    ]
}
```

Excerto de Código 4.1: Política de acesso ao recurso de S3 "arn:aws:s3:::ctw-ubi-demo/*"em código JSON com efeito de leitura ("s3:GetObject") e escrita ("s3:PutObject").

Aos utilizadores é possível atribuir credenciais de segurança individuais como palavras passe, chaves de acesso e dispositivos de autenticação multifator. As chaves de acesso são usadas para fazer chamadas programáticas para a AWS a partir do AWS CLI, ferramentas na PowerShell, AWS *Software Development Kit* (SDK)'s ou chamadas diretas na API da AWS. Estas credenciais de segurança também podem ser atribuídas temporariamente para o acesso temporário a serviços e recursos da AWS.

A utilização deste serviço não tem qualquer custo associado devido à Amazon considerar este serviço mandatório e essencial em qualquer sistema.

4.3 AWS Glue

A ingestão de grandes quantidades de dados tem como principal objetivo organizar, valorizar, validar e formatar estes dados para o seu armazenamento em serviços de *Data Warehouse* ou *Data Lake*. Com o armazenamento de dados ingeridos, é possível integrar informações de diferentes fontes e componentes de negócio para a sua análise e divulgação.

Os dados gerados pelas fontes podem ser transmitidos em tempo real (*streams*) ou inseridos em lotes (*Data Stores*). Caso os dados sejam transmitidos em tempo real, cada item é processado e importado à medida que é emitido pela fonte. No caso da inserção de dados em lotes, este processo é realizado sobre blocos discretos de dados em intervalos de tempo especificados. Qualquer dos dois processos de ingestão de dados têm de ser implementados de forma eficaz, gerindo as fontes de dados, validando e fazendo a inserção correta dos dados no destino correto.

Na camada de ingestão é utilizado o serviço de *Data Ingestion* AWS Glue para a execução automatizada de trabalhos *Extract Transform Load* (ETL). Estes trabalhos consistem em extrair os dados em bruto, transformando-os e organizando-os para formatos desejados, para que sejam armazenados em um serviço de *Data Warehouse*.

Há vários serviços compatíveis com o AWS Glue que podem ser usados como fontes de dados. O AWS Glue permite a ingestão de dados em tempo real ou em lote. Os serviços de *Data Stores* e de *Streams* compatíveis para cada um destes paradigmas são nomeadamente:

- *Data Stores* (transmissão de dados em lotes):
  - AWS S3;
  - AWS RDS;
  - Serviços de base de dados externas que suportem uma conexão JDBC;
  - Amazon DynamoDB;
  - MongoDB.
- *Streams* (transmissão de dados em tempo real):
  - Amazon Kinesis Data Streams;
  - Apache Kafka.

Após a ingestão de dados destas fontes compatíveis, é possível integrar dados em vários serviços de base de dados, *Data Wratehouses* e *Data Lakes*, estes serviços são detalhadamente:

- AWS S3;
- AWS RDS;
- Serviços de base de dados externas que suportem uma conexão JDBC;
- MongoDB.

O serviço AWS Glue foi desenhado para funcionar com dados estruturados e semi-estruturados, pois utiliza um componente para a abstração de dados chamado *dynamic frame*. Este componente deteta os formatos dos dados e organiza-os em linhas e colunas sem ser necessário definir um esquema inicial. Assim, há uma flexibilidade acrescida no

esquema a definir e são apresentadas sugestões de transformação projetadas para cada um dos tipos de dados organizados dinamicamente.

Para o complexo processo de ingestão de dados, AWS Glue está dividido em componentes, cada um destes componentes tem uma função e um propósito e a sua relação é visível na imagem 6.5. Estes componentes são nomeadamente os seguintes:

- **Conexão**: um objeto do *Data Catalog* que contém todas as propriedades necessárias para realizar uma conexão a uma certa entidade de um serviço compatível. Serve principalmente para auxiliar um *job* na sua tarefa de extração de dados de uma fonte, ou inserção de dados num serviço alvo. Este objeto pode consistir, como por exemplo, num *Uniform Resource Identifier* (URI) com credenciais de *login* AWS e informação relativa à VPC;
- **Segurança**: um objeto do *Data Catalog* que contém todas as propriedades necessárias para garantir a proteção de dados estáticos e em trânsito, contendo por exemplo, informação relativa às chaves geradas pelo serviço KMS;
- ***Crawler***: programa que se conecta a um ficheiro e, por meio de classificadores, determina o formato dos dados apresentando o esquema e criando tabelas de *metadata*. Estes classificadores são compatíveis com formatos de ficheiros como CSV, JSON, Apache Avro e *Extensible Markup Language* (XML). Os *Crawler* podem ser executados diretamente em bases de dados relacionais através de uma ligação JDBC, havendo também a possibilidade de criar um classificador através da implementação de padrões Grok ou através de especificar linhas em documentos XML;
- ***Jobs***: *scripts* em Python3 (com PySpark) ou Scala para a manipulação de dados nas várias fases ETL segundo a lógica de negócio. Iniciam através de um *trigger*, com a extração de dados das fontes, execução de código de transformação e manipulação, finalizando com a inserção no serviço alvo;
- ***Triggers***: planeamento de *jobs* e *crawlers* para as fases ETL despoletando a sua execução. Estas execuções podem ser agendadas para haver a sua execução cíclica num espaço de tempo. Podendo ser ativadas através de eventos como a finalização da execução de outro *job* ou *crawler* criando uma cadeia de execução autónoma, ou a presença de novos dados nas fontes. Finalmente, estes podem ser executados manualmente para fins de correção e verificação de erros.

AWS Glue contém um repositório chamado *AWS Glue Data Catalog*, tem como principal objetivo armazenar tabelas de *metadata* produzidas pelos *crawlers*. Este repositório também armazena *scripts* de conexão, configurações de segurança, *crawlers* e *triggers*. Enquanto que, os *scipts* dos *jobs* são automaticamente armazenados num *bucket* S3.

4.3.1 Python3 e PySpark

Os *jobs* de Glue para a transformação dos dados contêm código na linguagem Python com recurso à sua biblioteca PySpark. Python é uma versátil linguagem de alto nível e

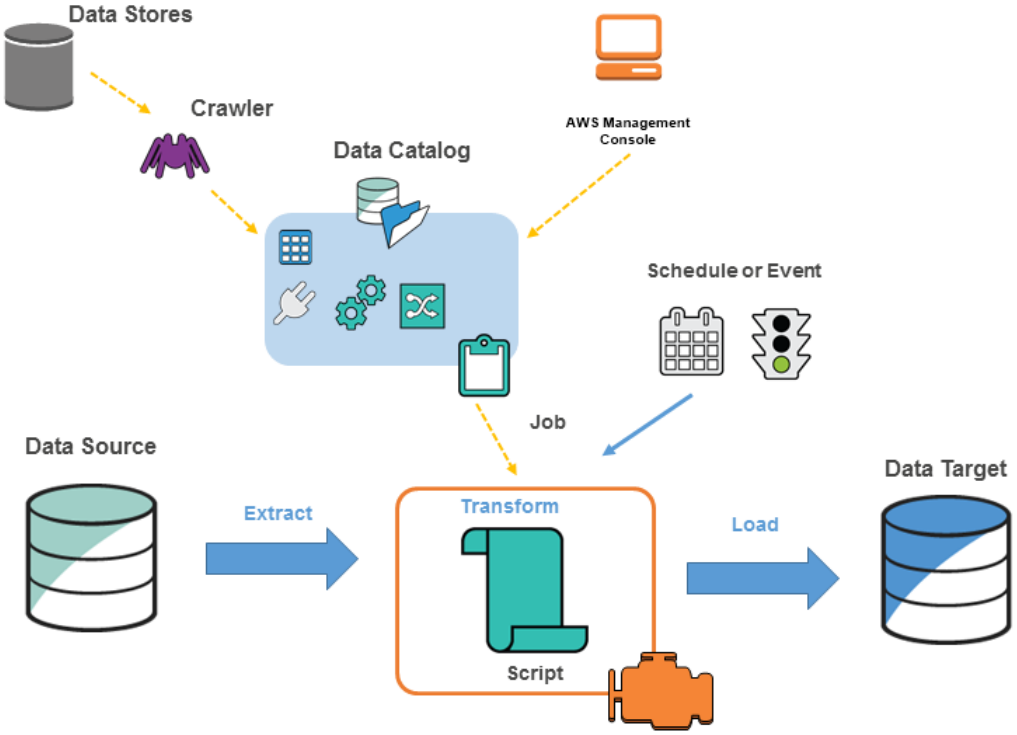

Figura 4.1: Diagrama representativo da relação entre as componentes de Glue no processo de ingestão de dados [Ama21a].

normalmente usada em sistemas de *Big Data* pela sua simplicidade e velocidade no processamento de dados.

Apache Spark é uma *framework* de código aberto para computação distribuída em grandes quantidades de dados, sendo escrito na linguagem de programação Scala. Devido às suas características é normalmente usada em sistemas de *streaming*, *machine learning* e análise de dados.

PySpark nasceu através de uma colaboração entre Apache Spark e Python. Apesar de PySpark ser uma biblioteca de Python, esta funciona como uma API de Python para Spark. Com Pyspark podemos usar funções semelhantes a código SQL ou até mesmo executar código SQL puro, isto é possível devido a um conjunto de funções de PySpark designado de PySparkSQL. As funções servem para aplicar funções e analisar grandes quantidades de dados estruturados ou semiestruturados.

PySparkSQL introduziu o DataFrame, uma representação tabular de dados estruturados que é semelhante a uma tabela de um sistema de gestão de banco de dados relacional. Estes DataFrames podem ser dados diretamente carregados de tabelas, onde são executadas as operações, originando um novo DataFrame que finalmente é transformado novamente numa tabela. No caso do Glue, os dados recebidos de objetos S3 são carregados em DataFrames e após as transformações nos dados, são inseridos novamente em S3 no formato CSV ou Parquet.

Um exemplo do funcionamento de um *job* em Glue com Python e PySpark é o exerto de código 4.2.

```
#Importar modulos de PySpark
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

#Importar modulos de Glue
from awsglue.utils import getResolvedOptions
from awsglue.context import GlueContext
from awsglue.dynamicframe import DynamicFrame
from awsglue.job import Job

#Inicializar a sessão
spark_context = SparkContext.getOrCreate()
glue_context = GlueContext(spark_context)
session = glue_context.spark_session

#Definir parametros
glue_db = "DataCatalog-DB"
glue_tbl = "table-name"
s3_write_path = "s3://bucket-name/write/"

#Carregar dados do DataCatalog para um DataFrame
data_frame_read_demo = glue_context.create_dynamic_frame.from_catalog(database
    = glue_db, table_name = glue_tbl)

#Fazer operação sobre o DataFrame originando a um novo
data_frame_write_demo = data_frame_read_demo.groupBy("column-name1").agg(
    F.count("*").alias("new_name1"),
    F.sum(data_frame_read_demo["column_name2"].cast("long")).alias("new_name2")
)

#Imprimir as 10 primeiras linhas do novo DataFrame
data_frame_write_demo.show(10)

#Guardar o DataFrame em formato Parquet
glue_context.write_dynamic_frame.from_options(
    frame = data_frame_write_demo,
    connection_type = "s3",
    connection_options = {
        "path": s3_write_path,
    },
    format = "Parquet"
)
```

Excerto de Código 4.2: Exemplo de código Python com a biblioteca PySpark na realização de um trabalho ETL simples.

4.3.2 Workflow

Para obter um processo complexo de ETL é preciso um conjunto de *crawlers*, *jobs* e *triggers*, podendo envolver dezenas ou centenas de componentes. Para ajudar na criação e visualização destes processos complexos, o Glue disponibiliza uma ferramenta chamada *workflow*. Esta ferramenta apresenta através de uma interface gráfica o grafo de relação entre os componentes de Glue e permite adicionar e ligar novas componentes.

Um exemplo de um *workflow* em Glue é o que está apresentado na figura 6.5. Tal como é visível neste exemplo, um *workflow* inicia normalmente com um *trigger* programado para executar de forma cíclica num intervalo de tempo definido. Neste caso, o *trigger* ao iniciar, despoleta a execução de dois *jobs*. De seguida, há novamente um *trigger*, este inicia segundo um evento, sendo este evento a garantia de conclusão de todos os componentes anteriores. A existência deste *trigger* permite controlar o *workflow* porque há uma elevada probabilidade dos componentes anteriores terem tempos de execução distintos. Por fim, o *trigger* despoleta um *crawler* que verifica os dados e deteta alterações nos dados e atualiza as tabelas no *Data Catalog*. O Workflow ao executar cada componente regista o processo e estado de execução para fins de deteção de falhas.

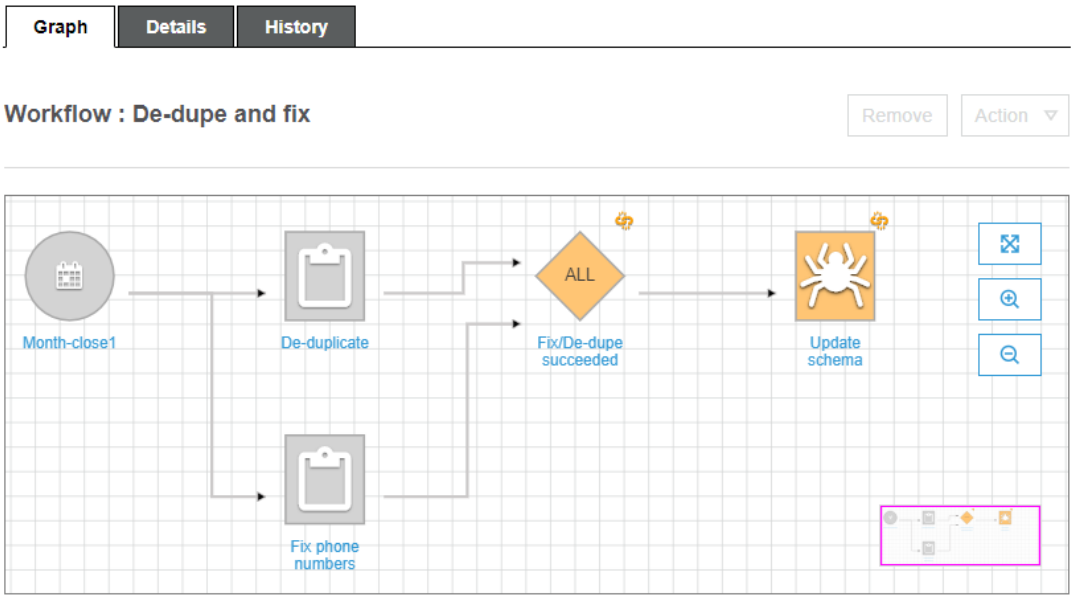

Figura 4.2: Exemplo de um Workflow de Glue onde é apresentado o grafo com a cadeia de execução das componentes do Glue no processo ETL [Ama21b]

4.4 AWS Kinesis

AWS Kinesis é um serviço de *streaming* que tem como objetivo recolher e processar grandes quantidades de dados em tempo real. Este serviço possibilita assim a criação de *streams* escaláveis e sem servidor. Assim, é possível obter em tempo real acesso a novos dados e reagir a estes rapidamente. Kinesis oferece recursos essenciais para processar dados de *streaming* em qualquer escala de forma económica e flexível através da existência de vá-

rias ferramentas de criação e configuração de *streams* que se adequam a cada caso de uso. As ferramentas que Kinesis oferece, são nomeadamente:

- **Kinesis Data Streams**: esta ferramenta viabiliza a criação de *streams* que recebem dados de uma fonte em tempo real e carrega-os num *buffer* como registos em formato JSON. Cada registo pode conter no máximo 1 MB. Estas *streams* necessitam de configuração em *shards*. Quando esta *stream* é chamada ativa um iterador que lê e transmite os dados em *buffer* de cada *shard*. Cada *buffer* é escalável e durável. Esta ferramenta é utilizada em sistemas onde há vários serviços ou aplicações a utilizar a mesma *stream* de forma concorrente.
- **Kinesis Data Firehose**: esta ferramenta de criação de *streams* recebe e publica dados em tempo real. Os possíveis destinos dos dados são qualquer serviço do ecossistema AWS (como S3 ou RedShift) ou outro serviço externo que contenha um *Hypertext Transfer Protocol* (HTTP) *endpoint* (como LogicMonitor ou MongoDB). Cada registo não pode ter um tamanho superior a 1,000 *Quilobyte* (kB). Em comparação com Kinesis Data Streams, esta ferramenta não necessita de configuração em *shards*, sendo assim menos personalizável. Esta ferramenta é usada, como por exemplo, em sistemas de IoT ou sistemas de sistemas *Big Data*.
- **Kinesis Video Streams**: permite criar *streams* de transmissão e processamento de vídeo em tempo real. Estas *streams* recebem e divulgam vídeo *frame* a *frame* com baixa latência. Esta transmissão pode ser feita diretamente para um serviço no ecossistema AWS ou é possível construir *websites* e aplicações que façam chamadas à API para divulgar o vídeo em direto. Há muitos métodos de gerar dados para estas streams, tais como: vídeos captados por uma *webcam* usando a biblioteca GStreamer ou por uma câmara numa rede através de *Real Time Streaming Protocol* (RTSP).
- **Kinesis Data Analytics**: possibilita criar *streams* que apliquem código SQL diretamente nos dados logo que estes são enviados para a stream. Os dados destas stream podem originar de outras streams tais como Kinesis Data Firehose e Kinesis Data Streams, e servem para servir aplicações de *dashboards* ou métricas em tempo real.

Para um sistema *Big Data* com vários ambientes de desenvolvimento é mais adequado usar *streams* de Kinesis Data Streams porque há a propagação destes dados para vários ambientes em simultâneo. Como uma *stream* de Kinesis Data Firehose apenas consegue enviar dados para um ambiente, seria necessário definir várias *streams* num sistema com vários ambientes, o que se torna menos vantajoso. Por outro lado, não são utilizadas *streams* criadas através da ferramenta Kinesis Data Analytics porque as transformações complexas executadas nos dados não são suportadas por este serviço.

Uma *stream* de Kinesis Data Streams pode ser facilmente criada utilizando a interface gráfica no *website* da AWS ou através de AWS CLI. Sendo apenas necessário definir um nome para a *stream* e o número de *shards* que a *stream* vai utilizar. Estes *shards*

são uma unidade de processamento dado pela Amazon. Cada *shard* pode ler no máximo cinco transações por segundo com o tamanho de leitura máximo de 5 MB por segundo. Em capacidade de escrita, cada *shard* consegue escrever no máximo 1000 registos por segundo, ou uma capacidade máxima de escrita de 1 MB por segundo. O comando AWS CLI para a criação de uma *stream* com 3 *shards* é o seguinte:

```
$ aws kinesis create-stream \
  --stream-name "nome-stream" \
  --shard-count 3
```

Com esta *stream* completamente funcional já se pode iniciar o envio de registos para esta *stream*. O comando AWS CLI para enviar um registo é o seguinte:

```
$ aws kinesis put-records \
  --stream-name "nome-stream" \
  --data '{ "chave1" : "valor1", "chave2" : "valor2" }'
```

Esta operação também pode ser facilmente efetuada através de um *script*. Uma das linguagens compatíveis com o serviço Kinesis é Python, esta linguagem pode aceder aos recursos de Kinesis ou de qualquer outro serviço AWS através da biblioteca boto3. A biblioteca boto3 cria um *client* com o serviço desejado e já se pode fazer chamadas a esse *client*, tal como é exemplificado no excerto de código 4.3.

```
#Importar as bibliotecas necessárias
import boto3
import json

#Definir o boto3 client com acesso ao serviço "Kinesis"
kinesis_client = boto3.client("kinesis", region_name = "eu-central-1")

#Definir o registo a inserir na stream
payload = {
    "chave1" : "valor1",
    "chave2" : "valor2"
}

#Enviar o registo para a stream
kinesis_client.put_record(
    StreamName = "nome-stream",
    Data = json.dumps(payload)
)
```

Excerto de Código 4.3: Exemplo de código Python com a biblioteca boto3 para o envio de um registo para uma stream Kinesis.

Ao enviar um registo para uma *stream* Kinesis, esta armazena em *buffer* código JSON o registo e informação útil adicional e detalhada relativa ao envio deste registo.

Estas informações contêm valores como: um número inteiro relativo ao tempo de chegada ”approximateArrivalTimestamp”e uma *string* com o identificador da IAM *role* do utilizador que efetuou o envio do registo ”invokeIdentityArn”. O excerto de código 4.4 contém um exemplo de código JSON gerado armazenado em *buffer* com os dados enviados e informações adicionais.

```
{
  "Records": [
    {
      "kinesis": {
        "partitionKey": "partitionKey-03",
        "kinesisSchemaVersion": "1.0",
        "data": { "chave1" : "valor1", "chave2" : "valor2" },
        "sequenceNumber": "49545115243490985018280067714973144582180062593244200961",
        "approximateArrivalTimestamp": 1428537600
      },
      "eventSource": "aws:kinesis",
      "eventID": "shardId-000000000001:49545115243490985018280067714973144582180062593244200961",
      "invokeIdentityArn": "arn:aws:iam::iam-role-name",
      "eventVersion": "1.0",
      "eventName": "aws:kinesis:record",
      "eventSourceARN": "arn:aws:kinesis:EXAMPLE",
      "awsRegion": "eu-central-1"
    }
  ]
}
```

Excerto de Código 4.4: Exemplo de código JSON gerado e armazenado automaticamente após o envio de um registo para uma *stream* Kinesis.

O processo de receção dos registos enviados para uma *stream* é feito de forma individual para cada *shard*. Inicialmente é necessário saber o valor inteiro que identifica cada um dos *shards*. De seguida, com o identificador é necessário declarar um *shard iterator*. Este iterador é um valor relativo à posição em memória onde se vai começar a ler os registos sempre que a *stream* é invocada. Assim, é possível começar iterativamente a receber registos. O primeiro comando AWS CLI listar os *shards* e os seus identificadores é o seguinte:

```
$ aws kinesis list-shards \
      --stream-name "nome-stream"
```

Com o identificador do *shard* é possível obter o *iterator*. Também é necessário o tipo de *iterator* a utilizar, o mais comum é o ”LATEST”que irá receber os registos mais

recentes. O *iterator* pode ser obtido através do comando:

```
$ aws kinesis get-shard-iterator \
      --stream-name "nome-stream" \
      --shard-id shardId-000000000001 \
      --shard-iterator-type LATEST
```

Por fim, para finalmente receber os registos através do *shard iterator* é necessário executar o seguinte comando:

```
$ aws kinesis get-records \
      --shard-iterator AAAAAAAAAAF7/omWD7IuHj1yGv/
          TKuNgx2ukD5xipCY4cy4gU96orWwZwcSXh3K9tAmGYeOZyLZrvzzeOFVf9iN99hUPw/
          w/boYWYeehfNvnf1DYt5XpDJghLKr3DzgznkTmMymDP3R+3wRKeuEw6/
          kdxY2yKJHoveaiekaVc4N2VwK/GvaGP2Hh9Fg7N++q0Adg6fIDQPt4p8RpavDbk+
          A4sL9SWGE1
```

Este processo de receber os registos de uma *stream* também pode ser feito recorrendo a um *script* semelhança ao excerto de código 4.3. Assim, recorrendo à biblioteca boto3 de Python é possivel fazer este processo com o excerto de código 4.5.

```
#Importar as bibliotecas necessárias
import boto3
import time

#Definir parametros
my_stream_name = 'nome-stream'

#Definir o boto3 client com acesso ao serviço "Kinesis"
kinesis_client = boto3.client('kinesis', region_name='eu-central-1')

#Receber o indentificador da stream
response = kinesis_client.describe_stream(StreamName=my_stream_name)

#Receber a string com o identificador do primeiro shard da stream
my_shard_id = response['StreamDescription']['Shards'][0]['ShardId']

#Receber um dicionário com as posição do shard em memoria (buffer) onde se lê
    os registos sequencialmente
shard_iterator = kinesis_client.get_shard_iterator(StreamName=my_stream_name,
                                                   ShardId=my_shard_id,
                                                   ShardIteratorType='LATEST
                                                       ')

#Receber a string do dicionário relativo à posição do shard
my_shard_iterator = shard_iterator['ShardIterator']

#Ler os registos presentes no shard
```

```
record_response = kinesis_client.get_records(ShardIterator=my_shard_iterator,
                                             Limit=2)

#Imprimir o registo
print(record_response['Records'][0]['Kinesis']['data'])
```

Excerto de Código 4.5: Exemplo de código Python com a biblioteca boto3 que recebe registos de uma *stream* Kinesis através de um dos seus *shards*.

4.5 AWS Lambda

O ecossistema AWS contém um serviço de computação sem servidor que permite correr código sem qualquer tipo de configuração. Este serviço é o AWS Lambda e é desenhado para executar curtas cargas de trabalho, estas têm um máximo de 15 minutos de execução e uma memória escalável entre 128 MB até 3 GB. Além de poder executar qualquer código personalizado, é possível definir dependências e bibliotecas numa das linguagens compatíveis com Lambda. Lambda é capaz de compilar código Java, JavaScript, dotNet, Go, Ruby ou Python.

A ferramenta Lambda *functions* permitem criar instâncias com *scripts* de código. Estas instâncias escalam os seus recursos automaticamente, e apenas é cobrado o tempo de execução de uma função ao ser ativada, não sendo cobrado qualquer valor enquanto a instância está inativa. A invocação de uma instância Lambda pode ser feita através da API ou em respostas a eventos de outros serviços AWS. Este serviço pode ser usado para como por exemplo, processar dados vindos de *streams* de Kinesis ou processar dados provenientes de S3 ou DynamoDB.

O serviço Lambda fornece ferramentas para ajudar no desenvolvimento e *debug* de código. Para ajudar no desenvolvimento de código, Lambda oferece uma série de *blueprints* e funções pré feitas e customizáveis para cada caso de uso. Enquanto que para o *debug* de código, oferece uma ferramenta de testes automatizados e personalizáveis para simular o envio de dados e eventos para uma instância através de código JSON como o da imagem 4.4.

Aquando da utilização de uma instância, o serviço Lambda disponibiliza recursos para monitorizar a atividade e utilização de cada instância. Todas as solicitações a cada instância são automaticamente armazenadas e registadas em *logs*. Para a monitorização das instâncias são apresentadas numa *interface* um conjunto de gráficos com métricas de monitorização em tempo real. Estas métricas vão desde o número de execuções concorrentes até à percentagem de sucesso/falha nas invocações de cada instância.

A componente principal de código presente numa instância é a função Handler, esta função é o pedaço de código chamado sempre que a instância é invocada. Este Handler recebe dois parâmetros designados de evento e contexto. O evento é informação passada à instância, inclui todos os dados e metadados que o código necessita para poder processar logicamente os dados. Por outro lado, o contexto é um conjunto de informações que permitem ao código interagir com o ambiente Lambda, isto inclui dados da entidade que

invoca a instância e o o nome do ficheiro *log* onde é registada a atividade da invocação. O excerto de código 4.6 exemplifica código Python numa Lambda *function* que recebe e imprime registos de uma *stream* Kinesis.

```
#Importar as bibliotecas necessárias
import base64
import json

#Função Handler
def lambda_handler(event, context):
    #Percorrer os registos recebidos
    for record in event['Records']:
        # Decodificar os registos da stream Kinesis que estão em base 64
        payload = base64.b64decode(record['kinesis']['data'])
        # Imprimir os registos
        print("Decoded payload: " + payload)
```

Excerto de Código 4.6: Exemplo de código Python que recebe e imprime registos de uma *stream* Kinesis.

Cada instância do serviço Lambda pode conter camadas, cada camada é composta por um ficheiro comprimido que contém bibliotecas ou dependências. Estas camadas servem para centralizar código usado em várias instâncias e facilitar o desenvolvimento de código. Cada instância pode ter no máximo 5 camadas e o conjunto dos 5 ficheiros descompactados podem ter no máximo 250 MB.

Para criar uma Lambda *function* através de CLI é necessário definir um nome, um ficheiro compactado que contém todo o código, o nome da função *Handler*, a linguagem de compilação e a IAM que dá permissões à *function*. Assim, o comando fica no seguinte formato:

```
$ aws lambda create-function \
  --function-name function-name \
  --zip-file fileb://lambda-code.zip \
  --handler index.handler \
  --runtime python3.8 \
  --role arn:aws:iam::iam-role-name
```

Por outro lado, a operação reversa para eliminar uma Lambda *functions* através de CLI é a seguinte:

```
$ aws lambda delete-function \
  --function-name my-function
```

Para enviar registos para uma Lambda *functions* apenas é necessário definir o nome da função que recebe os registos e o *payload* que contém os registos. O comando fica da seguinte forma:

```
$ aws lambda invoke \
  --function-name function-name \
```

```
    --payload '{"action": "insert", "number": 3, "product": "car"}'
```

4.6 Terraform

Terraform é uma ferramenta para criar, alterar e manter infraestruturas com segurança e eficiência. Isto é possível porque Terraform pode gerir recursos e serviços de vários provedores *cloud* como AWS, Azure, Docker, *Google Cloud Platform* (GCP) e *Oracle Cloud Infrastructure* (OCI).

O método utilizado por Terraform para gerir uma infraestrutura é através da sua descrição usando uma linguagem de alto nível com sintaxe de configuração. Ou seja, Terraform é uma ferramenta *Infrastructure as Code* (IaC) que permite criar e atualizar *blueprint's* de uma infraestrutura. Estas *blueprint's* são usadas para fazer *deploy* da infraestrutura num serviço ou datacenter.

Os *scripts* de configuração Terraform contêm granularmente definido cada componente necessário para criar um sistema ou aplicação. Por exemplo, uma infraestrutura que necessite de recursos e serviços AWS tais como S3 ou DynamoDB, define em código Terraform a configuração/configurações destes recursos, tal como é exemplificado no código 4.7.

```
# Criar um bucket s3
resource "aws_s3_bucket" "nome_recurso_bucket" {
  bucket = "bucket-name"
  acl    = "private"

  versioning {
    enabled = "true"
  }
}

# Criar uma tabela em DynamoDB
resource "aws_dynamodb_table" "nome_recurso_dynamodb" {
  name           = "dynamodb-table-name"
  read_capacity  = 5
  write_capacity = 5
  hash_key       = "LockID"

  attribute {
    name = "LockID"
    type = "S"
  }
}
```

Excerto de Código 4.7: Exemplo de código Terraform para a criação de dois recursos, um *bucket* S3 e uma tabela em DynamoDB.

Uma infraestrutura gerida por Terraform pode conter tanto componentes de baixo nível como de alto nível. As componentes de baixo nível suportadas incluem instâncias de computação, armazenamento e rede. Enquanto que, componentes de alto nível suportadas contêm entradas de *Domain Name System* (DNS) e recursos de *Software as a Service* (SaaS).

Ao executar o *script* de configuração, Terraform gera um plano de execução que descreve o que fará para atingir o estado desejado e, de seguida, executa esse plano para construir a infraestrutura descrita. Caso haja alterações na configuração, o Terraform deteta as alterações e cria um novo plano de execução incremental sem reescrever o plano anterior.

Após a execução do planeamento, Terraform cria um grafo de recursos demonstrando todos os recursos utilizados e as suas relações, e paraleliza a criação e modificação de quaisquer recursos não dependentes. Assim, Terraform cria e mantém a infraestrutura da forma mais eficiente possível. Através da criação de plano de execução e grafos de recursos, os utilizadores têm conhecimento das mudanças feitas por Terraform e a sua ordem a cada versão.

4.7 Bitbucket

Bitbucket Cloud é uma ferramenta de repositório de código baseada em Git, construída para auxiliar a colaboração de equipas profissionais. O uso desta ferramenta é vantajosa para uma equipa conseguir colaborar no código, desde o conceito até à sua versão final, permitindo ainda a execução de testes automatizados para conseguir assegurar um código confiável e de qualidade.

A integração de Bitbucket com outras ferramentas como Jira e Confluence, fazem desta ferramenta uma excelente escolha para qualquer equipa de desenvolvimento de *software*. Jira é uma ferramenta de planeamento e gestão de funcionalidades para equipas que utilizem uma metodologia Agile. Confluence é uma ferramenta para a organização e documentação de *software*.

Bitbucket também é compatível com o ecossistema AWS, ou seja é possível conectar um repositório Bitbucket a uma conta AWS. Com esta conexão, sempre que há alterações no código presente no repositório, este cria e executa automáticamente uma instância AWS Lambda que copia o código do repositório para um serviço AWS tal como S3 através de um *bucket*.

4.8 AWS CodeBuild

AWS CodeBuild é um serviço de CI que permite a construção completa e a manutenção de uma infraestrutura em *cloud*. Este serviço compila código-fonte, executa testes unitários e produz artefactos definidos pelo código-fonte. Estes artefactos são objetos de vários serviço, tais como *buckets* S3, Glue *jobs* ou instâncias Lambda. A utilização deste serviço

elimina a necessidade de configurar servidores e *software* de compilação e de aplicar manualmente atualizações no *software*.

Este serviço escala automaticamente os seus recursos consoante a exigência da construção da infraestrutura. CodeBuild pode gerir várias solicitações simultâneas, podendo executar várias construções de infraestruturas em paralelo. A utilização deste serviço é comum quando há vários ambientes de desenvolvimento, automatizando alterações e atualizações na configuração de cada ambiente.

4.9 AWS CodePipeline

Para qualquer sistema que esteja em constante evolução e modificação, é necessário automatizar atualizações, como novas funcionalidades e correção de erros. Por isso, é utilizado AWS CodePipeline, este serviço de CD automatiza todo o processo de entrega de *software* desde o repositório até ao cliente. Esta automação é feita através da criação de uma *pipeline* completamente personalizável para se adaptar a cada caso de uso.

Uma *pipeline* é composta por uma sequência de *stages*, cada *stage* é definida e configurada segundo as necessidades e casos de uso. Um *stage* pode conter por exemplo uma fase de aprovação manual ou uma série de testes de integração. Um exemplo de ciclo de entrega *software* usando uma *pipeline* com vários *stages* é a presente no diagrama 4.3.

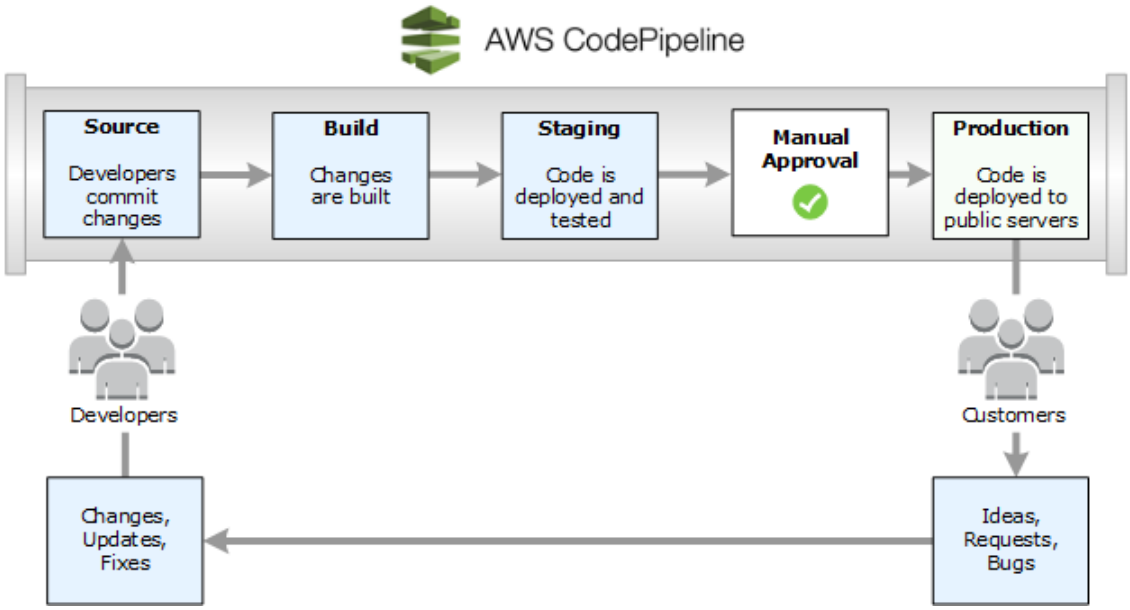

Figura 4.3: Exemplo de uma *pipeline* criada através de CodePipeline para o processo CD [AWS21].

Capítulo 5

Sistema *Big Data*

Este capítulo procura apresentar e analisar o funcionamento do sistema de *Big Data* CTW no qual este projeto se insere. Assim, é descrito o funcionamento do sistema desde a origem dos dados até à entrega ao cliente, utilizando as tecnologias e ferramentas apresentadas no capítulo 4.

A finalidade deste sistema é apresentar ao cliente uma plataforma onde este pode consultar dados ingeridos para os introduzir num serviço de Business Intelligence para a sua análise, como Tableu ou Sisense. O cliente pertence ao sistema financeiro da BMW no Reino Unido e os resultados obtidos na análise são usados para suportar a tomada de decisões de negócio naquela região.

De seguida, cada secção apresenta uma diferente componente deste sistema *Big Data*, iniciando com a apresentação dos dados tratados por este sistema. É apresentado a alteração na modelação de dados entre OLTP e OLAP, sendo esta uma das principais transformações nos dados. De seguida, é apresentado o processo ETL com o fluxo de dados desde a sua origem até ao seu armazenamento num serviço de *Data Warehouse*, usando principalmente AWS S3 e Glue. Por fim, o processo de entrega continua de *software* através do CI/CD é demonstrado, recorrendo principalmente aos serviços AWS CodeBuild e CodePipeline.

5.1 Fontes de dados

Este sistema está inserido no sistema financeiro da BMW no Reino Unido, tal como é apresentado no capítulo 1, logo há uma constante produção de dados nas áreas de negócio do grupo BMW. Estes dados incluem informações acerca de vendas de veículos do grupo BMW, incluindo marcas subsidiárias como a Mini e a Rolls-Royce Motor Cars, constituindo estas no maior volume e fluxo de dados para o sistema. Esta produção é originada em várias fontes como o *website* da BMW no Reino Unido (visível na imagem 5.1) ou em concessionários naquela região. Estes dados consistem em informações como o modelo de carro vendido e a sua configuração, dados pessoais do cliente, métodos de pagamento e contrato financeiro.

Os dados deste sistema são confidenciais porque contêm informações sensíveis e privadas. Por isso, neste capítulo apenas é apresentada informação concetual para questões meramente representativas e para proteger os clientes BMW de possíveis fugas de informação.

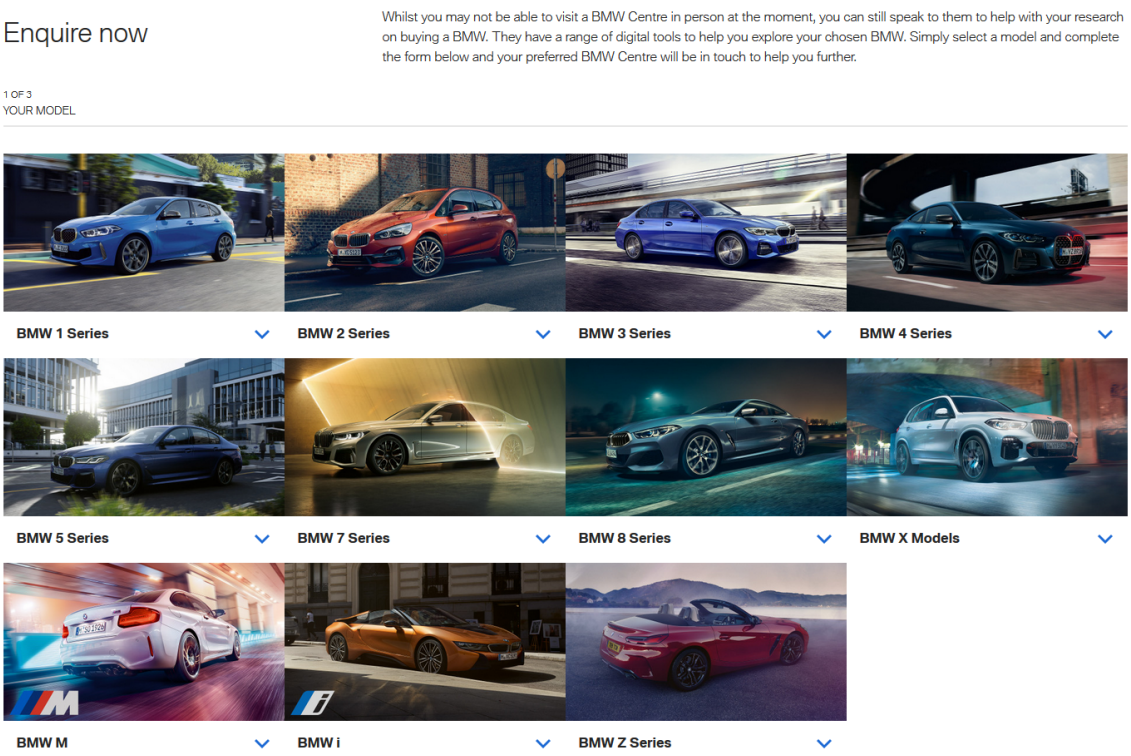

Figura 5.1: Página web com a seleção do modelo BMW a escolher pelo cliente na compra do automóvel [BMW21b].

5.2 Modelação de dados

5.2.1 OLTP e OLAP

Os dados presentes nas fontes são resultados de processos de negócios, destes processos resultam dados de operações em esquema OLTP. Porém, apesar deste esquema estar otimizado para haver um elevado volume de transações, este não é o mais adequado à analise de dados para o apoio da tomada de decisões de negócio. Para esta análise é necessário transformar os dados em modelo OLTP para um modelo OLAP.

Os esquemas do modelo OLTP normalmente contêm uma grande quantidade de tabelas e são desenhados para processar rapidamente grandes quantidades de transações em dados normalizados, e procura responder à pergunta: ”Quem comprou X?”. O problema de usar o modelo OLTP num serviço de *Data Warehouse* para a análise dos dados é que seria necessário aplicar várias funções de junção para obter a relação dos dados. Estas funções de junção tais como ”INNER JOIN”e ”OUTER JOIN”, são computacionalmente árduas de executar, especialmente com grandes quantidades de dados.

Para combater as lacunas que dificultam a análise de dados em modelo OLTP, é feita uma transformação para o modelo OLAP. Os esquemas deste modelo contêm poucas tabelas com grandes volumes de dados desnormalizados, e procura responder à pergunta: ”Quantas pessoas compraram X?”.

5.2.2 Tipos de tabelas em OLAP

Um modelo OLAP difere de OLTP principalmente pela sua estrutura. Na estrutura OLAP as tabelas são classificadas por tipos, cada tipo tem uma estrutura e propósito diferente. Os tipos de tabelas existentes em OLAP são nomeadamente os seguintes:

- **Tabelas factuais** : estas tabelas são usadas para conectar e analisar fontes de dados heterogéneas. Os dados nestas tabelas consistem em medições, métricas e factos de um processo de negócio. Assim, cada linha destas tabelas simboliza uma operação em negócio. A qualidade ou eficiência de cada processo de negócio pode ser medida com uma métrica chamada *Key Performance Indicator* (KPI). Esta é fundamental porque uma grande quantidade de análises feitas nos dados têm em considerar esta métrica na tomada de decisão;
- **Tabelas de dimensões** : são uma estrutura que categoriza os factos e métricas das tabelas factuais e responde a questões específicas de negócio. O principal objetivo destas tabelas é adicionar contexto a cada operação de negócio (linha da tabela factual) contendo atributos descritivos. Estas tabelas normalmente contêm informação específica sobre os clientes, produtos, endereços, etc;
- **Tabelas de Entidade associativa** : este tipo de tabela serve para resolver relações de muitos-para-muitos. Assim é usado em casos em que uma tabela de dimensão tem vários valores associados à tabela factual, ou quando há vários valores para o mesmo atributo.

5.2.3 Esquemas OLAP

Os dados de um modelo OLAP são tipicamente armazenados num esquema em estrela ou floco de neve num serviço de *Data Warehouse*. Estes dois esquemas diferem em *design*, normalização de dados e complexidade de consultas.

Um esquema em estrela, tal como é apresentado na figura 5.2, consiste em uma ou mais tabelas factuais a indexar múltiplas tabelas de dimensão. Normalmente os dados nas tabelas factuais incluem informação numérica como preço, tempo da operação, quantidades e entre outros. Já as tabelas de dimensão incluem informações não contáveis como localização geográfica, dados de clientes e dados de vendedores.

Por outro lado, o esquema em floco de neve , tal como apresentado na figura 5.3, difere do esquema em estrela pela normalização de dados presentes nas tabelas de dimensão. Assim, este esquema é considerado uma estrutura multi-dimensional, ou seja, há tabelas de dimensão a indexar outras tabelas de dimensão. A passagem de um esquema em estrela para um esquema em floco de neve consiste assim em dividir as tabelas de dimensão.

Devido à normalização parcial dos dados, os esquemas em floco de neve ocupam menos espaço em memória comparado aos esquemas em estrela. Pois, os dados normalizados contêm muito menos informação redundante e assim há uma redução de memória utilizada. Dados num esquema em floco também aparentam ser mais organizados sendo

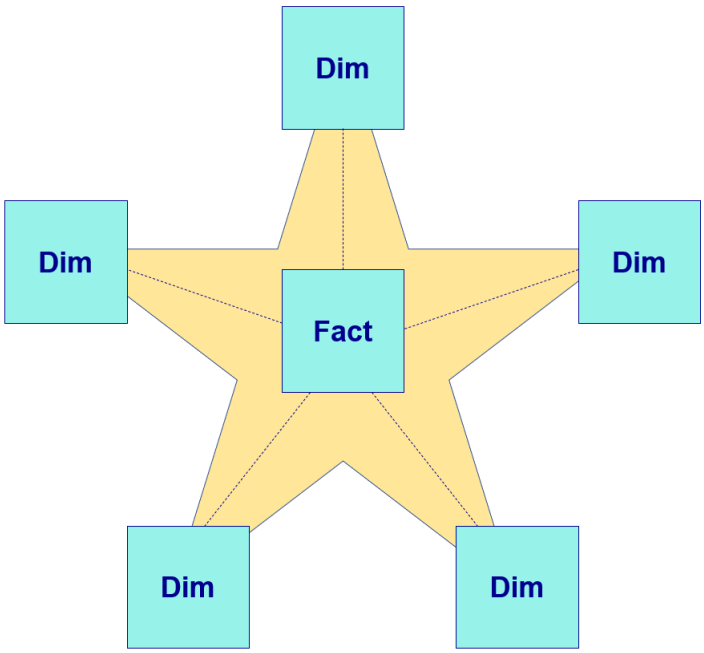

Figura 5.2: Representação da relação de tabelas factuais e de dimensão, num modelo OLAP com um esquema em estrela.

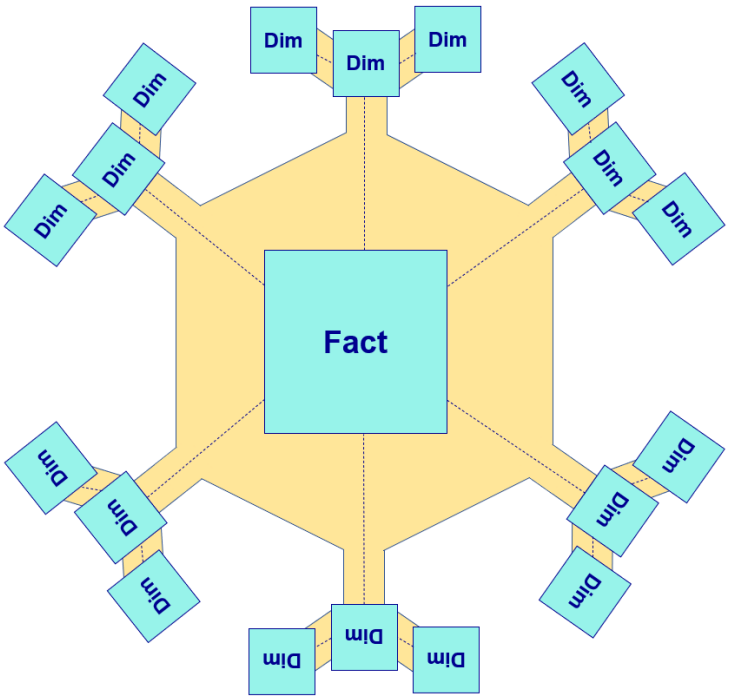

Figura 5.3: Representação da relação de tabelas factuais e de dimensão, num modelo OLAP com um esquema em floco de neve.

mais fácil a sua leitura e interpretação. No processo desnormalização de dados podem existir problemas de integridade dos mesmos, o que complica futuras modificações e manutenções. Porém, a utilização de um esquema em floco de neve aumenta a complexidade das *queries* executadas sobre os dados, visto que ao aumentar o numero de tabelas, aumentando novamente o número de funções de agregação necessárias para a consulta dos dados.

5.3 Processo ETL

A componente principal de um sistema *Big Data* é o processamento de dados. A este processamento de dados é atribuído o nome processo ETL, este processo é responsável por fazer transformações tais como alteração de esquema OLTP para OLAP e transformações no formato dos dados de acordo com os requisitos do cliente. No projeto *Big Data* CTW onde este projeto se insere, o processo ETL está dividido em duas fases. Estas fases são apresentadas nas subsecções seguintes.

5.3.1 Primeira fase

A primeira fase consiste na extração direta de dados nas fontes para um *bucket* S3 tal como demonstrado no diagrama 5.4. Esta extração é feita através de um *job* no serviço AWS Glue em Python com uma ligação JDBC, que faz uma cópia exata dos dados brutos no seu formato original para o *bucket*. A esta fase também é designada de *Raw Layer* porque contém os dados em bruto, nos quais as transformações vão ocorrer. Este processo de extração pode ser iniciado manualmente ou é iniciado ciclicamente numa série de tempo recorrente.

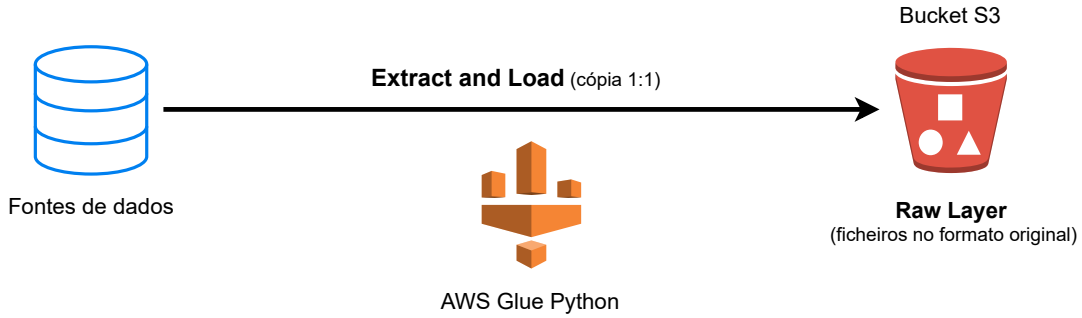

Figura 5.4: Diagrama de extração de dados das fontes através da execução de *jobs* em AWS Glue.

Em alternativa, existe a possibilidade de extração de dados em tempo real. Ou seja, assim que estes dados são gerados são enviados para um *stream* no serviço AWS Kinesis. Ao receber registos, a *stream* invoca uma instância de AWS Lambda que envia e armazena esses registos num objeto de um *bucket* S3, designado *Raw Layer*. Isto possibilita ao cliente ter acesso aos dados mais rapidamente, porém esta alternativa ainda está em desenvolvimento. Este processo é representado no diagrama 5.5.

5.3.2 Segunda fase

A partir dos dados na *Raw Layer* é iniciada a segunda fase. Esta fase transforma os dados e o seu esquema através de um *workflow* no serviço Glue composto por *triggers*, *crawlers* e *jobs* em Python e com as bibliotecas PySpark e boto3. De seguida, estes dados transformados são inseridos em formato Parquet num *bucket* S3. Por fim, são executados *crawlers* para a deteção de tabelas em Glue *Data Catalog*, estas tabelas permitem executar solicitações e *queries* SQL sobre os dados. Esta fase está representada no diagrama 5.6.

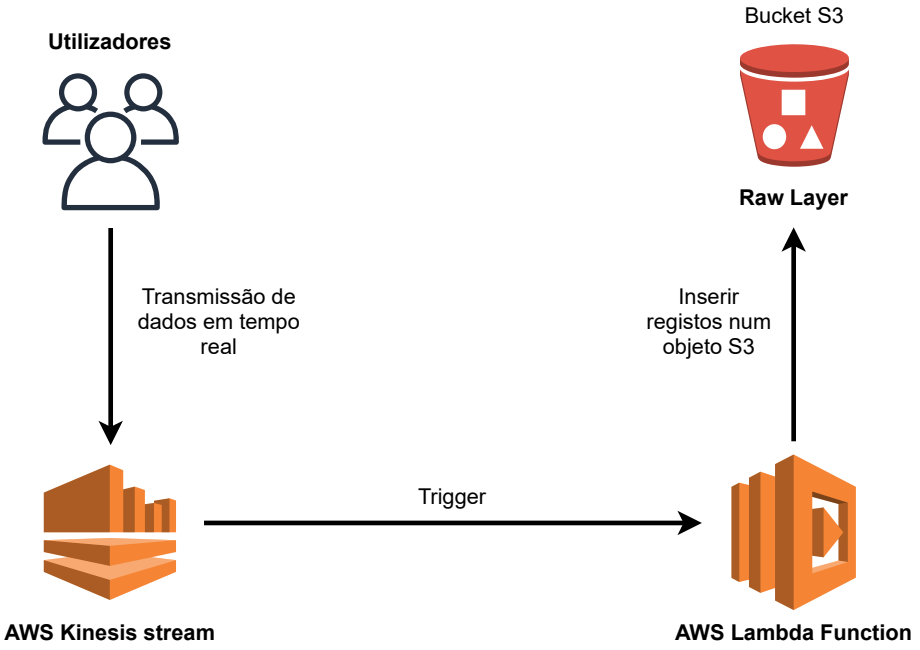

Figura 5.5: Diagrama de extração de dados das fontes em tempo real através da execução de uma *stream* em AWS Kinesis.

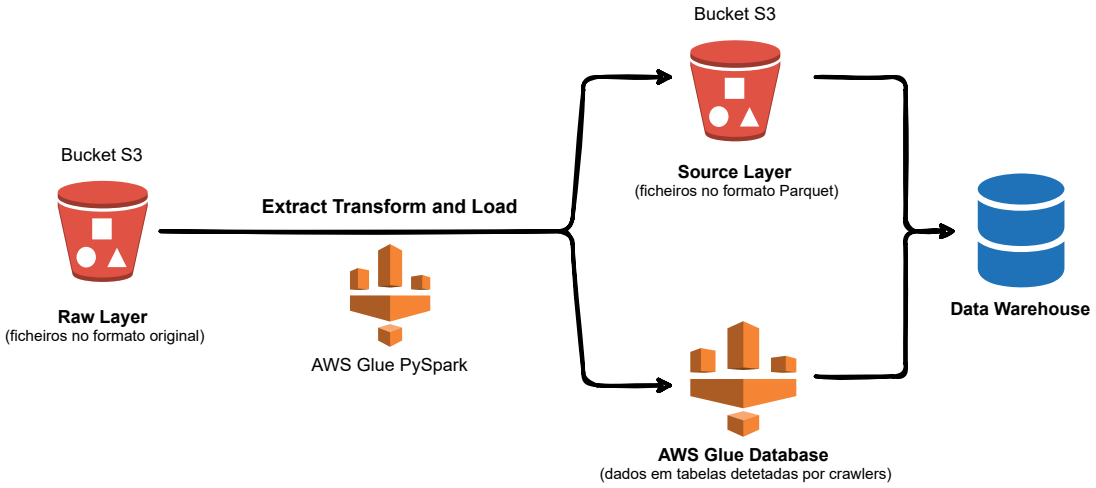

Figura 5.6: Diagrama do processo de transformação dos dados (ETL) usando AWS Glue.

5.4 CI/CD

Atualmente, qualquer aplicação ou sistema tem de evoluir rapidamente para ir ao encontro dos requisitos do cliente. Alterar e melhorar *software* a um ritmo acelerado para corresponder a estes requisitos é um componente essencial de qualquer negócio. A rápida atualização do sistema permite inovar e responder a mudanças no mercado mais rapidamente, o que leva a uma melhoria nos resultados de negócio. Como o sistema *Big Data* CTW tem um impacto direto na tomada de decisões, este necessita de ser ágil e de se adaptar rapidamente aos requisitos apresentados de forma eminente.

Os processos de entrega de *software* tradicionais que envolvem a integração manual dos incrementos desenvolvidos, são mais difíceis de gerir e mais susceptíveis a erros. As-

sim, para a coordenação de uma equipa no constante desenvolvimento de *software* e na entrega do mesmo, é usado um processo CI/CD. Este processo é constituído pela componente CI e CD, cada uma destas componentes tem um propósito:

- **CI**: é uma prática de desenvolvimento que coordena a equipa no controlo de versões após a entrega de código no mesmo local/ambiente. Cada incremento de código é construído no ambiente e testado para rápida deteção de erros ou falhas. Esta prática permite a uma equipa trabalhar em várias funcionalidades em paralelo sem haver conflitos no código;
- **CD**: esta metodologia é responsável por automáticamente lançar e distribuir o constante trabalho da equipa de desenvolvimento. Cada mudança de código é automáticamente verificada, testada e integrada em ambiente de produção. Antes do trabalho desenvolvido ser integrado em produção terá de haver sempre uma verificação, normalmente manual, se o incremento corresponde aos requisitos do cliente e se não existem erros.

Para conseguir aplicar uma processo completo de CI/CD são definidos 3 ambientes: desenvolvimento, integração e produção. O ambiente de desenvolvimento é onde a equipa desenvolve código, enquanto que o ambiente de integração é onde são efetuados testes para perceber se as novas funcionalidades vão de encontro aos requisitos dos clientes. Por fim, o ambiente de produção é onde está o produto final em que o cliente tem acesso.

Para CI, é criado código Terraform com a configuração de todos os artefactos necessários para a infraestrutura. Este código está presente num repositório BitBucket e quando há um *push request* despoleta o serviço AWS CodeBuild. Este serviço recebe o código Terraform e executa-o, construindo os artefactos declarados no código para cada um dos ambientes. Este processo de CI está demonstrado no diagrama da figura 5.7.

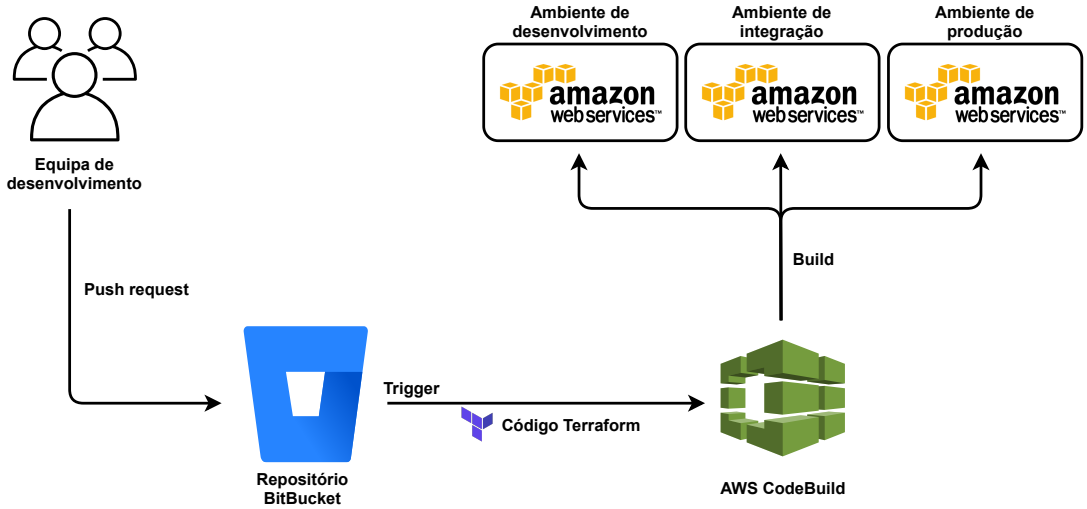

Figura 5.7: Processo de configuração dos ambientes de desenvolvimento através de código Terraform e CodeBuild.

Um dos artefactos mais importantes na construção de um ambiente é o *workflow*

de Glue. Apesar de Terraform conseguir criar o *workflow* e os seus componentes (*crawlers*, *triggers* e *jobs*), o código Python dos *jobs* não está contido no código Terraform. Este código está presente num *bucket* S3, ou seja, Terraform necessita primeiro de criar o *bucket*, de seguida é inserido o código nesse *bucket*, e por fim Terrafom pode criar os *jobs* do *workflow* através dos ficheiros no *bucket*.

A inserção de código Python num *bucket* S3 é parte integrante um processo CI. Esta inserção está representada no diagrama 5.8. À semelhança do código Terraform, o processo de CI inicia com um *push request* de código para o repositório BitBucket por parte da equipa de desenvolvimento. O serviço BitBucket invoca uma instância de Lambda que reescreve os *scripts* de código no *bucket*, permitindo a sua utilização pelo serviço Glue.

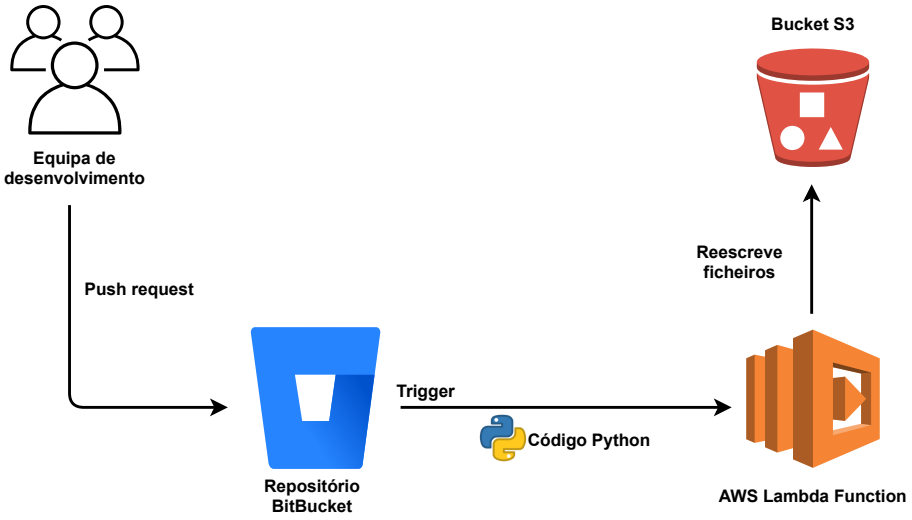

Figura 5.8: Processo de integração continua de código Python.

O processo CI em cada ambiente está dependente de CD. Para este processo CD é utilizado o serviço AWS CodeBuild, este serviço contém a *pipeline* que gera a relação entre os três ambientes. A pipeline inicia com a construção dos artefactos no ambiente de desenvolvimento usando CodeBuild e, após a aprovação manual de todas as alterações feitas por cada um dos elementos da equipa de desenvolvimento, inicia a mesma construção em ambiente de integração. Neste ambiente, o código passa por uma série de testes automatizados. Com os resultados obtidos nos testes a equipa de desenvolvimento verifica se o código corresponde aos requisitos do cliente e aprova manualmente. Por fim, após a aprovação de código em integração, o mesmo é inserido no ambiente de produção recorrendo novamente ao serviço CodeBuild. Este processo está representado no diagrama da figura 5.9.

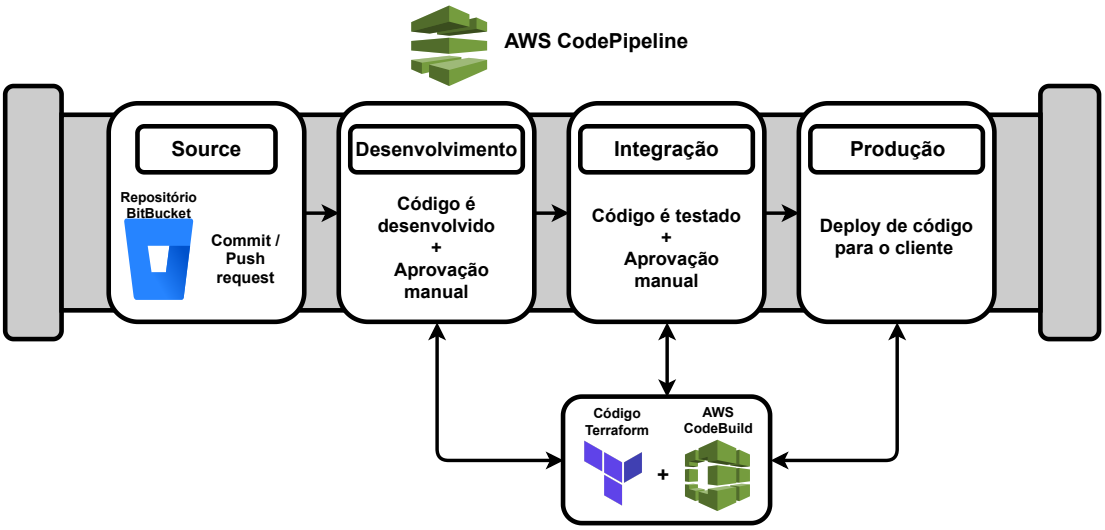

Figura 5.9: Exemplo de uma *pipeline* no serviço CodeBuild para o processo de CI/CD.

Capítulo 6

Comparação e análise dos serviços de *Data Warehouse* escolhidos

Através da análise de serviços de *Data Warehouse* presente no capítulo 2 e após a análise dos componentes integrantes de um sistema de *Big Data* presente no capítulo 4, é possível iniciar a integração e testagem dos serviços de *Data Warehouse* escolhidos. Com a integração e testagem dos serviços AWS Athena, Aurora e RedShift permite uma análise mais profunda das vantagens e desvantagens de integrar cada um destes serviços no sistema de *Big Data*.

Inicialmente é feita a integração e configuração (caso necessário) destes serviços no sistema alvo com exemplos e demonstrações. De seguida, é feita uma análise ao seu desempenho através de solicitações às bases de dados mediante da execução de *queries* SQL. É apresentada uma análise às propriedades de *Atomicidade, Consistência, Isolamento e Durabilidade* (ACID) presentes ou não nestes serviços seguido de um levantamento de funcionalidades adicionais proporcionadas por estes serviços.

6.1 Integração e configuração

6.1.1 AWS Athena

Athena é um serviço bastante simples de integrar no sistema visto que tem uma ligação direta com o *Data Catalog* do serviço Glue. Isto é, todas as tabelas detetadas pelos *crawlers* executados em Glue são automaticamente utilizáveis e imediatamente apresentadas na *interface* de Athena. Assim, este serviço não necessita de nenhuma configuração, uma vez que, gera e executa automaticamente o código DDL através de metadados e parâmetros detetados por *crawlers*, permitindo realizar *queries* imediatamente após a sua execução. Um exemplo de código DDL gerado e executado pelo serviço Athena após a execução de um *crawler* é o execerto de código 6.1.

```
CREATE EXTERNAL TABLE `customers_csv`(
  `coluna_1` bigint,
  `coluna_2` string,
  `coluna_3` bigint,
  `coluna_4` string,
  `coluna_5` string,
  `coluna_6` bigint,
  `coluna_7` bigint,
  `coluna_8` double)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS INPUTFORMAT  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
```

```
LOCATION 's3://bucket_name/read_object'
TBLPROPERTIES (
  'CrawlerSchemaDeserializerVersion'='1.0',
  'CrawlerSchemaSerializerVersion'='1.0',
  'UPDATED_BY_CRAWLER'='crawler_name',
  'areColumnsQuoted'='false',
  'averageRecordSize'='56',
  'classification'='csv',
  'columnsOrdered'='true',
  'compressionType'='none',
  'delimiter'=',',
  'objectCount'='1',
  'recordCount'='8699',
  'sizeKey'='487182',
  'skip.header.line.count'='1',
  'transient_lastDdlTime'='1615389220',
  'typeOfData'='file')
```

Excerto de Código 6.1: Código SQL gerado e executado após a execução de um *crawler*, com toda a *metadata* e parâmetros detetados por este.

Este serviço disponibiliza uma *interface* onde é possível executar *queries* SQL, esta ferramenta não é limitada a *queries* com baixas quantidades de dados resultantes ou com baixo tempo de execução e é completamente funcional sem necessitar de recorrer a um serviço ou ferramenta externa.

6.1.2 AWS Aurora *serverless*

Para usar o serviço Aurora, é necessário inicialmente configurar uma instância de RDS. Ao criar esta instância tem de se optar por usar o motor Aurora *serverless*, e de seguida, definir a capacidade de processamento mínima e máxima em vCPU. A escalabilidade entre estes valores é gerida de forma automática ao longo da necessidade e exigência das solicitações feitas ao serviço. As definições de VPC, *backup* e encriptação assumem valores pré-definidos sendo possível fazer alterações nestes parâmetros e configurações a qualquer momento.

As várias opções de configuração de capacidade de processamento variam entre 1 vCPU com 2 GB de memória *Random Access Memory* (RAM) e 256 vCPU com 488 GB de memória RAM. Respetivamente, o preço por hora destas configurações varia entre 0,20€ (euros) e 10,71€ (euros).

Ao contrário do serviço Athena, o serviço Aurora necessita primeiro de migrar dados presentes em S3 para a instância. Esta migração pode ser feita de duas formas distintas: através de um *job* em Glue ou *query* de migração entre serviços AWS.

Para fazer migração de dados por meio de um *job* em Glue usando Python, o método é muito semelhante ao processo ETL e até pode ser inserido no mesmo. Para isto, é necessário carregar os dados para um *DataFrame* e, de seguida, inserir na instância através de uma ligação ao *endpoint* ou através da utilização da biblioteca boto3. Finalmente, a execução deste *job* é moroso (dependendo também da capacidade de processamento da

instância), sendo que para uma instância com 2 vCPU e 4 GB de memória RAM a migração de dados com cerca de 18 GB por meio de um *job* sem qualquer transformação, demora aproximadamente 3 horas e 15 minutos.

Em alternativa à utilização de um *job* em Glue, existe a possibilidade de executar a migração de dados através de *query* SQL. Neste caso, Aurora reconhece o comando S3 com um URI de um objeto S3, no qual os dados vão ser migrados. É ainda necessário atribuir uma IAM *role* à instância com permissão de leitura sobre os objetos de um *bucket* para permitir esta migração. Um exemplo de uma *query* com a função S3, é o seguinte:

```
LOAD DATA FROM S3 's3://bucket_name/read_object'
INTO TABLE 'aurora_database'.'table_name';
```

Excerto de Código 6.2: Exemplo de *query* de migração de dados de um objeto S3 para uma tabela numa instância Aurora.

Aurora através da página do RDS oferece uma interface onde é possível escrever e executar *queries*, porém esta interface é limitada e só permite a execução de *queries* com pouco volume de dados (100 MB) e pouco tempo de execução (5 minutos). Assim, é necessário recorrer em ferramentas externas de manuseamento de base de dados como o DBeaver, DbVisualizer ou *SQL Server Management Studio* (SSMS).

6.1.3 AWS RedShift e Spectrum

Para utilizar o serviço RedShift ou RedShift Spectrum é necessário, em primeiro lugar, configurar uma instância. Ambos os serviços podem operar simultaneamente na mesma instância e até podem ser usados na mesma *query*, ou seja, é possivel, como por exemplo, executar uma query junção entre uma tabela Spectrum e uma tabela RedShift. Cada instância pode ter um máximo de 128 nós havendo uma série de configurações de *hardware* disponíveis para vários casos de uso. Estas configurações variam entre 2 vCPU com 160 GB de armazenamento *Solid State Drive* (SSD) e 48 vCPU com 128 TB de armazenamento SSD por nó. Assim, com estas configurações o preço estimado varia entre 191,77 € (euros) e 9 093,38 € (euros) a cada nó.

À semelhança de AWS Aurora *serverless* a migração de dados para uma instância RedShift pode ser feita através de um *job* no serviço AWS Glue usando Python e boto3, ou com a execução de uma *query*.

No caso de Spectrum, este executa *queries* diretamente nos dados em S3, sem necessitar de carregar dados para a instância RedShift. Assim, é apenas necessário declarar através de código DDL o formato da tabela, a sua origem e um conjunto de propriedades. Um exemplo de código SQL para a criação de uma tabela em Spectrum é o seguinte:

```
create external table schema-name.table-name(
  `coluna_1` bigint,
  `coluna_2` varchar,
  `coluna_3` bigint,
  `coluna_4` varchar,
  `coluna_5` varchar,
  `coluna_6` bigint,
```

```
  `coluna_7` bigint,
  `coluna_8` float)
row format delimited
fields terminated by ','
stored as textfile
location 's3://bucket_name/read_object'
table properties ('numRows'='200000000');
```

Excerto de Código 6.3: Exemplo de uma *query* que permite ao Spectrum aceder a dados num objeto S3.

Por outro lado, para usar o serviço RedShift normal, é necessário migrar dados para a instância. Isto é possível com recurso à função "COPY"de SQL sobre o URI do objeto S3 onde os dados estão inseridos. Nesta *query* há um parâmetro obrigatório que é o IAM *role* que dá permissão de leitura à instância sobre os objetos de S3. Assim a *query* tem o seguinte formato:

```
copy  schema-name.table-name
from 's3://bucket_name/read_object'
iam_role 'arn:aws:iam::iam-role-name'
csv;
```

Excerto de Código 6.4: Exemplo de *query* em Spectrum para aceder a uma tabela num objeto S3.

Para a execução de *queries* e solicitações às bases de dados, RedShift apresenta uma interface onde se podem executar, porém as *queries* de consulta são limitadas a 100 MB de dados. Por isso, à semelhança de AWS Aurora, é necessário usar um serviço externo para realizar consultas.

6.2 Desempenho

Num sistema de Big Data onde o principal objetivo é a análise de grandes quantidades de dados, esta análise é feita por norma através da execução de *queries* onde há um grande número de verificações e operações. O problema da execução destas *queries* é o seu desempenho, havendo a necessidade do tempo da sua execução não afetar ou atrasar a análise dos dados. Segundos de execução de *queries* representam dias em negócios, assim devido à vasta quantidade de negócios BMW, é imperativo conseguir analisar o máximo número de operações de negócio no menor tempo possível.

6.2.1 Desempenho entre configurações e serviços

Para uma comparação entre as várias tecnologias e serviços, é necessário em primeiro lugar perceber o quanto as várias configurações de cada serviço alteram o desempenho do mesmo. Athena é o único serviço em análise que não necessita de qualquer configuração servido como comparação relativa aos outros serviços.

6.2.1.1 RedShift e RedShift Spectrum

Como RedShift e RedShift Spectrum podem atuar sobre a mesma instância, é feita uma comparação direta entre ambos. Assim, é efectuada a execução de 3 *queries* com diferentes funções, estas *queries* são representativas da análise comum feita nestes dados. A execução destas *queries* é feita na mesma instância, ou seja, ambos partilham a mesma configuração de *hardware*.

Após a execução de *queries* iguais sobre RedShift e Spectrum, os valores de tempo de execução obtidos estão representados na tabela 6.1 e graficamente apresentados no gráfico 6.2.

Tabela 6.1: Tempos de execução para a comparação de desempenho entre os serviços Spectrum e RedShift nativo.

SQL	Tempo de execução com RedShift Spectrum (segundos)	Tempo de execução com RedShift (segundos)
Query 1	13	12
Query 2	13	8
Query 3	12	7

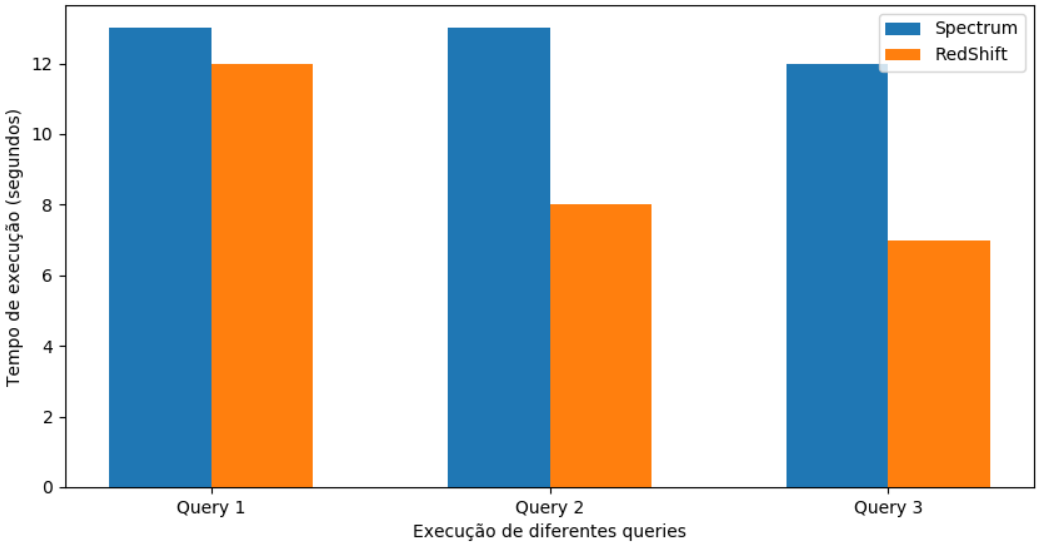

Figura 6.1: Gráfico comparativo de desempenho entre execução de *queries* em RedShift ou através da utilização de Spectrum.

Os resultados obtidos demonstram uma superioridade do serviço RedShift em relação à sua ferramenta Spectrum, já que para todas as *queries* executadas, RedShift teve um menor tempo de execução. Assim é possível concluir que executar *queries* em dados migrados para uma instância RedShift é mais rápido que realizar as mesmas *queries* diretamente em dados presentes em S3. Ou seja, considera-se que RedShift tem um melhor desempenho em relação à sua ferramenta Spectrum.

Para obter uma melhor perspetiva acerca no desempenho de RedShift no sistema, são executadas *queries* com uma configuração de RedShift mais aproximada do contexto do sistema. Assim, é criado um *cluster* tendo em conta o orçamento e a capacidade de armazenamento dos nós. O *cluster* criado é do tipo ”ds2.xlarge”com 4 nós, cada um com 2 TB de armazenamento *Hard Disk Drive* (HDD), 4 vCPU e 31 GB de memória RAM, totalizando um custo total previsto de 2457,99€ (euros) por mês. Ao executar 3 diferentes *queries* neste *cluster* e na *interface* Athena, os resultados obtidos são apresentados na tabela 6.2 e representados na imagem 6.2.

Tabela 6.2: Tempos de execução para a comparação de desempenho entre os serviços Spectrum e RedShift nativo.

SQL	Tempo de execução com Athena (segundos)	Tempo de execução com RedShift (segundos)
Query 1	953	1419
Query 2	487	586
Query 3	480	726

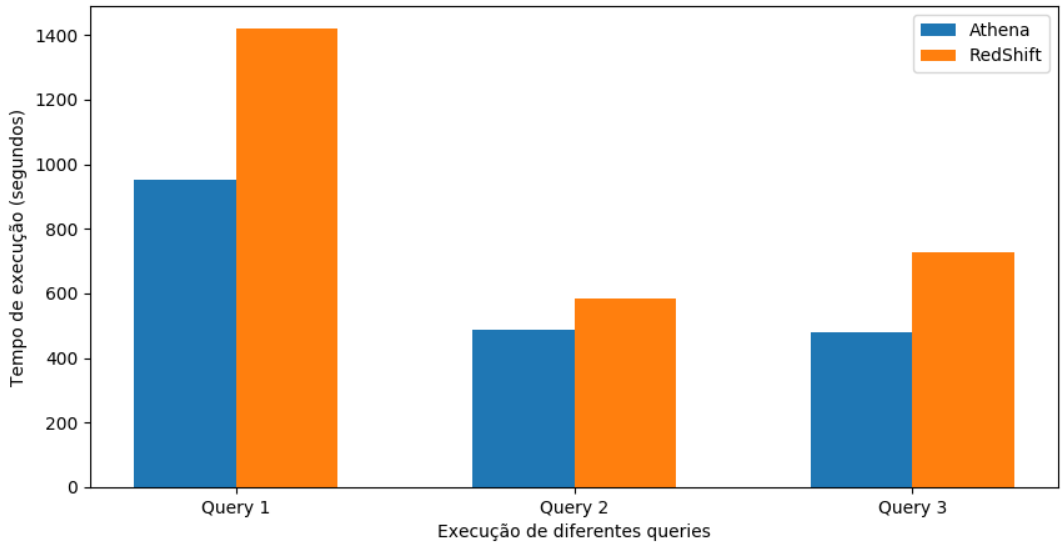

Figura 6.2: Gráfico comparativo de desempenho entre execução de *queries* na *interface* Athena ou num *cluster* Redshift.

Os resultados obtidos demonstram que para todas as *queries* executadas, Athena teve um desempenho melhor. Assim, é possível afirmar que para a configuração de RedShift usada, o seu desempenho é inferior ao de Athena. Em relação a custos, a instância de RedShift utilizada apresenta um custo de 2457,99€ (euros) mensais, em comparação com os 114,09€ (euros) estimados pelo uso de Athena.

6.2.1.2 Aurora

Para analisar o desempenho de Aurora, é necessário inicialmente verificar o desempenho das várias configurações possíveis. Por conseguinte, são definidas 3 instâncias de RDS com o serviço Aurora com diferentes configurações de *hardware*. Os parâmetros de mínima e máxima capacidade de processamento têm o mesmo valor, para não haver escalabilidade na capacidade de processamento obter valores granulares e fixos das configurações escolhidas. Estas configurações são designadamente:

- Aurora2: 2 vCPU e 4 GB memória RAM;
- Aurora64: 64 vCPU e 122 GB memória RAM;
- Aurora256: 256 vCPU e 488 GB memória RAM;

Com as configurações apresentadas, e após a migração de dados para estas instâncias, é iniciada a análise de desempenho destas configurações. É esperado haver uma melhoria significativa de desempenho consoante o aumento do número de vCPU. Assim, são executadas 3 *queries* diferentes em cada uma das instâncias. Os tempos de execução destas *queries* estão apresentados na tabela 6.3 e graficamente apresentados no gráfico 6.3.

Tabela 6.3: Tempos de execução para a comparação de desempenho entre várias configurações de Aurora *serverless*.

SQL	Tempo de execução com Aurora2 (segundos)	Tempo de execução com Aurora64 (segundos)	Tempo de execução com Aurora256 (segundos)
Query 1	811	381	308
Query 2	762	261	248
Query 3	722	244	220

Os resultados obtidos indicam uma escalabilidade de desempenho consoante o aumento de poder de processamento de cada configuração. De Aurora2 para Aurora64 houve em média um aumento de 57% em desempenho, enquanto que de Aurora64 para Aurora256 houve uma melhoria de 9,67%.

Apesar de haver uma boa redução gradual de tempo de execução à medida que o número de vCPU aumenta a cada configuração, estes tempos de execução são muito elevados. Para ter uma comparação desta desigualdade, as mesmas *queries* usadas para comparar as configurações são executadas em Athena e os seus resultados apresentados na tabela 6.4. Através dos resultados obtidos pode concluir-se que, mesmo com a melhor configuração, Aurora tem um desempenho muito abaixo de Athena. Este baixo desempenho revela que Aurora não é adequado a um sistema de *Big Data*, sendo assim um grande impedimento para a sua escolha na inclusão do produto final.

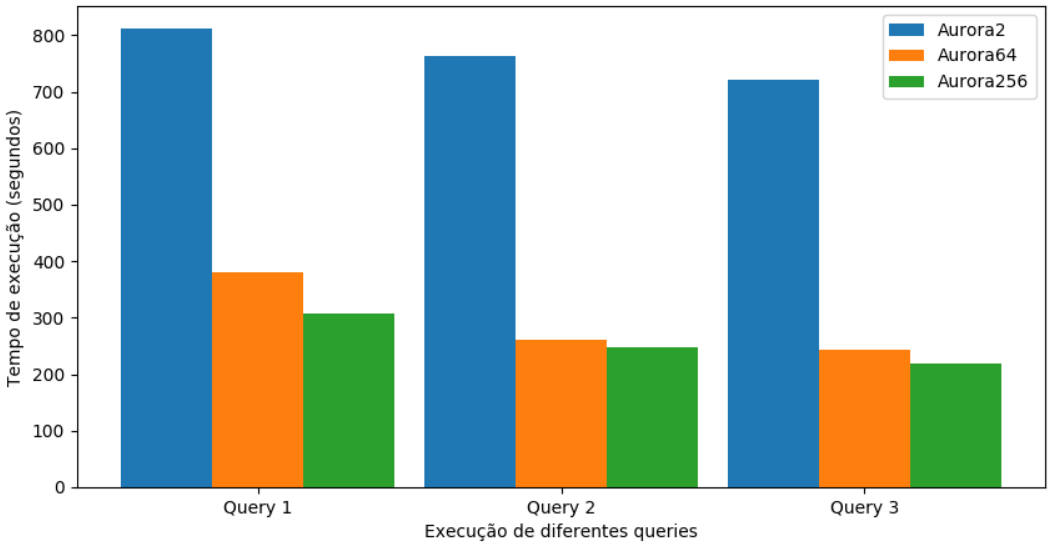

Figura 6.3: Gráfico comparativo de desempenho entre várias configurações de Aurora *serverless*.

Tabela 6.4: Tempos de execução para a comparação de desempenho entre os serviços Aurora e Athena.

SQL	Tempo de execução com Aurora256 (segundos)	Tempo de execução com Athena (segundos)
Query 1	308	4,29
Query 2	248	3,73
Query 3	220	3,22

6.2.2 *Cache*

Uma contribuição comum para o desempenho global de um serviço de base de dados é o uso de um sistema de *caching*. Este consiste na alocação em memória de componentes de *queries* executadas anteriormente. Com isto, é parcialmente evitado o cálculo total de *queries* executadas consecutivamente através da informação recolhida anteriormente, melhorando assim o desempenho. Os três serviços em análise contêm todos sistemas de armazenamento em *cache* distintas.

Por um lado, o serviço Aurora apesar de implementar automaticamente este sistema de *cache*, não apresenta qualquer tipo de melhoria notória na execução de *queries* consecutivas. Assim, adicionando esta deficiência aos insuficientes resultados de desempenho apresentados, faz deste serviço uma escolha cada vez mais difícil.

Por outro lado, o serviço Athena apenas implementa parcialmente o sistema de *cache*, isto é, guarda o código SQL e os resultados das *queries* executadas num *bucket* S3 temporário, organizado por dias. O resultado das *queries* é gravado com um formato CSV porém este é sempre gerado e gravado a cada *query*, mesmo que esta já tenha sido executada anteriormente. Isto é, Athena não verifica as *queries* executadas anteriormente e assim, não tem qualquer alteração no seu desempenho. Caso haja um grande volume de

dados consultados, estes são gravados de forma repetitiva num *bucket* S3 o que implica um custo adicional já que é cobrado um preço por dados armazenados em S3. As únicas utilidades desta funcionalidade é para a sua utilização manual em casos de *debug*, para fins de *log* ou para recuperação de falhas.

Finalmente, o serviço RedShift similarmente aos outros serviços, implementa e aplica automaticamente o sistema de *cache* tendo um grande impacto no tempo de execução de *queries* repetidas. Esta melhoria reduz em média cerca de 57% do tempo de execução. Redshift apresenta assim a melhor solução para um sistema de *cache* entre os três serviços analisados.

Para demonstrar a influência destes sistemas de cache foi executada uma *query* de forma consecutiva em cada um dos serviços sobre os mesmos dados. Após 3 execuções, os resultados são apresentados na tabela 6.5 e representados na figura 6.4.

Tabela 6.5: Tempos de execução consecutiva de uma *query* para a testagem do desempenho do sistema de *cache* de cada serviço.

SQL	Tempo de execução com Spectrum (segundos)	Tempo de execução com RedShift (segundos)	Tempo de execução com Athena (segundos)
Primeira execução	17	7	3,08
Segunda execução	8	2	4,47
Terceira execução	7	2	4,5

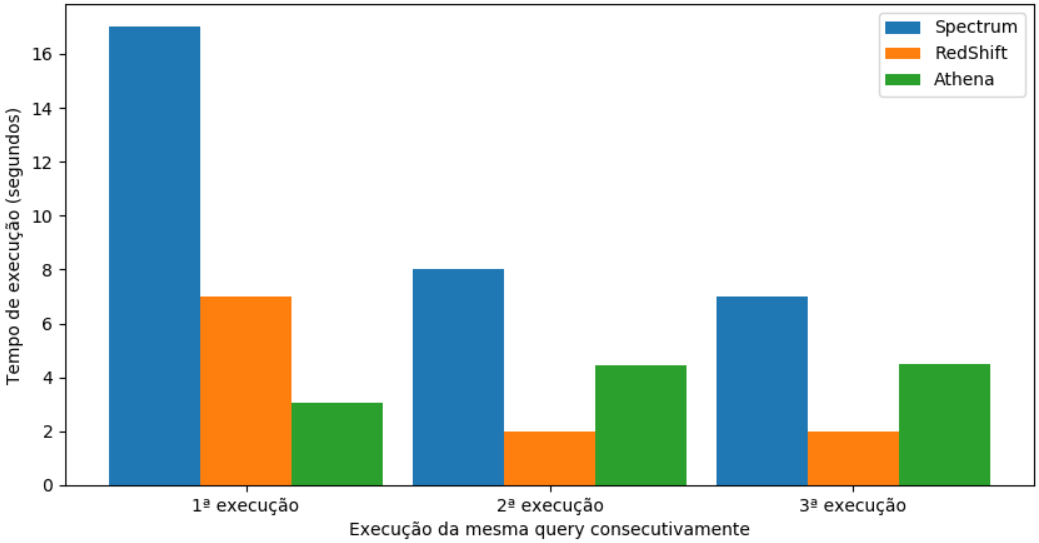

Figura 6.4: Gráfico comparativo de desempenho entre os sistemas de *cache* dos serviços em análise.

Através dos resultados apresentados, é possível concluir que num cenário em que a primeira execução de RedShift é mais lenta que Athena, o mesmo não se verificou nas execuções consecutivas. Estes resultados reforçam o poder do sistema de *cache* de RedShift, porém mesmo com este sistema o uso da ferramenta Spectrum não consegue ultrapassar

o desempenho de Athena.

6.2.3 Ficheiros Parquet

A Amazon produziu um formato de ficheiro compactado especializado em acelerar e simplificar as operações exaustivas feitas por sistemas como os de *Big Data*. Este tipo de ficheiro, designado de AWS Parquet, procura beneficiar o serviço S3 através da sua otimização em termos de utilização de memória e no tempo de execução de *queries*.

Parquet é um formato de ficheiro que utiliza um armazenamento colunar, semelhante a ficheiros ORC. Este formato colunar permite que ao executar uma *query* esta realize a leitura, descompactação e processamento apenas nas colunas necessárias para a execução dessa *query*.

A utilização deste formato de ficheiro tem bastante impacto no desempenho de Athena, constituindo assim mais um trunfo que Athena tem em relação aos outros serviços. Para demonstrar a diferença que este formato tem na gestão de memória e em desempenho, é executada a mesma *query* sobre o mesmo conjunto de dados em diferentes formatos e configurações. Os resultados desta análise estão representados na tabela 6.6 e na figura 6.5.

Tabela 6.6: Tempo de execução e quantidade de dados utilizados para a comparação de entre a utilização de vários formatos de dados.

Formato dos dados	Tempo de execução da *query* (segundos)	Quantidade de dados utilizados pela *query*
Dados brutos (CSV)	22	90 GB
Dados comprimidos	24	13 GB
Dados comprimidos e particionados	20	432 MB
Dados em Parquet	5	2 MB

Com esta análise é possível verificar que com a mesma *query* a utilização do ficheiro Parquet, melhorou o desempenho de Athena. Isto foi obtido através de uma redução na quantidade de dados verificados em 99.997% uma melhoria no tempo de execução da *query* em 77.27% em relação à sua execução com recurso à utilização de dados em ficheiros CSV.

6.3 Propriedades de bases de dados

Em bases de dados há normalmente vários utilizadores a realizar operações sobre os dados. Caso estas operações sejam executadas de forma concorrente, o sistema tem de garantir que não haja erros nem irregularidades nas operações executadas. Para garantir confiabilidade e consistência dos dados considera-se necessário as transações conterem as propriedades ACID.

Considera-se uma transação uma operação lógica ou uma sequência de operações realizadas nos dados. Um exemplo comum é a transferência bancária entre contas, o va-

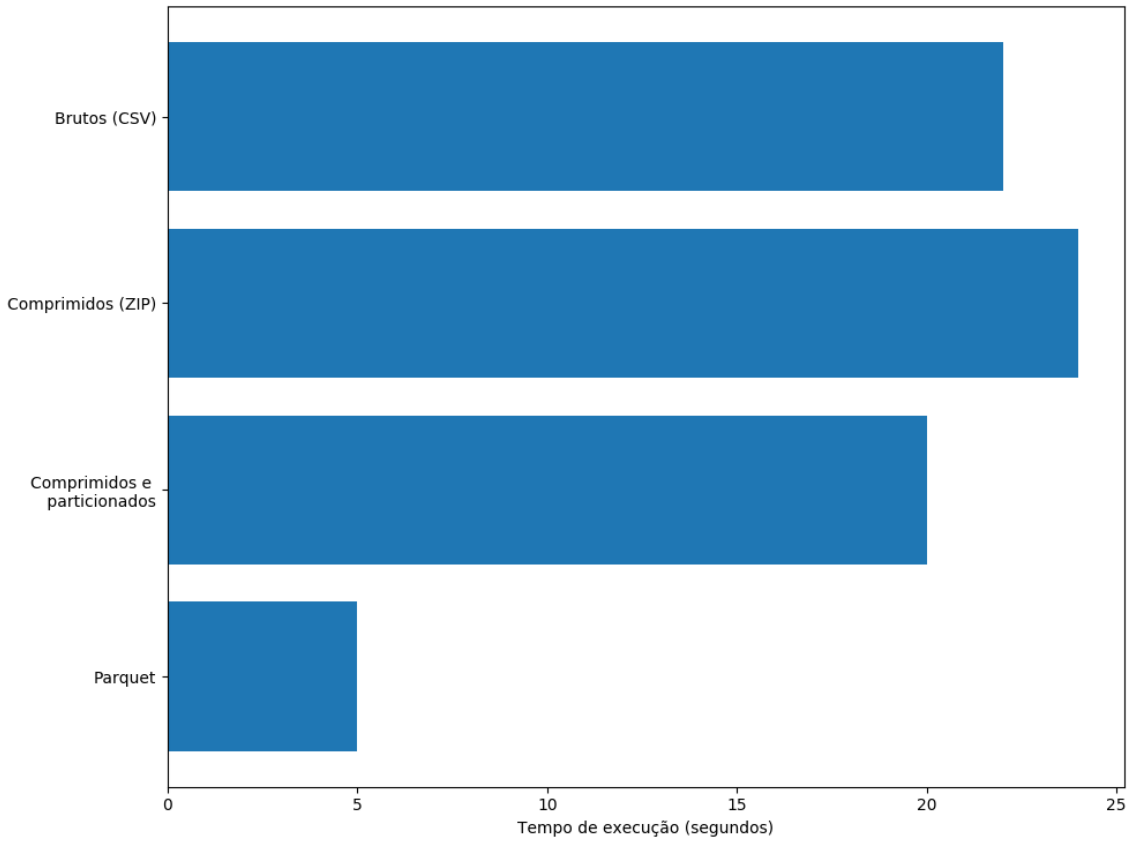

Figura 6.5: Gráfico de comparação de desempenho uma *query* executada em Athena em diferentes formatos de ficheiro.

lor da conta de origem tem de ser decrementado em conjunto com o incremento da conta alvo. Caso esta transação não seja feita corretamente sem consistência nos dados, os utilizadores ou a entidade bancária ficam lesados. Para evitar este tipo de inconsistência as transações têm seguir uma série de propriedades que no seu conjunto são designadas por ACID. As propriedades são nomeadamente as seguintes:

- **Atomicidade**: numa transação que envolve duas ou mais operações, esta é executada na sua totalidade ou não. Assim, se uma operação der erro, as operações precedentes vão revertidas, garantindo assim que as transações sejam atómicas;
- **Consistência**: garante a validade dos dados presentes na base de dados. Cada operação para ser executada tem de ser válida de acordo com as regras ou restrições definidas para os dados. Isto impossibilita que os dados sejam corrompidos por transações consideradas ilegais;
- **Isolamento**: uma transação ao ser iniciada cria um estado em que está isolada de outras transações concorrentes. Assim as alterações nos dados realizadas por outras transações não afetam as operações de outra transação;
- **Durabilidade**: caso uma transação tenha sido executada, as suas alterações nos dados persistem mesmo em caso de falha no sistema. Isto significa que as transações não podem estar registadas em memória volátil.

Para sistemas com um elevado número de utilizadores, a execução de transações concorrentes é inevitável. Assim, as propriedades ACID são cruciais para este tipo de sistemas. Porém, para um sistema de *Big Data* não há este tipo de requisito devido ao facto de haver um baixo número de utilizadores (neste caso programadores). Ainda assim, pode originar em erros nos dados ou à não deteção de falhas em dados provenientes das fontes.

A ausência das propriedades ACID podem originar a erros, como por exemplo, a existência de dados repetidos devido à falta de atomicidade (uma operação ser executada apesar da transação falhar) e à falta de consistência (ausência de verificação de chaves primárias).

As propriedades ACID também podem ter um efeito ligeiramente negativo no desempenho do serviço, já que qualquer operação terá verificações extras para garantir a consistência dos dados. Isto poderá ser mais notório em sistemas com grandes quantidades de dados onde há um grande fluxo de transações, tal como num sistema de *Big Data*.

No caso dos serviços de *Data Warehouse* em análise, os serviços Aurora e RedShift contêm estas propriedades ACID. Por outro lado, Athena não contém estas propriedades, o que permite possíveis erros nos dados, necessitando assim sempre de cuidados especiais e redobrados. Um exemplo de falhas recorrentes com esta falta de propriedades é a facilidade de haver repetição de dados, uma vez que há uma ausência de verificações tal como definir chaves primárias.

6.4 Funcionalidades adicionais

Athena, em termos de funções adicionais, é o serviço mais básico, apenas servindo para executar *queries* estando muito dependente do serviço S3. O serviço Aurora tem algumas funcionalidades adicionais devido a RDS tais como monitorização de desempenho de uma instância, *snapshots* e gestão automática de *backups*.

Por outro lado, RedShift é o serviço mais completo dos 3, devido ao facto de incluir todas as funcionalidades de RDS e adicionalmente conter uma vasta lista de funcionalidades. Esta lista contém funcionalidades como a criação de eventos para a sua integração com outros serviços como AWS Lambda e AWS *Simple Notification Service* (SNS) para a criação de alarmes personalizáveis para parâmetros de uma instância. Para este sistema há duas funcionalidades que se destacam, uma delas é a criação automática e gratuita de gráficos com resultados de *queries* após a sua execução, tal como é exemplificado na figura 6.6.

Por fim, RedShift está diretamente integrado na AWS Marketplace, isto significa que é possível integrar um serviço como Tableu numa instância RedShift através da criação automática de uma instância *Elastic Compute Cloud* (EC2) com Tableu. Isto elimina a necessidade de ter que ligar Tableu como serviço externo, mantendo assim todos os serviços dentro do ecossistema AWS. No caso de Tableu, a sua criação a manutenção numa instância EC2 tem o custo de 1,24€ (euros) por hora.

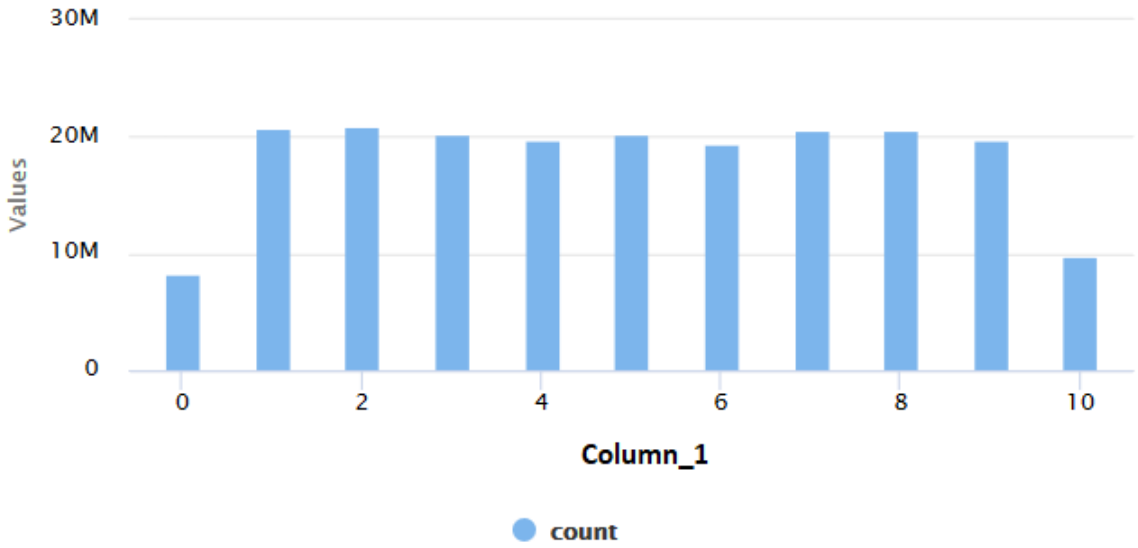

Figura 6.6: Resultado visual produzido automáticamente por RedShift após a execução de uma *query* com uma contagem de agregação da coluna "Column_1".

6.5 Conclusão

Em suma, este capítulo analisa os serviços de Data Warehouse escolhidos no capítulo 2 (AWS Athena, Aurora e RedShift) em contexto do sistema de *Big Data* alvo. Inicialmente, é estudada a integração de cada serviço neste sistema *Big Data*, apesar de todos os serviços disponibilizarem métodos e ferramentas para a sua simples integração, Athena destaca-se por não necessitar de nenhuma configuração já que esta é gerada e mantida automaticamente pelo serviço Glue com recurso a *Crawler's*.

É apresentada uma análise comparativa de desempenho tendo como referência o serviço Athena. Inicialmente é testada a diferença de desempenho entre a execução de *queries* em dados migrados para RedShift ou dados inseridos em S3 usando a ferramenta Spectrum. A execução de *queries* em dados migrados para RedShift têm um tempo de execução mais baixo, revelando um desempenho superior, porém em comparação com o serviço Athena, a utilização de RedShift revelou ter um desempenho inferior.

Ao testar o desempenho de Aurora, apesar de ser notória a escalabilidade de desempenho consoante a melhoria da configuração, é revelado que este serviço tem um desempenho muito baixo em relação a Athena. O que indicia que este serviço não é adequado à execução de *queries* com elevadas quantidades de dados.

Existem fatores adicionais nos serviços que podem influenciar o desempenho, sendo necessário fazer uma análise destas alterações. Os fatores considerados nesta análise são o uso de um sistema de *caching* e o uso de diferentes formatos de ficheiros em S3 no caso de Athena. No caso do uso de um sistema de cache, o único serviço a ter uma diferença impactante no desempenho de execução de *queries* consecutivas é RedShift, não havendo melhorias de desempenho nos outros serviços. Outro fator considerado é a utilização de dados em ficheiros no formato Parquet e o seu efeito no desempenho de *queries* executadas em Athena. É notável uma melhoria significativa no desempenho de Athena ao executar *queries* sobre dados armazenados em Parquet em relação a outros formatos

e configurações de dados.

Para os serviços observados foi realizada uma análise das propriedades de base de dados e funcionalidades adicionais. O serviço RedShift destacou-se nesta etapa visto que contém as propriedades ACID que garantem a consistência dos dados, tal como a inclusão de uma vasta lista de funcionalidades adicionais.

Finalmente, a escolha de qual o serviço de *Data Warehouse* mais adequado ao sistema *Big Data*, é um disputa principalmente entre RedShift e Athena. Isto porque, em todos os pontos considerados (integração, desempenho, propriedades de base de dados e funcionalidades adicionais) não houve um destaque particular do serviço Aurora. Athena é um serviço rápido e fácil de integrar, porém a falta de propriedades e funcionalidades adicionais fazem de RedShift uma escolha a considerar. O serviço RedShift para igualar o desempenho de Athena neste sistema, necessita de uma instância superior, o que obriga a um custo mais elevado. Porém, RedShift ao oferecer consistência nos dados e uma panóplia de funcionalidades que fazem deste serviço uma excelente escolha. Concluindo, a opção final de qual o serviço a integrar o sistema *Big Data* deve ser feita entre um serviço mais caro e completo, ou por outro lado, selecionar o serviço mais simples e mais barato.

Capítulo 7

Conclusão

Este documento trata um projeto de estágio inserido na CTW para integração de uma equipa de desenvolvimento a trabalhar num sistema *Big Data* para apoiar as decisões de negócio da BMW no Reino Unido. O objetivo principal deste projeto é aferir qual o serviço de *Data Warehouse* que mais se adequava a este sistema de *Big Data* em desenvolvimento. Este capítulo procura apresentar as conclusões principais resultantes da análise e comparação de serviços de *Data Warehouse* no contexto do sistema *Big Data* CTW. No final há um breve levantamento de funcionalidades desenvolvidas no futuro.

7.1 Conclusões Principais

Atualmente há um acréscimo exponencial de dados gerados nos sistemas de informação de empresas devido ao constante crescimento de clientes e consequente aumento no volume de negócios. O tratamento e análise destes dados é imperativo para estas empresas, pois há a necessidade de conseguir analisar o mercado tal como as tendências atuais. Com esta análise é possível tomar decisões adaptativas ao mercado garantido a subsistência e liderança no mercado das empresas.

Devido à variedade e volume de negócios BMW na região onde o sistema *Big Data* se insere, o sistema *Big Data* atualmente em construção deve ingerir e processar grandes quantidades de dados. Para isso, são selecionados serviços e tecnologias estado da arte para integrar este sistema garantindo o melhor desempenho possível. Nesse sentido, há uma necessidade de uma análise e auditoria a serviços de *Data Warehouse* no mercado para perceber e aferir qual ou quais se adequam a este sistema para a sua futura integração.

São analisados cinco serviços de *Data Warehouse* no capítulo 2, porém através do estudo e comparação das características chave destes serviços apenas são escolhidos três para uma análise mais aprofundada no contexto do sistema *Big Data*. Os serviços escolhidos são: AWS Athena, AWS Aurora e AWS RedShift. A análise destes serviço no capítulo 6 conclui que AWS Aurora apresenta défices de desempenho que impossibilitam a sua utilização no sistema. AWS RedShift apresenta uma integração simples, um desempenho favorável, um custo elevado, garante propriedades ACID e uma vasta lista de funcionalidades. Por outro lado, AWS Athena apresenta uma integração simples, um desempenho favorável e um baixo custo, porém é um serviço básico que não contém propriedades ACID nem funcionalidades adicionais. Em suma, a escolha de qual o serviço de *Data Warehouse* a integrar passa por uma decisão onde se analisa se as vantagens de AWS RedShift compensam o investimento adicional.

O objetivo principal deste projeto de estágio foi cumprido, permitindo à CTW realizar a escolha do serviço de *Data Warehouse* tendo em conta as análises e experiências demonstradas neste documento. Assim, com a escolha do serviço, a CTW pode continuar a desenvolver sistemas e produtos estado de arte para a histórica e conceituada marca BMW.

7.2 Trabalho Futuro

O sistema de *Big Data* no qual este projeto de estágio se insere está em constante evolução e atualização graças aos recorrentes requisitos dos clientes e devido à complexidade e importância deste sistema. Enquanto trabalho futuro, este passa por atualizar constantemente o sistema à medida que novos requisitos são apresentados, bem como melhorar o sistema para corresponder ao constante aumento do volume e complexidade de dados ingeridos pelo sistema. Adicionalmente, há certas funcionalidades que podem ser incorporadas no futuro, tais como: a ingestão de dados em tempo real através de *streams* Kinesis e funções Lambda; e a análise e integração de serviços ou ferramentas de *Business intelligence* (BI) como Power BI, Tableu e AWS QuickSight.

Bibliografia

[Ama12a] Amazon. Aws redshift [online]. 2012. Available from: https://aws.amazon.com/pt/redshift/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc [cited 03 fevereiro 2021]. 12

[Ama12b] Amazon. Aws redshift price definition [online]. 2012. Available from: https://aws.amazon.com/pt/redshift/pricing/ [cited 03 fevereiro 2021]. 13

[Ama14a] Amazon. Aws aurora [online]. 2014. Available from: https://aws.amazon.com/pt/rds/aurora/?nc2=type_a&aurora-whats-new.sort-by=item.additionalFields.postDateTime&aurora-whats-new.sort-order=desc [cited 03 fevereiro 2021]. 10

[Ama14b] Amazon. Aws aurora price definition [online]. 2014. Available from: https://aws.amazon.com/pt/rds/aurora/pricing/ [cited 03 fevereiro 2021]. 11

[Ama16a] Amazon. Aws athena [online]. 2016. Available from: https://aws.amazon.com/pt/athena/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc [cited 03 fevereiro 2021]. 9

[Ama16b] Amazon. Aws athena price definition [online]. 2016. Available from: https://aws.amazon.com/pt/athena/pricing/?nc=sn&loc=3 [cited 03 fevereiro 2021]. 10

[Ama21a] Amazon. Aws glue concepts [online]. 2021. Available from: https://docs.aws.amazon.com/glue/latest/dg/components-key-concepts.html [cited 28 Maio 2021]. xi, 32

[Ama21b] Amazon. Overview of workflows in aws glue [online]. 2021. Available from: https://docs.aws.amazon.com/glue/latest/dg/workflows_overview.html [cited 28 Maio 2021]. xi, 34

[Atl20] Atlassian. What is agile? [online]. 2020. Available from: https://www.atlassian.com/agile [cited 03 fevereiro 2021]. 21

[AWS20] AWS. Bmw group usa data lake baseado na aws para liberar todo o potencial dos dados [online]. 2020. Available from: https://aws.amazon.com/pt/solutions/case-studies/bmw-group-case-study/ [cited 13 Julho 2021]. 19

[AWS21] AWS. A quick look at codepipeline [online]. 2021. Available from: https://docs.aws.amazon.com/codepipeline/latest/userguide/welcome-introducing.html [cited 24 Junho 2021]. xi, 43

[Ber15] Sérgio Fernandes , Jorge Bernardino. What is bigquery? In *Proceedings of the 19th International Database Engineering Applications Symposium*, page 202–203. ACM Digital Library, 2015. 14

[BMW21a] BMW. The all-new bmw idrive [online]. 2021. Available from: https://www.press.bmwgroup.com/global/article/detail/T0327315EN/the-all-new-bmw-idrive?language=en [cited 24 Junho 2021]. xi, 3

[BMW21b] BMW Group. Enquire now [online]. 2021. Available from: https://www.bmw.co.uk/en_GB/topics/discover/forms/pdi_bmw_i2818_rfc.html [cited 26 fevereiro 2021]. xi, 46

[Cig19] J. Cigánek. Design and implementation of open-data data warehouse. In *6th International Conference on Advanced Control Circuits and Systems (ACCS) & 5th International Conference on New Paradigms in Electronics information Technology (PEIT)*, pages 185–190, 2019. 3

[Cri21] Critical TechWorks. Rethink the future [online]. 2021. Available from: https://www.criticaltechworks.com/ [cited 03 fevereiro 2021]. 2

[DOM21] DOMO. Data never sleeps 8.0 [online]. 2021. Available from: https://www.domo.com/learn/data-never-sleeps-8 [cited 31 março 2021]. 8

[Gar20] Gartner. Gartner magic quadrant for cloud infrastructure and platform services [online]. 2020. Available from: https://www.gartner.com/en/documents/3989743/magic-quadrant-for-cloud-infrastructure-and-platform-ser [cited 05 fevereiro 2021]. xi, 18

[Goo11a] Google. Google bigquery key features [online]. 2011. Available from: https://cloud.google.com/bigquery [cited 03 fevereiro 2021]. 14

[Goo11b] Google. Google bigquery pricing [online]. 2011. Available from: https://cloud.google.com/bigquery#section-11 [cited 03 fevereiro 2021]. 15

[JRRR16] Rabiul Jony, Rakibul Islam Rony, Musfiqur Rahman, and Abiduzzaman Rahat. Big data characteristics, value chain and challenges. 05 2016. xi, 7

[Mic11a] Microsoft. Azure synapse analytics [online]. 2011. Available from: https://azure.microsoft.com/pt-pt/services/synapse-analytics/ [cited 03 fevereiro 2021]. 16

[Mic11b] Microsoft. Price of azure synapse [online]. 2011. Available from: https://azure.microsoft.com/pt-pt/pricing/details/synapse-analytics/ [cited 03 fevereiro 2021]. 17

[Sar17] Apoorva Srivastava, Sukriti Bhardwaj, Shipra Saraswat. Scrum model for agile methodology. *International Conference on Computing, Communication and Automation (ICCCA)*, 0:864–869, 2017. 21

[SAS21] SAS Institute. Big data what it is and why it matters [online]. 2021. Available from: https://www.sas.com/pt_pt/insights/big-data/what-is-big-data.html [cited 03 fevereiro 2021]. 3