NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

Nelson Nascimento de Freitas

Licenciado em Engenharia Eletrotécnica e de Computadores

Desenho e desenvolvimento de uma camada de integração para sistemas ciber-físicos de produção na cloud

MESTRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

Universidade NOVA de Lisboa

Novembro, 2021

DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA E DE COMPUTADORES

Desenho e desenvolvimento de uma camada de integração para sistemas ciber-físicos de produção na cloud

Nelson Nascimento de Freitas

Licenciado em Engenharia Eletrotécnica e de Computadores

Orientador: André Dionísio Bettencourt da Silva Parreira Rocha,
Professora, Universidade NOVA de Lisboa

Coorientadores: Duarte José Marques Alemão,
Mestre, Universidade NOVA de Lisboa

Júri:

Presidente: Luís Bernardo,
Professor, Universidade NOVA de Lisboa

Arguentes: Ricardo Peres,
Professor, Universidade NOVA de Lisboa

Orientador: André Dionísio Bettencourt da Silva Parreira Rocha,
Professora, Universidade NOVA de Lisboa

MESTRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

Universidade NOVA de Lisboa
Novembro, 2021

Desenho e desenvolvimento de uma camada de integração para sistemas ciber-físicos de produção na cloud

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Agradecimentos

Gostaria de agradecer, primeiramente, à Universidade Nova de Lisboa e ao Departamento de Engenharia Eletrotécnica, pois permitiram-me seguir o meu sonho e desenvolver-me a nível pessoal e profissional, rodeado de pessoas incríveis cujos feitos nos levam a tentar fazer mais e melhor.

Um agradecimento especial ao meu orientador André Rocha e co-orientador Duarte Alemão, cuja disponibilidade permanente de me ajudarem e apontarem os meus esforços no caminho certo permitiram desenvolver este trabalho incrível.

Queria agradecer também à equipa da Introsys por terem colaborado no desenvolvimento da dissertação e permitido que todos os testes e validação fossem realizados na empresa, num ambiente industrial.

Queria também agradecer a todos os meus amigos e colegas de curso que me ajudaram sempre que precisei, a nível académico e pessoal, que contribuíram para chegar onde estou hoje, em especial, a uma amiga muito querida para mim, Bruna Cruz.

Por fim agradeço a toda a minha família, em especial aos meus pais, que me permitiram chegar onde cheguei, cujo esforço, dedicação e sacrifício foram iguais ou maiores que os meus, um grande obrigado.

Resumo

Os ambientes industriais são sistemas heterogéneos que criam desafios de interoperabilidade, limitando o desenvolvimento de sistemas capazes de trabalhar colaborativamente do ponto de vista do *hardware* e *software*. Desta forma, o trabalho proposto tem como objetivo apresentar uma solução de interoperabilidade baseada em eventos, de forma a reduzir a complexidade de integração. É proposta uma arquitetura baseada em *publish/-subscribe*, implementada com o *Apache Kafka*. A solução proposta foi implementada em duas células robóticas, constituídas por *hardware* diferentes, na indústria automóvel, permitindo testar a integração de diferentes componentes. Os dados do consumo energético e da produção, são enviados para uma base de dados *Postgres* onde uma interface gráfica permite ao operador monitorizar as diferentes células em relação ao consumo energético e produtividade. Os resultados são promissores, devido à habilidade do sistema integrar ferramentas de diferentes vendedores e diferentes tecnologias. Esta abordagem, possibilita ainda o desenvolvimento de sistemas mais sustentáveis utilizando soluções mais avançadas, como programas de produção, de forma a reduzir o consumo energético e aumentar a produtividade.

Palavras-chave: Apache Kafka; eficiência energética; produção; Indústria 4.0; interoperabilidade; integração; manufactura inteligente; sustentabilidade

Abstract

Industrial environments are heterogeneous systems that create challenges of interoperability limiting the development of systems capable of working collaboratively from the point of view of machines and software. Thus, the proposed work aims to present an interoperable solution based on events to reduce the complexity of integration. A publish/subscribe-based architecture is proposed, where the instantiation is based on Apache Kafka. The proposed solution was implemented in two robotic cells in the automotive industry, constituted by different hardware, which allowed testing the integration of different components. The energy consumption and production data was then sent to a Postgres database where a graphical interface allowed the operator to monitor the performance of each cell regarding energy consumption and productivity. The results are promising due to the system's ability to integrate tools from different vendors and different technologies. Furthermore, it allows the possibility to use these developments to deliver more sustainable systems using more advanced solutions, such as production scheduling, to reduce energy consumption and boost productivity.

Keywords: Apache Kafka; cyber-physical production systems; energy efficiency; productivity; Industry 4.0; interoperability; smart manufacturing; sustainability

Índice

Lista de Figuras

Lista de Tabelas

Siglas

API Application Programming Interface.

CM Cloud Manufacturing.

CPPS Cyber-Physical Production Systems.

CPS Cyber-Physical Systems.

HMI Human-Machine Interface.

HTTP Hypertext Transfer Protocol.

IoT Internet of Things.

IP Internet Protocol.

JDBC Java Database Connectivity.

MDPI Multidisciplinary Digital Publishing Institute.

MES Manufacturing Execution System.

SQL Structured Query Language.

URL Uniform Resource Locators.

Capítulo 1

Introdução

Neste capítulo será abordado um enquadramento que contempla as mudanças industriais que aconteceram ao longo do tempo e a motivação da dissertação. Será também descrita a organização do documento.

1.1 Enquadramento e motivação

Desde o início da humanidade, o ser humano sempre utilizou ferramentas e desenvolveu técnicas de forma a superar cada dificuldade que o mundo apresentava. No entanto tudo isto era muito rudimentar e produzido localmente. À medida que a população foi crescendo, as sociedades desenvolveram-se e novas maneiras de pensar emergiram, assim, também a produção necessitava de uma revolução. Era necessário produzir mais, com uma maior rapidez e qualidade e quando foi desenvolvida a máquina a vapor, uma oportunidade para isso surgiu utilizando-a para a produção têxtil, chamou-se a esta viragem histórica, a primeira revolução industrial.

A segunda revolução industrial, surgiu com a utilização de novos tipos de energia como a eletricidade e o petróleo. Além disso foi popularizado o uso de linhas de montagem, o que concretizou a produção em massa de veículos.

Chega-se à terceira revolução industrial, onde a digitalização e a era de informação tomam conta da indústria, passa-se de sistemas analógicos para digitais e os primeiros robôs aparecem no contexto industrial.

No entanto, da história passamos para o presente da nova e importante revolução que mais uma vez a indústria está a sofrer, precisando de se adaptar, mais uma vez, aos novos desafios. Esta revolução visiona um sistema de produção modular e eficiente em que os produtos controlam o processo de produção, sendo necessário uma extração em tempo real dos dados em ambiente industrial e a disponibilização dos mesmos a outros

serviços. Um método que se enquadra nesta abordagem é a utilização de um modelo *publish/subscribe* numa *cloud*, permitindo que os dados sejam recolhidos e apresentados a outras aplicações, com o objetivo de produzir pequenos lotes de produtos personalizados sem sacrificar as vantagens económicas da produção em massa. A esta quarta revolução deu-se o nome de *Indústria 4.0*.

O objetivo desta dissertação é utilizar alguns dos pilares da indústria 4.0, nomeadamente a *Cloud*, *Integração de sistemas* e conceitos como *sistemas ciberfísicos* para produzir uma camada de *software* capaz de interligar diferentes empresas e sistemas de forma a melhorar a sua eficiência.

Nesta dissertação é demonstrado como a integração dos sistemas com a camada de *software* pode ser executada, inicialmente com uma proposta de arquitetura e de seguida com várias demonstrações, tanto em laboratório como num ambiente industrial real, de como é possível implementar e colher os benefícios da sua implementação.

1.2 Organização da dissertação

O estado de arte está organizado em sete capítulos que serão descritos de seguida, onde se inclui a introdução.

No capítulo 1, é descrita a motivação que levou à decisão deste tema para a dissertação. É explicado de uma forma muito geral, a importância da indústria e do seu processo evolutivo.

No capítulo 2, é feita uma revisão do estado de arte, bem como são descritas algumas noções de conceitos que são necessários ter em conta para a compreensão do estado atual da indústria bem como alguns conceitos importantes como *Cyber-Physical Systems* e *Cloud Manufacturing*.

No capítulo 3, é descrita uma ferramenta bastante importante para a implementação dos sistemas descritos, nomeadamente o *broker*. É feita uma comparação entre diversos *brokers* de forma a perceber qual se enquadra melhor no objetivo da dissertação.

É descrito no capítulo 4, a arquitetura desenvolvida onde é explicado os diferentes componentes da mesma, em que consistem e a ideia por detrás de cada um.

No capítulo 5, encontra-se uma descrição da implementação da arquitetura desenvolvida, é explicada a escolha do *message broker*, como ele foi instanciado e a implementação das diferentes demonstrações desenvolvidas.

O capítulo 6 apresenta uma discussão de todo o processo desenvolvido. São realçadas as limitações das demonstrações apresentadas e da arquitetura, sendo por fim apresentado o artigo realizado sobre a mesma onde se incluem as demonstrações.

No capítulo 7, é por fim referida a conclusão, onde é apresentado um sumário do estado atual da indústria, destacando os pontos essenciais da mesma. É descrita a importância de um *message broker* instanciado em *Cloud* para o novo contexto industrial. Por fim é realçada a arquitetura desenvolvida e as demonstrações realizadas.

Capítulo 2

Estado de Arte

Em todas as revoluções industriais uma mudança radical na forma como os processos eram feitos, mudou a indústria de forma a ser mais eficiente. Com a primeira revolução industrial a utilização do carvão e água, permitiu que a produção têxtil fosse realizada a uma velocidade muito maior e com melhor qualidade, reduzindo os preços e melhorando os rendimentos da população [Mohajan 2019a].

A segunda revolução industrial permitiu que com a utilização da eletricidade e o petróleo, um novo bem fosse comercializado em massa, o automóvel, sendo o automóvel o bem pioneiro nas linhas de montagem que por sua vez foram expandidas para praticamente todos os bens comerciáveis [Mohajan 2019b].

A terceira revolução industrial, permitiu que a que a eletrónica entrasse na indústria e com isso o inicio da digitalização da informação. Os primeiros robôs industriais aparecem e processos começam a ser automatizados. A especialização de várias indústrias e a verticalização dos processos começa também a tomar forma com a terceira revolução [Mowery 2009].

A quarta revolução é construída em cima da terceira, com o objetivo de processar as informações digitalizadas durante a terceira revolução, de forma a otimizar todos os processos e ser possível uma personalização dos produtos sem sacrificar a velocidade de produção. Novos desafios surgem, como a integração de todos os diferentes sistemas e a segurança de ter todos os sistemas conectados em rede [Schwab 2016].

Para que seja possível concretizar todos os objetivos da indústria 4.0 mencionados, como um sistema de produção modular ou produtos facilmente personalizáveis, é necessário que diversas tecnologias trabalhem de forma colaborativa. Será referido neste capítulo como essas tecnologias trabalham e a sua relação, será também abordado em maior detalhe algumas dessas tecnologias como *Cyber-Physical Systems* e *Cloud Manufacturing*. Será ainda realizada uma descrição mais aprofundada da indústria 4.0 e os conceitos

que a constituem.

2.1 Indústria 4.0

Neste momento, a indústria e a sociedade atravessam uma quarta revolução industrial, denominada de indústria 4.0. Esta revolução, ao contrário das últimas três, não se limita à própria empresa, revolucionando a conectividade entre diversos elementos como sensores, maquinaria, produtos e até mesmo sistemas tecnológicos. Estes deixam de estar limitados a uma só entidade permitindo conectar empresas ou serviços de uma forma muito mais próxima e em tempo real. Esta nova abordagem é essencial para lidar com novos e complexos desafios que a sociedade apresenta à medida que evolui. Problemas como [Lasi et al. 2014]:

- **Personalização de produtos**: Os produtos devem poder ser adaptados às necessidades ou preferências dos compradores.
- **Flexibilidade**: Devido aos novos requisitos de estrutura, é necessária uma maior flexibilidade no desenvolvimento do produto, especialmente durante a sua produção.
- **Descentralização**: De forma a lidar com a personalização de produtos, é necessário realizar decisões de processo rapidamente. Como tal, é necessária uma redução das hierarquias organizacionais.
- **Eficiência dos recursos**: Uma maior eficiência do uso dos recursos utilizados, desde o transporte ao desperdício em cada produto produzido.

A indústria 4.0 possui diversas abordagens para lidar com estes problemas, conhecidas como os nove pilares do desenvolvimento tecnológico (figura 2.1). Estas tecnologias emergentes estão gradualmente a mudar o mundo da indústria, de forma a resolver os problemas descritos anteriormente [Rüßmann 2015].

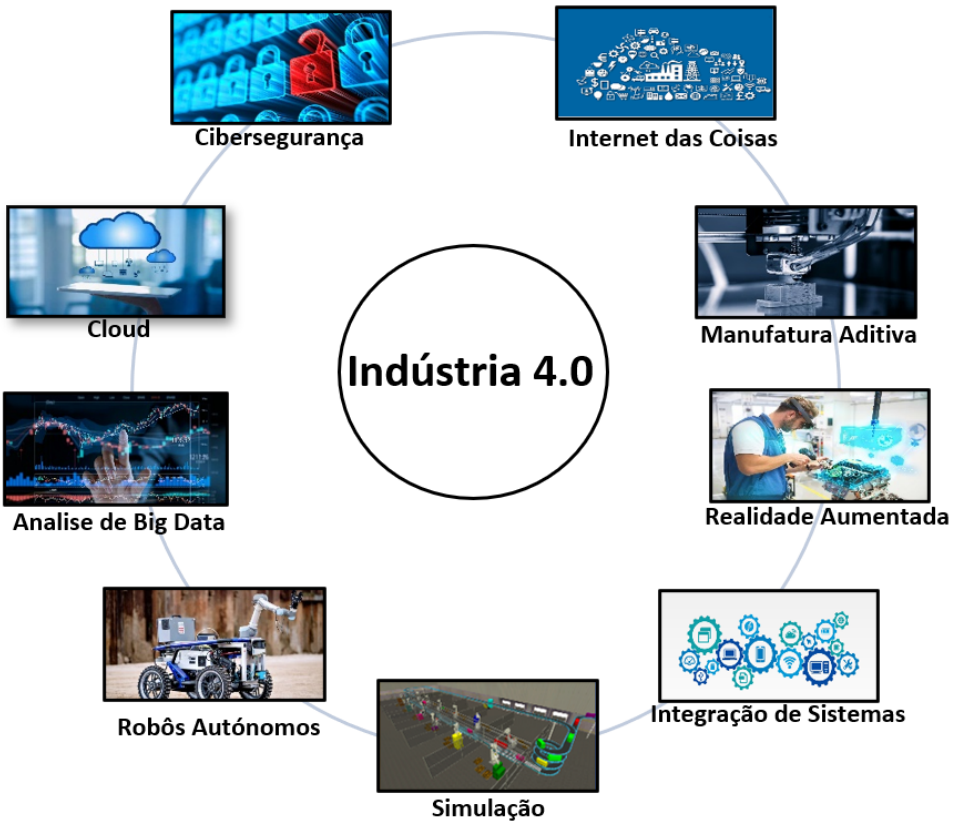

Figura 2.1: Os nove pilares de desenvolvimento tecnológico associados à indústria 4.0 (imagem adaptada de [Rüßmann 2015]).

Cada um destes pilares tem toda uma imensa complexidade e diversos subtemas que apesar de nem todos serem abordados com muita profundidade ao longo desta dissertação é importante para o entendimento da indústria 4.0 desenvolver um pouco mais de cada um dos pilares [AVANGARD Project. 2020; Rüßmann 2015]:

- **Análise de *Big Data***: Com a recente capacidade de recolha de dados através de melhores sensores ou câmaras e com tecnologias como redes neuronais para processar estes dados, é possível agora otimizar cada vez mais e melhor os processos de fabrico.

- **Robôs Autónomos**: Cada vez mais os robôs têm vindo a ser aperfeiçoados, tornando-os mais flexíveis, autónomos e cooperativos. Eventualmente será possível integrá-los de modo a cooperar lado a lado com um operador e aprender com ele.

- **Simulação**: Simulações de produtos, materiais e processos de produção, são utilizadas na indústria há bastante tempo, mas o objetivo é que sejam utilizadas de forma mais extensiva, simulando em tempo real e com valores reais, em paralelo com a produção. Isto pode ser usado para otimizar a maquinaria e parâmetros de forma a maximizar a eficiência.

- **Integração de Sistemas**: Atualmente a maioria dos sistemas não estão completamente integrados. Departamentos internos como os de engenharia, produção ou serviços raramente têm os seus sistemas interligados, bem como a nível externo nomeadamente empresas de logística, produção e clientes. A completa integração de todos estes sistemas possibilita interações mais coerentes, rápidas e autónomas.

- **Internet das Coisas**: Nos dias de hoje os sensores e máquinas estão organizados em sistemas de pirâmide, com limitada capacidade de processamento. No entanto, com a internet das coisas mais e novos dispositivos com maior capacidade de processamento estarão interligados, isto permitirá que os dispositivos interajam e comuniquem de forma descentralizada, realizando decisões em tempo real.

- **Cibersegurança**: Atualmente sistemas de logística e de produção estão muitas vezes desconectados da rede, com a necessidade de diversos sensores e maquinarias estarem interconectados, a segurança cibernética passa a ser fundamental. Melhores e mais avançados métodos de autenticação bem como um contacto próximo com empresas de cibersegurança, tornam-se essenciais para um ambiente seguro na rede empresarial.

- ***Cloud***: Um pouco por todo o mundo as empresas já utilizam aplicações em *cloud*, no entanto é possível expandir esta funcionalidade na indústria 4.0 deixando as barreiras físicas das empresas de parte. Pode ser bastante benéfico utilizar dados em tempo real de diversas empresas, facilitando a otimização em tempo real de sistemas de produção.

- **Manufatura Aditiva**: Sistemas de impressão 3D têm começado a fazer o seu aparecimento na indústria, estes sistemas têm a capacidade de facilmente produzir protótipos ou pequenos lotes de produtos personalizados. Estes métodos podem ser adaptados para a construção de grandes estruturas como veículos, isto permite fabricar produtos complexos e leves, além de permitir mais facilmente uma produção descentralizada.

- **Realidade Aumentada**: A realidade aumentada pode ser um sistema de suporte essencial ao treino ou melhoria de eficiência dos trabalhadores. Pode ser utilizada para demonstrar aos trabalhadores instruções em tempo real, melhorando as decisões e procedimentos em contexto laboral.

Surge desta forma a ideia de mapear o sistema físico para um sistema cibernético, podendo existir uma análise de dados e uma tomada de decisões de forma muito mais rápida, autónoma e eficaz. A este tipo de sistemas dá-se o nome de *Cyber-Physical Systems*.

2.2 Cyber-Physical Systems

Quando algumas ideias e conceitos da indústria 4.0 se juntam, tais como melhores e mais avançados sensores, sistemas de aquisição de dados, internet das coisas e computação distribuída, nasce um novo e revolucionário conceito designado por *Cyber-Physical System* (CPS). O CPS é um sistema de computação colaborativo que possui uma conexão extensiva com o mundo físico, serviços de processamento de dados e podendo ainda usufruir de uma extensa base de dados. Este é um conceito fundamental para as fábricas do futuro também designadas de *Smart Factories* [Weyer et al. 2016]. Diversos ramos empresariais veem os benefícios da utilização e integração de um CPS nos seus processos de fabrico, no entanto a concretização de um CPS tem desafios inerentes que precisam de ser superados antes de colher os benefícios [Müller et al. 2018]. Por exemplo, a integração entre o espaço cibernético e o espaço físico, em que os eventos que acontecem no espaço físico precisam de ser corretamente descritos no espaço cibernético e os comandos de produção definidos pelo espaço cibernético necessitam de ser comunicados ao espaço físico. Estas ações têm de ser executadas de forma rápida e precisa. É importante ainda considerar o problema de diversos componentes produzirem elevados volumes de informação dentro de um curto período de tempo e as infraestruturas empresariais precisam de ser capazes de processar esse volume de dados [Weyer et al. 2016].

De forma a superar as dificuldades existentes, diversas arquiteturas já foram estudadas e publicadas ao longo do tempo, apresenta-se assim uma arquitetura modelo [Liu e Jiang 2016] baseada em três camadas em que cada uma é responsável por diferentes aspetos essenciais num CPS (figura 2.2), bem como uma descrição especifica de cada camada e como operam:

- ***Physical connection layer*** - Esta camada é responsável pela recolha de dados de vários sensores e equipamentos de medição bem como a transmissão de parâmetros à maquinaria ou atuadores ao nível da produção e distribuição no ambiente de manufatura. A escolha destes componentes deve ser considerada tendo em conta diversos fatores como, protocolos, processamento, localização, distância e armazenamento.

- ***Middleware layer*** - O objetivo da camada de *Middleware* é transferir os dados recolhidos de diversos componentes para o servidor central, de modo a estes dados poderem ser analisados e transferir as instruções e parâmetros de produção da *Computation layer* ou aplicações externas para diferentes controladores. Esta camada tem de ser capaz de receber diferentes tipos de formatos de dados de todo o tipo de componentes além de traduzir os comandos dados pela *Computation layer* e aplicações externas.

- ***Computation layer*** - A camada é composta por modelos específicos, algoritmos e ferramentas de forma a analisar os padrões e nuances dos dados recolhidos por diferentes sensores, aparelhos de medida, maquinaria, sistemas de planeamento,

execução ou logística, entre qualquer outro tipo de entidade que possa fornecer dados que permitam ter uma melhor compreensão de todo o processo industrial. Nesta camada existem dois tipos de processamento de *big data* que necessitam de ser referidos, *batch computing* de forma a processar grandes volumes de dados e *stream processing* capaz de processar dados em quase tempo real.

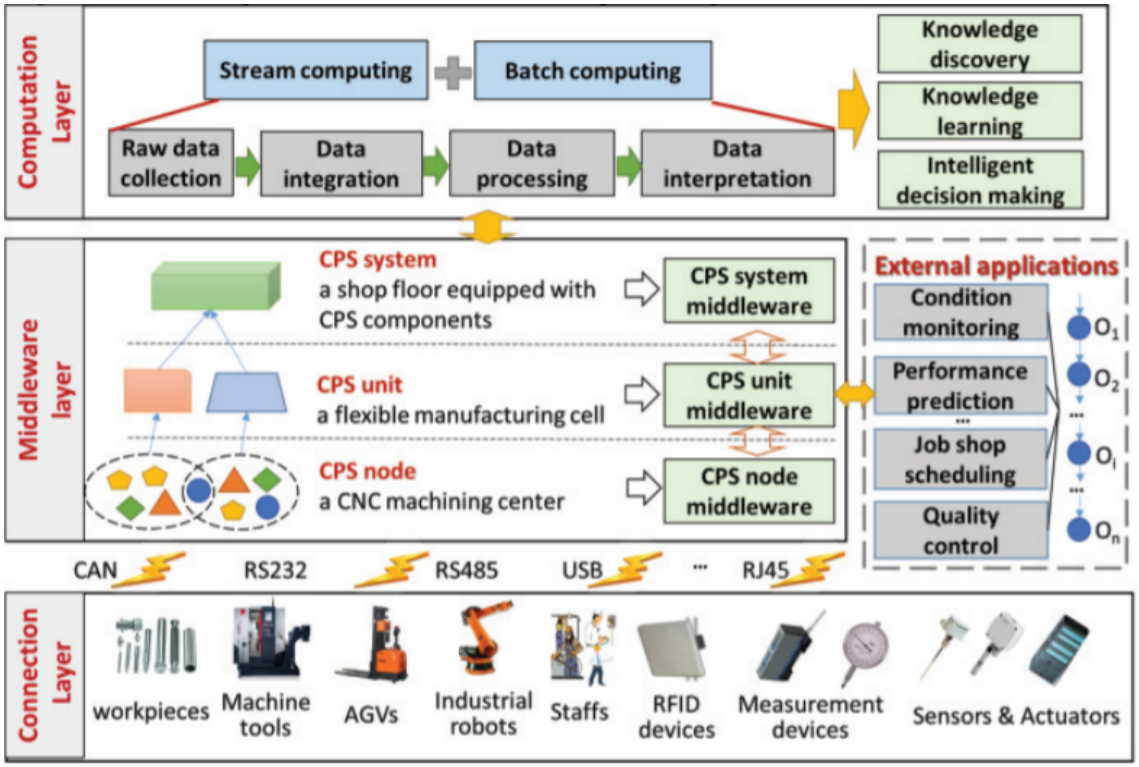

Figura 2.2: Arquitetura de *Cyber-Physical Systems* (retirado de [Liu e Jiang 2016]).

Esta ideia pode no entanto ser expandida, integrando diferentes CPS é possível criar um sistema de monitorização que beneficia dos conhecimentos adquiridos por todos os CPS integradas e utiliza este conhecimento para uma monitorização cada vez mais eficiente. Ao ser aplicado este conceito a sistemas de produção obtêm se um *Cyber-Physical Production Systems*.

2.2.1 Cyber-Physical Production Systems

O conceito de CPPS aparece, quando são aplicadas CPS num ambiente industrial, o que concede a habilidade de abstrair qualquer recurso de uma linha de produção com a sua representação lógica, ganhando a capacidade de lidar com complexas e avançadas funcionalidades [Rocha et al. 2019]. Este sistema trata-se de uma composição de vários recursos humanos, equipamentos de produção e produtos, estabelecidos numa ou várias interfaces CPS. O controlo e monitorização das diferentes interfaces CPS bem como o acesso às bases de dados de conhecimento (*knowledge database*) produzido por operadores,

equipamentos e produtos ao longo do seu ciclo de vida é uma das principais funções do novo sistema CPPS (figura 2.3) [Ribeiro 2017; Ribeiro e Bjorkman 2018].

Outra grande vantagem da utilização de CPPS é facto de ser possível abstrair a fábrica no processo de relação interempresarial, permitindo a implementação de processos distributivos e cooperativos. Esta abordagem é um método eficaz de desenvolver sistemas de manufatura flexíveis, reconfiguráveis e capazes de lidar perturbações externas[Rocha et al. 2019].

Um aspeto importante a considerar com a introdução das CPPS é o paralelismo e cooperação destes sistemas com o operador, dado que o conhecimento humano não pode ser totalmente formalizado e transferido, torna-se essencial para o melhor resultado possível, uma colaboração próxima com a entidade humana [Ribeiro 2017; Ribeiro e Bjorkman 2018].

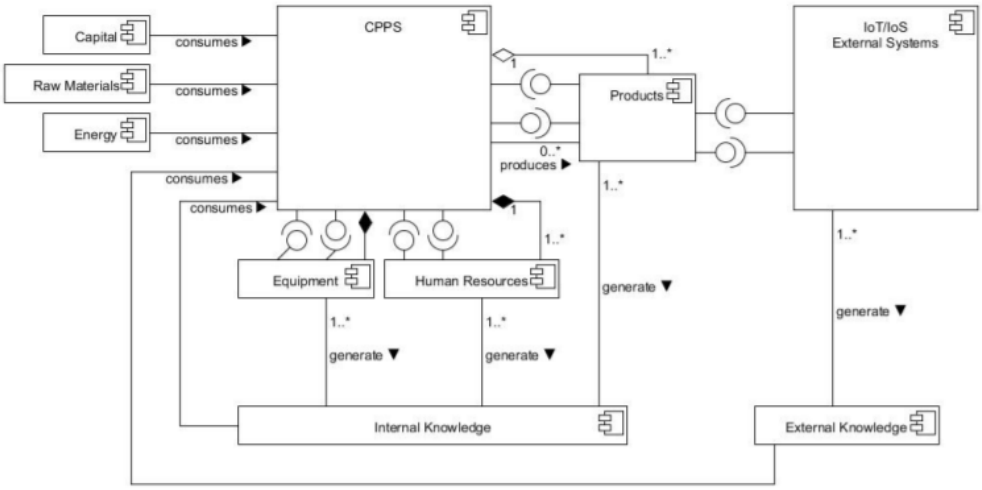

Figura 2.3: Diagrama UML de um *Cyber-Physical Production Systems* (retirado de [Ribeiro 2017]).

Quando começa a ser considerado um controlo centralizado e a partilha de recursos, estes possuem inúmeros desafios, relacionados com a sua eficácia e rapidez. De forma a fazer face a estes desafios, surgiu a ideia de tecnologia como um serviço. Com base nesta e outras ideias, o conceito de *Cloud Manufacturing* surgiu de forma a superar os desafios apresentados.

2.3 Cloud Manufacturing

Na indústria atual existem diversos problemas nomeadamente com a forma de gestão e partilha de recursos, sejam estes físicos ou digitais. Fala-se assim de problemas como [Zhang et al. 2014]:

- **O modelo de serviços**: a falta de um controlo centralizado de operações e serviços, além da sua eficiência, qualidade e rapidez serem raramente garantidos.
- **Partilha e atribuição de recursos**: os recursos raramente são partilhados e quando o são existem problemas como a otimização e falta de dinâmica. Isto incorre num problema fundamental em que os clientes têm uma necessidade personalizada, mas a empresa não possui recursos para o fazer.
- **A livre circulação e cooperação entre empresas**: Mesmo quando existe uma partilha ativa dos recursos (nomeadamente digitais), estes são limitados pelas barreiras físicas da empresa ou da rede. A ideia é uma livre partilha de recursos entre diferentes tipos de empresas de diversos ramos.

Com o objetivo de superar estas dificuldades foi criado o conceito de *Cloud manufacturing* (CM). CM refere-se a um modelo de manufatura avançado que utiliza conceitos e tecnologias tais como *cloud computing*, sistema *IoT*, a virtualização do espaço físico (*CPS*) e o conceito de tecnologia como um serviço, com o objetivo de adaptar e transformar a indústria de produção tradicional em produção por serviços. Esta visão é composta por três grupos (figura 2.4): os consumidores, os fornecedores das aplicações em *cloud* e por fim os fornecedores dos recursos físicos. Torna-se assim a integração multi-empresarial o principal ponto chave para uma CM de sucesso [Zhong et al. 2017].

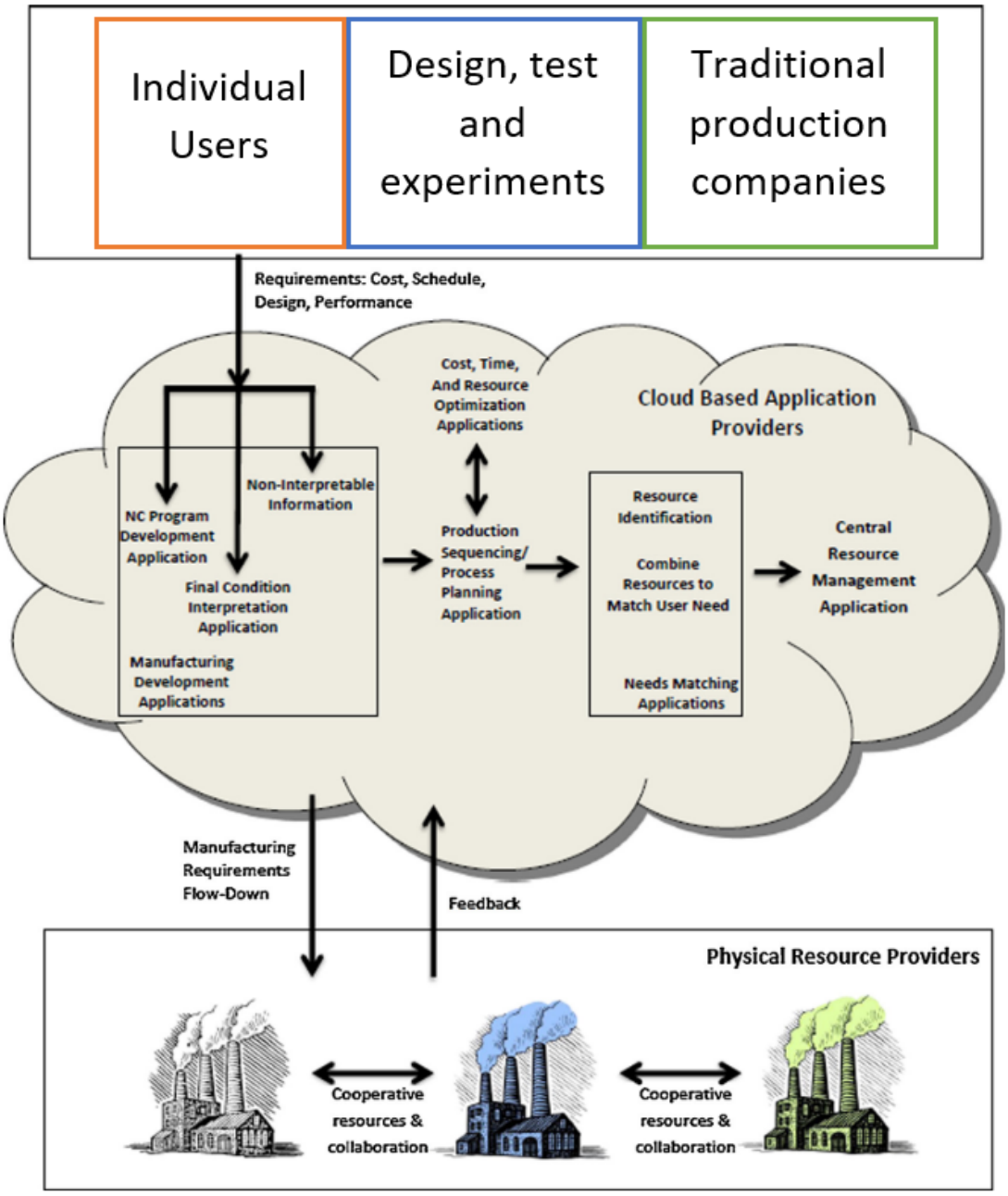

Figura 2.4: Visão conceptual do modelo de *Cloud manufacturing* (imagem adaptada de [Wu et al. 2013]).

Outro ponto essencial a ter em conta é que esta ideia torna o consumidor o ponto central do processo industrial [Wu et al. 2013], dado que todos os serviços passam a ser executados quando solicitados por um consumidor, assim todo o sistema original de produção fixos e pouco flexíveis, são agora substituídos ou integrados no conceito das CM (figura 2.5).

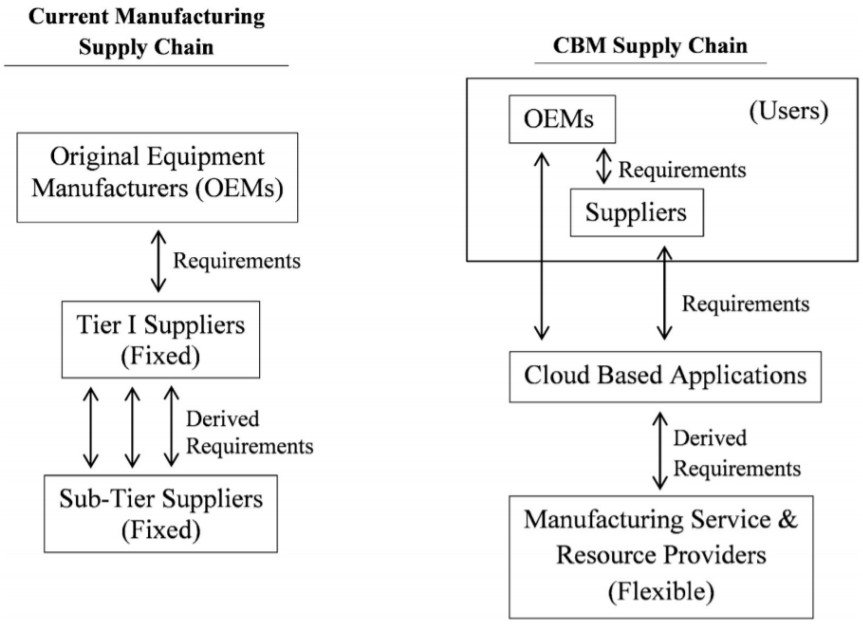

Figura 2.5: Comparação entre a indústria de produção tradicional e o conceito de *Cloud manufacturing* (imagem retirada de [Assari et al. 2018]).

Para realizar todas estas funcionalidades eficazmente é necessário existir uma arquitetura capaz de as satisfazer, como pode ser observada a arquitetura na figura 2.6. Cada camada é responsável por serviços ou comunicações específicas:

- ***Resource layer***: Esta camada proporciona todos os recursos e funcionalidades de manufatura que podem ser chamados como serviços por um consumidor.

- ***Perception layer***: É responsável por todos os sensores que analisam o que acontece no mundo físico e envia esses dados pela rede.

- ***Resource virtualization layer***: Compete a esta camada a virtualização dos recursos e funcionalidades de manufatura.

- ***Cloud service layer***: Esta camada possui duas funções principais, a primeira é *manufacturing cloud service* (MCS) em que dispõe ao consumidor todos os recursos e funcionalidades da manufatura como um serviço. A segunda função é a capacidade de gerir os acessos, invocar serviços, pesquisa, agendamentos, avaliações, entre outros processos de gestão industrial.

- ***Application layer***: Possui como objetivo a integração de diferentes sistemas de manufatura com a CM.

- ***Portal layer***: Dispõem diversas interfaces de interação pessoa-máquina, utilizadas para acessos aos serviços de manufatura e CM.

- ***Enterprise cooperation application layer***: Camada responsável pela cooperação entre diferentes serviços, como comércio, negócios, operações de manufatura e projetos colaborativos.

- ***Knowledge layer***: Uma camada transversal, responsável por proporcionar a todas as outras camadas no domínio do conhecimento, isto é valido para modelos, processos ou produção.

- ***Cloud security layer***: Proporciona transversalmente diferentes mecanismos, arquiteturas e estratégias de segurança.

- ***Wider internet layer***: Assegura transversalmente a comunicação com todo o ambiente CM.

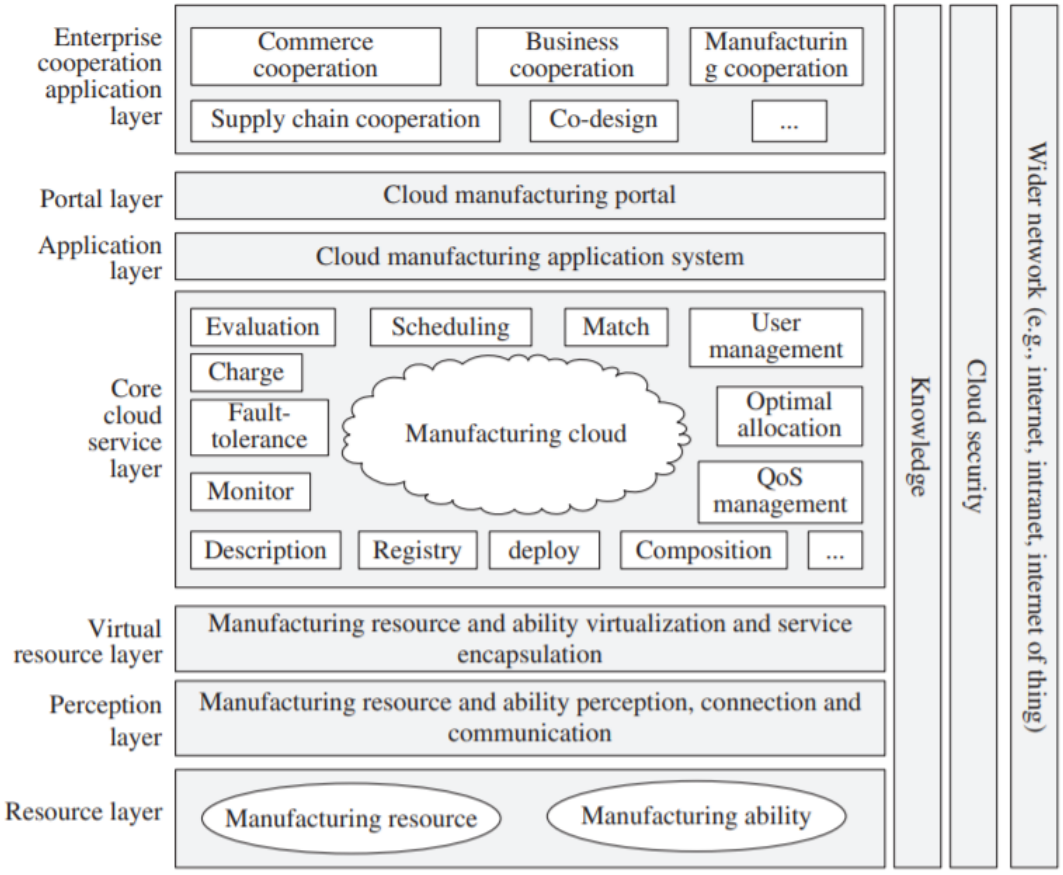

Figura 2.6: Arquitetura de uma *Cloud manufacturing* (imagem retirada de [Tao et al. 2011]).

Com as diferentes tecnologias necessárias para o desenvolvimento de uma CM a emergirem rapidamente, pode-se estar perante uma solução para a mudança necessária de paradigmas de manufatura (como demonstrado em 2.1) partilhando e otimizando a

utilização de recursos, aliada a uma manufatura por serviços, descentralizada, expansível e sustentável.

Para que seja possível a comunicações entre todas as camadas da arquitetura é essencial a existência de um serviço capaz de assegurar o envio e a distribuição das mensagens, sendo por norma assegurado por um *message broker*.

Capítulo 3

Conceitos de Suporte

Neste capítulo serão descritas as características e as funcionalidades de um *broker*. Posteriormente serão analisadas e comparadas diversas arquiteturas e tecnologias utilizadas na implementação de *brokers*.

3.1 *Broker*

Um sistema fundamental no envio e receção de mensagens em larga escala é o *broker*, este sistema intermédio é responsável por mediar a troca de mensagens, minimizando a necessidade das aplicações estarem cientes do recetor ou emissor. Para além disto, os *brokers* podem ter muitas outras funções e funcionalidades, mas por norma asseguram uma entrega segura e ordenada das mensagens ao recetor, podendo também ser responsáveis por assegurar taxas de envio, guardar registos de mensagens ou até mesmo responder a erros ou eventos [Stoja et al. 2013].

Devido à natureza dos *brokers*, estes operam normalmente entre dois tipos de modelo de mensagens, *point-to-point* em que existe uma relação de um-para-um entre o recetor e o emissor, este tipo de mensagens é utilizado quando a mensagem apenas pode ser executada uma única vez, como por exemplo em transações financeiras. O segundo modelo de mensagens é o *publish/subscribe* em que tópicos são criados pelos produtores de mensagens, podendo ser posteriormente subscritos pelos recetores que desejam receber mensagens deste tópico. O *broker* tem como função neste caso de gerir todos os tópicos replicando as mensagens para todos os recetores que desejarem recebê-las (figura 3.1).

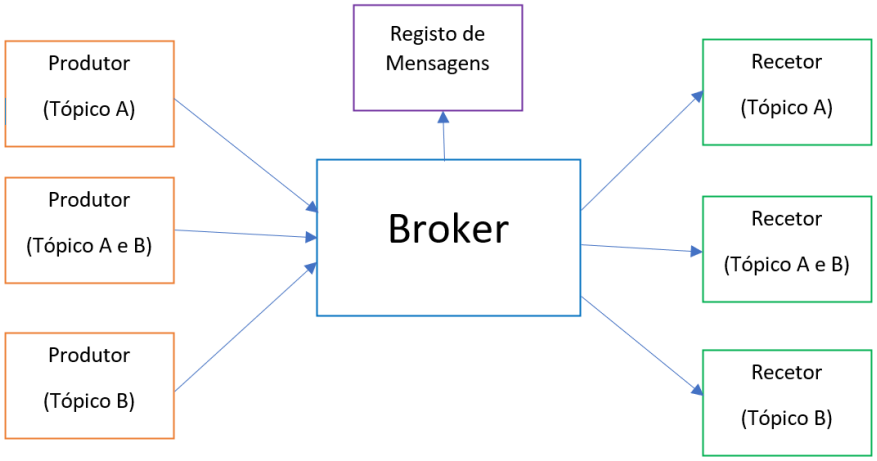

Figura 3.1: Ilustração da funcionalidade de um *broker*.

3.2 Soluções Existentes

Existem muitas tecnologias de *brokers*, cada uma melhor ou pior para determinadas circunstâncias ou problemas. Foram escolhidos diversos *brokers* de forma a ser realizado um estudo e comparar as suas diferentes características e defeitos, além disso foram analisados e comparados alguns pontos fundamentais para realizar um sistema como CPS e CM. Algumas características são o fato de permitirem transmitir grandes volumes de dados de forma rápida, mesmo quando um grande volume de entidades pretende comunicar simultaneamente, além de ser ainda essencial uma fácil integração do *broker* nestas entidades, estas características são essenciais no âmbito de implementação da Industria 4.0 [Rüßmann 2015;Liu e Jiang 2016]. Foram assim comparados os seguintes pontos :

- **Taxa de transferência**: A taxa de transmissão máxima que o *broker* consegue executar.
- **Escalabilidade**: Como o aumento dos *publishers/subscribers* afeta a taxa de transmissão e latência.
- **Latência**: Tempo que demora todo o processo de troca de uma mensagem.
- **Integração**: Suporte da integração de diferentes sistemas e linguagens.
- **Extras**: Este parâmetro tem em consideração desde: uma grande comunidade, continuidade de desenvolvimento, suporte de funcionalidade potencialmente úteis, entre outras.

3.2.1 Apache Kafka

O *Apache Kafka* é um software *open-source* com uma comunidade em crescimento, lançado em Janeiro de 2011. Possui várias bibliotecas que suportam quase todas as linguagens de programação para uma comunicação entre recetor/produtor e o *Apache Kafka*. É uma tecnologia com uma taxa de transmissão bastante rápida, esta rapidez deve-se maioritariamente ao *Apache Kafka* não ter a necessidade de receber um *acknowledge* por parte do recetor além de utilizar uma tecnologia de *pull*, sendo o recetor a pedir a próxima mensagem ou conjunto de mensagens. Em relação a escalabilidade, o *Apache Kafka* permite um grande número de *publishers/subscribers* sem grande perda na taxa de transmissão nem na latência, um exemplo desta capacidade é o facto da empresa *Netflix* usar o *Apache Kafka* suportando cerca de quinhentos mil milhões de eventos e totalizando cerca de 1.3*PB* de informação por dia [Posta 2016].

O *Apache Kafka* possui outras funcionalidades bastantes úteis como a persistência das mensagens, sendo estas guardadas num registo por tempo pré-determinado pelo administrador. O *Kafka* suporta também o ordenamento de mensagens, possuindo uma chave que permite entrega ao cliente as mensagens pela mesma ordem que é recebida no *broker*. Os protocolos de entrega de mensagens utilizados pelo *Kafka* são: No máximo uma entrega, Exatamente uma entrega, No mínimo uma entrega (*At most once*,*Exactly once*,*At least once delivery*)[Nikhil e Chandar 2020; Sharvari e Sowmya 2019].

Por fim o *Kafka* possui também toda uma plataforma chamada de *Confluent* que permite facilmente interpretar e configurar todos os parâmetros do *broker*. Esta plataforma é como uma camada que envolve todos os processos necessário para correr o *Apache Kafka* e que facilita a sua interação com o mundo exterior como representado na figura 3.2.

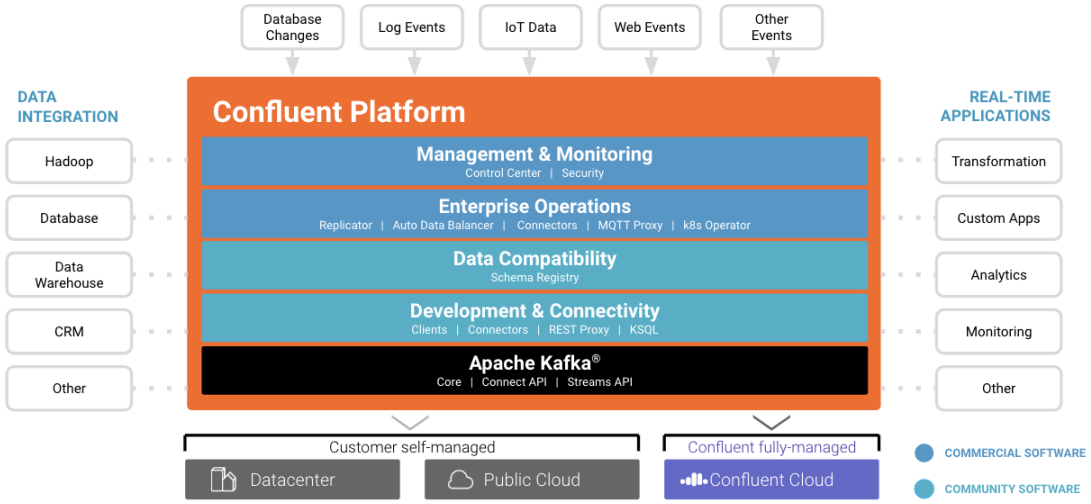

Figura 3.2: Ilustração do *Apache Kakfa*, *Confluent* e a sua interação com outras aplicações (imagem retirada de [Apache/Confluent 2021]).

3.2.2 RabbitMQ

O *RabbitMQ* é um software *open-source*, lançado em 2007 tendo alguma compatibilidade mais adaptada para sistemas mais antigos. Possui várias bibliotecas que suportam quase todas as linguagens de programação para uma comunicação entre recetor/produtor e o *RabbitMQ*. É uma tecnologia que consegue alcançar velocidades de taxa de transmissão bastante elevados, porém utilizando uma abordagem de *push* (comunicando previamente com o recetor a taxa de transmissão das mensagens que será efetuada), este nunca alcança a velocidade máxima do recetor, além disso a arquitetura utilizada consome mais recursos que outras tecnologias com a mesma abordagem, para realizar a mesma taxa de transmissão. Em relação a escalabilidade, o *RabbitMQ* perde o seu desempenho à medida que o número de *publishers/subscribers* aumenta, diminuindo assim a sua taxa de transmissão e aumentando a latência [Sharvari e Sowmya 2019].

O *RabbitMQ* não garante que as mensagens sejam guardadas em disco, devido a por exemplo um *hard reset*. Esta tecnologia não permite também o ordenamento de mensagens, não garantindo assim que as mensagens cheguem ao recetor pela mesma ordem que chegaram ao *broker*. Os protocolos de entrega de mensagens utilizados pelo *RabbitMQ* são: No máximo uma entrega, Exatamente uma entrega, No mínimo uma entrega (*At most once*,*Exactly once*,*At least once delivery*) [RabbitMQ 2020b; RabbitMQ 2020a].

O *RabbitMQ* é uma tecnologia adaptada principalmente para a comunicação em que a quantidade de *publishers/subscribers* não é muito elevada, no entanto têm uma vantagem muito interessante em que o tamanho máximo das mensagens transmitidas poderem ser até 2*Gb* permitindo enviar ficheiros de áudio ou vídeo, por exemplo.

É possível observar na figura 3.3 uma representação visual do funcionamento do *RabbitMQ*.

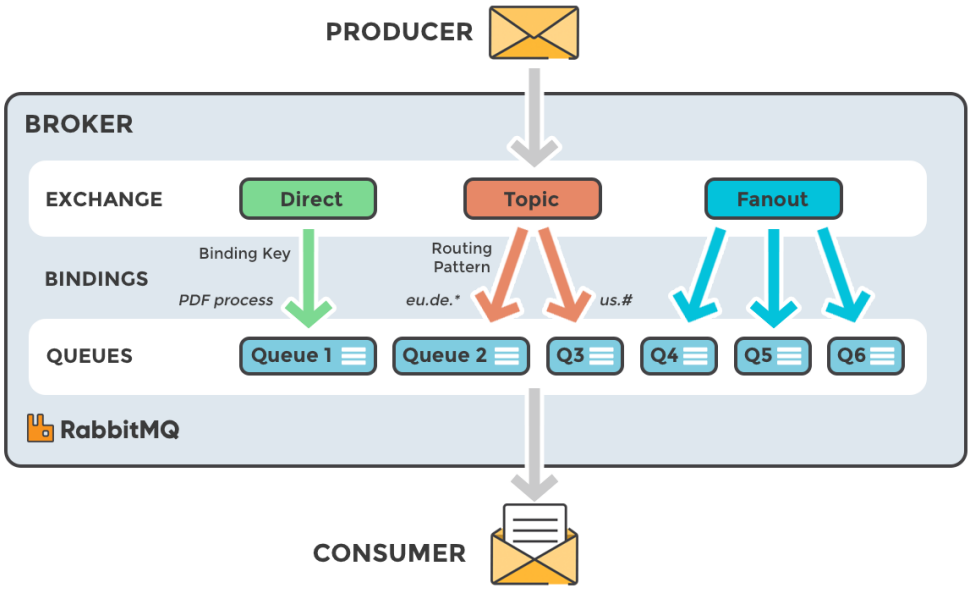

Figura 3.3: Ilustração do funcionamento do *RabbitMQ* (imagem retirada de [Jyoti 2018]).

3.2.3 Apache ActiveMQ

O *Apache ActiveMQ* é um software *open-source* lançado em Janeiro de 2004. Possui várias bibliotecas que suportam quase todas as linguagens de programação para uma comunicação entre recetor/produtor e o *Apache ActiveMQ*. É uma tecnologia focada essencialmente na comunicação com baixo nível, necessitando de um *acknowledge* do recetor sempre que envia uma mensagem, além disso usa uma abordagem tradicional de *push* (comunicando previamente com o recetor a taxa de transmissão das mensagens que será efetuada), desta forma o recetor nunca trabalha na sua velocidade máxima. Em relação a escalabilidade, o *Apache ActiveMQ* diminui bastante o seu desempenho à medida que o número de *publishers/subscribers* aumenta, diminuindo assim a sua taxa de transmissão e aumentando a latência [Apache/ActiveMQ 2019a].

O *Apache ActiveMQ* não garante que as mensagens sejam guardadas em disco, devido a por exemplo um *hard reset*. Esta tecnologia não permite também o ordenamento de mensagens, não garantindo assim que as mensagens cheguem ao recetor pela mesma ordem que chegaram ao *broker*. O *Apache ActiveMQ* garante a receção da mensagem devido ao *acknowledge* enviado pelo recetor [Apache/ActiveMQ 2019a; Apache/ActiveMQ 2019c].

Apache ActiveMQ é ideal a comunicação entre elementos físicos e envio de dados dos mesmos, podendo funcionar bastante bem numa empresa que pretender conectar diversos sensores e atuadores.

Observe-se assim na figura 3.4 a arquitetura do *ActiveMQ*, estando associados os diferentes protocolos para comunicar ou armazenar dados que a tecnologia utiliza.

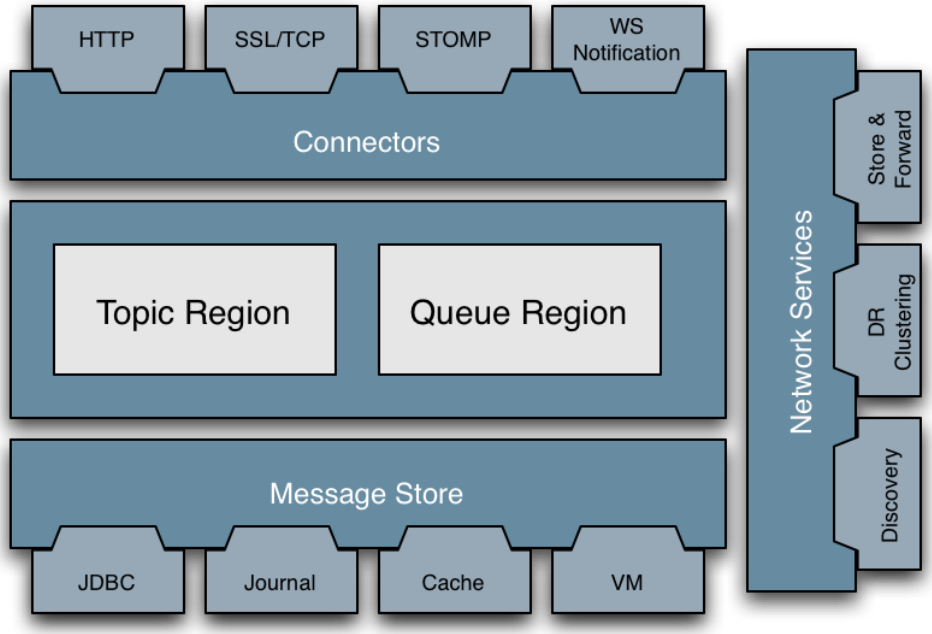

Figura 3.4: Ilustração da arquitetura do *ActiveMQ* (imagem retirada de [Apache/ActiveMQ 2019b]).

3.2.4 NATS Streaming

O *NATS* é um software *open-source* desenvolvido em linguagem *Go*, lançado em Janeiro de 2015. Possui várias bibliotecas que suportam quase todas as linguagens de programação para uma comunicação entre recetor/produtor e o *NATS*. É uma tecnologia adaptada a *streaming*, possuindo assim uma taxa máxima de transmissão bastante elevada, usando para isto uma abordagem *pull*, sendo o recetor a pedir a próxima mensagem ou conjunto de mensagens. Em relação a escalabilidade esta tecnologia começa a ter agravados problemas quando se fala em elevados números de *publishers/subscribers*, um dos maiores fatores para isto é a utilização de mecanismos de agenda, provenientes da linguagem de programação *Go* utilizada deste software. Outro problema bastante relevante é a latência bastante elevada que o *NATS* possui, isto deve-se essencialmente à necessidade de comunicara com o servidor *NATS* antes de qualquer ação [Sharvari e Sowmya 2019] como representado na figura 3.5.

O *NATS* não garante que as mensagens sejam guardadas em disco, devido a por exemplo um *hard reset*. Esta tecnologia não permite também o ordenamento de mensagens, não garantindo assim que as mensagens cheguem ao recetor pela mesma ordem que chegaram ao *broker*. Os protocolos de entrega de mensagens utilizados pelo *NATS* são: No máximo uma entrega, Exatamente uma entrega, No mínimo uma entrega (*At most once*,*Exactly once*,*At least once delivery*) [Tyler 2018].

Em Abril de 2019, *NATS* passou por uma melhoria substancial nos seus serviços de *streaming*, melhorando alguns parâmetros discutidos, contudo ainda não foram realizados artigos ou estudos sobre esta melhoria, um motivo disto pode ser a fraca comunidade que o *NATS* possui e isto também pode indicar que a melhoria não foi o suficiente para fazer o *NATS* se destacar face a outras tecnologias.

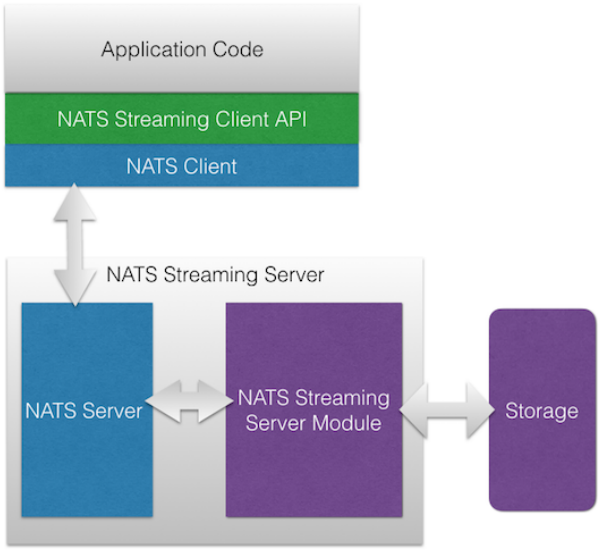

Figura 3.5: Ilustração do funcionamento do *NATS* (imagem retirada de [Tyler 2018]).

3.2.5 Apache Pulsar

O *Pulsar* é um software *open-source* desenvolvido pela *yahoo* e doado a *Apache* em 2016. Possui um número reduzido de bibliotecas que suportam apenas *Java*, *Go*, *Python* e *C++*. É uma tecnologia adaptada a *streaming*, possuindo assim uma taxa máxima de transmissão bastante elevada, usando para isto uma abordagem *pull*, sendo o recetor a pedir a próxima mensagem ou conjunto de mensagens. Em relação a escalabilidade esta tecnologia permite um grande número de *publishers/subscribers* sem grande perda na taxa de transmissão. No entanto *Pulsar* possui uma latência bastante elevada, com um crescimento quase exponencial, quando se aumenta linearmente o número de *publishers/subscribers* [Renart et al. 2017].

O *Pulsar* guarda todas as mensagens recebidas durante um tempo pré-determinado pelo administrador. Esta tecnologia suporta ordenamento de mensagens, possuindo uma chave que permite entregar ao cliente as mensagens pela mesma ordem que é recebida no *broker*. Os protocolos de entrega de mensagens utilizados pelo *Pulsar* são: No máximo uma entrega, Exatamente uma entrega, No mínimo uma entrega (*At most once*,*Exactly once*,*At least once delivery*) [Renart et al. 2017].

O *Pulsar* requer dois sistemas para funcionar, o *Apache BookKeeper* e o *Apache Zookeeper* (figura 3.6) isto aumenta a utilização de recursos no entanto confere algumas vantagens adicionais como *Multi Tenancy*, *geo-replication* entre outras.

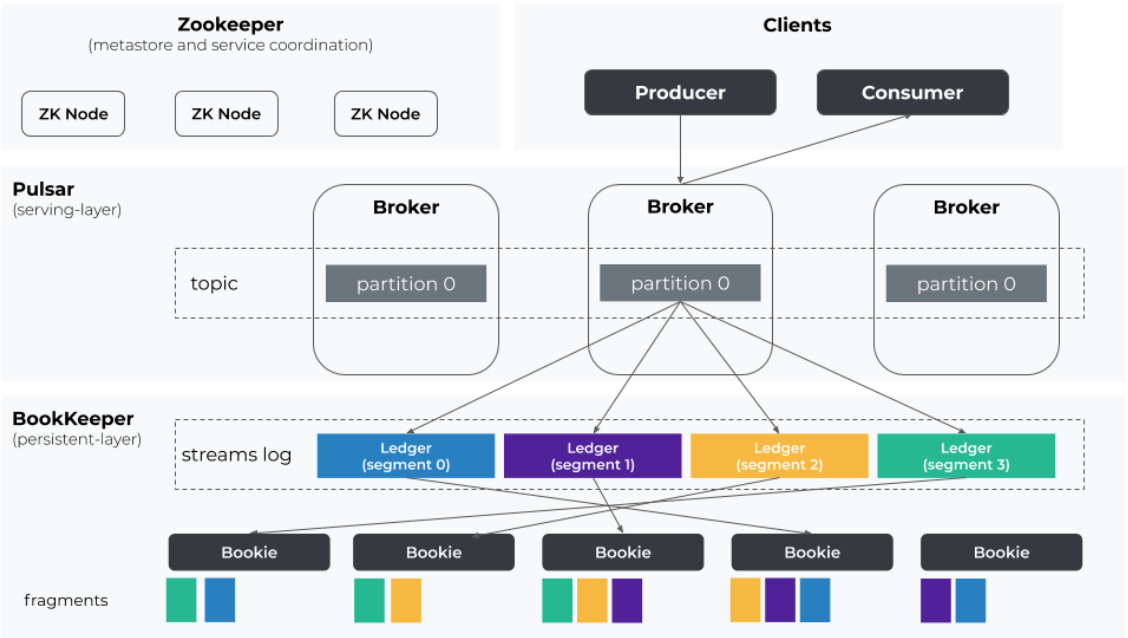

Figura 3.6: Ilustração do funcionamento do *Pulsar* aliado aos dois sistemas *Apache BookKeeper* e *Apache Zookeeper* (imagem retirada de [Hussonnois 2019]).

3.2.6 Outros

- ***Kestrel*** - descontinuado desde 2016.

- ***Mosquitto*** - possui funcionalidades praticamente iguais ao *ApacheMQ*.

- ***ZeroMQ*** - versão mais leve e de mais baixo nível do *ApacheMQ* reforçando os seus pontos fortes mas consequentemente reforçando também os seus pontos fracos.

3.2.7 Sumário

Existem muitos mais *software* que podem ser analisados, no entanto uma pesquisa considerável foi realizada em alguns dos mais conhecidos *brokers*.

Na tabela 3.1 está uma comparação, dos *brokers* estudados realizada pelo autor. A classificação foi dada atribuindo o valor 5 ao melhor *broker* em cada categoria e avaliando os restantes *brokers* em função do melhor em cada categoria. Esta avaliação é auxiliada por alguns artigos que complementam o estudo em causa [Sharvari e Sowmya 2019; Stoja et al. 2013; Renart et al. 2017]:

Tabela 3.1: Tabela de comparação geral dos *brokers* estudados.

Software	Taxa de Transmissão	Escalabilidade	Latência	Integração
Apache Kafka	5	5	4	4
RabbitMQ	3	2	5	5
Apache ActiveMQ	2	2	2	4
NATS Streaming	4	3	2	4
Apache Pulsar	4	5	2	2

Observa-se agora com maior facilidade a comparação entre as diferentes tecnologias, é possível constatar que o *Pulsar*, *NATS Streaming* e *Apache Kafka*, são tecnologias bastante boas em termos do volume de dados que conseguem transmitir, enquanto tecnologias como o *RabbitMQ* possuem latências muito baixas e uma grande quantidade de bibliotecas oficiais ou realizadas pela comunidade, para a integração em diversos sistemas.

É possível agora com este estudo, realizar uma decisão mais informada relativamente a que *brokers* serão mais indicados para diferentes tipos de projetos.

Capítulo 4

Arquitetura

Ao longo deste capítulo será apresentada a arquitetura do trabalho desenvolvido, será descrito como os diferentes processos e tecnologias, tanto na *cloud* como no *shopfloor*, devem operar de forma a que todo o sistema funcione. Serão também descritos os conceitos de *publish/subscribe* e como esta tecnologia se enquadra com a abordagem e arquitetura em estudo.

4.1 Arquitetura Desenvolvida

Na figura 4.1 está presente a arquitetura desenvolvida no âmbito da presente dissertação. Esta arquitetura demonstra como diferentes dispositivos, bases de dados ou até mesmo empresas colaborativas podem publicar ou receber dados de um *message broker*. Esta abordagem permite que não exista uma comunicação direta entre os diferentes *software* e *hardware* tendo os dados de passar pelo *message broker*, facilitando a integração de novas ferramentas que necessitem de obter dados tanto do *shopfloor* como da *cloud* dado que toda a informação passa pelo *message broker*. Importa salientar que todas as ferramentas conectadas ao *message broker* não necessitam de ter o mesmo *software* ou estarem conectados da mesma forma, isto permite que diferentes fabricantes ou empresas de ramos de negócio diferentes, possam colaborar e partilhar informações. Contudo, existem duas restrições impostas à utilização do *message broker* que devem ser consideradas. A primeira restrição é a comunicação com o *message broker*, utilizando bibliotecas próprias, *APIs* ou qualquer outra forma de comunicação que o *message broker* escolhido permita. A segunda restrição trata-se da utilização de um modelo de dados, os dispositivos que recebem os dados têm de estar cientes do formato e tipo de dados que receberão. Esta restrição pode tornar-se prejudicial e difícil de gerir a longo prazo à medida que mais sistemas vão sendo incorporados com diferentes modelos de dados, existindo algumas

medidas como por exemplo a adoção de *standard interfaces* abordadas um pouco mais a frente neste capítulo.

O *message broker* torna-se assim um pilar essencial na troca de informação entre os dispositivos no *shopfloor* e os *software* de desenvolvimento situados na *cloud*, tornando o *message broker* o responsável pela gestão de todo o fluxo de dados, conferindo aos dispositivos do *shopfloor* a liberdade de enviar a informação quando for mais conveniente, além de tornar esta informação disponível a ferramentas de alto nível, permitindo mais facilmente o armazenamento, processamento ou a visualização dos dados.

Os resultados processados pelas ferramentas de alto nível permitem agora que parâmetros sejam ajustados nos dispositivos de *shopfloor*, enviando dados ou comandos para os tópicos aos quais estes dispositivos se encontram conectados. Com este novo fluxo de informação é agora possível ter um sistema otimizado e uma monitorização em tempo real dos diferentes processos do *shopfloor*.

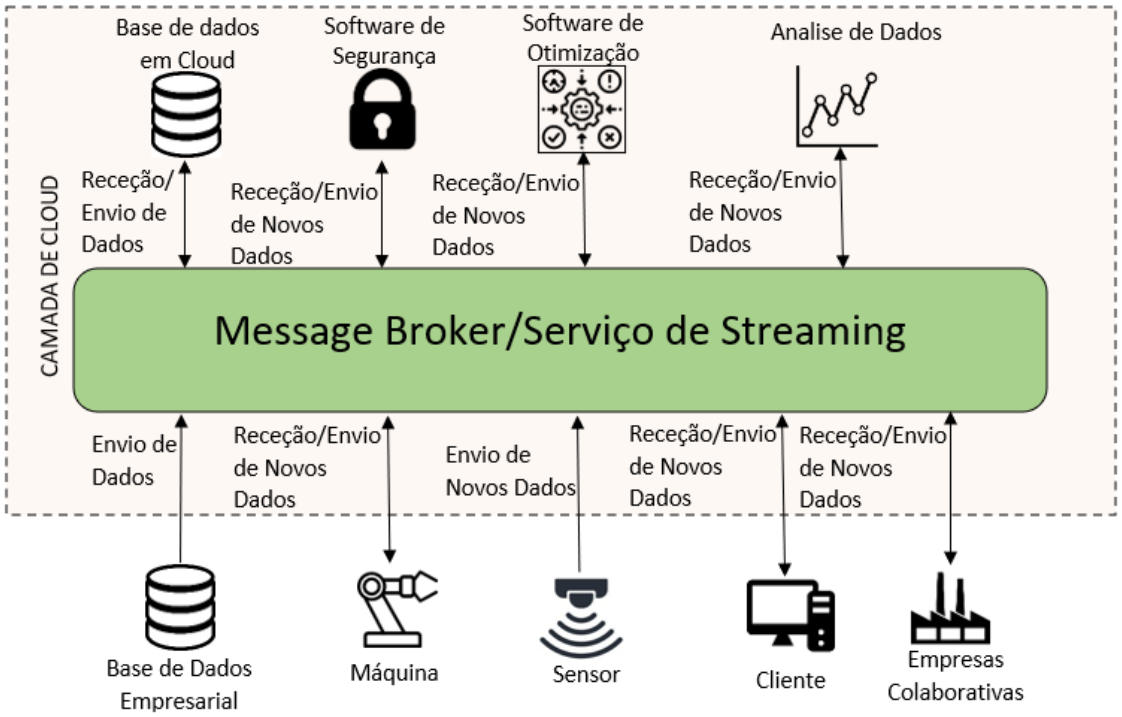

Figura 4.1: Arquitetura desenvolvida.

Na figura 4.2 encontra-se um exemplo da arquitetura proposta aplicada ao mundo real. Na aplicação exemplo da arquitetura é possível observar as máquinas do *shopfloor* a comunicarem com o *Message Broker*/ Serviço de *Streaming* de diferentes formas, diretamente (máquina da esquerda), com o auxílio de um programa intermédio (máquina central) ou com o auxílio de um programa como um serviço sendo esses serviços chamados posteriormente pela máquina à medida que são necessários (máquina da direita). Por sua vez na camada de *cloud* as bases de dados podem comunicar com o *Message Broker*/ Serviço de *Streaming* diretamente (base de dados da direita) ou com o auxílio de um programa intermédio (base de dados da esquerda). Posteriormente um HMI pode conectar-se

a uma das bases de dados facilitando a leitura de dados por parte de um operador.

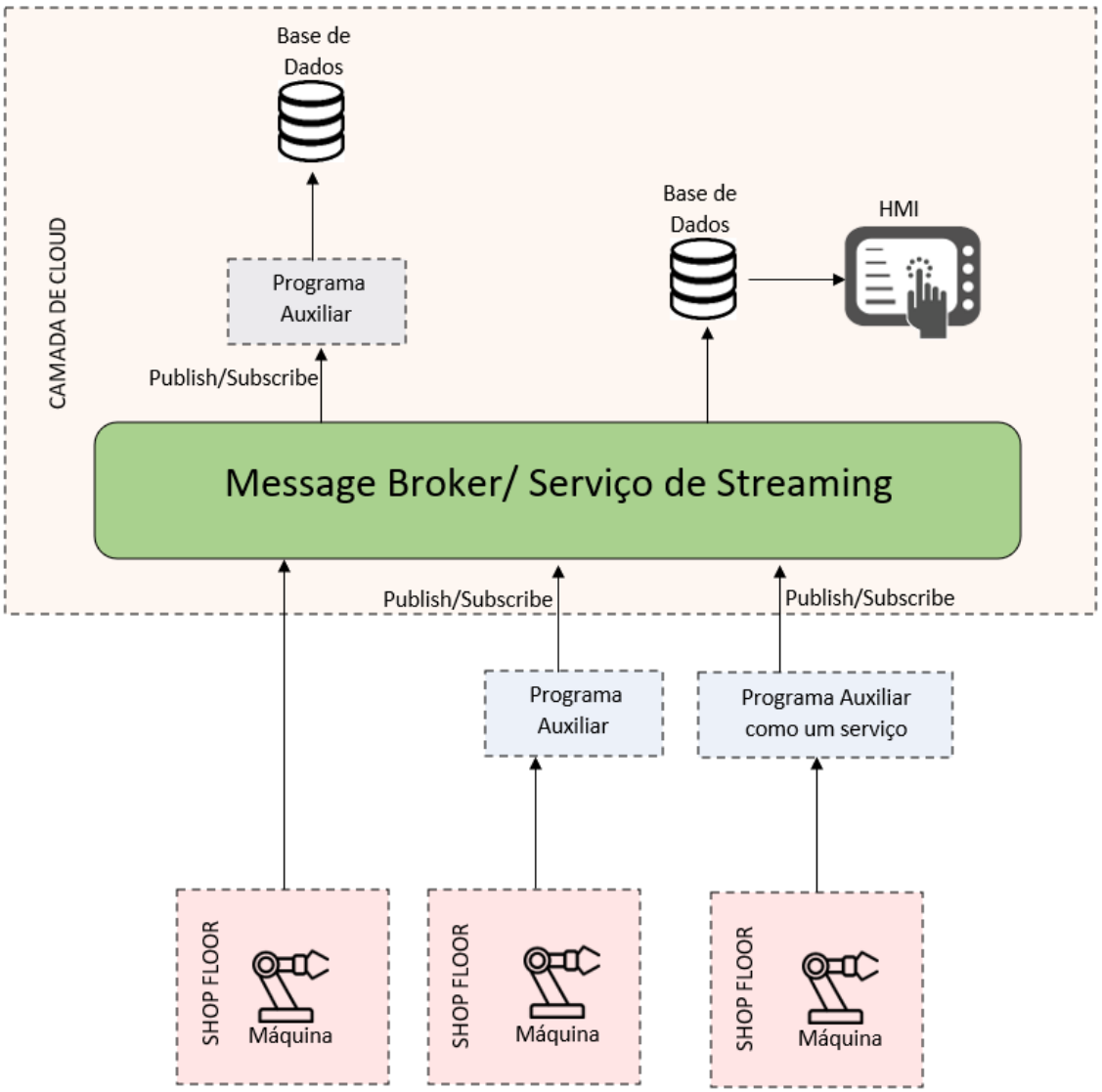

Figura 4.2: Exemplo da a aplicação da arquitetura

No entanto perante esta arquitetura todos os componentes utilizados que recebem a informação necessitam de saber qual o formato e tipo de informação que recebem, ou seja torna-se essencial existir um modelo de dados conhecido com a informação que vai ser recebida pelos *consumers* e enviada pelos *publishers*.

4.1.1 Modelo de dados

Na figura 4.3 está representado o modelo de dados utilizado para o sistema exemplo, demonstrando as sete tabelas e respetivas colunas, necessárias na base de dados. Este modelo de dados permite a visualização orientada ao consumo de cada célula robótica por produto ou receita (*task*) executada, permitindo o envio de diferentes tipos de informação necessária para a posterior análise de dados ou visualização dos mesmos.

Seja analisada brevemente cada tabela da figura 4.3 de forma a um melhor entendimento do fluxo de dados:

1. **Product_Execution_Data:** Representa os dados de execução de um produto, como o tempo e o ponto de situação. Estando conectada à tabela Recipe_Execution_Data, através do UniqueId de cada produto e a tabela dos Products através do ProductID, sendo este o ID único de cada produto.

2. **Recipe_Execution_Data:** Esta tabela guarda os dados de execução de cada receita, como o tempo, a estação, o produto em que a receita foi executada e um *array* com o ID dos consumos e parâmetros utilizados na execução. Desta forma esta tabela fica conectada a Product_Execution_Data, as Recipes, as Stations e ainda aos consumos que se encontram na tabela KPI_Execution_Data.

3. **KPI_Execution_Data:** São armazenados os consumos e parâmetros de execução, esta tabela é dimensionada de maneira a que caso os processos de produção e complexidade aumentem (com mais dados de consumos e mais parâmetros), o modelo de dados continua a ser o mesmo. Desta forma cada UniqueId corresponde a uma determinada receita que já foi executada, podendo assim ser observado todos os consumos da execução da receita.

4. **Products:** Nesta tabela estão todos os produtos existentes na fábrica, associados ao seu tipo e data de registo.

5. **Recipes:** Estão nesta tabela todas as receitas criadas, com o nome, o tipo de receita e a data que foi criada.

6. **Stations:** São armazenadas nesta tabela, as estações existentes, bem como a data de registo das mesmas.

7. **KPIs:** Esta tabela é utilizada para ter um nome associado ao ID de cada KPI_Execution_Data, bem como uma descrição do mesmo.

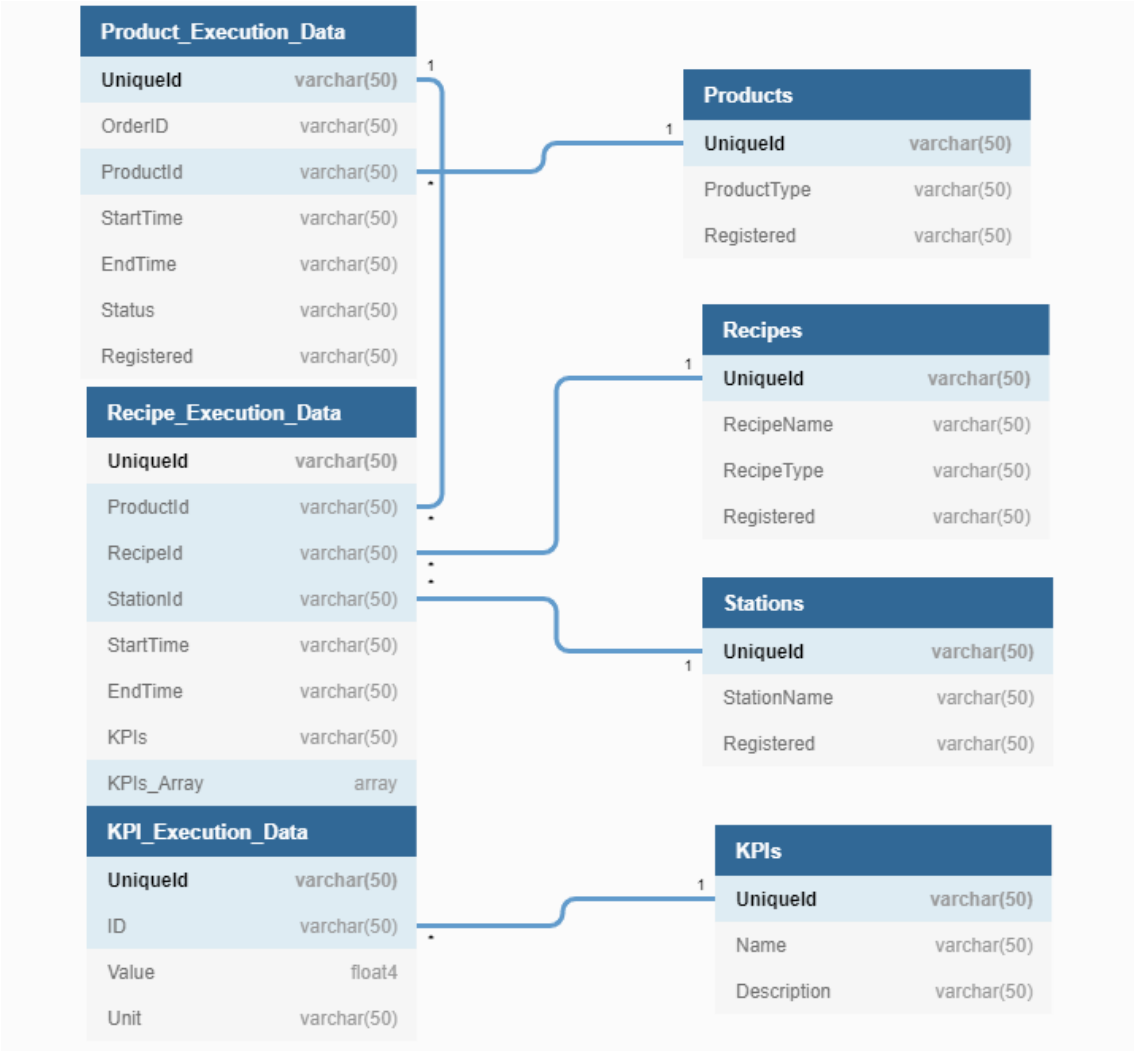

Figura 4.3: Modelo de dados.

Perante a descrição da arquitetura e do modelo de dados, começam a existir similaridades e sobreposição de conceitos com o modelo de mensagens *publish/subescribe*, sendo este modelo um dos melhores para desenvolver este tipo de arquiteturas.

4.1.2 *Publish/Subscribe*

Uma abordagem de *Publish/Subscribe* no *message broker* presente na figura 4.1, permite que vários sistemas publiquem os dados para múltiplos consumidores de forma assíncrona, em que por sua vez cada consumidor trabalha os dados da forma e com a velocidade mais conveniente.

Cada sistema (maquinaria ou *software*) que gera e partilha (publica) os dados, é conhecido como *publisher* e cada sistema que adquire os dados partilhados por um *publisher* é conhecido como *consumer*.

Um *publisher* é responsável por traduzir mudanças no sistema ou ações em mensagens, usando formatos conhecidos (JSON, Avro). Os dados gerados pelo *publisher* são

enviados para um tópico específico do *message broker*, por sua vez, todos os *consumers* interessados em receber os dados publicados, devem de estar subscritos ao tópico correspondente. Desta forma, o *message broker* sempre que deteta novos dados nos tópicos, envia esses mesmo dados a todos os *consumers* subscritos ao tópico. Estas interações podem ser demonstradas na figura 4.4.

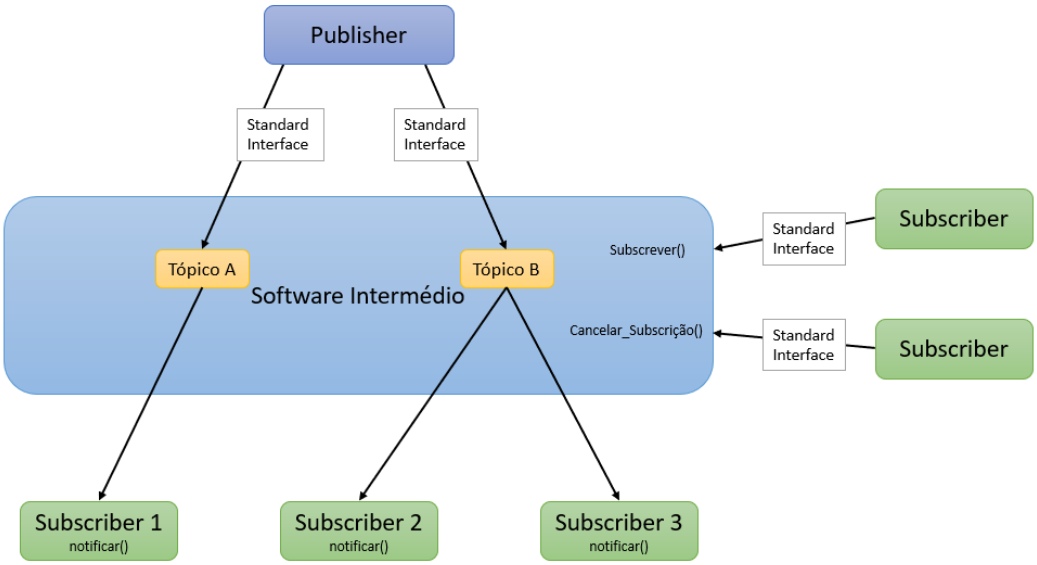

Figura 4.4: Modelo de *Publish/Subscribe*, retirado de [Rocha et al. 2021].

A utilização de um modelo *Publish/Subscribe* permite que toda a comunicação seja tratada de forma independente, evitando que os sistemas sejam bloqueados por estarem à espera de uma resposta e consequentemente fazendo com que todos os sistemas tenham a velocidade do elemento mais lento, o que melhora a escalabilidade e a resposta dos sistemas. Este modelo oferece ainda uma integração mais acessível entre diferentes sistemas, podendo estes usar diferentes plataformas, linguagens de programação, protocolos de comunicação, empresas ou aplicações de *cloud*.

Quando toda a comunicação é realizada através de *Publish/Subscribe* começa a surgir um problema no contexto moderno da industria, sistemas antigos ou com pouca capacidade de se adaptar podem impossibilitar a sua integração no modelo *Publish/Subscribe*. Para superar este desafio o conceito de *Standard Interfaces* foi criado.

4.1.3 *Standard Interfaces*

Um ponto importante a ser considerado quando existe um grande volume de dados a ser gerado por diversos sistemas em diferentes níveis operacionais é a representação desses mesmos dados, tendo como objetivo final alcançar a colaboração de diferentes *hardware* e *software*, como estações de trabalho, robôs, sistemas de gestão empresarial, bases de dados ou aplicações de *Manufacturing Execution System* (MES).

Seguindo esta linha de raciocínio, as *Standard Interfaces* devem ser um ponto central que facilite a integração e a interoperabilidade, permitindo a interação entre diferentes dispositivos de *hardware* e aplicações de *software*. As *Standard Interfaces* devem de disponibilizar um conjunto de funcionalidades que expõem e descrevem os serviços de uma forma sistemática e transparente. Algumas das funcionalidades mais comuns das *Standard Interfaces* são:

- Uma lista de serviços implementados na interface.
- Uma descrição de cada serviço (nome, parâmetros de entrada [input] e parâmetros de saída [output]).
- Uma descrição do modelo de dados de cada serviço.

As *Standard Interfaces* abstraem as funcionalidades fundamentais, tornando claro como os diferentes componentes interagem e operam. Porém, para um desenvolvimento das *Standard Interfaces* é necessário que exista uma abordagem orientada a serviços dos diversos sistemas, de forma a expor as funcionalidades e aplicações como um serviço.

Por fim é importante ter em conta a integração de sistemas mais antigos que não possuem a capacidade de se adaptar, possuem sistemas com modelos de dados ou requerimentos muito específicos. Perante as restrições mencionadas, pode ser necessário um sistema intermédio que traduza a informação para o *hardware* ou *software* mais antigo e vice-versa, estes sistemas possuem o nome de *gateways*, permitindo assim que mesmo *hardware* ou *sotware* com restrições, sejam incorporados na automação e integração de todos os sistemas [Babovic e Milutinovic 2013].

Capítulo 5

Implementação

Neste capítulo irão ser descritas as implementações dos diferentes casos de estudo realizados. Numa primeira instância irá ser por descrito o *broker* utilizado bem como o motivo da sua escolha e a abordagem em relação à sua implementação, seguida de uma apresentação individual dos casos de estudo com os respetivos diagramas, modelos de dados, observações e restrições de cada estudo.

5.1 *Broker*

Ao ser analisada a arquitetura proposta no capítulo 4, é possível observar que a peça central é o *message broker*, torna-se assim essencial escolher um *message broker* que satisfaça todas as necessidades presentes e futuras do sistema. Para que a escolha do *message broker* seja a mais correta possível, foi realizado um estudo das características mais importantes que um *broker* deve possuir, no contexto da Indústria 4.0 e arquiteturas CPPS. Neste estudo foi ainda realizada uma análise de alguns dos *brokers* mais conhecidos e foi feita uma comparação entre eles, que se encontra na secção 3.2.

Analisado o estudo, chega-se a duas opções bastante similares e igualmente conhecidas, nomeadamente o *Pulsar* e o *Apache Kafka*. Ambos têm uma taxa de transmissão e escalabilidade semelhantes, contudo a latência do *Apache kafka* é menor principalmente se for permitido ao *Apache Kafka* escrever em *batch*, além deste ponto, o *Pulsar* necessita de mais um sistema para correr e possui menos bibliotecas dedicadas à integração. Diversas empresas conhecidas como *Netflix*, *Linkedin*, *Twitter* utilizam o *Apache Kafka* [Apache/-Confluent 2017] elevando o reconhecimento a nível empresarial do *software*. Tendo em conta todos os pontos mencionados anteriormente foi selecionado o *Apache Kafka* como o *message broker*.

O *Apache Kafka* permite que os consumidores recebam a informação dos tópicos aos

quais eles estão subscritos como um *publish/subscribe* tradicional, no entanto o *Kafka* permite também que o consumidor funcione como um serviço, em que sempre que existe uma nova mensagem no tópico o serviço é invocado (figura 5.1). Esta característica torna o *Apache Kafka* bastante apelativo, podendo eliminar *software* intermédio e cria abordagens direcionadas a serviços, o que facilita na integração e utilização a nível empresarial.

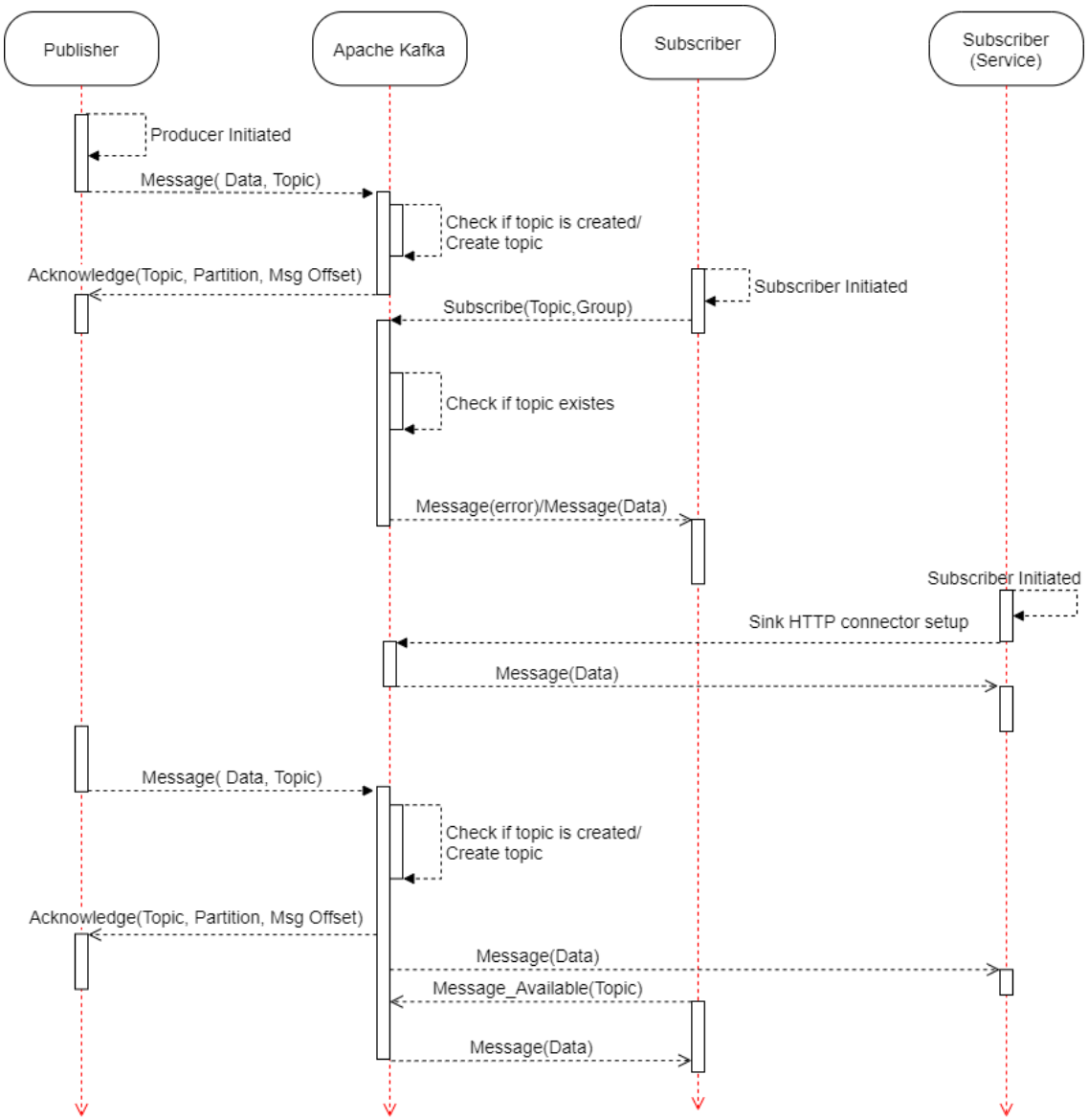

Figura 5.1: Diagrama de sequência do *Kafka*.

Outra funcionalidade bastante interessante que o *Apache Kafka* oferece, é o *Confluent*. Tal como brevemente descrito na secção 3.2.1, esta interface gráfica permite interpretar mais facilmente o fluxo de dados que circula pelo *broker* possuindo gráficos e alertas, além de ser possível configurar todos os parâmetros do mesmo, criando tópicos, conectores, *streams*, alterando parâmetros de conexão de *publishers* e *consumers* (figura 5.2). Esta visualização vai ser utilizada para mostrar as conexões e resultados, no *broker* ao longo

das demonstrações realizadas no âmbito da dissertação.

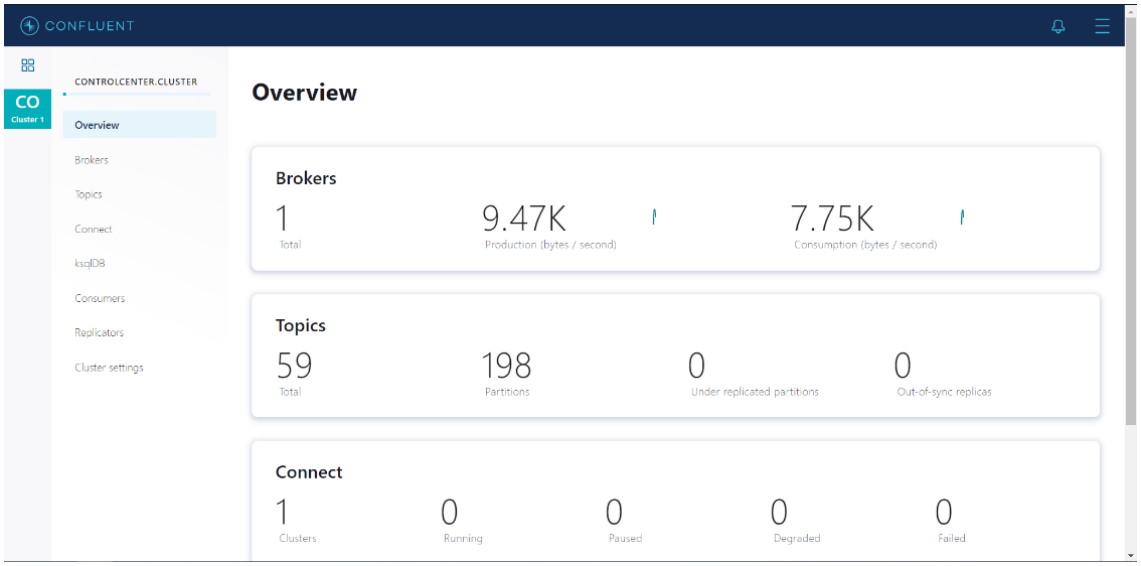

Figura 5.2: Visualização geral da plataforma *Confluent*.

Tendo o *broker* sido escolhido, é necessário criar um ambiente de testes, este ambiente pode ser por exemplo uma máquina virtual, a máquina local, um servidor ou uma rede de diversos computadores conectados sendo o *publisher*, o *consumer* e o *message broker* instanciados em diferentes máquinas. Devido às características das configurações e à perspetiva de poder facilmente instanciar todo o trabalho desenvolvido do *broker* em qualquer outra máquina ou *software*, foi decidido utilizar o *software Docker*, para correr o *message broker* bem como qualquer programa adicional necessário que devesse ser implementado num servidor ou *cloud*.

5.2 Docker

O *Docker* é uma "plataforma como um serviço" que utiliza virtualização do sistema operativo em ambientes únicos e isolados chamados de *containers*. Estes *containers* geralmente não possuem comunicação ou interação uns com os outros exceto em canais bem definidos pelo utilizador. O *Docker* possui repositórios próprios com várias tecnologias prontas a serem utilizadas, bem como várias versões das mesmas, permitindo integrar facilmente estas tecnologias em projetos existentes, novos projetos ou até mesmo utilizar versões especificas de *software* caso necessário.

Para instanciar os diferentes projetos, os ficheiros *.yml* foram criados. Estes ficheiros funcionam como guias, sobre que programas devem ser corridos, quais as versões dos programas, quais *containers* devem estar conectados, onde deve ser feito o armazenamento dos dados, entre muitas outras configurações e parâmetros que alteram ou adaptam o funcionamento dos *containers* e consequentemente dos programas que foram instanciados.

Este ambiente permite que, com o *Docker* instalado e um ficheiro de texto *.yml* qualquer computador com o sistema operativo de Linux, Windows ou Mac, consiga correr o programa com exatamente as mesmas características do computador original em que o ficheiro foi criado (figura 5.3). Um exemplo prático e que foi utilizado ao longo da dissertação, foi a utilização deste *software* para configurar e adaptar todas as variáveis no computador pessoal antes de ser instanciado no servidor de uma empresa, sendo possível fazer todos os testes e resolver problemas antes ser implementado a nível empresarial.

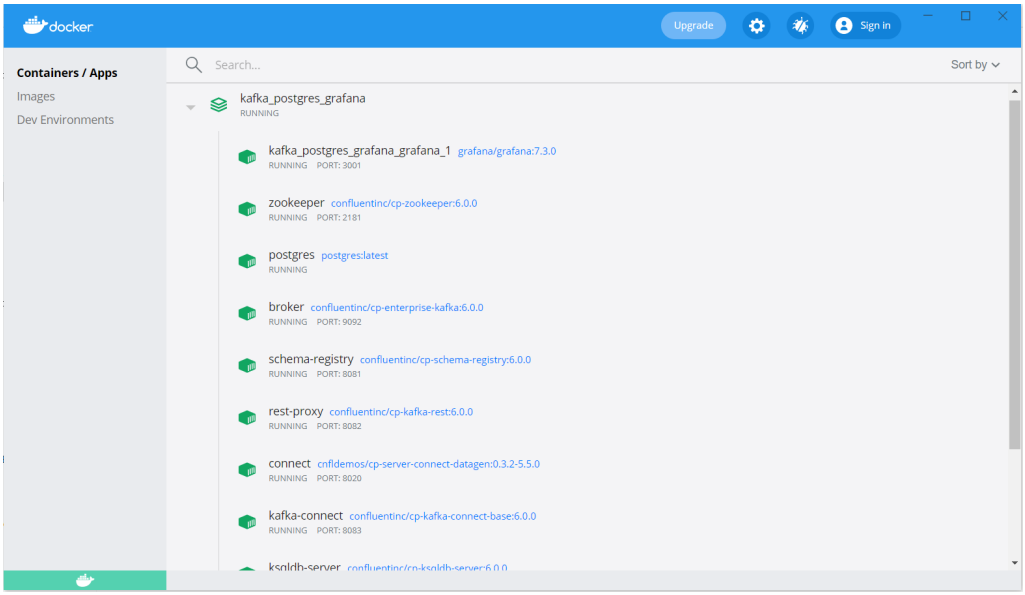

Figura 5.3: O software *Docker* e os diferentes *containers* com vários programas a correr em paralelo.

5.3 Implementação: Demonstração Laboratorial

Na primeira demonstração, foi realizado um estudo de como as interações entre o *Apache Kafka* com *publishers* de duas linguagens de programação diferentes (*Java* - Figura 5.4 e *Python* - Figura 5.5) e dois *consumers* programados nas mesmas linguagens. Foi utilizada duas linguagem de programação distintas de forma a demonstrar que diferentes linguagens de programação podem facilmente trocar mensagens entre elas, simulando os diferentes programas de diferentes maquinarias num ambiente industrial. Os *publishers* representam sensores a gerar dados cada um para o seu tópico específico, nomeadamente o tópico "SensorA", "SensorB" e "SensorC". Por sua vez, os *consumers* podem alternar os tópicos que querem ou não receber os dados, escrevendo esses dados numa caixa de texto sempre os recebem.

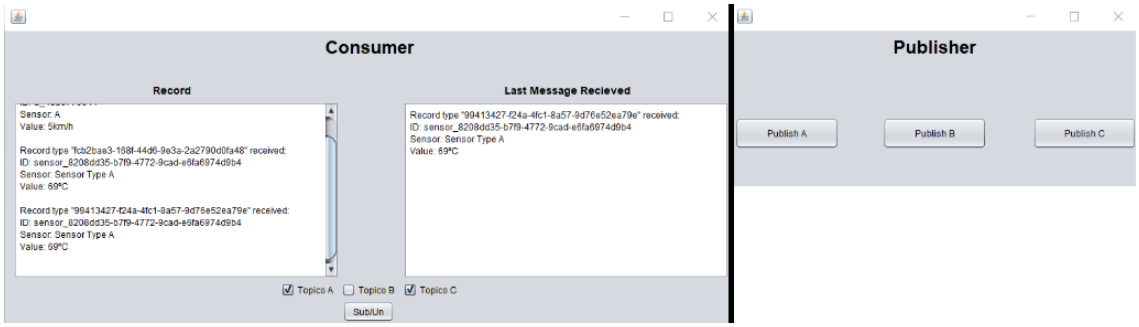

Figura 5.4: Demonstração dos programas em *Java*.

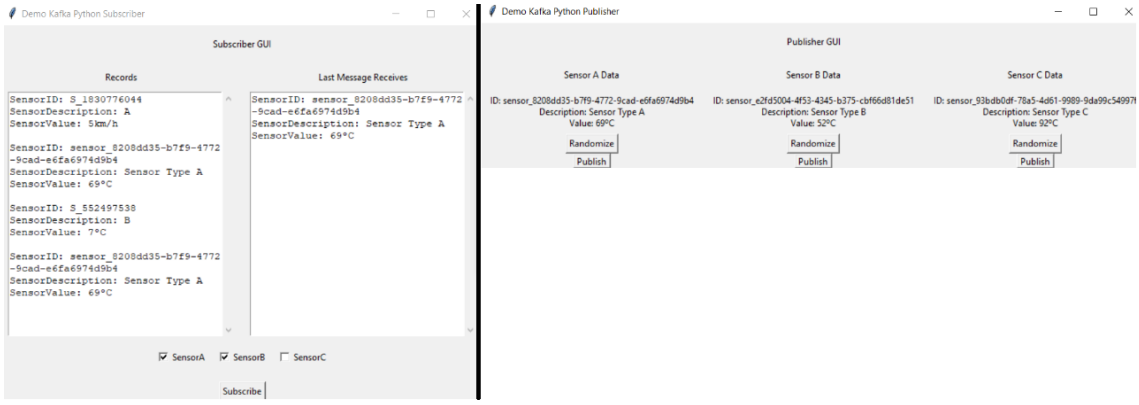

Figura 5.5: Demonstração dos programas em *Python*.

É possível observar nas figuras 5.4 e 5.5, que ambos os *consumers* estão subscritos ao tópico "SensorA", sendo possível ver a mesma informação nos dois *consumers*. Um dos *consumers* está subscrito ao tópico "SensorB" e o outro *consumer* ao tópico "SensorC", desta forma, a informação específica de cada tópico só é apresentada aos *consumers* que estão subscritos.

Por fim, foi criada uma *Application Programming Interface* (API) conectada ao *Kafka* (mais concretamente ao tópico "SensorA"), como um serviço utilizando um *sink connector*

HTTP, permitindo assim visualizar, utilizando o *browser*, os dados que estariam a ser escritos no tópico "SensorA" (figura 5.6). O conector HTTP é um conector integrado do *Kafka* que utilizando "POST"(método utilizado para adicionar um novo recurso a uma coleção) ou "PUT"(método utilizado para modificar um recurso já existente que seja parte de uma coleção) envia a informação para um URL pretendido, permitindo desta forma que qualquer aplicação disponível com um URL em rede, capaz de aceitar e processar "POST" ou "PUT", pode utilizar este conector.

Figura 5.6: Utilização de um *sink connector* HTTP para a visualização dos dados num *browser*.

É possível assim, observar através da plataforma *confluente* os tópicos criados (figura 5.7), bem como os *consumers* e conectores (figura 5.8).

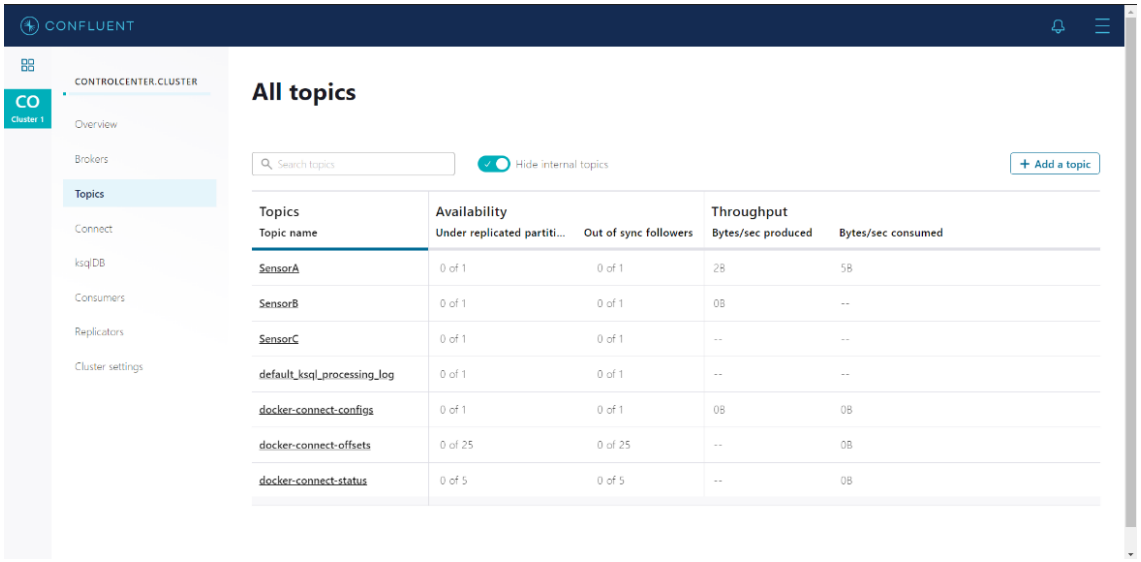

Figura 5.7: Visualização, através da plataforma *Confluent*, de todos os tópicos criados no *Kafka*.

É possível observar na figura 5.7 os tópicos descritos anteriormente (“SensorA”,“SensorB” e “SensorC”), além de alguns outros tópicos, utilizados tanto pela plataforma *Confluent* como pelo ambiente virtual *Docker*.

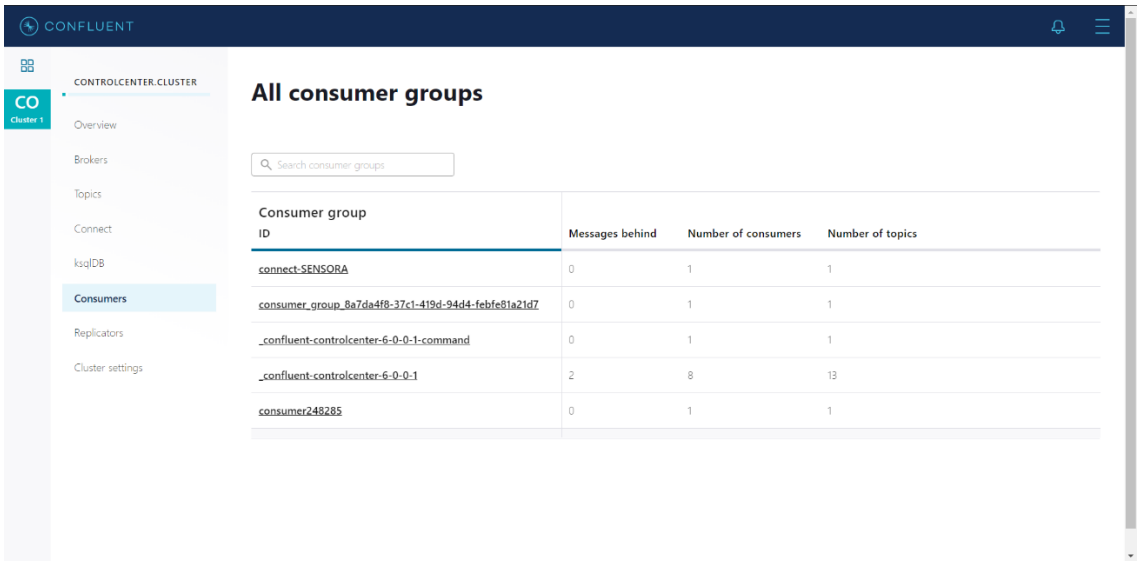

Figura 5.8: Visualização, através da plataforma *Confluent*, de todos os *Consumers* e conectores, conectados ao *Kafka*.

Analisando a figura 5.8 é possível observar o conector para a API denominado de “connect-SENSORA” e os dois *consumers* programados em *Java* e *Python* que recebem os dados dos tópicos que estão subscritos, além destes *consumers* existem outros dois utilizados pela plataforma *Confluent* de forma a extrair informação sobre o estado do *Kafka* para ser disponibilizado e visualizado na plataforma.

5.4 Implementação: Demonstração com Células Robóticas

A segunda demonstração realizada tem como objetivo validar a arquitetura e os *software* implementados num cenário real. Foram utilizadas duas células robóticas de fabricantes diferentes, que enviam dados de consumo energético e produção ao longo dos seus ciclos de trabalho. Todo o *software* desenvolvido foi instanciado numa *cloud* interna da Introsys na *Docker*. Desta forma, sempre que um produto é concluído, a sua informação de produção é enviada diretamente para o *Apache Kafka* que por sua vez comunica e guarda a informação na base de dados *Postgres*. Por fim foi realizado um *Human Machine Interface* (HMI) utilizando o *software Grafana* permitindo visualizar mais facilmente toda a informação armazenada na base de dados.

5.4.1 Células Robóticas

A presente demonstração tem por base as células robóticas utilizadas pela empresa Introsys, sediada na Quinta do Anjo, Portugal (figura 5.9). Estas células representam a

realidade atual das linhas de produção industrial da *Volkswagen* e da *Ford*, estando em conformidade em termos de equipamentos, segurança e padrões de controlo, usados pela indústria automóvel. Cada uma das células robóticas executa soldadura por pontos, possuindo um robô com 6 graus de movimento com um braço robótico equipado com uma garra personalizada, possuem ainda um suporte onde um operador coloca e retira a peça a soldar e por fim uma estação de soldadura. Tanto o suporte como a garra, possuem diversos sensores que detetam a presença do produto bem como garras pneumáticas que mantêm a peça segura.

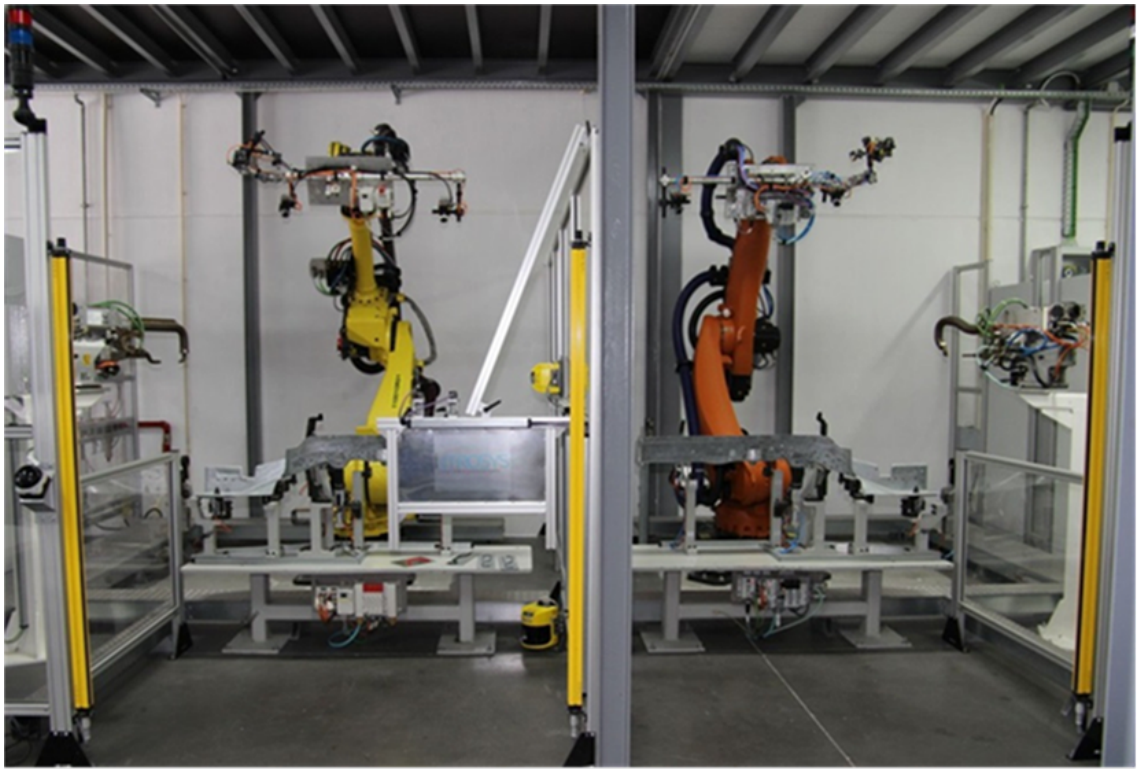

Figura 5.9: Células industriais robóticas da empresa Instrosys. À esquerda a célula da *Volkswagen* e à direita da *Ford*.

Observe-se em melhor detalhe a composição, fabricante e protocolo de comunicação da célula da *Volkswagen* (Tabela 5.1) e da célula *Ford* (Tabela 5.2).

Tabela 5.1: Componentes da célula robótica da *Volkswagen*.

Componente	Fabricante	Ferramenta	Interface de Comunicação
R30-iB	FANUC	Robot controller	Siemens/Profinet IO
R-2000iB 210F	FANUC	Robot	Digital wiring
Gripper	Introsys	Robot gripper	Profinet IO
PLC 319F 3PN DP	Siemens	Controller	Profinet IO
Simatic IPC677C	Siemens	HMI	Profinet IO
Loading Station (Tool)	Introsys	Sub-assembly	Profinet IO
Welding Gun	ARO	Joining Unit	Profinet IO
Scanner PLS3000	SICK	Operation safety	Digital wiring
Light Barrier C4000	SICK	Operation safety	Digital wiring
NZM3-XMC-MB	EATON	Energy measurement	Digital wiring

Tabela 5.2: Componentes da célula robótica da *Ford*.

Componente	Fabricante	Ferramenta	Interface de Comunicação
KR C4	KUKA	Robot controller	Proprietary/Ethernet IP
210 R2700 extra	KUKA	Robot	Digital wiring
Gripper	Introsys	Robot gripper	Ethernet IP
PLC	Allen Bradley	Controller	Ethernet IP
AB Safety I/O	Allen Bradley	I/O	Ethernet IP
Panel View Plus 1250	Allen Bradley	HMI	Ethernet IP
Loading Station (Tool)	Introsys	Sub-assembly	Ethernet IP
Welding Gun	ARO	Joining Unit	Ethernet IP
Scanner PLS3000	SICK	Operation safety	Digital wiring
Light Barrier C4000	SICK	Operation safety	Digital wiring
WAGO 750-493	WAGO	Energy measurement	Ethernet IP

Estas células robóticas foram desenhadas para manipular e soldar automaticamente uma longarina (peça específica dos veículos automóveis) e ambas executam praticamente o mesmo processo. A diferença entre as células robóticas consiste no facto que as soldaduras têm de estar em conformidade com os padrões de produção de cada fabricante de automóveis, divergindo assim, nos equipamentos utilizados, sistemas de controlo, sistemas de segurança, métodos de programação e ferramentas. Este caso de estudo oferece assim as condições ideais para demonstrar a interoperabilidade do sistema, existindo diversos programas de soldadura nas duas células que conseguem executar várias soldaduras em diferentes produtos.

Os processos de fabrico podem ser descritos da seguinte forma:

1. O operador coloca a peça no suporte da célula.
2. O robô move a peça do suporte até à estação de soldadura.
3. O robô move a peça entre os pontos pré-programados, de forma a soldar todos os pontos.
4. A peça é colocada novamente no suporte da célula.

5. O operador retira a peça do suporte da célula.

É importante salientar que estas células robóticas são abstraídas por uma camada de integração, implementada pela Introsys e programada em *Java*. Por sua vez, este programa comunica com as células robóticas e dispositivos adicionais através do protocolo de comunicação baseado em *Open Platform Communications* (OPC).

5.4.2 Implementação da Arquitetura

Na implementação específica da arquitetura analisada no capítulo 4, duas células robóticas estão conectadas através de dois métodos distintos ao *Apache Kafka*. Um desses métodos de conexão utiliza um programa em *Java*, responsável por recolher dados do *data bus* (figura 5.10 - máquina da esquerda). A outra célula robótica é conectada através de uma API e de um conector (figura 5.10 - máquina da direita). Implementadas as ligações, é possível as duas células robóticas enviarem dados em simultâneo para o *Apache Kafka*, que se encarrega de distribuir os dados por todos os *consumers* subscritos. Foi utilizado um conector JDBC (*Java Database Connectivity)*, suportado pelo *Kafka*, de forma a realizar uma *stream* (um fluxo continuo e em tempo real) de informação diretamente para a base de dados *Postgres*. Este conector elimina a necessidade de *software* adicional para o *message broker* comunicar com a base de dados. Posteriormente foi conectado ao *Postgres* uma interface HMI em *Grafana*, de forma a facilmente ler e mostrar os dados armazenados no *Postgres* (figura 5.10).

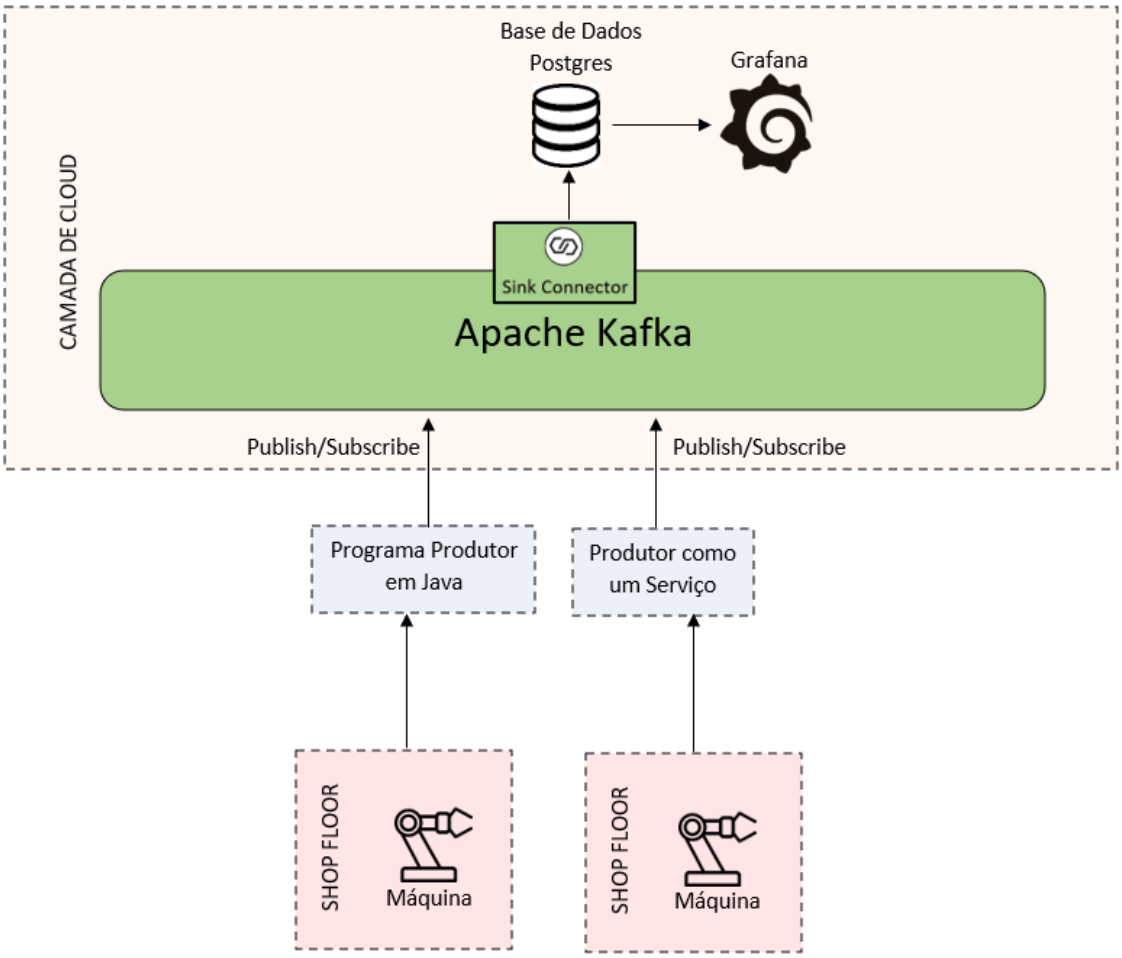

Figura 5.10: Implementação da arquitetura.

5.4.3 Publicação dos dados

Para a publicação dos dados das células robóticas nos tópicos específicos, um programa em *Java* foi criado em cooperação com a Introsys. Este programa utiliza as bibliotecas de *Java* do *Apache Kafka*, para primeiramente conectar-se ao *Kafka broker* e posteriormente publicar a informação nos tópicos existentes ou criar tópicos novos caso estes não existam.

Para ser possível conectar o programa ao *Kafka broker* é necessário atribuir um endereço IP e a porta de onde o *Kafka* está instanciado (ou localhost:port no caso de estar instanciado localmente), sendo possível configurar também diversos outros parâmetros, caso os valores pré-definidos não estejam de acordo com a implementação, como por exemplo os parâmetros de reenvio, alterando os valores das tentativas máximas de reenvio permitidas ou o tempo de espera entre cada reenvio.

O próximo passo no envio de informação é a criação do tópico, aqui é necessário fornecer o nome, o número de partições, o número de replicadores e o endereço IP do *Kafka broker*. O tópico não será criado caso já exista um tópico com o mesmo nome.

Por último é necessário definir o formato dos dados a enviar, os tópicos não necessitam

que seja especificado o formato, no entanto os *consumers* que irão ler a informação dos tópicos e as *streams* necessitam dessa informação. Para esta implementação foi escolhida o formato JSON, pois é um formato bastante conhecido e facilmente trabalhado por vários programas, além de ser armazenado facilmente e permite passar vários parâmetros ou guarda-los num único ficheiro.

5.4.4 Conectores utilizados no Apache Kafka

Uma das grandes vantagens do *Apache Kafka* é a utilização dos seus serviços de *streaming* e a utilização dos conectores, estes conectores podem ser rapidamente configurados, permitindo mover uma grande quantidade de dados para dentro (*source connector)* ou fora (*sink connector)* do *Kafka*. A utilização de conectores remove programas intermédios, cria novas funcionalidades de utilização das *streams* como um serviço e mantêm o fluxo de dados com baixa latência. Estes conectores podem ainda ser configurados para alterar dados (de forma continua ou em casos específicos), podendo ser utilizado para alterar o nome de dados, conjuntos de dados ou procurar dados específicos dentro da *stream* e alterar para outros valores, por exemplo alterar "null" para o valor "0".

Na implementação da presente demonstração, foram utilizados conectores (*sink connector*) JDBC, de forma a conectar os tópicos onde os dados são escritos à base de dados *Postgres*, permitindo a criação automática das tabelas na base de dados e o preenchimento das mesmas. Torna-se importante salientar que existem diferentes conectores e diferentes bases de dados que poderiam ser utilizadas, um exemplo seria utilizar uma base de dados não relacional, como *MongoDB* existindo também um conector do *Kafka* para esse efeito. Em caso da situação ser bastante específica, é possível criar o próprio conector e instanciá-lo no *Kafka*.

5.4.5 Instanciar a demonstração

A instanciação do projeto foi realizado na *cloud* interna da Introsys, através da *Docker* como referido e aprofundado no capitulo 5.2. Após o *Apache Kafka*, a base de dados *Postgres* e o *Grafana* estarem inicializados é necessário configurar as *streams* e os conectores no *Apache Kafka*, estas *streams* transformam os dados recebidos no tópico no formato JSON para formato Avro para que seja possível o conector JDBC ler a informação da *stream* e automaticamente escrever os dados em comandos de SQL na base de dados *Postgres*. As *streams* necessitam que o nome, tipo e formato dos dados sejam especificados, bem como as alterações que se pretenda executar no fluxo de dados (figura 5.11).

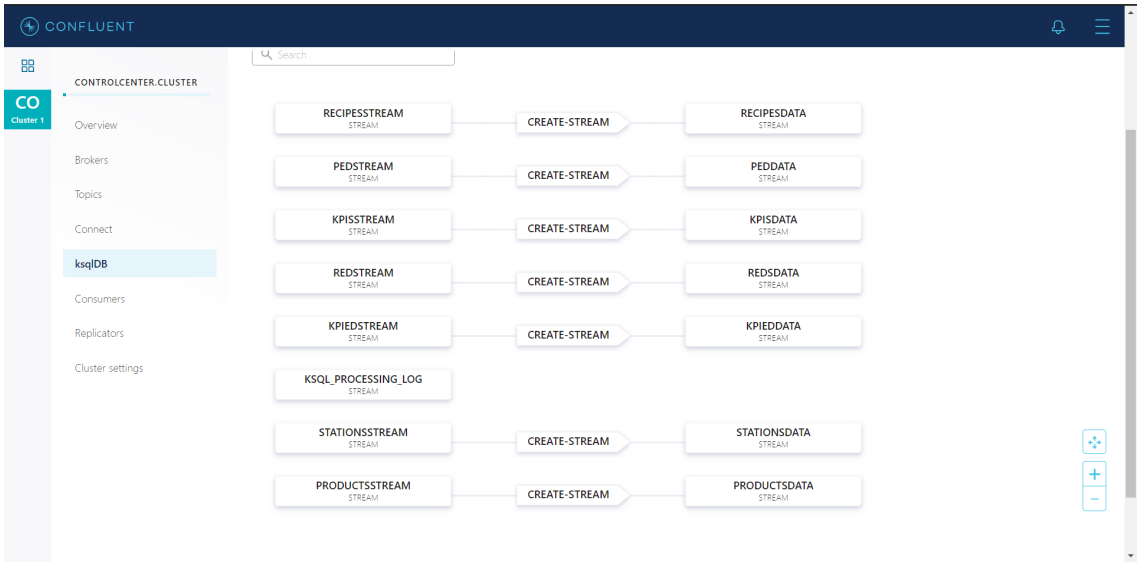

Figura 5.11: *Streams* realizadas no *Kafka* visualizadas através da plataforma *Confluent*.

Analisando a figura 5.11 é possível observar no *Confluent* o fluxo dos dados a passar por todas as *streams* criadas. Pode-se constatar que as primeiras *streams* (RECIPESSTREAM, PEDSTREAM, KPISSTREAM, REDSTREAM, KPIEDSTREAM, STATIONSSTREAM, PRODUCTSSTREAM) recebem os dados originais em formato JSON, que são por sua vez convertidos em formato Avro no segundo conjunto de *streams* (RECIPESDATA, PEDDATA, KPISDATA, REDSDATA, KPIEDDATA, STATIONSDATA, PRODUCTSDATA).

Após toda a informação ser convertida, ela é enviada para os conectores criados (figura 5.12).

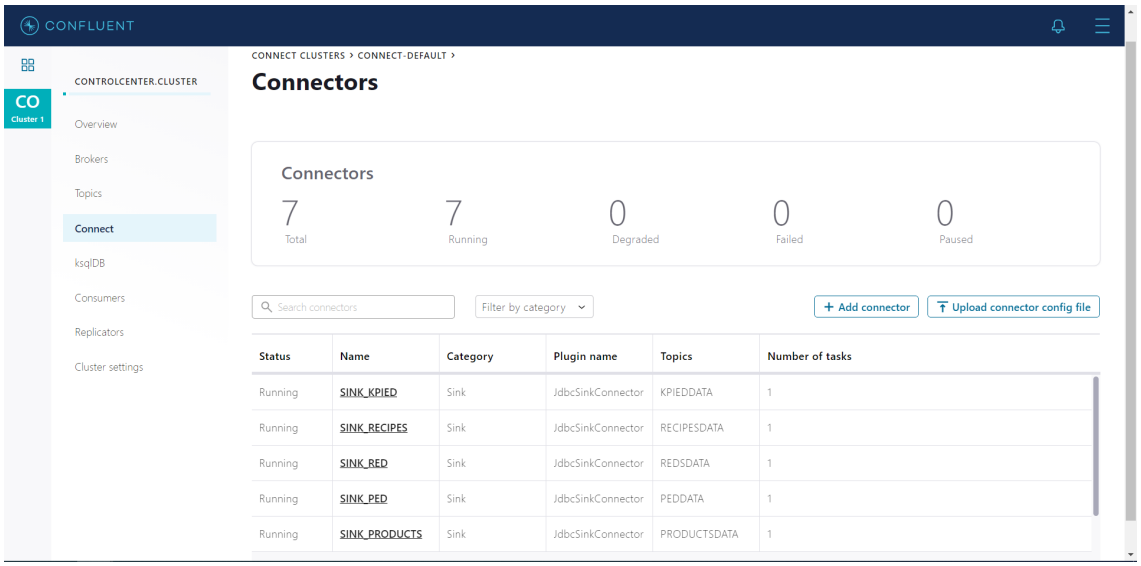

Figura 5.12: Conectores realizados no *Kafka* visualizadas através da plataforma *Confluent*.

Utilizando novamente a plataforma *Confluent* é possível observar na figura 5.13 todos os tópicos criados. E de notar que as *streams* também criam tópicos, não sendo estes

utilizados por nenhuma ferramenta externa ao *Kafka*.

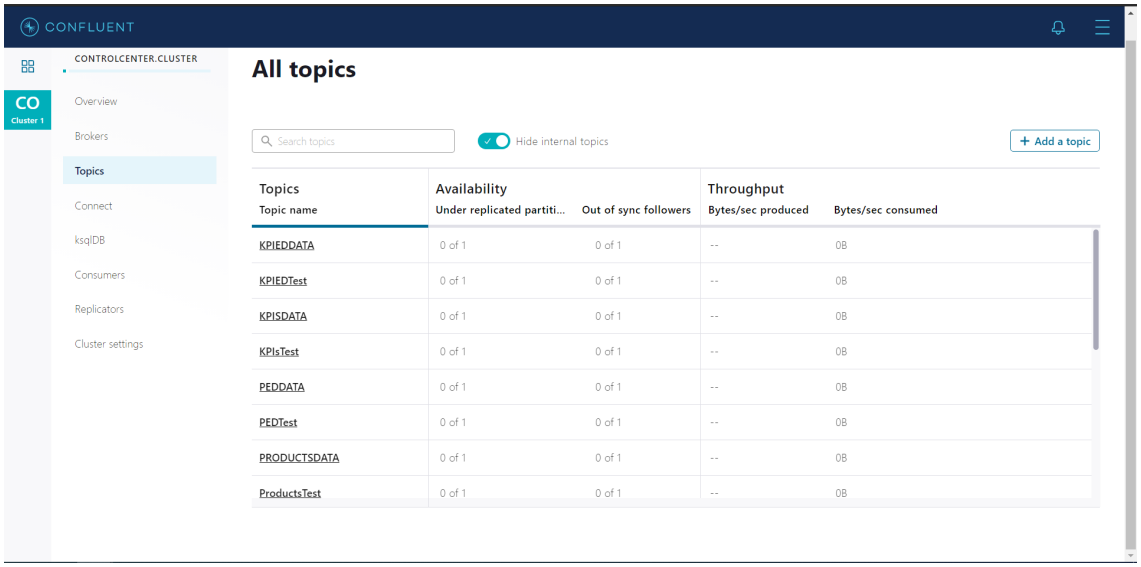

Figura 5.13: Tópicos existentes no *Kafka* visualizadas através da plataforma *Confluent*.

5.4.6 Visualização dos dados

Após o envio de alguns dados para os tópicos de teste, observa-se na figura 5.14 o preenchimento das tabelas automaticamente com os dados enviados pelos *publishers*. São demonstradas todas as tabelas criadas, seguido da informação presente na tabela "Product_Execution_Data".

```
adam=# \dt
              List of relations
 Schema |          Name          | Type  |  Owner
--------+------------------------+-------+----------
 public | KPI_Execution_Data     | table | postgres
 public | KPIs                   | table | postgres
 public | Product_Execution_Data | table | postgres
 public | Products               | table | postgres
 public | Recipe_Execution_Data  | table | postgres
 public | Recipes                | table | postgres
 public | Stations               | table | postgres
(7 rows)

adam=# select * from "Product_Execution_Data" ;
      UNIQUEID       | ORDERID |            PRODUCTID            |         STARTTIME         |          ENDTIME          |     STATUS     |        REGISTERED
---------------------+---------+---------------------------------+---------------------------+---------------------------+----------------+---------------------------
 152684yrhudfnchsug-1  | 1       | wqxzfafara-hytgrfdsasadqe-1201  | 2021-04-21T13:25:43+01:00 | 2021-04-21T14:25:43+01:00 | TO_BE_PRODUCED | 2021-04-21T13:25:43+01:00
 152684yrhudfnchsug-2  | 2       | wqxzfafara-hytgrfdsasadqe-1202  | 2021-04-21T13:25:43+01:00 | 2021-04-21T15:25:43+01:00 | TO_BE_PRODUCED | 2012-04-21T13:25:43+01:00
 152684yrhudfnchsug-3  | 3       | wqxzfafara-hytgrfdsasadqe-1203  | 2021-04-21T13:25:43+01:00 | 2021-04-21T16:25:43+01:00 | TO_BE_PRODUCED | 2012-04-21T13:25:43+01:00
 152684yrhudfnchsug-4  | 4       | wqxzfafara-hytgrfdsasadqe-1204  | 2021-04-21T13:25:43+01:00 | 2021-04-21T17:25:43+01:00 | TO_BE_PRODUCED | 2012-04-21T13:25:43+01:00
 152684yrhudfnchsug-5  | 5       | wqxzfafara-hytgrfdsasadqe-1205  | 2021-04-21T13:25:43+01:00 | 2021-04-21T18:25:43+01:00 | TO_BE_PRODUCED | 2012-04-21T13:25:43+01:00
 152684yrhudfnchsug-6  | 6       | wqxzfafara-hytgrfdsasadqe-1206  | 2021-04-21T13:25:43+01:00 | 2021-04-21T19:25:43+01:00 |                | 2012-04-21T13:25:43+01:00
 152684yrhudfnchsug-7  | 7       | wqxzfafara-hytgrfdsasadqe-1207  | 2021-04-21T13:25:43+01:00 | 20121-04-21T20:25:43+01:00 |                | 2012-04-21T13:25:43+01:00
 152684yrhudfnchsug-8  | 8       | wqxzfafara-hytgrfdsasadqe-1208  | 2021-04-21T13:25:43+01:00 | 2021-04-21T21:25:43+01:00 |                | 2012-04-21T13:25:43+01:00
 152684yrhudfnchsug-9  | 9       | wqxzfafara-hytgrfdsasadqe-1209  | 2021-04-21T13:25:43+01:00 | 2021-04-21T22:25:43+01:00 |                | 2012-04-21T13:25:43+01:00
 152684yrhudfnchsug-10 | 10      | wqxzfafara-hytgrfdsasadqe-12010 | 2021-04-21T13:25:43+01:00 | 2021-04-21T23:25:43+01:00 |                | 2012-04-21T13:25:43+01:00
(10 rows)

adam=#
```

Figura 5.14: Visualização do conteúdo da base de dados *Postgres*.

Utilizando o *software Grafana*, uma interface gráfica foi desenvolvida e integrada com o *Postgres*. Foram criados diversos *dashboard*, cada um utilizando diferentes dados, de forma a ser possível visualizar aspetos importantes ou realçar dados que possam revelar algum erro, facilitando a leitura de dados e deteções de erros.

Os *dashboard* são criados utilizando *SQL querys* e nomeando os diferentes dados e como devem ser apresentados. Na figura 5.15 estão representados três *dashboard*, estes contêm a informação de cada dos produtos que faltam produzir e as receitas executadas por cada estação.

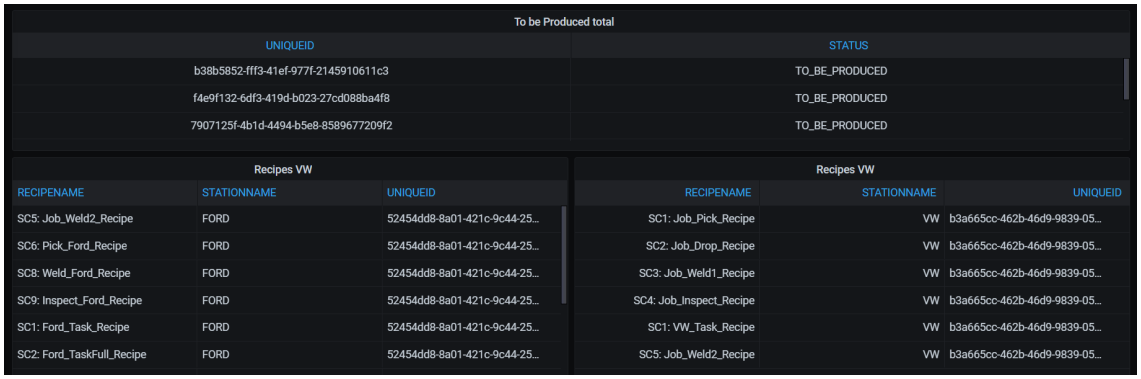

Figura 5.15: *Dashboard* executados no *Grafana* com a informação das receitas e produtos a serem produzidos.

É possível observar na figura 5.16, três *dashboard* com gráficos de consumo energético ao longo do tempo. Um gráfico com a informação do consumo total das duas estações e um gráfico com o consumo energético total, diferenciando a célula(azul), robô(amarelo) e sensores adicionais (verde), para cada estação.

Figura 5.16: *Dashboard* executados no *Grafana* com a informação do consumo energético ao longo do tempo.

Na figura 5.17, estão presentes seis *dashboard*, apresentando o consumo médio, as receitas executadas com um consumo 10% abaixo desse consumo médio e as receitas executadas com um consumo 10% acima do consumo médio, para cada célula robótica.

Figura 5.17: *Dashboard* executados no *Grafana* com a informação do consumo energético de cada receita executada.

5.5 Implementação: Testes de Stress no Kafka

No âmbito de entender um pouco mais as limitações do *Apache Kafka* foram realizados alguns testes de forma a entender a latência máxima alcançada pelo *message broker*. Para este teste foi utilizado a *Docker* para instanciar o *Apache Kafka* como explicado em detalhe na secção 5.2. Após o *Apache Kafka* estar instanciado foram desenvolvidos *scripts* em *Python 3.7* que atuavam como *Producers* e *Consumers*, os *Producers* enviavam mensagens individuais com diferentes intervalos entre elas até totalizar 10500 mensagens, após todas estas mensagens terem sido enviadas uma a uma com 100ms de intervalo entre elas, o intervalo era alterado e novamente eram enviadas 10500 mensagens, agora com 50ms entre elas e assim sucessivamente. Para estes testes foram assim utilizados 6 intervalos: 100ms, 50ms, 10ms, 5ms, 2ms, 1ms.

Este teste foi assim dividido em duas partes, a primeira apenas com um *Producer* e um *Consumer* no mesmo tópico e o segundo teste com 5 *Producers* e 5 *Consumers*, cada par num tópico diferente, de forma a simular um acréscimo de carga no sistema.

As mensagens enviadas são mensagens de dimensão reduzida, sendo constituídas por um ID e o *timestamp* de quando a mensagem foi enviada para o *Kafka*, totalizando cerca de $\leq 1\,\text{kB}$. Ao serem recebidas pelo *Consumer* é gerado um *timestamp* e colocado também na mensagem, sendo esta mensagem guardada por sua vez num *array*. Após

todas as mensagens terem sido recebidas esse *array* é escrito num ficheiro e guardado no computador.

Para a analise dos dados são retiradas as primeiras 500 mensagens, estas mensagens, principalmente em velocidades de envio mais baixas como 1ms ou 2ms, são enviadas muito rapidamente e acontece que o *Apache Kafka* acumula bastantes mensagens antes de inicializar a comunicação com o *consumer* e começar o envio das mensagens. De forma a poder ser obtido uma melhor comparação entre as diferentes velocidades, foram eliminadas essas primeiras mensagens.

Por fim este teste foi repetido 10 vezes, para existir uma amostra suficientemente grande para ser possível tirar algumas conclusões a respeito da latência do *Apache Kafka* e da instanciação utilizada.

Na tabela está uma representação dos testes que foram realizados:

Tabela 5.3: Testes realizados no *Apache Kafka*

Nº *Producer/Consumer*	Nº tópicos	Intervalos entre mensagens (ms)	Nº de mensagens	Nº de repetições
1	1	100, 50, 10, 5, 2, 1	10500	10
5	5	100, 50, 10, 5, 2, 1	10500	10

5.5.1 Teste 1 *Producer/Consumer*

Depois de todos os testes terem sido realizados, foi executada uma média de todas as repetições e compilado o gráfico da figura 5.18. Neste gráfico é possível observar que o X representa a média, a linha no centro da caixa representa a mediana, a aresta superior da caixa representa a mediana do terceiro quartil e a aresta inferior da caixa representa a mediana do primeiro quartil, as linhas que saem fora da caixa representam o valor máximo e o mínimo e por fim os pontos representam os *outliers*, isto significa que estão 1,5 vezes maiores que a mediana do primeiro ou do terceiro quartil.

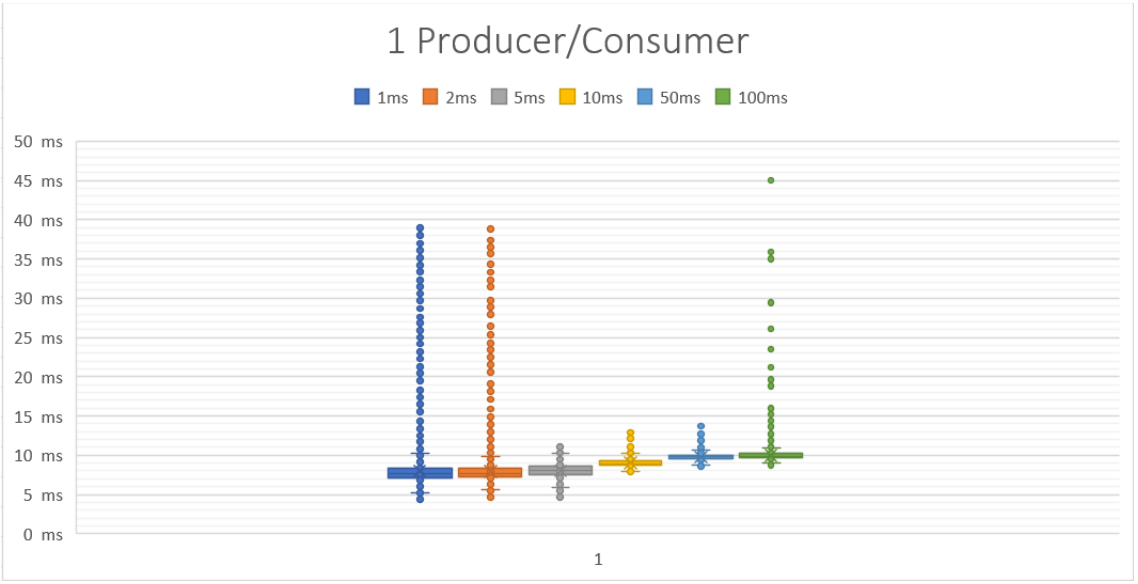

Figura 5.18: Gráfico dos testes para 1 *Producer* e 1 *Consumer*

Observe-se na tabela 5.4 a média e o desvio padrão de cada um dos intervalos realizados nos testes:

Tabela 5.4: Tabela dos testes 1 *Producer* e 1 *Consumer*

Intervalo entre mensagens (ms)	Média (ms)	Desvio Padrão
1	8,02	2,56
2	7,91	2,02
5	8,03	0,84
10	9,00	0,45
50	9,80	0,41
100	9,99	0,84

Quando se observa a tabela 5.4 é possível constatar que o tempo mais rápido de envio é cerca de 8ms e consoante as mensagens deixam de ser enviadas para o *Kafka* tão rapidamente, ou seja deixa de existir mensagens em espera para serem enviadas, o *Apache Kafka* envia a uma velocidade de cerca de 9-10ms. Uma observação importante é que com as configurações pré-definidas o *Apache Kafka* permite o envio de conjuntos de mensagens quando não consegue transmitir uma a uma, este parâmetro torna-se especialmente útil em circunstâncias como a do envio em intervalos de 1ms e 2ms, permitindo um envio destas mensagens com uma latência relativamente baixa.

Na figura 5.18 é possível observar que o *Apache Kafka* possui bastantes valores considerados como *outliers* nos intervalos de 1ms e 2ms, bem como a velocidade média é mais rápida com 2ms que com 1ms, estes pontos podem se dever a algumas limitações dos testes que serão abordadas mais a frente.

5.5.2 Teste 5 *Producer/Consumer*

A semelhança da secção anterior foi realizado um gráfico com a média de todos os *consumers* dos diferentes tópicos 5.19. Este teste foi concebido para obter uma melhor representação do funcionamento em paralelo com diferentes *Producers* e *Consumers*.

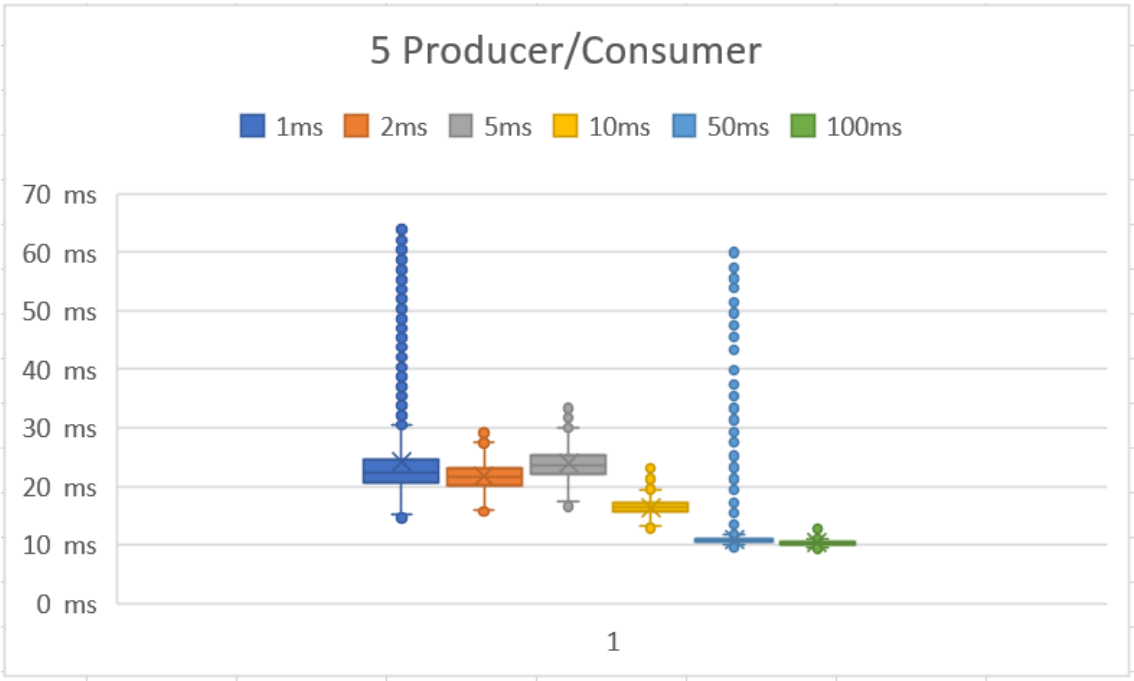

Figura 5.19: Gráfico dos testes para 5 *Producer* e 5 *Consumer*

Observe-se na tabela 5.5 a média e o desvio padrão de cada um dos intervalos realizados nos testes:

Tabela 5.5: Tabela dos testes 5 *Producer* e 5 *Consumer*

Intervalo entre mensagens (ms)	Média (ms)	Desvio Padrão
1	24,14	7,15
2	21,74	2,10
5	23,85	2,36
10	16,43	1,17
50	10,96	1,72
100	10,33	0,35

É possível observar na tabela 5.5 que os tempos aumentaram drasticamente face aos testes da secção 5.5.1, começando a igualar os tempos da secção anterior nos intervalos de 50ms e 100ms. O desvio padrão está também muito mais elevado nestas amostras, particularmente no intervalo de 1ms.

Na gráfico da figura 5.19 é possível observar um elevado numero de *outliers*, principalmente no intervalo de 1ms, alem de os das médias dos intervalos de 1ms, 2ms, 5ms e 10ms estarem muito altas, estes problemas podem dever-se as limitações dos testes, visto

que o *Apache Kafka* é conhecido por ser um dos *message brokers* mais rápidos [Nikhil e Chandar 2020].

5.5.3 Observações Finais e Limitações

O objetivo inicial desta implementação era executar alguns testes de forma a colocar em perspetiva a capacidade do *Apache Kafa* na rapidez do envio das mensagens. Contudo à medida que mais testes foram conduzidos tornou-se claro que existiam limitações em termos do ambiente de teste. Algumas limitações que podem prejudicar os testes realizados são:

- A falta de garantia da precisão dos intervalos, devido aos próprios processos do sistema operativo [Python Software Foundation 2009].
- A utilização do *software Docker* pode comprometer o uso de recursos por parte do *Apache Kafka*.
- A utilização de apenas um computador para correr os *scripts* de *Producer/Consumer* e do próprio *message broker* pode levar a divisão de recursos do computador contribuindo para o aparecimento de *outliers* nos dados.

Outro ponto importante a referir é as especificações da máquina no qual os testes foram realizados (tabela 5.6), sendo importante para a repetição dos mesmos testes ou a percepção do impacto do *hardware* nos testes realizados.

Tabela 5.6: Especificações do computador de teste

Sistema Operativo	Disco Rígido	RAM	Processador	Placa Gráfica
Windows 10	SSD 256GB	16GB	Intel Core i7-6700HQ 2.60Ghz	Nvidia Geforce GTX 950M

Apesar de algumas limitações já aqui abordadas, esta implementação concede uma percepção pessimista do desempenho do *Apache Kafka*, isto significa que em condições com uma máquina dedicada a apenas um *Producer*, uma apenas dedicada a um *Consumer* e uma máquina apenas dedicada ao *Apache Kafka* existe a possibilidade de serem obtidos resultados muito melhores.

Capítulo 6

Discussão

Irá ser descrito neste capítulo como a arquitetura pode beneficiar o avanço industrial, o desenvolvimento e a implementação da Indústria 4.0, apresentando não apenas a ideia conceptual do sistema, mas também exemplos reais aplicados ao processo industrial moderno. Será também referenciado um artigo desenvolvido em paralelo com a dissertação ao qual foram aplicados os mesmos princípios.

6.1 Discussão dos resultados

A Indústria 4.0 permite que mais produtos sejam produzidos de forma mais eficiente com melhor qualidade e melhor impacto a nível social (como a melhoria das condições dos trabalhadores) e ambiental, contudo muitos obstáculos ainda existem que impedem ou dificultam que este objetivo seja alcançado. Um destes desafios é a integração de diferentes sistemas, em que muitos encontram-se isolados a trabalhar de forma independente e desconectados da rede. Para integrar dois ou mais sistemas existem diversas abordagens, no entanto a comunicação direta entre estes sistemas pode trazer complicações, como todo o processo funcionar à velocidade do sistema mais lento, a dificuldade de comunicação entre sistemas ou não existir forma de utilizar as comunicações como um serviço [Margherita e Braccini 2020].

A abordagem proposta nesta dissertação, implementa um *message broker* instanciado em *cloud*, de forma a harmonizar a integração de diferentes sistemas com o objetivo de todos eles trabalharem como um só, através da partilha de dados e conhecimento. Esta partilha de informação permite que diferentes *software*, como por exemplo *software* orientados a simulação e logística, utilizarem os dados em tempo-real e adaptarem os processos de forma a melhorar o consumo de energia, produtividade, entre outros.

A abordagem proposta sublinha o facto de não ser necessário utilizar qualquer tecnologia utilizada na execução das demonstrações, desta forma é possível realizar uma integração mais rápida, fácil e eficaz, utilizando os *software* que cada empresa achar mais conveniente utilizar.

Na implementação proposta foi utilizada uma abordagem com o *message broker Apache Kafka*, na qual foi realizada uma primeira demonstração com o objetivo de efetuar alguns testes de laboratório, demonstrando uma integração entre diferentes *software* utilizando diferentes tecnologias, neste caso, *Java* e *Python*. Na segunda demonstração foi utilizado um ambiente industrial real com duas células robóticas de indústria automóvel, no qual foi enviado dados para tópicos do *kafka*, tornando esta informação imediatamente acessível numa base de dados *Postgres*. Os dados guardados na base de dados são mostrados numa interface gráfica na qual as informações das células podem ser vistas em tempo real. Na terceira demonstração foram realizados alguns testes para perceber qual a latência máxima do sistema utilizado nas demonstrações anteriores, contudo existe um limite de *hardware* que torna difícil perceber até que ponto se torna um limite do *Apache Kafka* e não dos recursos utilizados. Ainda assim a terceira demonstração serve para ter um patamar base, a qual num cenário real com recursos individuais dados a cada componente (*Producer*, *Consumer* e *Apache Kafka*) pode ser possível obter melhores resultados.

Os testes realizados demonstram o potencial de utilizar as abordagens descritas, num contexto industrial bem como a sua utilidade em contextos mais específicos como eficiência energética.

6.1.1 Limitações

Um aspeto relevante em relação a toda a integração, foi a utilização de um modelo de dados. Nas abordagens apresentadas todos os *software* têm de conhecer o formato dos dados que é partilhada e enviada pelos *publishers*. Uma solução para esta limitação do problema pode passar por um *software* modelar a informação recebida antes de ser publicada no *broker*, no entanto este processo aumenta bastante a complexidade à medida que mais sistemas podem ser integrados com diferentes necessidades de informação.

Um outro aspeto importante de mencionar é a necessidade de mais testes, com diferentes *hardware*, *software* e requisitos. Apesar de terem sido realizados testes em ambiente industrial, respondendo a todos os requisitos tanto da Introsys como da equipa de investigação, a amostra continua a ser muito baixa, sendo necessários mais testes em ambiente real e mais cenários para validar e expandir a utilização da arquitectura.

Por último e tal como mencionado na secção 5.5.3, existem limitações de *hardware* que podem penalizar a obtenção da maior velocidade possível do *software* implementado, a utilização de melhores equipamentos para casos de grande velocidades bem como a realização de mais testes para compreender o limite deste *software*.

6.2 Artigo desenvolvido

Com a arquitetura e demonstrações presentes nesta dissertação, foi realizado um artigo, que foi aceite e publicado na *Multidisciplinary Digital Publishing Institute* (MDPI) no *Energies journal* podendo ser encontrado em [Rocha et al. 2021]. Este artigo teve como colaboradores elementos da equipa de investigação da UNINOVA, nomeadamente o Professor Doutor José Barata, Professor Doutor André Rocha, o mestre e candidato a doutor Duarte Alemão e o candidato a mestre Nelson Freitas. A Introsys colaborou também na realização do artigo, auxiliando na descrição e funcionamento das estruturas internas da empresa, com o contributo de Magno Guedes e Renato Martins.

O artigo mencionado, aborda a utilização da arquitetura no âmbito da eficiência energética e como a utilização da Indústria 4.0 pode ajudar no desenvolvimento de soluções que monitorizem e tornem mais eficiente o consumo energético na indústria. É mencionado como esta arquitetura pode transmitir os dados necessários a *software* de simulação e logística de forma a melhorar a eficiência energético da empresa. É demonstrado, no artigo, da mesma forma que na presente dissertação, como foram realizadas todas as demonstrações bem como os testes, *software* e *hardware* utilizados, de modo a concretizar a arquitetura proposta e demonstrar os resultados.

Capítulo 7

Conclusão e trabalho futuro

7.1 Conclusão

A indústria e todo o processo industrial estão a atravessar uma grande mudança, os consumidores necessitam de produtos adaptados e personalizados aos seus gostos ou necessidades e soluções inovadoras têm de ser desenvolvidas. A Indústria 4.0 está a alterar este paradigma, com tecnologias emergentes que cada vez mais tornam o desafio da mudança numa realidade, conceitos como *Cyber-Physical Systems* e *Cloud Manufacturing* tornam a ideia de manufatura como um serviço, cada vez mais real.

Porém a ideia de manufatura como um serviço, só pode avançar com uma melhoria da interconectividade e cooperação entre diferentes empresas e serviços. É aqui revelada a importância de um *message broker* num serviço de *Cloud*, procurando assim assegurar a troca de dados entre todas as partes num ciclo de produção.

É desenvolvido ao longo da dissertação uma arquitetura que soluciona e simplifica o problema fundamental da troca de informação, nomeadamente entre o *shopfloor* e a *software* em *cloud*, mas que pode ser utilizado em qualquer comunicação. Foram apresentadas demonstrações e testes nas quais duas células robóticas de diferentes fabricantes com diferentes *softwares* conseguiram enviar os seus dados de execução e consumos energéticos para o *message broker*, que de seguida foram armazenados numa base de dados e demonstrados num HMI, reforçando a utilidade e demonstrando a utilização da arquitetura. É importante salientar que com este método de integração, facilmente outras células robóticas dos mesmos ou diferentes fabricantes poderiam se conectar ao *message broker* e popular automaticamente, em junção com as células já existentes, a base de dados.

Foi desenvolvido um artigo com a informação presente nesta dissertação, com a colaboração da equipa de investigação e dos engenheiros da Instroys, sendo este artigo aprovado e publicado pela MDPI.

É, desta forma, possível concluir que o objetivo da dissertação foi alcançado, tendo sido desenvolvido um sistema capaz de integrar diversas indústrias bem como diversos *hardware*, *software* ou aplicações, de modo a que a partilha de recursos digitais torne todas as empresas do ciclo de produção mais eficazes, eficientes e autónomas.

7.2 Trabalho Futuro

Como trabalho futuro existem diversas áreas que foram exploradas ao longo da dissertação que podem ser aprofundadas. Uma dessas áreas é a expansão das demonstrações apresentadas como a utilização de outros métodos de envio de dados para o *message broker*, como por exemplo uma placa de desenvolvimento (*Arduino*, *RasberryPi*).

Outra área importante desenvolver é a expansão dos testes de stress desenvolvidos na dissertação no 5.5, num ambiente mais controlado e com cargas que se assemelham as de um ambiente industrial, utilizando diversos computadores para a simulação dos diferentes componentes.

Por fim, como última sugestão, procurar expandir a utilização e compreensão dos recursos que o *Apache Kafka* possui. A utilização do *Kafka REST API*, a realização de conectores por parte do utilizador e implementação na plataforma, bem como outras funcionalidades que não foram consideradas na realização do *.yml*. Estas funcionalidades e conhecimentos podem ser úteis na resolução de problemas de integração que possam surgir tornando se importante um estudo cuidado de algumas funcionalidades do *message broker* utilizado.

Bibliografia

Apache/ActiveMQ (2019a). *ActiveMQ 5 Documentation*. URL: http://activemq.apache.org/components/classic/documentation (acedido em 09/01/2021).

— (2019b). *Code Overview*. URL: https://activemq.apache.org/code-overview (acedido em 16/12/2020).

— (2019c). *Cross Language Clients*. URL: https://activemq.apache.org/cross-language-clients.html (acedido em 09/01/2021).

Apache/Confluent (2017). *Powerd By*. URL: https://kafka.apache.org/powered-by (acedido em 15/06/2021).

— (2021). *What is Confluent Platform?* URL: https://docs.confluent.io/platform/current/platform.html (acedido em 16/12/2020).

Assari, M., J. Delaram e O. Fatahi Valilai (2018). "Mutual manufacturing service selection and routing problem considering customer clustering in Cloud manufacturing". Em: *Production and Manufacturing Research* 6.1, pp. 345–363. ISSN: 21693277. DOI: 10.1080/21693277.2018.1517056. URL: https://doi.org/10.1080/21693277.2018.1517056.

AVANGARD Project. (2020). "Deliverable 2.1 - Defenition of the cloud-based cyber-physical production system architecture". Em: URL: http://www.avangard-project.eu/.

Babovic, Z. e V. Milutinovic (2013). *Novel System Architectures for Semantic-Based Integration of Sensor Networks*. Vol. 90. Elsevier Inc., pp. 91–183. ISBN: 9780124080911. DOI: 10.1016/B978-0-12-408091-1.00002-6. URL: http://dx.doi.org/10.1016/B978-0-12-408091-1.00002-6.

Hussonnois, F. (2019). *Introduction to Apache Pulsar — Concepts, Architecture & Java Clients*. URL: https://medium.com/streamthoughts/introduction-to-apache-pulsar-concepts-architecture-java-clients-71f1a30b75d6 (acedido em 16/12/2020).

Jyoti, S. (2018). *Getting Started with RabbitMQ: Python*. URL: https://blog.knoldus.com/getting-started-with-rabbitmq-python/ (acedido em 16/12/2020).

Lasi, H., P. Fettke, H. G. Kemper, T. Feld e M. Hoffmann (2014). "Industry 4.0". Em: *Business and Information Systems Engineering* 6.4, pp. 239–242. ISSN: 18670202. DOI: 10.1007/s12599-014-0334-4.

Liu, C. e P. Jiang (2016). "A Cyber-physical System Architecture in Shop Floor for Intelligent Manufacturing". Em: *Procedia CIRP* 56, pp. 372–377. ISSN: 22128271. DOI:

10.1016/j.procir.2016.10.059. URL: http://dx.doi.org/10.1016/j.procir.2016.10.059.

Margherita, E. G. e A. M. Braccini (2020). "Industry 4.0 Technologies in Flexible Manufacturing for Sustainable Organizational Value: Reflections from a Multiple Case Study of Italian Manufacturers". Em: *Information Systems Frontiers*. ISSN: 15729419. DOI: 10.1007/s10796-020-10047-y.

Mohajan, H. (2019a). "The First Industrial Revolution: Creation of a New Global Human Era". Em: *Journal of Social Sciences and Humanities* 5.4, pp. 377–387.

— (2019b). "The Second Industrial Revolution has Brought Modern Social and Economic Developments". Em: *Journal of Social Sciences and Humanities* 6.1, pp. 1–14.

Mowery, D. C. (2009). "Plus ca change: Industrial R&D in the "third industrial revolution"". Em: *Industrial and Corporate Change* 18.1, pp. 1–50. ISSN: 09606491. DOI: 10.1093/icc/dtn049.

Müller, J. M., O. Buliga e K. I. Voigt (2018). "Fortune favors the prepared: How SMEs approach business model innovations in Industry 4.0". Em: *Technological Forecasting and Social Change* 132.December 2017, pp. 2–17. ISSN: 00401625. DOI: 10.1016/j.techfore.2017.12.019. URL: https://doi.org/10.1016/j.techfore.2017.12.019.

Nikhil, A. e V. Chandar (2020). *Benchmarking Apache Kafka, Apache Pulsar, and RabbitMQ: Which is the Fastest?* URL: https://www.confluent.io/blog/kafka-fastest-messaging-system/ (acedido em 09/01/2021).

Posta, C. (2016). *What is Apache Kafka? Why is it so popular? Should you use it?* URL: https://techbeacon.com/app-dev-testing/what-apache-kafka-why-it-so-popular-should-you-use-it (acedido em 09/01/2021).

Python Software Foundation (2009). *time — Time access and conversions*. URL: https://docs.python.org/3.0/library/time.html (acedido em 22/07/2021).

RabbitMQ (2020a). *Clients Libraries and Developer Tools*. URL: https://www.rabbitmq.com/devtools.html (acedido em 09/01/2021).

— (2020b). *Which protocols does RabbitMQ support?* URL: https://www.rabbitmq.com/protocols.html (acedido em 09/01/2021).

Renart, E., D. Balouek-Thomert e M. Parashar (2017). "Pulsar: Enabling Dynamic Data-Driven IoT Applications". Em: *Proceedings - 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems, FAS*W 2017*, pp. 357–359. DOI: 10.1109/FAS-W.2017.173.

Ribeiro, L. (2017). "Cyber-physical production systems' design challenges". Em: *IEEE International Symposium on Industrial Electronics*, pp. 1189–1194. DOI: 10.1109/ISIE.2017.8001414.

Ribeiro, L. e M. Bjorkman (2018). "Transitioning from Standard Automation Solutions to Cyber-Physical Production Systems: An Assessment of Critical Conceptual and Technical Challenges". Em: *IEEE Systems Journal* 12.4, pp. 3816–3827. ISSN: 19379234. DOI: 10.1109/JSYST.2017.2771139.

Rocha, A. D., N. Freitas, D. Alemão, M. Guedes, R. Martins e J. Barata (2021). "Event-Driven Interoperable Manufacturing Ecosystem for Energy Consumption Monitoring". Em: *Energies* 14.12. ISSN: 1996-1073. DOI: 10.3390/en14123620. URL: https://www.mdpi.com/1996-1073/14/12/3620.

Rocha, A. D., J. Tripa, D. Alemao, R. S. Peres e J. Barata (2019). "Agent-based plug and produce cyber-physical production system - Test case". Em: *IEEE International Conference on Industrial Informatics (INDIN)* 2019-July, pp. 1545–1551. ISSN: 19354576. DOI: 10.1109/INDIN41052.2019.8972169.

Rüßmann, M. et al (2015). "Future of Productivity and Growth in Manufacturing". Em: *Boston Consulting* April. ISSN: 9783935089296. DOI: 10.1007/s12599-014-0334-4. arXiv: arXiv:1011.1669v3.

Schwab, K. (2016). "The Fourth Industrial Revolution: what it means and how to respond". Em: *World Economic Forum*, pp. 1–7. ISSN: 13489216.

Sharvari, T. e N. K. Sowmya (2019). "A study on Modern Messaging Systems- Kafka, RabbitMQ and NATS Streaming". Em: *arXiv*. arXiv: 1912.03715.

Stoja, S., S. Vukmirovič e B. Jelačić (2013). "Publisher/subscriber implementation in cloud environment". Em: *Proceedings - 2013 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2013*, pp. 677–682. DOI: 10.1109/3PGCIC.2013.116.

Tao, F., L. Zhang, V. C. Venkatesh, Y. Luo e Y. Cheng (2011). "Cloud manufacturing: A computing and service-oriented manufacturing model". Em: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 225.10, pp. 1969–1976. ISSN: 09544054. DOI: 10.1177/0954405411405575.

Tyler, T. (2018). *Introducing Liftbridge: Lightweight, Fault-Tolerant Message Streams*. URL: https://bravenewgeek.com/introducing-liftbridge-lightweight-fault-tolerant-message-streams/ (acedido em 16/12/2020).

Weyer, S., T. Meyer, M. Ohmer, D. Gorecky e D. Zühlke (2016). "Future Modeling and Simulation of CPS-based Factories: an Example from the Automotive Industry". Em: *IFAC-PapersOnLine* 49.31, pp. 97–102. ISSN: 24058963. DOI: 10.1016/j.ifacol.2016.12.168. URL: http://dx.doi.org/10.1016/j.ifacol.2016.12.168.

Wu, D., M. J. Greer, D. W. Rosen e D. Schaefer (2013). "Cloud manufacturing: Strategic vision and state-of-the-art". Em: *Journal of Manufacturing Systems* 32.4, pp. 564–579. ISSN: 02786125. DOI: 10.1016/j.jmsy.2013.04.008. URL: http://dx.doi.org/10.1016/j.jmsy.2013.04.008.

Zhang, L., Y. Luo, F. Tao, B. H. Li, L. Ren, X. Zhang, H. Guo, Y. Cheng, A. Hu e Y. Liu (2014). "Cloud manufacturing: a new manufacturing paradigm". Em: *Enterprise Information Systems* 8.2, pp. 167–187. ISSN: 17517575. DOI: 10.1080/17517575.2012.683812.

Zhong, R. Y., X. Xu, E. Klotz e S. T. Newman (2017). "Intelligent Manufacturing in the Context of Industry 4.0: A Review". Em: *Engineering* 3.5, pp. 616–630. ISSN: 20958099. DOI: 10.1016/J.ENG.2017.05.015. URL: http://dx.doi.org/10.1016/J.ENG.2017.05.015.