escola superior de tecnologia e gestão
instituto politécnico de leiria

Projeto de Mestrado em Engenharia Eletrotécnica

Departamento de Engenharia Eletrotécnica e de Computadores

Cloud em Sistemas Industriais

André Ramos Fortes dos Santos

Orientado por Doutor Eliseu Ribeiro

Leiria, setembro de 2018

Esta página foi intencionalmente deixada em branco

escola superior de tecnologia e gestão
instituto politécnico de leiria

Projeto de Mestrado em Engenharia Eletrotécnica

Departamento de Engenharia Eletrotécnica e de Computadores

Cloud em Sistemas Industriais

André Ramos Fortes dos Santos

Projeto de Mestrado realizado sob a orientação do Doutor Eliseu Manuel Artilheiro Ribeiro, Professor da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria.

Leiria, setembro de 2018

Esta página foi intencionalmente deixada em branco

Resumo

O *cloud computing* surgiu através da evolução da computação e comunicação. É visto como uma melhoria em relação aos servidores tradicionais. Permite uma melhor otimização e controlo dos recursos e uma flexibilidade no serviço que é contratado [1]. O *cloud computing* é uma tecnologia que acrescenta valor ao conceito de Indústria 4.0, potenciando a computação, monitorização, gestão e controlo industrial [2].

O propósito deste projeto é colocar um *Programmable Logic Controller* (*PLC*) a comunicar com um serviço *cloud* através de um dispositivo intermediário. Os *PLCs* são utilizados para controlo de máquinas e processos industriais, mas usualmente ficam isolados do exterior no âmbito de fornecer informação. Ao interligar o *PLC* com o serviço *cloud*, a informação pretendida pode ser enviada para uma base de dados na *cloud*, para posteriormente ser analisada.

A revisão bibliográfica aborda o *cloud computing* segundo o modelo do *National Institute of Standards and Technology* (NIST) e a sua integração na Indústria 4.0. No fim da revisão bibliográfica são apresentados diversos fornecedores de *cloud* relevantes para o projeto. No caso de estudo é feita uma análise aos protocolos mais utilizados nos *PLCs* e na comunicação com a *cloud*. Após isso é definida uma arquitetura de como irá ser feita a comunicação. Por fim, serão analisados diversos dispositivos para servir de ponte entre o *PLC* e a *cloud*. O desenvolvimento da aplicação descreve como configurar o serviço *cloud* e explica os diversos passos para efetuar a comunicação e a gestão dos dados.

Os testes feitos mostram que a comunicação entre o *PLC* e o dispositivo desenvolvido é estável e robusta. A comunicação entre o dispositivo desenvolvido e o serviço *cloud* escolhido é satisfatória para aplicações com pouca cadência de dados. O serviço *cloud* escolhido mostrou-se rápido de configurar e de fácil utilização. A possibilidade de juntar facilmente a informação recolhida do *PLC* aos serviços disponibilizados pela *cloud*, permite que cada controlador seja integrado facilmente a serviços relacionados com monitorização, previsão e gestão de dados

Palavras-chave: Cloud Computing, Indústria 4.0, PLC, IoT

Esta página foi intencionalmente deixada em branco

Abstract

Cloud computing has emerged through the evolution of computing and communication. It is seen as an improvement over traditional servers, allows better optimization and control of resources and flexibility in the service [1]. Cloud computing is a technology that adds value to the concept of Industry 4.0, enhancing computing, monitoring, management and industrial control [2].

The purpose of this project is to put a *Programmable Logic Controller* (PLC) to communicate with a cloud service through an intermediary device. PLCs are used to control machines and industrial processes but are usually isolated from the outside in providing information. By interconnecting the PLC with the cloud service, the desired information can be sent to a database for later analysis.

The literature review of cloud computing is based on the *National Institute of Standards and Technology* (NIST) model and its integration into Industry 4.0. At the end of the literature review, several cloud providers relevant to the project are presented. In the case of study an analysis is made to the protocols most used in PLCs and in the communications with the cloud. After this is defined an architecture of how the communication will be done. Finally, several devices will be analyzed to serve as a bridge between the PLC and the cloud. The application development describes how to configure the cloud service and explains the various steps for communicating and managing the data.

The tests done show that the communication between the PLC and the developed device is stable and robust. The communication between the device and the chosen cloud service is satisfactory for applications with low cadence of data. The chosen cloud service was quick to set up and easy to use. The ability to easily gather information collected from the PLC to the services provided by the cloud allows each controller to be integrated with services related to monitoring, forecasting and data management.

Keywords: Cloud Computing, Industry 4.0, PLC, IoT

Esta página foi intencionalmente deixada em branco

Índice

Índice de Figuras

Índice de Tabelas

Lista de siglas

Sigla	Significado
ARM	Acorn RISC Machine
ASCII	American Standard Code for Information Interchange
AMQP	Advanced Message Queuing Protocol
CPU	Central Processing Unit
CSV	Comma-Separated Values
DB	Data Block
FullHD	Full High Definition
FB	Function Block
GPIO	General Purpose Input/Output
HDMI	High-Definition Multimedia Interface
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Security
I2C	Inter-Integrated Circuit
IaaS	Infraestruture as a Service
IDE	Integrated Development Environment
IoT	Internet Of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
LCD	Liquid Crystal Display
MQTT	Message Queuing Telemetry Transport
NIST	National Institute of Standards and Technology
PaaS	Platform as a Service
PLC	Programmable Logic Controller
RAM	Random Access Memory
SaaS	*Software* as a Service
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
TLS	Transport Layer Security
USB	Universal Serial Bus
UWP	Universal Windows Platform
XAML	eXtensible Application Markup Language

1. Introdução

Impulsionado pela evolução da Internet, o *cloud computing* surgiu a fim de melhorar a forma como os servidores e os seus recursos eram operados, fornecidos e utilizados. Com o aumento exponencial da utilização da Internet, vieram por acréscimo novas necessidades em relação ao uso dos servidores e à sua implementação. Um serviço *cloud* tem cinco características, quatro modelos de implementação e três tipos de serviços que permitem a sua distinção [3]. O uso de um serviço *cloud* proporciona ao utilizador algumas vantagens como: elasticidade, ferramentas de desenvolvimento, custo controlado, acesso generalizado à rede, entre outros [1].

A indústria ao longo do tempo foi moldada pela necessidade, passando por várias revoluções industriais que levaram à quarta revolução industrial, a Indústria 4.0. Esta sustenta-se em vários pilares que a caracterizam em relação às revoluções anteriores. Alguns dos pilares necessitam do *cloud computing*, a fim de se concretizarem e obterem o melhor desempenho possível. Atualmente, existem vários fabricantes na área de automação industrial a criar os seus próprios serviços de *cloud*, interligando dispositivos como: *PLCs*, computadores, sensores e dispositivos *Internet Of Things* (*IoT*). A computação em nuvem traz para a indústria, recursos e serviços que irão permitir melhorias significativas tanto na produção como na gestão e previsão [2].

Os dispositivos mais comuns para controlo de máquinas industriais são *PLCs*, deve-se especialmente à sua performance em ambientes rigorosos e a necessidades de controlo orientado para aplicações industriais. Nos últimos anos, os *PLCs* possuem cada vez mais capacidades de comunicação. Normalmente, utilizam protocolos indicados para o ambiente industrial e têm pouco suporte para outro tipo de comunicações [4].

Juntar a potencialidade da *cloud* com a do *PLC*, permite explorar a informação do chão de fábrica de uma forma simples e que pode levar a resultados significativos na produção, gestão, previsão e custos. Este tipo de aplicação tem como alvo as indústrias com máquinas controladas por *PLCs* e que pretendam ter um conhecimento mais alargado da produção de cada máquina.

Neste trabalho pretende-se tirar partido da integração do *cloud computing* na Indústria 4.0, a fim de conseguir a comunicação entre um *PLC* e um serviço *cloud*. Para efetuar a comunicação será desenvolvido um programa que irá operar num dispositivo *IoT*. O programa desenvolvido terá uma interface gráfica intuitiva para o operador configurar as comunicações e escolher as variáveis a serem transmitidas para a *cloud*. A comunicação com o *PLC* irá ser feita por cabo *Ethernet* e com a *cloud* através uma rede sem fios. Os protocolos de comunicação serão escolhidos com intuito de obter os melhores resultados, com a menor intervenção na programação do *PLC*.

O documento está organizado em quatro capítulos principais contendo vários subcapítulos. O capítulo *Cloud computing* apresenta o estado da arte dos temas *cloud computing* e Indústria 4.0. O seu objetivo é identificar a estrutura do *cloud computing*. Serão detalhados nos subcapítulos os diversos tipos, características e modelos de *cloud computing*, assim como a sua integração na indústria, remetendo para o tema Indústria 4.0. O capítulo *Caso de estudo* como o nome sugere, explica o caso de estudo deste projeto. É constituído por vários subcapítulos que apresentam a arquitetura da comunicação, os dispositivos testados e a escolha do dispositivo *IoT*. A concretização do programa e as configurações da *cloud*, explicando em detalhe os seus diversos passos, serão mostrados no capítulo *Desenvolvimento*. Por fim o capítulo *Testes à aplicação* mostra os testes efetuados ao programa, analisando a robustez das comunicações e da aplicação.

2. *Cloud computing*

2.1. Introdução ao *Cloud computing*

O *Cloud computing* é um conceito que alterou nos últimos anos a forma como os recursos computacionais são geridos e aplicados. A *cloud* não é um servidor diferente dos servidores tradicionais, pode-se considerar como uma evolução. Neste sentido, este conceito permite começar com poucos recursos, aumentando os mesmos à medida da necessidade do negócio. Além disso, a *cloud* foi impulsionada por vários fatores como: o aumento da taxa de transmissão da internet e redes móveis, melhoria do *hardware*, avanços no *software*, entre outros [1][5].

O *Cloud computing* traz algumas vantagens aos seus utilizadores, tais como: instalação, configuração, atualização, compatibilidade, poder computacional e custos. Contudo, é necessário ter em conta certas questões como a segurança informática e a capacidade da rede [5].

A computação em *cloud* é definida e caracterizada por vários aspetos [3]:

- 5 caraterísticas;
- 3 tipos de serviços;
- 4 modelos de implementação.

2.2. Características

A distinção do *cloud computing* para os servidores tradicionais, baseia-se em cinco características fundamentais. O serviço não pode ser *cloud*, se alguma das seguintes características não for cumprida.

1. **Autosserviço a pedido** – O serviço pedido pelo cliente é processado e finalizado sem a intervenção de recursos humanos. Assim sendo, os fornecedores de serviço podem gerir o consumo de recursos, mediante as suas necessidades [1] [5] [6] [3].
2. **Acesso generalizado à rede** – O serviço contratado pode ser acedido e configurado por qualquer dispositivo com ligação à internet, através de um *browser*. Atualmente, as *cloud*s estão localizadas em diversas partes do mundo como centros de dados. Desta forma, a fiabilidade aumenta e alcançam um alto desempenho [1] [5] [6] [3].
3. **Acesso partilhado a recursos** – Os recursos são dinâmicos, sendo alocados com a utilização do cliente. Se os recursos não estão a ser utilizados, podem ser alocados a outro cliente, dependendo do tipo de *cloud* [1] [6] [3]. Ao fornecer recursos dinâmicos, o fornecedor pode assim ter mais flexibilidade para gerir os seus próprios recursos e custos operacionais. Para complementar, os recursos de computação podem ser atribuídos e libertados na hora, sendo esta uma das características principais do *Cloud computing*. O provisionamento dinâmico de

recursos possibilita que os fornecedores de serviço adquiram recursos com base na procura atual. Este efeito pode ter como consequência um custo operacional mais reduzido [5].

4. **Elasticidade** – Permite ao cliente requisitar mais ou menos recursos instantaneamente. Assim, o recurso cresce com as necessidades da aplicação, sem intervenção [1] [6] [3].
5. **Serviço controlado** – O fornecedor do serviço informa o cliente sobre os recursos utilizados, ajudando-o a adequar o serviço à sua necessidade [6]. A faturação é baseada no conceito "paga apenas o que utiliza", ajustando-se às necessidades do cliente. Deste modo, é dado bastante destaque à gestão dos serviços [1] [5] [3].

2.3. Tipos de Serviços

Existem três principais tipos de serviço associados ao *Cloud Computing*:

- *Infrascture as a Service* (*IaaS*);
- *Plataform as a Service* (*PaaS*);
- *Software as a Service* (*SaaS*).

Estas categorias de serviço diferenciam-se nos recursos fornecidos ao utilizador e na sua aplicação [1] [5] [3].

2.3.1. *IaaS*

O serviço de mais baixo nível que se pode encontrar numa *cloud* é o *IaaS*. O *Infraestruture as a Service* ou *IaaS* remete para o uso de uma infraestrutura. O utilizador ao adquirir este serviço necessita de especificar os requisitos de *hardware* [1] [7] [8]. Os requisitos de *hardware* consistem em quantidades ou recursos, por exemplo: 8GB de memória, 500GB de disco e 3GHz de processador com 4 núcleos. O utilizador não tem acesso à escolha do *hardware* nem à sua montagem, apenas se limita a definir as quantidades necessárias para correr a sua aplicação [1].

Após a configuração estar efetuada, a infraestrutura está apta para ser utilizada. Como referido anteriormente, para ser um serviço de *cloud*, o recurso que o utilizador requisita tem de estar disponível em minutos, não necessitando de intervenção humana da parte do fornecedor [1].

Este tipo de nuvem possibilita a procura de recursos computacionais na forma de máquinas virtuais implantadas no *data center* de um fornecedor de nuvem. Este facto pode implicar a redução ou eliminação dos custos de capitais associados para os consumidores da respetiva *cloud*. Além disso, os clientes da *cloud* podem adicionar ou remover a capacidade da sua infraestrutura informática, atendendo às procuras de serviço ou de flutuação dos recursos. Nesta situação, só paga pela capacidade que foi utilizada [9].

Este serviço é semelhante a ter um servidor sem *software*. O cliente tem de instalar o sistema operativo, programas, bibliotecas, entre outros. Este recurso pode ser referido também como *Hardware as a Service* [7].

2.3.2. *PaaS*

A aplicação pretendida nem sempre necessita de uma infraestrutura que seja especificada pelo utilizador. O *Plataform as a Service* ou *PaaS* permite ao utilizador desenvolver a sua aplicação numa plataforma fornecida [1] [8].

Uma das principais vantagens do *PaaS* é a elasticidade do serviço, em que a gestão dos recursos computacionais é feita para correr a aplicação, evitando a alocação desnecessária de recursos. Outras facilidades deste serviço são bibliotecas, bases de dados, compiladores, entre outros, integradas na plataforma disponibilizada. Como é tudo fornecido na plataforma, os custos e a complexidade de instalação de *software* e infraestruturas é reduzido [1] [8].

Considerando que o utilizador pretende desenvolver uma página *web* para venda de produtos *online*. Ao contratar um serviço *PaaS* é disponibilizado um ambiente para o desenvolvimento, teste, implementação, gestão e atualização da página *web*. A *Google App Engine*, o *Microsoft Azure* são exemplos de fornecedores de *PaaS* [5].

2.3.3. *SaaS*

Se o utilizador apenas pretende uma aplicação, sem se preocupar com a infraestrutura ou plataforma que a alberga, estas condições levam ao uso de um serviço do tipo *Software as a Service* ou *SaaS* [1].

O *SaaS* é talvez o serviço mais simples para o utilizador, mas também o menos flexível. É um serviço que pode ser acedido através de um *browser* ou aplicação, por exemplo: *Google Apps*, *Dropbox*, *Office 365*, entre outros [7].

Para utilizar este serviço, não é necessário comprar, manter ou alterar *hardware*/*software*. O fornecedor do serviço tem a preocupação de ajustar o seu *hardware* e o *software* para aplicação. Como o *SaaS* permite o acesso através de um *browser*, o mesmo acrescenta mobilidade ao serviço, estando disponível em qualquer dispositivo com ligação internet e um *browser* [10] [7].

2.3.4. Resumo

Os serviços apresentados podem ser analisados segundo duas perspetivas. A primeira é do ponto de vista da quantidade de possibilidades que o utilizador dispõe ao adquirir o serviço. A segunda a quantidade de serviços pré configurados que dispõe.

O serviço *IaaS*, por fornecer apenas a infraestrutura, disponibiliza uma quantidade imensa de possibilidades ao utilizador. Inclusive o utilizador pode criar o seu serviço de *PaaS* e *SaaS* dentro dessa infraestrutura. O mesmo acontece para o *PaaS*, o utilizador pode criar um serviço *SaaS* dentro da plataforma, mas não consegue aceder à infraestrutura como no *IaaS*.

A Figura 1 apresenta a quantidade de aplicações que cada serviço dispõe. Quanto maior a área de cada serviço, mais aplicações e possibilidades são disponibilizadas. Também é percetível que os serviços menos divergentes ficam englobados dentro dos mais divergentes.

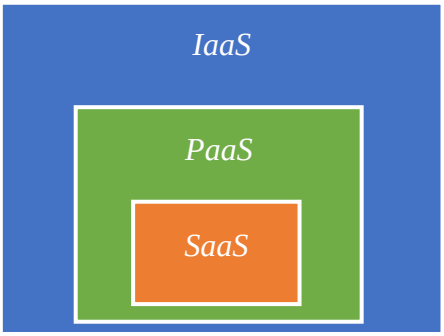

Figura 1 - Tipos de serviços

A segunda perspetiva pode ser encarada como a quantidade de serviços pré configurados que o utilizador dispõe ao adquirir o serviço. O *IaaS* permite ao utilizador desenvolver o que pretende, limitando por outro lado as aplicações que o fornecedor do serviço pode dispor, como está exemplificado na Figura 2.

Figura 2 - Serviços pré configurados [10]

2.4. Modelos de Implementação

Os modelos de implementação da *cloud* permitem ao cliente escolher a forma como quer que o serviço seja implementado nos servidores. Deste modo, os clientes podem escolher se preferem reduzir o custo operacional ou melhorar a confiabilidade e segurança [5]. Segundo o *National Institute*

of Standards and Technology (NIST), estão disponíveis quatro modelos de implementação: cloud pública, cloud privada, cloud de comunidade e cloud híbrida [3].

2.4.1. *Cloud* pública

O modelo mais usual e simples considera um servidor público. A *cloud* pública não é exclusiva para um cliente. Em adição, a partilha da infraestrutura é feita entre vários clientes. Se um cliente não está no momento a necessitar de todos os recursos computacionais, estes são alocados para outros clientes [1]. Uma das principais vantagens deste tipo de *cloud* é o facto de não ser necessário investimento de capital inicial em infraestrutura. Além disso, o risco é transferido para a infraestrutura do fornecedor. No entanto, estas *cloud*s podem dificultar a eficácia de alguns negócios, visto que não é assegurado o controlo pleno dos dados, rede e configurações de segurança [5].

Apesar da partilha dos recursos, um cliente não tem acesso aos dados dos outros clientes, assegurando a segurança. A localização é sempre na infraestrutura do fornecedor. A *AWS*, *Google Apps* e a *Blue cloud* são exemplos de fornecedores de *cloud* pública [11].

2.4.2. *Cloud* privada

Embora o conceito de *cloud* tenha como objetivo otimizar os recursos, existem algumas aplicações que não o permitem. Quando se fala de segurança informática é importante saber com quem partilhamos o servidor e onde estão localizados os dados.

A *cloud* privada tem esse objetivo, restringir a infraestrutura a um cliente. O servidor é apenas alocado a esse cliente e não existem alterações nos serviços disponibilizados. Caso o cliente tenha necessidade, pode alterar a localização do servidor [1] [8].

Este tipo de *cloud*s, também designada como *internal cloud*, são exclusivas a uma entidade. A criação e gestão deste tipo de *cloud*s é realizada pela própria entidade ou por um fornecedor externo. A principal vantagem assenta num controlo mais efetivo do desempenho, segurança e confiabilidade. Contudo, esta *cloud* é idêntica ao "*traditional proprietary server farms*", mas como vantagem os custos de capitais iniciais (*up-front capital cost*) não serem necessários [5].

Uma das principais vantagens deste tipo de *cloud* reside no facto de ser passível que os utilizadores possam usufruir de uma infraestrutura flexível e ágil para executar cargas de trabalho de serviço em seus domínios administrativos [9]. A *EMC*, *Eucalyptus* e *OpenStack* são exemplos de ferramentas de configuração de uma *cloud* privada [11].

2.4.1. *Cloud* de comunidade

Como sugere o nome, este modelo engloba vários clientes (comunidade). É uma mistura entre a *cloud* privada e a pública. É semelhante à *cloud* privada, na medida em que a localização da

infraestrutura pode ser definida pelos clientes e semelhante à *cloud* pública por estar disponível para vários clientes [1].

Um exemplo de utilização deste modelo pode ser quando várias organizações com os mesmos objetivos, pretendem utilizar um serviço de *cloud* privada em conjunto [1]. Um caso prático desta situação é um grupo de hospitais em Ontário que colaboram entre si, com o intuito de criar um serviço de *cloud computing* comum a todos os hospitais, intitulado como *3SO* [11].

2.4.2. *Cloud* híbrida

Uma infraestrutura de *cloud* que provém da combinação de dois ou mais modelos de implementação considera-se híbrida. Os dados e aplicações são partilhados entre a *cloud* privada e a pública. Este tipo de *cloud* tenta corrigir as limitações que a *cloud* pública e privada contêm [5] [8].

O processamento de tarefas mais sensíveis é feito na *cloud* privada, mantendo os dados e aplicações sobre o *firewall* e segurança do cliente. As tarefas mais básicas são encaminhadas para a *cloud* pública, evitando sobrecarregar a *cloud* privada e garantindo mais flexibilidade ao serviço [1]. A *cloud* híbrida permite um controlo de segurança mais rigoroso sobre os dados comparando com a *cloud* pública. Além disso, auxilia a expansão e contratação de serviços. A principal desvantagem desta *cloud* reside no facto de compreender quais os componentes que se situam na *cloud* privada e os que deverão estar na *cloud* pública [5]. Como exemplo deste tipo de *cloud* temos a *Intel Hybrid Cloud* e a *Microsoft Azure* [11].

2.5. *Cloud computing* e a Indústria 4.0

A primeira revolução industrial no século XVIII deveu-se à introdução de máquinas a vapor para ajudar na produção. No final do século XIX surgiram as primeiras linhas de montagem. Após quase 100 anos, na década de 1970, a tecnologia evoluiu, dando origem à computação, robótica e automação. Com produtos mais complexos, as necessidades computacionais e de gestão aumentaram [2]. A tecnologia adaptou-se, introduzindo à indústria: *cloud computing*, *IoT*, inteligência artificial e sistemas ciber-físicos [12]. A evolução levou-nos à quarta revolução industrial: **Indústria 4.0,** como demonstra a Figura 3.

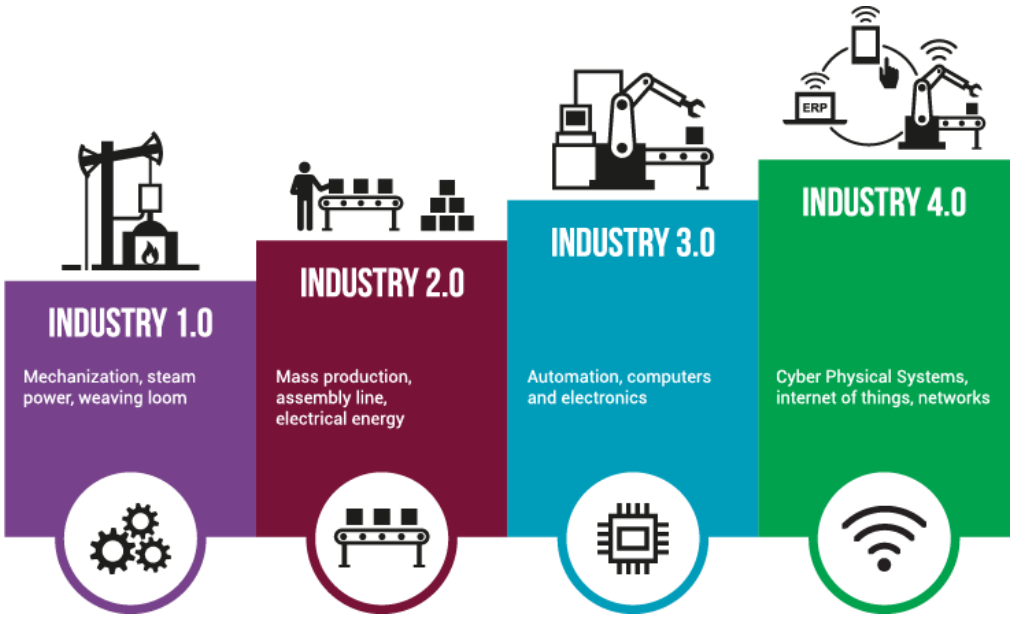

Figura 3 - Evolução da Indústria [13]

A Indústria 4.0 em termos tecnológicos tem vários objetivos, estando não só focados na parte da produção, mas também no desenvolvimento do produto. Ao facilitar a ligação entre o produtor e o cliente, permite acelerar o processo de produção e reduzir custos no desenvolvimento. A evolução da computação potenciou outro objetivo da Indústria 4.0: a interligação das tecnologias operacionais e das tecnologias de informação [2].

Os paradigmas da quarta Revolução Industrial passam por [12]:

- Interoperabilidade – capacidade de integração e cooperação entre máquinas, recorrendo a métodos de inteligência artificial;
- Virtualização – possibilidade de criar modelos ou cópias de uma fábrica inteligente por forma de gerir e prever o seu funcionamento;
- Descentralização – possibilidade de utilizar inteligência descentralizada, podendo assim diminuir a capacidade de processamento local. Neste sentido, será viável o controlo de cada processo de forma otimizada;
- Tempo real – aquisição e análise de dados em tempo real, usando métodos de inteligência artificial, com o intuito do processo ser adaptável em tempo real;
- Orientado ao serviço – cada serviço é controlado de forma dependente dos restantes serviços da organização;
- Modularidade – capacidade da indústria ser flexível mediante diversas situações, conseguindo reconfigurar a sua produção, a fim de atender as necessidades do momento.

2.5.1. Internet das Coisas (*IoT*)

Segundo uma publicação [14] da revista *Forbes*[1], "*Internet Of Things*" representa a interligação de qualquer dispositivo à internet. Estão englobados todos os dispositivos desde eletrodomésticos, máquinas, carros, entre outros. Esta definição também aponta para partes de máquinas como sensores e atuadores, possibilitando que comuniquem não só com o controlador da máquina, mas com outros dispositivos ou centros de informação.

O *IoT* é constituído por cinco camadas. A primeira refere-se à camada física. Esta representa os elementos físicos a serem controlados, como exemplo temos uma lâmpada a iluminar um espaço. A segunda camada é representada ao nível sensorial e de atuação. A mesma é descrita através do exemplo anterior em conjunto com um sensor de luz e um interruptor. Além disso, este tipo de *layer* cria valor à primeira camada, introduzindo algum tipo de controlo. A terceira corresponde à conectividade a um servidor, fornecendo e adquirindo informação. Desta forma, é possível deixar de ter uma simples lâmpada e passar a ter um dispositivo composto com possibilidade de controlo e monitorização remota. A penúltima camada introduz a análise dos dados, interligando com outras fontes. Por fim, isto leva-nos à última camada, criação de valor através da análise dos dados [15].

A consultora *Gartner*[2] em 2016 previu que até 2020 iram estar instalados um total de 20 mil milhões de dispositivos *IoT* pelo mundo. [16]

2.5.2. *Big Data*

O termo *Big Data* descreve uma grande quantidade de dados. Engloba os dados analisados e os dados por analisar. A organização e a filtragem da informação são a sua prioridade, com o intuito de tomar decisões adequadas a partir dessa informação [17] [6].

O *Big Data* pode ser caracterizado por cinco fatores, referenciados como os 5V's: Volume, Variedade, Velocidade, Valor e Veracidade (Figura 4). O *Big Data* representa um grande **volume** de dados provenientes de diversas fontes. O uso de diferentes formatos e estruturas apresentam uma grande **variedade** de informação a processar. Estes dados têm de ser recolhidos e analisados com uma **velocidade** adequada, tendo em conta a **veracidade** dos mesmos, a fim de garantir resultados adequados. Por fim, os resultados criam **valor** para a organização [18] [6].

[1] Forbes, é uma revista estadunidense de negócios e economia (Wikipédia)
[2] Gartner, Inc. é uma empresa de consultoria fundada em 1979 (Wikipédia)

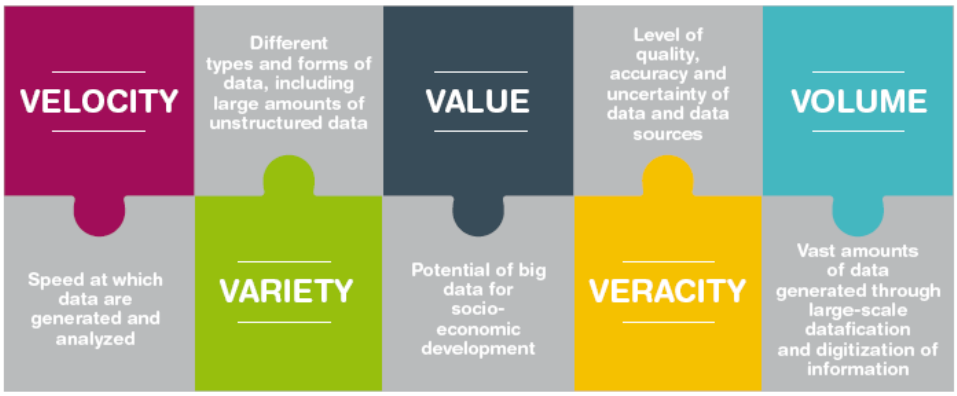

Figura 4 - Características do *Big Data* [19]

Como referido no capítulo Internet das Coisas (*IoT*), são previstos milhões de dispositivos ligados à internet. Considerando que cada dispositivo envia uma pequena quantidade de informação para um servidor por segundo, ao estarem previstos milhões de dispositivos a enviar, a quantidade informação recebida no servidor torna-se colossal.

Um servidor *cloud* pública disponibiliza o ambiente indicado para o *Big Data*, já que garante o armazenamento dinâmico, ajustando-se à quantidade de informação que o cliente necessita naquele momento. O uso da *cloud* pública para o *Big Data* traz problemas, como a segurança (consequência da partilha dos recursos entre os diversos utilizadores) [18].

A integração do *Big Data* na *cloud* gera vários desafios, tais como [20]:

- **Segurança dos dados e confidencialidade** - A segurança dos dados consiste em garantir que os dados estão criptografados, de modo a não serem acedidos por qualquer utilizador. Portanto, este desafio assegura a proteção dos dados;
- **Autenticação –** é necessário um sistema de autenticação com a capacidade de identificar se os dados vêm de fontes fidedignas/genuínas;
- **Integridade –** este desafio averigua se os dados enviados da fonte são os mesmos que foram recebidos no destino**;**
- **Disponibilidade –** garantir que os dados estão disponíveis para o utilizador quando necessário, recorrendo a métodos como a duplicação dos dados por vários servidores**;**
- **Remover os dados duplicados –** utilizar métodos de compressão e otimização para diminuir o espaço alocado pelos dados**.**

2.5.3. Cibersegurança

Como referido anteriormente, uma característica do serviço *cloud* é a computação normalmente estar fora da rede interna do cliente. Todos os dados trocados com a *cloud* passam pela rede pública, sendo expostos a ameaças, tornando o sistema vulnerável.

A cibersegurança é um requisito fundamental para a passagem segura da informação entre o cliente e o servidor. Assenta no uso de *firewall*, antivírus e encriptação, como meios de proteção e prevenção contra o acesso ou violação do sistema.

Na indústria, a segurança dos sistemas informáticos e de automação é critica. Mais do que a fuga de informação, a alteração do sistema pode levar a consequências graves. Em 2015 o *World Economic Forum* publicou: *"Aceder à informação da localização de um carro é simplesmente uma invasão de privacidade, mas aceder ao sistema de controlo do carro é uma ameaça à vida"* [21].

A cibersegurança pode atuar em três camadas. Na primeira camada estão os dispositivos sensoriais e de atuação com ligação à internet (*IoT*). Estes dispositivos permitem adquirir a informação, processar e controlar em tempo real a aplicação a que estão destinados. Para complementar, têm também uma comunicação constante ou intermitente com o servidor. A segunda camada engloba os meios que encaminham a informação, tais como: *routers*, *switch* e *gateway*. Podemos identificar a segunda camada como a "*Fog Layer*", isto é, uma camada que não está sob o controlo do utilizador. Resumindo, os pacotes são direcionados dos dispositivos para o servidor, contudo os pacotes podem tomar diversos caminhos até chegar ao seu destino, sendo difícil de prever eventuais ameaças. Por último, o servidor *cloud* é a terceira camada [22].

Na Figura 5 estão representadas as camadas onde a cibersegurança atua e alguns dos elementos que constituem cada camada.

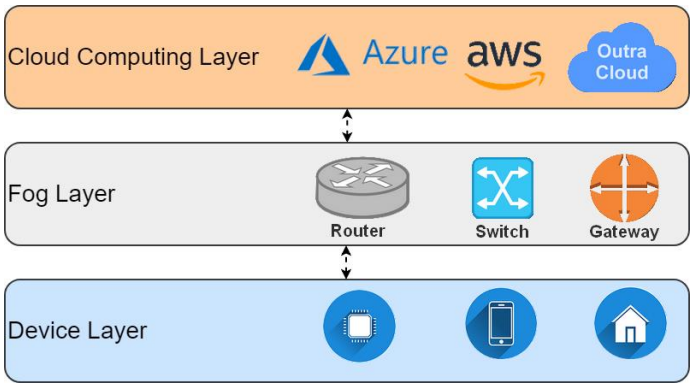

Figura 5 - Camadas de atuação da cibersegurança

O conceito de Indústria 4.0 depende da cibersegurança. Ao permitir que os sistemas comuniquem e sejam controlados à distância, a sua segurança é fulcral para garantir que o sistema não seja comprometido, protegendo o sistema e as pessoas [2].

2.5.4. *Clouds* industriais e fornecedores

Neste capítulo serão apresentados os diversos fornecedores de *cloud* e as variantes dos serviços disponíveis no mercado. O requisito principal para filtrar os diversos fornecedores é ter capacidade

para integrar dispositivos *IoT* ou *PLCs* sem necessidade de desenvolvimento de uma aplicação dentro da *cloud*. Outros aspetos a ter em conta estão relacionados com os serviços que são disponibilizados, tais como: inteligência artificial, protocolos suportados, necessidade de *hardware* extra, capacidade de armazenamento, entre outros [2].

Para fornecedores de *cloud*s genéricas, a escolha baseou-se no estudo feito pela *RightScale*[3] aos fornecedores de *cloud* mais utilizados a nível empresarial. Na Figura 6 estão representados os principais fornecedores de *cloud* e a adesão empresarial aos mesmos. Os dados apresentados são referentes aos anos de 2017 e 2018.

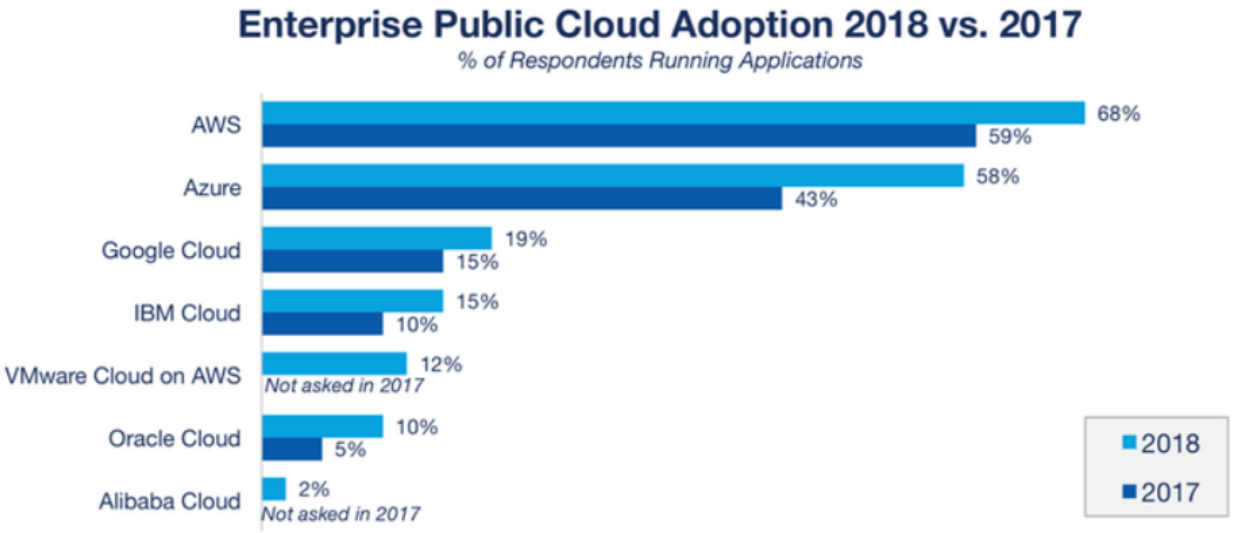

Figura 6 - Adesão empresarial aos vários fornecedores *cloud* [23]

As *cloud*s industriais foram escolhidas analisando os principais fabricantes de *PLCs* e sistemas para automação industrial.

MindSphere

O primeiro fornecedor pesquisado foi a *Siemens*. Este tem serviços focados para a indústria e integração de *PLCs* e *IoT*, todos englobados na sua *cloud MindSphere*. O objetivo desta é interligar dispositivos físicos e unidades de produção, recolhendo a sua informação para posteriormente ser tratada, aumentando o rendimento e eficácia da produção. Esta *cloud* disponibiliza ligação para dispositivos da *Siemens*, mas também está focada para os dispositivos *IoT*. O seu modelo de implementação é híbrido com um serviço do tipo *PaaS*. A *MindSphere* está instalada na *AWS* correspondente ao serviço *cloud* da *Amazon* [24].

[3] A RightScale é uma empresa que vende *SaaS* para gestão de *cloud computing* a vários provedores (Wikipédia)

Para aceder à *cloud*, a *Siemens* dispõe de quatro métodos. Dois dos métodos que recorrem ao *MindConnect* são: o *MindConnect IoT2040* e o *MindConnect Nano*. Estes são dispositivos que permitem servir de ponte entre diversos protocolos de comunicação e a *cloud*. Os outros dois métodos estão relacionados com a programação e permitem a ligação direta de *PLCs* e dispositivos *IoT* à *cloud*. Para a série de *PLCs Siemens S7-1500*, existe um *Function Block* (FB) denominado *MindConnect FB 1500*. Para acesso através de dispositivos *IoT*, a biblioteca *MindConnect LIB* pode ser utilizada. A informação ao passar por qualquer método *MindConnect* é encriptada para aumentar a segurança [25].

O acesso aos serviços da *MindSphere* é feito através do *MindAccess*. Este utiliza diversos planos, dependendo das necessidades da aplicação. Os planos diferem na capacidade de armazenamento, quantidade de elementos ligados e número de utilizadores associados [26].

Para tratar os dados, a *MindSphere* utiliza aplicações, intituladas *MindApps*, que ajudam a organizar, proteger e tomar decisões. Estas aplicações podem ser adquiridas à *Siemens* ou ser desenvolvidas pelo cliente, ajustando-se às suas necessidades. Ao utilizar estas aplicações, o tipo de serviço é *SaaS*, ou seja, o fornecedor disponibiliza o *software* e o cliente utiliza-o e molda-o às suas necessidades. A *MindSphere* tem duas aplicações principais, a *Product Intelligence* e a *Machine Tool Analytics*. As outras aplicações disponíveis são fornecidas por outras empresas [27].

ProfiCloud

Desenvolvida pela *Phoenix Contact*[4], a *ProfiCloud* permite integrar dispositivos *IoT*. O tipo de serviço é semelhante ao apresentado pela *Siemens*, disponibilizando uma plataforma para o cliente desenvolver as suas aplicações. As suas plataformas são orientadas para a automação, sobretudo para os *PLCs Phoenix Contact* [28].

A transmissão de dados é encriptada usando *Tranport Layer Security* (TLS), aumentando a segurança e tem a possibilidade de fazer *backups* automáticos, no caso de perdas de informação. As aplicações *web* na *ProfiCloud* transmitem os dados por *Hyper Text Transfer Protocol Security* (HTTPS) para o utilizador, acrescentando segurança à aplicação [28].

Para ligar à *cloud* são disponibilizados pela empresa vários métodos, dentro dos quais: o *Gateway IoT Cloud, AXC Cloud Pro e o Cloud Coupler Pro.* Estes ajudam o utilizador a gerir a sua rede. Além disso, têm a capacidade de utilizar protocolos seguros para comunicar. O primeiro método é utilizado para ligar dispositivos *IoT*. O segundo consiste num *PLC* com capacidade para comunicar com a *PorfiCloud*. Por último, o *Cloud Coupler Pro* é indicado para ligar redes industriais [28].

[4] A Phoenix Contact é uma empresa alemã que fabrica dispositivos para automação, comunicações industriais, bornes e outros componentes elétricos.

O serviço fornecido é de *PaaS*, mas com possibilidade de ser apenas *SaaS*, caso o cliente apenas use as aplicações fornecidas. A *ProfiCloud* utiliza um modelo de implementação de *cloud* híbrida, apoiando-se em operadores de serviços públicos e privados [28].

Os métodos de pagamento são baseados no modelo *Pay-per-Use*. Estes são feitos através de créditos designados *Cloud-Credits*. Além disso, permitem ao utilizador ter um controlo sobre os custos, não necessitando de contratos [28].

Microsoft IoT Edge* e *IoT Hub

A *Microsoft* tem um serviço de *cloud*, o *Azure*, que abrange imensas aplicações. Como por exemplo, o *IoT Edge* e o *IoT Hub*. Estes dois tipos de serviços são focados para ligação de dispositivos *IoT*, elevando as suas potencialidades. O modelo de implementação é híbrido. O serviço é gratuito para pequenas aplicações e para maiores quantidades de dados, apresenta vários escalões [29].

- *IoT Edge* - Permite introduzir inteligência artificial, fazer análises de dados, gestão de dispositivos, redução de custos de banda larga e simplificação do desenvolvimento. Outra vantagem consiste no *IoT Edge* ter capacidade de operar com ligação intermitente. Quando a ligação ao *IoT Edge é retomada*, existe o processo de sincronização dos dados [29].
- *IoT Hub* - Permite gerir milhões de dispositivos, com uma grande variedade de protocolos e sistemas operativos. Este tem possibilidade de adicionar segurança na ligação e estabelecer uma comunicação bidirecional fiável com um custo reduzido [29].

O benefício da *Azure* assenta na possibilidade de integração destes dois serviços nas suas outras aplicações. Estas variam desde a análise de dados, armazenamento, computação, aplicações *web* e outras ferramentas [29].

Google Cloud Platform

A *Google* gere a *Google Cloud Platform* que apresenta imensas soluções ligadas ao *cloud computing*. O *Cloud IoT Core* está orientando para interligar dispositivos *IoT* com o serviço *cloud*. Os recursos apresentados são direcionados à análise, monitoramento e gestão de dados, com suporte a milhões de dispositivos. Este adiciona segurança na ligação e criptografia dos dados com o recurso a protocolos, como o *Message Queuing Telemetry Transport* (MQTT) e o *Hyper Text Transfer Protocol* (HTTP). A comunicação para os dispositivos é bidirecional, permitindo assim atualizações automáticas do seu *firmware*. A Google disponibiliza bibliotecas para diversas linguagens de programação e com suporte a uma vasta gama de fabricantes de dispositivos *IoT* [30].

Uma vantagem de utilizar esta *cloud* é que a rede *Google* opera de forma privada em vários pontos do globo, ou seja, a segurança dos dados é superior e a continuidade de serviço é superior ao

da rede pública [30]. O tipo de serviço é *PaaS*, disponibilizando bibliotecas e outras ferramentas para o desenvolvimento de aplicações. O modelo de implementação é do tipo *cloud* híbrida [30].

AWS

Amazon Web Services (AWS) é o serviço *cloud* com mais recursos a nível mundial atualmente. Esta *cloud* é gerida pela *Amazon* e estima-se que tem uma capacidade de 1*Exabyte* [1].

Oito serviços da *AWS* estão associados à temática *IoT*:

- *AWS Greengrass* - ajuda a diminuir a quantidade de dados enviados para a nuvem, filtrando os que não são necessários. Também disponibiliza ferramentas de desenvolvimento com funções simplificadas e comunicação segura. Adicionando ainda o sincronismo de dados, recorrendo a funções do *AWS Lambda*[5] [31].
- *AWS IoT Core* - serviço que permite gerir dispositivos e processar milhões de mensagens, utilizando protocolos de comunicação de baixa largura de banda e com ligação intermitente. Permite também que vários dispositivos comuniquem, mesmo com protocolos diferentes, fazendo uma ponte entre os mesmos [31].
- *AWS FreeRTOS* - sistema operacional orientado para microcontroladores com baixa capacidade para usufruírem de comunicações para a *AWS* [31].
- *AWS IoT Device Management* – ajuda a gerir uma grande quantidade de dispositivos *IoT* ligados à *AWS* de forma segura [31].
- *AWS IoT Device Defender* – garante que as políticas de segurança associadas aos dispositivos estão a ser cumpridas, permitindo boas práticas de segurança. Além disso, tem a capacidade de detetar comportamentos anormais [31].
- *AWS IoT Analytics* – orientado para o *BigData*, este serviço analisa grandes volumes de dados, provenientes de dispositivos *IoT*. Neste sentido, estes dados irão ajustar a tomar decisões mais adequadas [31].
- *AWS IoT 1-Click* – permite executar funções do *AWS Lambda* em dispositivos que sejam compatíveis. Estes podem ser ações, como notificações de equipamentos sobre manutenção, alertas sobre fornecimento de mercadorias ou solicitações [31].
- *AWS IoT Button* – botão programável que permite introduzir os utilizadores às aplicações da *AWS* de forma facilitada, sem necessidade de conhecimentos de programação. Este botão pode ser utilizado para controlar, por exemplo, o ar condicionado [31].

[5] AWS Lambda é uma plataforma que gere os recursos computacionais que são alocados ao cliente. (Wikipédia)

Comparação

Para comparar as soluções disponíveis no mercado, vão ser levantadas várias potencialidades e serão apresentadas sobre a forma de tabela, ajudando a análise do conteúdo. Na Tabela 1 estão representados por colunas os vários fornecedores de *cloud* pesquisados e por linhas as capacidades relacionadas com a indústria. Estas capacidades podem ser inerentes ou disponíveis para adicionar ao serviço. Quando a capacidade não se aplica ao fornecedor, é apresentado um hífen.

Tabela 1 - Comparação dos diversos fornecedores *cloud*

	MindSphere	Profi*Cloud*	Azure *IoT* Hub	Google *Cloud*	AWS
Tipo de serviço	*PaaS*; *SaaS*	*PaaS*; *SaaS*	*PaaS*; *SaaS*; *IaaS*	*PaaS*; *SaaS*; *IaaS*	*IaaS*; *PaaS*; *SaaS*
Modelo de Implementação	Híbrido	Híbrido	Híbrido	Híbrido	Híbrido
Específico para a indústria	Sim	Sim	Genérico	Genérico	Genérico
Protocolos Comunicação	-	HTTPS	MQTT; AMQP; HTTP	MQTT; HTTP	MQTT; HTTP
Dispositivos dedicados para comunicação	MindConnect	Gateway *IoT Cloud*; AXC *Cloud* Pro; *Cloud* Coupler Pro	-	-	AWS *IoT* Button
Flexibilidade no serviço	Baixa	Baixa	Média	Média	Elevada
Escalabilidade	Baixa	Baixa	Elevada	Elevada	Elevada
Serviço Gratuito	Não	Não	*IoT* Hub até 8000 mensagens/dia	Durante um mês	Durante um mês

A comparação dos cinco fornecedores *cloud* que foram analisados, levantou as vantagens de cada serviço. As *cloud*s *MindSphere* e *Proficloud* mostram uma melhor integração nos controladores e redes industriais. Este facto deve-se a serem orientadas especificamente para a indústria e dos dispositivos dedicados facilitarem a integração. Ambas possuem dispositivos para servirem de *gateway* ou bibliotecas para integração de dispositivos *IoT*. A sua flexibilidade é baixa, caso o cliente necessite de um serviço que aumente consoante as necessidades, as opções são poucas. Estas *cloud*s têm uma quantidade de escalões que impede a flexibilidade do serviço e apresentam pouca escalabilidade, em que os serviços suportam uma quantidade limitada de dispositivos por conta, comparando com os outros fornecedores.

Analisando as *cloud*s genéricas (não especificas para a indústria), o foco destas é diferente da *MindSphere* e da *Proficloud*. Os serviços fornecidos são apenas tratados como *IoT*. Estas *cloud*s

apresentam uma maior variedade de aplicações e serviços, por exemplo: inteligência artificial, bases de dados, análise de dados, páginas *web*, segurança, aplicações móveis, entre outras. As capacidades de escalabilidade e de flexibilidade dos serviços é superior às *cloud*s industriais analisadas. Para conhecer os serviços existentes, são disponibilizados pelos fornecedores créditos ou 1 mês de experimentação, consoante a *cloud*.

Todos os fornecedores permitem um modelo de implementação híbrido, ou seja, a *cloud* pode ser pública, privada ou a junção dos dois. Os tipos de serviços suportados por todos os fornecedores são *SaaS* e *PaaS*, nas *cloud*s genéricas podemos encontrar também o *IaaS* (máquinas virtuais).

3. Caso de estudo

Este capítulo pretende demostrar o funcionamento da comunicação entre um *PLC Siemens* e o serviço *cloud* da *Microsoft Azure*: *IoTHub*.

3.1. Introdução

Atualmente na indústria, grande parte das máquinas industriais com controlo por *PLC* e que foram produzidas nos últimos anos têm uma ou mais portas *Ethernet*. Essas portas são dedicadas a comunicações com outros dispositivos utilizando redes industriais como: *Profinet*, *EtherCAT* ou *Modbus TCP*.

A programação do *PLC* é limitada ou inexistente para aplicações Web, focando-se mais na performance e fiabilidade do controlo a que está destinado. Além disso, restringem na programação o uso de protocolos que não sejam indicados para o uso industrial, fornecendo funções básicas e pouco apelativas. Os PLCs mais recentes e de alto desempenho já permitem o uso de protocolos de comunicação diferentes dos utilizados nas redes industriais, como exemplo: a gama *S7-1500* da *Siemens* atualmente, permite o uso do protocolo MQTT, muito utilizado em dispositivos *IoT*.

As vantagens das redes industriais é que são preparadas para situações de ambiente industrial, com intuito de funcionarem em condições adversas de temperatura, perturbações eletromagnéticas, vibração e humidade. Os fabricantes de *PLCs* têm preferências por estas redes, garantindo que mesmo sobre condições adversas, os seus produtos funcionam de forma robusta.

3.2. Arquitetura comunicação

No caso em estudo está previsto o uso de um *PLC Siemens S7-1200* ou *S7-1500*. As razões remontam por estes serem muito utilizados na indústria. Além do mais, possuem porta *Ethernet* e recursos para efetuar a comunicação pretendida. O serviço *cloud* escolhido é o *IoTHub* da *Microsoft Azure*. Deve-se a ser um serviço gratuito e ter os recursos necessários.

Confrontando as necessidades e contornando a dificuldade em utilizar o próprio *PLC* para comunicar diretamente com o servidor *cloud*, devido a só alguns modelos o permitirem, optou-se pelo desenvolvimento de um dispositivo de interligação entre o *PLC* e a *Cloud*. Na Figura 7 é observável o diagrama de blocos que representa a interligação do *PLC* com a *cloud* da *Azure*, passando pelo dispositivo desenvolvido. Este dispositivo atua como uma ponte entre as duas interfaces.

Figura 7 - Estrutura da comunicação

Focando na comunicação entre o dispositivo e o *PLC*, existem diversos protocolos que permitem o acesso aos dados de um *PLC*. Para este projeto pretende-se recolher informação de um *PLC Siemens* por *Ethernet* e, se possível, alterar o mínimo na programação do mesmo.

Analisando as possibilidades de comunicação destes *PLCs*, verificam-se dois protocolos de comunicação que possam ter uso na aplicação pretendida. Atendendo aos requisitos mencionados, na Tabela 2 estão representados esses protocolos com uma breve análise dos mesmos e por fim algumas características com relevância para o caso de estudo.

Tabela 2 - Protocolos disponíveis para PLCs Siemens

Protocolo	**S7 Communication**[6]	**Modbus TCP**[7]
Descrição	Utilizado para comunicações entre *PLCs Siemens*	Utilizado para comunicação em diversos *PLCs* e outros dispositivos
Exclusividade	Exclusivo para *PLCs Siemens*	Não tem
Programação no *PLC*	Reduzida	Elevada
Acesso aos dados no *PLC*	Completo e facilitado	Apenas aos disponibilizados no programa

Baseado na informação da Tabela 1 optou-se pelo uso do protocolo *S7 Communication*, devido ao objetivo ser apenas a ligação a um *PLC Siemens* e a programação para adquirir/escrever os dados não necessitar de estar interligada com a do *PLC*. Uma desvantagem deste protocolo é a segurança. Como consegue o acesso e escrita de qualquer informação nos registos do *PLC*, deve ser usado com cuidado e limitado a certos registos.

Com a escolha do protocolo para comunicar com o *PLC*, prosseguiu-se para o protocolo de comunicação com o *IoTHub*. Com uma pesquisa nos documentos da *Microsoft Azure*, um dos mesmos refere-se à escolha do protocolo [32]. A Tabela 3 contém um resumo do documento pesquisado, com a informação necessária para perceber as diferenças entre os protocolos disponibilizados para a comunicação com o *IoTHub*.

[6] S7 Communication – Protocolo TCP/IP desenvolvido pela *Siemens* para *PLCs Siemens* (Snap7)
[7] Modbus TCP – Protocolo TCP/IP baseado no Modbus, utilizado em automação (RTA automation)

Tabela 3 - Protocolos disponíveis para *IoT*Hub

Protocolo	MQTT	AMQP	HTTP
Quando utilizar	O dispositivo apenas comunica com outro dispositivo	Tirar proveito do multiplexing entre dispositivos	Uso genérico
Vantagens	Eficiente, biblioteca leve, pacote mais compacto	Eficiente, necessita de pouco código, pacote mais compacto, capacidade de multiplexing	Biblioteca leve, porta de comunicação utilizada (443)
Desvantagens	Restrição da rede à porta 8883	Restrição da rede à porta 5671, biblioteca pesada	Não eficiente, pacote extenso, comunicação lenta

O âmbito do projeto neste tópico é que a comunicação com o *IoTHub* seja do tipo "*device-to-cloud*", enviando e recebendo pacotes com regularidade e de preferência que sejam o mais compactos possível para evitar sobrecarga no servidor. Colocadas as condições, a escolha foi o MQTT pelas seguintes razões:

- a comunicação é exclusivamente "*device-to-cloud*";
- leve do ponto de vista do espaço que as bibliotecas ocupam e do código implementado;
- a comunicação com o *IoTHub* é eficiente;
- a restrição da porta de comunicação ser 8883 pode ser contornada utilizando *WebSockets*.

3.3. Dispositivos testados

A fim de garantir uma boa transição de dados entre o *PLC* e o *IoTHub*, é necessário que o dispositivo que faça a ponte entre as interfaces cumpra vários requisitos como: robustez; capacidade de interface gráfica; ligação a duas ou mais redes e custo baixo.

Com uma breve pesquisa sobre placas de prototipagem, chegou-se a vários microcontroladores e computadores com os requisitos pretendidos. Nos seguintes subcapítulos serão apresentadas as várias placas. No último subcapítulo será apresentado um resumo com a escolha da placa para o projeto, expondo as suas características, vantagens e desvantagens em relação às outras placas.

3.3.1. ESP8266

O *ESP8266* é um dos microcontroladores mais conhecidas na temática do *IoT*. O seu custo é inferior a 3€, tem um módulo *Wi-Fi* integrado, um *Central Processing Unit* (CPU) de 32bit a 80MHz, vários *General Purpose Input/Output* (GPIO) e pinos de comunicação. A programação do *ESP8266* pode ser feita através do *Arduino IDE* utilizando as bibliotecas do *Arduino* ou bibliotecas adaptadas [33]. Na Figura 8 encontra-se a placa *Wemos D1 mini*. Esta tem um chip *ESP8266* (placa preta com

corpo metálico) que dispõe de uma porta USB para a programação e de pinos de ligação rápida para aceder às entradas e saídas.

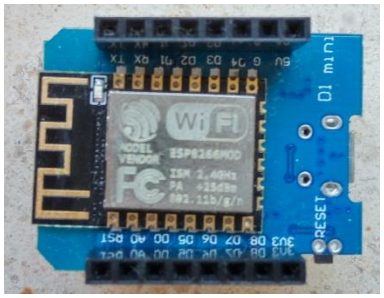

Figura 8 - Placa ESP8266

No teste feito, utilizou-se uma biblioteca disponibilizada em exemplos para o *IoTHub* [34] no *Arduino IDE*. O objetivo é verificar se os pacotes enviados pelo *ESP8266* chegam ao *IoTHub* e o tempo desde o envio até à receção. Para isso, foi gerada uma rotina simples que chama a função que envia a mensagem, inicia um temporizador e quando o pacote chegar ao *IoTHub*, é verificado o temporizador na função.

O ciclo enviou uma mensagem a cada 90 segundos, 60 segundos para verificar se a mensagem foi enviada e 30 segundos para registar os resultados numa folha de cálculo. A mensagem enviada leva o número associado à mesma, a fim de aferir qual a mensagem recebida na chegada ao destino. Após 100 amostras, obteve-se o gráfico da Figura 9.

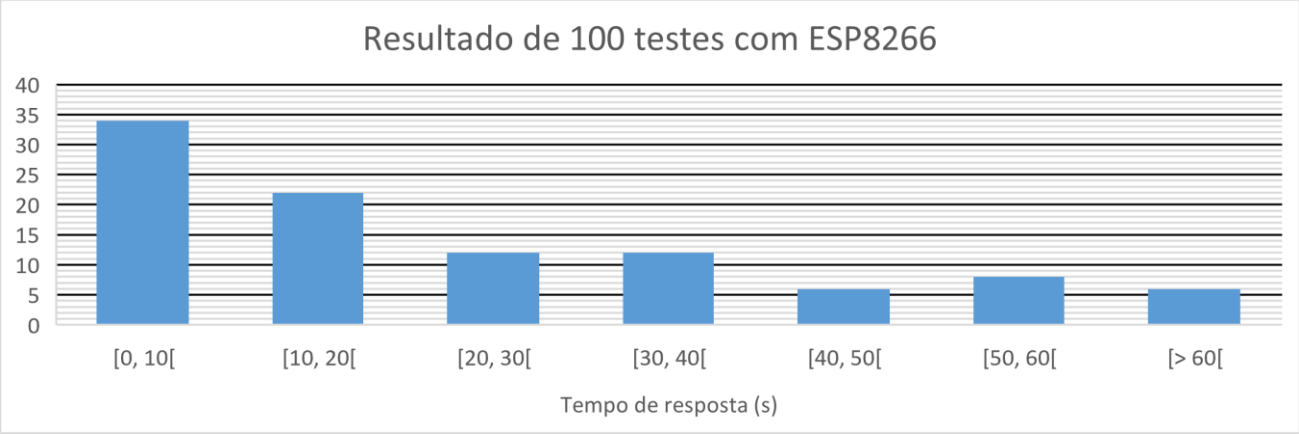

Figura 9 - Resultados de 100 testes com ESP8266

Para facilitar a análise, utilizou-se um histograma em divisões de 10 segundos. Analisando os resultados, verificou-se que 88% dos pacotes chegaram ao *IoTHub* em menos de sessenta segundos e os restantes 12% foram ignorados, saindo fora do limite de tempo (marcado a vermelho). Cerca de um terço dos pacotes chegaram entre 0 e 10 segundos. Um terço dos pacotes é que chegaram com mais de 30 segundos ao seu destino.

Concluindo a análise, observa-se que para aplicações que necessitem de uma taxa de amostragem e atualização inferior a 10 segundos, o *ESP8266* em conjunto com o *IoTHub* torna-se inviável. No entanto, para aplicações em que as mensagens demorem até 1 minuto a chegar ao destino, o *ESP8266* torna-se uma boa escolha devido ao seu baixo custo.

3.3.2. ESP32

O *ESP32* é uma evolução do *ESP8266*, sendo um microcontrolador com um CPU *dual-core* de 32 bits até 240MHz. Ao ser *dual-core*, melhora a estabilidade da rede *Wi-Fi*, alocando os processos de rede num dos núcleos e outras tarefas no outro núcleo. Contém mais pinos GPIO e portas de comunicação que o *ESP8266*. O custo é inferior a 5€, fazendo com que seja uma boa opção para aplicações *IoT* [35]. Encontra-se representado na Figura 10 a placa *Node MCU* com um processador ESP32. Esta placa permite um acesso facilitado às entradas e saídas e dispõe de uma porta *Universal Serial Bus* (USB) para programação.

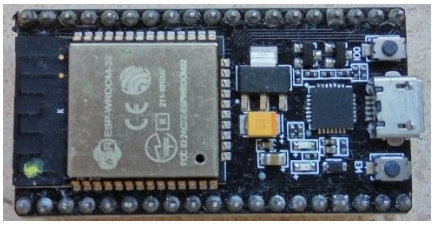

Figura 10 - Placa ESP32

O teste para determinar a performance do *ESP32* é igual teste utilizado para o *ESP8266*, a fim de garantir que as condições sejam o mais semelhantes possível. A biblioteca para comunicar com o *IoTHub* foi adaptada, devido ao processador do *ESP32* ser diferente.

Na Figura 11 está representado o resultado do teste. Tal como no teste anterior, foram feitas 100 amostras, com uma taxa de 1 amostra por 90 segundos.

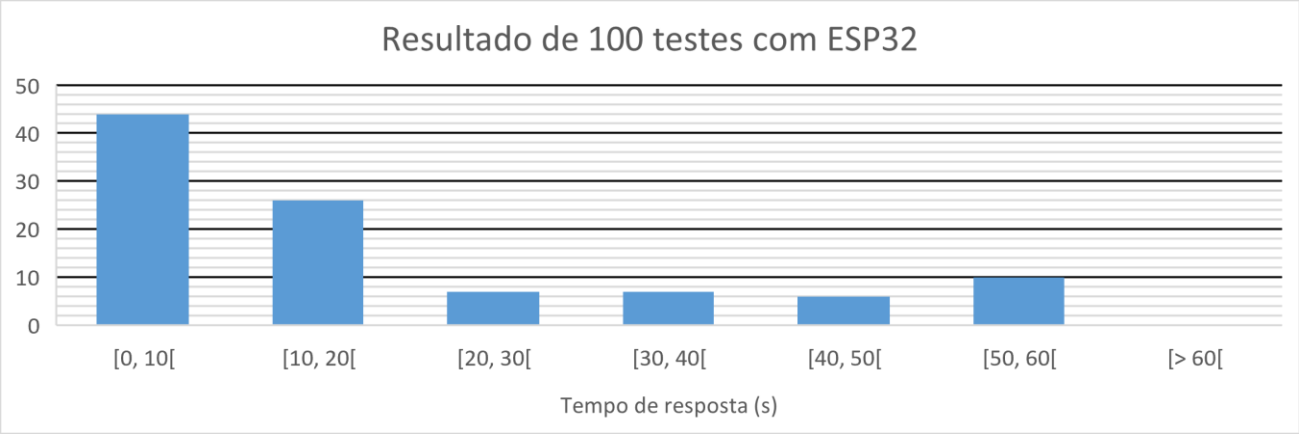

Figura 11 - Resultados de 100 testes com ESP32

Analisando o gráfico é percetível que mais de dois terços das amostras foram inferiores a 20 segundos e não houve nenhuma amostra a exceder os 60 segundos. Comparando com o teste do *ESP8266*, as amostras entre 20 e 60 segundos foram reduzidas, havendo uma melhoria no tempo de resposta em geral.

3.3.3. Raspberry Pi 3

O *Raspberry Pi* é um computador em que todo o *hardware* está numa placa com dimensão semelhante a um cartão multibanco. O CPU presente na placa tem arquitetura *Acorn RISC Machine* (ARM) a 1.2GHz e 1GB de *Random Access Memory* (RAM). Para comunicação e periféricos, possui 4 portas USB e uma *Ethernet*, tem um módulo *Wi-Fi* e outro *Bluetooth* incorporado. Permite a ligação de a um *display* com uma porta especifica ou por *High-Definition Multimédia Interface* (HDMI) a um monitor. Ainda tem alguns pinos GPIO e comunicações como: *Inter-Integrated Circuit* (I2C) e *Serial Peripheral Interface* (SPI) [36]. A placa *Raspberry Pi 3* encontra-se na Figura 12.

Figura 12 - Placa Raspberry Pi 3

Para efetuar o teste de comunicação com o *IoTHub,* foi necessária a instalação do sistema operativo. As escolhas podem variar, mas foram focadas duas:

- *Raspbian* – É uma versão do *Linux* para o *Raspberry Pi* que apresenta uma interface otimizada para o mesmo;
- *Windows 10 IoT Core* – Permite desenvolver aplicações como para o *Windows 10*, tendo a diferença para o *Windows* no aspeto, não tem um ambiente de trabalho, apenas uma aplicação inicial de interface instalada.

A escolha do sistema operativo foi o *Windows 10 IoT Core*. Como a *Azure* é do mesmo fabricante do sistema operativo, faz com que existam bibliotecas e exemplos funcionais. Acresce também a facilidade de programação, devido às aplicações serem iguais ao *Windows 10,* ou seja, para efeitos de teste a aplicação pode correr noutro computador com *Windows*. Neste sentido, não será necessário fazer depuração no *Raspberry Pi*.

Foi utilizado um exemplo fornecido pela *Microsoft*, denominado "*HelloCloud*" [37], para servir de base para o teste da comunicação com o *IoTHub*. Para tornar o teste semelhante aos anteriores, o código foi editado, correndo num ciclo que envia 100 mensagens, com uma cadencia de 90 segundos por mensagem.

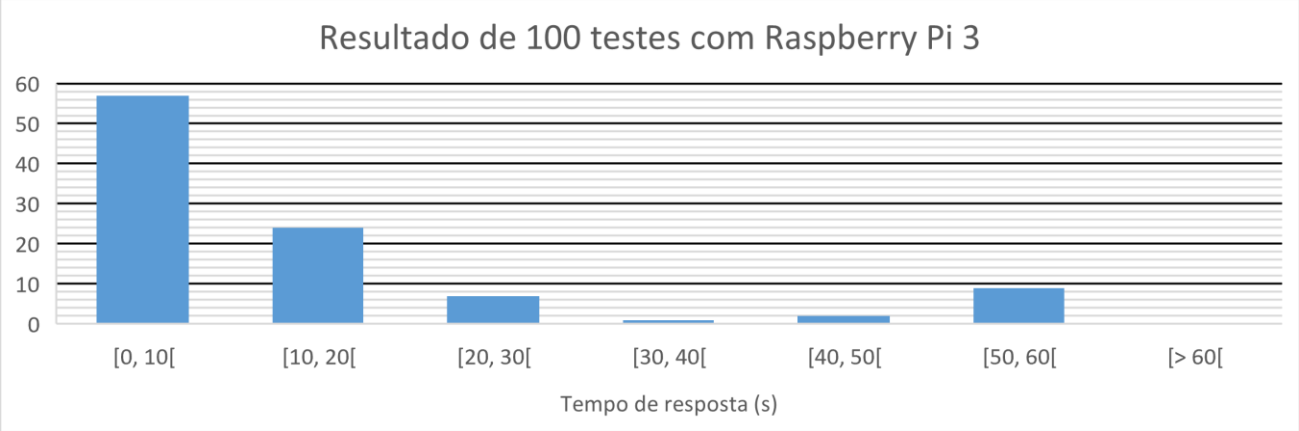

Figura 13 - Resultado de 100 testes com Raspberry Pi 3

O teste com o *Raspberry Pi 3* (Figura 13) mostrou melhorias significativas em relação aos testes anteriores. Quase 60% das mensagens chegaram em menos de 10 segundos, o resto ficaram entre os 10 e os 60 segundos, não havendo violações do limite imposto.

3.3.4. Comparação

Para comparar os dispositivos testados foram utilizadas várias capacidades pretendidas para o projeto. No caso específico da comunicação com o *IoTHub*, foi utilizado o tempo médio da comunicação entre o dispositivo e a *cloud* (quanto mais baixo melhor). As outras capacidades basearam-se na análise das especificações técnicas de cada placa, nos exemplos de aplicações encontrados no *GitHub* e no teste feito com a comunicação com o *IoTHub*. Os exemplos utilizados tiveram como objetivo visualizar as capacidades presentes na Tabela 4. Outras capacidades foram testadas, mas não se mostraram relevantes para o projeto em que vão ser inseridos.

Tabela 4 - Comparação de resultados

Dispositivo	ESP8266	ESP32	Raspberry Pi 3
Média tempo de comunicação	22.12 segundos	17.9 segundos	13.786 segundos
Processamento gráfico	Pouco	Pouco	Elevado
Tipos de comunicação na placa	*Wi-Fi*	*Wi-Fi, Bluetooth*	*Wi-Fi, Ethernet, Bluetooth*
Programação e capacidade	Programação fácil, mas limitada	Programação mais complexa e limitada	Programação complexa e elevada potencialidade
Estabilidade da comunicação	Baixa	Moderada	Elevada

Analisando as possibilidades presentes na Tabela 4, concluiu-se que o *Raspberry Pi 3* é o dispositivo com mais utilidade e capacidade de executar a tarefa que se pretende. Tem porta *Ethernet*, o que permite a ligação direta a um *PLC*, sem necessitar de um módulo externo como os *ESP*. A estabilidade da ligação *Wi-Fi* no *Raspberry Pi* é superior, retomando a ligação rapidamente sem necessidade de reiniciar o dispositivo. No caso do *ESP8266* e do *ESP32* é necessário reiniciar.

A programação do *Raspberry Pi* mostrou-se mais complexa, mas também mais completa, adicionando a possibilidade de interface gráfica igual às aplicações para o *Windows*. O *IoT Core* suporta as seguintes linguagens de programação: *C#*, *C++*, *JavaScript* e *Visual Basic*. Para o projeto, foi escolhida a linguagem *C#* por ser uma das linguagens mais utilizadas para programação em *Windows* e estar ligada com o ambiente gráfico.

Os *ESP* podem comunicar com um *Liquid Crystal Display* (LCD) de baixa resolução por I2C ou SPI, utilizando o código do programa principal para gerir a atualização e animação do LCD. No *Raspberry Pi* o cenário muda, este permite a ligação por HDMI a qualquer monitor compatível, com uma resolução até 4k. Ao utilizar um sistema operativo, a parte gráfica é gerida de forma independente da aplicação a ser desenvolvida. A interface gráfica é gerida pela placa gráfica embutida na placa.

4. Desenvolvimento

Neste capítulo serão apresentados os diversos passos do desenvolvimento da aplicação, assim como a sua estrutura e configuração. Pretende-se que o programa tenha um ciclo principal para a aquisição de dados e várias funções em plano de fundo para a interface gráfica e outras ações menos importantes.

As configurações do servidor *cloud* e do serviço *IoT Hub* também serão apresentados neste capítulo, a fim de garantir uma configuração correta e adequada ao projeto. O ciclo principal do programa (Figura 14) tem como objetivo ler os dados provenientes do *PLC*. Em seguida, mostrar os dados lidos para o ecrã. Por fim, envia os dados para o *IoTHub*. A leitura dos dados deve ser feita através de uma interrupção cíclica com o tempo do ciclo controlado pelo utilizador, permitindo atender às necessidades a que está inserido. Cada vez que a interrupção cíclica é feita, o programa verifica se o *PLC* está ligado e se existem dados para ler. Caso essas condições se verifiquem, o próximo passo é ler os dados do *PLC*. Com os dados lidos, a informação é atualizada e apresentada para o utilizador no ecrã. Caso a ligação com o *IoTHub* esteja configurada, os dados são enviados para o servidor da *Azure* e o ciclo volta ao início.

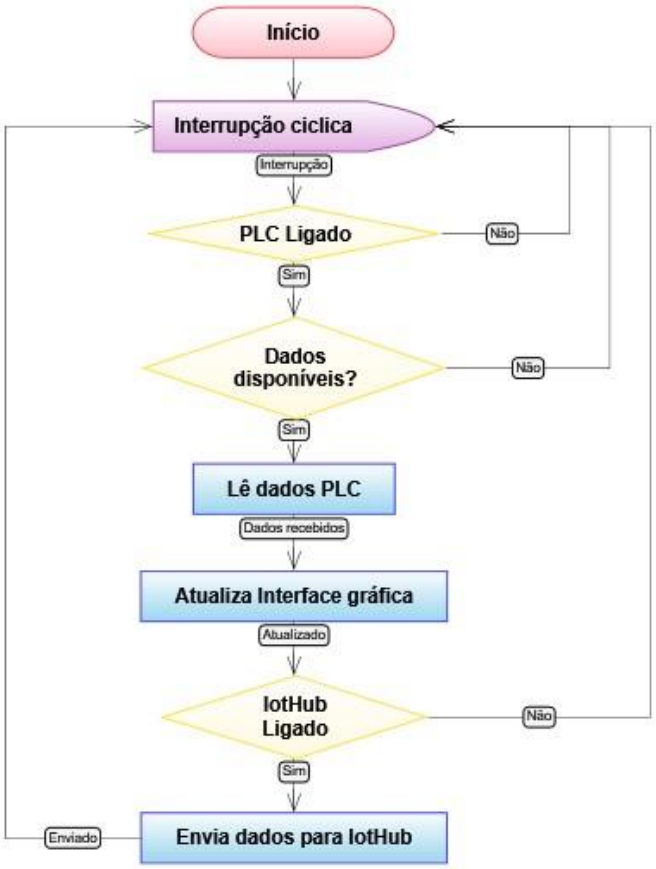

Figura 14 - Ciclo Principal

Fora do ciclo principal, a aplicação necessita de menus para configuração de parâmetros, visualização dos dados lidos e recebidos, entre outros.

4.1. Configuração da *cloud*

Como referido, o fornecedor de *cloud* escolhido para a aplicação é a *Microsoft Azure*. As suas características são apelativas para a aplicação pretendida. Disponibiliza o *IoTHub*, que é um portal que recebe mensagens de dispositivos *IoT*, com um serviço gratuito até 8000 mensagens diárias. O serviço gratuito é suficiente para os testes.

O serviço *IoTHub* da *Azure* é uma *cloud* do tipo *PaaS*. O modelo de implementação está disponível como *cloud* híbrida, embora para o caso em questão irá ser utilizada como uma *cloud* pública. A configuração é feita no site portal.azure.com, em que o primeiro passo é ter uma conta de email compatível com os serviços da *Microsoft*. Para o projeto, utilizou-se a conta de estudante com o domínio *my.ipleiria.pt* por fazer parte de uma conta *Microsoft*.

Com a conta da *Microsoft Azure*, é apresentada a página principal da *Azure*. Esta tem um *dashboard* com um menu lateral. Este menu apresenta os recursos mais utilizados e os serviços mais comuns. Para criar um novo *IoTHub* é necessário clicar no menu lateral em "*New*", aparecendo um novo menu com várias categorias. Dentro da categoria "*Internet Of Things*" está disponível uma secção com o *Azure IoT Hub*. Clicando em cima da secção aparece o menu de configuração.

As configurações do *IoT Hub* necessitam dos seguintes parâmetros:

- Nome – O nome do serviço *IoT Hub*;
- Contrato – Como referido, o *IoT Hub* possuí um serviço gratuito (F1 - *Free*) de 8000 mensagens/dia. Para quantidades maiores, este contrato pode ser alterado;
- Unidades – Quantidade de unidades do tipo *IoT Hub*;
- Partições – Cada comunicação dispositivo-*cloud* pode ter várias partições;
- Grupo de recursos – Grupo de recursos a que o serviço fica associado;
- Subscrição – A subscrição a que o serviço irá ser cobrado;
- Localização – Localização do servidor que irá albergar o serviço.

O único parâmetro que necessita de intervenção externa é a subscrição. Ao clicar no campo da subscrição, como não existe nenhuma, irá ser pedido que se crie uma nova. Para a subscrição é necessário que seja atribuído um cartão de crédito para que o valor mensal da subscrição seja descontado. Isto também é necessário para o serviço gratuito, embora não seja cobrado nenhum valor.

Com o preenchimento de todos os campos e a validação, é necessário aguardar alguns minutos para o serviço estar pronto. Quando finalizada a configuração, é mostrada uma notificação a informar

e o serviço aparece disponível no menu lateral. O serviço também pode ser adicionado no *dashboard*, clicando com o botão direito em cima do espaço livre e, de seguida, em editar. Desta forma, o serviço é adicionado, arrastando o ícone correspondente do menu lateral.

Clicando no serviço, uma página nova é mostrada com as opções do *IoT Hub*. O próximo passo é criar um novo dispositivo clicando em "*IoT devices*" no menu lateral e de seguida em "*Add"*. Tal como na configuração do *IoT Hub*, aparece um menu lateral com as informações do dispositivo. O único parâmetro a preencher é o "*DeviceID"*. Ao aplicar a configuração, aparece um novo dispositivo como na Figura 15. Ao clicar em cima do dispositivo são mostrados os dados de ligação específicos para aquele dispositivo. As configurações do *IoT Hub* estão concluídas e estão disponíveis todos os recursos necessários para iniciar a comunicação.

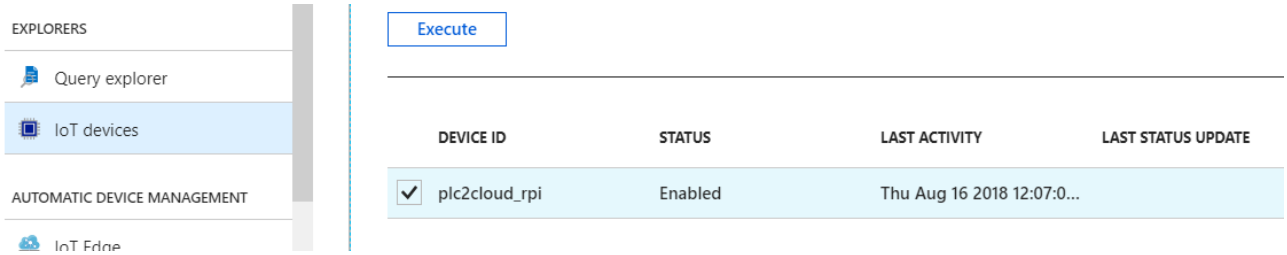

Figura 15 - Novo dispositivo no *IoT Hub*

4.2. Comunicação com *IoTHub*

Focando na programação da comunicação com o *IoTHub*, como referido no capítulo 3.3.3, foram utilizados os exemplos fornecidos pela *Microsoft*. O objetivo é garantir que a comunicação seja efetuada utilizando as bibliotecas adequadas e os métodos corretos.

A biblioteca disponível para comunicar com o *IoTHub* apresenta uma *class* que é utilizada para a comunicação com o *IoTHub*. Esta é denominada de *Microsoft.Azure.Devices.Client*. Esta *class* tem várias classes para gerir e configurar a comunicação. Para o que se pretende, é necessária a *class DeviceClient*.

A class *DeviceClient* foi inicializada como pública e estática, ficando acessível para o resto do programa caso seja necessário. O próximo passo, após a inicialização, é criar e configurar a ligação com o *IoTHub* recorrendo ao método *Create*. O código fica com o seguinte aspeto:

```
1. public static DeviceClient deviceClient;
2. deviceClient = DeviceClient.Create(azureCON.hostname, AuthenticationMethodFactory.CreateAuthentica
   tionWithRegistrySymmetricKey(azureCON.deviceID, azureCON.devicePass), TransportType.Mqtt);
```

O método *Create* tem vários parâmetros de entrada para configurar a ligação. Alguns têm de recorrer a outras bibliotecas do *Windows*, como por exemplo a autenticação. Os parâmetros de entrada são os seguintes:

- *Hostname* do *IoTHub*;
- Autenticação com o ID do dispositivo e a chave correspondente;
- O tipo de protocolo a utilizar.

O ID e a chave são disponibilizados no *IoTHub* quando se cria um novo dispositivo e se associa ao respetivo *Hostname*. No portal da *Azure*, esses dados são encontrados na página da mesma, acedendo a "*IoT Devices*" no menu lateral do *IoTHub*. Por último, é necessário selecionar o dispositivo. Como consequência, irá aparecer algo semelhante à Figura 16. É de salientar que existem duas chaves de ligação. Por predefinição, deve-se utilizar a chave primária. O *Hostname* é encontrado na "*Connection String*", neste caso é "*plc2cloud.azure-devices.net"*, ou acedendo à página principal do *IoT Hub* na descrição do serviço.

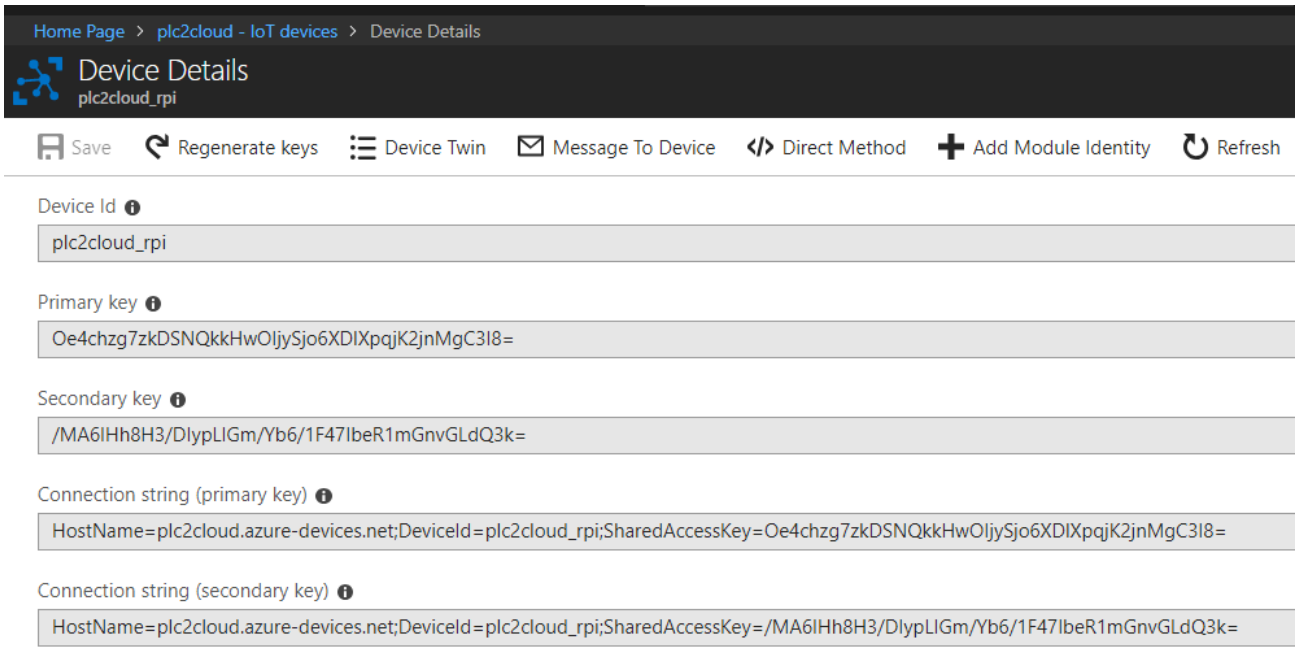

Figura 16 - Detalhes do dispositivo configurado no *IoTHub*

A *class* de comunicação com a *Azure* utilizada não permite um diagnóstico sobre se a mensagem foi enviada com sucesso. Para contornar essa falha, optou-se por fazer uma verificação de erros na função. Resumindo, se a função não for executada, o programa entra em erro. Esse erro pode servir como diagnóstico se foi executado com sucesso.

Para captar esse erro, a função *try/catch* é adequada. O funcionamento é simples: o código a testar é colocado dentro do *try* e a exceção criada pelo erro é colocada no *catch*. No programa ficou da seguinte forma:

```
1.  try
2.  {
3.      await deviceClient.SendEventAsync(message); // Tenta que a mensagem seja enviada
4.  }
```

```
5.  catch
6.  {
7.      // Captura do erro
8.  }
```

A função *SendEventAsync* permite enviar uma mensagem para a *Azure*. Esta opera de forma assíncrona, necessitando do operador *await*. A mensagem enviada deve estar organizada, utilizando um formato de troca organizada de informação. No exemplo foi utilizada o *JavaScript Object Notation* (JSON)[8] por ser um sistema de troca de dados mais compacto e suportado por muitas aplicações e linguagens de programação.

Os dados foram colocados no formato JSON, recorrendo à biblioteca *Newtonsoft.Json*. Tem como entrada, os dados a serem organizados e devolve-os no formato JSON. Após estarem organizados, para garantir que são transmitidos com a codificação certa, os dados são transformados por uma função do sistema que os coloca na codificação *American Standard Code for Information Interchange* (ASCII).

```
1.  var Data_to_Send = new // Dados a serem formatados
2.  {
3.      time = DateTime.Now.ToString(),        // Valor do relógio do sistema
4.      name = L_DataRead.ElementAt(i).Name,   // Nome da variável a enviar
5.      value = L_DataRead.ElementAt(i).Value  // Valor da variável a enviar
6.  };
7.  var messageString = JsonConvert.SerializeObject(Data_to_Send);  // Formatação em JSON
8.  var message = new Message(System.Text.Encoding.ASCII.GetBytes(messageString));// Codificação ASCII
```

A variável "*message*" apresentada no excerto de código anterior, tem a codificação e formatação certa para a função *SendEventAsync*, garantindo uniformização. Assim, a mensagem enviada consegue ser interpretada no seu destino.

4.3. Comunicação com *PLC*

Após uma pesquisa breve de bibliotecas sobre comunicações com *PLC Siemens* por S7 em *C#*, a que sobressaiu foi a *Sharp7* [38]. Esta biblioteca é baseada na *Snap7*, uma das bibliotecas mais conhecidas para comunicações com *PLC Siemens* por S7.

O *Sharp7* é constituído por funções que permitem ler e escrever em memórias de *PLCs* que suportem S7. Como referido no capítulo 3.2, o acesso ao *PLC* através do protocolo S7 não tem restrições se o *PLC* não tiver proteção. É possível aceder a entradas e saídas diretamente, ler/escrever memórias e *DataBlock* (DB)s não otimizados, ler as características do *PLC*, entre outros.

Como feito na comunicação com a *Azure*, a *class* foi inicializada como pública e estática para ser acedida em qualquer zona do programa. A configuração e comunicação é feita com uma variável do tipo *S7Client*, como declarado no excerto de código:

[8] JSON é um formato compacto de troca de dados simples e rápida entre sistemas (Wikipédia)

```
1.  public static Sharp7.S7Client Client;
```

A ligação é efetuada com a função *ConnectTo*. Para configurar a ligação são necessários vários parâmetros que identifiquem o *PLC*: o *Internet Protocol*(IP), a *rack* e o *slot* do CPU. Na gama mais recente da *Siemens,* a *rack* e o *slot* são fixos, 0 e 1 respetivamente. Na gama S7 mais antiga, estes podem tomar diversos valores, dependendo da configuração de *hardware* feita.

Os dados de IP, *rack e slot* têm de ser inseridos pelo utilizador e convém haver uma confirmação se o *PLC* ligado é o correto. A função *GetCpuInfo* lê a informação do modelo, a versão e o número de série do *PLC*. Assim, ao mostrar a informação, o utilizador consegue ter uma confirmação do *PLC* que foi ligado.

No excerto de código seguinte estão as duas funções implementadas, assim como os parâmetros de entrada e o resultado.

```
1.  CONresult   =   Client.ConnectTo(PLC.IP,   PLC.Rack,   PLC.Slot);   // Ligação ao PLC
2.  CPUresult   =   Client.GetCpuInfo(ref   s7CpuInfo);    // Informação do PLC
```

A leitura dos dados é feita indicando vários parâmetros na função *ReadArea*. Os parâmetros são: a área de memória, a DB (caso a memória seja do tipo DB), o endereço de partida, a quantidade de variáveis e o tamanho das variáveis a ler. O último parâmetro é um vetor de *bytes* para onde a função vai escrever a informação.

Para a aplicação em questão, a quantidade de variáveis a ler pode ser unitária. Facilita a integração com a interface gráfica, em que o utilizador deve apenas introduzir uma variável de cada vez, evitando erros. A função de escrita *WriteArea* tem um comportamento idêntico. O código seguinte exemplifica o uso da função de leitura e de escrita para a aplicação pretendida.

```
1.  Client.ReadArea(Area, DB, Start, 1, Wlen, buffer);       // Leitura do PLC
2.  Client.WriteArea(Area, DB, Start, 1, Wlen, buffer);    // Escrita no PLC
```

4.4. Interface gráfica

A fim de tirar partido da capacidade gráfica do *Raspberry Pi 3* e da possibilidade de ligar um monitor por HDMI, optou-se por colocar as partes gráficas do programa a serem mostradas como uma aplicação do *Windows 10*. Esta abordagem permite que o programa seja mais intuitivo para o utilizador e tira partido dos recursos disponíveis.

O *IoT Core* ao tirar partido da compatibilidade com as aplicações do *Windows 10*, permite o desenvolvimento da aplicação com a mesma compatibilidade e ambiente gráfico. O desenvolvimento é feito no *Visual Studio*[9], como uma aplicação genérica para *Windows 10* em *C#*. Este tipo de aplicação

[9] Visual Studio – IDE de programação, desenvolvimento e edição gráfica do Windows (Wikipédia)

coloca o código principal como dependente da interface gráfica, ou seja, a aplicação inicia junto com a parte gráfica e irá ter uma interligação constante com a mesma.

Com o intuito de facilitar o uso da aplicação, o ambiente gráfico foi feito com o uso de bibliotecas predefinidas da *Microsoft*. É recomendado que se opte por esta via para uniformizar o aspeto das aplicações. Assim, mesmo que o utilizador não tenha tido contacto prévio com a aplicação, irá reconhecer alguns dos comandos e facilmente se adapta.

A aplicação é do tipo *Universal Windows Platform* (UWP) Esta usa a linguagem *eXtensible Application Markup Language* XAML para programar a interface gráfica. O XAML é utilizado para aplicações do *Windows* em geral.

Na **primeira abordagem** pensou-se numa aplicação com múltiplas páginas. No fluxograma da Figura 17 encontra-se a interação entre as diversas páginas e a sua funcionalidade. A aplicação iniciava numa página de configuração e ligação ao *PLC*. Só passava à próxima página caso a ligação estivesse correta e estável. Na página seguinte, configurava-se os parâmetros da ligação com a *Azure* e caso estivessem corretos, avançava para a página final. Nesta página, o objetivo era mostrar apenas o estado das comunicações. Caso alguma comunicação falhasse, voltava às configurações novamente.

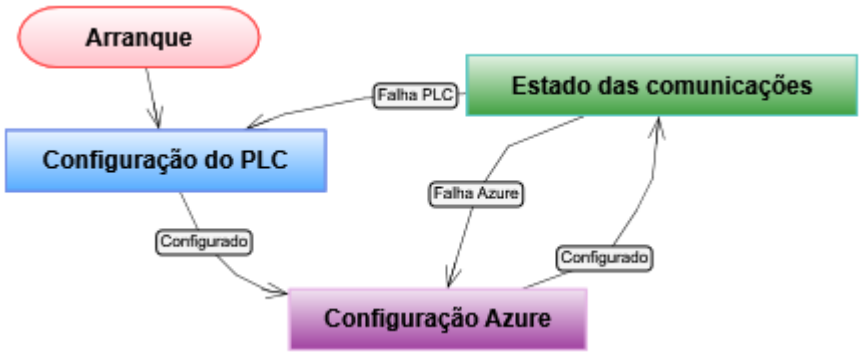

Figura 17 - Fluxograma aplicação inicial

A página de configuração do *PLC* dispunha de campos para preenchimento de modo a configurar a ligação entre a aplicação e o *PLC*. Apresentava os seguintes campos:

- IP do *PLC* – Quatro campos para colocar o endereço IPv4 do *PLC*;
- Modelo – Caixa de seleção com os diversos modelos de *PLC* compatíveis;
- Rack – A *rack* em que o *PLC* está inserido (exclusivo para *S7-400*);
- Slot – O *slot* correspondente ao CPU do *PLC* (exclusivo para *S7-400*);
- DB – Número da DB a adquirir/escrever os dados.

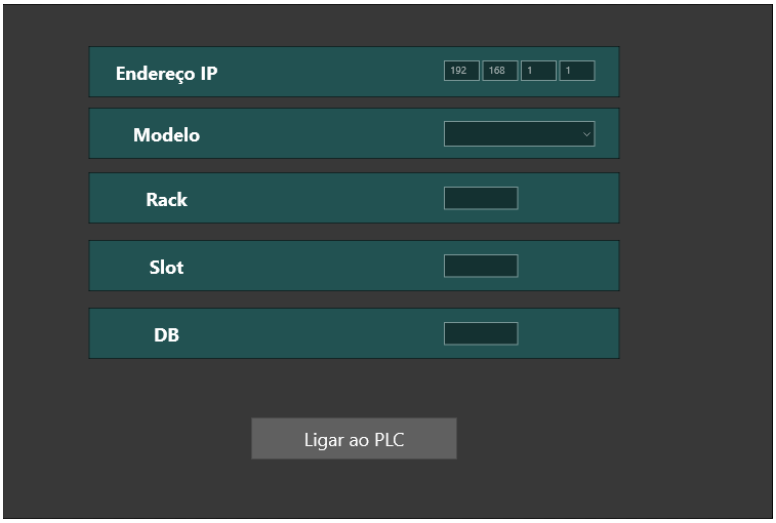

Figura 18 - Página de configuração do *PLC*

Na Figura 18 pode-se observar o aspeto gráfico da página de configuração dos parâmetros do *PLC*. Nesta encontram-se os campos para inserir os parâmetros e no final o botão para efetuar a ligação ao *PLC*. Caso não seja possível a ligação, fica na mesma página.

Os campos que definem o *PLC* são facilmente preenchidos sabendo a configuração e o modelo do *PLC*. O número da DB, que contém os dados a adquirir/escrever, torna-se mais complexo para determinar. Inicialmente, para otimizar a leitura, pensou-se em colocar todos os dados numa DB com um número fixo. Essa DB iria ter uma estrutura de dados fixa com o seguinte aspeto:

Tabela 5 - Estrutura de dados a adquirir

Variável	Tipo	*Offset (byte.bit)*
Var0	Bool	0.0
…	…	…
Var64	Int	8.0
…	…	…
Var74	DInt	28.0
…	…	…
Var82	Real	60.0
…	…	…
Var99	Real	96.0

A estrutura de dados presente na Tabela 5 contém 100 Bytes no total. Para a leitura ser otimizada, os 100Bytes eram lidos numa só vez e depois separados no programa pelas respetivas variáveis. Na parte do programa do *PLC*, os dados na DB são limitados e pouco otimizados. Além do

mais, pode-se tornar complexo o uso deste método, devido à falta de memória disponível. Considerando o pior caso de otimização, esta DB poderia estar a ser utilizada para um *bit*, ocupando apenas 0.125% da memória ocupada pela DB. Outro problema é não ter conhecimento do nome da variável. Facilmente pode conduzir a enganos na atribuição da função de cada variável. Por conseguinte, caso sejam necessárias mais variáveis do que as que estão na DB, tinha de se recorrer a outra DB, adicionando mais complexidade à estruturação do programa.

A página de configuração da *Azure* mostrou-se adequada, apenas com a desvantagem de só poder ser configurada após o *PLC*. Na página do estado de comunicação, a informação ficava limitada por só mostrar o estado de cada ligação e uma lista das últimas comunicações feitas entre o *PLC* e a *Azure*. Do ponto de vista prático, é útil para saber se a comunicação está a ser feita com sucesso, mas complica para efeitos de depuração de cada variável. Por exemplo, para o utilizador saber se a leitura de variáveis está a ser correta, tem de aceder ao *IoTHub* e ver se as mensagens recebidas estão de acordo com as variáveis lidas no *PLC*. Este processo é mais demorado, inserto e tem mais probabilidade de erros.

Após a verificação e reflexão sobre os problemas encontrados na primeira solução, optou-se por uma nova abordagem. Na **segunda abordagem** foram corrigidos alguns dos problemas encontrados na solução inicial e adicionadas outras funções que se mostraram relevantes para o bom funcionamento da aplicação.

No fluxograma da Figura 19 está representada a solução final da interface gráfica.

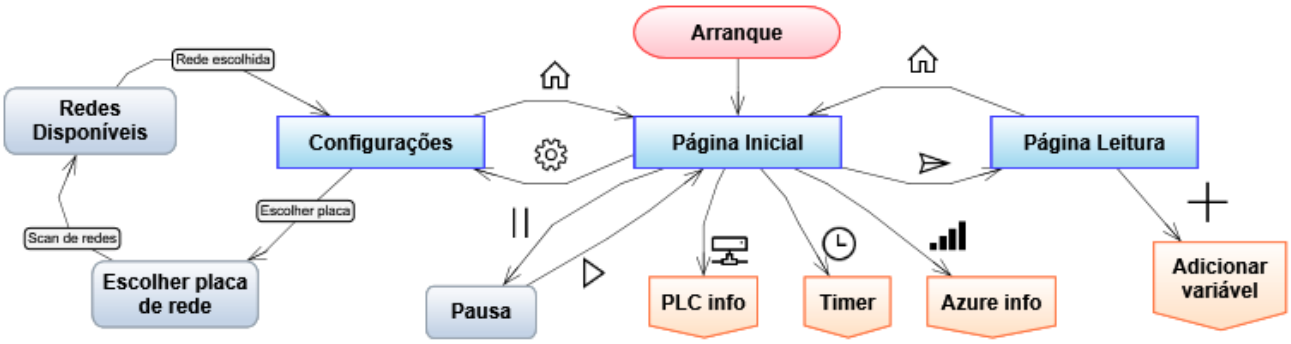

Figura 19 - Fluxograma da solução final

Quando a aplicação arranca é mostrada a página inicial que contém a informação sobre a aplicação e a integração com os outros menus. Esta página tem três colunas e um cabeçalho. A primeira coluna tem a introdução ao projeto e um resumo do mesmo. A informação sobre o estado de cada variável é apresentada na segunda e terceira coluna. Estas estão na forma de uma lista ordenada de forma ascendente pela ordem de introdução das variáveis. Na Figura 20 encontra-se a página principal como na descrição anterior.

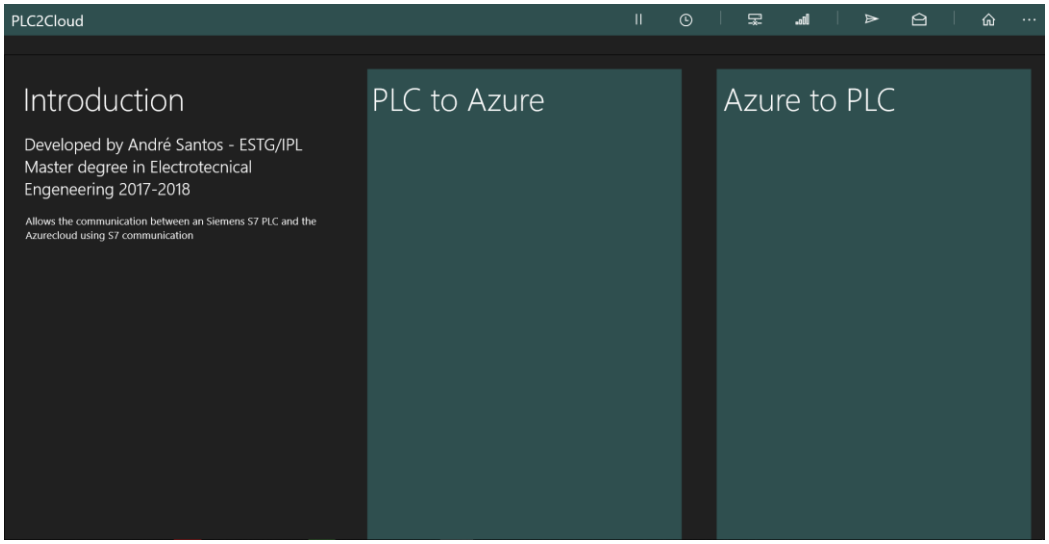

Figura 20 - Página principal da aplicação

O cabeçalho da aplicação é baseado nos cabeçalhos típicos de aplicações no *Windows 10*. A barra fica no topo da página e é comum ao resto das páginas. Os ícones da barra são dinâmicos, adaptam-se à página onde estão inseridos e a diversos estados do programa. Cada ícone está representado na Figura 19 junto à seta de transição entre páginas e menus.

Alguns dos ícones não levam a páginas novas, estes abrem menus flutuantes. Na Figura 19, os menus flutuantes estão representados a laranja. A vantagem desta abordagem é o menu aparecer e desaparecer sem necessidade de sair da mesma página. Neste sentido, o utilizador pode continuar a visualizar a página e ao mesmo tempo ter o menu aberto. Analisando a Figura 21, encontram-se os três menus de configuração da aplicação: à esquerda o tempo de aquisição que aparece clicando em ; no centro a configuração da ligação do *PLC*, clicando em ; à direita o menu de configuração da ligação com a *cloud* Azure, clicando em . Quando a ligação do *PLC* está operacional, o ícone é alterado para .O mesmo sucede com a *Azure* em que altera para .

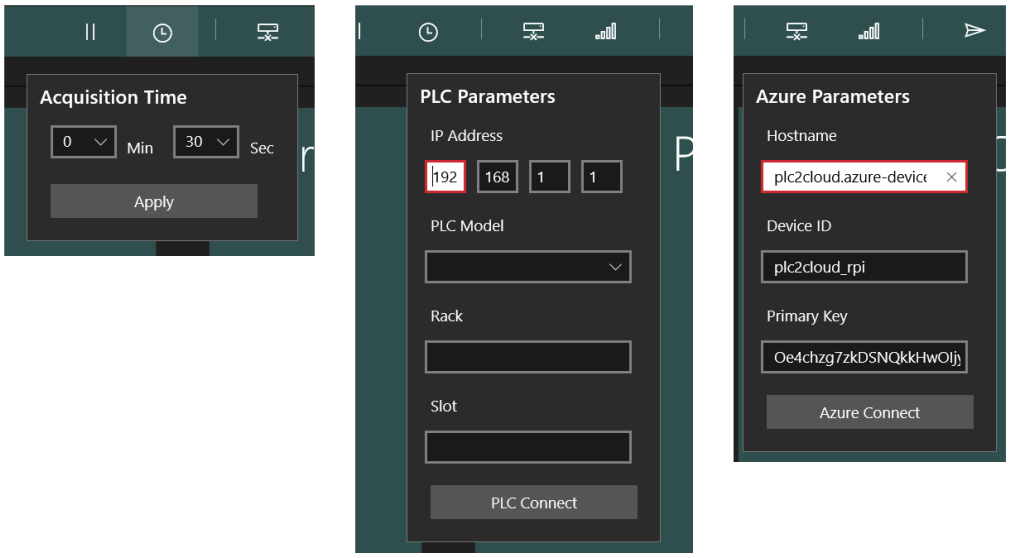

Figura 21 - Diversos menus flutuantes da página principal

O menu flutuante que configura o tempo de aquisição, permite variar a cadência com que as variáveis são lidas. Este campo tem uma importância elevada e está relacionado com vários fatores presentes no serviço *cloud*. O serviço *cloud* escolhido, que é fornecido pelo *IoTHub* de forma gratuita, só permite o envio de 8 mil mensagens diárias. Considerando um caso que se tem 10 dispositivos a comunicar, são 800 mensagens por dia por dispositivo, ou seja, aproximadamente 33 mensagens por hora. Outra situação é a saturação da rede com excesso de mensagens. Estas condições requerem que o tempo de aquisição seja regulado para a aplicação.

Focando no menu de configuração dos parâmetros do *PLC*, obtemos uma estrutura semelhante à primeira solução gráfica (Figura 18). Este tem quatro campos:

- Campo para introduzir o IP;
- campo para especificar o modelo do *PLC*;
- campo para a *Rack*;
- campo para o *Slot*.

Os campos de *Rack* e *Slot* são preenchidos de forma automática para os modelos *S7-1200* e *S7-1500*. Para concluir, é necessário clicar no botão "*PLC Connect*".

O último menu permite a configuração da *cloud*. Cada campo corresponde aos encontrados na Figura 16 e descritos no capítulo Comunicação com *IoTHub*4.2. Para terminar a configuração, é necessário clicar no botão "*Azure Connect*". Nestes menus, os campos que foram editados, guardam os valores para facilitar o utilizador, caso necessite de visualizar o estado das configurações.

As ações que não levam a páginas novas ou menus flutuantes, estão representadas a cinzento na Figura 19. Estas ações executam funções como: pausar a leitura de variáveis e configurar a placa

de rede. A interação com o utilizador na pausa da leitura é trocar o ícone de *Pause* ⏸ para *Play* ▷. No caso das configurações de rede, mostra as ligações de rede disponíveis e as placas na forma de uma lista.

Na barra da aplicação, ainda se encontra um ícone com três pontos. Ao clicar nele, a barra é expandida. Ao ser expandida, mostra uma descrição de cada ícone e dois ícones extra dentro dum menu flutuante. O ícone ? mostra uma mensagem de ajuda para a página que está a ser atualmente apresentada. O ícone ⚙ altera a página atual para a página de configuração de rede. Na Figura 22 está uma representação da barra expandida e dos ícones extra.

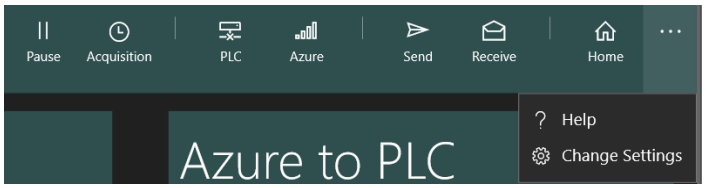

Figura 22 - Barra da aplicação expandida

A página de configuração de rede está desacoplada das outras páginas, com o intuito de separar o código. Esta separação é intuitiva já que o código desta página é o único que não influencia outras e facilita a sua programação estar separada.

Ao arrancar, a página apresenta o mesmo aspeto das outras páginas. A barra de aplicação é semelhante, mas tem apenas o botão para voltar à página principal e o de ajuda. Inicialmente, só está visível um botão com a descrição "*Network Adapters*". Este tem a funcionalidade de chamar uma função que mostra todas as placas de rede disponíveis numa lista. Após a lista ser preenchida é mostrada uma mensagem a indicar o estado da função. Cada elemento da lista contém um botão "*Use*". Ao clicar nele, assume-se que é essa a placa que se pretende utilizar. Com esta escolha, aparece um botão novo, "*Scan for Networks*" que permite fazer uma busca das redes disponíveis para a placa que foi escolhida. Novamente, aparece uma lista das redes disponíveis e um botão "*Connect*" em cada elemento da lista. Esse botão é o que permite escolher a rede a ligar. Caso seja necessária uma *password*, é apresentada uma caixa de introdução com um campo para preencher a mesma. Após estar alguma rede configurada, aparece outro botão para desligar a rede "*Disconnect*".

Na Figura 23 está representada a página de configuração de rede na fase de se ligar a uma rede e introduzir a *password*. As redes disponíveis são apresentadas junto do *Mac Address* e da amplitude de sinal.

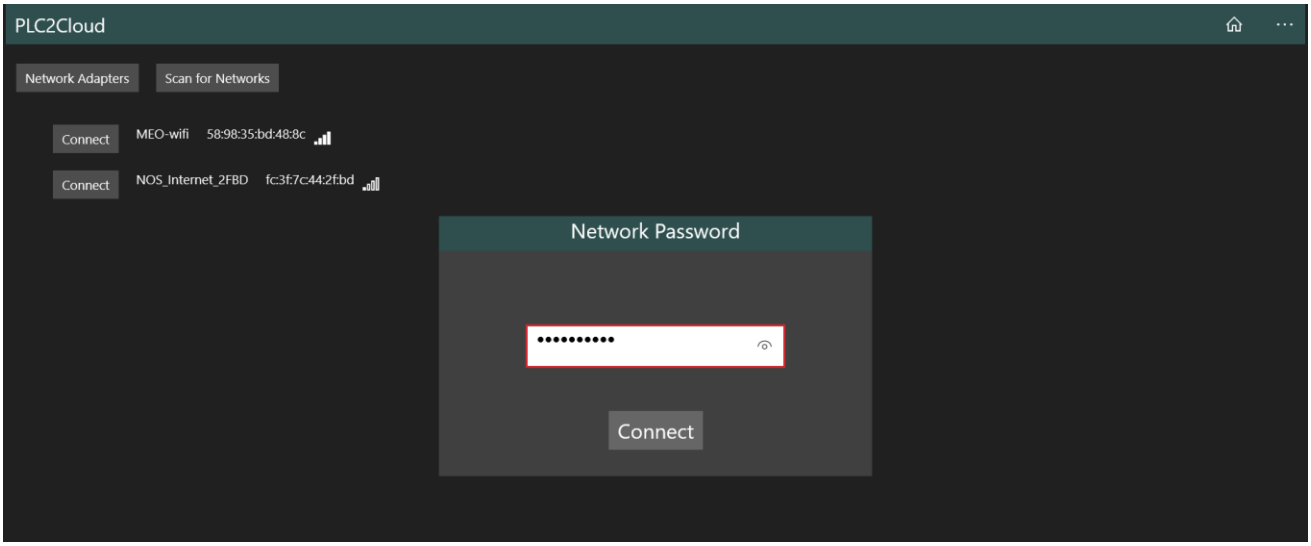

Figura 23 - Página de configuração de rede

A página que aparece ao clicar em ➢ é a página de leitura. Aqui estão presentes as variáveis adicionadas pelo utilizador e a introdução de novas variáveis. Ao entrar nesta página, aparece um botão extra na barra da aplicação que permite adicionar novas variáveis a monitorizar. O botão ┼, ao ser clicado, abre um novo menu flutuante. Este menu tem quatro ou cinco campos para preencher, dependendo do que se pretende. O primeiro campo é o nome da variável. Este nome não influencia a leitura ao *PLC*, mas é a descrição que vai distinguir a variável na *cloud*. O segundo campo é o tipo de memória a ler. Se o tipo de variável a ler pertence a uma DB, então abre um campo extra para indicar o número da DB. Caso contrário, esse campo é ocultado. O campo seguinte determina o endereço de memória onde a variável inicia. Por último, com o intuito de saber que variável está a ser lida e o número de *bits* que têm de ser lidos, é pedido o tipo de variável.

Na Figura 24 encontram-se os dois casos possíveis dos menus flutuantes para adicionar variáveis: à direita o tipo de memória não é do tipo DB e à esquerda o tipo de memória é DB e aparece o campo extra.

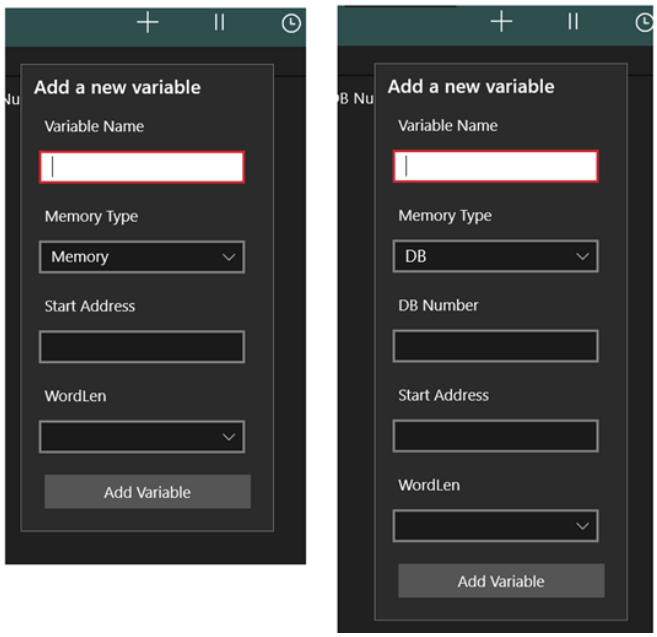

Figura 24 - Menu flutuante para adicionar nova variável

4.5. Estrutura e lógica do programa

Neste capítulo irá ser explicado em mais detalhe a lógica que opera em conjunto com a interface gráfica e em plano de fundo. As variáveis a monitorizar têm uma determinada estrutura. Dentro dessa estrutura, devem ser armazenados os dados que permitam identificar a variável e ver o seu estado. A quantidade de variáveis que o utilizador pode introduzir não é limitada pelo programa. Com esta condição é necessário criar uma lista de variáveis. A estrutura de dados tem o seguinte aspeto:

```
1.  public class t_dataRW: INotifyPropertyChanged
2.  {
3.      public int ListOrder {get; set;}
4.      public bool Enable {get; set;}
5.      public string Name {get; set;}
6.      public int Area {get; set;}
7.      public int DB {get; set;}
8.      public int Start {get; set;}
9.      public int Wordlen {get; set;}
10.     public string Type {get; set;}
11.     private string value;
12.     public string Value {get {return value;} set {this.value = value;OnPropertyChanged("Value");}
13.     public event PropertyChangedEventHandler PropertyChanged;
14.     private void OnPropertyChanged(string name)
15.     {
16.         PropertyChangedEventHandler handler = PropertyChanged;
17.         if (handler != null) {handler(this, new PropertyChangedEventArgs(name));}
18.     }
19. }
```

A *class* descrita no excerto de código anterior contém variáveis específicas para armazenar os dados que o utilizador introduz: *Name*; *Area*; *DB*; *Start*; *Wordlen*; *Type*. Três variáveis de estado: *ListOrder*; *Enable*; *value*. Por fim, tem uma função que é sensível a alterações da variável "*value*" dentro da *class*. As variáveis que o utilizador especifica são facilmente atribuídas às que se encontram

no menu flutuante de adicionar nova variável. A variável de estado *ListOrder* determina qual é a ordem que a variável irá aparecer na lista da página de leitura. A *Enable* permite saber se a leitura da variável está habilitada ou não. A *value* é intuitiva, representando o valor que foi lido do *PLC*. Esta *class*, para facilitar a estrutura do programa, está num ficheiro diferente do código principal denominado *DataTypes.cs*.

A lista de variáveis vai conter a *class* descrita anteriormente em cada posição. Logo, esta lista irá ser do tipo *DataTypes.t.dataRW*. São necessárias duas listas, uma para armazenar os dados lidos do *PLC* e outra para a escrita. No excerto de código seguinte estão descritas as duas listas:

```
1.  List < DataTypes.t_dataRW > L_DataRead = new List < DataTypes.t_dataRW > ();
2.  List < DataTypes.t_dataRW > L_DataWrite = new List < DataTypes.t_dataRW > ();
```

Como descrito no início do capítulo 4, o programa é cíclico, garantindo que as variáveis são lidas numa cadência adequada. Para gerar uma interrupção cíclica utilizou-se um *thread* do tipo *DispatcherTimer*. O tempo inicial da interrupção é de 30 em 30 segundos e o seu valor pode ser alterado, recorrendo ao menu flutuante do tempo de aquisição. No excerto de código seguinte está a declaração da interrupção na linha 1. Na linha 2 a função que é chamada cada vez que é despoletada a interrupção. Na linha 3 o intervalo de tempo. Na última linha está a inicialização do temporizador.

```
1.  private DispatcherTimer dis = new DispatcherTimer();
2.  dis.Tick += new EventHandler < object > (SendCloudMessage);
3.  dis.Interval = new TimeSpan(0, 0, 30);
4.  dis.Start();
```

A função *SendCloudMessage* tem como objetivo controlar a leitura das mensagens provenientes do *PLC* e enviá-las para a *cloud*. A primeira verificação que esta função executa é se existe alguma variável para ler e se a *cloud* está configurada. Se estas condições forem verificadas, é executado um ciclo que passa por todas as variáveis da lista para ler. Considerando que o ciclo está na primeira posição da lista, é verificado se o campo *Enable* está a *true*. Se sim, inicia a leitura da variável ao *PLC*, caso este estiver configurado. Após receber os dados do *PLC*, verifica se a ligação à *cloud* está operacional e envia os dados. Quando ocorre um erro, a função é terminada e é mostrada uma mensagem ao utilizador.

A interrupção cíclica pode ser colocada em pausa pelo utilizador. Isto faz com que a leitura das variáveis ao *PLC* e o envio para a *cloud* fiquem interrompidos. A solução encontrada mais eficiente passa por desabilitar o *DispatcherTimer*. O botão ao ser pressionado executa uma função que é declarada como no código XAML do botão. Quando essa função é executada, a primeira verificação é relacionada com o estado atual do *DispatcherTimer*. Se estiver habilitado, a interrupção é colocada em pausa, caso contrário é iniciado. Para complementar com a interface gráfica, o ícone é trocado do

símbolo *play* para o símbolo *pause*. No excerto de código seguinte, encontra-se a colocação em pausa da interrupção.

```
if (!dis.IsEnabled)
{
    Header_Pause.Icon = new SymbolIcon(Symbol.Pause);
    dis.Start();
} else
{
    Header_Pause.Icon = new SymbolIcon(Symbol.Play);
    dis.Stop();
}
```

Para alterar o tempo da interrupção, é modificado no *DispatcherTimer* o tempo entre interrupções, garantindo que o *thread* é chamado quando necessário. A função que executa a troca do tempo é chamada com o botão "*Apply"* presente no menu flutuante, correspondente ao tempo de aquisição. Dentro da função, é verificado se o tempo escolhido não é inválido (igual a zero). Por conseguinte, é atribuído diretamente na função "*Interval*" como efetuado anteriormente na configuração inicial.

A lista de variáveis disponibilizada ao utilizador é alterada de forma cíclica pela interrupção do *timer*. Com esta condição não há necessidade de estar constantemente a atualizar a lista na interface gráfica. Assim, existe menos consumo de recursos gráficos. Cada vez que a função *Listview_atualizar* é chamada, as listas na interface gráfica tomam o valor das listas onde as variáveis são armazenadas. Isto também acontece nas transições entre páginas para garantir que o valor mostrado é o mais atualizado.

Quando é clicado num dos botões presentes na barra de aplicação para mudar de página, além de ser chamada a função para atualizar as listas, a visibilidade das páginas também é modificada. Por exemplo, numa transição para a página principal, apenas a página principal fica visível, as restantes ficam ocultas. Para permitir que a página fique toda oculta, o conteúdo da página fica dentro de uma estrutura do tipo *Border*. Quando a página é ocultada é a esta estrutura que se altera a visibilidade.

Todos os campos editáveis pelo utilizador que não tenham valores predefinidos são sujeitos a revisão, com o objetivo de garantir que estão dentro dos limites impostos. A verificação é feita, convertendo a *string* que é recebida em inteiro e, após a conversão, o valor é comparado com os limites. Outro aspeto que é verificado reside se a *string* é só constituída por números. Desta forma, evita sinais e caracteres parasitas que possam levar a conversões erradas. Os campos que estejam fora dos limites ou com caracteres inválidos ficam com o fundo a vermelho.

No caso da verificação das *strings* de ligação ao *IoTHub*, o *hostname* tem sempre um conjunto de caracteres comum "*.azure-devices.net*". Cada vez que um *hostname* é introduzido, é procurada na *string* esse conjunto de caracteres. Caso não estiver disponível, a ligação não é possível e o campo

aparece com o fundo a vermelho. A outra verificação feita nestes campos é se os campos não estão vazios.

O utilizador quando clica no botão para adicionar uma nova variável a ser monitorizada, abre a função correspondente ao clique do botão. Essa função verifica primeiro se os campos que o utilizador introduziu estão corretos. Caso a condição se verificar, existem duas ações possíveis baseadas na área de memória escolhida:

- DB – As variáveis são todas introduzidas na lista;
- Não DB – As variáveis são atribuídas à lista e o campo de DB é colocado como zero.

Os dados armazenados que não são *strings* devem ser convertidos do mesmo para o formato específico de cada variável. Deste modo, a variável pode ser utilizada em qualquer parte do programa sem necessitar de mais conversões.

O botão para adicionar variável só é utilizado na leitura de variáveis do *PLC*. No caso da escrita, para a aplicação atual é irrelevante. Este facto deve-se às variáveis recebidas pelo *IoTHub* virem num formato pronto a ser descodificado no programa e enviado para o *PLC* para a área de memória pedida. Este botão pode ser relevante se a variável recebida apenas tiver o nome e o valor a modificar. Assim sendo, o utilizador tem de colocar os endereços do *PLC* em que o programa vai escrever o valor dessa variável recebido pelo *IoTHub*.

Cada elemento da lista tem um *ToggleSwitch* para habilitar a leitura. A lista está ligada com a lista da parte gráfica e os dados só transitam nesse sentido. Com este método, não é possível alterar diretamente os valores de cada *ToggleSwitch* na lista de variáveis. Para contornar o problema, ao clicar num *ToggleSwitch* é chamada uma função que é genérica a todos os *ToggleSwitchs* da lista, alterando o valor dentro da lista de variáveis. Este método otimiza a quantidade de funções, mas cada *ToggleSwitch* tem de ter algo que o diferencie dos restantes. Ao adicionar à *tag* do *ToggleSwitch* a posição da lista onde se encontra o elemento, o problema fica resolvido. Quando a função é chamada a *tag* é acedida através do parâmetro de entrada "*sender*", aplicando de um *cast* para o tipo *ToggleSwitch*. No excerto de código seguinte encontra-se a leitura do *ToggleSwitch*.

```
1.  private void ConfigRead_Toggle_Toggled(object sender, RoutedEventArgs e)
2.  {
3.      int i = (int)((ToggleSwitch) sender).GetValue(Grid.RowProperty) - 1;
4.      if (i < L_DataRead.Count && i >= 0)
5.          L_DataRead[i].Enable = ((ToggleSwitch) sender).IsOn;
6.      Listview_atualizar();
7.  }
```

O acesso à página de configuração de rede é diferenciado das outras páginas. Como o código está separado e a página não pertence à página principal, o seu acesso tem de ser através da função que navega entre páginas. Esta está presente no excerto de código seguinte:

```
1. Frame.Navigate(typeof(WifiConnectPage));
```

A transição para a página principal é feita com a mesma função. Apenas é necessário alterar a página que é chamada na função de *WifiConnectPage* para *MainPage*.

Após a transição para a página de configuração de rede, o novo *layout* é carregado. Quando é clicado no botão "*Network Adapters*", a visibilidade da lista de placas de rede é colocada como visível e a lista de redes fica a invisível. Quando a lista fica visível, é executada a função *AdapterRefresh* que pesquisa todas as placas de rede disponíveis. A pesquisa é feita através da *class WiFiAdapter* que devolve uma lista dos adaptadores *Wi-Fi* ligados. Após terminar, cada adaptador é colocado numa lista para ser mostrada na página. Em adição, é mostrada uma mensagem ao utilizador a avisar que a lista foi atualizada com sucesso.

A lista pode ter vários elementos e cada elemento tem um botão que permite a sua utilização para as próximas funções. O próximo botão a ser clicado é o "*Scan for Networks*", que como o nome sugere, lista todas as redes sem fios disponíveis. A procura é feita através de uma função assíncrona do adaptador escolhido. Como a procura pode ser demorada, é habilitado um anel rotativo a simbolizar algo em espera com um tempo indeterminado. Após a procura estar concluída, o anel rotativo é retirado. Seguidamente, é enviada uma mensagem para o utilizador com o sucesso da ação. Por último, a lista de redes sem fios é colocada como visível e a lista de adaptadores como invisível.

Após a lista ser mostrada, o utilizador já pode selecionar a rede sem fios que pretende. Ao clicar no botão "*Connect*" em linha com a rede desejada, está a despoletar uma função idêntica às outras utilizadas no programa para captar botões em listas. Essa função tem duas verificações para que a ligação seja ou não efetuada. A primeira verifica se a rede necessita de autenticação. Se não necessitar, liga-se automaticamente, caso contrário pede a *password* da rede ao utilizador. Quando esta for inserida e o botão "*Connect*" for clicado, é feita uma tentativa de ligação. A segunda verificação está relacionada com o resultado, verificando se houve sucesso a ligar-se à respetiva rede. Se sim, envia uma mensagem de sucesso na operação, caso contrário é enviada uma mensagem de erro. Quando a ligação tiver sucesso, a lista de redes desaparece.

5. Testes à aplicação

O objetivo deste capítulo é mostrar e tirar conclusões dos testes que foram feitos ao programa e ao serviço *cloud*. Os testes têm o intuito de verificar a robustez da comunicação entre o *PLC* e o dispositivo, a comunicação entre o dispositivo e o *IoTHub* e a estabilidade da interface gráfica.

Para testar a comunicação entre o *PLC* e o dispositivo utilizou-se um teste que permite saber a robustez da comunicação. Este tem os seguintes passos:

- Configurar o *PLC* e adicionar uma variável para leitura. Esperar um dia e verificar se houve falha na comunicação.
- Adicionar cinco variáveis e colocar um tempo de leitura de 5 segundos. Guardar o tempo entre cada leitura durante 100 amostras.

Para analisar os dados recebidos utilizou-se uma *class* com várias funções feitas especificamente para recolher os dados desta aplicação. Após a inicialização da leitura de uma nova variável, começa a registar os tempos entre leituras numa lista. No fim de uma quantidade de registos armazenados, os dados são guardados num ficheiro *Comma Separated Values* (CSV) para serem analisados posteriormente no *Excel*, tirando conclusões da performance do programa.

A função de leitura fica operacional ao iniciar um novo registo com a função *Start_reg*. Esta irá colocar as variáveis num estado inicial. No programa é necessário colocar a função *Start_read*, onde irá iniciar a contagem do tempo. Esta função reinicia um cronómetro. Para terminar o cronómetro deve-se chamar a função *Stop_read*. Esta guarda a nova amostra na lista de amostras. Se for a última amostra a ler, inicia a escrita do ficheiro CSV através da função *WriteFile*.

No primeiro teste, a ligação com o *PLC* mostrou-se estável. Durante o período de um dia, a variável foi lida com sucesso e sem interrupções. O *PLC* estava com 20% da capacidade de processamento dedicada a comunicações. Este valor é a configuração de fábrica.

O teste tem como objetivo verificar a resposta da comunicação. Assim, é possível aferir o tempo médio que a aplicação demora a adquirir uma variável. No gráfico da Figura 25 encontram-se os dados recolhidos. No eixo horizontal o tempo de aquisição de uma variável e no eixo vertical a quantidade de amostras associadas ao respetivo tempo de aquisição.

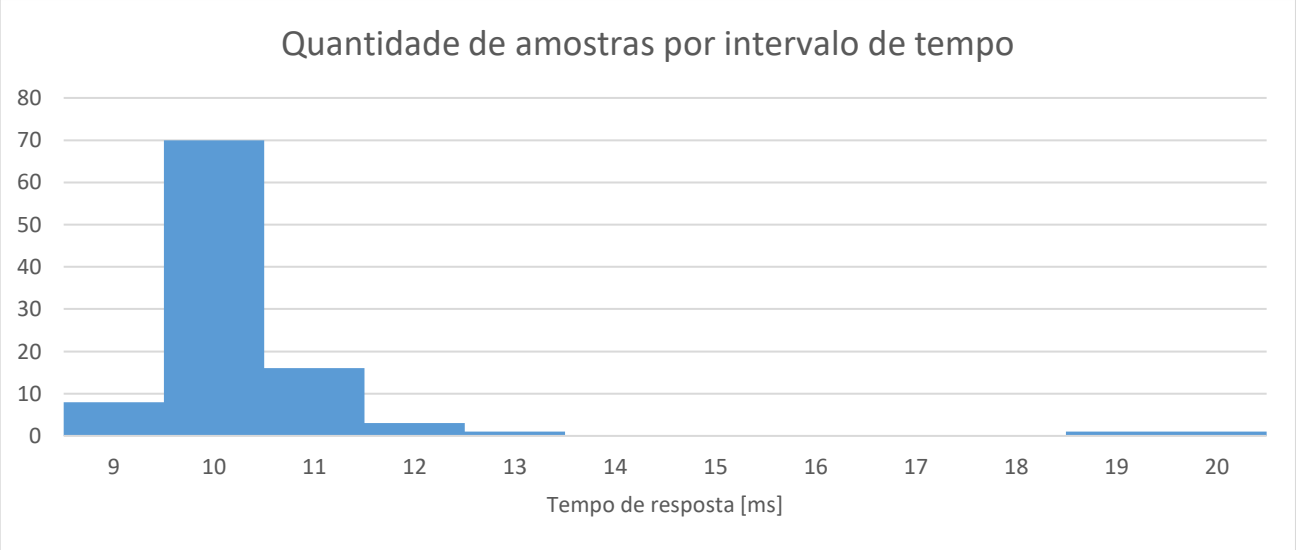

Figura 25 - Quantidade de amostras por intervalo de tempo (*PLC*)

Analisando o gráfico, o tempo de resposta que sobressai é de 10ms, com 70 ocorrências em 100 amostras. O tempo mínimo é de 9ms e o máximo de 20ms. O tempo médio é 10.36ms com desvio padrão de 1.4595ms. Estas amostras comprovam que o tempo de resposta é baixo para a comunicação entre o *PLC* e a aplicação. Como a leitura mínima que se pode definir é de 5s, com o tempo médio registado nas amostras e incluindo eventuais atrasos na manipulação dos dados, obtém-se uma quantidade de leituras aproximada de 400 variáveis por 5s.

O teste para a comunicação entre o dispositivo e a *cloud* implica que a comunicação com o *PLC* esteja estável. Caso se verifique essa condição, o teste pode ser feito com os seguintes passos:

- Configurar a ligação com a *cloud* e enviar uma variável de 30 em 30 segundos, esperar um dia e verificar se houve falha na comunicação.
- Adicionar cinco variáveis de cada tipo e colocar um tempo de leitura de 30 segundos. Guardar o tempo entre cada envio durante 1000 amostras.
- Verificar após um dia se a quantidade de mensagens recebidas na *cloud* é igual à quantidade de mensagens enviadas.

Tal como nas amostras recolhidas na comunicação com o *PLC*, no caso da comunicação com a *cloud*, as amostras são recolhidas com a mesma *class*. Deste modo, terão como objetivo gerar um ficheiro CSV para ser analisado e tirar conclusões.

O primeiro teste permitiu verificar a estabilidade da comunicação. Este teste não depende só da aplicação e da *cloud*, depende muito da infraestrutura de telecomunicações entre os mesmos. A comunicação falhou após um tempo de funcionamento aproximado de 3 horas, mas retomou logo de seguida devido a uma falha de rede. Esta falha mostrou a capacidade da aplicação se regenerar e voltar

a comunicar. Foram registadas outras 3 falhas devido a ocorrências externas ao programa, mas nenhuma impediu o funcionamento normal da aplicação.

No próximo teste, tal como no *PLC*, testou-se o tempo de resposta da função que envia o pacote através do protocolo MQTT para o *IoT Hub*. Na Figura 26 estão representadas através de um gráfico as amostras recolhidas. No eixo horizontal encontra-se o tempo de resposta e no eixo vertical a quantidade de amostras para cada tempo de resposta.

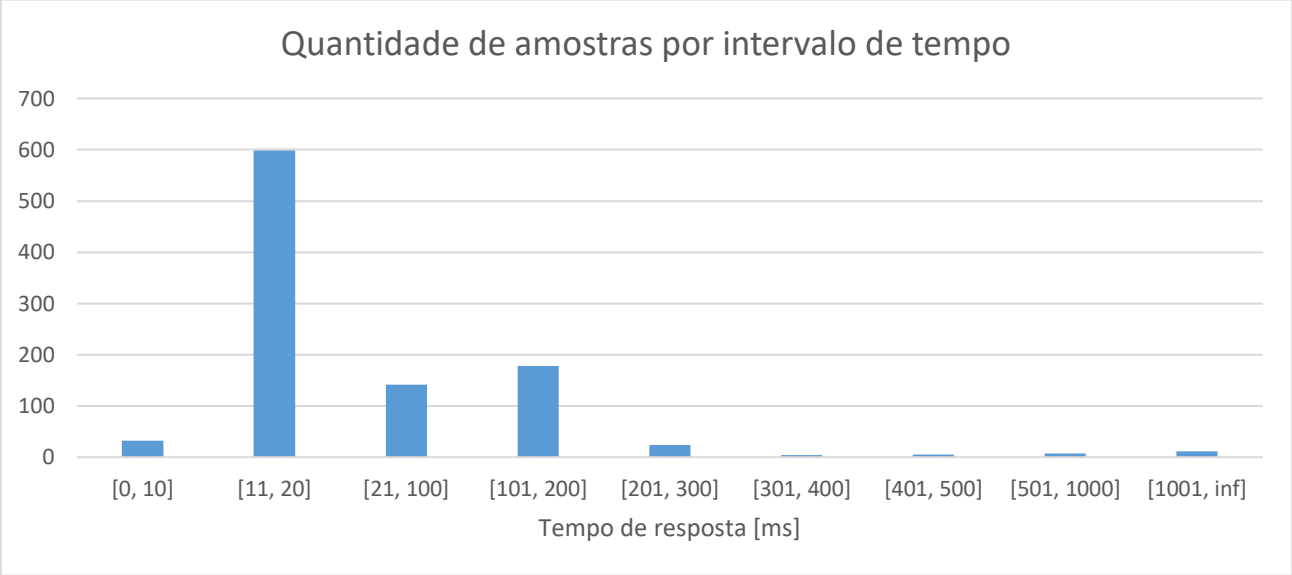

Figura 26 - Quantidade de amostras por intervalo de tempo (Azure)

As mil amostras recolhidas mostraram que cerca de 60% das mesmas estão na gama de 11ms a 20ms. Aproximadamente 97% das amostras são inferiores a 300ms e a média ronda os 111ms. Apesar disso, grande parte dos valores estão compreendidos entre os 11ms e os 20ms. Estes valores mostram que grande parte dos pacotes são enviados num tempo aceitável. Portanto, considerando a média registada, com um tempo de aquisição de 5s, são enviadas aproximadamente 45 variáveis por cada 5s. Este valor está escravo de diversos fatores como: o tempo de processamento entre envios; a capacidade da infraestrutura de comunicação; o tempo de aquisição da variável no *PLC*.

O último teste verifica se todas as variáveis foram recebidas no *IoT Hub*. As leituras estão disponíveis no portal da *Azure* num gráfico como disponibilizado na Figura 27.

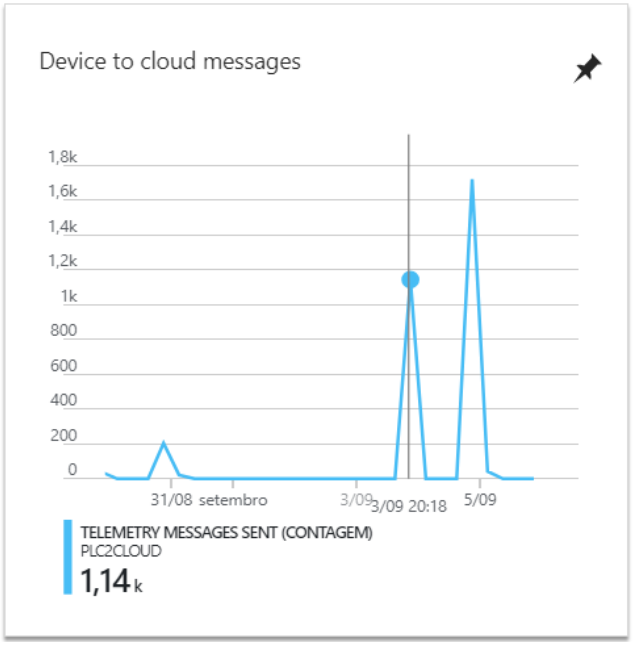

Figura 27 - Quantidade de mensagens recebidas pelo *IoT* Hub

A contagem que está na Figura 27 com a bola a cheio e o traço vertical, representa o dia em que o teste foi feito. A quantidade de amostras ultrapassa as mil mensagens, embora só tenham sido envidas mil. Isto deveu-se a 150 mensagens extra que foram enviadas no mesmo dia que a contagem foi habilitada. Subtraindo as 150 mensagens extra, verifica-se um sucesso de 99% nas mensagens enviadas da aplicação para o *IoT Hub*. O restante 1% corresponde a eventuais falhas de comunicação, tal como aconteceu no primeiro teste.

Os testes para a interface gráfica são feitos através da análise visual de falhas na aplicação e fluidez do ambiente gráfico. Os resultados apresentados terão em conta várias tentativas ao mesmo teste, a fim de garantir um resultado mais fiável. Caso existam melhorias possíveis para as falhas, serão apresentadas em conjunto com a descrição da mesma.

O primeiro teste efetuado verificou a fluidez do ambiente gráfico. Este parâmetro depende das características gráficas do computador. Além do mais, é influenciado pela quantidade de elementos que estão presentes no momento, pelas transições nos elementos e depende também da resolução do ecrã. Como a aplicação foi desenvolvida fora num computador com *hardware* muito superior ao *Raspberry Pi 3*, é de extrema importância que o teste seja feito também no *Raspberry Pi 3*.

A aplicação foi primeiro testada no computador onde foi desenvolvida, com resolução *Full High Defenition* (FullHD). As transições entre páginas são fluidas e as animações não apresentam arrastamento. Em funcionamento com dez variáveis a serem adquiridas de 30 em 30 segundos, a quantidade de memória utilizada é sempre inferior a 100MB. Os valores das variáveis são apresentados sem atrasos percetíveis.

O teste no *Raspberry Pi 3* para comparação foi feito no mesmo ecrã *FullHD*. O aspeto é semelhante ao encontrado no PC, mas a fluidez da aplicação é inferior. É notável algum arrastamento quando se abre um menu flutuante, mas não compromete a sua funcionalidade. As tarefas de configuração de rede, necessitam de pouco tempo para serem executadas. O desempenho a atualizar as listas de variáveis é semelhante ao encontrado no computador onde foi desenvolvido, não se notam atrasos no processo.

A fim de melhorar o funcionamento no *Raspeberry Pi 3*, optou-se por baixar a resolução do ecrã para 1280x800. Notaram-se algumas melhorias no desempenho, mas pouco significativas. Este facto conduziu que se regressasse à resolução *FullHD*, aumentando a qualidade de imagem. Os aplicativos a funcionar em segundo plano foram terminados, permitindo que o sistema operativo só tenha uma página *Web* para consultar o estado do mesmo e a aplicação desenvolvida. Para a aplicação iniciar em conjunto com o *Windows*, foi colocada como a aplicação principal de arranque.

Ao introduzir valores nos campos presentes nos menus flutuantes, a teclado ficou preso em alguns ensaios. Este fenómeno faz com que sejam escritos imensos caracteres. Ao sair do menu tudo volta ao normal. Os campos nos menus flutuantes são os únicos afetados por este fenómeno e não se encontrou solução para o remediar.

Na Figura 28 encontram-se vários gráficos a representar o desempenho do *Raspberry Pi 3* com a aplicação desenvolvida. No gráfico "*CPU*" encontram-se duas setas. Estas indicam o início e o fim de uma ação de abertura de um menu flutuante. Como é possível observar, houve um aumento aproximado de 20% para 70% da utilização do *CPU*. O mesmo efeito repete-se no gráfico "*I/O*", devendo-se à movimentação do rato e inserção de caracteres pelo teclado. Estes, ao estarem ligados por *USB*, influenciam diretamente as entradas e saídas. No gráfico "*Memory*" não se verificam alterações percetíveis, acontecendo o mesmo de quando a aplicação correu em modo depuração no computador de desenvolvimento. Ao analisar a memória requisitada verificou-se um total de 60MB, quando a aplicação está em funcionamento. A restante memória é utilizada pelo sistema operativo. O uso da rede é muito baixo, não influenciando no desempenho da aplicação.

Figura 28 - Desempenho da aplicação no Raspberry Pi 3

6. Conclusão

O trabalho desenvolvido permitiu a junção de dois campos: *PLC* e o *cloud computing*. As potencialidades da interligação dos dados provenientes dos *PLCs* com um serviço *cloud* proporcionam à indústria capacidades de monitorização, decisão e previsão. Este estudo também se enquadra em vários paradigmas do conceito de Indústria 4.0.

Este trabalho foi desenvolvido para indústrias que utilizem *PLCs* com comunicação através do protocolo S7 e que pretendam enviar os dados do *PLC* para o *IoT Hub*. A estrutura do programa está feita de forma a permitir o uso de outros protocolos e serviços *cloud*. No caso do uso de outro serviço *cloud*, é necessário alterar a função que se liga ao servidor e envia os dados para a pretendida.

Com a evolução da indústria e a necessidade eminente de ter conhecimento de cada processo, o trabalho desenvolvido enquadra-se na parte de aquisição dos dados. Após esta aquisição, os dados ficam disponíveis para diversas aplicações. Através de serviços disponibilizados pelos fornecedores da *cloud*, os dados podem ser analisados, armazenados e estruturados. Deste modo, a informação recolhida torna-se útil através de relatórios, gráficos, tabelas, entre outros. Além disso, a informação pode passar por algoritmos para otimizar a produção de forma automática. Assim sendo, a empresa pode estar constantemente a evoluir e a melhorar a sua produção.

A fim de interligar o *PLC* ao serviço *cloud*, utilizou-se um dispositivo intermédio funcionando como uma ponte. Este lê os dados diretamente da memória do *PLC*, processa-os de uma forma adequada a ser recebido pela *cloud* e envia-os. A comunicação é feita por cabo *Ethernet*, recorrendo ao protocolo S7. A escolha baseou-se na menor intervenção possível na programação do *PLC*. O protocolo de comunicação entre o dispositivo e a *cloud* foi o MQTT, pela sua baixa necessidade de recursos.

A comunicação entre a aplicação desenvolvida e o PLC mostrou-se com uma performance adequada. Os tempos de aquisição de uma variável rondam os 10 milissegundos e a estabilidade é elevada nos testes efetuados. A comunicação entre a aplicação e o *IoTHub* apresentou um tempo de transição médio de 111milissegundos para cada mensagem. A aplicação foi limitada a um tempo mínimo de 5 segundos entre leituras de cada conjunto de variáveis. Este limite deve ser ajustado com a quantidade de mensagens a ser enviada para a *cloud*. Aproximadamente 99% das mensagens que foram enviadas chegaram com sucesso à mesma. Para aplicações de monitorização, esta taxa de sucesso é suficiente.

O tempo entre a aquisição da variável e a sua chegada à *cloud* depende da estrutura de rede onde está inserido. Para melhorar o tempo de transição da mensagem entre o *PLC* e a *cloud*, uma

abordagem pode passar por melhorar a rede e o serviço de internet. Para aplicações com mais cadência de mensagens por segundo, uma melhoria será juntar várias variáveis em cada mensagem e enviar tudo no mesmo pacote. O desempenho gráfico da aplicação no *Raspberry Pi 3* mostrou-se suficiente e estável. As animações apresentam algum atraso quando comparadas com um computador, mas não afetam o seu funcionamento.

No projeto não correu tudo de forma expectável. Como tal, sugiro algumas alterações e sugestões de melhoria. Analisando a aplicação, esta pode ser otimizada e melhorada na interface gráfica e na lógica. As listas de variáveis a ler deviam ser mais dinâmicas, adicionando opções para as modificar e eliminar. As configurações feitas no programa deviam ser guardadas num ficheiro para não se perderem. A página de configuração de rede devia ser um menu flutuante acessível através da barra de aplicação, para se assemelhar à configuração de rede do *Windows*. A receção de variáveis pela *cloud* foi desabilitada, embora permaneça na interface gráfica. Esta decisão foi tomada devido ao seu potencial perigo a nível de segurança. A escrita em qualquer memória do PLC a partir de um servidor remoto sem qualquer controlo, falha na segurança.

O uso do *IoT Hub* mostrou facilidade no acesso à *cloud*. É uma plataforma poderosa quando se pretende ter muitos dispositivos a comunicar, em que cada um tem uma pequena quantidade de dados a ser transmitida/recebida. Os dados recebidos são facilmente distribuídos para outras aplicações disponíveis na *Azure* através do *Stream Analytics*. Para aplicações que necessitem transitar mais de 4kB de informação por mensagem, este serviço não o permite. Neste caso é sugerida uma abordagem diferente, como por exemplo um servidor na *cloud* que receba as mensagens.

Bibliografia

[1] A. M. Ferreira, Introdução ao Cloud Computing, FCA, 2015.

[2] Siemens, *O que é a Indústria 4.0*, 2017.

[3] P. Mell e T. Grance, "The NIST Definition of Cloud Computing," *National Institute of Standards and Technology Special Publication 800-145*, p. 7, Setembro 2011.

[4] "Wikipédia," Junho 2018. [Online]. Available: https://pt.wikipedia.org/wiki/Controlador_lógico_programável. [Acedido em Julho 2018].

[5] Q. Zhang, L. Cheng e B. Raouf, "Cloud computing: state-of-the-art and research challenges," 2010.

[6] Y. Kim, Y.-H. Kim, G.-W. Lee e E.-N. Huh, "Survey of Big Data-as-a-Serice Type," em *17th International Conference on High Performance Computing and Communications*, 2015.

[7] P. Chavan, P. Patil, G. Kulkarni, R. Sutar e S. Belsare, "IaaS Cloud Security," em *International Conference on Machine Intelligence and research Advancement*, 2013.

[8] M. K. Sarkar e S. Kumar, "A Framework to Ensure Data Storage Security in Cloud Computing," em *IEEE*, 2016.

[9] B. Sotomayor, R. Montero, I. Llorente e I. Foster, "Virtual Infraestruture Management in Private and Hybrid Clouds," 2009.

[10] "Microsoft Azure," [Online]. Available: https://azure.microsoft.com/pt-pt/overview/what-is-saas/. [Acedido em Janeiro 2018].

[11] A. Keshavarzi, A. T. Haghighat e M. Bohlouli, "research Challenges and Prospective Business Impacts of Cloud Computing: A Survey," em *7th IEEE International Conference on Intelligent Data Acquisition and advanced Computing Systems: Technology and Applications*, Berlin, 2013.

[12] S. Kozak, E. Ruzicky, J. Stefanovic e F. Schindler, "Research and Education for Industry 4.0," em *29th International Conference Cybernetics & Informatics*, Slovakia, 2018.

[13] "Simio," [Online]. Available: https://www.simio.com/applications/industry-40/index.php. [Acedido em Abril 2018].

[14] J. Morgan, "Forbes," 13 Maio 2014. [Online]. Available: https://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/#3750ea821d09. [Acedido em 02 2018].

[15] S. Bauk, T. Dlabac e M. Skuric, "Internet of Things, High Resolution Management and New Business Models," em *International Scientific-Professional Conference on Information Technology*, Montenegro, 2018.

[16] M. hung, "Gartner," 2017. [Online]. Available: https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf.

[17] "SAS," [Online]. Available: https://www.sas.com/en_us/insights/big-data/what-is-big-data.html. [Acedido em Março 2018].

[18] B. Imene e H. Salima, "Verifiable Outsorced Computation Integrity in Cloud-Assisted Big Data Processing," em *IEEE*, 2018.

[19] "Concrete," Junho 2017. [Online]. Available: https://www.concrete.com.br/2017/06/02/uma-pequena-introducao-a-big-data/. [Acedido em Março 2018].

[20] G. S. Aujla, R. Chaudhary, N. Kumar, A. K. Das e J. Rodrigues, "SecSVA: Secure Storage, Verification, and Auditing of Big Data in the Cloud Environment," em *IEEE Communications Magazine*, 2018.

[21] "World Economic Forum," 2015. [Online]. Available: http://reports.weforum.org/global-risks-2015/part-1-global-risks-2015/technological-risks-back-to-the-future/?doing_wp_cron=1523110797.2531700134277343750000.

[22] A. S. Sohal, R. Sandhu, S. K. Sood e V. Chang, "A cybersecurity framework to identify malicious edge device in fog computing and clou-of-things environments," *Computers and Security*, pp. 340-354, 2018.

[23] "RightScale," 2018. [Online]. Available: https://www.rightscale.com/lp/state-of-the-cloud.

[24] "Siemens MindSphere," [Online]. Available: https://www.siemens.com/global/en/home/products/software/mindsphere.html. [Acedido em Agosto 2018].

[25] "Siemens MindConnect," [Online]. Available: https://www.plm.automation.siemens.com/store/en-se/mindsphere/mindconnect/. [Acedido em Agosto 2018].

[26] "Siemens MindAccess," [Online]. Available: https://www.plm.automation.siemens.com/media/store/en_ie/Siemens-PLM-MindSphere-MindAccess-IoT-Value-Plan-fs-66303-A18_tcm34-19173.pdf. [Acedido em Agosto 2018].

[27] "Siemens MindApps," [Online]. Available: https://www.plm.automation.siemens.com/store/en-se/mindsphere/apps/. [Acedido em Agosto 2018].

[28] "Proficloud," [Online]. Available: https://www.phoenixcontact.com/assets/downloads_ed/global/web_dwl_promotion/52005773_EN_HQ_PROFICLOUD_LoRes.pdf. [Acedido em Julho 2017].

[29] Microsoft. [Online]. Available: https://azure.microsoft.com. [Acedido em Junho 2017].

[30] "Google Cloud Platform," [Online]. Available: https://cloud.google.com. [Acedido em Julho 2017].

[31] "Amazon Web Services," [Online]. Available: https://aws.amazon.com/pt/iot/?nc2=h_m1. [Acedido em Agosto 2018].

[32] D. Betts e R. Shahan, "Microsoft Azure," 29 01 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-protocols.

[33] "Wikipédia - ESP8266," 5 Setembro 2018. [Online]. Available: https://en.wikipedia.org/wiki/ESP8266.

[34] "GitHub," [Online]. Available: https://github.com/Azure-Samples/iot-hub-c-huzzah-getstartedkit. [Acedido em Maio 2018].

[35] "Wikipédia - ESP32," 3 Setembro 2018. [Online]. Available: https://en.wikipedia.org/wiki/ESP32.

[36] "Raspberry Pi," [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Acedido em 11 Setembro 2018].

[37] Microsoft, "GitHUB," [Online]. Available: https://github.com/Microsoft/Windows-iotcore-samples. [Acedido em 10 Julho 2018].

[38] "Snap7," [Online]. Available: http://snap7.sourceforge.net/sharp7.html.