Universidade do Minho
Escola de Engenharia

Gonçalo José da Silva Fontes

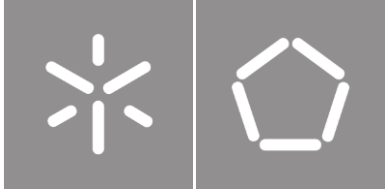
Desenvolvimento de uma plataforma de *Big Data Analytics* para desenvolvimento do produto em contexto industrial

Novembro 2022

Desenvolvimento de uma Plataforma de Big Data Analytics para desenvolvimento do produto em contexo industrial

Gonçalo Silva Fontes

UMinho | 2022



Universidade do Minho
Escola de Engenharia

Gonçalo José da Silva Fontes

Desenvolvimento de uma plataforma de *Big Data Analytics* para desenvolvimento do produto em contexto industrial

Dissertação de Mestrado

Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Trabalho efetuado sob a orientação de:
Professor Doutor Paulo Alexandre Ribeiro Cortez

Novembro 2022

AGRADECIMENTOS

Como qualquer etapa que já vivenciei e pretendo ainda vivenciar, esta não foi realizada sozinho. Por esse motivo aproveito para agradecer às pessoas que de alguma forma marcaram ou motivaram a que esta etapa fosse realizada.

Ao professor Doutor Paulo Cortez, agradeço por toda a disponibilidade, pela partilha de conhecimento e por todos os ensinamentos que tornaram possível o desenvolvimento das minhas competências e deste projeto.

Ao André Pilastri e a todos os colegas da equipa *CCG/EPMQ* agradeço por toda a passagem de conhecimento, o companheirismo e por toda a entre-ajuda, que sempre esteve presente no decorrer de toda a minha passagem pelo *CCG*.

A todos os parceiros do *INEGI* que foram incansáveis no decorrer do projeto e que sempre estiveram prontos a ajudar no desenvolvimento do mesmo, foram muito importantes para que este projeto avançasse e para que todas as partes interessadas fossem envolvidas.

À minha família, agradeço não só pelo apoio nesta etapa, mas sim por todo o amor e carinho de sempre. A vida foi feita para partilharmos as conquistas com quem amamos e esta eu partilho-a especialmente convosco.

À Natacha, obrigado pelo apoio incansável e por celebrares sempre o dobro de mim as minhas conquistas e por partilhares comigo as tuas.

Por fim agradeço a todas as pessoas que de alguma forma fizeram parte da minha jornada na Universidade do Minho e a todos aqueles que fizeram parte deste percurso.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.
Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Resumo

Os atuais ambientes de produção são pressionados para obter uma variedade crescente de produtos personalizados e de alta qualidade em lotes muito flexíveis. A elevada dinâmica dos mercados, o ciclo de vida dos produtos cada vez mais curto e a necessidade de flexibilidade marcam a transição dos sistemas de produção tradicionais para uma nova geração de sistemas de produção onde a flexibilidade, usabilidade, capacidade de autorregulação e reconfiguração são fundamentais.

Este trabalho de dissertação assume este contexto e está inserido no projeto de I&D *"PRODUTECH4S&C: PRODUTECH SUSTENTÁVEL & CIRCULAR"*. Em particular, foram abordados dois objetivos: a conceção e implementação de uma plataforma de *Big Data Analytics* capaz de armazenar e processar dados associados a sistemas produtivos; e estudo e aplicação de algoritmos de *Machine Learning* para deteção de anomalias, de forma a ser possível detetar falhas na produção.

Palavras-Chave: *Big Data*; Ecossistema *Hadoop*; Indústria 4.0; *Internet of Things*; *Machine Learning*.

Abstract

Today's production environments are under pressure to obtain a growing variety of high quality, customized products in very flexible batches. The high dynamics of the markets, the increasingly shorter product life cycle and the need for flexibility mark the transition from traditional production systems to a new generation of production systems where flexibility, usability, capacity for self-regulation and reconfiguration are fundamental.

This dissertation work assumes this context and is part of the R&D project "PRODUTECH4S&C: PRODUTECH SUSTAINABLE & CIRCULAR". In particular, two objectives were examined: the design and implementation of a Big Data Analytics platform capable of storing and processing data associated with production systems; and study and application of Machine Learning algorithms for detection of anomalies, in order to be able to detect failures in production environments.

Keywords: *Big Data; Hadoop Framework; Industry 4.0; Internet of Things; Machine Learning.*

Índice

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas, Siglas e Acrónimos

AE	*AutoEncoder*.
API	*Application Programming Interface*.
AUC	*Area Under Curve*.
AutoML	*Automated Machine Learning*.
BN	*Batch Normalization*.
CCG	*Centro de Computação Gráfica*.
CPU	*Central Process Unit*.
CRISP-DM	*Cross Industry Standard Process for Data Mining*.
DFFN	*Deep FeedForward Network*.
DSRM	*Design Science Research Methodology*.
ETL	*Extract, Transform and Load*.
FBP	*Flow Based Programming*.
FIFO	*First-In First-Out*.
FPR	*False Positive Rate*.
GBM	*Gradient Boost Machine*.
GC	*Gaussian Copula*.
GFS	*Google File System*.
GLM	*Generalized Linear Model*.
HDFS	*Hadoop Distributed File System*.
HQL	*Hive Query Language*.
IF	*Isolation Forest*.
IIoT	*Industrial Internet of Things*.
IoT	*Internet of Things*.
JVM	*Java Virtual Machine*.

LOF *Local Outlier Factor*.

MAE *Mean Absolut Error*.
ML *Machine Learning*.

NSA *National Security Agent*.

OC-SVM *One-Class Support Vector Machines*.

PMMGC *Predictive Maintenance Modeling Guide Collection*.

RFID *Radio Frequency Identification*.
ReLu *Rectified Linear Units*.
RF *Random Forest*.
ROC *Receiver Operating Characteristic*.

SMOTE *Synthetic Minority Oversampling Technique*.
SQL *Structured Query Language*.

TCP *Transmission Control Protocol*.
TPR *True Positive Rate*.

1

Introdução

1.1 Enquadramento e Motivação

A presente dissertação surge no âmbito do projeto *"PRODUTECH4S&C: PRODUTECH SUSTENTÁVEL & CIRCULAR"*, o qual foi desenvolvido no *CCG*. Este projeto insere-se no âmbito da crescente complexidade em criar valor no desenvolvimento de novas metodologias, estratégias e abordagens de forma a originar novos produtos e serviços que vão de encontro com uma economia circular, e garantindo a sustentabilidade dos vários setores industriais.

Particularmente, o *PPS2- Aceleração e informação no desenvolvimento de produto sustentável*, consiste no estudo de ferramentas e metodologias de desenvolvimento de produto que respondam às atuais necessidades de desempenho superior em termos de: flexibilidade, usabilidade, capacidade de autorregulação e disponibilização de informação. Reinventar e evoluir as metodologias e tecnologias de desenvolvimento de produto para poder fazer face a um ambiente mais exigente, de maior especialização e com uma forte componente de informação, dados, digitalização e inteligência, incorporando conceitos de criação de valor em ambientes colaborativos internos na empresa e na cocriação com parceiros externos.

Ainda dentro do *PPS2*, podem destacar-se várias atividades, sendo que a presente dissertação de mestrado realizou-se no âmbito da atividade *"Data-driven Product Development and Innovation"*, tendo como objetivo dar resposta às dificuldades atuais no que diz respeito à capacidade para lidar e tornar úteis no desenvolvimento do produto a elevada quantidade de dados (*Big Data*) gerados pelos próprios produtos e pelos sistemas de produção, acrescida da capacidade de processamento e tratamento local desses dados.

Embora se verifique a impossibilidade de trabalhar com um volume de dados suficiente para ser possível chegar à finalidade deste projeto, uma vez que por parte dos parceiros industriais não existiu uma sensorização do produto antecipada para realizar a etapa de aquisição de dados, o trabalho realizado enquadrar-se-á com o contexto do projeto de investigação, com a finalidade de cumprir os objetivos do mesmo, através da implementação de uma plataforma de *Big Data Analytics* para desenvolvimento do produto em fase de uso, assim como a aplicação de algoritmos de *Machine Learning* para deteção de anomalias.

1.2 Objetivos e Resultados Esperados

Se a existência de um problema é o motivo para a criação de um projeto, os objetivos neles definidos traçam o caminho que deve ser seguido para ser possível atingir a solução com poder para resolver o problema. A integração de metodologias e ferramentas desenvolvidas para o contexto do projeto é inovadora, podendo-se referir que no caso específico do setor dos bens de equipamento a sua integração é inexistente. Sendo assim, esta dissertação reúne um conjunto de objetivos associados ao projeto *PRODUTECH4S&C* que garantem o cumprimento das necessidades identificadas. Em particular, foram definidos dois objetivos:

1. **Estudo e implementação de uma plataforma de *Big Data Analytics*:** Estudo de conceitos e tecnologias existentes, garantindo a compreensão e identificação daquelas que podem ser utilizadas no contexto do problema. Sendo assim, o objetivo será o desenvolvimento de uma arquitetura/plataforma capaz de tornar útil a elevada quantidade de dados gerada pelos sistemas produtivos na fase de uso com o objetivo de se poder introduzir melhorias, atualização ou desenvolvimento de novos produtos
2. **Estudo e utilização de técnicas de *Machine Learning*:** Estudo e aplicação de algoritmos de *Machine Learning* para deteção de anomalias, de forma a ser possível detetar eventuais falhas nos produtos existentes nos sistemas produtivos.

1.3 Abordagem Metodológica

Da procura incessante de cumprir com os objetivos traçados, surge a necessidade de utilizar uma metodologia que permita orientar o trabalho a desenvolver. Como esta dissertação pode ser dividida em dois casos de estudo distintos, optou-se pela utilização de duas metodologias, a metodologia *Design Science Research Methodology (DSRM) for Information Systems* e a metodologia *CRISP-DM* (*Cross Industry Process for Data Mining*).

Relativamente à principal razão para a adoção da metodologia *DSRM* para o primeiro caso de estudo (Estudo e implementação de uma plataforma de *Big Data Analytics*) deve-se ao facto de se tratar de um trabalho que envolve um processo de criação de conhecimento, resultante do desenvolvimento ou inovação de artefactos e das análises da utilização dos mesmos em certos cenários (Hevner and Chatterjee, 2010). Relativamente à utilização da metodologia *CRISP-DM* para o segundo caso de estudo (Estudo e utilização de técnicas de *Machine Learning*, deve-se ao facto de esta permitir a diminuição da complexidade de projetos de *Data Mining*, tornando-os, por um lado, menos dispendiosos e repetíveis, mas, por outro, mais confiáveis, mais fáceis de gerir e rápidos (Wirth and Hipp, 2000).

1.3.1 *Design Science Research Methodology (DSRM) for Information Systems*

Existem várias propostas relativamente a este tipo de metodologia, sendo que no âmbito deste dissertação, a proposta de (Peffers et al., 2007) é a que irá ser utilizada no decorrer da mesma, sendo representada na figura 1.

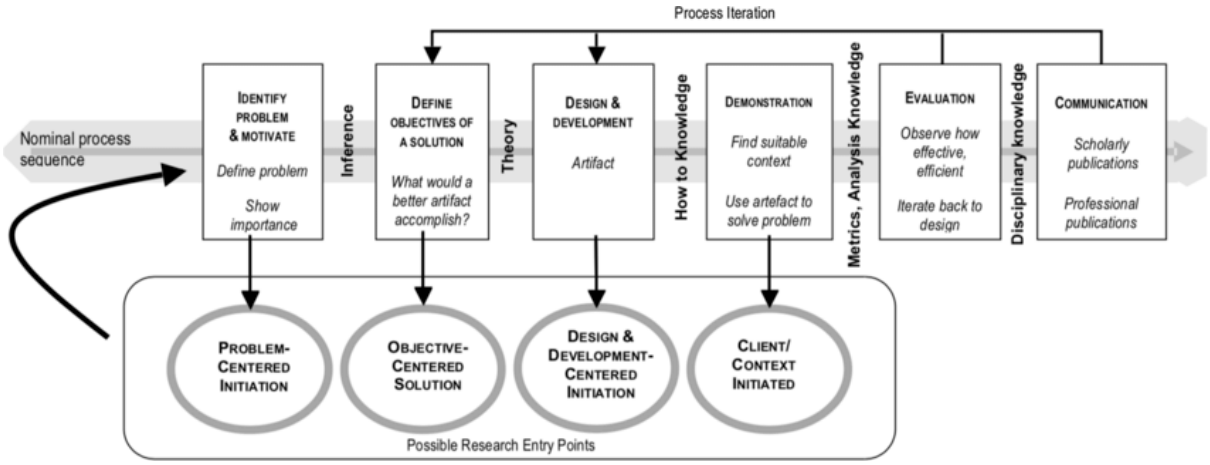


Figura 1: DSRM for Information Systems (retirado de Peffers et al., 2007).

O inicio do trabalho desta dissertação é marcada pela leitura de documentos científicos, para que seja possível a determinação do âmbito do trabalho, assim como o entendimento dos conceitos envolvidos. Inicia-se então, a elaboração do plano de trabalho, na qual é realizado o planeamento do que se pretende vir a desenvolver, assim como identificada a motivação, a importância e a necessidade de investigação no âmbito inserido, os objetivos, a abordagem metodológica e as tarefas que vão fazer parte do trabalho a ser desenvolvido. Esta metodologia é flexível, devido ao facto de permitir que o projeto de investigação possa ser iniciado em vários pontos de partida. No caso do trabalho desenvolvido nesta dissertação, iniciou-se a investigação centrada no problema, percorrendo as 6 fases que estão representadas na figura 1, nomeadamente (Peffers et al., 2007) (Geerts, 2011):

1. **Identificação do problema e motivação:** Identificação de aspetos que visam marcar a diferença, criando oportunidade na definição do problema e que fundamentalmente levarão ao desenvolvimento da solução.

2. **Definição de objetivos:** Definição dos objetivos que irão possibilitar a chegada a uma solução e a necessidade de obter conhecimento para a solução ser colocada em prática. Após este passo de definição de objetivos é necessário a realização da revisão do estado de arte, sendo uma etapa bastante por se realizar um enquadramento a nível de conceitos e trabalhos existentes, assim como, um enquadramento a nível tecnológico.

3. **Conceção e desenvolvimento:** Esta etapa visa representar a evolução e progresso dos vários métodos para a solução do problema, sendo que começa a ser desenvolvida uma proposta de arquitetura, estando sempre suscetível a alterações consoante aquilo que é desejado para o objetivo final.

4. **Demonstração:** Esta etapa consiste na validação da solução, nomeadamente na implementação da proposta de arquitetura mencionada no ponto anterior. Permite também ser provado se a implementação realizada é capaz de resolver o problema.

5. **Avaliação:** É a etapa onde são realizadas as conclusões do âmbito desta dissertação, podendo ser realizada uma comparação com diferentes trabalhos desenvolvidos. É observado e avaliado em que medida é que a solução responde ao problema, e eventualmente terem que ser feitos ajustes na solução.

6. **Comunicação:** A comunicação, como o próprio nome indica, tem como propósito a apresentação e divulgação dos resultados ao longo do desenvolvimento da dissertação, envolvendo a escrita da dissertação e também a divulgação do trabalho desenvolvido na comunidade científica.

1.3.2 *Cross Industry Standard Process for Data Mining*

Será também utilizada a metodologia *CRISP-DM (Cross- Industry Standard Process for Data Mining)* (ver fig. 2), servindo de suporte a projetos de *Data Mining* permitindo definir o plano de ação. Esta metodologia foi desenvolvida com o objetivo de servir como um guião para o desenvolvimento de projetos de *Data Mining*.

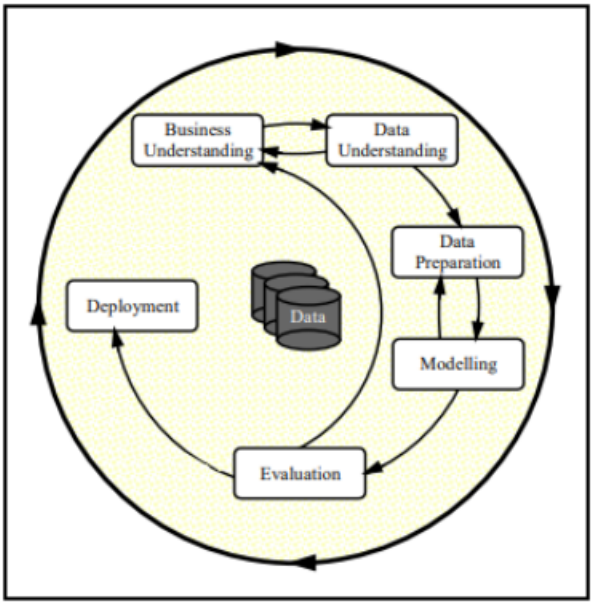


Figura 2: Modelo CRISP-DM (adaptado de Wirth and Hipp, 2000).

1. ***Business Understanding* (Compreensão do Negócio):** Caracteriza-se pela identificação dos objetivos e requisitos do projeto, de forma a compreender a perspetiva desta dissertação, sendo que após a definição dos requisitos, deverá ser estabelecido um plano de ação para alcançar os objetivos.

2. ***Data Understanding* (Compreensão dos Dados):** Etapa onde é feita uma recolha inicial dos dados, posterior análise dos mesmos, para se poder compreender as diversas relações entre as variáveis, e identificação dos problemas relacionados à qualidade.

3. ***Data Preparation* (Preparação dos Dados):** Esta etapa é caracterizada pelo tratamento e processamento dos dados, sendo as técnicas necessárias para o tratamento, de forma a ser elaborado o *dataset* final para a modelação. Sendo o *CRISP-DM* uma metodologia cíclica, caso sejam detetadas falhas que necessitem de serem tratadas novamente, esta tarefa deverá ser novamente executada.

4. ***Modeling* (Modelação):** Fase de seleção de técnicas e modelos de *Machine Learning* e respetivo ajuste dos hiper parâmetros de modo a atingir os melhores resultados possíveis. Também por isso se verifica as transições das fases de *evaluation* e *deployment*, pois são fases que permitem perceber se o modelo obtido na fase de modelação é, de facto, o mais adequado.

5. ***Evaluation* (Avaliação):** Esta etapa consiste na avaliação dos modelos aplicados na etapa anterior, de forma a selecionar um, ou vários modelos, cuja qualidade cumpra com os requisitos estipulados.

6. ***Deployment* (Implementação):** A fase da implementação pode ter diversos objetivos conforme os requisitos propostos para cada projeto, e não significa necessariamente o fim do projeto.

1.4 Contributos do Trabalho

O trabalho desenvolvido nesta dissertação permitiu a publicação do seguinte artigo cientifico:

1. Fontes, G., Matos, L.M., Matta, A., Pilastri, A., Cortez, P. (2022). An Empirical Study on Anomaly Detection Algorithms for Extremely Imbalanced Datasets. In: Maglogiannis, I., Iliadis, L., Macintyre, J., Cortez, P. (eds) Artificial Intelligence Applications and Innovations. AIAI 2022. IFIP Advances in Information and Communication Technology, vol 646. Springer, Cham. https://doi.org/10.1007/978-3-031-08333-4_7.

Este artigo cientifico pretendeu a avaliação referente à implementação de diferentes algoritmos de deteção de anomalias para conjuntos de dados extremamente desequilibrados estando enquadrado no projeto de investigação "*Produtech4S&C: Produtech Sustentável e Circular*".

1.5 Organização do Documento

O documento apresentado encontra-se dividido em cinco capítulos, entre os quais são abordados diferentes tópicos, de forma a serem apresentados todos os conteúdos relevantes para a dissertação.

Primeiramente, o Capítulo 1 consiste na introdução ao tema da dissertação, iniciando-se com um enquadramento e motivação para o projeto, seguida de uma apresentação dos objetivos e resultados esperados, e por fim a descrição das metodologias utilizadas para o desenvolvimento da investigação e das tarefas.

O Capítulo 2, Revisão de Literatura, é o capítulo onde são descritos os conceitos relevantes para a dissertação, de modo a alcançar uma visão abrangente relativamente ao tema. O capítulo inicia-se com uma introdução à Indústria 4.0, aos conceitos de *Internet of Things*, *Machine Learning*, *Big Data*, e por fim às tecnologias pertencentes ao *Ecossistema Hadoop*.

O capítulo 3, apresenta e explora o caso de estudo onde é definida e implementada a plataforma de *Big Data* e *Machine Learning*, assim como as tecnologias utilizadas com este propósito, tendo sido utilizados dados provenientes de um cliente para ser possível realizar uma prova de conceito.

O Capítulo 4, apresenta e explora o caso de estudo de algoritmos de deteção de anomalias para conjuntos de dados extremamente desequilibrados, tendo como base a estrutura da metodologia *CRISP-DM*, permitindo uma demonstração mais completa e detalhada do trabalho desenvolvido. Aqui são apresentados os passos realizados assim como os resultados obtidos.

Por fim, o Capítulo 5 reúne as considerações finais relativas ao trabalho desenvolvido, aos contributos fornecidos pelo mesmo e à investigação e objetivos futuros.

2

Revisão da Literatura

A revisão de literatura é constituída por uma referência ao estado, sobre um determinado tópico, no momento em que a investigação científica é iniciada, sendo assim, é estabelecido um ponto de partida para o estudo. Deste modo, a revisão da literatura permite ao autor contextualizar-se relativamente aos conceitos e desenvolvimentos do tema a ser desenvolvido.

2.1 Estratégia de Pesquisa Bibliográfica

Com o objetivo de elaborar o presente documento, antes de dar início à pesquisa e seleção de fontes bibliográficas de suporte ao projeto de dissertação, revelou-se a necessidade de delinear uma estratégia que fosse consistente para a obtenção de material relevante, cujas fontes incluem artigos científicos, dissertações, artigos de conferências, livros, entre outros.

Para cumprir com a necessidade de escolher fontes de informações credíveis foram utilizados vários motores de busca de literatura, como o *Google Scholar*, *Scopus*, *Web of Science* e *Science Direct*. Para os artigos selecionados, foram considerados fatores como o número de citações da fonte, o ano de publicação, entre outros.

Assim sendo, para a realização da pesquisa foram utilizados conceitos fundamentais para a elaboração do documento como *"Industry 4.0"*, *"Machine Learning"*, *"Big Data"*, *Internet of Things"*, *"Hadoop"*, *"Machine Learning Ops"*. À medida que os artigos foram analisados foram surgindo novos conceitos, podendo os mesmos ter sido alterados para melhorar a pesquisa.

Com todos os artigos selecionados, foi então realizado um estudo da literatura selecionada, de modo a serem utilizados os artigos mais relevantes para apoiar o presente documento.

2.2 Indústria 4.0

A indústria é a parte da economia que produz bens materiais que são altamente mecanizados e automatizados. Desde o início da industrialização, saltos tecnológicos, levaram a mudanças de paradigma denominadas "revoluções industriais": no campo da mecânica (1° revolução industrial), do intensivo uso de energia elétrica (2° revolução industrial) e da digitalização generalizada (3° revolução industrial). Na base de uma digitalização avançada dentro das fábricas, a combinação de tecnologias parece resultar numa nova mudança de paradigma fundamental na produção industrial. A visão da produção futura

contém sistemas de manufatura modulares e eficientes e caracteriza cenários nos quais os produtos controlam o seu próprio processo de manufatura. Tentado por essa expectativa, o termo "Indústria 4.0" foi estabelecido tendo como foco uma "4º revolução industrial".

O fenómeno da Indústria 4.0 foi mencionado pela primeira vez em 2011 na Alemanha como uma proposta para o desenvolvimento de um novo conceito de política económica alemã baseado em estratégias de alta tecnologia. A indústria 4.0 envolve especificamente uma mudança radical na forma como os *shop floors* operam. Definido por muitos como uma transformação global da indústria de manufatura com a introdução da digitalização e da *Internet*, estas transformações envolvem melhorias revolucionárias nos processos de manufatura, operações e serviços de manufatura de produtos e sistemas. Partilham muitas semelhanças com desenvolvimentos como *Smart Factories*, *Smart Industry*, *Advanced Manufacturing* e *Industrial Internet of Things* (*IIoT*) (Tjahjono et al., 2017).

No contexto da Indústria 4.0, há ainda a possibilidade de conectar máquinas com humanos através da utilização de Sistemas Ciber-Físicos. Este tipo de sistemas monitoriza processos físicos, criando uma cópia virtual do mundo físico, que toma decisões descentralizadas (Lu, 2017).

Os principais atributos da Indústria 4.0 são a digitalização, otimização e a customização da produção; automatização e adaptação; interação homem-máquina; serviços e negócios de valor acrescentado; comunicação e troca de dados automática. A Indústria 4.0 fornece mudanças disruptivas nas cadeias de abastecimento, modelos de negócio e processos de negócio. A Indústria 4.0 pode ser resumida como um processo de manufatura integrado, adaptado, otimizado, orientado a serviços e interoperável, que está correlacionado com algoritmos, *Big Data* e tecnologias de ponta. Sendo assim, os princípios da Indústria 4.0 são a interoperabilidade, virtualização, descentralização, capacidade em tempo real, orientação para serviço e modularidade (Shafiq et al., 2015).

2.3 *Internet of Things*

A *Internet of Things* é um fator fundamental na Indústria 4.0, tendo um papel importante no âmbito do desenvolvimento sustentável, devido ao potencial de sensorização de informação, monitorização das emissões e gastos de energia industriais, assim como gerar dados que permitam obter informações do ciclo de vida do produto, Desta forma é possível obter dados que permitam melhorar aspetos do fim do ciclo de vida do produto como, melhorar a eficiência da reciclagem e da reutilização dos mesmos (Ness et al., 2015). Os dados gerados são possíveis de obter em tempo real, através da implementação de tecnologias de *IoT*, como sensores, *software* e outras tecnologias que permitam a interligação de máquinas, de forma que estas possam trocar dados obtidos entre si.

Outro aspeto que tem tornado o conceito de *IoT* relevante é a introdução de métodos e tecnologias de *Big Data Analytics*, capazes de guardar, processar e visualizar o grande volume de dados que é gerado através destes dispositivos. Estas funcionalidades de análise de dados irão permitir às organizações adquirir conhecimento em relação às necessidades dos clientes, possíveis ajustes nos processos de produção, assim como obter *insights* que permitam transitar para uma produção sustentável (Yang et al., 2016). Com o objetivo das *Smart Manufactory* conectarem todos os seus dispositivos, de forma a melho-

rarem o processo de tomada de decisão, a adoção de dispositivos *IoT* é possível devido a três tecnologias (Yang et al., 2016):

- ***RFID* (*Radio Frequency Identifier*):** Os sistemas de *RFID* consistem em duas partes, uma etiqueta *RFID* que se encontra no objeto que armazena as suas características, e por um leitor de *RFID* que possibilita a leitura das etiquetas. Este sistema é gerado pelo uso de ondas de rádio que transmitem os dados, para que seja possível identificar e rastrear objetos, de uma forma automatizada, sem a necessidade de um campo de visão limpo entre a etiqueta e o leitor.

- ***Wireless Sensor Networks*:** Esta rede é composta por um conjunto de sensores interligados sem fios, capazes de sensorizar o ambiente em que estão inseridos, obtendo informação em tempo real da temperatura, nível de poluição, ruído, entre outros.

- ***Cloud Computing e Big Data*:** Devido ao grande volume de dados gerado pelos dispositivos de *IoT* no processo de produção, o sucesso da implementação dos mesmos depende diretamente da tecnologia *Big Data* e consequentemente da *Cloud Computing*. Sendo que o ciclo de vida da *Big Data* consiste no processo de aquisição, extração, integração, análise e interpretação dos dados, o *Cloud Computing* também desempenha um papel fundamental neste processo, através da capacidade de armazenamento e de computação que oferece.

2.4 *Machine Learning*

O conceito de *Machine Learning* foi descrito pela primeira vez por *Turing*, em 1950, referindo-se a uma máquina, que simulando a mente adulta, conseguisse aprender por si própria. Mais tarde, *Arthur Samuel* cunhou o termo *Machine Learning*, através da aplicação de técnicas de aprendizagem automática no jogo das damas, sendo dadas à máquina apenas as regras do jogo, uma direção e uma lista de parâmetros relacionados com o jogo, mas cujos pesos não são especificados ou conhecidos. Em 1997, *Tom Mitchell*, atribuiu uma definição mais formal a este conceito: "É dito a um programa de computador para aprender através de uma experiência E relativamente a uma tarefa T e da medição da performance P, se a performance em T, medida através de P, melhora através da experiência E" (Mitchell and Mitchell, 1997). Assim sendo, *Machine Learning* pode ser amplamente definido como métodos computacionais utilizando a experiência para melhorar o desempenho ou fazer previsões precisas. Aqui, experiência refere-se às informações anteriores à disposição do algoritmo de aprendizagem, que normalmente se apresentam na forma de dados disponibilizados na forma de conjuntos de treino com identificação humana ou outro tipo de informações obtidas por meio da interação com o ambiente. Em todo o caso, a qualidade e tamanho, são cruciais para o sucesso das previsões feitas pelo algoritmo de aprendizagem. Um exemplo de um problema de aprendizagem é como utilizar uma amostra finita de documentos selecionados aleatoriamente, cada um rotulado com um tópico, para prever com precisão o tópico de documentos não vistos. Claramente, quanto maior for a amostra, mais fácil é a tarefa. Mas a dificuldade da tarefa depende

também da qualidade dos *labels* atribuídos aos documentos da amostra, uma vez que os *labels* podem não estar todos corretos, e do número de tópicos possíveis (Mohri et al., 2018).

2.4.1 *Abordagem Supervisionada*

O contexto de *Machine Learning* tem diversos tipos de aprendizagem subjacentes, podendo ser realizado de forma supervisionada como não supervisionada. A abordagem supervisionada consiste em mapear um conjunto de variáveis de *input* para variáveis *output*, para que através deste mapeamento seja possível realizar previsões de pontos desconhecidos. Sendo assim, sempre que forem introduzidos novos *inputs*, o sistema seja capaz de prever os *outputs*, sendo o modelo ajustado sempre que são introduzidos *inputs* no mesmo (Ayodele, 2010).

Os algoritmos de regressão e classificação são algoritmos de aprendizagem supervisionada. Ambos os algoritmos são utilizados para previsão e ambos funcionam com conjuntos de dados rotulados, no entanto, há diferenças entre os dois e são utilizados para diferentes contextos de *Machine Learning*. Uma tarefa é designada de regressão quando os *outputs* são valores numéricos e chamada de classificação quando os *outputs* tomam valores discretos ou categóricos. Por exemplo, quando se tenta prever o preço de uma casa a partir de um *dataset* trata-se de um caso de regressão pois o preço é um atributo numérico, no entanto, se pretendermos prever se o preço vai ser superior ou inferior ao preço de custo, será um problema de classificação pois o *output* só pode estar entre duas categorias (Matloff, 2017).

Alguns exemplos de algoritmos de regressão são *Linear Regression, Support Vector Regression*, e *Regression Trees* (Doan and Kalita, 2015). Quanto a algoritmos de classificação, alguns exemplos mais comuns são o *Naive Bayes, K Nearest Neighbors, Logistic Regression* e *Decision Trees* (Vijayan et al., 2017).

2.4.2 *Abordagem Não-Supervisionada*

Ao contrário da aprendizagem supervisionada, na aprendizagem não supervisionada não existe um par *input-output*, apenas *input*, sendo que é recebido exclusivamente dados não categorizados (Mehryar et al., 2012). Este tipo de aprendizagem tem como principal objetivo a identificação de padrões previamente desconhecidos nos dados, uma vez que, os dados não se encontram rotulados, classificados ou categorizados previamente, sendo assim, são identificadas semelhanças nos dados e há uma reação com base na presença ou ausência de tais semelhanças em cada novo *input*.

Um dos tipos de algoritmos de aprendizagem não supervisionada mais comum é o de *clustering*, onde é destacado o *K-means*. O algoritmo *K-means* coloca no mesmo *cluster* itens que possuem características semelhantes, sendo que poderá haver k *clusters* distintos, sendo a média dos valores de um determinado *cluster* o centro desse mesmo *cluster*. Outro grupo de algoritmos deste tipo de aprendizagem são os de deteção de anomalias, onde o objetivo é identificar observações que diferem da maioria dos dados (Hodge and Austin, 2004).

2.4.3 *Deteção de Anomalias*

Uma anomalia pode ser definida como um ponto ou sequência de pontos onde existe um desvio do comportamento normal dos dados (Chandola et al., 2009). É um problema que surge numa grande variedade de domínios, incluindo manufatura, economia, transporte e saúde (Aggarwal, 2017). Não existe apenas uma forma de formular uma resposta para a deteção de anomalias, pois diferentes domínios podem definir anomalias de diferentes maneiras (Grubbs, 1969).

As anomalias podem aparecer em diferentes formas. Comumente, existem três tipos diferentes de anomalias (Chandola et al., 2009):

- **Anomalias Pontuais:** Se um ponto se desviar significativamente dos restantes dados, este é considerado um ponto de anomalia. Por exemplo, uma transação de grande quantidade que difere de outras transações é uma anomalia pontual. Assim, um ponto é uma anomalia pontual, se o seu valor difere significativamente de todos os pontos no intervalo.

- **Anomalias Coletivas:** Existe casos em que pontos individuais não são anómalos, no entanto, uma sequência de pontos podem ser rotulados como uma anomalia. Por exemplo, um cliente do banco levanta 500€ da sua conta todos os dias da semana, embora levantar 500€ ocasionalmente seja normal, uma sequência de levantamentos pode ser considerado um comportamento anómalo.

- **Anomalias Contextuais:** Alguns pontos podem ser considerados normais num certo contexto, enquanto detetados como anomalias num outro contexto. Ter uma temperatura média de 35°C no verão em Portugal é normal, enquanto a mesma temperatura no Inverno é considerado uma anomalia.

Ter o conhecimento *a priori* do tipo de anomalia que os dados possam conter, auxilia na seleção dos métodos e técnicas para a deteção das mesmas, uma vez que, algumas abordagens são capazes de detetar anomalias pontuais, no entanto, falham em identificar anomalias coletivas ou contextuais.

A utilização de *Machine Learning* para a deteção de anomalias pode ser abordada de maneira supervisionada e não-supervisionada, sendo a quantidade de informação disponível sobre os dados que determina o tipo de abordagem. No contexto dos sistemas produtivos, o *Machine Learning* pode ser aplicado a conjuntos de dados que contém dados previamente rotulados, como pode ser aplicado a dados com poucos ou nenhuns rótulos, alterando assim os métodos e modelos utilizados. Para além disso, as anomalias ocorrem com pouca frequência nos dados, criando assim uma distribuição desequilibrada de exemplos anómalos e normais, sendo necessário aplicar outro tipo de abordagens como é o caso de técnicas de balanceamento de dados (Chandola et al., 2009).

2.5 *Big Data*

Com os atuais avanços tecnológicos, as organizações enfrentam cada vez mais desafios crescentes para lidar com grandes fontes de dados e de rápido crescimento, apresentando grande complexidade de análise e de uso, procurando ganhar vantagem competitiva através da recolha, armazenamento, processamento e análise dos dados (Villars et al., 2011).

O conceito de *Big Data* tornou-se ubíquo, no entanto o conceito é ambíguo e é difícil de quantificar a partir do momento em que o volume de dados se torna muito grande (Costa and Santos, 2017). Destaca-se sobretudo que os dados não podem ser manipulados ou processados pela maioria dos sistemas de informação, ou métodos atuais, porque os dados na era do *Big Data* tornarão-se-ão grandes demais para serem carregados numa única máquina, mas também implica que a maioria dos métodos de *Data Mining* ou de *Data Analytics* desenvolvidos para uma análise de dados centralizada, podem não ser capazes de ser aplicados diretamente a *Big Data*.

Existe um modelo de 3Vs capaz de caracterizar *Big Data* através de volume, variedade e velocidade (Tsai et al., 2015). O volume é a caraterística mais importante e mais distinta de *Big Data*, que impõe requisitos adicionais e específicos a todas as tecnologias e ferramentas tradicionais utilizadas atualmente. Geralmente, o volume refere-se ao espaço de armazenamento necessário para registar e armazenar dados, normalmente referido em *Terabytes* ou *Petabytes* de espaço de armazenamento (Kitchin and McArdle, 2016). Relativamente à variedade dos dados, estes podem ser classificados como estruturados, semiestruturados e não estruturados (Demchenko et al., 2013). Por fim, a velocidade refere-se à taxa de processamento de dados ou à velocidade de análise e suporte à tomada de decisão (Gandomi and Haider, 2015).

Com o passar do tempo foram surgindo novas caraterísticas, nomeadamente o valor e a veracidade. O valor é uma caraterística importante dos dados, que é definida pelo valor agregado que os dados que são coletados podem trazer para o processo pretendido, atividade ou análise preditiva (Demchenko et al., 2013). A veracidade chama a atenção para possíveis dados imprecisos, sendo que, a análise é feita baseada em conjuntos de dados com vários graus de precisão, autenticidade e confiabilidade, destacando a falta de confiabilidade em certas fontes de dados, no entanto é reconhecido que estas possam ser valiosas quando técnicas e tecnologias adequadas forem implementadas (Chandarana and Vijayalakshmi, 2014). Na figura 3 está representado o modelo capaz de caracterizar *Big Data*.

Existem também outras características que caracterizam *Big Data*, não são tão reconhecidas na literatura, sendo estas a variabilidade e a complexidade (Gandomi and Haider, 2015). A variabilidade está relacionada com as diferentes taxas em que os dados fluem, de acordo com as velocidades inconsistentes, na qual os dados são carregados nas bases de dados, enquanto a complexidade faz referência ao desafio de lidar com múltiplas fontes de dados (Gandomi and Haider, 2015). Além disso, há ainda outras três características: ambiguidade, estando relacionada à falta de metadados adequados, resultando da combinação de volume e variedade; viscosidade, referente ao arrasto causado nos fluxos de dados através do volume e da velocidade; viralidade que mede o tempo de propagação de dados entre os pares numa rede (Krishnan, 2013)).

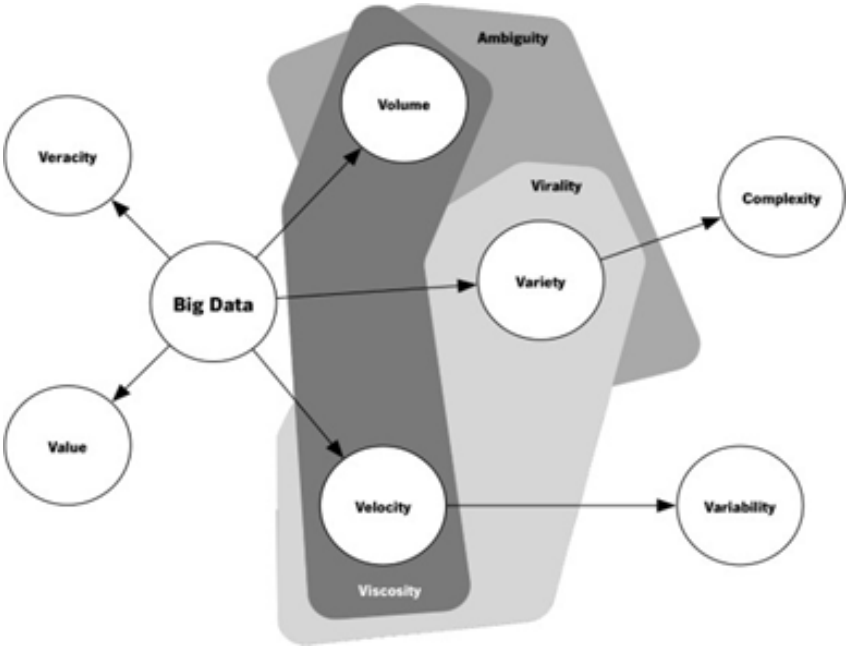


Figura 3: Características de Big Data (adaptado de Costa and Santos, 2017).

Como foi visto anteriormente, quantificar qualquer uma destas características torna-se uma tarefa impossível, sendo assim, o conceito de *Big Data* permanece abstrato, geralmente este refere-se simplesmente ao uso de análises preditivas ou outros métodos avançados para extrair valor dos dados, no entanto, as principais características são 5, sendo as 3 principais variedade, velocidade e volume e as outras duas veracidade e valor definidas posteriormente (Begum et al., 2019).

O *Big Data* não é nada mais do que um processo de recolha de dados com complexidade, diversidade, heterogeneidade e com alto valor potencial, sendo difícil de processar e analisar em tempo razoável (Xu and Shi, 2015). Posto isto, o processo de recolher, armazenar, processar e analisar todos os dados torna-se cada vez mais desafiante. Como existe uma falta de consenso e rigor na definição de *Big Data*, podem surgir diversos desafios, tais como a falta de padronização, assim como a falta de avaliação da qualidade dos dados. A falta de *benchmarks* que permitam a comparação de diferentes tecnologias é agravada pelo constante desenvolvimento tecnológico em ambientes de *Big Data* (Chen and Liu, 2014).

Relativamente ao ciclo de vida de *Big Data* há um conjunto de desafios, relativamente a técnicas de *Big Data* como a recolha, integração, limpeza, transformação, armazenamento, processamento e análise de dados. É necessário repensar e redesenhar as bases de dados e algoritmos para ser possível manipular a quantidade crescente de dados, a escalabilidade é crucial para armazenar e analisar os dados. Sendo assim, a computação distribuída tem um papel crucial no que toca a *Big Data*, pois assegura a disponibilidade, gestão de custos e elasticidade (Chen and Zhang, 2014). Assegurar a qualidade dos dados e extrair valor, através da preparação dos mesmos, é também um desafio em contextos de *Big Data*, sendo que, diferentes fontes de dados têm diferentes problemas de qualidade de dados. Relativamente à visualização, é necessário repensar as abordagens tradicionais devido ao volume dos dados, estas novas abordagens têm de ter a capacidade de escalar para milhares ou milhões de registos, várias fontes de dados, e de fácil utilização, sendo que, há que ter em conta que unir aparências e funcionalidades

é crucial (Chen and Zhang, 2014). É necessário ter em atenção os desafios em relação ao controlo e autoridade sobre grandes quantidades de dados de diferentes fontes, gerir esse ambiente heterogéneo, para serem planeadas políticas de acesso e rastreabilidade, pois pode rapidamente tornar-se um processo quase impossível de realizar sem as ferramentas adequadas.

Por fim, *Big Data* pode parecer muito apelativo para muitas organizações, mas frequentemente os seus líderes não compreendem o seu valor e como extraí-lo. As organizações que implementam iniciativas de *Big Data* enfrentarão muitos desafios, entre os quais se destacam: a falta de consenso em definições, modelos ou arquiteturas; desafios relacionados ao ciclo de vida do *Big Data*; preocupações relacionadas à segurança, privacidade e monitorização; e novas mudanças organizacionais, tal como novas habilidades e mudanças nos processos de negócio (Manyika et al., 2011).

2.6 Arquiteturas Big Data

Com o grande crescimento dos dados, novos métodos para manipulação dos mesmos tiveram que ser criados, uma vez que as abordagens tradicionais de processamento de dados, como a utilização de bases de dados relacionais, não são capazes de lidar com essa quantidade de dados. Os novos sistemas de *Big Data* são projetados para lidar com a ingestão e processamento de uma enorme quantidade de dados para que estes possam ser analisados.

Nas secções apresentadas de seguida, irá ser fornecida uma visão geral de duas arquiteturas de processamento de *Big Data* - *Lambda* e *Kappa*. Estas duas arquiteturas, independentes de tecnologia, são amplamente adotadas nos dias de hoje. São discutidas as vantagens e desvantagens da mesma, assim como a sua aplicabilidade para diferentes casos de uso e para os sistemas produtivos.

Antes de ser aprofundada a análise das diferentes arquiteturas é necessário entender a diferença entre dois conceitos principais de processamento de *Big Data* - *batch processing* e *stream processing*.

2.6.1 *Batch vs. Stream Processing*

Batch processing é um tipo de processamento executado em grandes blocos *(batches)* de dados por um determinado período de tempo. Estes dados, são geralmente armazenados em formato bruto ou em forma de registos em tecnologias de armazenamentos de dados altamente escaláveis, como é o caso do *HDFS* e do *Cassandra*, e são analisados periodicamente em *batches* por tecnologias de processamento de *Big Data*, sendo que para estes casos de uso o tempo não é um fator crítico. Por outro lado, o *stream processing* é necessário quando existe a necessidade de processamento de dados em tempo real, sendo necessário ter em conta que é preciso agir à medida que os dados vão sendo gerados pelo sistema produtivo, sendo de extrema importância a capacidade de se atingir latências muito baixas e altas taxas de transferência de dados (Benjelloun et al., 2020).

Sendo assim, a principal diferença entre *batch processing* e *stream processing* é o propósito de utilizar cada um deles. O *batch processing* é utilizado ao lidar com quantidades muito elevadas de dados que

necessitam de processamento complexo ou simplesmente quando a fonte de dados não consegue fornecer um fluxo contínuo. Por outro lado, o *stream processing* é utilizado quando existe um objetivo de obter resultados em tempo real, sendo os dados processados à medida que são gerados para se poder obter análises em tempo real. Os dados aqui são caracterizados pela velocidade e não pelo volume, embora estes também possam lidar com grandes quantidades de dados (Yassine et al., 2019).

2.6.2 *Arquitetura Lambda*

A arquitetura *Lambda*, representada na fig. 4), é projetada para lidar com uma grande quantidade de dados de forma eficiente, aproveitando os métodos de *stream e batch processing*. Eficiência neste contexto, como apontado anteriormente, traduz-se em taxas de transferência altas, baixa latência e tolerância a falhas (Samizadeh, 2018). O crescimento deste tipo de arquitetura está diretamente relacionado ao grande crescimento dos dados e à velocidade com que os mesmos são gerados.

Geralmente, uma arquitetura *Lambda* é constituída por três camadas distintas: *Batch Layer*, *Speed Layer* e *Serving Layer*. A *Batch Layer* é responsável por proporcionar visualizações abrangentes e precisas de dados *batch*, enquanto que a *Speed Layer* fornece visualizações de dados quase em tempo real.

Os dados que entram no sistema geralmente são temporais, imutáveis e armazenados em formato bruto. Os dados podem alimentar o sistema através de vários sistemas, por exemplo, utilizando um sistema de mensagens distribuídas, uma base de dados ou um sistema de arquivos distribuído. Para a ingestão de dados, normalmente são utilizadas tecnologias de mensagens distribuídas, uma vez que fornecem grande escalabilidade e fácil acoplamento. *Apache Kafka* e *Amazon Kinesis* são tecnologias populares de mensagens distribuídas e podem ser aplicadas no contexto dos sistemas produtivos. Estes fornecem um sistema confiável, de alto rendimento e baixa latência de fluxos de dados em tempo real (Ranjan, 2014). As estruturas de *Big Data* geralmente não suportam o processamento em *batch* e *streaming*, sendo assim, uma combinação híbrida de *frameworks* é necessária para ser atingido este propósito, por exemplo, uma combinação de *Apache Hadoop* e *Apache Spark* pode ser utilizado para suportar totalmente a arquitetura *Lambda*.

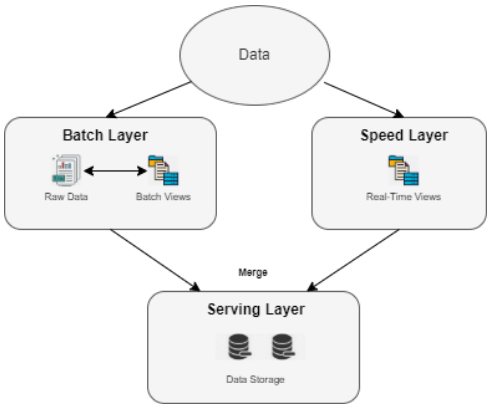


Figura 4: Arquitetura Lambda (adaptado de Iftikhar et al., 2020).

Batch Layer

Os fluxos de dados que entram no sistema de *Big Data* são alimentados duplamente em ambas as camadas, *Batch Layer* e *Speed Layer*. A *Batch Layer* armazena dados brutos à medida que estes chegam e realiza a computação dos mesmos em intervalos de tempo específicos. Os dados podem ser armazenados utilizando diferentes tecnologias, podendo ainda ser utilizados algoritmos de *Machine Learning* que podem ser úteis para a análise dos dados. Entre os *frameworks* mais populares que suportam o processamento de dados em *batch* no contexto dos sistemas produtivos podem destacar-se o *Apache Hadoop*, *Apache Spark* e o *Apache Storm* (Cui et al., 2020; Zhong et al., 2017; Lima-Monteiro et al., 2016; Lee et al., 2017). Cada um destes *frameworks* tem as suas vantagens e desvantagens que têm de ser consideradas na escolha dos mesmos. Os parâmetros mais importantes que devem ser considerados são a latência, tolerância a falhas e as taxas de transferência, sendo a latência neste contexto o tempo de processamento do conjunto de dados, que pode variar entre vários minutos a várias horas, dependendo da quantidade de dados a serem processados.

Speed Layer

Esta camada processa os fluxos de dado em tempo real com foco em latências mínimas, sendo que, normalmente, as latências podem variar entre milissegundos a vários segundos. Várias *frameworks* de processamento de *Big Data* suportam processamento em tempo real, como por exemplo, *Apache Flink*, *Apache Storm*, *Apache Spark Streaming* e *Apache Samza*.

A *Speed Layer* é responsável por indexar visualizações em tempo real, compensando a alta latência da camada *batch*. Em particular, devido aos grandes conjuntos de dados na camada *batch*, leva tempo até que as vistas sejam calculadas causando alguma indisponibilidade de dados. A *speed layer* é utilizada para fechar essa lacuna, fornecendo uma maneira eficiente de calcular os dados mais recentes (Warren and Marz, 2015). Assim que a camada *batch* computou as visualizações, a *speed layer* descarta os dados redundantes, e portanto, passam a ser fornecidos pela camada *batch*.

Serving Layer

Os *outputs* das camadas *batch* e *speed* na forma de visualização são armazenadas na *serving layer*. Como *frameworks* de armazenamento são frequentemente utilizados *data stores* altamente escaláveis e distribuídos. No contexto dos sistemas produtivos, a escolha geralmente está dividida entre bancos de dados *NoSQL*, sistemas distribuídos de arquivos, bases de dados de séries temporais ou combinação de mais tipos de sistemas de armazenamento de dados. Entre os armazenamentos de dados de *Big Data* mais populares estão *Apache Cassandra*, *Apache HBase*, *HDFS*, *OpenTSDB* e *KairosDB*.

Sendo assim, esta camada trata da indexação e fornecimento das visualizações de ambas as camadas anteriores, permitindo o fácil acesso ao utilizador.

Vantagens vs. Desvantagens

Nesta secção irão ser descritas algumas vantagens e desvantagens da utilização deste tipo de arquitetura (Iftikhar et al., 2020):

Vantagens:

- É um tipo de arquitetura tolerante a falhas. Após a ocorrência de falhas, os resultados podem ser recalculados porque os mesmos são armazenados em ambiente replicado e distribuído.
- Permite escalabilidade, uma vez que se pode adicionar mais máquinas à arquitetura, tornando o sistema mais distribuído e aumentando a *performance*.
- É adequado a algoritmos complexos, possibilitando o uso de qualquer modelo de *Machine Learning* neste tipo de arquitetura, desde que acompanhado dos *frameworks* que o permitam.
- Fornece latências próximas do tempo real.

Desvantagens:

- Pode tornar-se uma arquitetura de elevada complexidade, uma vez que alguns dos *frameworks* possam ter de suportar processamento *batch* e *streaming* ou utilizar combinações híbridas.
- A nível de código, é necessário um maior controlo e manutenção dos mesmos, uma vez que necessitam de ser mantidos conjuntos de código para *batch* e *streaming*.

2.6.3 *Arquitetura Kappa*

Esta arquitetura, representada na figura-5, foi inicialmente proposta em 2014 (Kreps, 2014), tendo em vista alguns problemas da arquitetura *Lambda*, como a manutenção de duas bases de código e a complexidade geral da plataforma. Esta arquitetura propõe melhorar o sistema de processamento de *streaming* para lidar com todos os conjuntos de problemas que possam surgir.

A arquitetura *Kappa* não é um substituto, mas uma alternativa à arquitetura *Lambda* e pode ser utilizado em certos cenários, onde o esforço para a utilização da arquitetura *Lambda* não seja necessário (Feick et al., 2018). *Ranjan*, em 2014, propôs uma arquitetura baseada em monitoramento de dados *IoT* utilizando *Spark Streaming* (Ranjan, 2014). A arquitetura *Kappa* consiste em duas camadas distintas: *Speed Layer* e *Service Layer*.

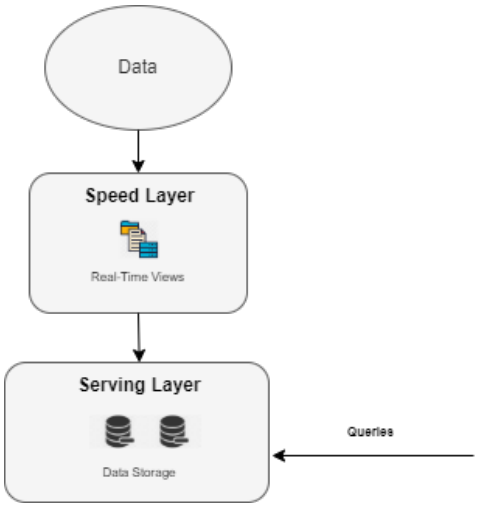


Figura 5: Arquitetura Kappa (adaptado de Feick et al., 2018).

Speed Layer

Esta camada processa os fluxos de dados da mesma forma que a arquitetura *Lambda*. Além disso, podem ser utilizadas as mesmas tecnologias, por exemplo, *Apache Flink, Apache Samza, Apache Storm*, sendo que, a única diferença é que quando existe uma mudança de código, é necessário reprocessar os dados novamente. Isto ocorre porque a arquitetura *Kappa* atua como um *online learner*, permitindo que o modelo seja continuamente atualizado conforme a introdução de novos dados.

Serving Layer

Quanto a esta camada é utilizada da mesma forma que na *Arquitetura Lambda*, podendo ser utilizadas as mesmas tecnologias.

Vantagens vs. Desvantagens

Nesta secção irá ser descrito algumas vantagens e desvantagens da utilização deste tipo de arquitetura (Feick et al., 2018):

Vantagens:

- É um tipo de arquitetura em que são necessários pouco recursos, uma vez que é apenas necessário manter um tipo de *framework*.
- Permite escalabilidade, uma vez que se pode adicionar mais máquinas à arquitetura, tornando o sistema mais distribuído e aumentando a *performance*.
- Pode ser utilizado em sistemas em que são necessários *online learners*, isto é, que se visa recolher, integrar, processar, analisar e visualizar os dados em tempo-real
- O reprocessamento dos dados é apenas necessário quando existe mudanças no código.

Desvantagens:

- A ausência de uma camada *Batch* restringe a arquitetura *Kappa* a ser adequada apenas para um conjunto específico de problemas de processamento de dados.

2.6.4 *Arquitetura Lambda vs. Arquitetura Kappa*

Embora existam inúmeras semelhanças, diferenças consideráveis surgem do facto de que a arquitetura *Kappa* é desprovida de uma camada *batch*. Dessa forma, a arquitetura requer apenas uma base de código, em vez de serem implementados dois sistemas heterogéneos(Ordonez-Ante et al., 2016). Como resultado, processos relacionados ao desenvolvimento como a implementação e manutenção de código são simplificadas. Por outro lado, a não existência de uma camada *batch* resulta na incapacidade da arquitetura gerir e implementar *softwares* de computação intensiva. Este é o caso relativo a cenários de *Machine Learning* em grande escala em que um modelo necessita de ser treinado (Zhelev and Rozeva, 2017). Além disso, o desempenho das tarefas de processamento *batch* sofre da indisponibilidade de uma camada *batch* (Lin, 2017). No entanto, as desvantagens da arquitetura *Kappa* não são particularmente problemáticos, pois *Kreps* sugeriu esta abordagem como uma alternativa à arquitetura *Lambda* valorizando a simplicidade sobre a eficiência (Kreps, 2014). Isso significa que uma comparação entre as duas arquiteturas é difícil, uma vez que, o desempenho da arquitetura em grande parte depende do caso de uso. Portanto, a arquitetura mais adequada deve ser sempre escolhido tendo em conta o cenário de aplicação fornecido.

2.7 *Hadoop Framework*

A era do *Big Data* em que entramos há vários anos mudou a nossa imaginação sobre o tipo e o volume de dados que podem ser processados, assim como o valor dos dados. O *Apache Hadoop* é uma tecnologia de *Big Data* tendo sido projetada para que fosse evitado o baixo desempenho e a complexidade no processamento e análise de grandes quantidades de dados utilizando sistemas tradicionais. Esta tecnologia tem capacidade para processar grandes conjuntos de dados, graças aos *clusters* paralelos e sistema de arquivos distribuído, é capaz também de executar programas ao mesmo tempo, sendo garantida a tolerância a falhas, evitando assim a perda de dados através da replicação de dados em servidores. O poder da plataforma *Hadoop* é baseado em dois subcomponentes principais: o *Hadoop Distributed File System* (*HDFS*) e a estrutura *MapReduce*. Para além disso, os utilizadores podem adicionar módulos conforme necessário, de acordo com os objetivos ou requisitos (capacidade, desempenho, escalabilidade, segurança) (Oussous et al., 2018).

2.7.1 *Hadoop Distributed File System*

O *HDFS* é um sistema de arquivos distribuído projetado para ser executado em *hardware* de comodidade. Este tem muitas semelhanças com os sistemas de arquivos distribuídos existentes, no entanto, este é altamente tolerante a falhas e projetado para ser implementado em *hardware* de baixo custo, fornecendo alta taxa de transferência de acesso aos dados e é adequado para grandes conjuntos de dados (Borthakur et al., 2008).

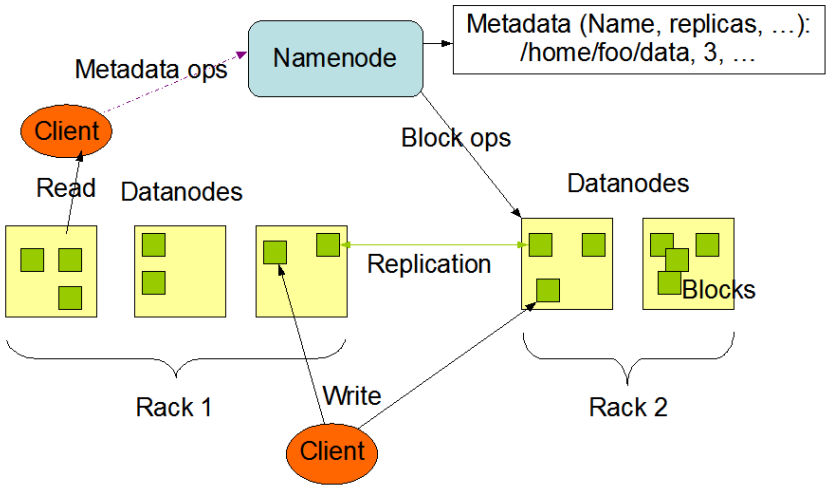


Figura 6: Arquitetura HDFS (adaptado de Saraswat et al., 2017).

O *HDFS* é baseado na arquitetura *master-slave* (ver fig.6). Um *cluster HDFS* consiste em um único *NameNode*, um servidor que gere o *namespace* do sistema de arquivos e regula o acesso aos arquivos pelos clientes. Além disso, existe uma série de *DataNodes*, geralmente um por cada nó do *cluster*, que gere o armazenamento aos nós em que eles são executados. Internamente, um arquivo é dividido em um ou mais blocos e esses blocos são armazenados num conjunto de *DataNodes*. O *NameNode* executa operações do sistema, como abrir, fechar e renomear arquivos ou diretórios, sendo ainda responsável por atender solicitações de leitura e gravação dos clientes do sistema de arquivos. Os *DataNodes* podem também criar, excluir e replicar blocos sob instrução do *NameNode*.

2.7.2 *MapReduce*

O *MapReduce* proporciona que todos os programadores pensem de uma forma centralizada nos dados, isto é, focados na aplicação de transformações a conjuntos de dados e permite que os detalhes das execuções distribuídas, das comunicações em rede e a tolerância a falhas sejam tratados pela própria *framework MapReduce*. (Condie et al., 2010).

Posto isto, o *MapReduce* é um modelo de programação, bem como uma estrutura que suporta esse modelo. A ideia principal do *MapReduce* é ocultar os detalhes de execuções paralelas e permite que os utilizadores se foquem apenas nas estratégias de processamento.

Arquitetura

O modelo *MapReduce*, representado na figura-7, consiste em duas funções primitivas: *Map* e *Reduce*. A entrada para a ferramenta *MapReduce* é uma lista de pares (chave1, valor1) e a função *Map*() é aplicada a cada par para calcular pares de chave-valores intermediários, (chave2, valor2). Os pares de chave-valor intermédios são agrupados com base na igualdade de chave, ou seja, (chave2, lista (valor2)). Para cada chave2, a função *Reduce*() trabalha na lista de todos os valores e, de seguida, produz zero ou mais resultados agregados, sendo que, os utilizadores podem definir as funções *Map*() e *Reduce*() da forma que quiserem. Esta tecnologia utiliza o *GFS* como uma camada de armazenamento subjacente para ler a entrada e armazenar a saída (Lee et al., 2012).

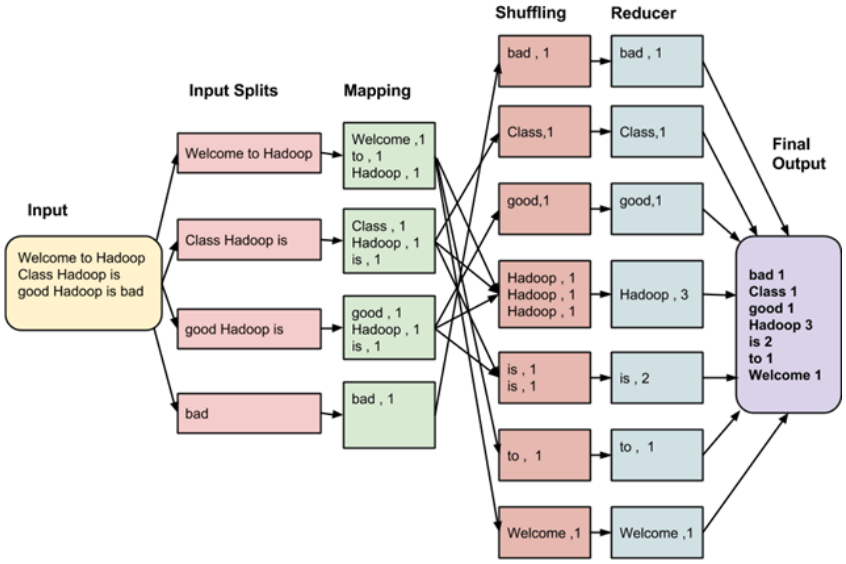


Figura 7: Arquitetura MapReduce (retirado de Lee et al., 2012).

- ***Input Splits*:** Os *Input Splits* são partições do *input* que são consumidas por um único *Map*.
- ***Mapping*:** Esta é a primeira fase na execução do programa *MapReduce*. Nesta fase, os dados em cada divisão são passados para uma função de mapeamento para produzir valores de saída.
- ***Shuffling*:** Esta fase consome a saída da fase do *Mapping*, tendo como função consolidar os registos mais importantes.
- ***Reducer*:** Nesta fase, os valores de saída da fase de *Shuffling* são agregados. Esta fase combina os valores da fase *Shuffling* e retorna um único valor de saída. Em suma, esta fase resume o conjunto de dados completo.

2.7.3 *Síntese da revisão de literatura Hadoop Framework*

Hadoop é uma das melhores *frameworks* para manter, analisar, processar e gerir grandes quantidades de dados. Por outro lado, permite ainda a facilidade e a rapidez para aceder a dados em diferentes servidores do *cluster*. Dentro dos trabalhos efetuados podem destacar-se:

- *Honnutagi* (2014), utilizou o modelo de programação *MapReduce* para realizar o processamento de dados armazenados no *HDFS*. Através do *HDFS* é possível processar os dados e contar o número de palavras num ficheiro de dados de grande dimensão (Honnutagi, 2014).

- *Sudheer and Lakshmi (2015),* realizaram uma comparação entre a *performance* de um *cluster HDFS single node* e um *cluster HDFS multinode*. Chegaram à conclusão de que no *cluster single node* todos os *demons* são executados em uma única máquina e por esse motivo a carga dos processos *Java* é pesada. Consequentemente grandes conjuntos de dados não podem ser processados com eficiência pelo *HDFS*, visto que a utilização de recursos é maior do que aquela que é possível fornecer. Por outro lado, no *multi node cluster* todos os *demons* são executados em máquinas diferentes e, por esse motivo, o processamento de dados será mais rápido e eficiente do que o *single node cluster*. Sendo assim, o *HDFS* apenas alcançará alta disponibilidade, confiabilidade, tolerância a falhas e mais recursos no *multi node cluster* (Sudheer and Lakshmi, 2015).

- *Sajwan et al.* (2015), realizaram uma revisão profunda ao *HDFS*, a arquitetura e os seus componentes. Analisaram diferentes desafios no que diz respeito à utilização do *HDFS* e as suas soluções e ainda alguns atributos do mesmo, com o objetivo de demonstrar o porquê de utilizar o *HDFS* sobre bases de dados relacionais (Sajwan et al., 2015).

- *Lu and Alwerfali* (2016) realizaram a avaliação da *performance* de um *cluster Hadoop* através da construção de um ambiente constituído por 9 nós, utilizando 2 *datasets* para medir a *performance*. Avaliaram o desempenho de escrita do *Hadoop* e chegaram à conclusão de que é melhor do que a leitura com pequenos conjuntos de dados. No entanto, não tem grande relevância, uma vez que o *Hadoop* foi projetado para o processamento em lote de grandes conjuntos de dados. Verificaram também, que o desempenho de escrita e leitura era bastante rápido e que quantos mais nós de dados alocarem, mais rápido é o processamento (Lu and Alwerfali, 2016).

- *Lin and Lin* (2017), adotaram uma abordagem holística para alcançar o balanceamento de carga no *HDFS*, considerando todas as situações que podem influenciar o estado de balanceamento de carga. Embora o *HDFS* tenha uma ferramenta integrada que permite alcançar o balanceamento de carga, este tem um desempenho reduzido devido à movimentação de blocos. Sendo assim, foi projetada uma nova função chamada *BalanceNode* para ajudar na correspondência de *DataNodes* com carga elevada e carga leve, para que esses nós com carga leve possam compartilhar parte da carga dos nós de carga elevada. Foi também projetada uma estratégia de posicionamento de

bloco de forma a tornar a carga de armazenamento mais equilibrada. Os resultados da simulação demonstraram que a abordagem pode alcançar um estado de balanceamento de carga melhor do que os algoritmos existentes (Lin and Lin, 2017).

Na Tabela 1 está representada a revisão da literatura do *Hadoop Framework* referida anteriormente.

Tabela 1: Revisão da Literatura Hadoop Framework

Autor	**Título**	**Objetivo**
(Honnutagi, 2014)	The hadoop distributed file system	Utilização do modelo de programação *MapReduce* para processamento de dados armazenados no *HDFS*.
(Sudheer and Lakshmi, 2015)	Performance evaluation of Hadoop distributed file system	Avaliar a *performance* de um *single node cluster e* de um *multi node cluster* com as mesmas configurações e parâmetros.
(Sajwan et al., 2015)	The hadoop distributed file system: Architecture and internals	Implementação *single rack vs. multi rack* do *HDFS* e análise dos problemas de *performance*.
(Lu and Alwerfali, 2016)	Implementation and Performance Analysis of Apache Hadoop	Avaliar a *performance* de um ambiente simulado de um *cluster Hadoop* composta por *NameNodes* e *DataNodes*.
(Lin and Lin, 2017)	An overall approach to achieve load balancing for Hadoop Distributed File System	Projeção de uma nova função denominada *BalanceNode* para balanceamento da carga de *DataNodes*.

2.8 Ecossistema *Apache Hadoop* e Tecnologias relacionadas

Nesta secção irão ser identificadas algumas tecnologias do ecossistema *Apache Hadoop*, assim como tecnologias compatíveis com este ecossistema que poderão ser utilizadas na construção da plataforma. O ecossistema *Apache Hadoop* revolucionou a maneira como processamos e analisamos os dados e a quantidade de informações importantes e valiosas que podemos obter dos dados. O ecossistema *Apache Hadoop* cobre uma ampla coleção de plataformas, estruturas, ferramentas, bibliotecas e outros serviços capazes de fazer análises de dados rápidas, confiáveis e escaláveis, assim como realizar o processamento distribuído de grandes conjuntos de dados em *clusters* de computadores (Mrozek, 2018).

2.8.1 *Apache NiFi*

Apache NiFi foi desenvolvido anteriormente pela *National Security Agent* (*NSA*), é um projeto *open source* para automatização e monitorização de movimento de dados entre redes de sistemas heterogéneos. É capaz de analisar diferentes formatos de dados e procolos, sendo que opera em *JVM* (Chatti, 2019). O *NiFi* implementa o conceito de *FBP* (*Flow Based Programming*) como um processo de *ETL* em tempo real para coletar grandes quantidades de dados num ambiente distribuído. O processamento em tempo real é possível, caso sejam necessários operações complexas, pode ser utilizado em conjunto com outras tecnologias como o *Spark*. Os componentes do *NiFi* são os seguintes (Kim et al., 2019):

- ***FlowFile*:** Um objeto que representa dados. Este pode conter atributos de dados e conteúdo na forma de chave/valor. Os dados podem ser movidos entre sistemas através do *FlowFile*.
- ***FlowFile Processor*:** *FlowFile* pode ser adicionado ou alterado em várias etapas. O *FlowFile Processor* fornece mais de 150 processadores, que permite que se possa ler, alterar e guardar um *FlowFile* de vários sistemas.
- ***Connection*:** Fornece a utilização de funções poderosas como definição de rotas, limitação de taxa de transferência, controlo de prioridade.
- ***Flow Controller Processor*:** Agenda o processador para que este seja executado em um intervalo ou data específica.
- ***Process Group*:** Uma tarefa específica ou unidade funcional pode agrupar vários processadores, sendo possível mover dados entre grupos de processos, fornecendo portas de *input* e *output*.

2.8.2 *Apache Kafka*

Nos dias de hoje, as informações em tempo real são continuamente geradas por aplicações e essas informações necessitam de maneiras fáceis de serem distribuídas para vários tipos de consumidores. Na maioria das vezes, as aplicações que estão a produzir informações e as aplicações que estão a consumir as informações estão bem separadas e inacessíveis umas às outras (Thein, 2014). O *Apache Kafka* é desenvolvido pela *Apache Software Foundation* escrito em *Scala* e *Java*, é uma plataforma de processamento *open source* com o objetivo de fornecer uma plataforma unificada, de alto rendimento e baixa latência para lidar com dados em tempo real (Shaheen, 2017).

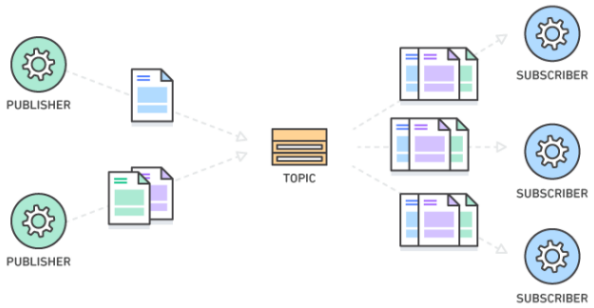


Figura 8: Publish/Subscribe System (retirado de Hiraman et al., 2018).

O *Apache Kafka* permite lidar com volumes em tempo real de dados e encaminhar os mesmos para vários consumidores rapidamente. É uma tecnologia para *streaming* de eventos, que permite que sejam criados fluxos de dados entre vários sistemas e/ou aplicações. Este segue o padrão *publisher/subscriber* (ver fig.8), permitindo a importação e exportação dos dados do sistema continuamente. Permite também o armazenamento de fluxos de *streaming* durante um longo período, de forma confiável, tornando possível que se realize o processamento desses fluxos, conforme eles ocorrem. Para além disso, estas funcionalidades são fornecidas de forma distribuída, altamente escalável, tolerante a falhas e com um elevado nível de segurança (Hiraman et al., 2018).

Arquitetura

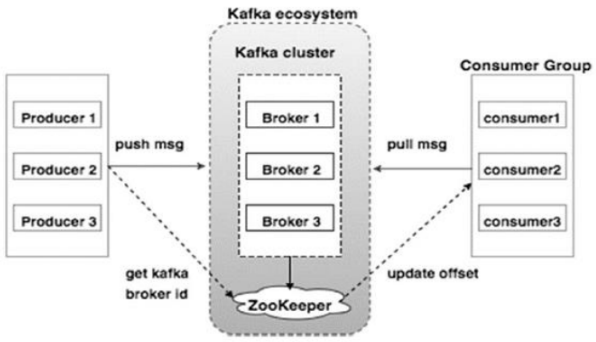


Figura 9: Arquitetura Apache Kafka (Adaptado de Le Noac'H et al., 2017).

Sendo assim, o *Kafka* é um sistema distribuído que consiste em servidores e clientes que se comunicam por meio de um protocolo de rede *TCP* de alto desempenho. Este pode ser implementado em *hardware* de comodidade, máquinas virtuais e *containers*, bem como em ambientes *cloud*. O *Kafka* é executado como um *cluster* de um ou mais servidores que podem abranger vários *datacenters* ou regiões na *cloud*. Alguns desses servidores formam uma camada de armazenamento, chamada de *brokers*. Outros servidores executam o *Kafka Connect* para importar e exportar dados continuamente como fluxos de eventos para integrar o *Kafka* com os seus sistemas, bem como outros *clusters Kafka*. Um *cluster Kafka* é altamente escalável e tolerante a falhas, se algum dos servidores falhar, os outros servidores assumirão o seu trabalho para garantir operações contínuas sem qualquer perda de dados. Os *brokers* desempenham uma função determinante para o funcionamento do *Kafka*, onde cada *broker* é executado numa única máquina, e o conjunto dos *brokers* compõem o *cluster* do *Kafka*. Ainda dentro do *cluster*, o *Apache Zookeeper*, sendo um serviço centralizado, coordena o *Kafka* como um sistema distribuído (Le Noac'H et al., 2017).

2.8.3 *Apache Spark*

Apache Spark é uma tecnologia que surgiu como um mecanismo unificado para análises de grandes quantidades de dados. Introduziu uma nova abordagem para as áreas de ciência e engenharia dos dados, onde é possível resolver os problemas que podem estar inerentes aos dados utilizando um único mecanismo de processamento. É atualmente a ferramenta de *Big Data* mais utilizada a nível global (Salloum et al., 2016).

Spark é um sistema de computação em *cluster* especializada em tornar análises de dados mais rápidas, suportando computação em memória. Unifica *streaming, batch* e *workload* de dados para criar aplicações. Esta tecnologia tem uma programação semelhante ao *MapReduce*, porém, com uma caraterística adicional, a abstração da partilha dos dados chamada "*Resilient Distributed Datasets*". Com esta extensão, o *Spark* consegue detetar *workflows* que precisam de diferentes mecanismos de processamento, incluindo *SQL*, *Machine Learning*, e processamento de gráficos. *Spark* é caraterizado por inúmeras vantagens tais como, a maior facilidade em desenvolver aplicações visto que é utilizada uma *API* unificada, a maior eficiência para combinar tarefas de processamento e a disponibilidade de novas aplicações como por exemplo as consultas interativas em gráficos (Zaharia et al., 2016). Esta sistema também fornece um grande número de ferramentas de alto nível, como a ferramenta de *Machine Learning MLib*, a ferramenta *SparkSQL* para processamento de dados estruturados, ferramenta de processamento de gráficos, *Graph X* e o motor de processamento de *streaming* conhecido como *Spark Streaming* (Shoro and Soomro, 2015). Na figura 10 está representada a arquitetura do *Apache Spark*.

Arquitetura

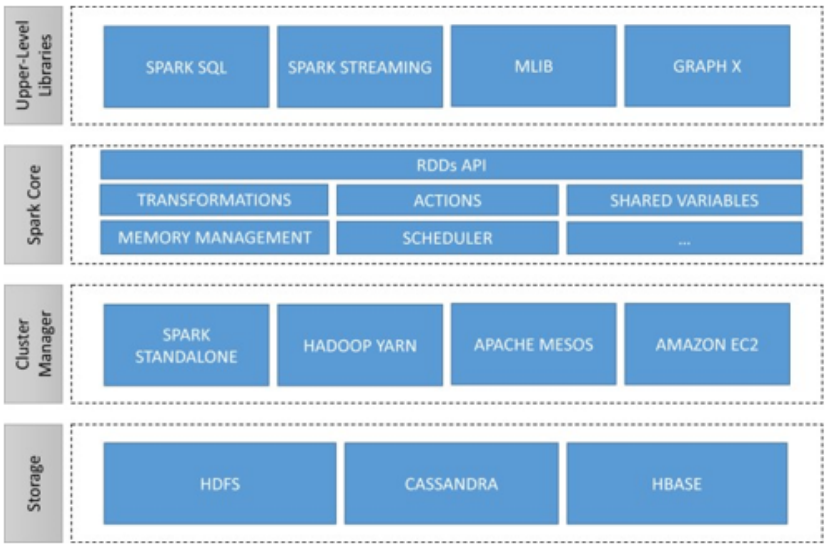


Figura 10: Arquitetura Apache Spark (Adaptado de Salloum et al., 2016).

A arquitetura do *Apache Spark* representada anteriormente é constituída pelos seguintes módulos:

- ***Upper-Level Libraries*:** Como referido anteriormente, as livrarias existentes no *Spark* são o *Spark SQL* para processar dados estruturados, o *Spark Streaming* para análises de *streaming*, o *MLLib* destinado a *Machine Learning* e o *GraphX* para processamento de gráficos.
- ***Spark Core*:** *Spark Core* é responsável por fornecer uma interface de programação simples para o processamento de grandes quantidades de dados, a *RDD API*. É implementada em *Scala* mas fornece várias *API* em *Scala*, *Java*, *Python* e *R* que suportam diferentes tipos de operações essenciais para algoritmos de análise de dados.
- ***Cluster Manager and Storage*:** É utilizado para adquirir recursos de *cluster* para executar *jobs*. O *Spark Core* é executado em diferentes gestores de *cluster* como o *Hadoop Yarn*. O *Cluster Manager* lida com o compartilhamento de recursos entre aplicações *Spark* que pode ainda aceder aos dados em *HDFS, Cassandra, HBase, Hive* e qualquer outra fonte de dados *Hadoop* (Salloum et al., 2016).

2.8.4 *Apache Druid*

O *Apache Druid* é uma base de dados analítica que potencializa casos de uso de análise em tempo real, tendo sido projeto para alcançar o máximo de desempenho e escalabilidade que pode ser alcançada ao ingerir e explorar grandes quantidades de eventos transacionais combinando várias técnicas, ideias e arquiteturas de vários *softwares* dedicados a resolver problemas particulares (Correia et al., 2019)

Arquitetura

Um *cluster Druid* consiste em diferentes nós, em que cada nó é designado para um conjunto de funções específicas. Este formato separa as preocupações e simplifica a complexidade do sistema, uma vez que, os diferentes tipos de nós operam de forma independente entre si e existe pouca interação entre eles. Sendo assim, as falhas de comunicação *intra-cluster* têm um impacto mínimo nos dados disponíveis. Para resolver problemas complexos de análise de dados, os diferentes tipos de nós reúnem-se para formar um sistema totalmente funcional.(Yang et al., 2014) A composição de um *flow* de dados num *cluster Druid* está representado na figura 11.

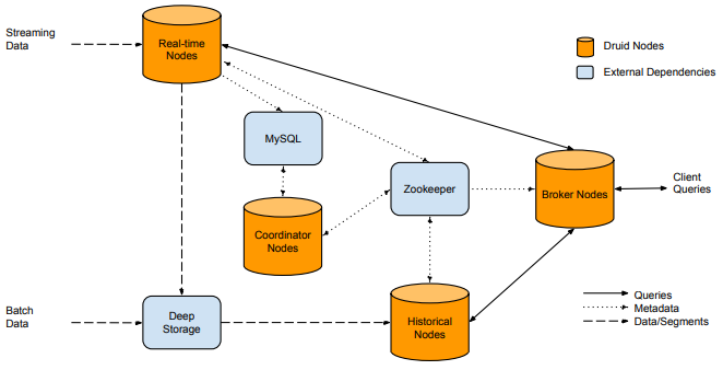


Figura 11: Composição de um Druid Cluster e flow dos dados dentro do cluster (Adaptado de Yang et al., 2014).

Sendo assim, os tipos de nós constituintes de um *cluster Druid* são:

- ***Real-Time Nodes*:** Este tipo de nós encapsulam a funcionalidade de gerir e consultar fluxos de eventos. Os eventos indexados por meio desses nós ficam imediatamente disponíveis para consulta.

- ***Historical Nodes*:** Este tipo de nós ficam encarregues de carregar e servir os blocos imutáveis de dados (segmentos) criados pelos *Real-Time Nodes*. Em muitos fluxos de dados, a maioria dos dados carregados num *cluster Druid* são imutáveis e portanto, este tipo de nós são os principais trabalhadores de um *cluster Druid*.

- ***Broker Nodes*:** Atuam como consultores de dados para os *Real-Time Nodes* e para os *Historical Nodes*. Estes entendem os metadados publicados no *Zookeeper* e identificam quais são os segmentos que são consultáveis e onde esses segmentos estão localizados.

- ***Coordinator Nodes*:** São responsáveis pela gestão e distribuição de dados nos *Historical Nodes*. Coordenam os restantes nós para que seja possível carregar, eliminar e replicar dados para balanceamento de carga.

2.8.5 *Apache Cassandra*

Apache Cassandra é um sistema de gestão de bases de dados distribuído de código aberto. É uma solução *NoSQL* que foi inicialmente desenvolvido pelo *Facebook*, pertencendo à *Apache Software Foundation* desenhado para lidar com grandes quantidades de dados espalhados por vários servidores enquanto fornece um serviço altamente disponível sem um único ponto de falha (Cassandra, 2014).

Modelo de Dados

A estrutura lógica da base de dados é denominada modelo de dados. Esta interpreta como os dados são armazenados, acedidos e define as relações entre os diferentes tipos de dados. No *Cassandra*, os dados são armazenados em *key stores* que são semelhantes a uma base de dados num ambiente de bases de dados relacionais. Dentro desses *key stores*, os dados são armazenados em forma de tabelas, podendo ser chamado de família de colunas. Uma tabela no *Cassandra* é um mapa multidimensional distribuído indexado por uma chave (REDDY and REDDY, 2020). As chaves mapeiam para vários valores, que são agrupados em famílias de colunas. As famílias de colunas são fixas quando uma base de dados *Cassandra* é criada, mas as colunas podem ser adicionadas a uma família a qualquer momento. Para além disso, as colunas são adicionadas apenas às chaves especificadas, portanto, chaves diferentes podem ter diferentes números de colunas em qualquer família (Lakshman and Malik, 2010). O modelo de dados do *Cassandra* está representado na figura 12.

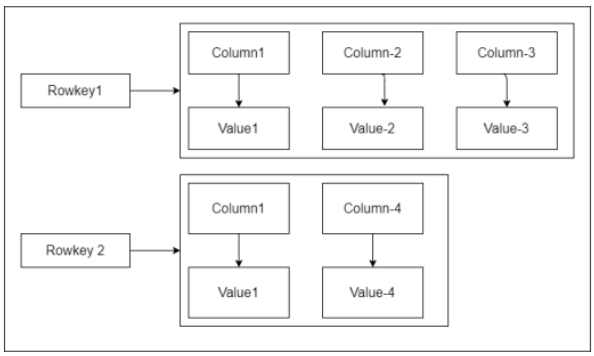


Figura 12: Modelo de dados Cassandra (Adaptado de REDDY and REDDY, 2020).

2.8.6 *Apache Hive*

Apache Hive foi introduzido pelo *Facebook* em 2010, escrito em *Java* e disponível em *SQL*. Encontra-se no topo do *Apache Hadoop* para fornecer consulta e análise de dados. O *Hive* oferece uma interface semelhante ao *SQL*, para consultar dados e armazená-lo em várias bases de dados.

Apache Hive é uma interface de código aberto geralmente utilizado para análise de dados. É uma linguagem, como referido anteriormente, semelhante ao *SQL*, conhecido como *HQL* sendo conhecida

como uma ferramenta *ETL* apropriada. Esta tecnologia suporta dois motores de execução, o *MapReduce* e o *Spark* (Huai et al., 2014). Os componentes do *Hive* podem ser divididos em (Bhardwaj et al., 2015):

- ***User Interface*:** Suporta 3 tipos de interface gráfica como é o caso do *WebUI*, *Command Line* e *Hive HD Insight*.
- ***Meta Store*:** Para armazenamento dos esquemas ou metadados das tabelas, tipo de dados e colunas.
- ***HiveQL Process Engine*:** *HiveQL* é similar ao *SQL* sendo uma substituição à abordagem tradicional de programação *MapReduce*.
- ***Execution Engine*:** É utilizado para processamento de consultas aos dados e gera resultados similares ao *MapReduce*.

2.8.7 *Presto*

É um mecanismo de consulta distribuído de baixa latência, interativo e compatível com o *SQL*, sendo otimizado para análise *ad-hoc*. Todo o processamento é realizado em memória e canalizado através de uma rede entre as etapas, que reduz o tempo de leitura/gravação no disco, melhorando assim o desempenho. As deficiências do sistema são a sua incapacidade de gravar dados de saída novamente em tabelas, pois este só suporta o modo de leitura. Na arquitetura *Presto*, como demonstrado na Figura 13, existe um *coordinator* que recebe consultas *SQL* do cliente, que então analisa e planeia a execução. Em seguida, o *scheduler* conecta-se ao *pipeline* de execução e atribui as funções para os nós que residem mais próximo dos dados.

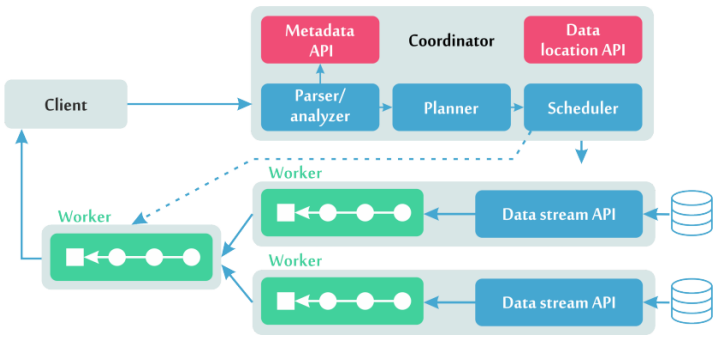


Figura 13: Arquitetura Presto (Adaptado de Sethi et al., 2019).

O Presto é extensível para que qualquer variedade de armazenamento possa ser conectada, no entanto, requer um conector que forneça os metadados ao *Presto*, informações sobre quais nós armazenam os dados e uma maneira de realmente ter acesso aos dados como um fluxo. Os *plug-ins* de armazenamento suportados pelo presto são o *HDFS*, *Hive, HBase e Scribe*.

2.8.8 *Apache Superset*

O *Apache Superset* foi criado como um projeto para o *AirBnB*, tendo como objetivo a criação de um *software* totalmente personalizável para visualizar e explorar grandes quantidades de dados rapidamente e intuitivamente (Fallucchi et al., 2018).

As funcionalidades do *Apache Superset* podem dividir-se em:

- ***Visualização de dados*:** A técnica de representação visual de forma a que seja possível informar os utilizadores, geralmente de maneira compreensível. A visualização de dados pode ser utilizada para diferentes propósitos, mas geralmente destina-se a fornecer informações sobre grandes números ou outros pontos de dados.
- ***Exploração de dados*:** É o processo de examinar os dados de várias perspectivas. É uma maneira de entender o conteúdo de maneiras novas e criativas. A exploração de dados também é conhecida como análise exploratória de dados.
- ***Análise de dados*:** É um método de extrair informações de dados coletados de várias medições e observações para definir padrões, verificar conclusões, fazer previsões e decidir como alocar recursos. Este ajuda a examinar vários padrões e o desempenho, podendo fornecer auxílio a realizar julgamentos baseados em tendências.

O *Apache Superset* pode ser executado no modo sequencial e distribuído. No modo sequencial, este só pode executar consultas abaixo de 60 segundos. No modo distribuído, o superconjunto espalha as consultas entre os *workers*. O *Superset* pode utilizar *RabbitMQ* ou *Redis* para distribuir as tarefas e utiliza o *Redis* para consultas de *cache*.

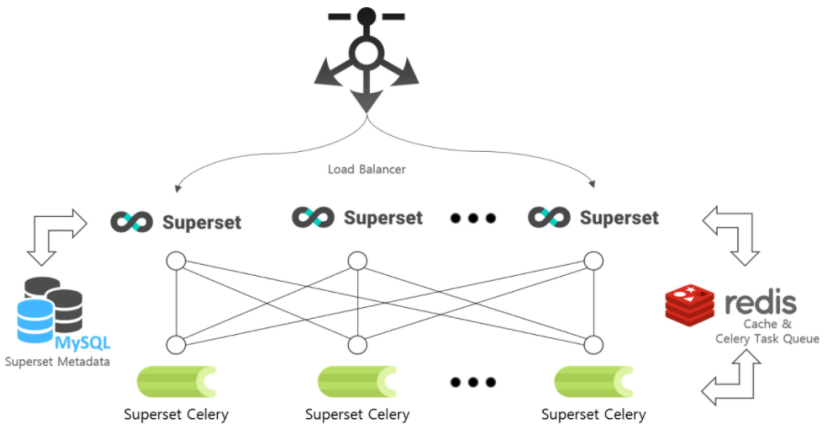


Figura 14: Arquitetura Apache Superset (Adaptado de Fallucchi et al., 2018).

2.9 Tecnologia de *Containers*

Os *containers* são blocos de virtualização ao nível do sistema operativo, permitindo a criação de um ambiente virtual isolado sem a necessidade de supervisores. Estes são executados de forma independente, mas o sistema operativo subjacente é compartilhado por eles. Além de fornecer um ambiente virtual isolado, os *containers* permitem uma instalação simplificada de *software*, incluindo todas as dependências necessárias das ferramentas fornecidas (Merkel et al., 2014).

Uma questão importante é a portabilidade das tecnologias de uma máquina para a outra. Ao desenvolver uma infraestrutura complexa, geralmente existe uma combinação de muitos *softwares* e bibliotecas que necessitam de ser instalados no sistema para este ser executado. Antes de executar a infraestrutura numa máquina diferente, existe a necessidade dos *softwares* dependentes serem instalados. Sendo assim, existe a necessidade de se poder transitar aplicações de um sistema para o outro de forma eficiente, surgindo então o *Docker*, introduzido por *Merkel* (Merkel et al., 2014). O *Docker* é uma plataforma de código aberto para desenvolvimento, envio e execução de aplicações, permitindo que os utilizadores separem as aplicações da infraestrutura subjacente. Dessa forma, a mesma aplicação normalmente pode ser executada em todas as máquinas que partilham do mesmo mecanismo *Docker*. Os principais componentes por trás do *Docker* são os *containers* e as imagens *Docker*.

Um *container* do *Docker* fornece a capacidade de empacotar e executar uma aplicação num ambiente isolado. Isto significa que muitos *containers* com diferentes aplicações podem ser executados no mesmo *host*. Os *containers* são leves e não dependem do sistema operacional da máquina *host*, nem do *software* instalado. Estes contêm tudo o que é necessário para ser possível a execução de aplicações, sendo que, um *container* do *Docker* constitui uma instância executável de uma imagem do *Docker*. Os *containers* podem ser associados como máquinas virtuais leves que hospedam aplicações. A figura 15 demonstra a diferença entre *containers* e máquinas virtuais, enquanto que as máquinas virtuais exigem o seu próprio sistema operacional, o *Docker* não necessita do mesmo, tornado os *containers Docker* muito menos volumosos.

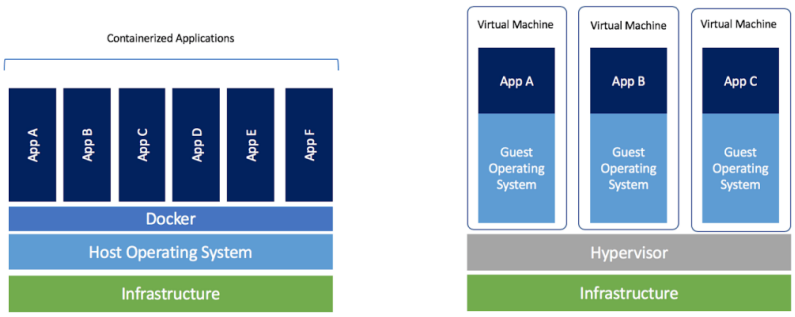


Figura 15: Containers vs. Virtual Machines (Adaptado de Joy, 2015).

Uma imagem *Docker* contém instruções de como criar um *container Docker*. Muitas vezes, a imagem é uma extensão de outra imagem. As imagens *Docker* são bastante poderosas, especialmente quando as atualizações de *software* são introduzidas. Por exemplo, num cenário em que uma aplicação seja

executado por mais de um ano e o *back-end* é um servidor *PostgreSQL*. Após um ano de produção, é necessário atualizar o servidor para uma versão atual, essa alteração pode ser realizada com muita facilidade quando o servidor *PostgreSQL* é executado no *Docker*, pois a única alteração necessária está relacionada à imagem. Mais especificamente, um novo *container* deve ser lançado pela imagem recém criada, desta forma os utilizadores experimentarão tempos praticamente nulos de inatividade das aplicações, uma vez que as mudanças entre *containers* podem ser executadas em questão de minutos.

3

Caso de Estudo: Plataforma de Big Data Analytics para desenvolvimento do produto em contexto industrial

A transformação digital é uma realidade que tem provocado efeitos transformadores na sociedade, a necessidade de criação de valor tem aumentado a complexidade inerente ao desenvolvimento de produto, com implicações no processo produtivo e nas cadeias de fornecimento, ou seja, com repercussões em todo o ecossistema industrial. Os atuais ambientes de produção são pressionados para obter uma variedade crescente de produtos personalizados de alta qualidade em lotes muito flexíveis.

Neste contexto, o projeto mobilizador *Produtech* pretende dar resposta a estes desafios através da capacidade para lidar e tornar úteis no desenvolvimento de produto a elevada quantidade de dados *(Big Data)* gerados pelos próprios produtos, acrescida da capacidade de processamento e tratamento local desses dados, e simultaneamente tendo a capacidade de fornecer às equipas de desenvolvimento de produto a informação necessária para os seus sistemas de suporte à decisão.

Nesta indústria, as máquinas existentes no sistema produtivo deverão ter implementados sensores capazes de extrair informação importante das mesmas, como é o caso de temperaturas, vibrações, ruído, entre outras. Assim, pretende-se a implementação de uma plataforma de *Big Data Analytics* capaz de gerir e analisar diferentes tipos de dados e de interoperar entre diferentes produtos, durante o uso, pretendendo dar informações às equipas de desenvolvimento de produto, para que estas possam desenvolver produtos customizados, atualizar produtos ou criar variantes de produtos.

Para este caso de estudo, foram utilizados dados sensoriais de um cliente do projeto mobilizador. Estes dados não estão associados a um elevado volume que permite efetuar um *benchmarking* da plataforma desenvolvida, de modo a medir o seu desempenho para processar um elevado conjunto de dados (espera-se que tal esforço seja realizado em trabalho futuro). Ainda assim, considera-se que os dados obtidos permitem realizar uma prova de conceito, demonstrando um correto funcionamento de diversos elementos que constituem a plataforma.

3.1 Introdução

Nos sistemas atuais, a elevada quantidade de dados disponibilizados pode ser utilizada para permitir que os sistemas implementados nos clientes expressem necessidades que dificilmente seriam captadas por formas tradicionais de análise de mercado. Ao adquirir essas informações, os desenvolvedores po-

dem captar oportunidades para desenvolver produtos com um maior foco nas necessidades dos seus clientes. Este caso de estudo centra-se na oportunidade de usar dados pós-implementação, ou seja, dados que são gerados enquanto os sistemas produtivos são utilizados, e se encontram no seu ambiente natural de funcionamento, e que servirão de base para introduzir melhorias no produto ou conduzir ao desenvolvimento de novos produtos. Os sistemas produtivos integram cada vez mais sensores e funcionalidades controladas por *software*, o que cria um potencial crescente de geração de dados que podem ser recolhidos. A enorme coleção de dados *(Big Data)* faz com que seja humanamente impossível a procura de padrões relevantes e úteis. Daí que atualmente o foco passe pela adoção de abordagens de *Data Mining* e *Big Data Analytics* para automatizar esta extração de conhecimento.

O equipamento a ser utilizado neste caso de estudo, pertence a um dos clientes do projeto mobilizador *Produtech*, tratando-se de um equipamento de testes para vários produtos eletrónicos composto por um módulo de interface para os componentes eletrónicos e sistema de eixos para movimento. Este equipamento está representado na figura 16.

Figura 16: Equipamento de testes utilizado.

Este equipamento foi sensorizado em certos componentes e foi possível obter dados relativos a 4 dias da máquina em funcionamento (367398 registos), relativamente aos seguintes parâmetros: corrente, frequência, voltagem, potência ativa, potência aparente e potência real. Apesar de não haver por parte do cliente, um volume de dados suficiente e não haver a sensorização completa do produto para que se consigam obter os devidos *outputs* desejados, foram utilizados os dados extraídos até ao momento para ser realizada uma prova de conceito e comprovar que a plataforma é funcional de ponta a ponta.

Este tipo de prova de conceito é muito importante, uma vez que, caso seja demonstrado que a mesma é funcional de ponta a ponta, pode futuramente ser alimentada com o volume de dados adequado e irá ter a capacidade de transformar os dados brutos em informação útil para as equipas de desenvolvimento para que se possam obter atualizações e melhorias de produto ou até novos produtos customizados.

3.2 Arquitetura

Nos últimos anos, o desafio de lidar com um grande volume de dados tem sido assumido por muitas empresas, levando a uma mudança completa de paradigma no que se refere ao armazenamento, processamento e visualização dos dados. Surgiram várias novas tecnologias, cada uma visando aspectos específicos do processamento de dados distribuído em grande escala. Todas essas tecnologias, como sistemas de computação em lote e bases de dados não estruturadas, podem lidar com volumes de dados muito grandes com pouco tempo, mas com sérios compromissos, como alta latência.

Para a conceção desta plataforma era imperativo, por um lado, a existência de visualizações em tempo real para que no *shop-floor* se pudesse visualizar os parâmetros sensorizados em tempo real e agir conforme necessário. Por outro lado, a necessidade de haver visualizações de dados históricos, assim como a descoberta de padrões nesses dados através da utilização de técnicas de *Machine Learning*, para que as equipas de desenvolvimento pudessem realizar alterações a nível do produto.

Por esse motivo, foi decidido seguir o modelo de arquitetura *Lambda* (Warren and Marz, 2015), que é identificada por três componentes principais:

- ***Batch Layer*:** Esta camada é responsável pelo armazenamento de grandes conjuntos de dados em constante crescimento, fornecendo a capacidade de computação de funções arbitrárias nos mesmos.

- ***Speed Layer*:** Esta camada é responsável pelo processamento de fluxos de dados em tempo real. Esta camada sacrifica a taxa de transferência, pois visa minimizar a latência fornecendo visualizações em tempo real dos dados mais recentes. Essencialmente, a camada de velocidade é responsável por preencher a lacuna causada pelo atraso da *batch layer* em fornecer visualizações com base nos dados mais recentes. As visualizações dessa camada podem não ser tão precisas ou completas quanto as eventualmente produzidas pela *batch layer*.

- ***Serving Layer*:** Esta camada é responsável pelo armazenamento de dados para serem consumidos pelos serviços de processamento.

A arquitetura da plataforma está representada na figura 17, que possui como principais componentes a *input layer*, a *batch layer*, a *speed layer*, a *serving layer*, a *output layer* e as tecnologias de monitorização do sistema. Entre estes componentes são estabelecidos relacionamentos, de acordo com os dados que fluem entre eles.

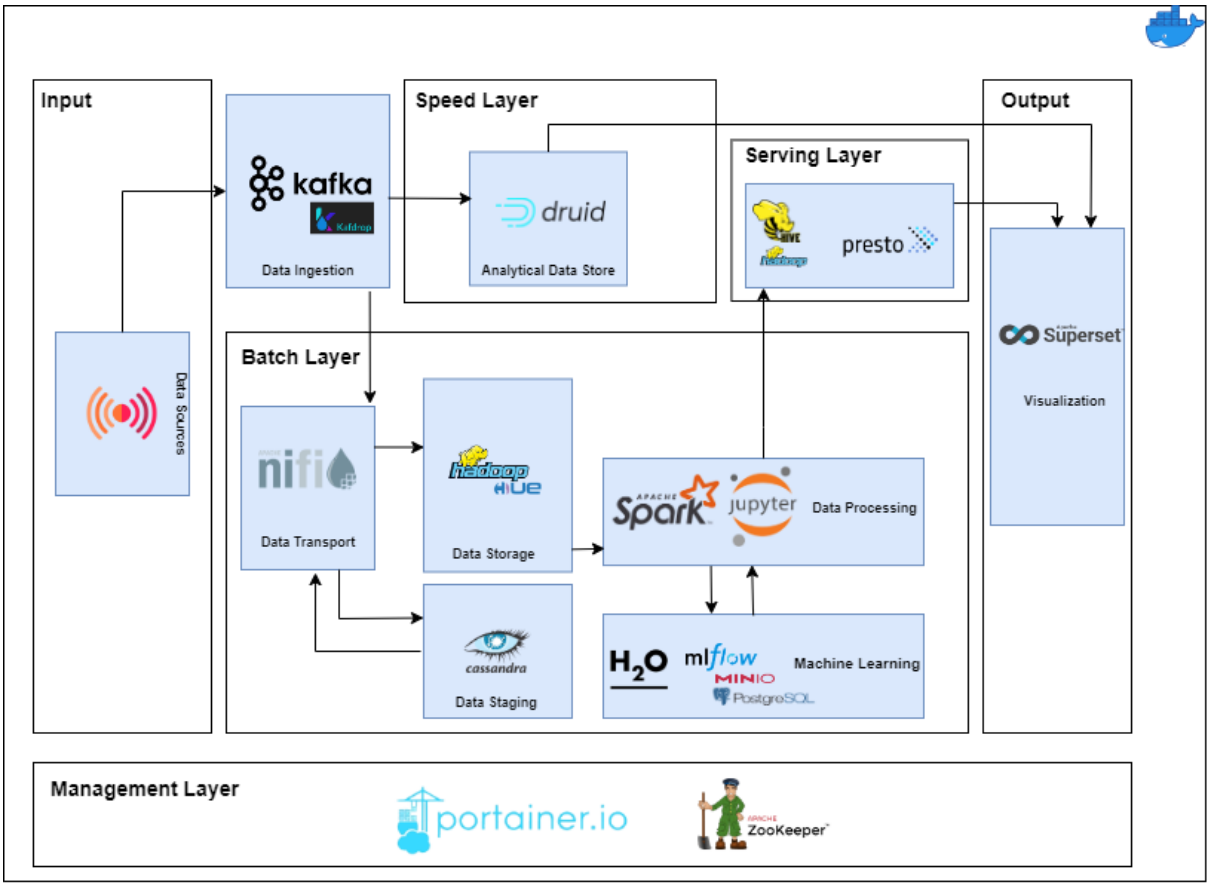


Figura 17: Arquitetura de Big Data e Machine Learning para desenvolvimento de produto

Na arquitetura apresentada é possível verificar que o *broker* do *Kafka* encaminha os dados para alimentar o fluxo *streaming* e o fluxo *batch*. Continuando na explicação do fluxo *streaming*, os eventos que vão sendo gerados pelo *producer* e encaminhados pelo *broker* passam depois por um processo no *Apache Druid* de transformação em dados tabulares. No fluxo *batch*, é utilizado o *Apache NiFi* para realizar a transformação e transporte dos dados que se encontravam em formato *raw* para o *Cassandra*, sendo esta uma área de *staging*. Quando existe um volume de dados definido, os dados são armazenados no *HDFS*. Estes dados podem ainda passar por um processo de tratamento e enriquecimento, através do *Spark*. O componente do *Spark*, no final, armazena os dados novamente no *HDFS* e os metadados num *hive metastore*. É ainda utilizado o *Presto* no fluxo *batch*, que se trata de um *query engine* capaz de realizar consultas analíticas rápidas a dados de qualquer tamanho.

Por fim, como componente de visualização de dados foi utilizado o *Superset*, principalmente por ser possível a sua integração com o *Druid* e com o *Hive*. É importante realçar que esta plataforma foi construída de forma a que facilmente se possam adicionar ou modificar tecnologias de acordo com casos de uso específicos.

Para poder ser possível conceber a plataforma, foi necessária a utilização da tecnologia *Docker*, abordado na secção 2.9. A utilização desta tecnologia permitiu que fosse criada uma virtualização ao nível do sistema operacional para que seja possível entregar *software* em pacotes chamado *containers*. A

utilização desta tecnologia irá permitir também que esta plataforma seja implementada em qualquer servidor *Linux* que poderá estar presente em cada um dos clientes do projeto mobilizador, uma vez que foi concebido para permitir a flexibilidade e portabilidade no que diz respeito à execução. Sendo assim, para a implementação da plataforma foi utilizado o sistema operativo *Linux x86 Canonical Ubuntu 21.04 LTS* e a versão 19.04 do *Docker*, sendo que foram utilizados vários *Docker Containers*. Na tabela 2 está representado o endereço *IP* de cada *container*, o *container port* e o *host port*.

Tabela 2: Endereço IP e portas para cada container.

Container Name	Endereço IP	Host Port	Container Port
Apache NiFi	172.16.131.118	9090	9090
HDFS	172.16.131.118	9870	9870
Jupyter Notebook	172.16.131.118	4040	4040
Apache Cassandra	172.16.131.118	9042	9042
Kafdrop	172.16.131.119	19000	19000
Apache Kafka	172.16.131.119	9092	9092
Apache Druid	172.16.131.119	8081	8081
Hue	172.16.131.118	8888	8000
Apache Hive	172.16.131.118	908	9083
Apache Presto	172.16.131.118	8081	8081
Apache Spark	172.16.131.118	7077	7078
Apache Zookeeper	172.16.131.118	22181	22181
Portainer	172.16.131.118	443	9443
Apache Superset	172.16.131.118	8088	8079
H2O	172.16.131.118	54321	54321

As respetivas camadas da plataforma, assim como as tecnologias utilizadas, serão abordadas nas próximas secções.

3.2.1 *Input Layer*

Relativamente à camada de *input*, esta pode ser descrita como a fonte de dados, que representa o local inicial onde os dados nascem ou onde as informações físicas são digitalizadas pela primeira vez. Sendo assim, a camada de *input* pode ser resumido como uma coleção de registos onde se encontra a informação necessária para o processo de negócio.

Neste caso de estudo, como referido anteriormente os dados têm origem na sensorização da máquina, representada na figura 16. Após isto, é realizada uma comunicação com a plataforma implementada para que seja possível realizar a ingestão destes dados.

No que diz respeito aos dados, foi possível realizar a sensorização dos seguintes parâmetros:

- ***Motor 1*:** voltagem, corrente, potência aparente, potência real, potência ativa e temperatura;
- ***Motor 2*:** voltagem, corrente, potência aparente, potência real, potência ativa e temperatura;

- ***Motor 3*:** voltagem, corrente, potência aparente, potência real, potência ativa e temperatura;
- ***Temperaturas*:** Ambiente e máquina.

3.2.2 *Ingestão de Dados*

A ingestão de dados pode ser definido como o processo de absorção de dados de uma variedade de fontes de dados e a transferência para um local de destino onde podem ser depositados e analisados.

Esta ingestão de dados pode ser realizada de diferentes formas:

- ***Ingestão em tempo real*:** A ingestão de dados em tempo real, também conhecida como *streaming data*, é útil quando os dados coletados são extremamente sensíveis ao tempo. Os dados são extraídos, processados e armazenados assim que são gerados para a tomada de decisões em tempo real.
- ***Ingestão de lote*:** Quando a ingestão ocorre em lotes, os dados são movidos em intervalos agendados de forma recorrente. Esta abordagem é benéfica para processos repetitivos.

Neste tipo de arquitetura utilizada, arquitetura *Lambda*, esta equilibra a vantagem dos dois métodos mencionados anteriormente, uma vez que utiliza ambos os métodos para fornecer diferentes tipos de visualizações ou informações dependendo do caso de uso em questão.

Para este propósito, foi utilizada a ferramenta *Kafka*, que se trata de uma plataforma *open source* de processamento de *streams* desenvolvida pela *Apache Software Foundation*, que tem como objetivo fornecer uma plataforma de alta capacidade e baixa latência para tratamento de dados em tempo real, permitindo assim alimentar a camada de *batch* e a camada de *streaming*.

Para a utilização desta tecnologia foi criado um *Kafka cluster*, representado na listagem 3.1, utilizando a tecnologia *Docker*, este *cluster* foi implementado numa máquina diferente das restantes tecnologias da plataforma, por questões de segurança, uma vez que é neste *cluster* que é feita a comunicação com servidores externos, tendo sido também necessário a utilização do serviço *zookeeper* uma vez que este serve como um controlador centralizado para gerir todas as informações de metadados relativamente aos produtores, *brokers* e consumidores do *Kafka*. Foi ainda utilizado um serviço denominado *Kafdrop* que se trata de uma interface *web* para a visualização dos tópicos do *Kafka*.

```
version: '3'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    networks:
      - broker-kafka
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
```

```
  kafka:
    image: confluentinc/cp-kafka:latest
    networks:
      - broker-kafka
    depends_on:
      - zookeeper
    ports:
      - 9092:9092
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT_HOST
          ://172.16.131.212:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,
          PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1

  kafdrop:
    image: obsidiandynamics/kafdrop:latest
    networks:
      - broker-kafka
    depends_on:
      - kafka
    ports:
      - 19000:9000
    environment:
      KAFKA_BROKERCONNECT: kafka:29092

networks:
  broker-kafka:
    driver: bridge
```

Listagem 3.1: *Docker Compose Kafka Cluster*.

Após a configuração correta do *cluster*, foi possível definir o *host:193.136.14.119* e criado o tópico *EquipamentoDados* para que os dados seguissem o fluxo descrito anteriormente.

Para que seja possível que os dados fluam pela plataforma, existem três atores principais que fazem com que o ato do envio e receção de dados aconteça:

- ***Produtor*:** Quem produz e envia mensagens para uma ou mais *queues* presentes na arquitetura do *Kafka*, que neste caso trata-se dos dados enviados por parte do cliente através de um protocolo de comunicação *TLS*.

- ***Queue*:** Uma estrutura de dados de *buffer*, que recebe (dos produtores) e entrega mensagens (aos consumidores) na forma *FIFO*. Quando uma mensagem é entregue ao consumidor, esta é removida da *queue*, não havendo oportunidade de a recuperar.

- ***Consumidor*:** Quem está inscrito em uma ou mais *queues*, responsável por receber as mensagens quando estas são publicadas. No caso específico desta plataforma, podem ser considerados consumidores as tecnologias *Apache NiFi* e *Apache Druid*.

3.2.3 *Batch Layer*

No centro da Arquitetura *Lambda* está o conjunto de dados mestre. O conjunto de dados mestre é a fonte da verdade na Arquitetura *Lambda*. Mesmo que se perca todos os conjuntos de dados da *serving layer* e todos os conjuntos de dados da *speed layer*, é possível reconstruir todo o processo a partir do conjunto de dados mestre. Isso ocorre porque as visualizações *batch* atendidas pela *serving layer* são produzidas por meio de funções no conjunto de dados mestre e, como a *speed layer* é baseada apenas em dados recentes, esta pode ser construída em pouco tempo.

O conjunto de dados mestre é a única parte da Arquitetura *Lambda* que absolutamente deve ser protegida contra corrupção. Máquinas sobrecarregadas, discos com falha e quedas de energia podem causar erros, e o erro humano com sistemas de dados dinâmicos é um risco intrínseco e uma eventualidade inevitável. É necessário que seja projetado cuidadosamente o conjunto de dados mestre para evitar corrupção em todos esses casos, pois a tolerância a falhas é essencial para a integridade de um sistema de dados de longa duração.

Em primeiro lugar, é necessário que os dados no formato *raw* que estão a ser enviados para o *Kafka* sejam consumidos e transportados para sistemas dentro da plataforma para que sejam armazenados no formato correto e de forma segura. Por esse motivo, foi utilizado o *Apache NiFi* que é uma ferramenta que visa a automatização do fluxo de dados entre sistemas de *software*.

O fluxo representado na figura 18 trata-se do fluxo do *Apache NiFi* em que existem processadores que ficam encarregues do consumo de dados provenientes do *Apache Kafka* e a realização de transformações dos dados *raw* num formato que possam ser armazenados no *Apache Cassandra*.

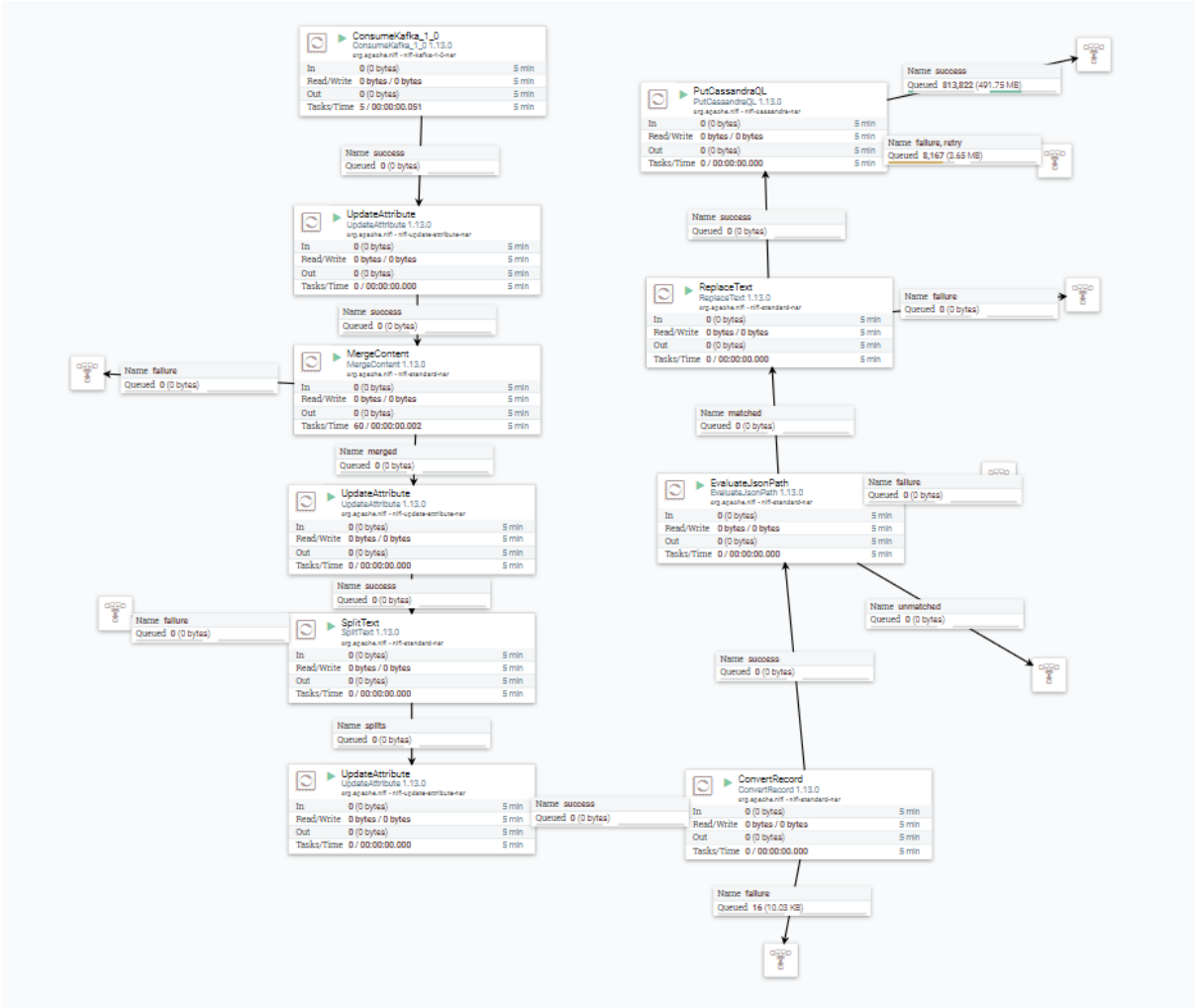


Figura 18: Fluxo Apache NiFi

O primeiro processador presente neste fluxo denomina-se *ConsumeKafka*, que consiste no consumo das mensagens presente no tópico *Kafka EquipamentoDados*, no qual foi necessário efetuar a configuração de algumas propriedades para que fosse possível o consumo das mesmas. Depois de efetuado este processador, as mensagens estão a ser consumidas em formato *raw*, sendo necessário efetuar algumas transformações para que possam seguir o resto do fluxo.

Sendo assim, os dados foram convertidos em formato *json* para que sejam armazenados em pares chave/valor, visto ser uma maneira valiosa de se poder armazenar grandes quantidades de dados para que estes possam ser organizados e classificados facilmente. Para este propósito foi utilizado um processador denominado *ConvertRecord* que converte registos de um formato de dados para outro utilizando um *RecordReader* e *RecordWrite* com a devida configuração. Para haver sucesso nesta conversão é necessário a configuração do esquema de entrada e do esquema de saída, tendo em atenção que devem ser configurados com esquemas correspondentes. A listagem 3.2 apresenta parte do esquema de saída, constituído pelos pares de chave/valor.

```
{
    "type": "record",
    "name": "UserRecord",
```

```
    "fields" : [
        {"name": "timestamp", "type": "int"},
    {"name": "time1", "type": "int"},
        {"name": "voltage1", "type": "int"},
        {"name": "current1", "type": "int"},
        {"name": "activepower1", "type": "int"},
        {"name": "reactivepower1", "type": "int"},
        {"name": "aparentpower1", "type": "int"},
        {"name": "frequency1", "type": "int"},
        {"name": "powerfactor1", "type": "int"},

    ...

        {"name": "aparentpower3", "type": "int"},
        {"name": "frequency3", "type": "int"},
        {"name": "powerfactor3", "type": "int"},
        {"name": "timertemp", "type": "int"},
        {"name": "temperature1", "type": "float"},
        {"name": "temperature2", "type": "float"}
    ]
}
```

Listagem 3.2: *Esquema de saída ConvertRecord*.

Após estas conversões, foi utilizado o *Apache Cassandra* como uma área de *staging*. Esta pode ser entendida como uma área de armazenamento temporário entre as fontes de dados e o *data storage*, sendo que, é um local onde os dados podem ser guardados de forma segura até ser atingido um volume de dados suficiente para serem armazenados no *HDFS*.

Com este intuito, foi criada uma tabela no *Apache Cassandra*, no *keyspace produtechequipamento*. A *query* para a criação dessa tabela está representada na listagem 3.3.

```
CREATE TABLE equipamentodados (timestamp int,time1 int,voltage1 int,
    current1 int,activepower1 int,reactivepower1 int,aparentpower1 int,
    frequency1 int,powerfactor1 int,time2 int,voltage2 int,current2 int,
    activepower2 int,reactivepower2 int,aparentpower2 int,frequency2 int,
    powerfactor2 int,time3 int,voltage3 int,current3 int,activepower3 int,
    reactivepower3 int,aparentpower3 int,frequency3 int,powerfactor3 int,
    timertemp int,temperature1 float,temperature2 float,temperature3 float,
    temperature4 float,temperature5 float,temperature6 float);
```

Listagem 3.3: *Criação de tabela Apache Cassandra*.

Após a criação da tabela, foi necessário a criação de processadores no *Apache NiFi* para que os dados sejam enviados para o *Apache Cassandra*. Foi necessário definir algumas propriedades de configuração como o *Cassandra Contact Points: 172.16.131.118:9042* que se trata do endereço do nó do *Cassandra*,

o *keyspace: produtechequipamento* e a *query*, representada na listagem 3.4, para que os dados sejam inseridos na tabela correta.

```
INSERT INTO equipamentodados (timestamp,time1,voltage1,current1,
    activepower1,reactivepower1,aparentpower1,frequency1,powerfactor1,time2,
    voltage2,current2,activepower2,reactivepower2,aparentpower2,frequency2,
    powerfactor2,time3,voltage3,current3,activepower3,reactivepower3,
    aparentpower3,frequency3,powerfactor3,timertemp,temperature1,
    temperature2,temperature3,temperature4,temperature5,temperature6) VALUES
     (${'timestamp'},${'time1'}, ${'voltage1'}, ${'current1'}, ${'
    activepower1'}, ${'reactivepower1'}, ${'aparentpower1'}, ${'frequency1'
    }, ${'powerfactor1'}, ${'time2'}, ${'voltage2'}, ${'current2'}, ${'
    activepower2'}, ${'reactivepower2'}, ${'aparentpower2'}, ${'frequency2'
    }, ${'powerfactor2'}, ${'time3'}, ${'voltage3'}, ${'current3'}, ${'
    activepower3'}, ${'reactivepower3'}, ${'aparentpower3'}, ${'frequency3'
    }, ${'powerfactor3'}, ${'timertemp'}, ${'temperature1'}, ${'temperature2
    '}, ${'temperature3'}, ${'temperature4'}, ${'temperature5'}, ${'
    temperature6'});
```

Listagem 3.4: *Query para inserção de dados na tabela Cassandra.*

Após estes passos, já existem dados a serem armazenados no *Apache Cassandra*, no formato pretendido. É então necessário garantir que os dados sejam enviados para o *data storage*, *HDFS*, que é um sistema de arquivos distribuído que foi desenvolvido para detetar falhas e recuperar automaticamente, fornecendo acesso a dados com altas taxas de transferência e é adequado para grandes conjuntos de dados. Para garantir este transporte de dados entre *softwares* foi utilizado processadores do *Apache NiFi*, estando o processo representado na figura 19.

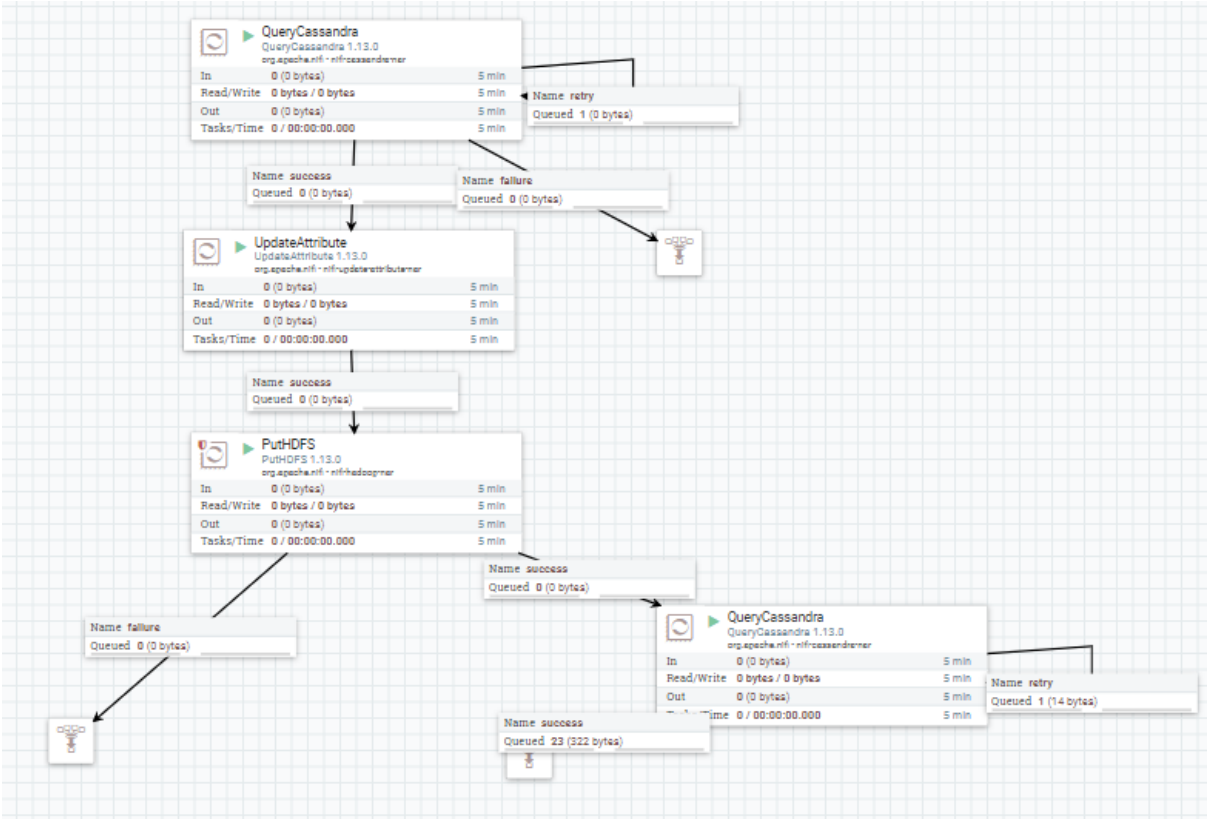


Figura 19: Fluxo Apache NiFi

Este processo é iniciado com a execução de uma *query* que seleciona todos os dados da tabela do *Apache Cassandra* e armazena os dados no *HDFS*. Após isso, é executada uma *query* que elimina todos os dados existentes na tabela do *Apache Cassandra* para que sejam apenas considerados novos dados na próxima execução. Este processo é realizado todos os dias às 16:05, tendo como auxílio uma expressão *CRON* que permite que permite o agendamento de tarefas repetidamente em um horário específico. A expressão *CRON* pode ser representada por $(015161/1 * ?*)$. No caso do *HDFS*, este processo permite que seja criado um ficheiro diariamente no sistema de ficheiros com os respetivos dados, no horário especificado anteriormente (ver figura 20).

☐	-rw-r--r--	nifi	supergroup	83.93 MB	Apr 12 16:05	3	128 MB	controlardados_20220412150515
☐	-rw-r--r--	nifi	supergroup	74.57 MB	Apr 13 16:05	3	128 MB	controlardados_20220413150516

Figura 20: Sistema de ficheiros HDFS com os ficheiros especificados

Estando os dados armazenados no *data storage*, a próxima etapa seria o processamento e enriquecimento dos dados, assim como a utilização de algoritmos de *Machine Learning* para alimentar as visualizações *batch*.

Com este intuito, foi implementado um *Spark Cluster* que se trata de um mecanismo rápido para processamento de dados em larga escala, sendo mais eficiente do que abordagens anteriores como o caso do *MapReduce*, devendo-se ao facto de que o *Spark* é executado em memória, tornando o processo

muito mais rápido. Foi também utilizado o *Jupyter Notebook* que se trata de uma plataforma *web* de computação que permite também a conexão ao *Spark Cluster*.

Para interagir com o *Spark* é necessário a criação de uma *Spark Session* que é o ponto de entrada unificado de um aplicativo *Spark*, fornecendo uma maneira de interagir com várias funcionalidades do *Spark* com um menor numero de configurações, estando o mesmo representado na listagem 3.5.

```
"spark = SparkSession,
    "     .builder,
    "     .appName(\"Equipamento\")",
    "     .config(\"spark.sql.warehouse.dir\", warehouse_location)",
    "     .config(\"hive.metastore.uris\", \"thrift://hive-metastore:9083\")
        ",
    "     .master(\"spark://172.16.131.140:7077\")",
    "     .enableHiveSupport()",
    "     .getOrCreate()"
```

Listagem 3.5: Código *Spark Session*.

Depois de iniciada a *SparkSession* conseguimos então ter acesso aos dados armazenados no *HDFS*. Após uma análise aos dados, foi necessário realizar algumas transformações nos mesmos, representada na listagem 3.6, como é o caso da definição de valores mínimos, médios e máximos por hora de cada parâmetro, uma vez que os dados estavam a ser armazenados a cada segundo, não sendo relevante para o caso de estudo uma análise com essa granularidade.

```
df_equipamento = spark.read.json('hdfs://namenode:8020//produtech/
    equipamento/dados/5663a9f3-b9dd-4852-82f9-cf7903acd0d1', multiLine =
    True)

dfequipamentohour = df_equipamento.withColumn("timestamp") \
              .withColumn("timestamp_hour", substring("timestamp", 11, 13))
                  \
               .orderBy("timestamp")

dfequipamentoagg = dfequipamentohour.groupBy("timestamp_hour") \
      .agg( \
      min("activepower1").alias('min_activepower1'), \
      max("activepower1").alias('max_activepower1'), \
      round(avg("activepower1"), 2).alias('avg_activepower1'), \
      min("reactivepower1").alias('min_reactivepower1'), \
      max("reactivepower1").alias('max_reactivepower1'), \
      round(avg("reactivepower1"), 2).alias('avg_reactivepower1'), \
```

Listagem 3.6: *Código de leitura dos dados armazenados no HDFS e a transformação em valores mínimos, médios e máximos por cada hora*.

De seguida, foi realizada a transformação dos dados em formato *Parquet*, uma vez que este formato tem maior velocidade de execução em comparação com outros formatos de arquivos padrão como é o caso do formato *Avro* e do formato *Json* e também consome menos espaço em disco. A criação da tabela em formato *Parquet* e posterior armazenamento no *HDFS* está representado na listagem 3.7.

```
spark.sql(
    """
    CREATE TABLE produtech.equipamento (
        timestamp TIMESTAMP,
        min_activepower1 INT,
        max_activepower1 INT,
        avg_acrtivepower1 DOUBLE,
        min_reactivepower1 INT,
        max_reactivepower1 INT,
        avg_reactivepower1 DOUBLE,

        ...

    )
    STORED AS PARQUET
    PARTITIONED BY (
        temp_count_hour INT
    )
    LOCATION 'hdfs://namenode:8020/warehouse/produtech.equipamento/
       produtech/'
    """
)
```

Listagem 3.7: *Exemplo de código de criação de tabela Parquet e armazenamento no HDFS.*

Neste caso de estudo, não foi possível abordar as tecnologias de *Machine Learning*, uma vez que não havia volume de dados suficiente para que fosse possível retirar informação útil para o desenvolvimento do produto e para as técnicas de *Machine Learning*. No entanto, estas tecnologias serão abordadas na secção 4 com a utilização de dados que não provenientes do projeto mas que permitem a realização de uma prova de conceito.

Tendo acabado o fluxo da *batch layer* foram utilizadas diferentes tecnologias com o intuito de armazenar e gerir grandes quantidades de dados, permitindo que os dados possam ser facilmente acedidos por diferentes serviços e permitindo que o dimensionamento deste sistema seja flexível conforme o necessário, tendo sido projetado para que se possa aumentar os recursos sempre que seja necessário.

3.2.4 *Serving Layer*

A *serving layer* pode ser descrita como o último componente da *batch layer* da Arquitetura *Lambda*. Está fortemente vinculado à *batch layer*, uma vez que esta é responsável por atualizar continuamente as visualizações da *serving layer*. Apesar de estas visualizações estarem sempre desatualizadas devido à natureza de alta latência da computação *batch*, isso não é uma preocupação, pois a *speed layer* é responsável por quaisquer dados que ainda não estão disponíveis na *serving layer*.

Para esta camada foi utilizado o *Hive Metastore* como um repositório central de todos os metadados. Este armazena todos os metadados de tabelas *Hive* como o *schema*, a localização e as partições. No caso da plataforma, as tabelas *Hive* são armazenadas no *HDFS*. Como é possível de verificar na figura 21 com os processos descritos anteriormente existe um ficheiro no *HDFS* correspondente à tabela *Hive* no formato *Parquet*.

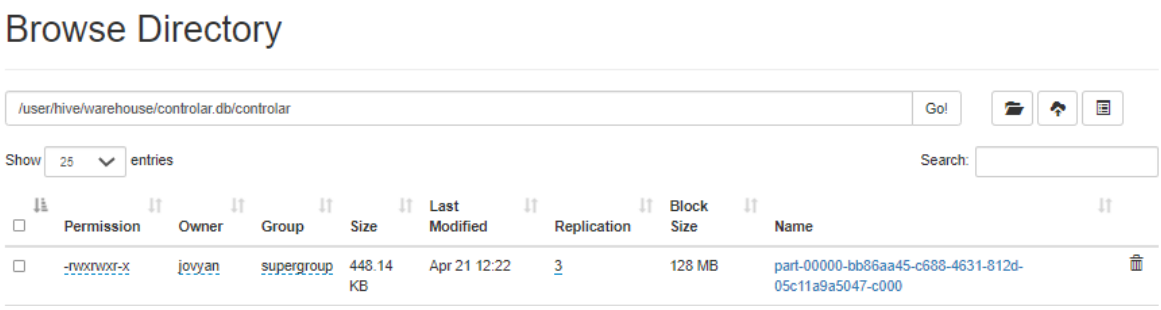


Figura 21: Tabela Hive armazenada no HDFS

Para esta camada, foi também utilizada a tecnologia *Apache Presto*, que se trata de uma ferramenta projetada para consultar com eficiência grandes quantidades de dados utilizando consultas distribuídas. Pode ainda ser estendido para operar em diferentes tipos de fontes de dados, incluindo bancos de dados relacionais e outras fontes de dados, como o *Apache Cassandra*. Foi possível realizar a conexão ao *HDFS* através de um conector *Hive*, utilizado para realizar consultas aos dados armazenados no *HDFS*.

Ao ser projetada esta camada, foi necessário ter em atenção duas métricas de desempenho principais: taxa de transferência e latência. Nesse contexto, a latência é o tempo necessário para responder a uma única consulta, enquanto a taxa de transferência é o número de consultas que podem ser atendidas em um determinado período de tempo. Apesar de não ser possível efetuar uma avaliação destas métricas no que diz respeito ao futuro caso de uso, devido a volumes de dados não representativos da realidade, foi possível testar que é possível a execução de consultas simultaneamente que podem ser atendidas em curtos períodos de tempo como representado na figura 22.

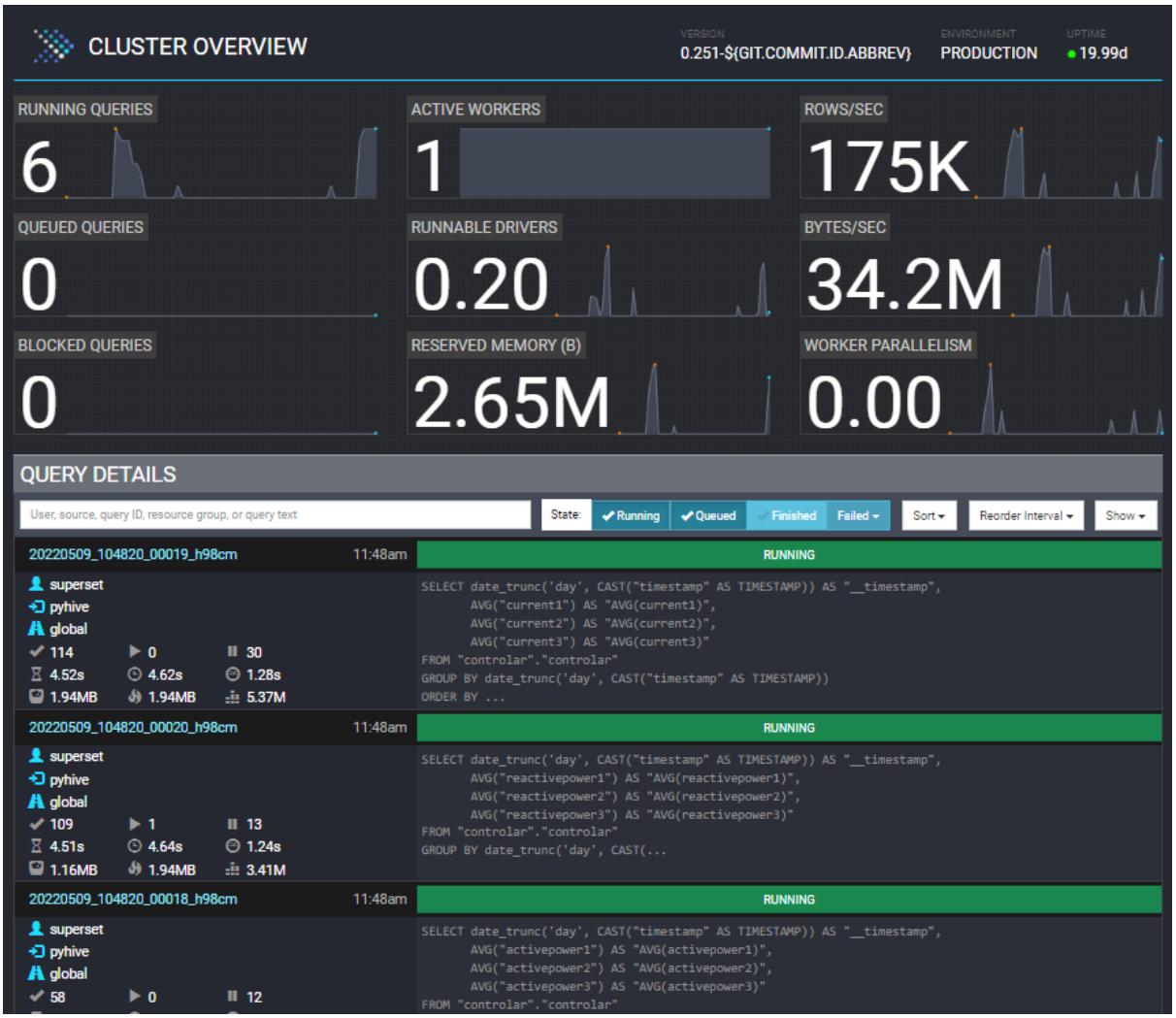


Figura 22: Interface gráfica Presto

3.2.5 *Speed Layer*

Esta camada lida com os dados que ainda não foram entregues na *batch layer* devido à latência da mesma. Além disso, este lida apenas com dados recentes para fornecer uma visão completa dos dados ao utilizador, criando visualizações em tempo real. Com este propósito, foi utilizado a tecnologia *Apache Druid*, sendo um banco de dados de análise em tempo real que potencializa os casos de uso em que a ingestão em tempo real, o desempenho rápido de consulta e o alto tempo de atividade são importantes.

Com o propósito de poder consumir os dados presentes no tópico *Kafka* foi necessário definir configurações no *Apache Druid* em que é definido o *broker* a ser acedido, o tópico do qual se pretende consumir os dados e os respetivos campos e valores a serem consumidos, como é possível confirmar pelo seguinte fragmento de código:

```
{
  "type": "kafka",
  "spec": {
    "ioConfig": {
```

```
"type": "kafka",
"consumerProperties": {
  "bootstrap.servers": "172.16.131.212:9092"
},
"topic": "EquipamentoDados",
"inputFormat": {
  "type": "csv",
  "findColumnsFromHeader": false,
  "columns": [
    "timestamp",
    "time1",
    "voltage1",
 ...
    "aparentpower3",
    "frequency3",
    "powerfactor3",
    "timertemp"
    "temperature1"
    "temperature2"
    "temperature3"
},
```

Listagem 3.8: Código *Configurações Apache Druid*.

Sendo assim, o *Apache Druid* consome as mensagens em formato *raw* presentes no tópico *Kafka* e transforma as mesmas em dados tabulares para que possam ser consumidas pelo *software* de visualização e permitir que sejam realizadas visualizações em tempo real.

O *Apache Druid* foi inserido na plataforma, uma vez que, foi desenhado para alimentar cargas de trabalho analíticas em tempo real para dados orientados a eventos. Foi projetado para lidar com muitos casos de uso que os *data warehouse* tradicionais não conseguem. Para este caso de estudo em concreto, a utilização do *Apache Druid* vai permitir que, futuramente, se configurem no *shop-floor* dos clientes *dashboards* analíticas. Desta forma, poderá ser possível a visualização dos dados relativos à sensorização, em tempo real, sem ter que aguardar pelo processamento dos dados provenientes da *batch layer*.

3.2.6 *Output Layer*

Relativamente à camada de *output*, esta pode ser descrita como a camada de visualizações, que representa o local final dos dados. Para esta camada foi utilizada a ferramenta *Apache Superset*, por ser rápida, leve, intuitiva e repleta de opções que facilitam a exploração e visualização de dados pelos utilizadores, desde gráficos de linhas simples até gráficos geoespaciais altamente detalhados.

Para ser possível a visualização dos dados, tanto da *speed layer*, como da *serving layer* foi necessário proceder à configuração de conectores tanto para o *Apache Druid*, como para o *Apache Presto*, representados na listagem 3.9.

Para ser realizado a configuração dos conectores é necessário o preenchimento dos seguintes campos para cada conector:

***User*:** credenciais do utilizador para aceder à base de dados.

Password: palavra-passe de acesso à base de dados.

Host: endereço *IP* da máquina *host* onde é executada a base de dados.

Port: porta específica que está exposta na máquina *host* onde é executada a base de dados.

```
Configuração Apache Druid:

druid://<User>:<password>@<Host>:<Port-default-9088>/druid/v2/sql

druid://equipamentodados:equipamento123@193.136.14.119:9088/duid/vs/sql

Configuração Presto:

presto://{hostname}:{port}/{database}

presto://193.136.14.119:9086/prestodb
```

Listagem 3.9: *Configuração dos conectores Apache Superset.*

Com a configuração dos conectores realizada é possível aceder a ambas as fontes de dados através da interface gráfica do *Apache Superset*, representada na figura 23.

Database	Backend
PrestoBD	presto
Apache Druid	druid

Figura 23: Bases de dados Apache Druid e Apache Superset

Após as configurações é possível proceder à ultima etapa da plataforma que se trata da implementação de visualizações em tempo real e de visualizações *batch*. Na figura 24 está representado um exemplo de visualização *batch* com os dados processados em que se pretendeu realizar uma comparação entre os diferentes parâmetros relativamente aos 3 motores existentes, tendo como granularidade a hora. Na figura 25 está representado um exemplo de uma visualização em tempo real com os dados processados relativo ao parâmetro *Active Power*.

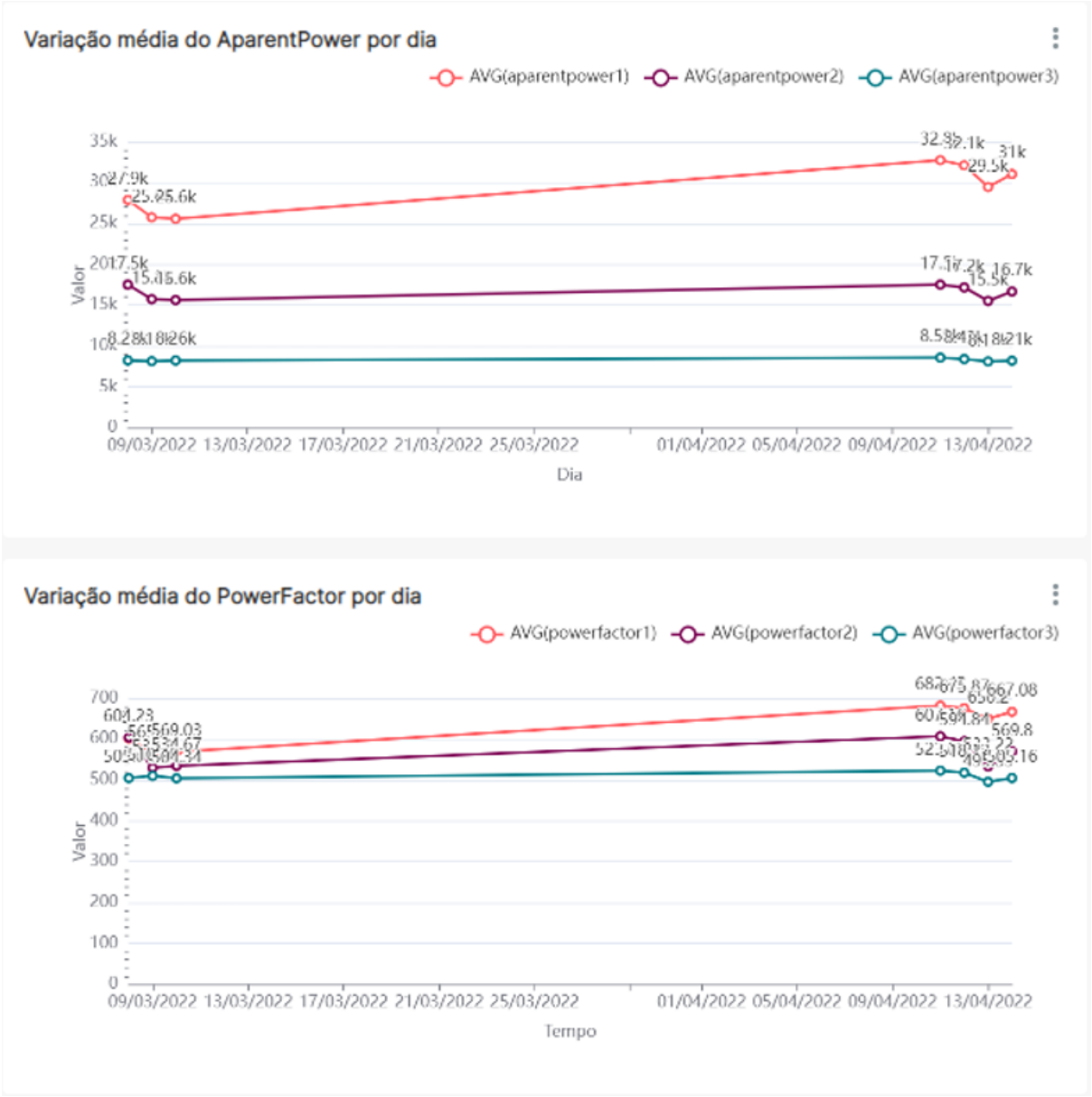


Figura 24: Visualização Apache Superset referente à Batch Layer.

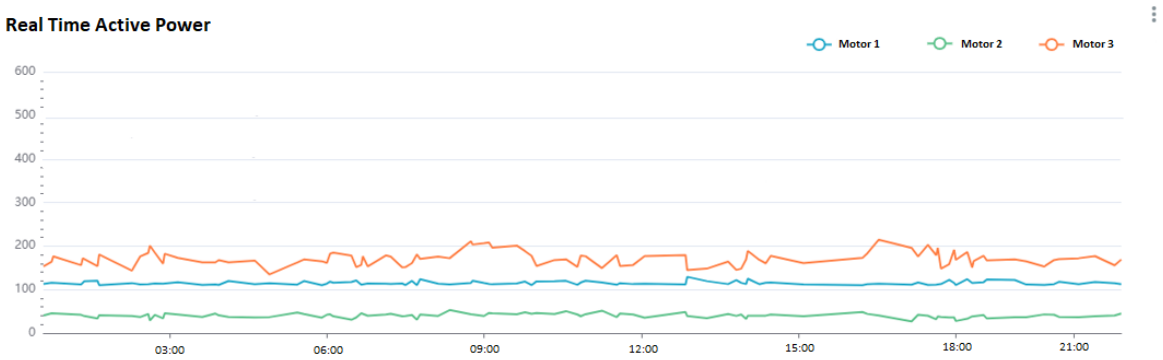


Figura 25: Visualização Apache Superset referente à Speed Layer.

A etapa de visualização de dados é considerada a etapa mais importante no que diz respeito ao utilizador final, uma vez que é aqui que as equipas de desenvolvimento do produto vão poder visualizar toda a informação útil relativa aos parâmetros sensorizados da máquina para que possam através desta informação atualizar, realizar modificações ou criar novos produtos costumizados. Por este motivo é imperativo que toda a informação necessária seja apresentada nesta etapa.

3.3 Análise de *performance*

Na implementação da plataforma, como referido anteriormente, é necessário que o conjunto de dados mestre seja totalmente protegido contra corrupção, uma vez que representa a fonte de verdade em plataformas em que é utilizada a arquitetura *Lambda*. Na arquitetura apresentada anteriormente (ver figura ??), as tecnologias como o *Apache Kafka* e o *Apache NiFi* são os principais intervenientes na produção deste conjunto de dados mestre e por esse motivo têm de ter a capacidade de lidar com mudanças rápidas no crescimento de dados. Nesse sentido, foram realizados alguns testes de forma a verificar de que forma a plataforma suporta o volume de dados atual e como efetivamente irá suportar o crescimento do volume de dados.

3.3.1 *Apache NiFi*

Relativamente ao fluxo presente no *Apache NiFi*, este é constituído por um conjunto de processadores onde é realizado o consumo de dados provenientes do *Apache Kafka*, é feita a conversão destas mensagens em formato *JSON* e por fim é realizada a inserção destes dados através de uma *query SQL* no *Apache Cassandra*. Foram então criados 4 casos de teste para medir de que forma um crescimento rápido no volume de dados consumidos poderia afetar a *performance* do *Apache NiFi*, tendo como base o atual consumo de dados (Teste 1):

- **Teste 1**: Consumo de 2 mensagens/seg das mensagens no tópico do *Apache Kafka* sobre um período de tempo de 5 minutos.
- **Teste 2**: Consumo de 20 mensagens/seg das mensagens no tópico do *Apache Kafka* sobre um período de tempo de 5 minutos.
- **Teste 3**: Consumo de 200 mensagens/seg das mensagens no tópico do *Apache Kafka* sobre um período de tempo de 5 minutos.
- **Teste 4**: Consumo de 2000 mensagens/seg das mensagens no tópico do *Apache Kafka* sobre um período de tempo de 5 minutos.

Estes casos de teste foram realizados aos processadores *Consume Kafka*, *Merge Content*, *Split Text*, *Convert Record* e *Insert Cassandra*, sendo estes os processadores principais presentes no fluxo do *Apache NiFi* e são responsáveis por:

- ***Consume Kafka*:** Este processador é responsável pelo consumo de mensagens do *Apache Kafka*.
- ***Merge Content*:** Este processador é responsável por unir as mensagens para que seja adicionado um *header* às mesmas.
- ***Split Text*:** Este processador é responsável por separar as mensagens para que cada mensagem tenha um par de chave-valor.
- ***Convert Record*:** Este processador é responsável por converter as mensagens para formato *JSON* através dos pares chave-valor criados anteriormente.
- ***Insert Cassandra*:** Este processador é responsável por inserir os pares de chave-valor no *Apache Cassandra*.

Na tabela 3 está representado, para cada caso de teste, o tempo de execução (mm:ss.ZZZ) de cada processador e o tempo total de execução dos mesmos que representa a soma dos tempos individuais. Está também representado o tempo de execução total do fluxo, ou seja, o tempo total para que todas as mensagens consumidas já tenham sido inseridas no *Apache Cassandra*.

Tabela 3: Tempo de execução dos processadores e do fluxo para cada caso

Processador	Teste 1	Teste 2	Teste 3	Teste 4
Consume Kafka	00:00.264	00:02.155	00:13.624	02:29.264
Merge Content	00:00.034	00:00:224	00:01.856	00:13.135
Split Text	00:00.074	00:00.345	00:03.425	00:17.343
Convert Record	00:00.038	00:00.201	00:05.798	00:22.160
Insert Cassandra	00:00.068	00:00.609	00:06.120	00:37.572
Tempo Total Processadores	00:00.498	00:03.544	00:32.523	03:45.536
Tempo Total Fluxo	05:00.045	05:02.300	05:05.700	08:58.075

Nas figuras 26 e 27 está representado, em milissegundos, o tempo de execução total dos processadores e o tempo de execução total do fluxo sobre um período de tempo de 5 minutos em que houve consumo de mensagens. Para os casos de teste 2 e 3, é possível verificar que apesar de ter havido um aumento no tempo de execução dos processadores (ver figura 26), não se reflete num aumento considerável no tempo de execução total do fluxo para os mesmos casos, sendo que há um aumento de apenas 0,4% e 1,6%, respetivamente, tendo como base o caso de estudo 1 (ver figura 27). Apenas para o caso de teste 4 é que se verifica que o aumento no tempo de execução dos processadores é refletido por um aumento considerável no tempo de execução total do fluxo, aumento de 78% comparativamente ao caso 1 e por esse motivo apenas neste caso de estudo é que existe efetivamente uma menor *performance* do *Apache NiFi*.

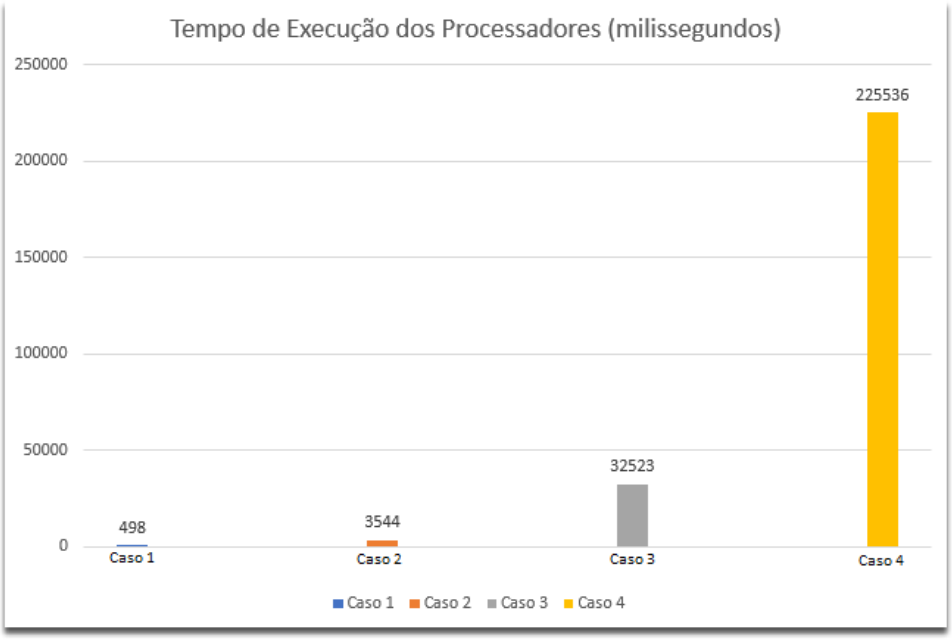


Figura 26: Tempo de execução dos processadores para diferentes casos no Apache NiFi

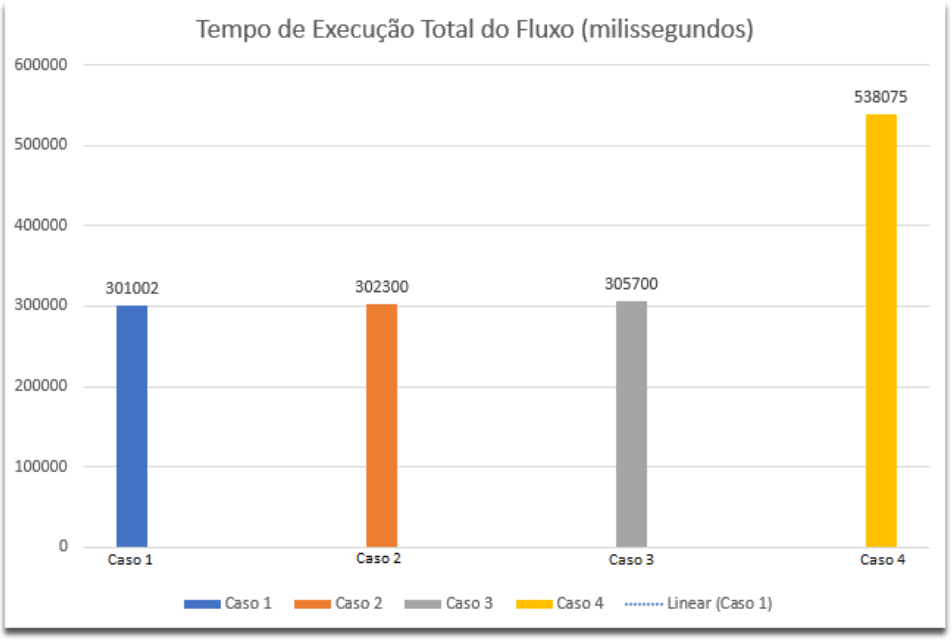


Figura 27: Tempo de execução total do fluxo para diferentes casos no Apache NiFi

A perda de *performance* evidenciada no caso de teste 4, deve-se ao facto de que o aumento do número de consumo de mensagens resultou num maior tempo para a execução de cada processador, o que fez com que para alguns processadores o tempo de execução seja maior do que o tempo de chegada de novas mensagens, o que levou à criação de filas de espera entre processadores (ver figura 28) e consequentemente ao aumento no tempo total do fluxo.

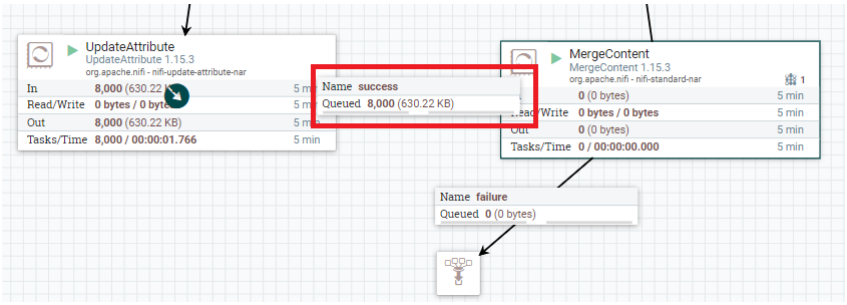


Figura 28: Fila de espera para o caso 4 no Apache NiFi

Foi realizado ainda a avaliação da utilização do *CPU* pelo *container* do *Apache NiFi* para os mesmos casos de teste. Esta avaliação foi realizada através do comando *docker stats ContainerID* e os resultados estão representados na figura 29. Para o caso de teste 4, a utilização do *CPU* é bastante superior aos restantes casos e pode implicar no futuro que o *container* do *Apache NiFi* utilize todos os recursos do *CPU* disponíveis, fazendo com que haja uma redução na velocidade de execução de processos de outros *containers* e até com que os mesmos deixem de responder, sendo que pode originar ainda a perda de dados.

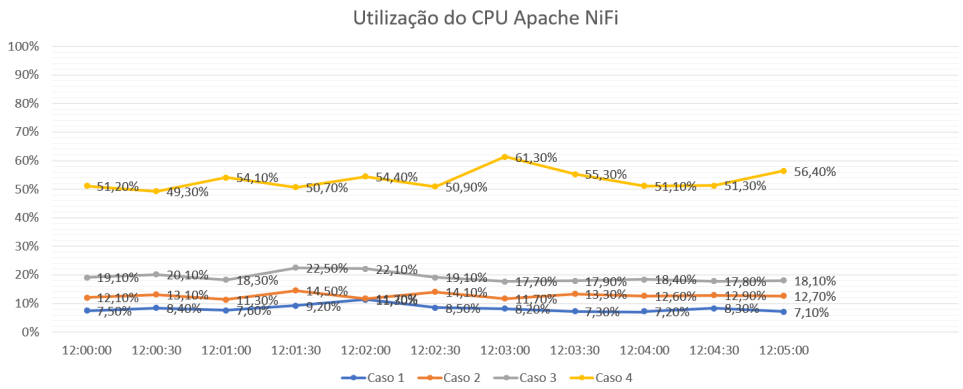


Figura 29: Utilização do CPU para diferentes casos no Apache NiFi

Com a realização destes casos de teste foi possível verificar que o container do *Apache NiFi* implementado na plataforma de *Big Data Analytics* consegue suportar um crescimento do volume de dados até 200 vezes superior ao volume atual de dados, como comprovado na figura 27, sem que isso represente um decréscimo de *performance* no fluxo existente e sem que a utilização do *CPU* represente um risco para os restantes *containers* da plataforma. Estes casos de teste permitiram também perceber que caso o crescimento do volume de dados eventualmente atinja os valores representados no caso de estudo 4, que representaria a sensorização de 1,000 máquinas em funcionamento em que cada uma estaria a enviar para o *Apache Kafka* 2 mensagens por segundo existe a necessidade de modificar o fluxo construído para conseguir lidar com este volume ou que seja adicionado mais *hardware* à plataforma.

3.3.2 *Apache Kafka*

Relativamente ao *Apache Kafka*, este é constituído por produtores que publicam registos/mensagens para um tópico e consumidores que subscrevem um ou mais tópicos *Kafka* (ver figura 30). Um tópico é uma fila de mensagens escritas por um ou mais produtores e lidas por um ou mais consumidores. Atualmente, na plataforma as mensagens são produzidas através da sensorização da máquina em estudo e o consumidor é o *Apache NiFi*.

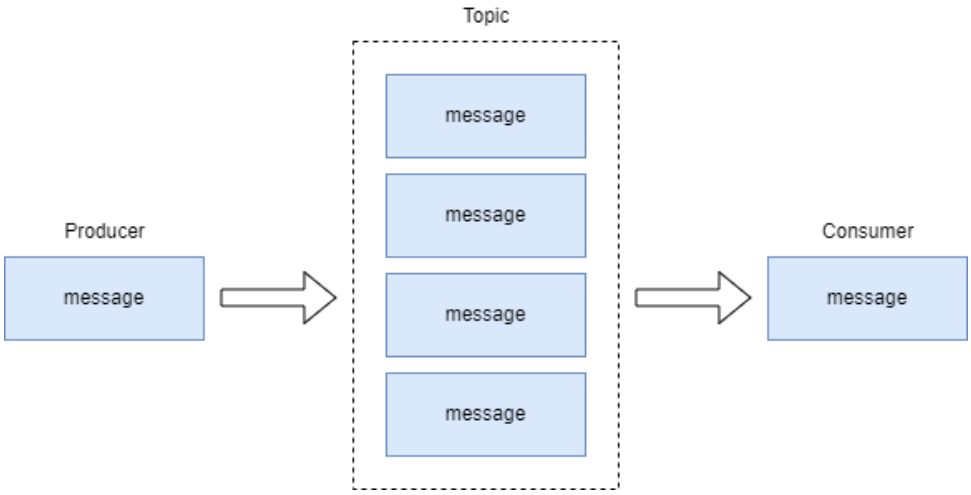


Figura 30: Fluxo Apache Kafka

Para o propósito de avaliar a *performance* do *Apache Kafka* foram realizados testes para simular diferentes tipos de carga, através de ferramentas fornecidas pelo *Apache Kafia* (*kafka-producer-perf-test e kafka-consumer-perf-test)*. A partir da utilização destas ferramentas é possível realizar testes de carga para o produtor e consumidor, para concluir quantas mensagens um produtor pode produzir e um consumidor pode consumir num determinado período de tempo.

Em primeiro lugar, foi necessário criar um tópico para realizar os vários testes. O tópico criado com o nome "test-topic"(ver Listagem 3.10) é composto por 6 partições, 1 replicação de fator e *bootstrap-server* que é composto pelo endereço *IP* e pela porta do *broker* do *Apache Kafka*.

```
$ docker exec -it e3b764b3d57f /bin/bash \
kafka-topics \
--create \
--partitions 6 \
--replication-factor 1 \
--topic test-topic \
--bootstrap-servers=172.16.131.119:29092
```

Listagem 3.10: *Criação do tópico Apache Kafka*.

Depois de efetuada a criação do tópico, foi necessária a criação de um produtor para poderem ser realizados os testes, sendo que foi necessário definir os seguintes parâmetros:

- ***Throughput*:** Medida de quantos eventos chegam ao *Apache Kafka* num período de tempo específico;
- ***Num-records*:** Número total de registos enviados;
- ***Record-Size*:** O tamanho de cada registo em *bytes*;
- ***Topic**:* Nome do tópico para onde são enviados os eventos;
- ***bootstrap-servers**:* A localização do *broker* do *Kafka*.

Para o primeiro teste efetuado replicou-se as configurações presentes atualmente na utilização da plataforma, em que chegam ao *Kafka* 2 eventos por segundo (*throughput*) e o tamanho de cada mensagem é de 6500*bytes*. Para esse propósito foi executado o comando representado na listagem 3.11.

```
$ docker exec -it e3b764b3d57f /bin/bash \
kafka-producer-perf-test \
--throughput 2
--num-records 1000
--topic test-topic
--record-size 6500
--producer-props bootstrap.servers=172.16.131.119:29092
```

Listagem 3.11: *Criação do produtor Apache Kafka*.

Os resultados da execução deste comando são apresentados na figura 31, sendo que os parâmetros necessários para a avaliação da *performance* são o número de registos por segundo e a latência, que representa o tempo que o *Apache Kafka* demora a processar um único evento. Pelos resultados apresentados, podemos verificar que são enviados 2 registos por segundo devido ao facto de termos definido um *troughput* de 2 e uma latência máxima de 2.0*ms*.

```
10 records sent, 2.0 records/sec (0.01 MB/sec), 1.4 ms avg latency, 2.0 ms max latency.
10 records sent, 2.0 records/sec (0.01 MB/sec), 1.8 ms avg latency, 2.0 ms max latency.
10 records sent, 2.0 records/sec (0.01 MB/sec), 1.6 ms avg latency, 2.0 ms max latency.
10 records sent, 2.0 records/sec (0.01 MB/sec), 1.7 ms avg latency, 2.0 ms max latency.
10 records sent, 2.0 records/sec (0.01 MB/sec), 1.5 ms avg latency, 2.0 ms max latency.
10 records sent, 2.0 records/sec (0.01 MB/sec), 1.4 ms avg latency, 2.0 ms max latency.
10 records sent, 2.0 records/sec (0.01 MB/sec), 1.3 ms avg latency, 2.0 ms max latency.
10 records sent, 2.0 records/sec (0.01 MB/sec), 1.6 ms avg latency, 2.0 ms max latency.
```

Figura 31: Produtor com throughput 2

Os resultados apresentados anteriormente servirão para avaliar de que forma uma modificação no *throughput*, medida de quantos eventos chegam ao *Apache Kafka* por segundo, afetará os restantes parâmetros e consequentemente a *performance do Apache Kafka*. Os testes são representados por:

- ***T1*:** 2 de *Throughput*;
- ***T2*:** 20 de *Throughput*;

- ***T3*:** 200 de *Throughput*;
- ***T4*:** 2,000 de *Throughput*;
- ***T5*:** 20,000 de *Throughput*;
- ***T6*:** 200,000 de *Throughput*;

Foi executado o mesmo comando apresentado anteriormente para os diferentes testes, alterando o *throughput* e os resultados foram os seguintes:

Tabela 4: Tempo de execução dos processadores e do fluxo para cada caso

	T1	T2	T3	T4	T5	T6
Throughput	2	20	200	2,000	20,000	200,000
Registos/seg	2	20	200	2,000	20,000	22,000
Mb/seg	0.1	0.12	1.20	12.4	124.01	135
Latência Média(ms)	1.4	1.5	1.5	1.9	3.4	10.7
Latência Máxima(ms)	2	4	6	12	19	157

Através dos resultados apresentados na tabela 5, podem ser tecidas algumas considerações relativas à *performance* do *Apache Kafka* e à sua capacidade para lidar com o crescimento do volume de dados:

- O *Apache Kafka* é capaz de lidar com um volume de dados, relativamente ao produtor, 10,000 vezes superior ao atual sem que isso comprometa a sua *performance*. Como é possível verificar através do teste 5, apesar de ter sido definido um *throughput* de 20,000 que resultou no envio de 20,000 registos/segundo pelo produtor do *Kafka*, este não implicou um aumento anormal da latência média e da latência máxima (ver figura 32), o que significa que o *cluster Kafka* é capaz de lidar com um número de eventos consideravelmente mais alto sem que seja necessário efetuar modificações ou optimizações ao *cluster* implementado.
- Relativamente ao teste 6, é possível verificar que apesar de ter sido definido um *throughput* de 200,000, o produtor do *Apache Kafka* apenas conseguiu enviar em média 22,000 registos por segundo e também existiu um aumento anormal da latência máxima comparativamente aos outros testes efetuados (ver figura 32). Sendo assim, é necessário que o *cluster Kafka* seja otimizado ou que sejam efetuadas mudanças a nível de *hardware* se o volume de crescimento de dados alcançar o valor aqui representado, que significaria uma sensorização de 100,000 máquinas em que cada uma enviaria 2 eventos por segundo.

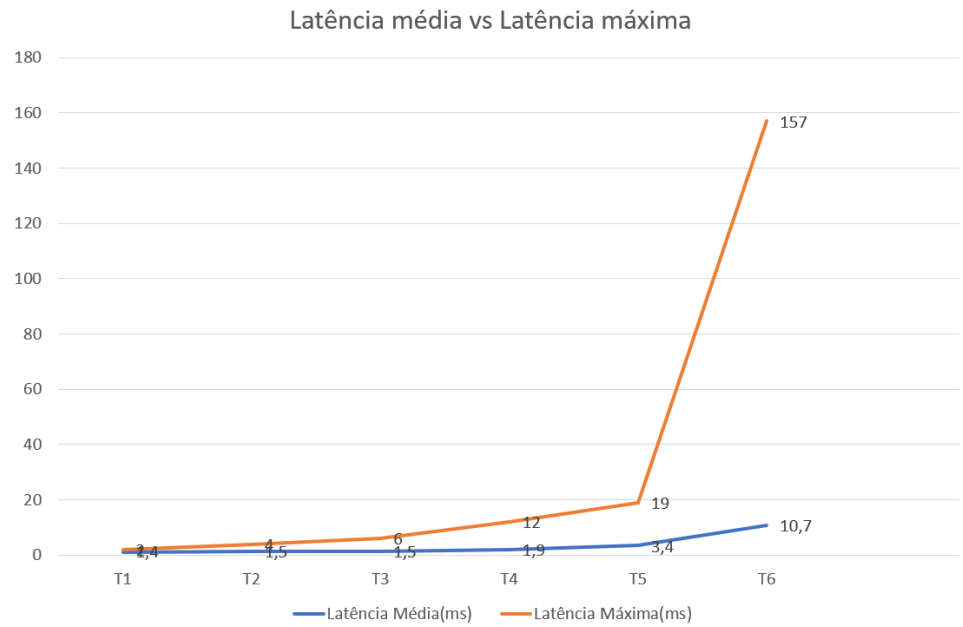


Figura 32: Latência média e Latência máxima para os diferentes testes do produtor

Em suma, podemos afirmar que o *Apache Kafka* implementado na plataforma consegue lidar, sem que seja afetada a performance um crescimento do volume de dados até 10,000 vezes superior ao atual, uma vez que atualmente estão a ser produzidos 2 eventos/seg o que representaria a sensorização de 10,000 máquinas se o número de eventos produzido por cada uma se mantivesse igual à máquina em estudo atualmente.

Realizado os testes ao produtor e tecidas as considerações foi necessário realizar alguns testes relativamente ao consumidor. Para esse propósito foi criado um consumidor que irá consumir 10,000,000 de mensagens, cada uma com um tamanho de 650 bytes, para poder ser analisado quanto tempo o *Apache Kafka* demoraria para as mensagens serem consumidas do tópico, tendo sido executado o seguinte comando:

```
$ docker exec -it e3b764b3d57f /bin/bash \
kafka-consumer-perf-test \
--messages 10000000 \
--timeout 1000000 \
--topic demo-topic \
--reporting-interval 1000 \
--show-detailed-stats \
--bootstrap-server 172.16.131.119:29092
```

Listagem 3.12: *Criação do consumidor Apache Kafka.*

Com a execução deste comando foi possível obter o resultado representado na figura 33.

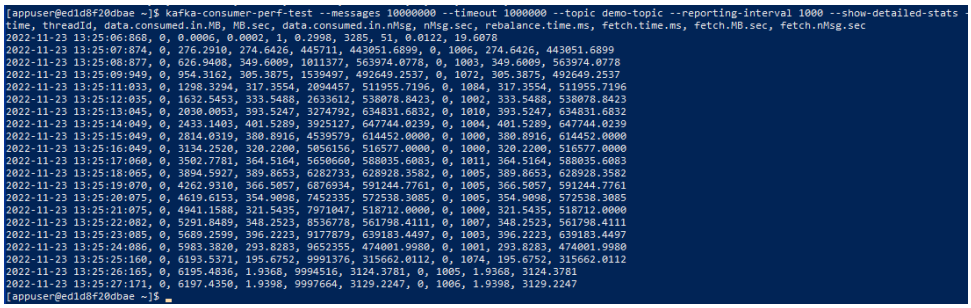


Figura 33: Resultados da execução do comando para o consumidor

Realizada uma filtragem aos resultados obtidos podemos obter os seguintes resultados:

Tabela 5: Resultados do teste realizado ao consumidor

Tempo	Dados consumidos (Mb)	Mensagens Consumidas
13:25:06	274	443,051
13:25:07	349	563,974
13:25:08	305	492,649
13:25:09	317	511,955
13:25:10	333	538,078
13:25:11	393	634,831
13:25:12	401	647,744
13:25:13	380	614,452
13:25:14	320	516,577
13:25:15	464	588,035
13:25:16	389	628,928
Média dos resultados/seg	357	561,843

Com os resultados apresentados é possível verificar que o *cluster Kafka* implementado na plaforma permite que sejam consumidas em média 561,843 mensagens por segundo, sendo equivalente a 357*Mb* de dados por segundo. Atualmente estão apenas a ser consumidas a partir do *Apache NiFi* 2 mensagens por segundo que corresponde ao número de mensagens que está atualmente a ser produzido pela sensorização da máquina. Sendo assim, é possível afirmar que o *Apache Kafka* permite que haja um crescimento de dados ao nível do consumidor, sem que isso afete a performance do mesmo, uma vez que este aguenta com cargas bastante superiores à existente.

3.4 Sumário

Esta secção marca o final do atual capítulo e, como tal, são aqui tecidas algumas considerações sobre os diferentes aspetos abrangidos e respetivos resultados alcançados. Este caso de estudo apresentou uma instanciação da camada tecnológica da arquitetura de *Big Data Analytics*, através do armazenamento, processamento e visualização de dados provenientes de uma máquina em funcionamento em

ambiente industrial, tendo sido possível obter dados relativos a 4 dias de funcionamento da máquina e que compreendem um total de 367398 registos.

Para ser possível a realização deste caso de estudo, foi utilizado o *DSRM* como abordagem metodológica (ver secção 1.3.1). Por esse motivo, o trabalho aqui desenvolvido teve inicio na identificação do problema, motivação e definição de objetivos. Após estas etapas, foi realizado uma revisão do estado da arte, uma vez que permite um enquadramento a nível de conceitos e a nível tecnológico para que seja possível realizar a etapa de conceção, desenvolvimento e de demonstração presente neste caso de estudo, em que foi apresentada e implementada a plataforma de *Big Data Analytics*, tendo em conta todas as tecnologias utilizadas para esse propósito e onde foi utilizado um fluxo de dados proveniente de uma máquina em funcionamento em ambiente industrial. Foi também realizado uma avaliação a nível da *performance* do *Apache Kafka* e do *Apache NiFi* com um eventual crescimento do volume de dados, provando que a plataforma é capaz de lidar com volumes de dados bastante superiores aos atuais.

A figura 34 demonstra a arquitetura conceptual concebida para a plataforma desenvolvida. Tendo em conta este caso de estudo, foi aqui demonstrado a capacidade da plataforma de lidar com as seguintes etapas:

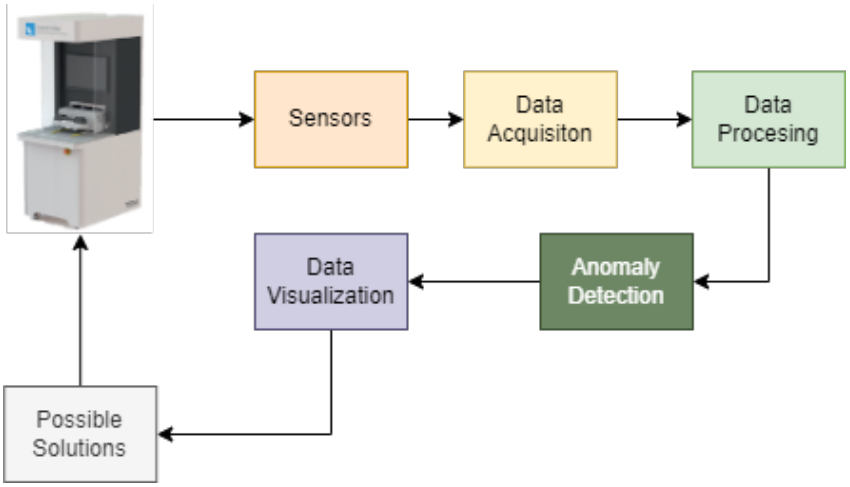


Figura 34: Arquitetura Conceptual Plataforma Big Data Analytics.

- ***Aquisição de dados*:** A etapa de aquisição de dados, abordada na secção 3.2.2, foi garantida através da utilização do *Apache Kafka* que foi responsável pelo consumo de dados provenientes da sensorização da máquina, permitindo que os dados sejam transportados para a plataforma e seguirem assim o resto do fluxo.

- ***Processamento de dados*:** A etapa de processamento de dados, abordada nas secções 3.2.3 e 3.2.5, pode ser definida como um tratamento sistemático de dados, com o objetivo de ordenar, classificar ou efetuar quaisquer transformações nos dados, visando a obtenção de um determinado resultado. Esta etapa foi efetuada através de vários processos e tecnologias, como é o caso dos processos utilizados através do *Apache NiFi* e do *Apache Spark* que permitiram a transformação dos dados brutos em informação útil para as equipas de desenvolvimento.

- ***Deteção de anomalias*:** Relativamente a esta etapa, uma vez que não existiu até ao momento qualquer registo de falhas ou anomalias, foi realizado um caso de estudo acional e relativo a algoritmos de deteção de anomalias na secção, , sendo mesmo apresentado no capítulo 4 . O objetivo é futuramente integrar na plataforma os modelos desenvolvidos de *Machine Learning*.

- ***Visualização de dados*:** Na etapa de visualização de dados, abordado na secção 3.2.6, foi utilizado o *Apache Superset* que possibilitou a realização de visualizações *batch* e visualizações *streaming*.

4

Caso de Estudo: Algoritmos de deteção de anomalias para conjuntos de dados extremamente desequilibrados

O caso de estudo que irá ser apresentado neste capítulo foi realizado com o intuito de ser integrado futuramente na plataforma, uma vez que a mesma já possui as tecnologias necessárias para esse propósito. No entanto, não foi possível incluir os algoritmos de deteção de anomalias no caso de estudo do capítulo 3, uma vez que até ao momento não existe volume de dados para esse propósito. No entanto, para o objetivo proposto, a capacidade de deteção de anomalias é um tópico bastante importante para as equipas de desenvolvimento do produto e por esse motivo foi abordado este caso de estudo para auxiliar a integração na plataforma quando existir volume e qualidade de dados.

A deteção de anomalias é uma tarefa importante de *Machine Learning* que afeta vários domínios (por exemplo, Finanças, Fraude, Indústria, Segurança). Com este propósito, os primeiros estudos sobre esta tarefa datam da década de 1960 (Grubbs, 1969). Recentemente, uma gama diversificada de algoritmos foi proposta para detecção de anomalias, incluindo algoritmos baseado em estatística (Hodge and Austin, 2004), *clustering* (Chandola et al., 2009; Ahmed et al., 2016), classificação (Chandola et al., 2009; Rai et al., 2016; Muharemi et al., 2019; Ribeiro et al., 2021; Matos et al., 2021) e mineração de gráficos (Akoglu et al., 2015). Em particular, as abordagens de classificação supervisionada exigem dados rotulados que muitas vezes são difíceis de obter (por exemplo, exigindo esforço humano). Quando os dados rotulados estão disponíveis, geralmente são extremamente desequilibrados, uma vez que eventos anómalos tendem a ser raros. Assim, alguns estudos de aprendizagem supervisionada utilizam métodos de balanceamento, como o *SMOTE* (Chawla et al., 2002) ou *GC* (Pereira et al., 2021).

Um dos desafios que a deteção de anomalias tem de resolver é que os limites entre dados normais e anormais geralmente não são claramente definidos, são normalmente abordados utilizando métodos de aprendizagem não supervisionados ou métodos de uma classe (Cao et al., 2018; Ruff et al., 2018) . Sob a abordagem de aprendizagem de uma classe, os conjuntos de dados de treino contêm apenas exemplos ”normais”. A suposição é que qualquer anomalia deve estar mais distante do espaço de aprendizagem de treino. Exemplos de algoritmos de *ML* de classe única incluem (Breunig et al., 2000; Alla and Adari, 2019): *LOF*, *OC-SVM* e *IF*.

Mais recentemente, *Deep Learning* tem sido proposto para deteção de anomalias em diversas aplicações (Chalapathy and Chawla, 2019; Pang et al., 2021). Um modelo popular de *Deep Learning* é o *AutoEncoder* (*AE*), que quando comparado com outros métodos de uma classe (por exemplo, *LOF*, *OC-*

SVM e *IF*), tende a fornecer tempos de treino mais rápidos, portanto, são capazes de lidar com uma quantidade maior de dados de treino.

Na detecção de anomalias, há um problema recorrente, que é a escassez de dados anómalos quando aplicados a uma abordagem supervisionada. Essa escassez torna bastante desafiador modelar modelos *Machine Learning*, pois não há muitos exemplos para alimentar o modelo. Em alguns estudos, técnicas de balanceamento são utilizadas para gerar exemplos minoritários suficientes para que modelos mais robustos aprendam (*oversampling*) ou até mesmo reduzam a classe maioritária para que o modelo aprenda as duas classes na mesma proporção (*Undersampling*) (Matos et al., 2018).

Uma vez que nenhum dos parceiros industriais realizou a sensorização antecipada das máquinas industriais a serem analisadas, foi realizado um caso de estudo que se enquadra nas necessidades e objetivos desta solução, garantindo que os dados utilizados para este efeito seguem a mesma lógica dos dados que irão ser sensorizados pelos parceiros industriais. Pretendendo-se assim a validação de algoritmos de deteção de anomalias, foi utilizado um *dataset* que é constituído por dados de sensores de telemetria instalados em várias máquinas num ambiente produtivo.

Para esta demonstração, foi também utilizada a metodologia *CRISP-DM*. Para melhorar a compreensão das necessidades do presente trabalho prático, foi seguida a estrutura das seguintes fases da metodologia *CRISP-DM*: compreensão do negócio, compreensão dos dados, preparação dos dados, modelação, avaliação e implementação.

4.1 Compreensão do Negócio

Após a Revolução Industrial, o processo "manual" da indústria de manufatura foi ficando cada vez mais automatizado, dando o lugar para máquinas que fazem o mesmo trabalho, potencialmente sem erros e com maior velocidade. Passou-se da utilização de máquinas a vapor, para a utilização de soluções elétricas, automação e para a utilização de Inteligência artificial e *Big Data*.

A área da indústria, especialmente, explora melhorias nos seus processos, com melhor aproveitamento e controlo, não só com máquinas melhores, mas com tecnologias que criam um ambiente estratégico de melhorias. Estamos, neste momento, na era da Indústria 4.0, em que a tecnologia pode ser uma aliada para qualquer tipo de empresa. A agilidade dos processos torna-se vital para ser possível competir com o mercado, além de outras exigências que também necessitam de ser atendidas, como qualidade da entrega, sustentabilidade, responsabilidade social, segurança do trabalho, entre outros.

A transformação digital no setor de manufatura começa com a implementação de novas tecnologias e soluções mais bem pensadas para a realidade atual, para que seja possível realizar promessas de maior flexibilidade na fabricação, juntamente com personalização em massa, melhor qualidade e produtividade aprimorada.

Com estas transformações a nível da indústria, surge o conceito de *Smart Manufacturing* que tende a alavancar diferentes tecnologias capacitadoras para alcançar altos níveis de produtividade, transformando organizações tradicionais em modelos de negócios inteligentes e hiperconectados. Além disso, a cres-

cente concorrência, a necessidade de reduzir o tempo de colocação no mercado e aumentar a inovação, exige que a organização adapte o seu comportamento para responder a este contexto de mudança.

Num ambiente de manufatura em que se pretenda atingir estes propósitos é necessário recorrer à telemetria. A telemetria é constituída pela medição e transmissão automática de dados de fontes remotas. No geral, a telemetria proporciona o acesso eletrónico a dados de máquinas existentes em ambiente de produção através da transmissão dos mesmos em tempo real para que se possa monitorizar os mesmos remotamente.

Embora a tecnologia telemétrica seja relativamente recente, a indústria está a ser alterada como um todo em direção à análise telemétrica orientada por dados e à tomada de decisões. A tecnologia telemétrica certamente desempenhará um papel significativo nesta tendência e um papel ainda mais importante na indústria da manufatura.

4.2 Compreensão dos Dados

De forma a demonstrar a aplicabilidade da componente de algoritmia em dados de telemetria, o presente estudo foca na utilização de algoritmos de deteção de anomalias para conjuntos de dados extremamente desequilibrados. Uma vez que não foi possível obter um registo histórico de dados suficiente dos parceiros industriais para se efetuar esta componente, foi utilizado dados de um *dataset open-source* permitindo que se efetue uma prova de conceito para esta componente da plataforma.

Sendo assim, o *dataset open-source* utilizado foi o *Predictive Maintenance Modeling Guide Collection (PMMGC)* que é constituído por leituras de telemetria em tempo real e histórico de falhas adquiridos ao longo do ano de 2015 para 100 máquinas (Microsoft, 2016). O conjunto de dados completo tem 876.142 registos, sendo que há um registo por hora e por máquina (aproximadamente 8.761 registos por máquina) capturados por quatro sensores instalados em cada máquina que medem tensão, pressão, vibração e rotação.

Para esta prova de conceito foram selecionado cinco conjuntos de dados relativos às máquinas que contém mais falhas. Os números de identificação das máquinas estão representados na tabela 6. Cada falha indica a ocorrência de substituição de um componente da máquina.

Tabela 6: Sumário do conjunto de dados de detecção de anomalias adotado

Source	Dataset	Records	Input Features	Failures	Failures (%)
PMMGC (Microsoft, 2016)	Machine 17	≈8,761	4	15	0.17
	Machine 22			15	0.17
	Machine 83			14	0.16
	Machine 98			16	0.18
	Machine 98			19	0.22

Assim, considerando o conjunto de dados de deteção de anomalias adotado, este é composto por dois *datasets*, um deles composto por dados coletados por hora a partir de sensores instalados em 100 máquinas diferentes referentes a voltagem, rotação, pressão e vibração no ano de 2015 e o outro *dataset* composto pelas falhas existentes em cada máquina, assim como o componente que foi substituído nas máquinas. Na tabela 7, são apresentados estes dados de forma mais clara.

Tabela 7: Síntese dos dados utilizados.

Colunas	**Descrição**	**Unidade**
MachineID	ID da máquina	integer
Volt	Potencial elétrico entre dois pontos	V^2
Rotate	Valor do deslocamento rotativo	RPS
Pressure	Valor da pressão exercida	Pa
Vibration	Valor da mudança de velocidade	m/s^2
Failure	Registo da ocorrência de uma falha	integer

4.3 Preparação dos Dados

A etapa de preparação dos dados é uma etapa muito importante e que antecede a modelação. Esta etapa é responsável por reunir todas as alterações necessárias aos dados anteriormente apresentados. Normalmente, nesta etapa, ocorre a seleção dos atributos e linhas a utilizar, a transformação e a limpeza dos dados necessários para que se possa, finalmente, aplicar técnicas e modelos de *Machine Learning*. Nas etapas posteriores, pode ser necessário voltar a esta etapa para realizar uma melhor adaptação dos dados.

Considerando os dados, nomedamente as *labels* descritas na Tabela 7, numa primeira instância não se verifica a necessidade de selecionar variáveis, uma vez que todas as variáveis apresentadas são importantes para o problema apresentado. Posteriormente, foi também possível constatar que os dados recolhidos, sendo estes recolhidos por sensores, não apresentam erros a serem corrigidos, nem valores em falta.

Numa segunda instância foi necessário agregar o conjunto de dados telemétricos com o conjunto de dados de registo de falhas, uma vez que, estes se encontravam separados em dois *datasets* diferentes. Para esse propósito foi criada uma função demonstrada em 4.1 que lê os dois ficheiros *csv* e agrega os mesmos através das variáveis *datetime* e *MachineID*. Posteriormente, foi substituído os valores dos campos da variável *failure* para 1 quando ocorre uma falha e para 0 quando não existe qualquer ocorrência.

```
def main():
    telemetry = pd.read_csv('data/PdM_telemetry.csv', on_bad_lines='
        skip')
    failures = pd.read_csv('data/PdM_failures.csv', on_bad_lines='skip'
        )
```

```
    data = pd.merge(telemetry, failures, on=['datetime', 'machineID'],
        how='left')
    data['failure'] = (
        data['failure']
        .replace(to_replace='comp.', value=1, regex=True)
        .fillna(0)
        .astype(int)
)
```

Listagem 4.1: Código *Merge and Replace function*.

Foi também necessário dividir o conjunto de dados em duas partes, uma delas onde apenas existem ocorrências normais, ou seja, onde não existe qualquer tipo de falhas, e outra em que existem ocorrências anormais, ou seja, onde existem falhas. Esta divisão está demonstrada em 4.2 e irá ser utilizada posteriormente na etapa de Modelação.

```
print("Total data size: %s" % data.shape[0])

normal = data[data.failure.values == 0]
failure = data[data.failure.values == 1]

print("\tNormal: %s" % normal.shape[0])
print("\tFailure: %s" % failure.shape[0])
```

Listagem 4.2: Código *Function to separate the data in normal and failure*.

Concluído o pré-processamento dos dados e obtida a representação do conjunto de dados, procede-se agora à seleção das variáveis necessárias para a modelação. Assim sendo, foi apenas removida a variável ***'datetime'***, uma vez que não é necessária para o problema em questão.

4.4 Modelação

Todos os métodos de *Machine Learning* foram implementados utilizando a linguagem *Python* e os seguintes módulos: *scikit-learn* – para *IsolationForest e RandomForest*; *TensorFlow* – para *AutoEncoders*; e *H2O* para a componente de *AutoML*.

O algoritmo *Isolation Forest* é um algoritmo de *Machine Learning* que deteta anomalias através do isolamento (distância entre um ponto de dados e os restantes dados), em vez de modelar através dos pontos normais. Este tira proveito de duas características significativas de instâncias anormais (Liu et al., 2008): estão presentes em menores quantidades e também são numericamente diferentes das instâncias normais. Este algoritmo adota este principio, construindo um conjunto de várias árvores de isolamento, cada uma contendo as instâncias anormais mais próximas da raíz da árvore, representado na figura 35.

A implementação do *scikit-learn Isolation Forest* fornece uma pontuação de decisão que varia de (pontuação anormal mais alta) a $\hat{y}_i = 1$ (pontuação normal mais alta). Para obter uma pontuação de proba-

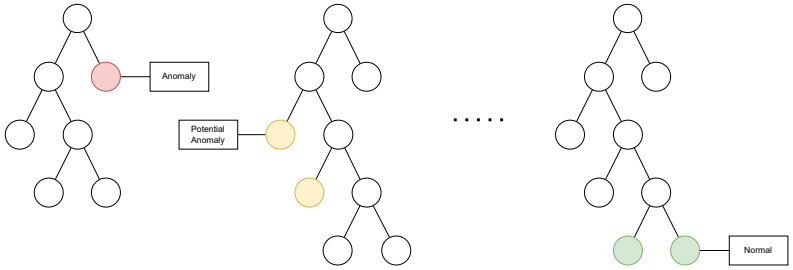


Figura 35: Exemplo da estrutura de IF adotada

bilidade de anomalia ($d_i \in [0,1]$, para um exemplo de entrada i), é realizado um redimensionamento das pontuações do *Isolation Forest* através da computação de $d_i = (1 - \hat{y}_i)/2$.

Autoencoders são técnicas de aprendizagem não supervisionada que compactam e codificam dados de forma eficiente em uma representação de menor dimensão, assumindo uma *bottleneck layer* (com L_b unidades ocultas) (Hinton and Salakhutdinov, 2006). Seja $(L_I, L_1, ..., L_H, L_O)$ a denotação da estrutura (totalmente conectada) de uma *DFFN* com os tamanhos dos nós da camada, onde L_I e L_O representam a entrada e tamanhos de camada de saída e H é o número de camadas ocultas. O *AE* proposto, representado na figura 36, é baseado em uma arquitetura que anteriormente obteve resultados de deteção de anomalias de alta qualidade em uma tarefa de deteção de anomalias industriais (Matos et al., 2021). Este assume $L_I = L_O$, uma estrutura simétrica de codificador e decodificador (por exemplo, $L_1 = L_{O-1}$) e a popular função de ativação *ReLu* que é utilizada por todas as unidades neurais ocultas, com exceção da camada de saída, que assume uma função de ativação linear.

No componente do codificador, o número de unidades de camada oculta diminui pela metade em cada camada oculta subsequente até que o tamanho do *bottleneck* (L_b) seja alcançado: $L_1 = L_I/2$, $L_2 = L_1/2$ e vice-versa. Cada camada oculta também é anexada com uma camada de *BN*. *AE* representam a estrutura base utilizada para todos os conjuntos de dados deste trabalho.

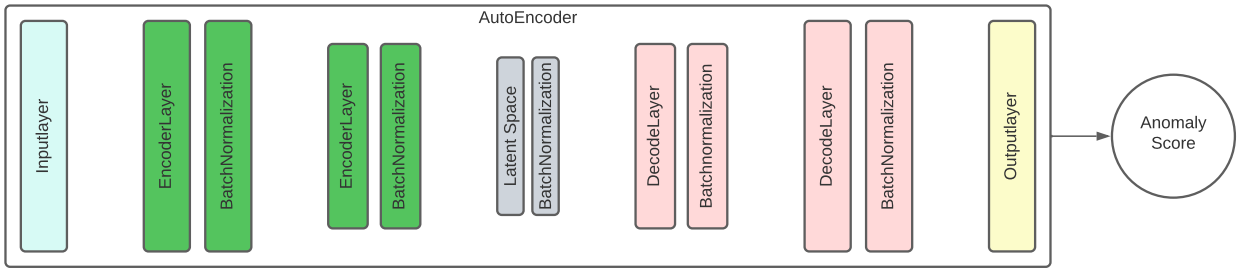


Figura 36: Exemplo da estrutura de AE adotada.

Quando adaptado para deteção de anomalias, o algoritmo *AE* é alimentado apenas com instâncias padrão (normais), visando gerar valores de saída idênticos às suas entradas. Neste trabalho, o *AE* é treinado com o otimizador *Adam* utilizando um tamanho de lote de 32.200 *epochs* e *early stopping* (utilizando 10% dos dados de treino como o conjunto de validação). O erro médio absoluto (*MAE*) é utilizado como função de perda e erro de reconstrução: $MAE_i = \sum_{k=1}^{n} \frac{|x_{i,k} - \hat{x}_{i,k}|}{n}$,onde $x_{i,k}$ e $\hat{x}_{i,k}$ denotam o valor de entrada e saída do *AE* para a i-th instância de dados e para k-th *input* ou *output node*.

O erro de reconstrução *MAE* é utilizado como pontuação de decisão $d_i = MAE_i$, onde quanto maiores os erros de reconstrução, maior a probabilidade de anomalia.

Relativamente aos métodos de aprendizagem supervisionada, o *RF* é um método popular que tende a obter resultados de previsão de alta qualidade ao assumir os valores de hiperparâmetros padrão (Delgado et al., 2014). O algoritmo funciona como um conjunto de árvores de decisão que formam uma "floresta". Cada árvore depende dos valores de um vetor de entrada amostrado aleatoriamente e de uma seleção de amostras de treino (Breiman, 2001). *RF* pode ser utilizado para tarefas de regressão e classificação. Neste trabalho, foi utilizado o *RF* para gerar probabilidades de classe de anomalia ($\in$[0.0,1.0]), que são usadas como pontuação de decisão (d_i). Para o método de *AutoML*, foi adoptado a ferramenta *H2O*, que realiza um treino e ajuste automático de vários algoritmos de *ML* dentro de um limite de tempo especificado pelo utilizador. Sob a configuração padrão *H2O* adotada, a ferramenta treina quatro algoritmos distintos: *RF*, *GLM*, *GBM* e uma rede *DFFN* padrão. Em seguida, faz uso de um *Stacking Ensemble*, que utiliza todos os modelos previamente treinados para gerar entradas para outro modelo *GLM*. Para proceder à seleção do modelo, foi configurado a ferramenta para otimizar a *AUC* da análise *ROC* (Fawcett, 2006). A ferramenta foi corrida por um máximo de 10 minutos, assumindo um esquema interno de *5-fold cross-validation* (aplicado aos dados de treino).

4.5 Avaliação

Esta fase, considerada uma das mais importantes da metodologia *CRISP-DM*, consiste na avaliação dos modelos assim como os resultados obtidos pelos modelos. É possível verificar, através desta análise, qual o melhor modelo para o problema, garantindo que é validado aquele com melhor capacidade preditiva.

Para avaliar o desempenho da deteção de anomalias, foi aplicado um esquema externo de *5-fold cross-validation* a todos os dados disponíveis. Durante cada uma das cinco iterações, os dados de treino foram utilizados para ajustar o modelo de *ML* e os dados de teste foram utilizados para calcular as pontuação de decisão de anomalia previstas e respetiva análise *ROC* (Fawcett, 2006). Quando um modelo gera uma pontuação de decisão d_i, a classe pode ser interpretada como positiva se $d_i > K$, onde K é um limite de decisão fixo, pelo contrário o mesmo é considerado negativo. A curva *ROC* demonstra o desempenho de um classificador de duas classes em todos os $K[0,1]$ valores, imprimindo um menos a especificidade (x-eixo), ou *FPR*, versus a sensibilidade (y-eixo), ou *TPR*.

O desempenho de discriminação é dado por $AUC = \int_0^1 ROC dK$. É importante de mencionar que a medida *AUC* é uma medida importante que contém duas principais vantagens (Ribeiro et al., 2021): os valores de qualidade não são afetados pela taxa desbalanceada da classe-alvo; e os valores de qualidade são fáceis de interpretar (50% – desempenho de um classificador aleatório; 70% - bom; 80% – muito bom; 90% - excelente; e 100% - classificador ideal).

Após a execução do *5-fold cross-validation*, os resultados do *AUC* são agregados pelo cálculo do valor mediano (que é menos sensível a valores discrepantes quando comparado com o valor médio).

No primeiro conjunto de experimentos computacionais, executamos o *5-fold cross-validation* para os conjuntos de dados testados e quatro algoritmos de *ML* de linha de base (*RF*, *AutoML*, *IF*) e uma estrutura

de *AE* genérica. Os resultados estão resumidos na tabela 8. Ao comparar os resultados da aprendizagem supervisionada (*RF* e *AutoML*), o método automatizado *H2O* tende a obter uma melhor desempenho de deteção de anomalias. Por exemplo, apresenta um valor médio de *PMMGC* mais alto (66,4% versus 62,5%). Voltando aos métodos de uma classe, tanto o *IF* quanto *AE* obtêm os melhores valores médios de *AUC* em três casos (*IF* – Máquina 17 e 98; *AE* – Máquina 22 e 99). No geral, *AE* produz uma melhor precisão de discriminação (em termos de *AUC* mediana) para *PMMGC*. Mais importante, a comparação de métodos supervisionados *versus* métodos de uma classe tende a favorecer os últimos. Com efeito, há apenas um caso (Máquina *PMMGC* 83) em que um algoritmo de aprendizagem (*RF*) obtém o melhor desempenho de deteção de anomalias. Para todos os outros casos, os algoritmos *IF* e *AE* tendem a superar os métodos de aprendizagem supervisionada. Este é um resultado importante, pois os métodos de uma classe têm a vantagem de não exigir dados rotulados durante o procedimento de treino, que geralmente é mais caro, devido ao esforço manual necessário para inspecionar e marcar os exemplos.

Tabela 8: Valores médios de AUC para o primeiro conjunto de experiências.

		Supervised		Unsupervised	
Dataset		RF	AutoML	IF	AE
PMMGC:	Machine 17	0.5000	0.6664	0.6830	0.6459
	Machine 22	0.6667	0.6661	0.7069	0.7173
	Machine 83	0.6667	0.6638	0.6114	0.4768
	Machine 98	0.6250	0.6250	0.8155	0.7979
	Machine 99	0.5000	0.4986	0.6818	0.7047
Median PMMGC value:		0.6250	0.6638	0.7069	0.7173

Para verificar ainda, se uma técnica de balanceamento melhoraria os resultados da aprendizagem supervisionada, foi executado um segundo conjunto de experimentos adotando o mesmo conjunto de dados *PMMGC*, para quais o desempenho de deteção de anomalia de aprendizagem supervisionada é fraco (por exemplo, classificação aleatória para *RF* e Máquina 17). Nesse conjunto de experiências, os dados de treino de *RF* e *AutoML* foram balanceados pela primeira vez atráves de duas técnicas de sobre-amostragem, *SMOTE* e *GC*, conforme implementado pelo *imblearn* (Lemaître et al., 2017) e *sdv* (Patki et al., 2016) módulos *Python*, utilizando as configurações base. Observamos que nessas experiências, os dados de teste obtidos pelo método de *5-fold cross-validation* são mantidos sem alterações. Os resultados obtidos são mostrados em Table 9.

Em vez de melhorar o desempenho dos métodos de aprendizagem supervisionada, ambas as técnicas de reamostragem tendem a diminuir a capacidade de detecção de anomalias.

Por exemplo, o valor médio de *AUC* da máquina para *RF* diminuiu de 62,5% para 49,49% (*SMOTE*) e 50,0% (*GC*). Da mesma forma, o desempenho da mediana do *AutoML* diminuiu de 66,4% para 50% (para *SMOTE* e *GC*).

Os resultados abaixo do expectado obtido pelas técnicas de sobreamostragem pode ser explicado pela natureza extremamente desequilibrada dos conjuntos de dados analisados. Como a percentagem de falhas é inferior a 1%, os métodos *SMOTE* e *GC* têm que gerar um número substancialmente elevado de

Tabela 9: Valores médios de AUC para o segundo conjunto de experiências.

Machine	SMOTE		Gaussian Copula	
	RF	AutoML	RF	AutoML
Machine 17	0.4949	0.4991	0.6667	0.6641
Machine 22	0.6584	0.6624	0.5000	0.4991
Machine 83	0.4949	0.4986	0.5000	0.5000
Machine 98	0.6561	0.6621	0.6250	0.4997
Machine 99	0.4874	0.4951	0.5000	0.4977
Median machine value:	0.4949	0.4991	0.5000	0.4997

casos sintéticos associados a anomalias, o que cria registos da classe menos representativa com a possibilidade de conter informações incorretas que por sua vez prejudicam a aprendizagem supervisionada.

A abordagem de aprendizagem de uma classe usa a própria classe para estabelecer os limites que a distinguem de outras classes, provando ser mais útil do que balancear conjuntos de dados extremamente desequilibrados. No geral, os melhores resultados de deteção de anomalias foram obtidos pelos algoritmos de uma classe (*IF* e *AE*), que têm a vantagem de não exigir dados rotulados durante o procedimento de treino, provando ser mais eficazes na deteção de anomalias do que aplicar técnicas de sobreamostragem ao conjunto de dados (por exemplo, *SMOTE* e *GC*).

4.6 Implementação

O trabalho desta dissertação não pode englobar a componente final de implementação, uma vez que se tornou impossível a realização das experiências de *Machine Learning*, no contexto do projeto mobilizador *Produtech*. Sendo assim, este trabalho foi desenvolvido localmente, não com o objetivo de ser colocado em produção, mas sim com foco na apresentação de contributos válidos para que possa servir futuramente a componente de algoritmia do projeto.

Mais concretamente, no contexto do projeto *Produtech*, a implementação do trabalho aqui desenvolvido corresponderia à integração na arquitetura e plataforma apresentada no caso anterior. Esta implementação será feita nos servidores de cada cliente do projeto e integrada com a plataforma apresentada, responsável por incorporar os modelos de *Machine Learning* para a possível deteção de falhas e anomalias nas máquinas industriais que estão posicionadas no *shop-floor* dos vários clientes.

5

Conclusões

O último capítulo desta dissertação, apresenta uma visão global do trabalho realizado, com destaque para os aspetos desenvolvidos e para as conclusões retiradas do mesmo. Está dividido em três subcapítulos. No primeiro, Síntese do Trabalho Realizado, é feita a descrição do trabalho realizado ao longo do documento. De seguida, os contributos alcançados e por último são identificados os próximos passos para a continuidade do projeto através de trabalhos futuros.

5.1 Síntese do Trabalho Realizado

Esta dissertação foi realizada no âmbito do projeto *"PRODUTECH4S&C: PRODUTECH SUSTENTÁVEL CIRCULAR"*, o qual foi desenvolvido no Centro de Computação Gráfica. Este projeto mobilizador surgiu no âmbito da crescente complexidade em criar valor no desenvolvimento de novas metodologias, estratégias e abordagens de forma a originar novos produtos e serviços que vão de encontro com uma economia circular, e garantindo a sustentabilidade dos vários setores industriais. Desta forma, os objetivos eram a implementação de uma plataforma/arquitetura de *Big Data Analytics* capaz de tornar útil no desenvolvimento do produto a elevada quantidade de dados gerados pelos próprios produtos em fase de uso e a utilização de algoritmos e técnicas de *Machine Learning* para deteção de anomalias. Com os objetivos traçados, esta dissertação iniciou-se com a explicação das metodologias utilizadas, *CRISP-DM* e *DSRM*. Seguiu-se a revisão do estado da arte, onde foram abordados alguns conceitos importantes para o desenvolvimento da dissertação. Em particular foram destacadas as áreas de conhecimento estudadas, relacionadas com os conceitos de *Machine Learning*, *Big Data*, *Arquiteturas Big Data* e *Hadoop Framework*. Foram ainda apresentadas as ferramentas que tornaram possível o desenvolvimento dos diferentes casos de estudo. Por último, os casos de estudo foram apresentados e analisados, na sequência das metodologias apresentadas. No primeiro caso de estudo foi utilizada a metodologia *DSRM* para o desenvolvimento e implementação da plataforma de *Big Data Analytics* resposável pela aquisição, processamento e visualização dos dados gerados pelos próprios produtos em fase de uso. No segundo caso de estudo, Estudo e utilização de técnicas de *Machine Learning* para deteção de anomalias, o trabalho foi organizado pelas 5 fases da metodologias *CRISP-DM*.

5.2 Contributos

Concluído o trabalho desenvolvido, é necessário fazer um ponto de situação relativamente aos resultados alcançados, tendo em conta os objetivos que foram inicialmente delineados. É importante mencionar que as expectativas iniciais relativas ao trabalho a desenvolver foram alcançadas e que o presente trabalho cumpre com aquilo que foi inicialmente proposto tendo em conta a plataforma implementada, assim como a utilização de algoritmos de *Machine Learning* para deteção de anomalias.

Embora uma das expectativas fosse a utilização de dados do projeto mobilizador em todos os casos de estudo efetuados, este requisito não foi cumprido devido à falta de sensorização do produto e à falta de volume de dados para que fosse possível retirar as devidas conclusões. Em alternativa, recorreu-se à realização de duas provas de conceito, uma utilizando os dados existentes até ao momento num dos clientes do projeto mobilizador, e outra à utilização de conjuntos de dados públicos, constituído por leituras de telemetria em tempo real e histórico de falhas.

Para além do estudo teórico, foram atingidos dois contributos principais sendo estes a elaboração e implementação de uma plataforma de *Big Data Analytics* com o intuito de tornar útil no desenvolvimento do produto a elevada quantidade de dados e a utilização de técnicas de *Machine Learning* para a deteção de anomalias em ambiente industrial.

Por fim, é ainda possível destacar a publicação de um artigo de investigação cientifica na conferência *18th International Conference on Artificial Intelligence Applications and Innovations*.

5.3 Trabalho Futuro

Os resultados aqui demonstrados, abrem possibilidades para trabalho futuro. Por exemplo, considera-se fundamental para a componente de deteção de anomalias, o processamento de um volume de dados considerável, sendo que não foi realizada tal análise no decorrer do projeto. Assim, é necessário que com o decorrer do tempo novos dados sejam armazenados na plataforma aqui implementada.

Dentro do projeto *I&D* no qual este trabalho de dissertação se encontra inserido, há a ambição da plataforma ser implementada em todos os parceiros industriais, garantindo que a mesma é funcional em diferentes casos de uso e em diferentes ambientes industriais. Assim, há a necessidade de implementar os algoritmos de *Machine Learning* nos diferentes parceiros industriais, de modo a gerar alarmes para a deteção de falhas em máquinas no ambiente industrial e fornecer essas informações aos atores envolvidos para que possam atualizar ou gerar produtos desenvolvidos.

No futuro, quando a plataforma for colocada efetivamente em produção, pode haver a necessidade de recorrer a tecnologias de *scheduling* como o *Apache Airflow*, sendo esta uma plataforma de código aberto para o desenvolvimento, agendamento e monitorização de fluxos de trabalho orientado a fluxos *batch*.

Referências Bibliográficas

Aggarwal, C. C. (2017). An introduction to outlier analysis. In *Outlier analysis*, pages 1–34. Springer.

Ahmed, M., Mahmood, A. N., and Islam, M. R. (2016). A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems*, 55:278–288.

Akoglu, L., Tong, H., and Koutra, D. (2015). Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688.

Alla, S. and Adari, S. K. (2019). *Beginning Anomaly Detection Using Python-Based Deep Learning*. Apress, Berkeley, CA.

Ayodele, T. O. (2010). Types of machine learning algorithms. *New advances in machine learning*, 3:19–48.

Begum, A., Fatima, F., and Haneef, R. (2019). Big data and advanced analytics. In *International Conference on Information Technology & Systems*, pages 594–601. Springer.

Benjelloun, S., Aissi, M. E. M. E., Loukili, Y., Lakhrissi, Y., Ali, S. E. B., Chougrad, H., and Boushaki, A. E. (2020). Big data processing: Batch-based processing and stream-based processing. In *2020 Fourth International Conference On Intelligent Computing in Data Sciences (ICDS)*, pages 1–6.

Bhardwaj, A., Vanraj, Kumar, A., Narayan, Y., and Kumar, P. (2015). Big data emerging technologies: A casestudy with analyzing twitter data using apache hive. In *2015 2nd International Conference on Recent Advances in Engineering Computational Sciences (RAECS)*, pages 1–6.

Borthakur, D. et al. (2008). Hdfs architecture guide. *Hadoop apache project*, 53(1-13):2.

Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

Breunig, M. M., Kriegel, H., Ng, R. T., and Sander, J. (2000). LOF: identifying density-based local outliers. In Chen, W., Naughton, J. F., and Bernstein, P. A., editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 93–104. ACM.

Cao, N., Lin, Y.-R., Gotz, D., and Du, F. (2018). Z-glyph: Visualizing outliers in multivariate data. *Information Visualization*, 17(1):22–40.

Cassandra, A. (2014). Apache cassandra. *Website. Available online at http://planetcassandra. org/what-is-apache-cassandra*, 13.

Chalapathy, R. and Chawla, S. (2019). Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*.

Chandarana, P. and Vijayalakshmi, M. (2014). Big data analytics frameworks. In *2014 International Conference on Circuits, Systems, Communication and Information Technology Applications (CSCITA)*, pages 430–434. IEEE.

Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58.

Chatti, S. (2019). Using spark, kafka and nifi for future generation of etl in it industry. *Proceedings of Journal of Innovation in Information Technology*.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *J. Artif. Intell. Res.*, 16:321–357.

Chen, C. P. and Zhang, C.-Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information sciences*, 275:314–347.

Chen, M. and Liu, Y. (2014). Big data: A survey mobile networks and application.

Correia, J., Costa, C., and Santos, M. Y. (2019). Challenging sql-on-hadoop performance with apache druid. In *International Conference on Business Information Systems*, pages 149–161. Springer.

Costa, C. and Santos, M. Y. (2017). Big data: State-of-the-art concepts, techniques, technologies, modeling approaches and research challenges.

Cui, Y., Kara, S., and Chan, K. C. (2020). Manufacturing big data ecosystem: A systematic literature review. *Robotics and computer-integrated Manufacturing*, 62:101861.

Delgado, M. F., Cernadas, E., Barro, S., and Amorim, D. G. (2014). Do we need hundreds of classifiers to solve real world classification problems? *J. Mach. Learn. Res.*, 15(1):3133–3181.

Demchenko, Y., Grosso, P., De Laat, C., and Membrey, P. (2013). Addressing big data issues in scientific data infrastructure. In *2013 International conference on collaboration technologies and systems (CTS)*, pages 48–55. IEEE.

Doan, T. and Kalita, J. (2015). Selecting machine learning algorithms using regression models. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 1498–1505. IEEE.

Fallucchi, F., Petito, M., and Luca, E. W. D. (2018). Analysing and visualising open data within the data and analytics framework. In *Research Conference on Metadata and Semantics Research*, pages 135–146. Springer.

Fawcett, T. (2006). An introduction to ROC analysis. *Patt. Recognition Letters*, 27:861–874.

Feick, M., Kleer, N., and Kohn, M. (2018). Fundamentals of real-time data processing architectures lambda and kappa. In Becker, M., editor, *SKILL 2018 - Studierendenkonferenz Informatik*, pages 55–66, Bonn. Gesellschaft für Informatik e.V.

Gandomi, A. and Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. *International journal of information management*, 35(2):137–144.

Geerts, G. L. (2011). A design science research methodology and its application to accounting information systems research. *International Journal of Accounting Information Systems*, 12(2):142–151. Special Issue on Methodologies in AIS Research.

Grubbs, F. E. (1969). Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21.

Hevner, A. and Chatterjee, S. (2010). Design science research in information systems. In *Design research in information systems*, pages 9–22. Springer.

Hinton, G. and Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507. cited By 9376.

Hiraman, B. R. et al. (2018). A study of apache kafka in big data stream processing. In *2018 International Conference on Information, Communication, Engineering and Technology (ICICET)*, pages 1–3. IEEE.

Hodge, V. and Austin, J. (2004). A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126.

Honnutagi, P. S. (2014). The hadoop distributed file system. *International Journal of Computer Science and Information Technologies (IJCSIT)*, 5(5):6238–6243.

Huai, Y., Chauhan, A., Gates, A., Hagleitner, G., Hanson, E. N., O'Malley, O., Pandey, J., Yuan, Y., Lee, R., and Zhang, X. (2014). Major technical advancements in apache hive. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1235–1246.

Iftikhar, N., Lachowicz, B. P., Madarasz, A., Nordbjerg, F. E., Baattrup-Andersen, T., and Jeppesen, K. (2020). Real-time visualization of sensor data in smart manufacturing using lambda architecture. In *DATA*, pages 215–222.

Joy, A. M. (2015). Performance comparison between linux containers and virtual machines. In *2015 international conference on advances in computer engineering and applications*, pages 342–346. IEEE.

Kim, S.-S., Lee, W.-R., and Go, J.-H. (2019). A study on utilization of spatial information in heterogeneous system based on apache nifi. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1117–1119. IEEE.

Kitchin, R. and McArdle, G. (2016). What makes big data, big data? exploring the ontological characteristics of 26 datasets. *Big Data & Society*, 3(1):2053951716631130.

Kreps, J. (2014). Questioning the lambda architecture. the lambda architecture has its merits, but alternatives are worth exploring. *O'Reilly Media. https://www. oreilly. com/ideas/questioning-the-lambda-architecture. Zugegriffen am*, 21:2020.

Krishnan, K. (2013). *Data warehousing in the age of big data*. Newnes.

Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40.

Le Noac'H, P., Costan, A., and Bougé, L. (2017). A performance evaluation of apache kafka in support of big data streaming applications. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 4803–4806. IEEE.

Lee, J. Y., Yoon, J. S., and Kim, B.-H. (2017). A big data analytics platform for smart factories in small and medium-sized manufacturing enterprises: An empirical case study of a die casting factory. *International Journal of Precision Engineering and Manufacturing*, 18(10):1353–1361.

Lee, K.-H., Lee, Y.-J., Choi, H., Chung, Y. D., and Moon, B. (2012). Parallel data processing with mapreduce: a survey. *AcM sIGMoD record*, 40(4):11–20.

Lemaître, G., Nogueira, F., and Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5.

Lima-Monteiro, P., Parreira-Rocha, M., Rocha, A. D., and Barata Oliveira, J. (2016). Big data analysis to ease interconnectivity in industry 4.0—a smart factory perspective. In *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*, pages 237–245. Springer.

Lin, C.-Y. and Lin, Y.-C. (2017). An overall approach to achieve load balancing for hadoop distributed file system. *International Journal of Web and Grid Services*, 13(4):448–466.

Lin, J. (2017). The lambda and the kappa. *IEEE Internet Computing*, 21(05):60–66.

Liu, F. T., Ting, K. M., and Zhou, Z. (2008). Isolation forest. In *Proc. of the 8th IEEE Int. Conf. on Data Mining (ICDM), Pisa, Italy*, pages 413–422. IEEE.

Lu, S. and Alwerfali, H. (2016). Implementation and performance analysis of apache hadoop. *IOSR Journal of Computer Engineering (IOSR-JCE)*, 18(5):48–58.

Lu, Y. (2017). Industry 4.0: A survey on technologies, applications and open research issues. *Journal of industrial information integration*, 6:1–10.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Hung Byers, A., et al. (2011). *Big data: The next frontier for innovation, competition, and productivity*. McKinsey Global Institute.

Matloff, N. (2017). *Statistical regression and classification: from linear models to machine learning*. Chapman and Hall/CRC.

Matos, L. M., Cortez, P., Mendes, R., and Moreau, A. (2018). A comparison of data-driven approaches for mobile marketing user conversion prediction. In Jardim-Gonçalves, R., Mendonça, J. P., Jotsov, V., Marques, M., Martins, J., and Bierwolf, R. E., editors, *9th IEEE International Conference on Intelligent Systems, IS 2018, Funchal, Madeira, Portugal, September 25-27, 2018*, pages 140–146. IEEE.

Matos, L. M., Domingues, A., Moreira, G., Cortez, P., and Pilastri, A. L. (2021). A comparison of machine learning approaches for predicting in-car display production quality. In Yin, H., Camacho, D., Tiño, P., Allmendinger, R., Tallón-Ballesteros, A. J., Tang, K., Cho, S., Novais, P., and Nascimento, S., editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2021 - 22nd International Conference, IDEAL 2021, Manchester, UK, November 25-27, 2021, Proceedings*, volume 13113 of *Lecture Notes in Computer Science*, pages 3–11. Springer.

Mehryar, M., Afshin, R., and Ameet, T. (2012). Foundations of machine learning. adaptive computation and machine learning.

Merkel, D. et al. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2.

Microsoft (2016). Predictive maintenance modelling guide.

Mitchell, T. M. and Mitchell, T. M. (1997). *Machine learning*, volume 1. McGraw-hill New York.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). *Foundations of machine learning*. MIT press.

Mrozek, D. (2018). Scalable big data analytics for protein bioinformatics. *Computational Biology*.

Muharemi, F., Logofătu, D., and Leon, F. (2019). Machine learning approaches for anomaly detection of water quality on a real-world data set. *Journal of Information and Telecommunication*, 3(3):294–307.

Ness, D., Swift, J., Ranasinghe, D. C., Xing, K., and Soebarto, V. (2015). Smart steel: new paradigms for the reuse of steel enabled by digital tracking and modelling. *Journal of Cleaner Production*, 98:292–303.

Ordonez-Ante, L., Vanhove, T., Van Seghbroeck, G., Wauters, T., and De Turck, F. (2016). Interactive querying and data visualization for abuse detection in social network sites. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 104–109. IEEE.

Oussous, A., Benjelloun, F.-Z., Ait Lahcen, A., and Belfkih, S. (2018). Big data technologies: A survey. *Journal of King Saud University - Computer and Information Sciences*, 30(4):431–448.

Pang, G., Shen, C., Cao, L., and Hengel, A. V. D. (2021). Deep learning for anomaly detection: A review. *ACM Comput. Surv.*, 54(2).

Patki, N., Wedge, R., and Veeramachaneni, K. (2016). The synthetic data vault. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 399–410.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77.

Pereira, P. J., Pereira, A., Cortez, P., and Pilastri, A. L. (2021). A comparison of machine learning methods for extremely unbalanced industrial quality data. In Marreiros, G., Melo, F. S., Lau, N., Cardoso, H. L., and Reis, L. P., editors, *Progress in Artificial Intelligence - 20th EPIA Conference on Artificial Intelligence, EPIA 2021, Virtual Event, September 7-9, 2021, Proceedings*, volume 12981 of *Lecture Notes in Computer Science*, pages 561–572. Springer.

Rai, K., Devi, M. S., and Guleria, A. (2016). Decision tree based algorithm for intrusion detection. *International Journal of Advanced Networking and Applications*, 7(4):2828.

Ranjan, R. (2014). Streaming big data processing in datacenter clouds. *IEEE cloud computing*, 1(1):78–83.

REDDY, K. and REDDY, R. (2020). Improving efficiency of data compaction by creating & evaluating a random compaction strategy in apache cassandra.

Ribeiro, D., Matos, L. M., Cortez, P., Moreira, G., and Pilastri, A. L. (2021). A comparison of anomaly detection methods for industrial screw tightening. In Gervasi, O., Murgante, B., Misra, S., Garau, C., Blecic, I., Taniar, D., Apduhan, B. O., Rocha, A. M. A. C., Tarantino, E., and Torre, C. M., editors, *Computational Science and Its Applications - ICCSA 2021 - 21st International Conference, Cagliari, Italy, September 13-16, 2021, Proceedings, Part II*, volume 12950 of *Lecture Notes in Computer Science*, pages 485–500. Springer.

Ruff, L., Görnitz, N., Deecke, L., Siddiqui, S. A., Vandermeulen, R. A., Binder, A., Müller, E., and Kloft, M. (2018). Deep one-class classification. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4390–4399. PMLR.

Sajwan, V., Yadav, V., and Haider, M. (2015). The hadoop distributed file system: Architecture and internals. *International Journal of Combined Research & Development (IJCRD)*, 4(3):541–544.

Salloum, S., Dautov, R., Chen, X., Peng, P. X., and Huang, J. Z. (2016). Big data analytics on apache spark. *International Journal of Data Science and Analytics*, 1(3):145–164.

Samizadeh, I. (2018). A brief introduction to two data processing architectures—lambda and kappa for big data. *March https://towardsdatascience. com*.

Saraswat, H., Sharma, N., and Rai, A. (2017). Enhancing the traditional file system to hdfs: a big data solution. *Int. J. Comput. Appl*, pages 0975–8887.

Sethi, R., Traverso, M., Sundstrom, D., Phillips, D., Xie, W., Sun, Y., Yegitbasi, N., Jin, H., Hwang, E., Shingte, N., et al. (2019). Presto: Sql on everything. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1802–1813. IEEE.

Shafiq, S. I., Sanin, C., Toro, C., and Szczerbicki, E. (2015). Virtual engineering object (veo): Toward experience-based design and manufacturing for industry 4.0. *Cybernetics and Systems*, 46(1-2):35–50.

Shaheen, J. (2017). Apache kafka: real time implementation with kafka architecture review. *International Journal of Advanced Science and Technology*, 109:35–42.

Shoro, A. G. and Soomro, T. R. (2015). Big data analysis: Apache spark perspective. *Global Journal of Computer Science and Technology*.

Sudheer, D. and Lakshmi, A. R. (2015). Performance evaluation of hadoop distributed file system. *Pseudo Distrib Mode Fully Distrib Mode (9)*, pages 81–86.

Thein, K. M. M. (2014). Apache kafka: Next generation distributed messaging system. *International Journal of Scientific Engineering and Technology Research*, 3(47):9478–9483.

Tjahjono, B., Esplugues, C., Ares, E., and Pelaez, G. (2017). What does industry 4.0 mean to supply chain? *Procedia manufacturing*, 13:1175–1182.

Tsai, C.-W., Lai, C.-F., Chao, H.-C., and Vasilakos, A. V. (2015). Big data analytics: a survey. *Journal of Big data*, 2(1):1–32.

Vijayan, V. K., Bindu, K., and Parameswaran, L. (2017). A comprehensive study of text classification algorithms. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1109–1113. IEEE.

Villars, R. L., Olofson, C. W., and Eastwood, M. (2011). Big data: What it is and why you should care. *White paper, IDC*, 14:1–14.

Warren, J. and Marz, N. (2015). *Big Data: Principles and best practices of scalable realtime data systems*. Simon and Schuster.

Wirth, R. and Hipp, J. (2000). Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, volume 1, pages 29–40. Manchester.

Xu, Z. and Shi, Y. (2015). Exploring big data analysis: fundamental scientific problems. *Annals of Data Science*, 2(4):363–372.

Yang, C., Shen, W., and Wang, X. (2016). Applications of internet of things in manufacturing. In *2016 IEEE 20th international conference on computer supported cooperative work in design (CSCWD)*, pages 670–675. IEEE.

Yang, F., Tschetter, E., Léauté, X., Ray, N., Merlino, G., and Ganguli, D. (2014). Druid: A real-time analytical data store. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 157–168.

Yassine, A., Singh, S., Hossain, M. S., and Muhammad, G. (2019). Iot big data analytics for smart homes with fog and cloud computing. *Future Generation Computer Systems*, 91:563–573.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., et al. (2016). Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65.

Zhelev, S. and Rozeva, A. (2017). Big data processing in the cloud-challenges and platforms. In *AIP Conference Proceedings*, volume 1910, page 060013. AIP Publishing LLC.

Zhong, R. Y., Xu, C., Chen, C., and Huang, G. Q. (2017). Big data analytics for physical internet-based intelligent manufacturing shop floors. *International journal of production research*, 55(9):2610–2621.

A

Anexos

a.1 Docker Compose Plataforma Big Data Analytics.

```
version: '3.7'

services:


  # Hadoop master
  namenode:
    image: bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8
    container_name: namenode
    ports:
      - 9870:9870
      - 8020:8020
    volumes:
      - ./namenode/home/${ADMIN_NAME:?err}:/home/${ADMIN_NAME:?err}
      - ./namenode/hadoop-data:/hadoop-data
      - ./namenode/entrypoint.sh:/entrypoint.sh
      - hadoop-namenode:/hadoop/dfs/name
    env_file:
      - ./hadoop.env
      - .env
    networks:
      - hadoop

  resourcemanager:
    restart: always
    image: bde2020/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8
    container_name: resourcemanager
    ports:
      - 8088:8088
    environment:
      SERVICE_PRECONDITION: "namenode:9870 datanode1:9864"
    env_file:
```

```
    - ./hadoop.env
  networks:
    - hadoop

# Hadoop slave 1
datanode1:
  image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8
  container_name: datanode1
  volumes:
    - hadoop-datanode-1:/hadoop/dfs/data
  environment:
    SERVICE_PRECONDITION: "namenode:9870"
  env_file:
    - ./hadoop.env
  networks:
    - hadoop

nodemanager1:
  image: bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8
  container_name: nodemanager1
  volumes:
    - ./nodemanagers/entrypoint.sh:/entrypoint.sh
  environment:
    SERVICE_PRECONDITION: "namenode:9870 datanode1:9864 resourcemanager
        :8088"
  env_file:
    - ./hadoop.env
    - .env
  networks:
    - hadoop

# Hadoop slave 2
datanode2:
  image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8
  container_name: datanode2
  volumes:
    - hadoop-datanode-2:/hadoop/dfs/data
  environment:
    SERVICE_PRECONDITION: "namenode:9870"
  env_file:
    - ./hadoop.env
  networks:
    - hadoop

nodemanager2:
  image: bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8
  container_name: nodemanager2
```

```
    volumes:
      - ./nodemanagers/entrypoint.sh:/entrypoint.sh
    environment:
      SERVICE_PRECONDITION: "namenode:9870 datanode2:9864 resourcemanager
          :8088"
    env_file:
      - ./hadoop.env
      - .env
    networks:
      - hadoop

# Hadoop slave 3
  datanode3:
    image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8
    container_name: datanode3
    volumes:
      - hadoop-datanode-3:/hadoop/dfs/data
    environment:
      SERVICE_PRECONDITION: "namenode:9870"
    env_file:
      - ./hadoop.env
    networks:
      - hadoop

  nodemanager3:
    image: bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8
    container_name: nodemanager3
    volumes:
      - ./nodemanagers/entrypoint.sh:/entrypoint.sh
    environment:
      SERVICE_PRECONDITION: "namenode:9870 datanode3:9864 resourcemanager
          :8088"
    env_file:
      - ./hadoop.env
      - .env
    networks:
      - hadoop

  historyserver:
    image: bde2020/hadoop-historyserver:2.0.0-hadoop3.2.1-java8
    container_name: historyserver
    ports:
      - 8188:8188
    environment:
      SERVICE_PRECONDITION: "namenode:9870 datanode1:9864 datanode2:9864
          datanode3:9864 resourcemanager:8088"
    volumes:
```

```
    - hadoop-historyserver:/hadoop/yarn/timeline
  env_file:
    - ./hadoop.env
  networks:
    - hadoop

# HUE (Management node, similar to Ambari)
hue:
  container_name: hue
  image: gethue/hue:4.4.0
  ports:
    - 8000:8888
  env_file:
    - ./hadoop.env
  volumes:
    - ./hue/hue-overrides.ini:/usr/share/hue/desktop/conf/hue-overrides.
      ini
  depends_on:
    - namenode
    - resourcemanager
  networks:
    - hadoop

hue-db:
  container_name: hue-db
  restart: always
  image: postgres:11.9
  environment:
    - POSTGRES_USER=user
    - POSTGRES_PASSWORD=secret
    - POSTGRES_DB=hue
  ports:
    - 5432:5432
  volumes:
    - hue-db:/var/lib/postgresql/data
    - ./hue-db/init.sql:/docker-entrypoint-initdb.d/init.sql
  networks:
    - hadoop

# HIVE
hive-server:
  image: bde2020/hive:2.3.2-postgresql-metastore
  container_name: hiveserver
  env_file:
    - ./hadoop.env
  environment:
```

```
    HIVE_CORE_CONF_javax_jdo_option_ConnectionURL: "jdbc:postgresql://
        hive-metastore/metastore"
    SERVICE_PRECONDITION: "hive-metastore:9083"
  networks:
    - hadoop

hive-metastore:
  container_name: hivemetastore
  image: bde2020/hive:2.3.2-postgresql-metastore
  env_file:
    - ./hadoop.env
  command: /opt/hive/bin/hive --service metastore
  networks:
    - hadoop
  environment:
    SERVICE_PRECONDITION: "namenode:9870 datanode1:9864 datanode2:9864
        datanode3:9864 hive-metastore-postgresql:5432"

hive-metastore-postgresql:
  container_name: hivedb
  image: bde2020/hive-metastore-postgresql:2.3.0
  volumes:
    - hive-metastore:/var/lib/postgresql/data
  networks:
    - hadoop

presto-coordinator:
  container_name: presto
  image: shawnzhu/prestodb:0.181
  networks:
    - hadoop

# Spark
spark-master:
  image: bde2020/spark-master:3.1.1-hadoop3.2
  container_name: spark-master
  ports:
    - 8080:8080
    - 7077:7077
  env_file:
    - ./hadoop.env
  networks:
    - hadoop

spark-worker-1:
  image:  bde2020/spark-worker:3.1.1-hadoop3.2
  container_name: spark-worker-1
```

```
    depends_on:
      - spark-master
    ports:
      - 8081:8081
    environment:
      - SPARK_MASTER=spark://spark-master:7077
    env_file:
      - ./hadoop.env
    networks:
      - hadoop

spark-worker-2:
    image:  bde2020/spark-worker:3.1.1-hadoop3.2
    container_name: spark-worker-2
    depends_on:
      - spark-master
    ports:
      - 8082:8081
    environment:
      - SPARK_MASTER=spark://spark-master:7077
    env_file:
      - ./hadoop.env
    networks:
      - hadoop

spark-worker-3:
    image:  bde2020/spark-worker:3.1.1-hadoop3.2
    container_name: spark-worker-3
    depends_on:
      - spark-master
    ports:
      - 8083:8081
    environment:
      - SPARK_MASTER=spark://spark-master:7077
    env_file:
      - ./hadoop.env
    networks:
      - hadoop

# Spark notebooks
jupyter-spark:
    # To see all running servers in this container, execute
    # 'docker exec jupyter-spark jupyter notebook list'
    container_name: jupyter-spark
    build:
      context: jupyter-spark
      args:
```

```
      - SPARK_VERSION=3.1.1
      - HADOOP_VERSION=3.2
      - SPARK_CHECKSUM=
        E90B31E58F6D95A42900BA4D288261D71F6C19FA39C1CB71862B792D1B5564941A320227F

      - OPENJDK_VERSION=11
      # Make sure the python version in the driver (the notebooks) is the
          same as in spark-master,
      # spark-worker-1, and spark-worker-2
      - PYTHON_VERSION=3.7.10
  ports:
    - 8888:8888
    - 8889:8889
    - 4040:4040
    - 4041:4041
  volumes:
    - ./jupyter-spark/work:/home/jovyan/work
    - ./jupyter-spark/drivers:/drivers
  pid: host
  environment:
    - TINI_SUBREAPER=true
  env_file:
    - ./hadoop.env
  networks:
    - hadoop

# Mongo
mongo:
  container_name: mongo
  image: mongo:4.4
  restart: always
  environment:
    - MONGO_INITDB_ROOT_USERNAME=kevinsuedmersen
    - MONGO_INITDB_ROOT_PASSWORD=secret
  volumes:
    - mongo_data:/data/db
    - ./mongo/mongo-data:/mongo-data
  ports:
    - 27019:27017
  networks:
    - hadoop

mongoexpress:
  container_name: mongoexpress
  image: mongo-express:0.54
  restart: always
  ports:
```

```
    - 8091:8081
  environment:
    - ME_CONFIG_MONGODB_ADMINUSERNAME=kevinsuedmersen
    - ME_CONFIG_MONGODB_ADMINPASSWORD=secret
  networks:
    - hadoop
  depends_on:
    - mongo

# neo4j graph database
neo4j:
  container_name: neo4j
  image: neo4j:4.2
  restart: always
  ports:
    - 7474:7474
    - 7687:7687
  environment:
    # By default, this requires you to login with neo4j/neo4j and change
      the password.
    # You can, for development purposes, disable authentication by
      setting
    - NEO4J_AUTH=none
  volumes:
    # Volume for uploading local files into the DB
    - ./neo4j/neo4j-data:/neo4j-data
    # Volume managed by neo4j
    - neo4j_data:/data

# Superset
superset:
  container_name: superset
  image: apache/superset:latest
  command: bash -c "superset fab create-admin --username admin --
      firstname Superset --lastname Admin --email admin@superset.com --
      password admin && superset db upgrade && superset load_examples &&
      superset init && /usr/bin/docker-entrypoint.sh"
  ports:
    - 8079:8088
  networks:
    - hadoop

timescaledb:
  build: "docker/postgres"
  ports:
    - "7432:5432"
  environment:
```

```
      POSTGRES_MULTIPLE_DATABASES: "mlflow"
      POSTGRES_USER: "postgres"
      POSTGRES_PASSWORD: "postgres"
      MAX_CONNECTIONS: 100
      SHARED_BUFFERS: "1GB"
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data
    networks:
      - hadoop

  minio:
    image: minio/minio:RELEASE.2021-08-05T22-01-19Z
    volumes:
      - ./docker/minio/buckets:/data:consistent
    ports:
      - "9000:9000"
      - "9001:9001"
    environment:
      MINIO_ROOT_USER: "minio"
      MINIO_ROOT_PASSWORD: "minio123"
    command: server --address ":9000" --console-address ":9001" /data
    healthcheck:
      test: [ "CMD", "curl", "-f", "http://localhost:9000/minio/health/live
          " ]
      interval: 30s
      timeout: 20s
      retries: 3
    networks:
      - hadoop

  mlflow:
    build: "docker/mlflow"
    environment:
      MLFLOW_S3_ENDPOINT_URL: "http://minio:9000"
      AWS_ACCESS_KEY_ID: "minio"
      AWS_SECRET_ACCESS_KEY: "minio123"
    ports:
      - "5000:5000"
    command: mlflow server --backend-store-uri postgresql+psycopg2://mlflow
        :mlflow@timescaledb:5432/mlflow --default-artifact-root s3://mlflow
        --host 0.0.0.0 --port 5000
    networks:
      - hadoop
    depends_on:
      - timescaledb
      - minio
  portainer:
```

```
    image: portainer/portainer-ce:2.9.2
    restart: unless-stopped
    command: -H unix:///var/run/docker.sock
    ports:
      - 443:9443
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - dataportainer:/data
    environment:
      TZ: "Europe/Lisbon"

networks:
  hadoop:

volumes:
  hadoop-namenode:
  hadoop-datanode-1:
  hadoop-datanode-2:
  hadoop-datanode-3:
  hadoop-historyserver:
  hue-db:
  hive-metastore:
  mongo_data:
  neo4j_data:
  superset:
  postgres-db-volume:
  dataportainer:
```

Listagem A.1: Docker Compose Plataforma Big Data Analytics.

a.2 Código Python: Algoritmos de deteção de anomalias para conjuntos de dados extremamente desequilibrados.

```
import os
import warnings
from typing import Tuple, Iterator, Callable, Any

import cane
import h2o
import numpy as np
import pandas as pd
import tensorflow as tf
from h2o.automl import H2OAutoML
from keras.layers import Dense, BatchNormalization
from keras.models import Sequential
from sklearn.ensemble import RandomForestClassifier, IsolationForest
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import KFold, StratifiedKFold
from imblearn.over_sampling import SMOTE
from sdv.tabular import GaussianCopula

warnings.filterwarnings("ignore")

TOP_N = 5
SEED = 42
TARGET = 'failure'
CV_FOLDS = 5
N_JOBS = 3
DATASET = 'microsoft'


def kfold_split(
        X: pd.DataFrame,
        y: pd.Series,
        n_splits: int = 5,
        stratify: bool = True
) -> Iterator[Tuple[int, pd.DataFrame, pd.DataFrame, pd.Series, pd.Series
    ]]:
    """
    It splits the data into k folds, and returns the folder id, the
       training and test sets,
    and the training and test labels for each fold.

    :param X: The dataframe of features
    :type X: pd.DataFrame
```

```
    :param y: The target variable
    :type y: pd.Series
    :param n_splits: int, default=5, defaults to 5
    :type n_splits: int (optional)
    :param stratify: If True, the folds are made by preserving the
        percentage of samples for each class,
        defaults to True
    :type stratify: bool (optional)
    :return: The indices of the training and test sets for each fold
    :rtype: Iterator[Tuple[int, pd.DataFrame, pd.DataFrame, pd.Series, pd.
        Series]]
    """
    kf = KFold(n_splits=n_splits, shuffle=True)
    if stratify:
        kf = StratifiedKFold(n_splits=n_splits, shuffle=True)
    for fid, (train_index, test_index) in enumerate(kf.split(X, y)):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        yield fid, X_train, X_test, y_train, y_test


def cross_validation(
        fit_func: Callable[[Any, pd.DataFrame, pd.Series], Any],
        X: pd.DataFrame,
        y: pd.Series,
        n_splits: int = 5,
        stratify: bool = True,
        score_func: Callable[[Any, pd.DataFrame, pd.Series, pd.Series],
            float] = None,
        predict_func: Callable[[Any, pd.DataFrame], pd.Series] = None,
        return_estimator: bool = False,
) -> Iterator[Tuple[int, float, pd.Series, pd.Series]]:
    """
    This function is used to perform cross validation

    :param fit_func: This is the function that will be used to fit the data
        . Must return an estimator
    :type fit_func: Callable[[Any, pd.DataFrame, pd.Series], Any]
    :param X: The dataframe of features
    :type X: pd.DataFrame
    :param y: The series of target values
    :type y: pd.Series
    :param n_splits: The number of folds to use for cross validation,
        defaults to 5
    :type n_splits: int (optional)
    :param stratify: If True, the folds are made by preserving the
        percentage of samples for each class,
```

```
        defaults to True
    :type stratify: bool (optional)
    :param score_func: This is the scoring function to use. If None, a
        simple accuracy score is used, defaults to None
    :type score_func: Callable[[Any, pd.DataFrame, pd.Series, pd.Series],
        float]
    :param predict_func: This is the function that will be used to predict
        the labels. Must return a series with the
        predicted values. If None, the estimator's predict method is used,
            defaults to None
    :type predict_func: Callable[[Any, pd.DataFrame], pd.Series]
    :param return_estimator: If True, the estimator is returned, defaults
        to False
    :type return_estimator: bool (optional)
    :return: An iterator of tuples containing the fold id, the score, the
        test labels, and the predicted labels
        for each fold. If return_estimator is True, the estimator is also
            returned.
    :rtype: Iterator[Tuple[int, float, pd.Series, pd.Series]]
    """

    def _default_scoring(estimator, X_test, y_true, y_pred):
        return np.mean(y_true == y_pred)

    def _default_predict(estimator, X_test):
        return estimator.predict(X_test)

    if score_func is None:
        score_func = _default_scoring

    if predict_func is None:
        predict_func = _default_predict

    estimator = None

    for fid, X_train, X_test, y_train, y_test in kfold_split(X, y, n_splits
        , stratify):
        estimator = fit_func(estimator, X_train, y_train)
        y_pred = predict_func(estimator, X_test)
        score = score_func(estimator, X_test, y_test, y_pred)
        if return_estimator:
            yield fid, score, y_test, y_pred, estimator
        else:
            yield fid, score, y_test, y_pred


def AutoEncoder(data: pd.DataFrame):
```

```
    def layerBuilder(InitalNumberNode, alpha, numberLayers):
        return [np.ceil(InitalNumberNode * alpha ** i) for i in range(
            numberLayers)]

    _layerEncode = layerBuilder(data.shape[1], 0.5, 3)
    _layerDecode = _layerEncode.copy()
    _layerDecode.reverse()

    def AEModel(inputshape, layerEncode, layerDecode):
        model = Sequential()
        model.add(Dense(inputshape, input_dim=inputshape, activation='relu'
            ))
        for i in layerEncode:
            model.add(Dense(i, activation='relu'))
            model.add(BatchNormalization())
        for i in layerDecode[1:]:
            model.add(Dense(i, activation='relu'))
            model.add(BatchNormalization())

        model.add(Dense(inputshape, activation='linear'))
        model.compile(loss='mean_squared_error',
                      optimizer='adam')
        return model

    return AEModel(data.shape[1], _layerEncode, _layerDecode)


def smote_resample(X, y):
    sm = SMOTE(random_state=SEED)
    X_resampled, y_resampled = sm.fit_resample(X, y)
    return X_resampled, y_resampled


def gc_resample(X, y):
    gc = GaussianCopula()
    idx = np.where(y == 0)[0]
    gc.fit(X.iloc[idx].copy())
    x_gc = gc.sample(num_rows=len(X) - len(X.iloc[idx]))
    y_gc = pd.Series(np.ones(len(x_gc)), name=y.name, dtype=int)
    X_resampled = pd.concat([X, x_gc]).reset_index(drop=True)
    y_resampled = pd.concat([y, y_gc]).reset_index(drop=True)
    return X_resampled, y_resampled


def rfc_smote_fit(estimator, X, y):
    x_smote, y_smote = smote_resample(X, y)
```

```
    rfc = RandomForestClassifier(n_estimators=100, random_state=SEED,
        n_jobs=N_JOBS)
    rfc.fit(x_smote, y_smote)
    return rfc


def rfc_gc_fit(estimator, X, y):
    x_gc, y_gc = gc_resample(X, y)
    rfc = RandomForestClassifier(n_estimators=100, random_state=SEED,
        n_jobs=N_JOBS)
    rfc.fit(x_gc, y_gc)
    return rfc


def rfc_fit(estimator, X, y):
    """
    Fit a Random Forest Classifier to the data and return the fitted model

    :param estimator: the model to fit
    :param X: The training data
    :param y: The target variable
    :return: The fitted model
    """
    rfc = RandomForestClassifier(n_estimators=100, random_state=SEED,
        n_jobs=N_JOBS)
    rfc.fit(X, y)
    return rfc


def rfc_scoring(estimator, X_test, y_true, y_pred) -> float:
    """
    Given a Random Forest model, test data, true labels, and predicted
        labels, return the AUC score

    :param estimator: The model estimator
    :param X_test: The test data
    :param y_true: The true labels for the test set
    :param y_pred: The predicted labels
    :return: The AUC score
    :rtype: float
    """
    fpr, tpr, _ = roc_curve(y_true.copy(), y_pred.copy())
    return auc(fpr, tpr)


def aml_smote_fit(estimator, X, y):
    """
```

```
    Fit an AutoML to the data and return the fitted model

    :param estimator: The model to fit
    :param X: The training features
    :param y: The target variable
    :return: The fitted model
    """
    aml = H2OAutoML(
        nfolds=5,
        seed=SEED,
        max_runtime_secs=60 * 10,
    )
    x_smote, y_smote = smote_resample(X, y)
    h2o_frame = h2o.H2OFrame(pd.concat([x_smote, y_smote], axis=1))
    h2o_frame[y.name] = h2o_frame[y.name].asfactor()
    aml.train(x=X.columns.tolist(), y=y.name, training_frame=h2o_frame)
    return aml


def aml_gc_fit(estimator, X, y):
    """
    Fit an AutoML to the data and return the fitted model

    :param estimator: The model to fit
    :param X: The training features
    :param y: The target variable
    :return: The fitted model
    """
    aml = H2OAutoML(
        nfolds=5,
        seed=SEED,
        max_runtime_secs=60 * 10,
    )
    x_gc, y_gc = gc_resample(X, y)
    h2o_frame = h2o.H2OFrame(pd.concat([x_gc, y_gc], axis=1))
    h2o_frame[y.name] = h2o_frame[y.name].asfactor()
    aml.train(x=X.columns.tolist(), y=y.name, training_frame=h2o_frame)
    return aml


def aml_fit(estimator, X, y):
    """
    Fit an AutoML to the data and return the fitted model

    :param estimator: The model to fit
    :param X: The training features
    :param y: The target variable
```

```
    :return: The fitted model
    """
    aml = H2OAutoML(
        nfolds=5,
        seed=SEED,
        max_runtime_secs=60 * 10,
    )
    h2o_frame = h2o.H2OFrame(pd.concat([X, y], axis=1))
    h2o_frame[y.name] = h2o_frame[y.name].asfactor()
    aml.train(x=X.columns.tolist(), y=y.name, training_frame=h2o_frame)
    return aml


def aml_predict(estimator, X):
    """
    It takes an estimator and a dataframe as input, and returns the
        predictions of the estimator on the dataframe

    :param estimator: the model we are using to make predictions
    :param X: the training data
    :return: The predicted values.
    """
    return estimator.predict(h2o.H2OFrame(X)).as_data_frame()["predict"]


def aml_scoring(estimator, X_test, y_true, y_pred) -> float:
    """
    Given an AutoML model, test data, true labels, and predicted labels,
        return the AUC score

    :param estimator: The model estimator
    :param X_test: The test data
    :param y_true: The true labels for the test set
    :param y_pred: The predicted labels
    :return: The AUC score
    :rtype: float
    """
    fpr, tpr, _ = roc_curve(y_true.copy(), y_pred.copy())
    return auc(fpr, tpr)


def ifc_fit(estimator, X, y):
    """
    Fit an isolation forest to the negative class (label = 0) and return
        the fitted model

    :param estimator: The model to fit
```

```
    :param X: The training data
    :param y: The target variable
    :return: The fitted model
    """
    ifc = IsolationForest(n_estimators=100, random_state=SEED, n_jobs=
        N_JOBS)
    idx = np.where(y == 0)[0]
    ifc.fit(X.iloc[idx])
    return ifc


def ifc_scoring(estimator, X_test, y_true, y_pred) -> float:
    """
    Given an Isolation Forest model, test data, true labels, and predicted
        labels, return the AUC score

    :param estimator: The model estimator
    :param X_test: The test data
    :param y_true: The true labels for the test set
    :param y_pred: The predicted labels
    :return: The AUC score
    :rtype: float
    """

    def iForestScoreAUC(y):
        return (-y + 1) / 2

    scoresFull = iForestScoreAUC(estimator.score_samples(X_test))
    fpr, tpr, _ = roc_curve(y_true.values, scoresFull, pos_label=1)
    return auc(fpr, tpr)


def ae_fit(estimator, X, y):
    """
    Fit an autoencoder to the negative class (label = 0) and return the
        fitted model

    :param estimator: The model to fit
    :param X: The training data
    :param y: The target variable
    :return: The model is being returned.
    """
    ae = AutoEncoder(X)
    cp = tf.keras.callbacks.ModelCheckpoint(
        filepath="model/AENormalData",
        mode='min',
        monitor='val_loss',
```

```
        verbose=0,
        save_best_only=True
    )
    early_stop = tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
        min_delta=0.0001,
        patience=10,
        verbose=0,
        mode='min',
        restore_best_weights=True
    )
    idx = np.where(y == 0)[0]
    ae.fit(X.iloc[idx], X.iloc[idx], epochs=200, verbose=0, callbacks=[
        early_stop, cp], validation_split=0.10)
    return ae


def ae_scoring(estimator, X_test, y_true, y_pred) -> float:
    """
    Given an Auto-Encoder model, test data, true labels, and predicted
        labels, return the AUC score
    Note: using MAE for calculating the score. The higher the MAE the
        higher the chance of being anomalous.

    :param estimator: The model estimator
    :param X_test: The test data
    :param y_true: The true labels for the test set
    :param y_pred: The predicted labels
    :return: The AUC score
    :rtype: float
    """
    scoresFull = np.mean(np.abs(X_test - y_pred), axis=1)
    fpr, tpr, _ = roc_curve(y_true.values, scoresFull, pos_label=1)
    return auc(fpr, tpr)


def run_models(X: pd.DataFrame, y: pd.Series, return_estimator: bool =
    False) -> dict:
    """
    It runs the four models, and returns the AUC scores for each fold for
        each model

    :param X: The dataframe containing the features
    :type X: pd.DataFrame
    :param y: The target variable
    :type y: pd.Series
```

```
:param return_estimator: If True, the estimator is returned, defaults
    to False
:type return_estimator: bool (optional)
:return: A dictionary containing the AUC for each fold for each model.
    If return_estimator is True, the estimator
     name is also returned
:rtype: dict
"""

print("Running Random Forest Classifier")
rfc_auc = {}
for ret in cross_validation(rfc_gc_fit, X, y, n_splits=CV_FOLDS,
    score_func=rfc_scoring,
                            return_estimator=return_estimator):
    # print(f"[Random Forest] Fold {fid}: {score}")
    rfc_auc[ret[0]] = ret[1]
    if len(ret) == 5:
        rfc_auc[ret[0]] = ret[1], 'RandomForestClassifier'

print("Running AutoML")
aml_auc = {}
for ret in cross_validation(aml_gc_fit, X, y, n_splits=CV_FOLDS,
    score_func=aml_scoring, predict_func=aml_predict,
                            return_estimator=return_estimator):
    # print(f"[AutoML] Fold {fid}: {score}")
    aml_auc[ret[0]] = ret[1]
    if len(ret) == 5:
        leader = ret[4].leaderboard.as_data_frame()['model_id'][0]
        aml_auc[ret[0]] = ret[1], leader

print("Running Isolation Forest")
ifc_auc = {}
for ret in cross_validation(ifc_fit, X, y, n_splits=CV_FOLDS,
    score_func=ifc_scoring,
                            return_estimator=return_estimator):
    # print(f"[Isolation Forest] Fold {fid}: {score}")
    ifc_auc[ret[0]] = ret[1]
    if len(ret) == 5:
        ifc_auc[ret[0]] = ret[1], 'IsolationForest'

print("Running AutoEncoder")
ae_auc = {}
for ret in cross_validation(ae_fit, X, y, n_splits=CV_FOLDS, score_func
    =ae_scoring,
                            return_estimator=return_estimator):
    # print(f"[AutoEncoder] Fold {fid}: {score}")
    ae_auc[ret[0]] = ret[1]
```

```
        if len(ret) == 5:
            ae_auc[ret[0]] = ret[1], 'AutoEncoder'

    return {
        'rfc': rfc_auc,
        'aml': aml_auc,
        'ifc': ifc_auc,
        'ae':  ae_auc,
    }


def compute_all(data: pd.DataFrame, return_estimator: bool = False) -> dict
    :
    """
    It takes a dataframe, one-hot encodes the machineID column, and then
        runs the models
    :param data: The dataframe to run the models on
    :type data: pd.DataFrame
    :param return_estimator: If True, the estimator is returned, defaults
        to False
    :type return_estimator: bool (optional)
    :return: A dictionary of models and their respective scores.
    :rtype: dict
    """
    _data = cane.one_hot(data, columns_use=["machineID"], column_prefix='
        column')

    predictor_cols = [col for col in _data.columns if col != TARGET]

    x_original = _data[predictor_cols]
    y_original = _data[TARGET]

    x_original = x_original.astype(
        {c: int if c.startswith('MachineID') else float for c in x_original
            .columns}
    )

    return run_models(X=x_original, y=y_original, return_estimator=
        return_estimator)


def compute_single(data: pd.DataFrame, machine_id: int, return_estimator:
    bool = False) -> dict:
    """
    It takes a dataframe and a machine id, and returns a dictionary of the
        results of running the
    models on the dataframe
```

```
    :param data: The dataframe you wish to compute on
    :type data: pd.DataFrame
    :param machine_id: The machine ID you want to run the models on
    :type machine_id: int
    :param return_estimator: if True, return the trained estimator,
        defaults to False
    :type return_estimator: bool (optional)
    :return: a dictionary with the results of the models.
    :rtype: dict
    """
    machines = data.groupby('machineID')
    x = machines.get_group(machine_id)
    x.drop(columns=['machineID'], inplace=True)
    y = x.pop(TARGET)
    return run_models(X=x, y=y, return_estimator=return_estimator)


def compute_top_n(data: pd.DataFrame, top_n: int = 5, return_estimator:
    bool = False) -> Iterator[Tuple[int, dict]]:
    """
    It takes a dataframe and a number as input, and returns a generator of
        tuples with machine ID
    and a dictionary of models and their respective scores.

    :param data: The dataframe containing the data
    :type data: pd.DataFrame
    :param top_n: The number of top machines to return, defaults to 5
    :type top_n: int (optional)
    :param return_estimator: If True, the estimator is returned, defaults
        to False
    :type return_estimator: bool (optional)
    :return: An iterator of tuples with machine ID and a dictionary of
        models and their respective scores.
    :rtype: Iterator[Tuple[int, dict]]
    """
    machines = data.groupby('machineID')
    top_machines_with_failure = (
        machines
            .agg(count=pd.NamedAgg(column=TARGET, aggfunc='value_counts'))
            .loc[(slice(None), 1), :]
            .reset_index(TARGET, drop=True)
            .sort_values(by='count', ascending=False)
    )

    for machine in top_machines_with_failure.index[:top_n]:
```

```
        yield int(machine), compute_single(data=data, machine_id=int(
            machine), return_estimator=return_estimator)


def save_results(results: list, dirpath: str) -> None:
    """
    It saves the results of the cross-validation in a csv file

    :param results: list of tuples (machineID, model, foldID, AUC)
    :type results: list
    :param dirpath: the path to the directory where the results will be
        saved
    :type dirpath: str
    """
    if DATASET == "microsoft":
        results_per_fold = pd.DataFrame(results, columns=['machineID', '
            model', 'foldID', 'AUC'])
        results_per_fold.to_csv(os.path.join(dirpath, f'{DATASET}_auc_fold.
            csv'), index=False)

        results_per_fold.loc[
            ~results_per_fold['model'].isin(['RandomForestClassifier', '
                IsolationForest', 'AutoEncoder']),
            'model'
        ] = 'AutoML'

        results_per_machine = (
            results_per_fold
            .groupby(['machineID', 'model'])
            .agg(AUC_mean=('AUC', 'mean'), AUC_median=('AUC', 'median'))
            .reset_index()
        )

        results_per_machine.to_csv(os.path.join(dirpath, f'{DATASET}_auc.
            csv'), index=False)
    else:
        results_per_fold = pd.DataFrame(results, columns=['model', 'foldID'
            , 'AUC'])
        results_per_fold.to_csv(os.path.join(dirpath, f'{DATASET}_auc_fold.
            csv'), index=False)

        results_per_fold.loc[
            ~results_per_fold['model'].isin(['RandomForestClassifier', '
                IsolationForest', 'AutoEncoder']),
            'model'
        ] = 'AutoML'
```

```
        results_per_machine = (
            results_per_fold
                .groupby(['model'])
                .agg(AUC_mean=('AUC', 'mean'), AUC_median=('AUC', 'median')
                    )
                .reset_index()
        )

        results_per_machine.to_csv(os.path.join(dirpath, os.path.join(
            dirpath, f'{DATASET}_auc.csv')), index=False)


def process_microsoft_dataset():
    telemetry = pd.read_csv('data/PdM_telemetry.csv', on_bad_lines='skip').
        drop_duplicates(ignore_index=True)
    failures = pd.read_csv('data/PdM_failures.csv', on_bad_lines='skip')
    data = pd.merge(telemetry, failures, on=['datetime', 'machineID'], how=
        'left')
    data[TARGET] = (
        data[TARGET]
        .replace(to_replace='comp.', value=1, regex=True)
        .fillna(0)
        .astype(int)
    )
    data.drop(columns=['datetime'], inplace=True)
    return data




def main():
    process_dataset = {
        'microsoft': process_microsoft_dataset,
    }

    try:
        data = process_dataset[DATASET]()
    except KeyError:
        raise KeyError(f'The dataset {DATASET} is not available')

    print(f"Total data size: {data.shape[0]}")

    normal = data[data[TARGET].values == 0]
    failure = data[data[TARGET].values == 1]

    print(f"\tNormal: {normal.shape[0]}")
```

```
    print(f"\tFailure: {failure.shape[0]}")

    results = []
    extend_algo_name = {
        'rfc': 'RandomForestClassifier',
        'aml': 'AutoML',
        'ifc': 'IsolationForest',
        'ae':  'AutoEncoder',
    }

    h2o.init()

    if DATASET == "microsoft":
        print(f"\nTop {TOP_N} Machines")
        for mid, ret in compute_top_n(data=data, top_n=TOP_N,
            return_estimator=True):
            print(f"Machine {mid}")
            for algo in ret.keys():
                print(f"\t{extend_algo_name[algo]}")
                for fold, info in ret[algo].items():
                    if len(info) == 2:
                        print(f"\t\tFold {fold}")
                        print(f"\t\t\tAUC: {info[0]}")
                        print(f"\t\t\tModel: {info[1]}")
                        results.append((mid, info[1], fold, info[0]))
                    else:
                        print(f"\t\tFold {fold}")
                        print(f"\t\t\tAUC: {info}")
                        results.append((mid, {extend_algo_name[algo]}, fold
                            , info))


    h2o.cluster().shutdown()


if __name__ == "__main__":
    main()
```

Listagem A.2: Código Python: Algoritmos de deteção de anomalias para conjuntos de dados extremamente desequilibrados .