FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Cloud Platform for the Deployment of Online Data Analytics Application Oriented Services

Carlos Manuel Carvalho Boavista Samouco

DISSERTAÇÃO

Mestrado Integrado em Engenharia Informática e Computação

Orientador: João Paulo Trigueiros da Silva Cunha

31 de Julho de 2018

Cloud Platform for the Deployment of Online Data Analytics Application Oriented Services

Carlos Manuel Carvalho Boavista Samouco

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Pedro Alexandre Guimarães Lobo Ferreira Souto
Arguente: Rolando da Silva Martins
Vogal: João Paulo Trigueiros da Silva Cunha
31 de Julho de 2018

Resumo

Os dados recolhidos por dispositivos IoT não se tornam úteis sem serem analisados. Neste sentido, os sistemas de análise desempenham um papel importante pois permitem, a partir desses dados, a tomada de decisões mais informadas.

O propósito desta dissertação de mestrado é o estudo de uma solução para a implementação de um motor na cloud para análise de dados oriundos de sensores de dispositivos IoT. Este motor destina-se à integração dinâmica de aplicações de processamento que, após serem devidamente configuradas no servidor, são passíveis de ser invocadas remotamente através de uma interface programável. A utilização de técnicas de virtualização para o isolamento de aplicações e de mecanismos de comunicação entre processos, constituem algumas das estratégias-chave utilizadas na implementação deste sistema.

Dado que permitimos a execução de aplicações remotamente num servidor, é importante garantir que estas não interferem com o seu funcionamento. Assim optou-se pela utilização de contentores para isolar o seu ambiente de execução, dado estes corresponderem à solução de virtualização mais leve.

Com vista à aplicação deste motor para análise de dados num sistema IoT real, foi desenvolvida uma aplicação para análise de dados de batimento cardíaco recolhidos através de dispositivos biomédicos.

Com a criação do motor proposto, pretendemos alcançar uma maior modularidade nos sistemas de análise, permitindo aos investigadores focarem o seu trabalho no desenvolvimento dos algoritmos de processamento e abstraírem-se da forma como estes são integrados na *cloud*.

Abstract

The data collected by IoT devices is not useful without being analysed. Therefore, analytics systems play an important role, allowing the making of better and more informed decisions based on these data.

The purpose of this master's dissertation is the study of a solution for implementing an analytics engine in the cloud to assess data collected by sensors of IoT devices. This engine is set to allow the dynamic integration of processing applications. Once the analytics algorithms are properly configured in the server, they can be remotely invoked through a programmable application interface. The use of virtualization techniques to isolate applications and mechanisms of interprocess communication are some of the key strategies applied during the implementation of this system.

Since we allowed applications to run remotely on a server, it is important to ensure that they do not interfere with its behaviour. Considering this, we decided to use containers to isolate their execution environment, since these correspond to a lighter virtualization solution.

In order to apply this motor for data analysis in a real IoT environment, we developed an application for the analysis of heart beat data collected through biomedical devices.

With the development of such engine, we intend to achieve a greater modularity of the analytics systems, allowing researchers to focus their work on the development of analysis algorithms and abstract themselves from how they are integrated in the cloud.

Agradecimentos

Ao Professor João Paulo Cunha pela orientação desta dissertação. Agradeço a paciência que teve comigo, a liberdade que me deu, o incentivo e toda a ajuda prestada. A sua sabedoria fascina-me profundamente.

Aos colegas do INESC com quem trabalhei, pela entre-ajuda e disponibilidade.

A todos os meus amigos com quem cresci ao longo destes cinco anos. Obrigado por todos os mementos que passamos juntos. Vocês são pessoas incríveis.

Aos meus pais e irmãs pelo amor e preocupação constante, por estarem sempre ao meu lado e me apoiarem nos momentos mais difíceis da minha vida.

Aos meus avós, pelo amor que me deram. Admiro a vossa persistência e capacidade de viver.

Quero deixar ainda um agradecimento à Susana Fagundes por cuidar de mim e se preocupar tanto com o meu bem estar. Não tenho palavras para descrever o quão importante foste para mim neste último ano da minha vida.

Carlos Samouco

"... mas deixou-se levar pela sua convicção de que os seres humanos não nascem para sempre no dia em que as suas mães os dão à luz, mas que a vida os obriga uma e outra vez ainda a parirem-se a si mesmos."

Gabriel García Márquez

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

ECG	Eletrocardiograma
HRV	Heart Rate Variability
IoT	Internet of Things
API	Application Programming Interface
JSON	JavaScript Object Notation
REST	Representational State Transfer
NoSQL	Not Only SQL
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
I/O	Input/Output
argv	Argument Vector
stdin	Standard Input Stream
stdout	Standard Output Stream
stderr	Standard Error Stream
cwd	Current Working Directory
ssh	Secure Shell

Capítulo 1

Introdução

Este capítulo é destinado à apresentação do contexto, motivação e objetivos desta dissertação de mestrado. Descreve-se ainda a estrutura deste documento.

1.1 Contexto/Enquadramento

Este trabalho foi concebido no contexto do projeto *VR2Market* apoiado pelo instituto Carnegie Mellon Portugal e desenvolvido por um consórcio liderado pelo INESC TEC que envolve a universidade norte-americana Carnegie Mellon, o Instituto das Telecomunicações e a Universidade de Aveiro.

O INESC TEC é um instituto de investigação e desenvolvimento, sem fins lucrativos, nas áreas de computação, tecnologia e ciência com a sua sede no campus da faculdade de engenharia da universidade do Porto. Este instituto tem como principal premissa o estudo de novas técnicas em diversas áreas da ciência e tecnologia para que possa transferir os novos conhecimentos para a indústria, serviços e administração pública [TEC18].

O grupo de investigação em engenharia biomédica, BRAIN/C - BER, envolvido no projeto *VR2Market* do INESC TEC tem vindo a desenvolver um sistema de monitorização e análise de dados fisiológicos para a detecção de níveis de stress e fadiga em profissionais de risco, como bombeiros e policiais, entre outros.

Este sistema de monitorização baseia-se na recolha de dados fisiológicos e ambientais, como recurso a dispositivos wearable biomédicos. Os dados recolhidos são transmitidos a uma aplicação mobile por bluetooth onde são pré-processados e enviados, posteriormente, a um servidor que gere e armazena a informação recolhida.

Todavia, deparando-se com um a necessidade de realizar uma análise dos dados fisiológicos previamente recolhidos, procurou-se desenvolver uma ferramenta que viesse dar uma resposta, permitindo automatizar o processamento da informação.

Figura 1.1: Logotipo do INESC TEC

1.2 Motivação

A análise da variabilidade da frequência cardíaca, possibilitada pela realização de ECGs, tem demonstrado possuir um papel muito importante na detecção e prevenção de diversos problemas de saúde, de entre os quais, aqueles que estão relacionado com o sistema nervoso autónomo (composto pelo sistema nervoso simpático e parassimpático), uma vez que, em situações de stress o sistema nervoso simpático ativa-se, somatizando através do aumento da frequência cardíaca (entre outras sintomatologias). [SC15].

Uma vez que a varabilidade do batimento cardíaco é um fator importante a ter em consideração no desenho de sistemas de monitorização de saúde remotamente assistida, e dados os objetivos do projeto da equipa de investigação, foram desenvolvidos uma série de algoritmos para o cálculo e análise de diversos parâmetros relacionadas com a frequência cardíaca.

Assim, com o intuito de utilizar essas aplicações para processamento online de ECGs recolhidos a partir de dispositivos wearable ambulatórios, optou-se pela criação de uma plataforma destinada à integração de aplicações para análise de dados.

Apesar da presente dissertação de mestrado ter surgido da necessidade emergente de se desenvolver uma ferramenta que agregasse as diferentes aplicações de análise relativos com a variabilidade cardíaca, pretendemos que o sistema não se restrinja a este problema, e possa ser aplicado em outro tipo de soluções relacionadas com o processamento de dados em sistemas IoT.

1.3 Objetivos

Esta dissertação tem como objetivo o estudo e desenho de um motor de análise, centrado na disponibilização de aplicações online para efetuarem o processamento de informação recolhida por fontes externas. O motor de análise será disponibilizado sob a forma de um serviço web para possibilitar a automatização do processamento dos dados e a sua integração com outros sistemas.

O motor proposto procura assim permitir a instalação de diversos tipos de aplicações destinadas ao processamento de informação, sendo que este será também responsável por gerir a sua execução e o fluxo de dados de entrada e saída. Os dados processados poderão depois ser visualizados mediante uma aplicação cliente do motor de análise com uma gráfica com o utilizador.

O principal enfoque desta dissertação passa, inicialmente, pelo estudo dos mecanismos que viabilizam a invocação remota de aplicações e culmina no desenho e implementação do sistema de *analytics* referido e de uma aplicação cliente que demonstra a sua funcionalidade e performance.

1.4 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais 5 capítulos. No capítulo 2, é apresentado o estado da arte, onde constam as tecnologias e os trabalhos relacionados com os objetivos desta dissertação. No capítulo 3, é analisado o problema e apresentadas as decisões de alto nível tomadas. No capítulo 4, é explicado o processo de implementação do sistema e as decisões de baixo nível tomadas. No capítulo 5, é feita uma análise relativa ao desempenho do sistema e resultados da sua aplicação num problema real. No capítulo 6, são apresentadas as conclusões derivadas da implementação e teste do sistema, discutindo-se formas de o melhorar em iterações futuras.

Capítulo 2

Revisão Bibliográfica

Este capítulo visa apresentar uma revisão da literatura a respeito de questões relacionadas com a importância de sistemas na área da *Internet of Things* (IoT).

É apresentada uma visão geral relativa aos tipos de sistemas de análise de dados, a sua arquitetura e aplicações destes sistemas em problemas reais.

Dado que se pretende desenvolver um sistema baseado na publicação e partilha de aplicações de análise e processamento de dados, estudou-se alguns sistemas relacionados com este tipo de abordagem, por forma a encontrarmos as principais normas arquiteturais para implementação do nosso motor de análise.

Referimos ainda a importância da virtualização como método de isolamento do ambiente de execução das aplicações, sendo efetuada uma apresentação das tecnologias existentes com vista à sua utilização para implementação do nosso motor de análise.

2.1 Análise de Dados em IoT

Os dados recolhidos através de dispositivos IoT não se tornarão úteis sem serem analisados. O aumento do volume de dados recolhidos por dispositivos IoT tem levado à evolução de sistemas de análise de big data, sendo que tais sistemas desempenham um importante papel para permitir que associações possam ter um melhor entendimento sobre os dados e assim auxiliar na tomada de decisões melhores e mais informadas [Gol15].

2.1.1 Tipos de sistemas de análise

Segundo evidenciado em [MNG+17], dependendo do tipo de sistema IoT, a análise de dados pode ser categorizada de acordo com os seguintes tipos:

- **Análise em tempo real:** Aplica-se normalmente a sistemas de sensores. Os dados são recolhidos e processados em períodos pequenos.

- **Análise offline:** Aplica-se quando não é necessário providenciar uma resposta imediata em relação aos dados recolhidos.

- **Análise ao nível de memória:** Utiliza-se quando o volume de dados é compatível com o seu armazenamento numa base de dados convencional.

- **Análise Massiva:** Adota-se quando o volume de dados é superior ao de armazenamento suportado por bases de dados convencionais. A informação sofre operações de mapeamento e redução para poder ser armazenada.

2.1.2 Arquitetura IoT para análise de big data

A arquitetura de um sistema IoT pode ter vários domínios conforme a área de aplicação do sistema. No entanto, quando se trata de análise de grandes quantidades de dados, é possível reconhecer padrões na arquitetura destes sistema. Em [MNG+17], é apresentado um modelo onde os sensores e objetos associados, efetuam a de transmissão de dados por meio de um rede interna sem fios. Os dados passam por sua vez, através de uma ponte de ligação com a internet ou outra rede para serem armazenados na cloud. Essa informação é então finalmente processada e analisada por meio de um motor de análise. O sistema pode ainda ser integrado com uma API de gestão dos dados da plataforma e uma interface para análise e visualização da informação recolhida. É apresentado na Figura 2.1 a arquitetura do sistema descrito.

2.1.2.1 Machine-to-Machine

Machine-to-Machine (M2M) consiste num tipo de comunicação distribuída entre duas ou mais entidades heterogêneas de forma assíncrona [MGMTG15].

As soluções oferecidas pela M2M operam no interior de uma rede colaborativa e interoperável que permite a transmissão dos dados, que foram previamente recolhidos por dispositivos, para um *back-end*. Estas soluções de processamento comportam diferentes etapas de inferência, envolvendo várias frentes de recolha de dados, além de constituir a base para a *Internet of Things* (IoT) [CJS12].

A arquitetura distribuída da M2M caracteriza-se por uma complexidade cada vez maior e uma heterogeneidade, o que impõe uma gestão que exige um nível de abstração adequado, para que se possa proceder à unificação dos recursos e das capacidades, no sentido de se obter um sistema global mais homogéneo e interoperável [CJS12].

Desta forma, vêm possibilitar a incorporação de todos os recursos IoT, em sistemas orientados a serviços, mediante plataformas de integração, para potencializar o seu funcionamento conjunto [CJS12]. Neste sentido, cabe às plataformas M2M gerir os desafios emergentes, explorar os recursos desses objetos e permitir a colaboração entre as diferentes partes do sistema.

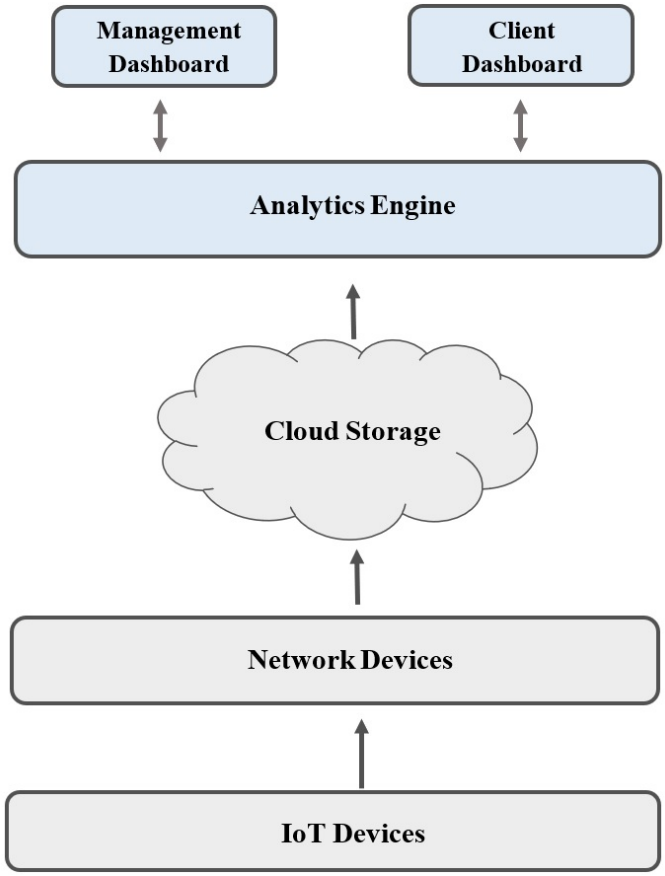

Figura 2.1: Arquitetura de um sistema IoT para análise de Big Data sendo o modelo apresentado em [MNG+17]

2.1.3 Principais Áreas de Aplicação de Sistemas de Análise em IoT

Através da pesquisa levada a cabo em [MNG+17], conclui-se que as principais áreas de aplicação de sistemas de análise de dados provenientes de dispositivos IoT são:

- **Agricultura e Industria:** Assim a utilização de IoT nestas áreas, visa essencialmente controlar e optimizar a produção para obtenção de maior retorno. Na agricultura, através da monitorização de dados ambientais e do solo, os sistemas IoT vêm ajudar a garantir que as colheitas possuem as condições mais favoráveis para o seu desenvolvimento. Por sua vez, na industria estes sistemas são usados para monitorizar produtos e equipamento ao longo de uma linha de produção por forma a aumentar a eficiência operacional e reduzir custos.
- **Cidades Inteligentes:** Este tipo de aplicação, visa a tomada de decisões que permitam uma utilização mais eficiente dos recursos e serviços de uma cidade, através da recolha de dados relativos ao grau de utilização desses serviços. Este tipo de sistemas podem passar, por exemplo, pela monitorização da utilização de uma rede elétrica para permitir o fornecimento mais eficiente de energia e pela monitorização do transito para escolher as melhores rotas de viagem [hKRM17].

- **Retalho e Logística:** Em termos de logística, os dispositivos IoT podem ser usados para acompanhar a localização e quantidade de mercadoria além de permitir também, por exemplo, analisar o volume de vendas e o comportamento de clientes numa loja para prever tendências de compra e procura de bens [DJ16].

- **Prestação de Cuidados de Saúde:** Este tipo de sistemas são usados para monitorizar a condição física de pacientes, por forma a permitir a detecção precoce de problemas de saúde e assim melhorar a qualidade dos cuidados de saúde prestados. Geralmente, estes sistemas tem por base a recolha e análise continua de grandes volumes de dados provenientes de eletrocardiogramas, de medidores de temperatura ou de monitores do nível de glicose no sangue.

2.2 Trabalho relacionado

Actinium [KLD12], trata-se de uma plataforma destinada à instalação e execução de aplicações baseadas em JavaScript que permitem recolher e processar dados de dispositivos IoT. O grande propósito deste sistema consistem em transferir os módulos de processamento e lógica de atuação para fora do ambiente dos dispositivos embebidos, sendo estes controlados e executados por um servidor com maior capacidade de processamento.

A comunicação com os dispositivos IoT é inteiramente gerida pelo servidor, sendo que a troca de dados com esses dispositivos é desencadeada por uma série de instruções definidas nas aplicações em execução.

Este sistema é essencialmente orientado para a instalação, remoção, execução e monitorização de aplicações de forma dinâmica por meio de uma interação RESTful. As aplicações podem ainda ser descarregadas e partilhadas entre utilizadores.

Uma das preocupações, em termos arquiteturais, deste sistema passou pela implementação de métodos de segurança que permitem o isolamento das aplicações em execução por meio de uma sandbox de JavaScript. O isolamento das aplicações é apontado como um fator importante para garantir a confiabilidade do servidor, de forma a que aplicações não possam influenciar o comportamento de outras aplicações.

Em [MCK+09], é apresentado um sistema que foi desenvolvido com intuito de disponibilizar um serviço interativo de aplicações sob a internet. O principal objectivo era permitir que investigadores tivessem acesso a um conjunto de aplicações de análise numa plataforma online, podendo essas aplicações possuir uma interface gráfica com o utilizador.

A implementação deste sistema passou pelo estudo de uma infraestrutura na cloud que permitisse a disponibilização do serviço de computação sob demanda, por meio de uma abordagem distribuída. Na gestão da execução das aplicações foram utilizadas máquinas virtuais hipervisionadas.

As aplicações são instaladas e configuradas numa imagem de uma máquina virtual. Para executar uma determinada aplicação, a máquina virtual correspondente a essa aplicação é inicializada, e através do browser, pode-se interagir com esta por meio de um visualizador. A comunicação com a máquina virtual em questão tem por base uma ligação SSH.

2.3 REST

REST (*Representational State Transfer*) corresponde a um estilo de arquitectara que define um conjunto de princípios com base no protocolo HTTP. Os serviços web implementados de acordo com estes estilo de arquitetura são designados de RESTfull.

Como descrito em [TCP+12], os princípios REST são os seguintes:

- Devem se utilizar URIs para identificar recursos expostos pelo servidor.
- Deve ser utilizado um conjunto de operações uniforme. Em HTTP essas operações são mapeadas sob nomes POST, GET, PUT e DELETE.
- As mensagens devem ser auto-descritivas e conter toda a informação necessária para o seu processamento.
- O servidor não deve guardar nenhuma informação sobre o estado do cliente entre pedidos.

No serviço web RESTful, os pedidos direcionados a um determinado URI de um recurso, vão gerar uma resposta que pode ser nos formatos HTML, XML, JSON ou outro formato.

Num estudo efetuado em [NLB18], que analisou cerca de 4000 APIs de serviços web, revelou que, na troca de mensagens, o formato JSON é significativamente mais usado em relação ao XML. Assim, dada a sua versatilidade e elevada taxa de utilização, a API de análise aqui desenvolvida teve por base a troca de mensagens nesse formato, por forma a ser compatível com o maior número de sistemas.

2.4 Virtualização

A virtualização dos recursos é um dos conceitos chave em Cloud Computing. A virtualização pode ser vista como a divisão de uma máquina física em diversas máquinas virtuais. Existem essencialmente dois tipos de tecnologias de virtualização: hipervisionada e baseada em contentores.

A virtualização hipervisionada requere um software hipervisor executado sob um sistema operativo, que abstrai a interação das máquinas virtuais (VMs) do hardware da máquina hospedeira. Neste tipo de virtualização, cada VM possui o seu próprio sistema operativo que se encontra isolado do sistema hospedeiro, sendo que é possível executar múltiplos sistemas operativos numa única máquina física.

Para além da virtualização hipervisionada, existe ainda outro tipo de virtualização mais recente baseada em contentores. Este tipo de virtualização ocorre ao nível do próprio sistema operativo

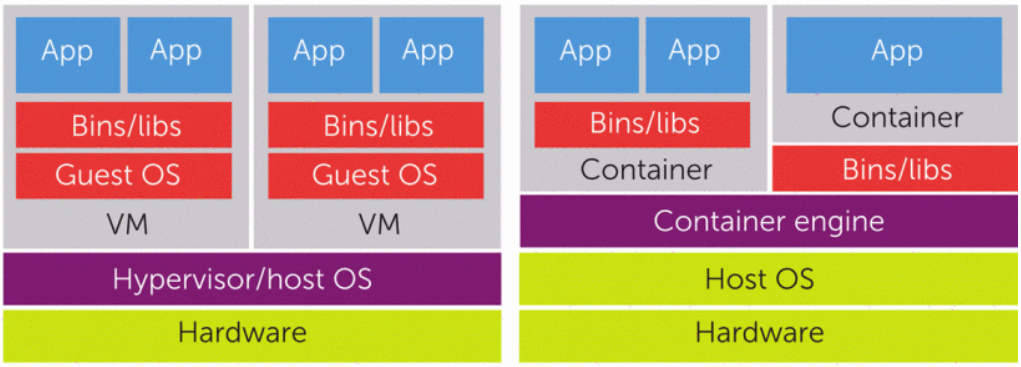

Figura 2.2: Arquiteturas de virtualização [Pah15]. Na imagem da esquerda encontra-se representada a arquitetura da virtualização hipervisionada, e na imagem direita a arquitetura da virtualização baseada em contentores.

e consiste na divisão dos recursos de uma máquina física para criar múltiplas instâncias isoladas a nível do espaço do utilizador [CMF+16]. Um contentor é formado por um sistema de ficheiros próprio e pelas bibliotecas ou binários necessários para executar uma determinada aplicação ou conjunto de aplicações [Pah15].

Enquanto a virtualização hipervisionada fornece uma abstração aos sistemas operativos convidados, a virtualização de contentores fornece uma abstração a nível das chamadas de sistema feitas pelos processos que são executados nas instancias virtualizadas. Estas diferenças entre a virtualização hipervisonada e de contentores são retratadas na figura 2.2.

2.4.1 Virtualização de Contentores

A virtualização baseada em contentores é suportada pelo kernel do Linux, sendo denominada Linux Container Virtualization (LCV). Esta tecnologia permite executar múltiplas instancias do mesmo sistema operativo ao nível do utilizador, através da partilha do kernel do sistema operativo hospedeiro.

Dado que um contentor se comporta como um sistema operativo, este possuí o seu próprio sistema de ficheiros, processos e interfaces de rede. Comparando este tipo de virtualização com o uso de virtualização hipervisionada, e de acordo com as informações apresentadas em [CMF+16], podemos destacar as seguintes métricas:

- **Tempo de Inicialização:** Em geral, os contentores por serem mais leves, possuem um tempo de inicialização menor quando comparado com as VMs hipervisionadas.

- **Controlo da Execução:** Visto que os contentores fazem parte do mesmo sistema operativo, torna-se, significativamente mais fácil iniciar ou terminar aplicações dentro de um contentor do que em relação as VMs hipervisionadas.

- **Consumo de recursos:** Os contentores utilizam, consideravelmente, menos recurso, uma vez que partilham o kernel de sistema ou outros recursos do espaço do utilizador definidos. Já as VMs, por possuírem um kernel independente, assumem um peso computacional maior.

- **Isolamento:** Dada a partilha de recursos com o sistema hospedeiro, a utilização de contentores não fornece um isolamento total do sistema. Por outro lado, as VMs garantem um isolamento efetivo do sistema.

A utilização de contentores proporciona uma solução mais leve e eficiente para a implementação de aplicações isoladas, quando comparada com uma solução baseada na virtualização hipervisionada, já que a partilha do kernel do sistema operativo permite uma redução tanto da utilização de RAM como de CPU. No entanto, a virtualização com contentores oferece um grau de isolamento menor quando comparada com a virtualização hipervisionada [CMF+16]. A utilização de contentores para o isolamento de aplicações requere maiores cuidados em termos de segurança na sua utilização, devido ao facto de se correr o risco que partes sensíveis do sistema operativo hospedeiro sejam expostas, dentro dos contentores, comprometendo a sua segurança.

Dado que os contentores constituem uma solução de virtualização mais leve, permitem que num contexto de *cloud computing* possam ser utilizados em maior número, quando comparados com o uso de VMs tradicionais numa mesma máquina física.

2.5 Tecnologias

2.5.1 Docker

Docker é um motor de virtualização baseada em Linux Containers (LXC) que permite orquestrar a construção de imagens e criação de contentores, sendo estes usados para encapsular aplicações e as suas dependências num ambiente virtualizado [Ber14].

Uma imagem corresponde a um objeto imutável que combina um conjunto de ficheiros e diretórios do qual depende uma aplicação ou conjunto de aplicações para serem executadas. Um contentor, por sua vez, é inicializado a partir de uma determinada imagem, e executará um conjunto de aplicações. Os ficheiros de uma imagem que pretendem ser modificados por um contentor são copiados e instanciados por forma a ser mantida a integridade da imagem, dado que esta pode ser usada por mais que um contentor em simultâneo. Esta abordagem designada copy-on-write permite diminuir a utilização do disco requerida pelos contentores [Inca].

Segurança

Quando inicializamos um contentor com o Docker, este encarrega-se de definir um conjunto de *namespaces* e *countrol groups* [Incb]).

O uso de *namespaces* corresponde ao método principal de isolamento. Os *namespaces* permitem criar abstrações de recursos de sistema, sendo que processos pertencentes a um determinado *namespaceiop* apenas conseguem aceder a uma instancia de um recurso global. Essencialmente, a

sua utilização permite que os processos que são executadas dentro de um contentor não consigam ver nem aceder a outros processos que são executados noutro contentor ou no próprio sistema principal [Incb].

O uso de *control groups* é também essencial na criação de um contentor. Eles permitem limitar os recursos computacionais atribuídos a um contentor. Assim, por forma a que um contentor não possa exaustar os recursos disponíveis e deitar o sistema abaixo, este apenas recebe uma porção de memória, CPU e I/O de disco que seja suficiente para executar as tarefas destinadas e assim impedir ataques de negação de serviço (*denial-of-service*) [Incb, MKPG16].

Devemos ainda ter em atenção que um contentor Docker não deverá ser executado como utilizador privilegiado, por forma a que os processos que são executados num dado contentor não possam modificar as configurações de sistema e conseguirem libertar-se das limitações impostas ao contentor [Incb].

Dado que também é possível montar volumes num contentor, devemos ter cuidado para não partilhar diretórios que contenham informação sensível.

Assim, de modo a endurecer a segurança num contentor, podem ser utilizadas outras tecnologias em conjunto com o Docker. Essas tecnologias, tais como AppArmour [Wik], SELinux [Pro], GRSEC [grs], permitem limitar o acesso ou a realização de um conjunto de operações sobre recursos sensíveis de sistema.

2.5.2 Node.Js

Node.Js [Nod], é um JavaScript runetime que tem como base o motor JavaScript V8 da Google. Este fornece uma arquitetura assincronia orientada a eventos e uma API de I/O não bloqueante, o que facilita a criação de sistemas escaláveis. O Node é *opensource* e suporta múltiplas plataformas.

Apesar de o Node não ser desenhado para usar *threads*, este disponibiliza uma API que permite tirar partido de multi-core quando necessário pela criação de processos individuais ou *clusters*.

Além disso, o Node possuí uma biblioteca que permite a gestão de comunicações HTTP e HTTPS, fornece módulos que permitem interagir diretamente com o sistema de ficheiros do sistema operativo, entre outras funcionalidades.

A API disponibilizada pelo Node é desta forma útil para a criação de serviços Web que podem integrados com sistemas IoT.

2.5.3 MongoDB

MongoDB [Mon] é um software de base de dados opensource e multiplataforma, orientada ao armazenamento de dados sob a forma de documentos. É classificada como uma base de dados não relacional (NoSql),uma vez que não segue o modelo relacional que é fornecido pelas bases de dados tradicionais.

Os documentos armazenados por esta base de dados possuem um formato parecido com JSON, o que permite que os campos de um ficheiro possam ser acedidos individualmente. Este modelo permite mapear facilmente os objetos do código de uma aplicação com os documentos da base de

dados, o que simplifica a sua implementação. A sua utilização enquadra-se assim no ambiente de programação JavaScript proporcionado pelo Node.js

O MongoDB tem como base uma arquitetura distribuída integrada que lhe proporcionam características como alta disponibilidade, escalabilidade horizontal e distribuição geográfica, prontas a serem utilizadas.

2.5.4 Nginx

Nginx (engine-x) [ngi], é um servidor proxy reverso open source para os protocolos HTTP, HTTPS, SMTP, POP3 e IMAP. Além disso trata-se também de um *load balancer*, permite ainda servir conteúdo web com servidor de origem e é capaz de suportar armazenamento em cache como servidor proxy de HTTP. O Nginx foi desenhado tendo em vista a baixa utilização de memória e alta performance.

2.6 Sumário

Em virtude da pretensão de aplicar o motor de análise em conjunto com sistemas IoT, foram analisadas as principais áreas de aplicação e a arquitetura usada por estes sistemas, com o proposito de enquadrar o motor proposto neste trabalho.

Após revisão de alguns trabalhos relacionados com a disponibilização de aplicações na *cloud*, podemos concluir que estas passam por um processo de instalação e configuração antes de poderem ser utilizadas no processamento de dados. O isolamento das aplicações é tido como uma condição importante para a não interferência durante a sua execução.

Tanto quanto é do conhecimento dos autores, a literatura sobre este teme é limitada. Com efeito, foram apenas encontrados dois trabalhos sobre o tema na pesquisa bibliográfica. Assim, espera-se que este trabalho contribua para a consolidação dos conhecimentos nesta área.

Com vista à execução de aplicações na *cloud*, foram analisadas e comparadas as técnicas de virtualização hipervisionada e baseada em contentores. Uma vez que procuramos permitir o processamento de grandes quantidades de dados, a utilização eficiente dos recursos disponíveis torna-se um fator bastante importante para permitir a escalabilidade do sistema. Assim conclui-se que a utilização de contentores constituí um mecanismo leve e capaz de permitir a execução de aplicações em ambiente isolado. Foram ainda analisado as vantagens do Docker como ferramenta de gestão de contentores e as garantias de segurança oferecidas por este com objetivando a sua utilização para a execução de aplicações não confiáveis.

Foi efetuada ainda uma revisão sobre algumas tecnologias tendo em vista a sua utilização para o desenvolvimento do projeto proposto.

Capítulo 3

Apresentação do problema

Neste capítulo será analisado o problema que é alvo de estudo nesta dissertação, o que inclui a definição dos principais objetivos do trabalho, a apresentação da abordagem ao problema e guias gerais sobre a forma como será implementado.

3.1 Visão geral do problema

Para dar resposta ao problema apresentado, o sistema de análise aqui proposto é composto por um servidor *cloud* responsável pelo processamento dos dados e por uma aplicação cliente usada na configuração do motor de análise e na visualização da informação.

Pretende-se também a criação de uma API com o objectivo de que outros sistemas possam, de forma automatizada, efetuar o processamento dos dados que recolheram. No entanto, nem sempre os dados que se tenciona processar estão disponíveis nesses sistemas. É possível configurar para cada parâmetro de uma aplicação um *endpoint* ao qual pode ser enviado um pedido para permitir a recolha de dados externos.

Os principais requisitos do sistema proposto nesta dissertação são os seguintes:

- **Servidor *Cloud* de análise**
  - Permitir a adição de novas aplicações de processamento dinâmico.
  - Fornecer um ambiente propício à execução de aplicações independentemente da linguagem em que foram desenvolvidas.
  - Gerir a execução e fluxo de dados de uma aplicação.
  - Fornecer uma API com vista a possibilitar o processamento de dados a partir de fontes externas.
- **Aplicação cliente**
  - Instalar aplicações e configurar os seus parâmetros de entrada e *output*.

- Configurar *endpoints* para recolha de dados externos.
- Listar as aplicações e endpoints instalados.
- Invocar remotamente as aplicações solicitadas.
- Visualizar os dados resultantes de um processamento.

3.1.1 Casos de Uso

O motor de análise visa ser operado tanto por um utilizador, mediante uma interface gráfica, como também por sistemas externos através de um API. O utilizador terá capacidade para interagir com todas as funcionalidades e customizações oferecidas pelo servidor. Cabe ao utilizador a instalação de aplicações de modo a que outros sistemas possam recorrer aos serviços de processamento oferecidos remotamente.

A figura 3.1 explicita detalhada e esquematicamente as interações efetuadas no sistema por parte dos seus utilizadores, tendo por base os requisitos descritos anteriormente.

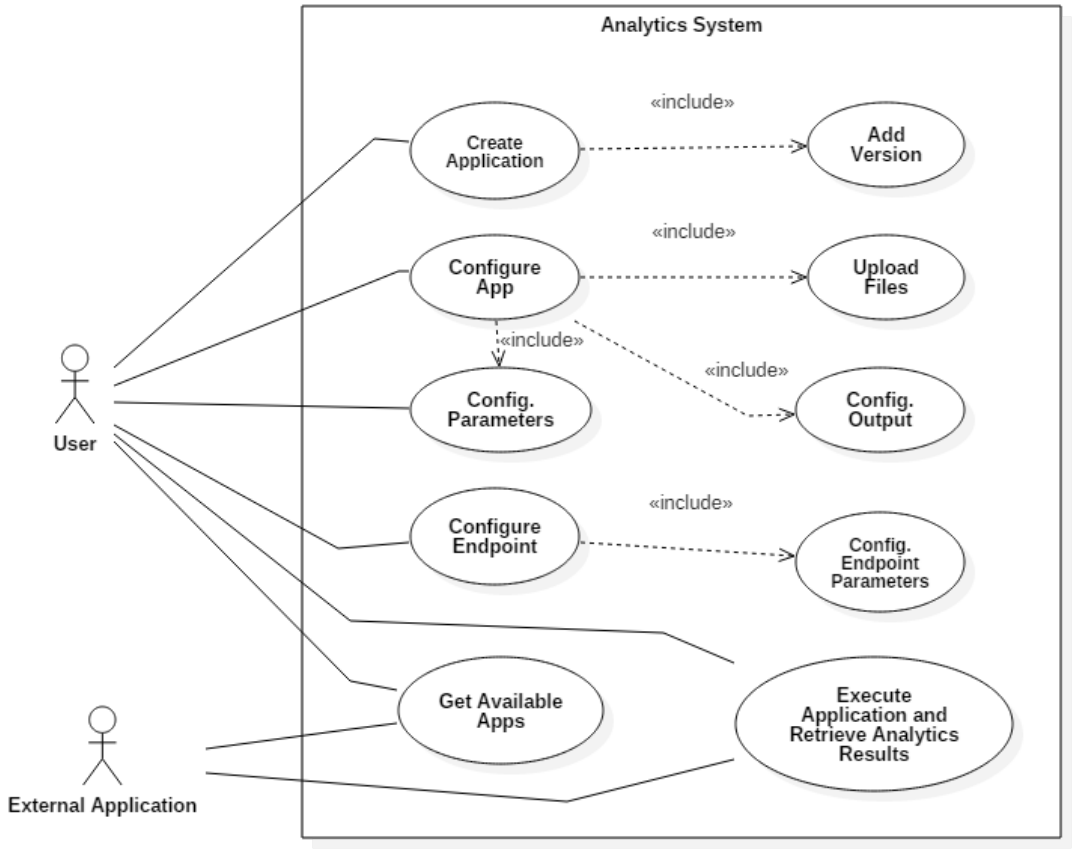

Figura 3.1: Diagrama de casos de uso do sistema. Presenta os principais requisitos do sistema e os atores envolvidos.

3.1.2 Aplicações do motor de análise

Esta ferramenta foi primariamente desenhada com vista ao processamento e análise de dados fisiológicos recolhidos a partir de um sistema IoT de monitorização contínua, tal como foi referido no capítulo introdutório.

Apesar desta ferramenta ter como principio base o processamento de dados produzidos por dispositivos IoT, a sua aplicação não é limitada a esta área. Através desta ferramenta, é possível a publicação de uma grande variedade de aplicações na *cloud*, tal como a criação serviços web com as mais diversas finalidades de forma rápida.

3.2 Método de execução das aplicações

Procurando-se instalar aplicações num servidor que permita sua invocação remota, torna-se basilar perceber que características devem ter as aplicações para poderem ser instaladas e executadas dinamicamente num servidor.

As aplicações são executadas sem intervenção direta do cliente e uma vez que estamos somente interessados no resultado do processamento dos dados, as aplicações não necessitam de interface gráfica, e podem implementar o mínimo necessário para obterem a funcionalidade pretendida. Por outro lado, as aplicações receberão todos os dados necessários ao seu processamento aquando do início da sua execução, pelo que não deverão bloquear para operações de I/O.

Argumentos de um Programa

Para executarmos as aplicações, teremos de criar um processo separado, passar-lhe os dados que queremos que sejam processados e esperar que este termine para lermos o seu *output*. Os dados de *input* são enviados no vetor de argumentos do programa (*argv*), cabendo a cada aplicação validar e processar os dados que recebe. O *argv* corresponde a um vetor cujos elementos deste correspondem a cada uma das *strings* de uma tipica invocação do programa pela linha de comandos [Libb]. Assim, é da responsabilidade do servidor fornecer os dados na ordem esperada pela aplicação, isto implica que os parâmetros de entrada sejam previamente configurados.

Apesar de o vetor de argumentos de cada aplicação ser um vetor de *strings*, o *input* das aplicações não se limita a dados simples (*e.g.* *strings* e números). A aplicação pode necessitar, por exemplo, de ler uma série de dados provenientes de um ficheiro, e nesse caso, o *path* para o ficheiro deverá ser passado no vetor de argumentos do processo. O servidor deverá encarregar-se de gerir, também, os ficheiros de entrada que serão utilizados pela aplicação, e fornecer-lhe o caminho para esses ficheiros.

Output de um Programa

Em relação aos dados produzidos após execução da aplicação, o servidor deverá ser o responsável por recolher esses dados e devolvê-los ao cliente. Os dados produzidos pelas aplicações

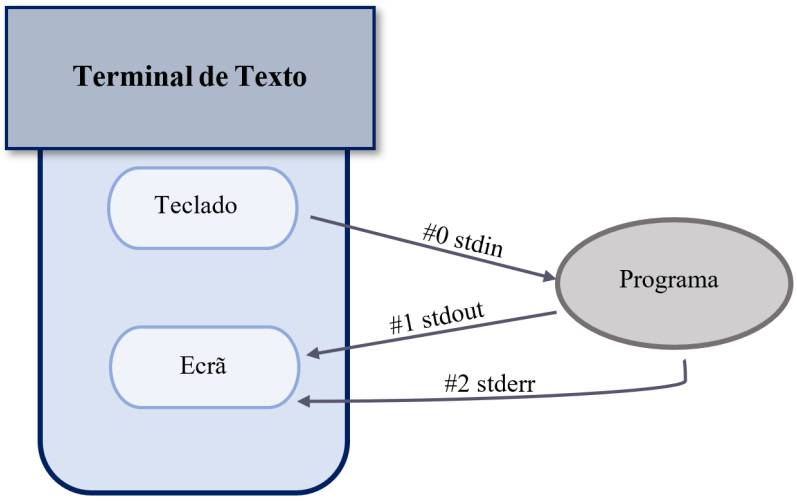

Figura 3.2: Os descritores de ficheiro para entrada, saída e erro.

podem ser de dois tipos: ficheiros criados pela aplicação ou dados enviados para as *standard streams* do processo.

Um processo possuí três tipos de *standard streams*: *standard input* (stdin), *standard output* (stdout) e *standard error* (stderr) [Libd]. Estes *streams* são representados por um descritor de ficheiro numerado, sendo que o stdin possuí valor 0, o stdout valor 1 e o stderr valor 2 [Liba]. A figura 3.2 representa esses descritores face a um terminal de texto. Uma vez que, o valor destes descritores de ficheiros é fixo, torna-se possível redirecionar os dados destes *streams* para um ficheiro ou para um processo através de um *pipe*. Posto isto, visto que estamos interessados nos dados de *output* das aplicações, as *streams* de *output* e de erro de um processo serão redirecionados para o domínio do servidor como forma de este interceptar os dados produzidos pelas aplicações e enviar esses dados ao cliente.

Em relação ao *output* de dados em formato ficheiro, surgem algumas dificuldades na forma como interceptar as chamadas ao sistema de escrita e redirecionar os dados para outro processo. Dado não ser possível prever que ficheiros vão ser usados pelo processo, nem os descritores atribuídos a esses ficheiros pelo sistema operativo na sua abertura [Liba], não existe uma maneira eficaz que permita redirecionar as chamadas de escrita para o domínio do servidor. Consequentemente, para contrariar esta dificuldade, algumas restrições têm de ser impostas para que consigamos saber que ficheiros foram criados pela aplicação e ler o seu conteúdo. Por conseguinte, os ficheiros de *output* de uma aplicação terão de ser criados numa pasta especial gerida pelo servidor, para este poder facilmente determinar a sua localização e recolher a informação neles contida.

Em suma, as aplicações suportadas pelo motor de análise terão um ambiente de execução semelhante ao proporcionado pela linha de comandos, sendo que serão suportados executáveis compatíveis com o sistema operativo sobre o qual o servidor é executado, ou outras aplicações instaladas no sistema presentes na variável de ambiente *PATH* [Libc], tal com Node ou Python.

3.2.1 Considerações de segurança do sistema

O motor de análise permite a publicação de aplicações numa plataforma online, e por esse motivo, é importante conseguirmos garantir que essas aplicações são executadas isoladamente para que não interfiram umas com as outras, nem com o próprio servidor que as controla, de forma a que partes sensíveis do sistema não sejam expostas e por conseguinte, o seu funcionamento não seja comprometido.

O grau de isolamento concedido por uma máquina virtual assegura que as aplicações e serviços que serão executados dentro de uma máquina virtual não possam interferir com sistema operacional original e nem em outras máquinas virtuais [Pah15].

A virtualização do sistema, onde são executadas as aplicações, é desta forma essencial na construção deste motor, uma vez que permite a criação de um modo de execução seguro, que permite salvaguardar a integridade do sistema. Dado que os contentores constituem um método de virtualização leve, a sua utilização para esse fim, possibilita a optimização dos recursos computacionais disponíveis.

Para além disso, a segurança do sistema pode ser ainda melhorada caso restrinjamos o acesso aos recursos disponibilizados através de autenticação prévia dos utilizadores do sistema, sendo que a comunicação entre cliente e servidor deverá ser efetuada mediante canais de comunicação seguros, nomeadamente pela utilização do protocolo HTTP sobre TLS/SSL — HTTPS.

3.3 Conclusões

O motor de análise proposto visa a publicação de aplicações na *cloud* que podem mais tarde ser executadas para o processamento de dados, por parte de um utilizador ou entidade externa, via uma API fornecida.

As aplicações que temos em vista serem suportadas pelo sistema de análise deverão poder ser compatíveis com a passagem de argumentos proporcionada pela linha de comandos.

Devido à generalização que se pretende obter em relação às aplicações suportadas para instalação no servidor, algumas condições devem ser tidas em consideração durante a fase de implementação destas, nomeadamente em relação à localização onde são criados os ficheiros de *output* das aplicações. No entanto consideramos que esta restrição não traz grande impacto sob a lógica da execução das aplicações.

Dada que este sistema concede liberdade de executar aplicações remotamente, a virtualização do seu ambiente de execução constitui um ponto crucial para que estas não possam prejudicar o sistema.

Capítulo 4

Implementação do Motor de Análise

Este capítulo destina-se à apresentação de detalhes relativos à implementação do projeto apresentado.

Inicialmente apresenta-se a arquitetura e os componentes do sistema, seguidamente são exibidos os dados requiridos na configuração e instalação de uma aplicação, logo após é explicado o modo como as aplicações são executadas e por último, é apresentada a lógica de funcionamento das aplicações em modo seguro.

4.1 Arquitetura do Sistema

Como mencionado no capítulo 3, este sistema proposto é composto por duas aplicações distintas: um servidor que fornece uma API *RESTful* destinada ao processamento de dados e uma aplicação configuração que serve de interface com o servidor e que permite a instalação de aplicações e a sua invocação remota.

O servidor é o componente principal deste sistema, sendo que a tecnologia escolhida para o seu desenvolvimento foi o Node.js. O Node.js trata-se de um JavaScript *Runtime*, de código livre, compatível com múltiplas plataformas (Windows, Linux, Unix, Mac OS, Android). Esta tecnologia foi escolhida pois, para além de ser muito eficiente na utilização de recursos computacionais, possui uma extensa biblioteca para comunicação entre processos, constituindo desta forma numa ferramenta muito útil durante a implementação da solução proposta para este problema.

O servidor é o gestor de toda a informação do sistema relativa às configurações de aplicações e *endpoints*. Para garantir a persistência desses dados, o servidor comunica com uma base de dados MongoDB para armazenamento e posterior consulta dos mesmos. O MongoDB é uma base de dados NoSQL orientada a documentos que se assemelham a json. Isto permite que a representação interna dos dados no servidor possua uma semântica idêntica à utilizada pela base de dados.

O servidor não fornece qualquer tipo de interface gráfica com o utilizador, uma vez que foi desenhado para efetuar a gestão de dados e do ambiente de execução das aplicações. Em vez disso,

as suas funcionalidades são expostas mediante uma aplicação de configuração desenvolvida em Angular, que fornece um ambiente gráfico ao utilizador. Desta forma, com o desacoplamento das duas aplicações, permitimos separar as responsabilidades de visualização e gestão das aplicações, das responsabilidades de processamento. Além disso, dado que o servidor de análise tem em vista disponibilizar uma API para ser usado por aplicações terceiras, não é relevante as funcionalidades de interface com o utilizador estarem incluídas neste. No entanto, as duas aplicações não fazem sentido separadas, uma vez que servem um objetivo comum.

Estas aplicações, incluindo a base de dados, são executadas a partir de contentores Docker isolados. Através desta abordagem garantimos que o software contido nestes contentores se comporte sempre da mesma forma, independentemente do local onde é instalado. Não obstante, apesar da utilização desta ferramenta não ser obrigatória para o funcionamento destes sistemas, a sua utilização permite reduzir o tempo dispendido na configuração e instalação de todo o software necessário para executar estas aplicações.

Ao assumirmos a divisão do sistema em dois servidores, um destinado ao acesso e gestão de dados (*back end*) e outro responsável pela apresentação desses dados (*front end*), introduzimos assim um servidor *proxy* reverso NGINX para fazer o roteamento dos pedidos a estes servidores, e desta forma permitir que ambos possam virtualmente escutar na porta 80. Este servidor *proxy* reverso encontra-se também instalado num contentor, tirando partido das vantagens apresentadas anteriormente. Para além disso, este servidor permite ainda a configuração de mecanismos de controlo sobre o tráfego de entrada, o que contribui para aumentar a segurança do sistema.

Na figura 4.1 pode ser visualizada a arquitetura descrita e o modo como os diferentes componentes do sistema estão ligados entre sí.

4.1.1 Dados do Sistema

Existem duas classes principais de dados: Aplicações e *Endpoints*. O diagrama apresentado na figura 4.2 evidencia o modo como essas classes se relacionam e as respectivas subclasses.

Aplicações

Cada aplicação possuí um nome, autor e uma descrição da sua funcionalidade. Um autor não pode publicar duas aplicações com o mesmo nome. Por sua vez, cada aplicação pode possuir diferentes versões. Isto permite que, caso se pretenda atualizar as aplicações para, por exemplo, receber um parâmetro adicional de dados, os sistemas que dependam dessa aplicação possam continuar a funcionar corretamente até serem atualizados para a nova versão. É possível adicionar também uma descrição a cada nova versão para indicar as alterações que foram efetuadas em relação à sua versão anterior.

Cada versão de uma aplicação é caracterizada pelo conjunto de ficheiros necessários à sua execução, pelos parâmetros de entrada e ainda pelas *streams* e ficheiros de *output*. Uma descrição mais detalhada a respeito do modo como as aplicações são configuradas será fornecida posteriormente na secção 4.3 deste capítulo.

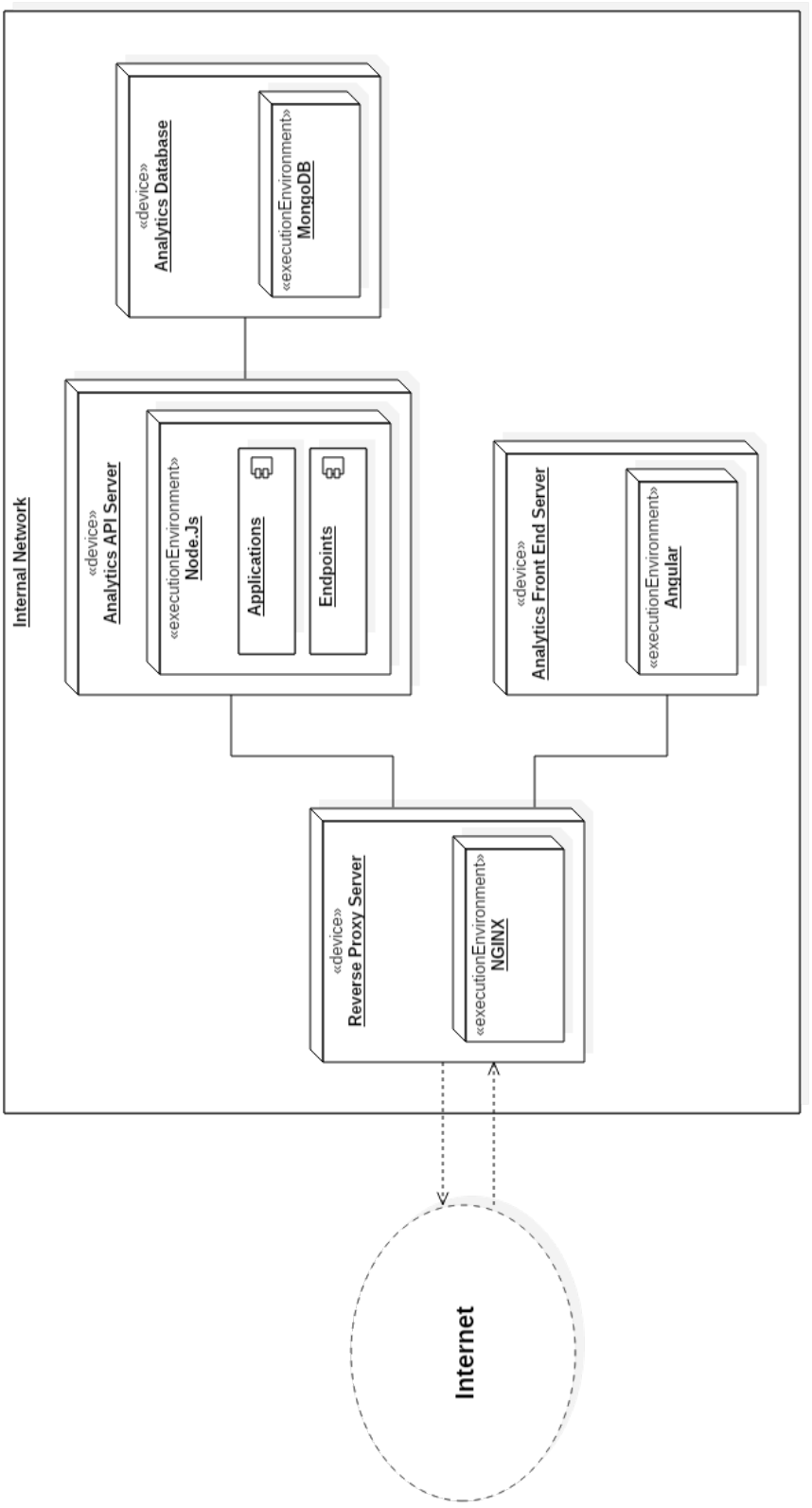

Figura 4.1: Diagrama de implementação do sistema.

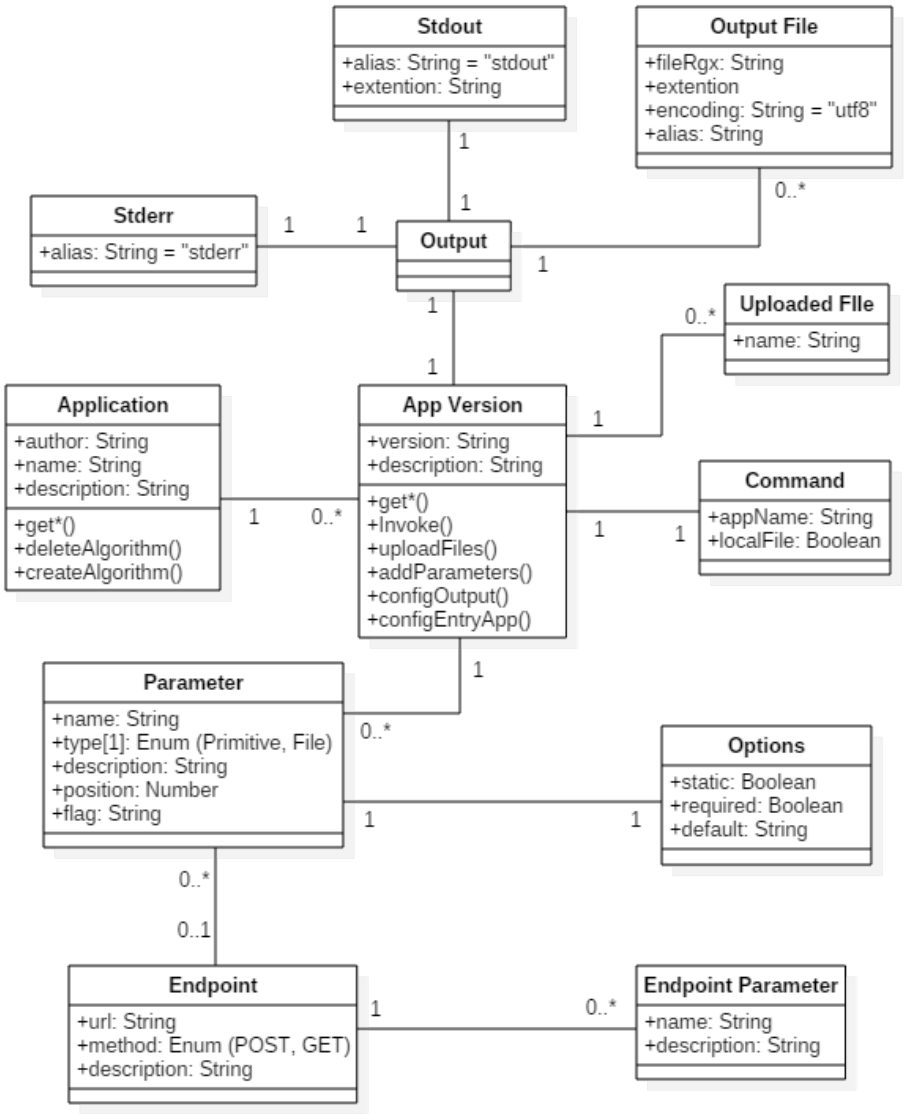

Figura 4.2: Diagrama de classes do sistema. Representa a informação usada na configuração das aplicações.

Endpoints

Um *endpoint* corresponde a um endereço de outro servidor destinado à recolha de dados externos, a serem fornecidos a uma dada aplicação. Os *endpoints* são caracterizados por um url, pelo tipo do método HTTP do pedido (POST/GET), por uma descrição e por um conjunto de parâmetros requiridos no pedido. Cada parâmetro possuí uma descrição relativa ao tipo do valor esperado.

4.2 Aplicação de Configuração

A aplicação de configuração desenvolvida tem como objetivos servir de assistente à gestão de aplicações no servidor e demonstrar as capacidades de processamento de dados e a API de invocação remota.

Com vista à gestão das aplicações do sistema (*BackOffice*), a interface gráfica desenvolvida permite efetuar a listagem e filtragem das aplicações instaladas no motor de análise, a criação e configuração de novas versões de uma aplicação e ainda eliminar quaisquer aplicações obsoletas. Ademais, é ainda oferecida a possibilidade de criar, listar, editar e remover os *endpoints* a serem usados para recolha de dados de fontes externas.

Foi ainda implementada uma valência que permite invocar as aplicações instaladas, possibilitando a atribuição de valores aos parâmetros de entrada de forma manual ou pela escolha de um *endpoint*, a partir do qual será recolhida a informação a ser analisada.

Aquando da invocação de uma aplicação, é possível, inclusive, definir uma série de opções para controlo do retorno e do ambiente de execução. Uma apresentação detalhada sobre esta funcionalidade será apresentada detalhadamente na secção 4.4.1 em conjunto com outras considerações relativas à possibilidade de codificação de dados num pedido de processamento.

Após a execução de uma aplicação, os dados retornados podem ser visualizados havendo a possibilidade de descarregar qualquer ficheiro de saída individualmente.

4.3 Instalação e Configuração de aplicações

Tal como referido no capítulo anterior, o ambiente de execução das aplicações é semelhante ao da linha de comandos. Desta forma, na evocação das aplicações é criada uma *string* de execução pelo servidor com o comando e lista de argumentos que serão usados para a criação de um processo. A *string* de execução possuí o seguinte formato:

<Command> *<Parameter 1> <Parameter 2> ... <Parameter N>*

O principal objectivo durante esta fase é perceber qual o comando a executar, quais os parâmetros de entrada e qual o *output* gerado pela aplicação.

Submissão de Ficheiros

Um dos primeiros passos para a instalação de uma aplicação passa pelo envio de todos os ficheiros necessários para executar uma determinada aplicação. Estes ficheiros são colocados num certo diretório relativo à aplicação.

Com o intuito de preservarmos a organização de ficheiros que possam depender de um programa específico, é possível efetuar o envio de ficheiros comprimidos, *e.g.*, .zip, .tar, sendo que estes permitem a codificação de diretórios e de todos os ficheiros nele contidos. Após a submissão, é efetuada a sua descompressão e remontada a ordem estrutural dos diretórios nele codificados.

Comando

O comando traduz-se no programa a ser executado. Esse programa pode referir-se a um ficheiro executável que tenha sido submetido para o servidor ou uma aplicação instalada no sistema.

Parâmetros de Entrada

Os parâmetros de entrada de uma aplicação são mapeados num vector de argumentos ao qual o processo acede para executar o pedido. Em geral, os parâmetros podem assumir uma posição específica no vector de argumentos ou serem identificados por uma *flag* precedente ao valor do argumento, *e.g.*, '-s size'.

Um parâmetro pode ainda referenciar um ficheiro. Neste caso, os dados de entrada terão de ser escritos num ficheiro cuja localização será exposta no valor do respectivo argumento.

Assim sendo, um parâmetro é identificado por um nome único, por uma descrição, pela sua posição no vector de argumentos, por um tipo (File/Primitive) e por uma *flag*.

É possível ainda incluir algumas opções adicionais durante a configuração de um parâmetro. Essas opções são:

- **Valor pré-definido:** Valor a ser usado no caso em que não sejam fornecidos dados de entrada para o parâmetro. A atribuição de um valor pré-definido a um parâmetro, do tipo ficheiro, implica que o ficheiro em causa tenha previamente sido submetido no servidor.
- **Parâmetro Obrigatório:** Indica que determinado parâmetro necessita de assumir obrigatoriamente um valor. A não atribuição de um valor a esse parâmetro aquando da invocação da aplicação irá gerar um erro e esta não será executada.
- **Parâmetro Estático:** Um parâmetro estático assumirá sempre o valor pré-definido atribuído, independentemente do valor que seja submetido no pedido de invocação.

Streams e Ficheiros de Output

O retorno dos dados de *output* das aplicações pode ser feito através da escrita da informação processada em ficheiros ou nas *streams* de *output*. Tendo em conta as dificuldades apresentadas no capítulo anterior, relativamente a detecção da localização onde os ficheiros de *output* são criados,

ao configurarmos pelo menos um ficheiro de *output* é automaticamente adicionado um parâmetro designado "OutputDir". Este parâmetro irá conter o caminho para um diretório de *output* de ficheiros criados em tempo de execução. A posição deste argumento é customizável, sendo possível adicionar-lhe uma *flag*.

Ao definirmos um ficheiro de *output*, para retornar a informação nele contida, podemos configurar as seguintes propriedades:

- **Nome:** Identificador do ficheiro de *output*. Usado para referenciar o ficheiro na mensagem de resposta a um pedido de análise.
- **Parser:** Nome do *parser* a ser utilizado para pós-processamento do conteúdo do ficheiro, de acordo a sua extensão. (Será apresentada posteriormente a importância desta operação.)
- Codificação: Indica o tipo de codificação a ser aplicada na leitura dos dados de um ficheiro. Por exemplo, base64, utf8, utf16le, etc...
- **Expressão Regular:** Expressão usada para determinar qual ou quais os ficheiro a serem retornados com base no seu nome, dentro do diretório de *output*.

Ambas as *streams* de *output* e erro possuem um nome identificador, que é usado na resposta ao cliente, para referenciar os valores retornados. A *stream* de *output* permite, ainda, definir um *parser* para pós-processamento dos dados.

4.3.1 Erros de Configuração

Uma configuração defeituosa de uma aplicação, tendo por base uma definição dos parâmetros imprecisa ou um comando inválido, irá ocasionar excepções que podem ter a sua génese tanto ao nível da criação do processo, como durante a sua execução. Nestes casos, o erro gerado será captado e enviado ao cliente para que possa, facilmente, ser detectado e corrigido.

4.4 Invocação e Execução das Aplicações

Após a fase de instalação e configuração, uma aplicação fica disponível para ser invocada remotamente. A invocação remota envolve as seguintes etapas:

1. Recepção de pedido de processamento de dados.
2. Preparação do ambiente de execução.
3. Execução da aplicação.
4. Recolha e envio do retorno ao cliente.

4.4.1 Pedido de Processamento

O pedido de invocação de uma aplicação para o processamento de dados pode ser efetuado a partir, tanto da aplicação de configuração desenvolvida neste projeto como a partir de aplicações externas, através de uma rota pública. O formato do conteúdo relativo ao pedido de processamento escolhido foi json, por ser amplamente usado na troca de informação entre aplicações.

Para cada parâmetro do tipo simples o valor será impreterivelmente uma *string*. Todavia, no que concerne aos parâmetros do tipo ficheiro, a informação pode ser submetida por três meios: pelo *upload* do ficheiro paralelamente ao pedido, através da referenciação de um ficheiro submetido durante a fase de instalação ou por meio do envio do seu conteúdo sob a forma embebida no json do pedido de processamento.

Optou-se por disponibilizar o envio dos dados correspondentes a um ficheiro diretamente no pedido de processamento, pelo facto de termos procurado desenvolver uma API que fosse de fácil utilização por parte de outros sistemas. Por conseguinte, nos casos em que os dados para o processamento não se encontrem no formato de ficheiro, não é necessário sua criação prévia.

Se optarmos pelo envio do conteúdo de um ficheiro diretamente no json do pedido, é ainda possível definir a sua extensão (*e.g.* xml, csv) e o tipo de codificação do texto (*e.g.* utf8, utf16le, base64, asci), sendo que por definição o texto será codificado em utf8.

Embora seja possível enviar o conteúdo de um ficheiro como texto no pedido de processamento, esta abordagem obriga a que as aplicações externas saibam como codificar os dados de acordo com a extensão do ficheiro de entrada esperado. De modo a contornar este problema, procuramos incluir a possibilidade de envio dos dados de um ficheiro em formato json. Esta funcionalidade poderá ser aplicada diretamente a ficheiros do tipo xml, csv e json por serem os formatos de ficheiro mais usados para representar informação. No entanto, como iremos ver a seguir, é possível a instalação de *plugins* de modo a suportar a conversão de outros tipos de ficheiros. Uma explicação mais detalhada sobre como os ficheiros de entrada são manipulados será apresentada na secção 4.4.2.

Ainda que da submissão dos dados de ficheiro em formato json facilite a comunicação de dados entre o servidor e os sistemas externos, esta funcionalidade pode acarretar um grande impacto ao nível do desempenho do servidor de análise. Ao receber um pedido neste formato, o servidor terá de processa-lo, carrega-lo em memória e converter para um objecto de JavaScript, o que pode gerar alguns problemas ao nível de utilização da memória, caso a quantidade de dados envolvida seja grande. Para fazer face a este problema, o servidor só aceitará pedidos cujo tamanho do corpo da mensagem não exceda os 2MB.

Apesar deste valor poder ser configurável, decidimos criar uma rota adicional do tipo multipart/form-data para permitir que a submissão de ficheiros possa ser feita de forma paralela ao pedido. Assim, os ficheiros são diretamente armazenados no servidor, minimizando a utilização de memória e tornando o serviço mais eficiente.

Opções de Processamento

Ao efetuarmos o envio de dados para processamento podemos ainda especificar algumas opções para controlar a forma como o pedido será executado.

Uma destas configurações corresponde ao tempo máximo em milissegundos que uma aplicação poderá estar em execução. Caso o tempo definido se esgote sem que a aplicação tenha retornado, o processo principal e os processos filho serão terminados e o cliente informado do ocorrido. Esta condição permite prevenir situações em que as aplicações, ao entrarem num estado de bloqueio, como num *loop* infinito, não fiquem por um tempo indefinido a consumir recursos do servidor. Por definição uma aplicação terá 10 segundos de tempo máximo de execução.

Em termos de retorno da informação de saída das aplicações existe a opção entre retornar os dados processados sob a forma de texto ou em formato json. A forma como os dados de *output* são convertidos para json depende das extensões configuradas para os ficheiros de *output*. As extensões suportados diretamente para pós-processamento são xml, csv e json, mas podem ser instalados *plugins* para processar outros tipos de ficheiros. O formato json produzido é semelhante àquele que pode ser usado para definir o conteúdo dos ficheiros de *input*, pelo que este pós-processamento pode ser visto como a operação reversa do pré-processamento, aplicada a dados diferentes.

Os dados retornados podem ainda ser filtrados, caso não se esteja interessado num ou mais tipos de *output*, o que incluí ficheiros e *streams*. Isto permite reduzir o tempo e recursos requeridos pelo servidor para processar um pedido.

É ainda possível executar a aplicação em modo seguro. Mais detalhes sobre a implementação desta funcionalidade serão apresentados na secção 4.5.

4.4.2 Preparação do ambiente de execução

Após a recepção de um pedido de análise, é efetuada uma validação dos dados recebidos de modo a serem filtrados pedidos mal formulados. Além disso, é realizada uma validação do conteúdo dos parâmetros recebidos com base nas restrições impostas durante a fase de configuração das aplicações. Os pedidos são recusados caso, por exemplo, seja referenciado um ficheiro inválido para processamento ou não sejam respeitadas as regras configuradas. Desta forma, evitamos colocar uma determinada aplicação em execução, poupando recursos computacionais no processo.

No seguimento da validação do pedido, procede-se à compilação da *string* de execução, que envolve a preparação dos parâmetros e dos ficheiros que serão processados pela aplicação.

A primeira operação efetuada resume-se a localizar a pasta onde se encontram guardados os ficheiros submetidos durante a fase de configuração. O caminho para essa pasta será o diretório de execução do processo (*current working directory - cwd*). Isto permite que a aplicação possa encontrar qualquer ficheiro relativo à sua localização. Caso não fosse definido o cwd do processo filho, este seria herdado do processo pai, neste caso do servidor, e a aplicação poderia não funcionar corretamente.

Seguidamente devemos encontrar o comando que queremos executar relativamente ao processo pai. No caso de corresponder a uma aplicação instalada globalmente no sistema, apenas necessitamos do nome dessa aplicação. Caso se trate de um executável que tenha sido submetido, necessitamos da localização desse ficheiro no sistema.

Antes de efetuarmos a compilação dos parâmetros, é criada uma pasta temporária que irá conter todos os ficheiros submetidos no pedido de processamento e uma pasta destinada a receber os ficheiros de *output* da aplicação, cuja localização será passada no vetor de argumentos, na eventualidade de terem sido configurados ficheiros de *output*.

Terminada esta fase, os dados contidos no json do pedido são convertidos e copiados para ficheiros por meio dos *plugins* de conversão instalados, os restantes ficheiros submetidos são movidos para a pasta temporária e os valores dos parâmetros são adicionados pela ordem em que foram configurados ao vector de argumentos.

Plugins

Tal como foi explicado, é possível enviar o conteúdo de um ficheiro num formato json. Apesar de suportarmos as extensões mais usadas (xml, csv, json), podem existir outros tipos de ficheiros que gostaríamos de usar.

Assim, criamos a possibilidade de instalar *plugins* no servidor para converter outros formatos de ficheiros. Um *plugin* consiste num ficheiro JavaScript, que contém um método *register*, para informar qual a extensão de ficheiro que será processada, e um método *parse* que converte o objecto json submetido no pedido para uma *string* que corresponde ao conteúdo do ficheiro temporário a ser criado.

Para os *plugins* de conversão de ficheiros de *outputs* para json, os mesmos métodos são usados, no entanto, o método *parse*, converte uma *string*, que corresponde ao conteúdo do ficheiro para um objecto json.

Os *plugins* são instalados na pasta "plugins/parsers", sendo que o servidor encarregar-se-à automaticamente de carrega-los.

Com esta abordagem é possível redefinir os conversores pré-instalados para as extensões xml e csv por forma a aceitarem um formato de json diferente e a criação de outros conversores para formatos json que mais se adequem às aplicações externas envolvidas.

4.4.3 Execução da Aplicação

Após prepararmos os ficheiros e os parâmetros que a aplicação irá receber, estamos prontos para iniciar a sua execução. Desta forma, um processo filho é criado, e os *streams* de *output* e de erro redirecionados para um servidor. Os dados são enviados em *runtime* para o servidor, que por sua vez os guarda num ficheiro para poupar memória.

Quando o programa termina, o código de terminação é guardado e os ficheiros que armazenam os dados das *streams* são fechados. Caso ocorra algum problema durante a criação e execução do processo, este é registado para informar o cliente do sucedido.

4.4.4 Recolha e Envio do Output

Após a execução das aplicações, necessitamos de devolver ao cliente a informação que foi analisada. Assim, os ficheiros de *output* são lidos e enviados em sequência, sendo que a informação neles contida, é convertida de acordo com o tipo de extensão do ficheiro e as opções de retorno. Escolheu-se esta abordagem de envio de dados sequencial em *stream*, em virtude de estarmos a trabalhar com ficheiros que podem conter muitos dados, e carregá-los todos para memória, com a finalidade de construir a resposta completa antes de o retornar, seria bastante custoso.

Por último, terminado o envio dos dados, todos os ficheiros temporários, que foram utilizados durante a fase de execução da aplicação, são eliminados por forma a libertar espaço em disco.

Considerações Finais

Tal como foi referido anteriormente, os dados submetidos por parâmetro podem ter como origem sistemas externos, pelo que se adicionou a possibilidade de associarmos a cada parâmetro um *endpoint*. No entanto, optou-se por transferir a responsabilidade de efetuar pedidos HTTP a esses *endpoints* às aplicações cliente. Uma vez que os serviços externos podem estar indisponíveis ou os *endpoints* mal configurados, seriam gerados erros que deveriam ser tratados ao nível do cliente. Além disso, a aquisição de dados por parte de um cliente permite que este possa facilmente filtrar a informação recolhida antes de enviar o pedido de processamento ao servidor. Esta transferência de responsabilidades permite ainda que possa ser poupado tempo de processamento ao servidor.

Desta forma, quando um cliente inquere o servidor para saber detalhes relativos a uma aplicação, é enviada juntamente a informação sobre os *endpoints* associados a cada parâmetro, para que este possa efetuar os pedidos de dados localmente.

4.5 Modo Seguro

Tal como foi identificado no capitulo anterior, um dos problemas desta abordagem é a questão das aplicações poderem interagir diretamente com o sistema operativo subjacente. Isto causa um problema grave em termos de segurança, visto que ficam expostos dados sensíveis, e o correto funcionamento do servidor pode ficar comprometido. Mesmo que uma aplicação seja tida como segura, esta pode acabar por ter interferência no modo de execução de outras aplicações e até mesmo no sistema de ficheiros subjacente causando alterações irreversíveis.

Não devemos confundir o problema de segurança em questão, relacionado com acesso a recursos críticos do sistema operativo do sistema, dos problemas relacionados com o acesso a informação sensível do servidor ou base de dados. Posto isto, propomos a utilização da virtualização como uma forma de garantir um isolamento entre o ambiente de execução de cada aplicação e o sistema operativo hospedeiro [Pah15].

Deparando-nos com esta problemática, consideramos a utilização de contentores como a forma de virtualização mais adequada, por ser mais leve em termos de utilização de recursos computacionais. Além disto, é necessário algum cuidado adicional, na sua implementação, já que os

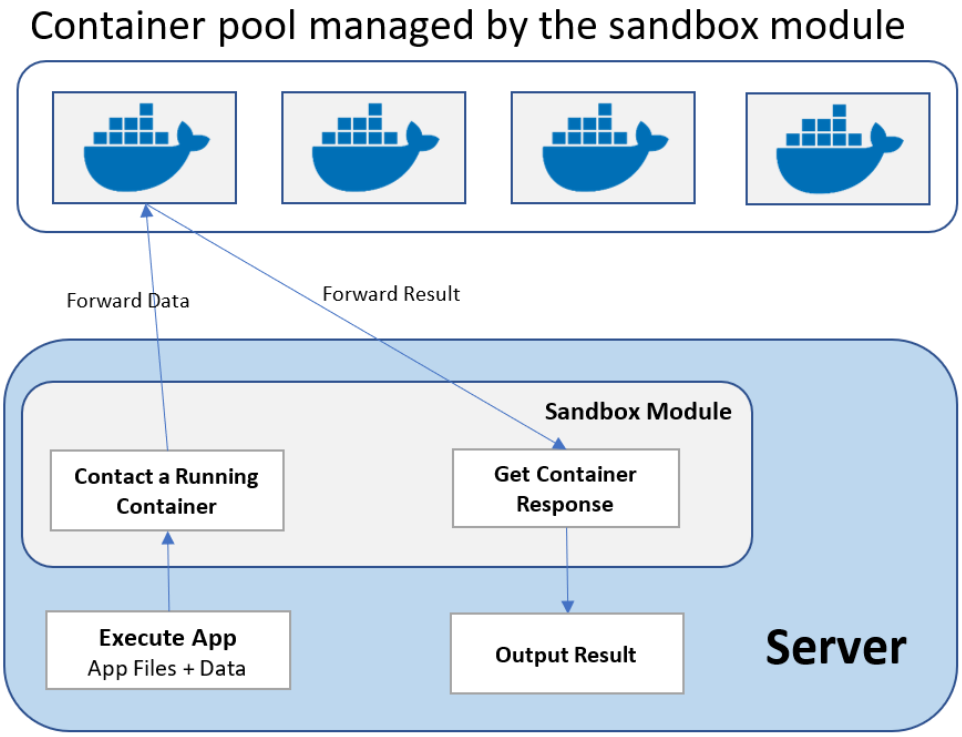

Figura 4.3: Principais tarefas do módulo de Sandbox.

contentores partilham alguns recursos diretamente com o sistema operativo. Para implementação desta funcionalidade foi usado o Docker, que fornece uma interface programável para gestão dos contentores.

Assim, implementou-se um módulo no servidor destinado à gestão de uma *pool* de contentores com tamanho configurável. Este módulo é responsável pelas seguintes tarefas:

- Inicializar a *pool* de contentores.
- Transferir os dados de execução para um contentor iniciado quando solicitado o modo seguro.
- Recolher os dados de retorno de uma aplicação executada num contentor.
- Destruir o contentor após terminada a execução e substitui-lo por um novo.

A imagem 4.3 resume de forma esquemática as principais tarefas realizada por este módulo.

4.5.1 Dinâmica dos Contentores

Cada contentor é inicializado a partir de uma mesma imagem que foi desenhada para cooperar diretamente com o servidor. Esta imagem contém uma aplicação Node que possuí uma rota destinada a receber pedidos para executar uma determinada aplicação. Procurou-se simplificar

ao máximo a aplicação Node, possibilitando que esta seja iniciada o mais rapidamente possível e melhorar a eficiência do sistema. Além disso, a aplicação foi compilada num *bundle* com o código minificado.

Registo do Contentor

Quando um contentor termina a sua inicialização e está pronto a receber pedidos de execução, deve informar de volta o servidor, invocando uma rota especial chamada \r*egister*, cujo acesso é limitado à rede interna do servidor.

Para completar esse registo, a aplicação Node executada no contentor terá de detetar o endereço IP deste e qual o endereço IP do *gateway*,determinando para onde deve efetuar o envio da mensagem de registo e submeter o seu endereço ao servidor. Os endereços IP são identificados através de chamadas ao sistema. De notar que os contentores são executados na mesma sub-rede do servidor de forma a permitir o correto roteamento das mensagens. Esta sub-rede é gerida automaticamente pelo Docker.

O servidor ao receber o pedido de registo, irá verificar se o IP submetido corresponde a algum dos contentores que foram inicializados, e em caso afirmativo, irá adiciona-lo à lista de contentores preparados para execução.

Execução de Aplicação

Quando o servidor necessitar de executar uma aplicação em modo seguro, deverá contactar um dos contentores que esteja disponível e submeter todos os dados e ficheiros temporários requeridos para executar essa aplicação.

O contentor, ao receber o pedido, irá atualizar a localização dos ficheiros no vector de argumentos, criar uma pasta para receber os ficheiros de *output* da aplicação e após determinar a localização do executável, irá criar um processo filho para processar os dados de entrada.

Após a preparação dos dados de entrada, a aplicação é executada e os dados produzidos pelas *streams* de *output* são guardados em ficheiros de forma paralela à sua execução. Após terminado o processamento dos dados, todos os ficheiros de *output* são arquivados num ficheiro .zip e retornados ao servidor.

O contentor será posteriormente destruído pelo servidor e um novo será criado para o substituir. Assim, por cada contentor não é executada mais que uma aplicação, uma vez que aplicações mal intencionadas podem danificar o sistema de ficheiros e a aplicação Node que controla o serviço de processamento do contentor e assim inviabilizar a execução de uma nova aplicação nesse ambiente.

4.5.2 Dinâmica do Servidor

O modo de execução seguro é em tudo semelhante ao modo não seguro, até ao ponto de criação do processo filho. Após a criação dos ficheiros temporários e a preparação da *string* de execução, os dados são submetidos a um dos contentores inicializados. Caso não haja nenhum

contentor disponível, é criado um tarefa assincrona que será colocada numa fila de espera. Assim que haja um contentor disponível, uma tarefa é retirada da fila e são submetidos os dados e ficheiros necessários para a execução da aplicação no contentor. O servidor aguarda, posteriormente, a resposta ao pedido de processamento efetuado ao contentor e, ao obter essa resposta, irá transferir e descomprimir os ficheiros de *output*. Caso o contentor não responda dentro do um limite máximo de tempo imposto nas opções de execução, a processamento é abortado e o contentor destruído.

A fase seguinte corresponde ao envio em *stream* dos dados ao cliente, sendo análogo ao anteriormente explanado. De notar que a destruição do contentor antigo e a criação de um novo, ulteriormente ao processamento dos dados, é feita em simultâneo com o envio da resposta ao cliente.

Na figura 4.4 é apresentado um diagrama de atividades que resume o modo como um pedido de execução remota é atendido.

4.5.3 Considerações de Segurança

A virtualização das dependências de uma aplicação por meio de um contentor permite isolar o seu ambiente de execução do servidor e das demais aplicações e, desta forma, havendo a possibilidade de uma aplicação mal intencionada comprometer um contentor, não causará qualquer impacto no funcionamento do sistema.

O modulo de segurança implementado permite ainda:

- Executar um programa não confiável num contentor Docker não privilegiado, como utilizador não privilegiado.
- Limitar a utilização máxima de memória e a quota de CPU disponível por contentor.
- Limitar a utilização máxima de espaço em disco em cada contentor.
- Terminar um programa se o seu tempo de execução exceder o limite imposto.

No entanto, esta implementação apresenta algumas vulnerabilidades, sendo que as seguintes foram identificadas:

- Não são impostos nenhum tipo de limites quanto à realização de operações de I/O pelas aplicações.
- Não é imposto nenhum limite em termos de acesso à rede por parte de uma aplicação.

Estas restrições são difíceis de serem implementadas corretamente através do Docker. Além disso, existem outras bibliotecas mais especializadas em solucionar este tipo de problemas tal como o recurso a uma *firewall* e a configuração de perfies em AppArmor/SELinux.

Além destas considerações de segurança que teriam de ser ratificadas antes do sistema ser colocado em modo de produção, deveríamos ainda analisar o modo como a utilização de contentores afeta a escalabilidade do serviço. De facto, um dos factores limitantes da escalabilidade advém da

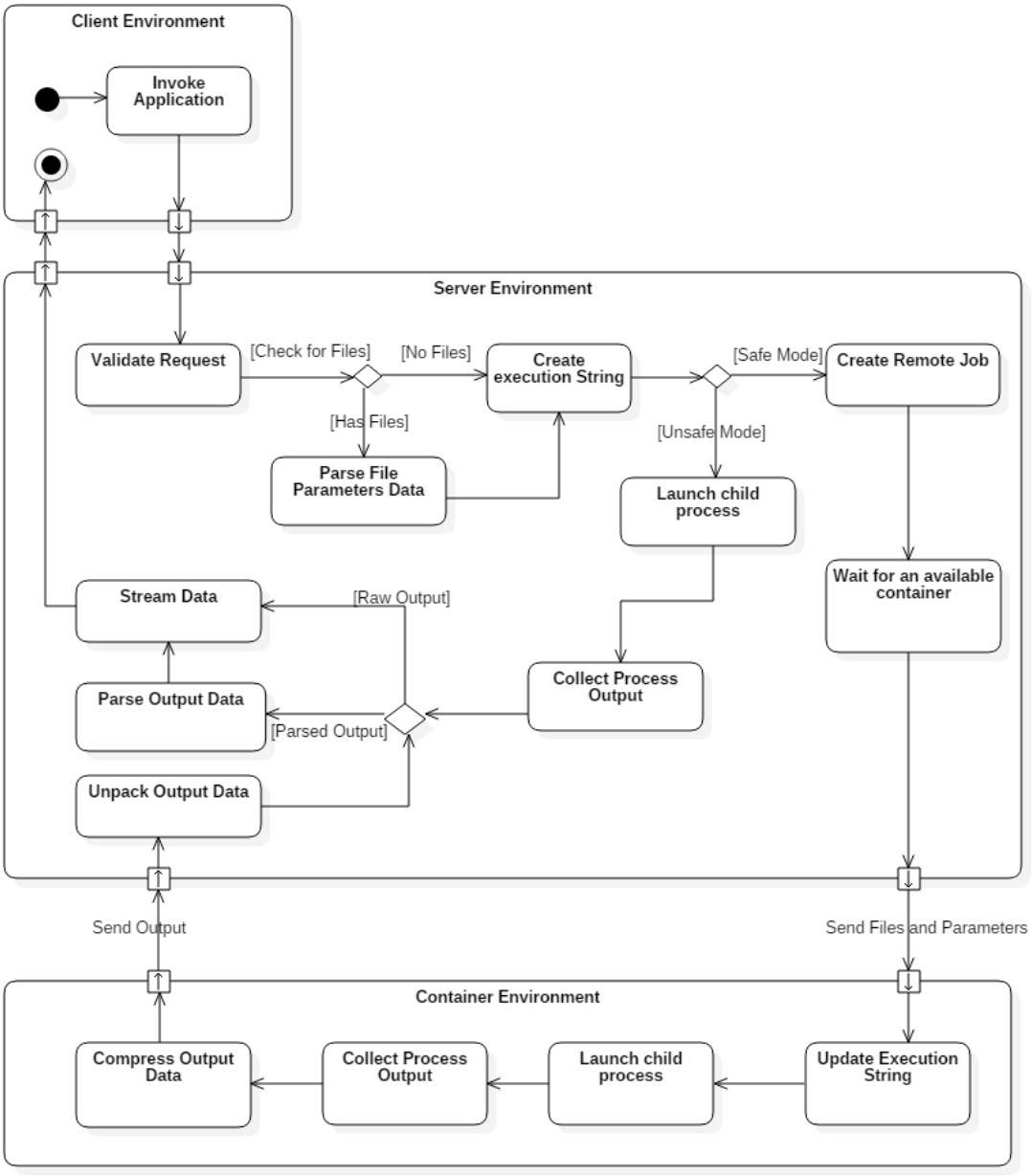

Figura 4.4: Diagrama de atividades resultante da invocação remota de uma aplicação. Representa o modo como a informação fluí no sistema dependendo das configurações de execução usadas.

quantidade de contentores necessários para responder ao número de pedidos esperado em pico máximo de utilização do sistema. Considerando que as capacidades computacionais de uma máquina física não são ilimitadas, o gestão de contentores teria de ser distribuído por outros máquinas. Felizmente o Docker possuí um modo *swarm* que facilita a orquestramento de contentores em diferentes máquinas e assim permite o balanceamento da carga computacional.

O isolamento oferecido pelo Docker é baseada em contentores LXC, que é uma funcionalidade oferecida pelo Kernel do Linux. Dado que tanto o *host* como os contentores partilham o mesmo kernel, a segurança pode ser comprometida se uma vulnerabilidade for encontrada no Kernel do Linux. A forma mais segura para correr aplicações não confiáveis traduz-se pela utilização de VMs tradicionais. Infelizmente, esta abordagem é mais difícil de ser implementada, uma vez que as VMs usam, significativamente, mais recursos computacionais que os contentores [CMF+16].

4.6 Resumo e Conclusões

Em resumo, o sistema de análise proposto é dividido em dois momentos distintos, sendo que um deles corresponde à instalação e configuração das aplicações e o outro à invocação remota das mesmas.

Uma vez que o propósito do sistema visa o processamento de dados, podendo estes ser de uma dimensão considerável, procurou-se durante o desenho do sistema ter alguns cuidados na implementação de certas funcionalidades,objetivando, principalmente, a minimização do uso de memória, poupando-se alguns recursos computacionais.

Procurou-se ainda oferecer formas de customização do formato dos dados de entrada e de saída, ambicionando-se a melhoraria da usabilidade e a facilitação do processo de integração da API de análise, por parte de outras entidades externas.

Com vista a dar resposta às questões de segurança levantadas, apostou-se na utilização da virtualização com recurso a contentores como método de isolamento do sistema.

Capítulo 5

Testes e Resultados

Este capítulo tem como objetivo apresentar os resultados e reflexões relativas ao trabalho efetuado ao longo da presente dissertação de mestrado. A fim de testar o sistema, foram criadas algumas aplicações em diferentes linguagens e seguidamente foram instaladas na plataforma desenvolvida para verificarmos o modo de atuação do sistema.

Estabelecemos, ademais, uma comparação entre os tempos de resposta a um pedido em modo seguro e não seguro das aplicações desenvolvidas, e apresentamos as condições em que foram executados os testes.

5.1 Ambiente e Condições de Teste

O servidor, aplicação de configuração e contentores do modo seguro foram executados através do Docker. A máquina *host* tinha como sistema operativo o Windows 10 Pro e estava equipada com um processador Intel Core i7-7700HQ @ 2.80GHz e com 16.0GB de RAM.

Visto que as funcionalidades requiridas pelo Docker para inicializar contentores dependem do Linux Kernel foi necessário recorrer a uma máquina virtual, através da tecnologia Hyper-V. A máquina virtual foi configurada com dois processadores virtuais e 2GB de RAM.

Não foram impostos quaisquer limites nas quotas de utilização do CPU e de memória nos contentores usados na virtualização do modo seguro, de forma a que as condições de execução fossem semelhantes as encontradas no servidor.

Para a disponibilização de um modo seguro, foi utilizada uma *pool* com apenas um contentor, sendo que no momento dos pedidos de invocação segura, o contentor estava completamente inicializado e pronto a receber pedidos. Esta abordagem foi implementada com o objetivo de avaliar o impacto introduzido no tempo de processamento do pedido, causado pela utilização de contentores. No entanto, como os pedidos podem ser colocados numa fila de espera quando não existem contentores disponíveis, uma análise relativa ao impacto do tempo de inicialização será posteriormente feita.

Os tempos apresentados na secção seguinte resultam de uma média, tendo sido realizados 20 pedidos consecutivos com os mesmos dados de entrada para cada cenário de execução. O tempo dispendido para o envio dos dados do pedido e da resposta não foram contabilizados na medição.

5.2 Aplicações de teste

Para testar a usabilidade do sistema e validar as funcionalidades implementadas foram criadas quatro aplicações. Foram ainda recolhidos alguns tempos de execução nas condições referidas na secção anterior para avaliar o atraso computacional introduzido pelo modo seguro.

Aplicação 1 - Hello World

Esta aplicação foi desenvolvida em C++, e cinge-se a escrever a mensagem "hello world"para o *stdout*, tendo sido desenvolvida com o intuito de provar a capacidade de executar ficheiros binários e a captação das mensagens oriundas das *standard streams*.

Dadas as condições de teste apresentadas, obtivemos os resultados listados na tabela 5.1. De notar que volume total de dados movimentados entre servidor e contentor foi de aproximadamente 12KB.

Tabela 5.1: Tempo médio de execução da aplicação de teste número 1.

Processos Simultâneos	Modo de Execução	Tempo Médio (ms)	Desvio Padrão (ms)
1	Não Seguro Seguro	30 132	3 10
5	Não Seguro Seguro	62 215	8 14
15	Não Seguro Seguro	191 567	11 21
30	Não Seguro Seguro	369 1119	15 38

Aplicação 2 - Node App

Esta aplicação foi desenvolvida em JavaScript para ser interpretada pelo Node.Js. Esta recebe como parâmetros a localização de um ficheiro de entrada e da pasta de *output*. A aplicação lê o ficheiro de entrada e escreve o seu conteúdo num outro ficheiro localizado no diretório de *output*. O propósito desta aplicação foi testar o *upload* de ficheiros, os conversores de json para xml e csv, e os reconversores desses formatos para json.

Dadas as condições de teste apresentadas, obtivemos os resultados listados na tabela 5.2. De notar que volume total de dados movimentados entre servidor e contentor foi de aproximadamente 6KB.

Tabela 5.2: Tempo médio de execução da aplicação de teste número 2.

Processos Simultâneos	Modo de Execução	Tempo Médio (ms)	Desvio Padrão (ms)
1	Não Seguro	103	10
	Seguro	214	12
5	Não Seguro	178	13
	Seguro	324	15
15	Não Seguro	418	17
	Seguro	811	36
30	Não Seguro	792	31
	Seguro	1740	48

Aplicação 3 - Python App

Esta aplicação desenvolvida em Python permite ler as horas, quando disponibilizada a imagem de um relógio analógico não rodado. Esta aplicação foi desenvolvida para demonstrar a possibilidade de executar aplicações instaladas no sistema, visto que depende do interpretador de Python e da biblioteca OpenCV, sendo que permitiu também testar a codificação de ficheiros binários no pedido json.

Atendendo às condições de teste apresentadas, obtivemos os resultados listados na tabela 5.3. De notar que volume total de dados movimentados entre servidor e contentor foi de aproximadamente 60KB.

Tabela 5.3: Tempo médio de execução da aplicação de teste número 3.

Processos Simultâneos	Modo de Execução	Tempo Médio (ms)	Desvio Padrão (ms)
1	Não Seguro	223	10
	Seguro	321	11
5	Não Seguro	453	13
	Seguro	552	17
15	Não Seguro	1181	32
	Seguro	1470	39
30	Não Seguro	2249	63
	Seguro	2847	67

Aplicação 4 - HRV App

Esta aplicação foi desenvolvida em MatLab com o objectivo de calcular diferentes métricas relativas à variabilidade do batimento cardíaco a partir dos intervalos temporais entre picos R normais. No anexo A, podem ser consultadas mais informações relativas à importância da variabilidade do batimento cardíaco, sendo que também são apresentadas algumas das métricas calculadas.

Dadas as condições de teste apresentadas, obtivemos os resultados listados na tabela 5.3. De notar que volume total de dados movimentados entre servidor e contentor foi de aproximadamente 1.26MB.

Tabela 5.4: Tempo médio de execução da aplicação de teste número 4.

Processos Simultâneos	Modo de Execução	Tempo Médio (s)	Desvio Padrão (s)
1	Não Seguro Seguro	7.15 7.62	0.09 0.13
5	Não Seguro Seguro	9.28 9.67	0.26 0.32
15	Não Seguro Seguro	24.97 27.11	0.68 0.81
30	Não Seguro Seguro	47.56 49.59	1.03 1.18

Conclusões

Como podemos ver através da comparação das medições de tempos por pedido nos dois modos de execução, o modo seguro demonstrou ter um tempo médio ligeiramente superior. Isso deve-se ao facto dos dados de entrada, depois de serem preparados pelo servidor, necessitarem de serem enviados a um contentor que, após recolher o *output*, terá que o enviar de volta ao servidor. Esta transmissão dos dados de entrada e de saída entre contentor e servidor é apontada como a principal causa para a diferença de tempos entre os dois modos de execução. Desta forma, quanto maior for o volume de dados a transferir, maiores serão as diferenças esperadas.

5.3 Análise do Impacto do Tempo de Inicialização dos Contentores

Decorridas algumas medições, concluímos que o tempo de inicialização, de apenas um contentor no ambiente considerado, foi de cerca de 2,08 segundos. Este tempo, no entanto, só terá impacto no tempo total de serviço caso o pedido de processamento seja colocado na fila por falta de contentores disponíveis, uma vez que quando a fila está vazia, poderá existir um contentor completamente inicializado ou em processo de inicialização, pelo que o tempo de serviço adicional será inferior. Além disso, os contentores são criados em paralelo, pelo que o tempo necessário para criar C contentores é bastante inferior a $C \times 2,08$. Num teste realizado com uma *pool* de 10 contentores, a inicialização simultânea destes demorou 10,13 segundos. Com a utilização de uma *swarm* distribuída por diferentes máquinas, este valor tem potencial para ser bastante inferior.

Para utilização deste sistema em modo de produção, teria de haver uma cuidada análise em relação à taxa média de chegada de pedidos ao sistema (λ) e à taxa média do serviço (μ), que depende do tempo de criação de um contentor e de execução de uma aplicação. Visto que estamos na presença de um problema de filas de espera, o número mínimo de contentores (C) requiridos para o funcionamento do sistema deve ser suficiente para que a equação 5.1 seja respeitada.

$$\frac{\lambda}{C\mu} < 1 \tag{5.1}$$

É possível também calcular o tempo de espera na fila procurando minimizá-lo e, assim, aumentar a responsividade do serviço.

Um outro fator ainda a considerar para diminuir o tempo de inicialização, passa pela simplificação das imagens usadas para inicializar contentores. Caso estejamos perante um problema muito específico, podemos diminuir o número de dependências de *software* da imagem.

Podemos ainda assumir a possibilidade de as aplicações, após instalação, passarem por um processo de validação para estas não necessitarem de correr em modo seguro e diminuirmos a necessidade de contentores.

Não obstante, não encontramos a necessidade de utilizar mais que um contentor para realizar os testes apresentados. Em ambiente de produção é possível gerir um número de contentores que pode ascender à ordem das centenas por cada máquina física, dependendo da quantidade de memória e da quota de CPU atribuída a cada contentor.

5.4 Uso da API para criação de uma aplicação de análise

Após o desenvolvimento da API de análise e da aplicação de configuração (*Backoffice*) de gestão das aplicações instaladas, criou-se uma aplicação Web para o processamento de dados relativos à variabilidade do batimento cardíaco. Esta aplicação Web foi concebida tendo em vista a demonstração das capacidades da API desenvolvida para o desenho de serviços de análise e processamento de dados.

O grande propósito desta aplicação, foi testar a aplicabilidade do motor de análise para o contexto apresentado no capítulo inicial desta dissertação. Assim, foi instalada no motor a aplicação de HRV descrita na secção 5.2 e configurados os seus parâmetros e o seu *output*. Na figura 5.1 é apresentado a interface de configuração dessa aplicação a partir do *backoffice* do servidor de análise.

Os dados a serem processados são descarregados a partir de um servidor desenvolvido na instituição, com o objetivo de se fazer o processamento online, configurou-se um *endpoint*. Esse servidor visa, primariamente, o registo de eventos operacionais de profissionais de risco, procurando permitir uma melhor coordenação das suas atividades e prevenir eventuais situações críticas. Assim, são recolhidos diversos dados ambientais e fisiológicos dos indivíduos, recorrendo a dispositivos *wearable* biomédicos. Dado que a prevenção de riscos passa por uma monitorização contínua dos dados recolhidos, aplicações de análise são essenciais neste sistema. Assim, esta aplicação visa ser integrada neste sistema, fornecendo uma interface orientada à monitorização de dados recolhidos, relativos à variabilidade da frequência cardíaca para detecção de níveis de stress e fadiga.

A aplicação Web inquere o servidor para obter informação relativa à aplicação de HRV nele instalada, e desta forma saber quais os seus parâmetros de entrada. Na sua interface é apresentado ao utilizador um formulário com os parâmetros requiridos pela aplicação e *endpoint*, e após submissão, os dados processados são agrupados de acordo com o seu domínio e apresentados ao

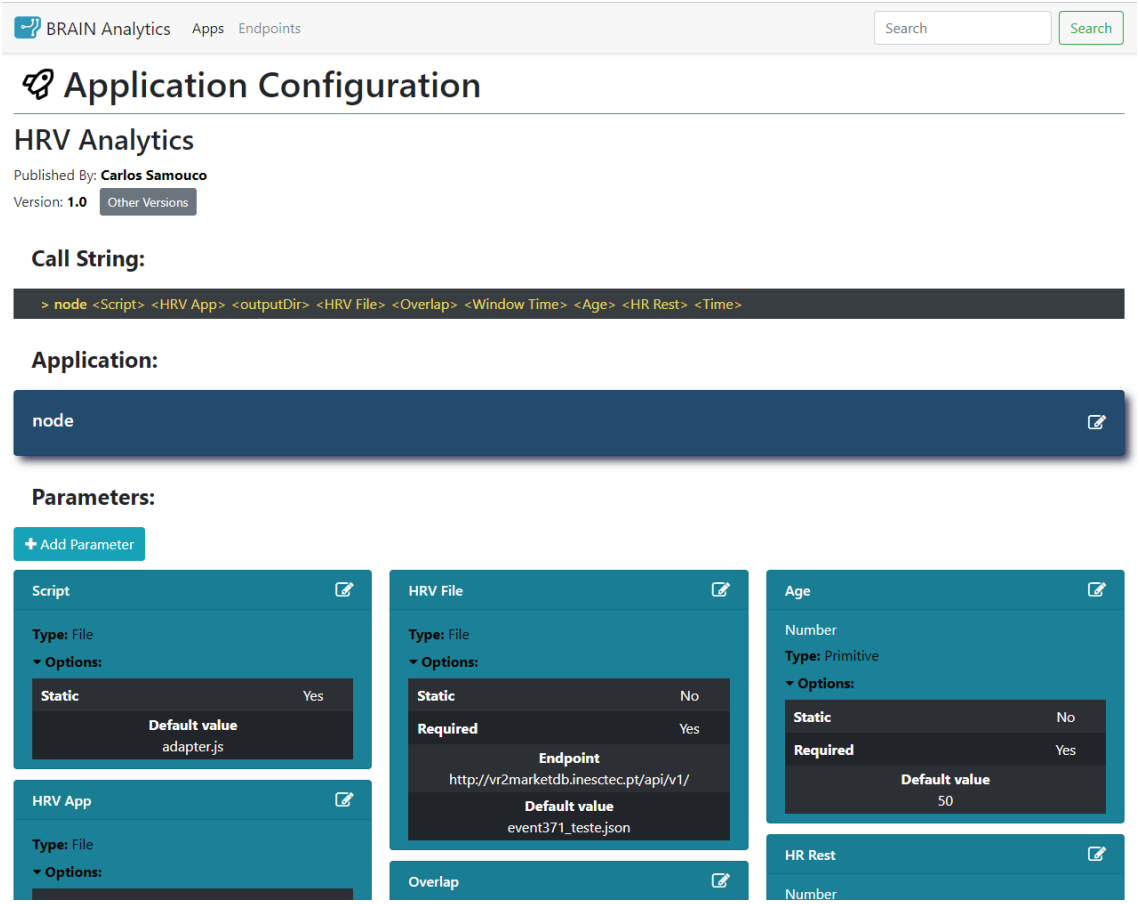

Figura 5.1: A figura mostra a interface de configuração da aplicação de análise.

utilizador sobe a forma de gráficos. Na figura 5.2 é mostrada a interface da aplicação de análise de dados de HRV, após estes terem sido processados pelo servidor.

Uma vez que a aplicação de HRV retorna dados sob a forma de um ficheiro csv, tirou-se partido das capacidades de pós-processamento do servidor para converter esses dados para formato json, e assim facilitar a programação da aplicação Web de análise destes dados.

5.5 Sumário

A utilização de contentores torna-se crucial para possibilitar a execução de aplicações não confiáveis e permite o isolamento das partes sensíveis do sistema. Devemos notar, no entanto, que a tecnologia de contentores não é suportada pelo Windows, pelo que foi usado um ambiente virtualizado. Desta forma, é presumível que se os testes tivessem sido realizados num ambiente com Linux nativo os resultados alcançados seriam ligeiramente melhores.

Como verificado, a utilização de contentores na implementação apresentada introduz uma pequena sobrecarga a nível da execução das aplicações. Além disso, os contentores necessitam de

algum tempo para inicializarem, sendo que isso pode ter impacto no tempo de resposta do serviço, caso o número de contentores não faça face ao número médio de pedidos de processamento seguro.

Por forma a utilizar a API criada para análise de dados num problema real, foi desenvolvida uma aplicação de análise.

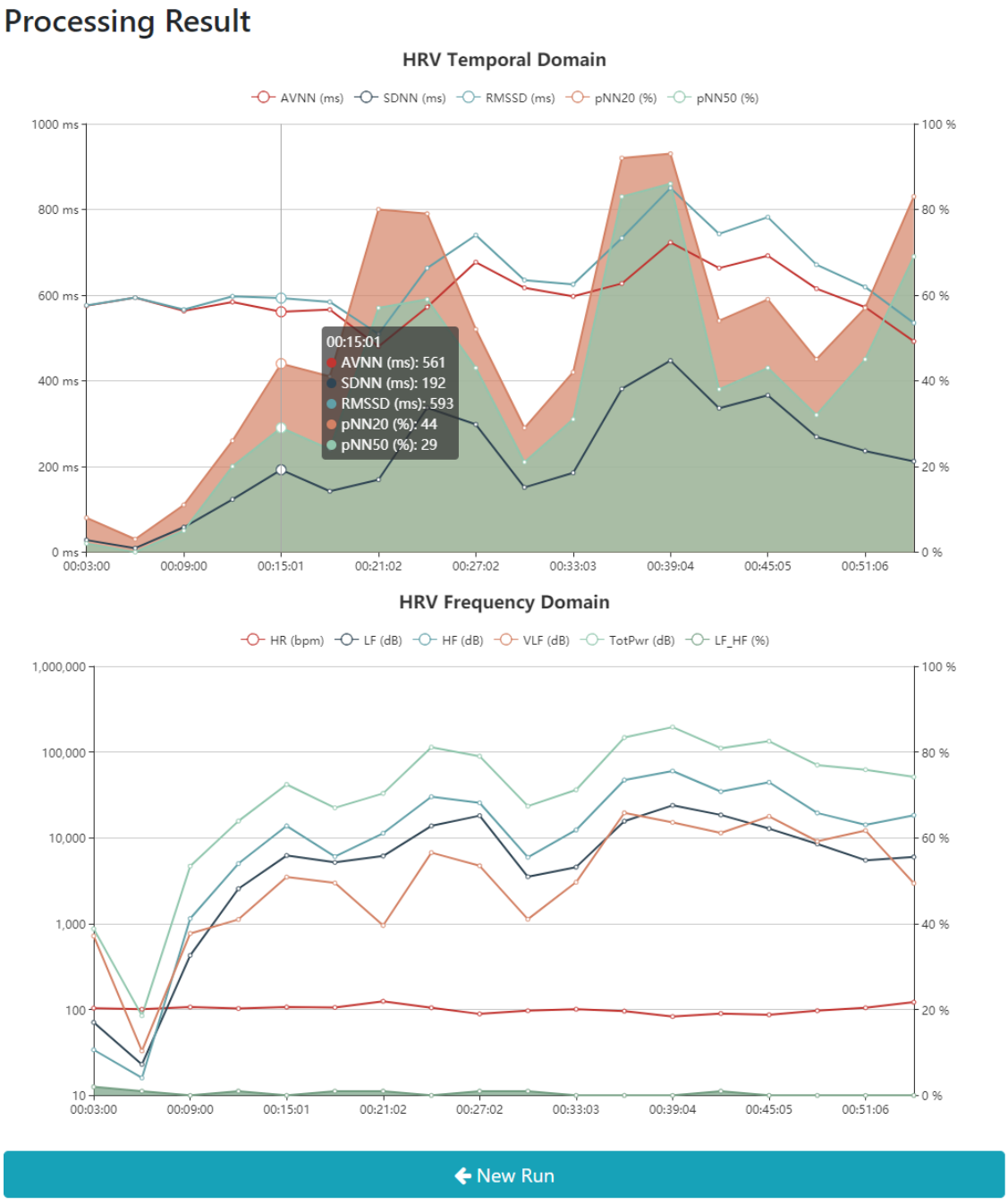

Figura 5.2: A figura mostra a interface de visualização do gráfico de análise. O gráfico superior corresponde às variáveis de HRV relativas ao domínio temporal e o gráfico inferior às variáveis de domínio de frequência – ver anexo A.

Capítulo 6

Conclusões e Trabalho Futuro

Este capítulo visa oferecer uma reflexão sobre os principais objetivos que foram alcançados durante a realização do trabalho. São apresentadas ainda uma série de guias com vista a melhorar e introduzir novas funcionalidades numa futura interação do trabalho até aqui desenvolvido.

6.1 Conclusões

Apesar da maior parte das funcionalidades propostas terem sido completadas face ao plano de trabalhos, devemos notar que, por via da natureza do planeamento do projeto, foram desenvolvidas primeiro as funcionalidades mais relevantes e as menos importantes foram deixadas para o final, sendo que gostaríamos de ter adicionado mais funcionalidades ao sistema de análise desenvolvido.

O trabalho teve um enfoque muito grande no desenho do motor de análise, nomeadamente na fase de estudo relativo à estrutura e tipos de dados necessários para executar as aplicações e também no estudo de uma arquitetura para gestão dos modos de execução referidos.

A aplicação cliente, foi desenhada a compreender as funcionalidades necessárias para permitir a instalação e execução de aplicações no servidor de análise. Pelos meandros desta dissertação, tomei a iniciativa de implementar um modo de execução seguro, o que se demonstrou bastante relevante para salvaguardar a proteção do sistema. Assim, por uma questão temporal, não foi possível implementar uma funcionalidade relativa à escolha de modos de visualização pré-definidos para representar os dados após o processamento.

O servidor de análise por seu lado, efetua a gestão das configurações e ficheiros das aplicações submetidas, além de controlar o fluxo de execução e os dados de entrada e saída usados no processamento. O servidor é, ainda, responsável pela gestão de uma *pool* de contentores que são usados para virtualizar o ambiente de execução das aplicações e assim criar uma barreira de proteção no sistema contra usos mal intencionados. Contudo, a utilização deste mecanismo implica um maior dispêndio de recursos computacionais.

Por forma a testar a aplicabilidade do motor de análise num problema real, este foi integrado com um servidor de recolha de dados fisiológicos para o cálculo de diversas vareáveis relacionadas com a variabilidade do batimento cardíaco.

6.2 Trabalho Futuro

Considerando a continuação deste trabalho, o acesso as funcionalidades de configuração e instalação de aplicações necessitavam de ser restringidas por meio de autenticação.

Para aumentar as capacidades do serviço de processamento, poderiam ainda ser implementadas *websockets* que permitiriam a realização de operações de I/O diretamente com as *standard streams* de um processo e desta forma possibilitariam o acompanhamento da execução de uma aplicação em tempo real.

Poderia ainda ser criado um método de instalação remota de *plugins* por forma a fornecer um maior nível de abstração neste processo.

Seria ainda importante adicionar mais tipos de validação de dados de entrada para diminuir a taxa de erros na utilização das aplicações. Isso seria alcançável, por exemplo, pela verificação do tipo de dados (números/strings) e pela definição de uma escala ou conjunto de valores permitidos.

A utilização de contentores poderia ainda tirar partido de uma implementação distribuída com recurso ao uso de Docker Swarm, e assim possibilitar a aplicação deste serviço numa larga escala.

A segurança do sistema necessitava de ser endurecida pela utilização conjunta de contentores e outras funcionalidades adicionais de segurança do kernel Linux como por exemplo, AppArmour, SELinux e GRSEC.

Relativamente à aplicação cliente desenvolvida para análise da variabilidade do batimento cardíaco, seria vantajoso implementar a monitorização contínua e em tempo real destes dados, todavia, o estado de maturação do projeto onde foi integrada esta dissertação não o permitiu.

Posto isto, no seguimento deste projeto, estes seriam aspetos a melhorar.

Referências

[Ber14] D. Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, Sept 2014.

[CJS12] M. Castro, A. J. Jara e A. F. Skarmeta. An analysis of m2m platforms: Challenges and opportunities for the internet of things. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 757–762, July 2012.

[CMF+16] A. Celesti, D. Mulfari, M. Fazio, M. Villari e A. Puliafito. Exploring container virtualization in iot clouds. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–6, May 2016.

[DJ16] N. N. Dlamini e K. Johnston. The use, benefits and challenges of using the internet of things (iot) in retail businesses: A literature review. In *2016 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, pages 430–436, Nov 2016.

[Gol15] N. Golchha. Big data-the information revolution. *Internetional Journal of Applied Research*, 1(12):791–794, 2015.

[grs] grsecurity. What is grsecurity. Disponível em https://grsecurity.net, acedido a última vez em 19 de Junho de 2018.

[hKRM17] Tai hoon Kim, Carlos Ramos e Sabah Mohammed. Smart city and iot. *Future Generation Computer Systems*, 76:159 – 162, 2017.

[Inca] Docker Inc. About storage drivers. Disponível em https://docs.docker.com/storage/storagedriver/, acedido a última vez em 19 de Junho de 2018.

[Incb] Docker Inc. Docker security. Disponível em https://docs.docker.com/engine/security/security/, acedido a última vez em 19 de Junho de 2018.

[Kha04] Ehsan Khan. Clinical skills: the physiological basis and interpretation of the ecg. *British Journal of Nursing*, 13(8):440–446, 2004.

[KLD12] M. Kovatsch, M. Lanter e S. Duquennoy. Actinium: A restful runtime container for scriptable internet of things applications. In *2012 3rd IEEE International Conference on the Internet of Things*, pages 135–142, Oct 2012.

[Liba] The GNU Library. Descriptors and streams. Disponível em https://www.gnu.org/software/libc/manual/html_node/Descriptors-and-Streams.html, acedido a última vez em 19 de Junho de 2018.

[Libb] The GNU Library. Program arguments. Disponível em https://www.gnu.org/software/libc/manual/html_node/Program-Arguments.html, acedido a última vez em 19 de Junho de 2018.

[Libc] The GNU Library. Standard environment variables. Disponível em https://www.gnu.org/software/libc/manual/html_node/Standard-Environment.html, acedido a última vez em 19 de Junho de 2018.

[Libd] The GNU Library. Standard streams. Disponível em https://www.gnu.org/software/libc/manual/html_node/Standard-Streams.html, acedido a última vez em 19 de Junho de 2018.

[MCK+09] N. Markatchev, R. Curry, C. Kiddle, A. Mirtchovski, R. Simmonds e T. Tan. A cloud-based interactive application service. In *2009 Fifth IEEE International Conference on e-Science*, pages 102–109, Dec 2009.

[MGMTG15] Yasir Mehmood, Carmelita Görg, Maciej Muehleisen e Andreas Timm-Giel. Mobile m2m communication architectures, upcoming challenges, applications, and future directions. *EURASIP Journal on Wireless Communications and Networking*, 2015(1):250, Nov 2015.

[MKPG16] Amith Raj MP, A. Kumar, S. J. Pai e A. Gopal. Enhancing security of docker using linux hardening techniques. In *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, pages 94–99, July 2016.

[MMG18] Joseph E. Mietus, George B. Moody e M.D. Ary L. Goldberger. Heart Rate Variability Analysis with the HRV Toolkit. PhysioNet, available at em http://physionet.org/tutorials/hrv-toolkit/, February 2018.

[MNG+17] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa e I. Yaqoob. Big iot data analytics: Architecture, opportunities, and open research challenges. *IEEE Access*, 5:5247–5261, 2017.

[Mon] MongoDB. What is mongodb? Disponível em https://www.mongodb.com/what-is-mongodb, acedido a última vez em 19 de Junho de 2018.

[ngi] nginx. Avout nginx. Disponível em https://nginx.org/en/, acedido a última vez em 19 de Junho de 2018.

[NLB18] A. Neumann, N. Laranjeiro e J. Bernardino. An analysis of public rest web service apis. *IEEE Transactions on Services Computing*, pages 1–1, 2018.

[Nod] Node.js. About node.js®. Disponível em https://nodejs.org/en/about/, acedido a última vez em 19 de Junho de 2018.

[PACA17] Tânia Pereira, Pedro R. Almeida, João P.S. Cunha e Ana Aguiar. Heart rate variability metrics for fine-grained stress level assessment. *Computer Methods and Programs in Biomedicine*, 148:71 – 80, 2017.

[Pah15] C. Pahl. Containerization and the paas cloud. *IEEE Cloud Computing*, 2(3):24–31, May 2015.

[Pro] SELinux Project. What is selinux. Disponível em http://www.selinuxproject.org/page/Main_Page, acedido a última vez em 19 de Junho de 2018.

[SC15] P. M. Pinto Silva e João Paulo Silva Cunha. SenseMyHeart: A service and API for wearable heart monitors. *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 4986–4989, 2015.

[TCP+12] S. Turchi, L. Ciofi, F. Paganelli, F. Pirri e D. Giuli. Designing epcis through linked data and rest principles. In *SoftCOM 2012, 20th International Conference on Software, Telecommunications and Computer Networks*, pages 1–6, Sept 2012.

[TEC18] INESC TEC. Instituição inesc tec. Disponível em https://www.inesctec.pt/pt/instituicao, February 2018.

[Wik] Ubuntu Wiki. Apparmor. Disponível em https://wiki.ubuntu.com/AppArmor, acedido a última vez em 19 de Junho de 2018.

[XMM+12] Borejda Xhyheri, Olivia Manfrini, Massimiliano Mazzolini, Carmine Pizzi e Raffaele Bugiardini. Heart rate variability today. *Progress in Cardiovascular Diseases*, 55(3):321–331, Nov-Dec 2012.

REFERÊNCIAS

Anexo A

Variabilidade do Batimento Cardíaco

A.1 O que é a variabilidade do batimento cardíaco?

Em cada batimento cardíaco é gerado um sinal elétrico nas células nervosas do coração que desencadeia a contração auricular e ventricular. Através de eletrodos é possível medir as diferenças de potencial elétrico geradas pelo impulso nervoso produzido no coração, sendo que essa informação pode ser registada graficamente através de um eletrocardiograma [XMM+12, Kha04].

A contração simultânea das cavidades ventriculares é representada no ECG pelo intervalo QRS [Kha04]. A frequência do batimento cardíaco é influenciada pelo sistema nervoso autônomo, do qual fazem parte os sistemas nervosos simpático e parassimpático. O stress é ativador do sistema nervoso simpático. O efeito deste no coração é entre outros aumentar a frequência cardíaca, ou seja o número de batimentos cardíacos por minuto [SC15].

Através da análise de ECGs é possível medir os intervalos de tempo entre batimentos cardíacos (intervalos RR), que corresponde à duração de um ciclo cardíaco [Kha04]. A variabilidade do batimento cardíaco centra-se desta forma na análise da variação do intervalo de tempo entre os picos R normais, que são chamadas intervalos NN (*normal-to-normal RR intervals*). Os intervalos NN são utilizados para o calculo de diversas vareáveis que permitem a detenção de estados anormais nos indivíduos tal como stress e fadiga [XMM+12].

Os métodos de análise relativos à variabilidade do batimento cardíaco podem ser agrupadas de acordo com o seu domínio, podendo este ser temporal ou de frequência.

De entre as vareáveis calculadas são apresentadas as seguintes [MMG18, XMM+12, PACA17]:

- **Domínio temporal:**
  - **AVNN (ms)** — valor médio dos intervalos de NN.
  - **SDNN (ms)** — desvio-padrão dos intervalos de NN.
  - **rMSSD (ms)** — raiz quadrada da média das diferenças entre os intervalos sucessivos.
  - **pNN20 (%)** — rácio dos intervalos de NN consecutivos, cuja diferença é superior a 20ms, dividido pelo número total de intervalos de NN.

  - **pNN50 (%)** — rácio dos intervalos de NN adjacentes, diferindo em mais de 50ms em relação a todos os intervalos NN.

- **Domínio de frequência:**

  - **HR (bpm)** — número de batimentos por minuto.
  - **LF** — potência espectral total de todos os intervalos NN entre 0.04 e 0.15 Hz.
  - **HF** — potência espectral total de todos os intervalos NN entre 0.15 and 0.4 Hz.
  - **VLF** — potência espectral total de todos os intervalos NN entre 0.0033 to 0.04 Hz.
  - **TOTPWR** — potência espectral total de todos os intervalos NN até 0.04 Hz.
  - **LF/HF (%)** — rácio da potência de LF a HF.