Faculdade de Ciências e Tecnologia

Departamento de Engenharia Informática

Ferramenta de visualização para melhorar a observação de aplicações de microsserviços

Joel de Sousa Fernandes

Proposta de dissertação no contexto do Mestrado em Engenharia Informática, Especialização em Engenharia de Software orientada pelos professores Filipe Araújo e Rui Pedro Paiva apresentada à Faculdade de Ciências e Tecnologia, Departamento de Engenharia Informática

Janeiro de 2020

1 2 9 0

UNIVERSIDADE D
COIMBRA

Esta página foi intencionalmente deixada em branco.

Resumo

O surgimento de arquitecturas em microsserviços procura combater a complexidade crescente das arquitecturas implementadas nos habituais monólitos, devido à separação do *software* em partes lógicas, ou seja, em vários serviços. No entanto, com a redução da complexidade pela separação em serviços, aumenta dificuldade em monitorizar o comportamento da aplicação devido à sua natureza distribuída.

Com o objectivo de analisar as actuais ferramentas existentes que possam solucionar este problema, foi realizada uma fase de estudo das mesmas. O estudo das aplicações existentes teve como conclusão o facto de, actualmente, as soluções existentes de visualização de grafos provenientes da recolha de *traces* serem pouco objectivas na visualização e/ou inexistentes. Outro problema é a compatibilidade das aplicações de monitorização de performance com o *tracing* de *OpenCensus* ou *OpenTracing*, as especificações de *tracing* existentes nos dias de hoje. Surge como consequência a necessidade de desenvolver uma aplicação que solucione as lacunas anteriormente descritas.

A solução desenvolvida procura resolver o problema da fácil visualização de informação relativa a arquitecturas distribuídas, mais precisamente, da informação recolhida através dos *traces*. A entrada de dados da aplicação é um conjunto de informação dos *traces* recolhidos, anteriormente processados pelo cliente. Como resultado, é fornecida ao utilizador uma visão geral da informação recolhida nos *traces* sob a forma de grafos de dependências de serviços.

Palavras-Chave

Microsserviços, Sistemas distribuídos, Monitorização, *Tracing*, Grafos.

Esta página foi intencionalmente deixada em branco.

Abstract

The emergence of architectures in microservices seeks to fight the growing complexity of the architectures implemented in the usual monoliths, due to the separation of the *software* in logical parts, that is, in several services. However, with the reduction of complexity due to the separation in services, it becomes more difficult to monitor the application behaviour due to its distributed nature.

In order to analyse the existing tools that can solve this problem, a study phase of these tools has been carried out. The study of the existing applications concluded that, currently, the existing solutions for visualization of graphs from the collection of *traces* are poorly implemented for the visualization of this graphs and/or non-existent. Another problem is the compatibility of performance monitoring applications with the existing *tracing* or *OpenCensus* or *OpenTracing* specifications. As a consequence, it is necessary to develop an application that solves the shortcomings described above.

The solution developed seeks to solve the problem of the easy visualization of information regarding distributed architectures, more precisely, of the information collected through *traces*. The application data input is a set of information from the *traces* collected, previously processed by the client. As a result, the user is provided with an overview of the information collected in the *traces* in the form of graphs of service dependencies.

Keywords

Microservices, Dystributed Systems, Monitoring, Tracing, Graphs.

Esta página foi intencionalmente deixada em branco.

Agradecimentos

Ao longo da dissertação de mestrado algumas pessoas foram importantes na orientação e na tomada de decisões, por essa razão quero agradecer às pessoas que me ajudaram durante este percurso. Ao professor assistente na Universidade de Coimbra e orientador da tese, Filipe Araújo, que contribuiu com o seu vasto conhecimento em sistemas distribuídos nas decisões tomadas e na orientação do trabalho a realizar ao longo das semanas. Também ao professor e arquitecto chefe das operações *cloud* inteligentes na *Huawei Technologies*, Jorge Cardoso, foi importante durante as reuniões semanais. Ao engenheiro Jaime Correia, estudante de doutoramento no Departamento de Engenharia Informática foi uma importante ajuda ao longo das semanas, principalmente nas decisões sobre a definição dos requisitos do projecto. Para além das referidas pessoas, quero também agradecer ao doutorando André Bento que contribuiu no trabalho desenvolvido com o seu conhecimento proveniente do seu trabalho de investigação da sua dissertação de mestrado.

Quero expressar a minha mais profunda gratificação aos meus pais e irmãs por sempre me apoiarem no meu percurso académico, cada um deles desempenhou um papel importante na minha vida até ao presente momento.

Quero agradecer a todos os meus amigos por me acompanharem nesta jornada, aos amigos do curso de Engenharia Mecânica que me acompanharam durante todo o estágio, um muito obrigado. Para além das referidas pessoas, também quero agradecer de forma especial ao colega António Sequeira por me apoiar numa jornada a ele paralela.

Quero também expressar a minha gratidão ao INCD - Infraestrutura Nacional de Computação Distribuída, por fornecerem acesso ao seus recursos computacionais.

A Todos, Muito Obrigado!

Esta página foi intencionalmente deixada em branco.

Conteúdo

Esta página foi intencionalmente deixada em branco.

Lista de Figuras

Lista de Tabelas

Esta página foi intencionalmente deixada em branco.

Capítulo 1

Introdução

Este documento apresenta o relatório da T*ese de Mestrado* em *Engenharia Informática* do aluno *Joel de Sousa Fernandes* durante o ano lectivo de 2018/2019, tendo lugar no *Departamento de Engenharia Informática* da *Universidade de Coimbra*.

O trabalho realizado ao longos dos dois semestres da dissertação de mestrado em Engenharia informática foi parcialmente realizado no âmbito do projto PTDC/EEI-ESS/1189/2014 — Data Science for Non- Programmers, apoiado por COMPETE 2020, Portugal 2020- POCI, UE-FEDER and FCT.

O trabalho desenvolvido na presente dissertação de mestrado surge da necessidade de desenvolver uma plataforma de visualização de grafos e de *workfows* no contexto da monitorização distribuída. Isto é, representar sob a forma de grafos as dependências entre os serviços numa aplicação desenvolvida numa arquitetura distribuída. As aplicações atualmente existentes que tentam resolver o problema da visualização da estrutura e dependências, para além de terem um preço de venda relativamente caro para o utilizador comum, a sua principal funcionalidade não é a apresentação deste tipo de informação. Em adição, o suporte deste tipo de ferramentas de *traces* não é totalmente garantido e a documentação relativa a este problema é demasiado vaga.

Existe, como consequência das lacunas anteriormente referidas, a necessidade de criação de uma plataforma que permita a visualização da informação proveniente dos *traces* de uma forma objetiva e intuitiva.

1.1 Motivação

As arquiteturas em monólito foram e continuam a ser o *standard* no desenvolvimento de software. Devido à complexidade crescente e de forma proporcional com o crescimento do número de linhas de código numa solução de *software*, surge a necessidade de separar o sistema em módulos lógicos, ou seja, a migração do sistema para uma arquitetura em microsserviços.

Este tipo de arquiteturas, arquiteturas distribuídas, solucionam alguns dos problemas relacionados com a complexidade das arquiteturas de monólito devido à fragmentação da mesma. Apesar de existir uma melhoria significativa no desenvolvimento, fruto da repartição do sistema, existe um *tradeoff* com a capacidade de observar o seu comportamento. A rápida evolução, desde a introdução do conceito de microsserviços em 2011, da aceitação do mercado às arquiteturas deste tipo, até aos dias atuais, não permitiu a evolu-

ção e desenvolvimento de ferramentas verdadeiramente capazes de monitorizar e apresentar o estado da performance de software distribuído.

As soluções existentes para a monitorização de software implementado segundo arquiteturas distribuídas baseiam-se na análise dos tradicionais *logs*, na recolha de métricas como a utilização do processador ou de memória. No entanto, para a identificação da raiz de um problema numa arquitetura distribuída, é necessário uma analise em recursão à sequência de ações desencadeadas ao longo do sistema por um pedido.

A tecnologia de *tracing* fornece competências aos utilizadores para uma analise deste género, mantendo um identificador nos pedidos *HTTP* realizados entre serviços é possível refazer o caminho de trabalho realizado pelo sistema para responder a uma determinada acção.

O rastreamento da sequência de pedidos realizados entre serviços através de *tracing* apresenta uma importância bastante elevada pois permite o acompanhamento das respostas desencadeadas por um determinado pedido e consequentemente uma análise detalhada do comportamento do sistema.

As aplicações existentes com suporte a monitorização de sistemas distribuídos não garantem uma análise aos traces do sistema e a documentação relativa à sua compatibilidade com os mesmos é escassa e não objetiva. Para além disso, as ferramentas não são de código aberto e as mais populares apresentam um preço para a sua subscrição bastante elevado que pode limitar o acesso às mesmas por parte da maioria dos interessados.

Além da informação escassa sobre a utilização de *traces* nas aplicações atuais de monitoria de performance, a visualização da estrutura da arquitetura neste tipo de aplicações é confusa e pouco objectiva. Para além deste inconveniente, o acompanhamento e visualização da sequência de pedidos ao longo dos vários serviços, desencadeados por algum tipo de ação na aplicação não assume um papel principal na observação do comportamento da aplicação das ferramentas atuais de monitorização de performance.

Desta forma surge a necessidade de desenvolver uma ferramenta de código aberto e que tenha o seu foco na objectividade na análise de traces e na sua amostragem, ou seja, a apresentação das dependências dos serviços sob a forma de um grafo de apresentação do caminho de pedidos desencadeado por uma acção na plataforma. A ferramenta que surge com esta necessidade, obtém a informação sobre os *traces* de uma aplicação e após algum tipo de processamento, é apresentada ao utilizador a informação relativa aos mesmos, sob a forma de grafos e de árvores.

1.2 Objectivos

O principal objetivo da dissertação de mestrado é desenvolver uma aplicação que forneça uma capacidade de observar a informação relativa à arquitetura de um sistema do utilizador de uma forma intuitiva e objetiva, para além do desenvolvimento, surge também a necessidade de analisar as soluções com objectivos semelhantes existentes atualmente. Desta forma,os objetivos da dissertação podem-se dividir nos seguintes:

1. Criação automática de um grafo. Um dos aspectos mais importantes do estágio é a construção de grafos. A construção de um grafo intuitivo pode ser complexa dependendo do número de nós, desta forma, um dos objectivos principais é obter capacidade de construir qualquer grafo de forma automática. A construção de um

grafo é a mote para o desenvolvimento de todas as funcionalidades, desta forma, este objectivo assume um papel de especial prioridade.

2. Gerir o problema da visualização de elevada quantidade de informação num grafo. A quantidade de informação presente num grafo, isto é, o número de nós e ligações entre nós é de extrema importância para a legibilidade do mesmo. Desta forma, um dos objectivos do estágio é desenvolver uma solução lógica para minimizar este problema e melhorar a qualidade da experiência do utilizador.

3. Um dos possíveis problema a solucionar é o tempo de construção de um grafo. A construção de grafos com número elevados de nós, por exemplo 100 nós, é lenta. Desta forma, surge como objectivo do estágio minimizar o tempo de construção dos grafos. O problema anteriormente descrito pode ser resolvido de forma mutua com o problema da visualização de um elevado número de nós, uma solução conjunta emerge como objectivo comum.

4. Fornecimento de informação à plataforma de visualização. O pré-requisito do desenvolvimento da plataforma de visualização é o fornecimento de informação para testar a plataforma bem como suportar o seu desenvolvimento. Assim, o desenvolvimento de *software* que resolva este problema é um dos principais objectivos.

5. Desenvolvimento de uma plataforma interactiva. Como forma de cativar e facilitar a utilização da plataforma por parte dos utilizadores, é necessário garantir a interatividade da mesma.

1.3 Planeamento

Nesta secção o planeamento do trabalho realizado é apresentado bem como as técnicas utilizadas para a orientação do trabalho ao longo do semestre. Todos os evolventes na dissertação e na tomada de decisões são identificados e a sua contribuição é explicada.

A orientação do trabalho foi conduzida pelas reuniões que foram realizadas a cada duas semanas. As reuniões eram atendidas por mim mesmo, autor do presente documento, pelo professor e orientador Filipe Araújo, também pelo professor e arquiteto chefe das operações *cloud* inteligentes na *Huawei Technologies*, Jorge Cardoso, participa nas reuniões por chamada, pelo engenheiro e estudante de doutoramento Jaime Correia e André Bento e pelos meus colegas António Sequeira e Cristiano Campos.

O tempo disponível e expectável para ser aproveitado para trabalhar na tese durante o seu primeiro semestre é de 16 horas por semana, tempo correspondente aos 12 créditos ECTS. A primeira reunião e que iniciou o trabalho da tese ocorreu a 5 de fevereiro de 2019, o fim do primeiro semestre da tese corresponde à data de entrega do presente documento, a 2 de Julho de 2019. Somando as semanas correspondentes a este intervalo, obtêm-se o tempo teórico do trabalho do primeiro semestre da dissertação de 336 horas.

O segundo semestre de dissertação corresponde a 30 créditos ECTS, ou seja, a 40 horas de trabalho por semana. Durante este semestre, o planeamento das reuniões manteve-se em relação ao primeiro semestre, desta forma, o segundo semestre teve o seu inicio após a defesa intermédia o seu término ocorreu na data da defesa da dissertação.

1.3.1 Primeiro semestre de estágio

O diagrama de *Gant* da distribuição dos trabalhos ao longo do primeiro semestre do estágio é apresentado na figura 1.1.

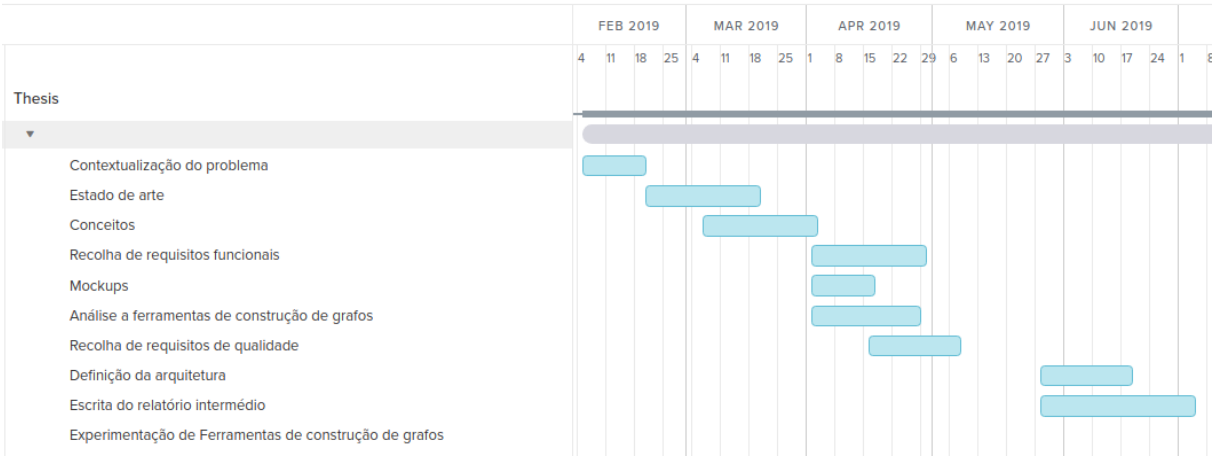

Figura 1.1: Plano de trabalhos do primeiro semestre da dissertação

O projeto começou com uma primeira reunião para a contextualização do mesmo e para a partilha de documentos e temas a pesquisar. Após duas semanas de contextualização, com a leitura de teses relacionadas com o tema, a seguinte reunião serviu para introduzir o tema das aplicações de monitoria de performance e iniciar a sua análise. Nos meses seguintes teve lugar a análise ao estado de arte das ferramentas de monitorização de performance de soluções de software. Quase em simultâneo foi também estudado a teoria necessária para o projeto e para a escrita do relatório intermédio. O estudo dos conceitos e das ferramentas de monitoria permitiu uma vaga definição dos possíveis requisitos funcionais da futura plataforma. Nos 2 meses seguintes foram abordadas as questões técnicas, a definição dos requisitos funcionais e de qualidade, a construção de protótipos de baixa fidelidade e a análise de ferramentas de visualização de grafos. A análise destas ferramentas surgiu com o objectivo de facilitar o trabalho do segundo semestre relativo a esta questão. No último mês disponível a arquitectura relativa à plataforma foi estudada e documentada, simultaneamente também decorreu a escrita do relatório intermédio.

1.3.2 Segundo semestre de estágio

O diagrama de *Gant* da distribuição dos trabalhos ao longo do segundo semestre do estágio é apresentado na figura 1.2.

Durante o segundo semestre do estágio foram desenvolvidas as funcionalidades identificadas durante a analise aos requisitos.

Numa fase inicial do semestre as ferramentas de construção de grafos foram testadas, como suporte à tomada de decisão na escolha de uma das ferramentas. Após a escolha de uma ferramenta de auxilio na construção de um grafo, durante as semanas seguintes foi adaptada a ferramenta para a construção de um qualquer grafo relativo à informação disponível na plataforma de visualização. Após a modulação da construção de grafos, foi necessário desenvolver uma ferramenta de criação de informação aleatória para ajudar no desenvolvimento da ferramenta de visualização. A funcionalidade de criação de um grafo em conjunto com a disponibilização de informação aleatória à ferramenta de visualiza-

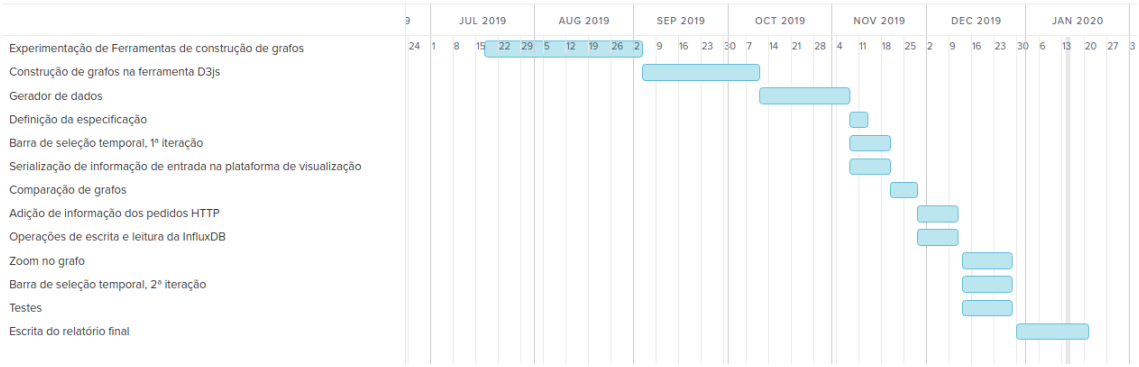

Figura 1.2: Plano de trabalhos do primeiro semestre da dissertação

ção permitiram o desenvolvimento de uma série de funcionalidades analisadas durante o documento. Na fase final do estágio, foi reservado um mês para a escrita do relatório final.

1.4 Organização do documento

A estrutura do seguinte documento e apresentada de seguida, para cada secção é resumido o seu conteúdo principal.

O primeiro capítulo, a introdução, apresenta o problema relacionado com dissertação. A motivação para a sua resolução, na secção de motivação. Na secção dos objetivos e no planeamento é abortado como o problema de irá resolver ao longo dos semestres e quais as técnicas utilizadas para orientar ao trabalho.

Os conceitos fundamentais, apresentados no segundo capítulo, introduzem conceitos abrangidos durante o primeiro semestre da dissertação, a explicação dos mesmos é essencial para a compreensão do restante do documento. Neste capítulo é abordada a virtualização em *containers*, as arquiteturas em microsserviços e uma comparação com os tradicionais monólitos, tecnologias de implementação de microserviços em *containers* e por fim técnicas de monitorização deste tipo de arquiteturas.

O terceiro capítulo é o estado de arte das aplicações de monitoria de performance e de ferramentas de visualização e de construção de grafos. No primeiro ponto deste capítulo são analisadas as aplicações de monitoria de performance para estabelecer as principais funcionalidades já implementadas e lacunas possíveis de existir. Pode-se estabelecer uma divisão nesta secção do capítulo, numa primeira parte, as características gerais da ferramentas são descritas. Numa fase posterior, algumas das principais e mais utilizadas aplicações são analisadas mais em pormenor. O segundo ponto ao qual este capítulo incide é a construção e visualização de grafos, neste são abordadas algumas técnicas de construção de grafos. De seguidas são analisadas algumas aplicações que fazem uso destas técnicas e oferecem uma solução de construção e visualização de grafos aos utilizadores.

No quarto capítulo é apresentada a proposta de solução que foi desenvolvida no segundo semestre da dissertação. O capítulo da solução compreende uma descrição do trabalho a ser aproveitado da tese de mestrado do colega André Bento, a identificação dos requisitos funcionais e de qualidade, a arquitetura proposta e também são apresentados os protótipos de baixa fidelidade desenvolvidos. Os requisitos funcionais da plataforma são associados a um cenário e priorizados de seguida. Os requisitos não funcionais são apresentados sob a forma de uma árvore de utilidade e para cada são descritas algumas decisões

ou informações importantes relativamente ao mesmo. Para cada protótipo é explicado o seguimento dos passos do utilizador e os requisitos representados em cada.

O quinto capitulo do documento é referente à fase de desenvolvimento do trabalho prático. Neste, são analisadas as escolhas realizadas ao longo do segundo semestre de estágio. O capítulo é iniciado por uma pequena introdução ao trabalho prático. Após a introdução ao trabalho, é analisada a especificação de dados utilizada ao longo de todas as componentes do trabalho prático. Para além do anteriormente referido, são também apresentadas as duas principais partes constituintes do trabalho prático. Por fim, são analisadas as funcionalidades principais desenvolvidas com recurso a ambas as partes do trabalho.

Capítulo 2

Conceitos Fundamentais

Nesta secção os conceitos teóricos relacionados com o trabalho realizado durante o primeiro semestre da dissertação são apresentados.

Os tópicos de seguida abordados são os seguintes: containerização, microsserviços, *logging*, monitorização e *tracing*.

2.1 Containerização

Um *Container* é "um sistema operativo leve em execução num sistema operativo de um host, corre instruções nativas para o CPU, eliminando a necessidade da emulação das instruções e compilação ao momento" [1]. Um *container* é um conjunto de software que permite encapsular uma aplicação e as suas dependências de modo a que seja possível abstrair o *hardware* necessário à execução da aplicação, deste modo, é reduzida a dependência do ambiente de execução da aplicação permitindo a sua execução em ambientes de *software* e *hardware* diferentes [2].

Os *Containers* aparecem como uma alternativa às máquinas virtuais porque a ideia fulcral deste tipo de virtualização em *containers* é fornecer isolamento e gestão de recursos num ambiente *linux* em semelhança a uma máquina virtual. No entanto a execução de aplicações em *containeres* não adiciona o *overhead* típico de máquinas virtuais. Das diversas características dos *containers* podem destacar-se as capacidades de:

- Fornecer um *runtime* portátil e bastante leve [3].
- Desenvolver, testar e fazer o *deployment* de aplicações para um número elevado de servidores [3].
- Carregar múltiplas aplicações distribuídas devido à partilha do *OS kernel*. [4].

Em últimas distribuições do linux está implementada a solução de conteinerização *Linux Container Virtualization*, isto é, uma tecnologia de virtualização que garante a criação de *containers* num sistema operativo *Linux*, o isolamento e gestão de ficheiros de um *linux container* e a virtualização do sistema operativo devido à partilha do *kernel* do sistema operativo do *host* [5]. As principais características de *Linux containers* são [6]:

- A cada container é atribuído um *PID* único, cada *container* pode correr um único serviço.

- Isolamento de recursos.
- Isolamento da rede.
- Isolamento do sistema de ficheiros, cada *container* tem um sistema de ficheiros privado.

2.1.1 Containers vs Virtualização

Uma típica virtualização de um sistema operativo requer a utilização de um elevado armazenamento para guardar o seu sistema de ficheiros e por norma usa um único processo no seu *host*. Devido, à necessidade de guardar imagens do sistema operativo completo da máquina virtualizada e de todos os seus binários e ficheiros necessários, a utilização de uma virtualização tradicional requer uma grande alocação de espaço no disco do *host* [3]. Como consequência da necessidade de guardar uma grande quantidade de dados o arranque de uma máquina virtual é lento.

Devido às semelhanças entre *Containers* e máquinas virtuais, a comparação entre os dois paradigmas é apresentada na tabela 2.1, as fontes da informação para cada parâmetro da tabela 2.1 são [1] [5].

Parâmetro	**Máquina virtual**	***Container***
Performance	Máquinas virtuais sofrem de algum *overhead*, devido á necessidade de tradução das instruções do *Gues OS* para o sistema operativo nativo.	Performance idêntica comparando com o *Host OS*.
Guest OS	Cada máquina virtual corre no seu *hardware* virtual e o *Kernel* é carregado na sua região de memória.	Todos os *guests* usam o mesmo *kernel*, a imagem do *kernel* é carregada na memória física.
Isolamento	A partilha de dados entre *guests* não é possível, é necessário a inicialização do sistema no seu próprio *kernel* e espaço de utilizador, assegurando o isolamento efetivo ao nível do sistema e do utilizador.	Subdiretorias podem ser partilhadas, não assegura isolamento efetivo ao nível do sistema.
Armazenamento de dados	Utilizam mais memoria, porque é necessário carregar o *OS kernel* e programas associados.	Utilizam menos quantidade de memória, devido à partilha do sistema operativo base.
Comunicação	Comunicação através de *ethernet devices*	Sinais, *pipes, sockets*, etc.
Acesso ao *hardware*	Garante acesso direto ao *Hardware*	Não é possível um acesso direto ao *hardware* numa solução de containerização.

Tabela 2.1: Comparação de Container e Máquina virtual

2.1.2 Orquestradores de Containers

Num ambiente de conteinerização múltiplos serviços são hospedados por diferentes *containers* em *clusters* de máquinas físicas ou virtualizadas. Este isolamento dos serviços tem como consequência a necessidade de um orquestrador garantir a unidade do sistema. Os orquestradores de *containers* simplificam o *deployment* e manutenção de um conjunto de *containers* porque os abstrai num único sistema, garantindo a sua disponibilidade, escalonamento e direcionamento de tráfego [7]. O *depoyment* de um novo container é auxiliado pelo orquestrador que planeia a sua implementação e encontra o *host* mais apropriado de acordo com os seus requisitos [8]. Uma funcionalidade extra de um orquestrador de *containers* é providenciar uma interface amigável para a configuração do *deployment* do sistema e para a sua manutenção. A interface representa todos os aspetos relativos ao conjunto de serviços, como as suas dependências, os ficheiros necessários para cada *container* e configurações respectivas.
Para além de providenciar uma interface para o *deployment* e manutenção dos serviços um orquestrador de *containers* também assegura as seguintes funcionalidades:

- Garante a descoberta de serviços.
- Balanceia a carga para cada *container*.
- Garante a disponibilidade dos *containers*.
- Assegura o realocamento de *containers* entre *hosts* na eventual falta de recursos.
- Expõe os *containers* a serviços externos ao sistema.
- Monitoriza a saúde dos *containers* e de *hosts*.

Analisando a solução de containerização *Docker* é possível distinguir os tipos de arquiteturas de virtualização. *Containers* implementados em *Docker engine* usam *lunix containers*, criando diferentes ambientes de sistemas operativos dentro do mesmo *kernel* do *host* e apenas os binários, bibliotecas e outros componentes necessários para cada *container* são executados separadamente [9].

Analisando uma solução de conteinerização em *docker*, a arquitetura de *Docker* é uma composição de camadas sobrepostas pois esta monta o sistema de ficheiros do *boot* em modo de leitura, em vez de mudar o sistema de ficheiros para modo de escrita são usados vários sistemas de ficheiros sobrepostos em modo de leitura com a exceção do último que estará aberto a leitura e a escrita. A arquitetura descrita anteriormente baseada em vários *sistema de ficheiros* sobrepostos permite a criação de várias imagens pela sobreposição de várias camadas que podem ser combinadas, reaproveitadas e partilhadas pela comunidade.

Kubernetes é uma solução de orquestração e implementação de *containers*, que permite a implementação e o escalonamento de aplicações hospedadas em múltiplos *containers*. *Kubernetes* foi desenvolvido pela *Google* como sucessor do *Google Borg*, esta é uma ferramenta de código aberto e tornou-se uma ferramenta de orquestração de *containers* bastante popular. *Kubernetes* permite a orquestração não só de *docker containers* mas, também de outro tipo de *containers* implementados de acordo com o *standard Open Container Initiative* [10].
A arquitetura do *kubernetes* pode dividir-se nos seguintes componentes [11]:

- *Cluster* que é um conjunto de nodos com pelo menos um *Master*.

- *Master* é o cérebro do *cluster*, num nodo *master* está o servidor *API* principal que mantém serviços *web* de acesso ao *cluster* e consulta do estado da carga de pedidos.
- *Node*, em cada nodo existe um conjunto de componentes que garantem a atualização do estado do nodo, balanceamento de pedidos e direcionamento de tráfego.
- *Pods* são constituídos por um ou mais *containers* que são capazes de partilhar recursos. A cada *pod* é atribuído um endereço *IP* dentro do *cluster*, permitindo uma correta comunicação da aplicação respetiva.

Para um melhor entendimento da divisão de nós na arquitetura de kuberntes é apresentada a figura 2.1. Na figura é representado um *cluster* de *containers*, os n *container* existentes estão ligados ao nó *master*.

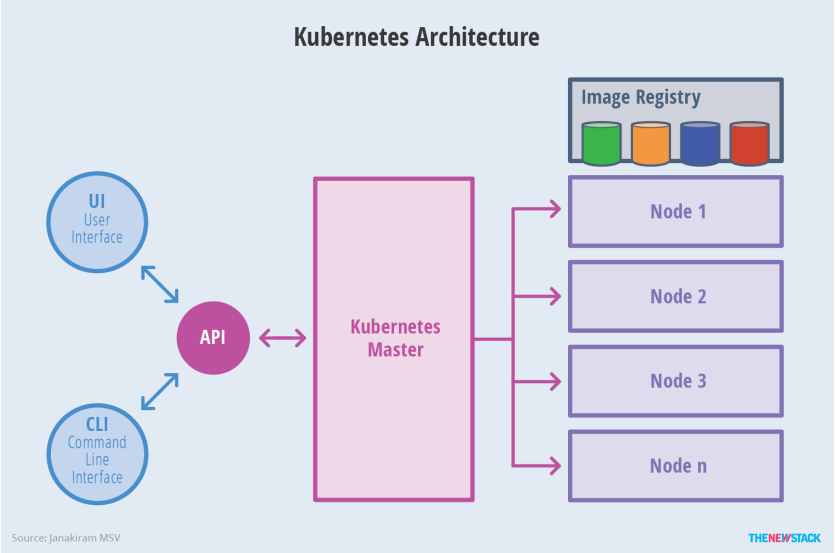

Figura 2.1: Arquitetura de kubernetes [12]

2.2 Microsserviços

O interesse na palavra "Microservices" tem vindo a aumentar desde a introdução do conceito em 2011 numa conferencia para arquitetos de *software* com o objetivo de explicar a arquitetura adotada pela *Netflix* e *Amazon* [13]. A figura 2.2 representa o interesse pela palavra "Microservices" deste 2013 até maio de 2019 segundo [14].

Microsserviços são aplicações de tamanho reduzido focadas em fazer apenas uma tarefa corretamente. O tamanho de um microserviço está dependente da sua funcionalidade respetiva, cada microsserviço apenas pode realizar uma funcionalidade e por isso é boa prática ter apenas algumas centenas de linhas de código em cada. A principal característica de um microsserviço é a sua independência, uma vez que cada aplicação numa arquitetura em microsserviços é independente e comunica com as restantes do mesmo sistema, através de algum serviço de mensagens [15] [16].

A razão pela escolha de uma arquitetura em microsserviços é baseada na independência de cada aplicação, isto possibilita que cada parte do sistema seja independente e consequentemente um melhor *deployment* do próprio sistema, melhor isolamento de problemas e melhor testabilidade.

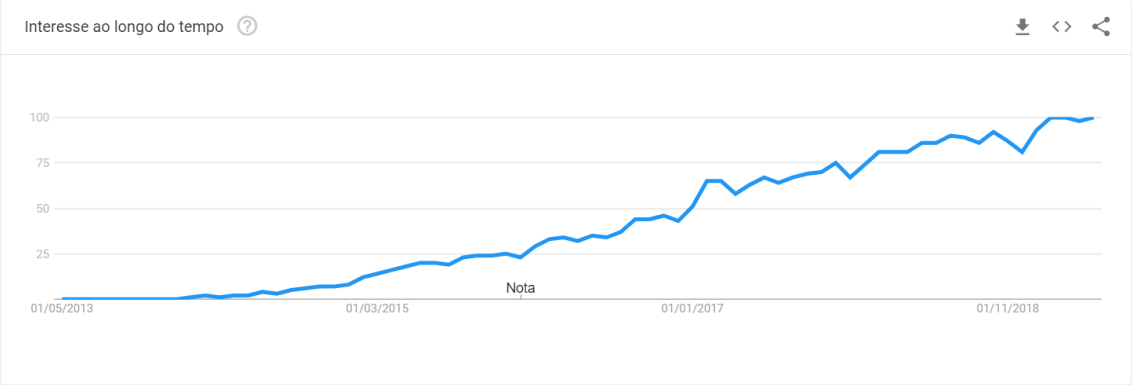

Figura 2.2: Tendência da palavra "Microservices"

As ideologias reflectidas numa arquitetura em microsserviços são bastante semelhantes às de uma arquitetura baseada em serviços, *Service-oriented architecture* (SOA), porque esta surge como uma evolução de *Service-oriented-architecture* na tentativa de resolver alguns problemas existentes.

A principal inovação que surge nesta arquitetura em relação às tradicionais, *Layered,Client/Server, etc,* é forma como as funcionalidades são disponibilizadas. Numa arquitetura SOA as funcionalidades são disponibilizadas sob a forma de serviços o que permite desenvolver serviços mais independentes e como consequência o seu não acoplamento e futura reutilização. Na figura 2.3 [17] é representada uma arquitetura *SOA* e tipo de arquitetura podem ser identificadas 3 camadas lógicas: Serviços produtores, serviços consumidores e camada de orquestração. Um pedido para um serviço produtor é manipulado na camada de orquestração e encaminhado até ao serviço produtor num *Enterprise-service-bus*, que assegura a comunicação entre todos os serviços [18].

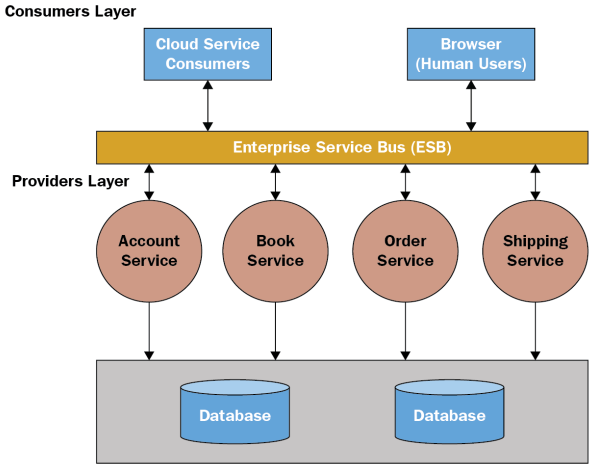

Figura 2.3: Arquitetura de SOA

Apesar das vantagens que possam existir ao adotar uma arquitetura em microsserviços, existem inúmeros problemas que é necessário ultrapassar. Sam Newman diz: "...I should call out that microservices are no free lunch or silver bullet, and make for a bad choice as a golden hammer. They have all the associated complexities of distributed systems...". Um dos principais desafios é a integração dos microsserviços, pois cada microsserviço pode usar diferentes tecnologias (REST, HTTP, protocol buffers, JAVA RMI) e diferentes formatos

de partilha de dados o que pode desencadear uma assincronia no sistema. Devido a esta situação é necessário um cuidado extra no modo como os microsserviços de adaptam e comunicam na rede de microsserviços existentes.[16]

2.2.1 Solução de problemas de monólitos com microsserviços

A arquitetura em microsserviços combate as falhas que um grande monólito apresenta. Um monólito é uma aplicação de software em que os seus módulos não podem ser executados independentemente.

As diferenças entre arquiteturas baseadas em monólito e microsserviços são bastante notórias, para um melhor entendimento sobre as diferenças entre ambas as arquiteturas são apresentadas as duas arquiteturas na figura 2.4.

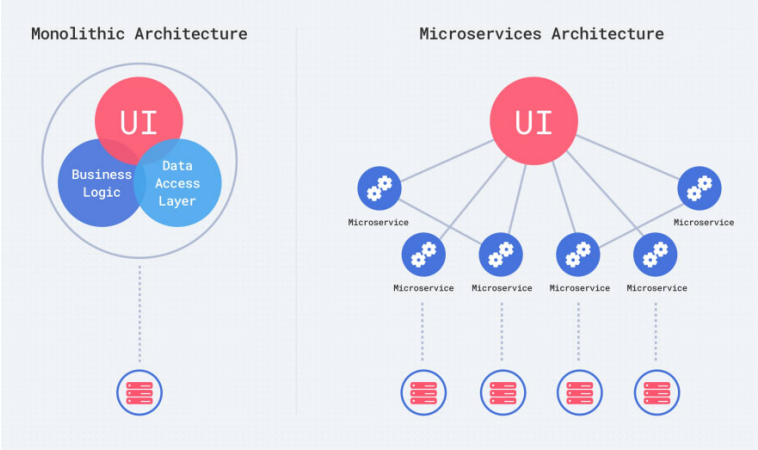

Figura 2.4: Arquitetura de monólito e de microsserviços [19]

Uma arquitetura de um típico monólito é caracterizada pela unidade, isto é, as camadas lógicas do software funcionam em conjunto e expõem as suas funcionalidades encobrindo a estrutura do software, por sua vez, uma *user interface* faz uso dessas capacidades. Contrariamente, uma arquitetura baseada em microsserviços expõe e divide os módulos do *software* de acordo com as funcionalidades de negócio. De forma semelhante é aplicado este fenómeno às base de dados, como de pode ver na figura, numa arquitetura em monólito existe apenas uma base de dados, já numa arquitetura em microsserviços pode existir várias base de dados para serviços ou conjunto de serviços

O principal motivo para a escolha ou mudança para uma arquittura em microsserviços em vez de um tradicional monólito é o seu rápido crescimento não controlado e consequente desorganização [20]. A arquitetura em microsserviços é uma tentativa de solucionar os problemas relativos a um monólito, assim podem destacar-se as seguintes características dos microsserviços:

- Tamanho de cada serviço, para manter a granularidade típica deste tipo de arquitetura, cada serviço apenas pode desempenhar uma única capacidade de negócio e o seu tamanho não pode ser demasiadamente elevado de maneira a que se justifique a sua separação em dois serviços diferentes. O tamanho de cada microsserviço e as suas limitadas funcionalidades são uma base para as restantes características que tornam esta arquitetura adaptadas ao rápido crescimento e desorganização de *software*. [21]

- Rápido *deployment*, devido à sua estrutura repartida o *deployment* de um sistema baseado em microsserviços não impõe o seu total *deployment*, desta forma a implementação de cada serviço é independente das restantes

- Resiliência, um outro contra da utilização de uma arquitetura em monólito são as falhas de software que possam ocorrer devido à sua estrutura não repartida que é bastante frágil à propagação de erros, a utilização de microsserviços surge como uma solução para este problema. Uma das suas características é o isolamento de falhas que este tipo de arquitetura garante, ao contrário de uma arquitetura em monólito em que uma falha pode desencadear a falha total do *software*, numa arquitetura em microsserviços o isolamento que é característico de cada microsserviços garante que apenas parte do sistema é afetado, havendo um isolamento do erro [21]

- Flexibilidade, o isolamento de cada serviço permite acompanhar as mudanças do mercado e de tecnologias das *information technologies*. Em cada microsserviço é possível escolher diferentes linguagens ou *frameworks* que melhor se adaptem às funcionalidades do serviço [20].

- Escalabilidade, segundo Susan J. Flowler, [22], uma arquitetura de microsserviços é escalável, de alto desempenho e consegue lidar com um grande número de pedidos por minuto.

- Modificabilidade, com a utilização de uma arquitetura em monólito pode surgir a dificuldade em alterar ou acrescentar partes do sistema sem que surjam erros consequentes desta alteração. A natureza de um monólito dificulta o seu *deployment*, após alguma alteração no código é necessário fazer o *deployment* total do sistema. Microsserviços combatem esta dificuldade devido à sua granularidade.

Apesar dos benefícios fornecidos pela implementação de uma arquitetura em microsserviços existem alguns problemas associados a este tipo de arquitetura, comparando com um tradicional monólito. Devido às suas características que promovem a independência dos serviços podem destacar-se as seguintes desvantagens:

- Complexidade, o maior contratempo na implementação de uma arquitetura de microsserviços é a sua complexidade. Apesar de se apresentar como uma solução para a complexidade crescente que um monólito pode sofrer, a migração para uma arquitetura em microsserviços é sujeita dificuldades devido ao isolamento de cada serviço. Um requisito para a integração dos serviços é o conhecimento prévio por parte dos serviços do esquema de transmissão e formatos de dados partilhados, é necessário um maior esforço no desenvolvimento da documentação de cada serviço para garantir este requisito. Existe também um maior número de processos no total da aplicação, devido ao desenvolvimento de software extra para o balanceamento e encaminhamento dos pacotes.

- Sincronia das base de dados, cada serviço pode ter acesso a uma base de dados própria e por esta razão é necessário garantir a sincronia dos dados nas várias bases de dados.

- Custos, a necessidade do isolamento dos serviços leva a uma maior gasto de recursos de hardware, como o tempo de cpu e uso da rede. Para garantir a baixa latência na resposta aos pedidos é exigido um maior gasto relativamente a uma arquitetura em monólito.

- Segurança, comparando com um monólito existe uma grande quantidade de pedidos entre os vários serviços do sistema, posto isto, o aumento do número de canais de comunicação de um sistema aumenta a probabilidade de uma intrusão.
- Testabilidade, a testabilidade de uma aplicação em monólitos é um tema bastante estudado e existem ferramentas que auxiliam na realização de testes em *black-box* e *white-box*. Devido à natureza recente do conceito de microsserviços e da realização de testes em sistemas distribuídos, para testar um sistema deste tipo é necessário recorrer a *logs*, ferramentas de monitorização e sobretudo a *tracing*. Na secção 2.3 é abordado o tema de monitorização de sistemas distribuídos em mais detalhe.

2.2.2 Implementação de Microsserviços em Containers

As principais características da virtualização em *containers* são o seu isolamento e a sua arquitetura que garante uma virtualização que não consome muitos recursos e consequentemente uma mais fácil e rápida implementação de um serviço.

Devido à arquitetura repartida de uma solução que implemente o conceito de microsserviços, cada serviço desempenha um papel diferente, por conseguinte são necessários requisitos diferentes para o acesso e uso de *hardware*. Por exemplo, um serviço necessita de um tempo de processamento elevado, enquanto que um outro necessita de um maior acesso à rede devido ao elevado número de pedidos, a virtualização de *hardware* baseada em *containers* resolve esse problema combinando a distribuição da carga para os vários microsserviços em execução.

Outra das vantagens inerentes à implementação de microsserviços em *containers* é a possibilidade de uma *implementação* independente de serviços pelos vários containers disponíveis. O isolamento garantido por uma implementação em *containers* garante o não conflito pela utilização de linguagens, *frameworks* ou bibliotecas diferentes, prática típica numa arquitetura em microsserviços.

Uma solução de microsserviços requer um conjunto de operações para garantir a sua integridade e disponibilidade, isto posto, a sua implementação com recurso a *containers* e a um orquestrador para o mesmo, como o *Kubernetes*, garante:

- Gestão de dependências e configuração dos serviços
- Facilidade de escalonamento horizontal pela possibilidade de replicar serviços.
- Distribuição da carga pelos serviços e réplicas de serviços.
- Recuperação de um serviço em falha.
- *Logging* do serviço.

2.3 Monitorização

A maioria do tempo no ciclo de vida de um projeto de software é gasto na sua manutenção. Devido à elevada importância de uma correta manutenção do software, é importante conhecer algumas técnicas para a manutenção e correção de falhas num arquitetura distribuída como é o caso dos microsserviços. Uma correta manutenção do software está intimamente relacionada com a possibilidade de identificar a origem dos problemas.

Enquanto que num monólito a raiz de uma falha pode ser encontrada com uma série de testes unitários, numa arquitetura em microsserviços existe um maior problema na identificação da causa devido à estrutura repartida da mesma. Assim sendo é possível identificar três técnicas *white-box* distintas para a monitorização e observabilidade de uma aplicação distribuída: *logging*, *tracing* e monitorização. Nas secções seguintes serão apresentadas cada uma destas técnicas.

2.3.1 Logging

Logs são usados para representar aplicação durante a sua execução, as mensagens de *logs* guardam sobre a forma de texto a informação sobre a execução do programa. As mensagens de *logs* podem descrever os seguintes estados do sistema [23]:

- Erro: um ficheiro de *logs* pode apresentar uma falha não fatal para a normal execução.
- Erro fatal, o processo é encerrado após a mensagem de *logging*.
- Informação, *logs* podem guardar informações importantes sobre um acontecimento não considerado de falha.
- *Debug*, são guardadas informações em ambiente de execução de modo *debug*.

A informação guardada em ficheiros de logs pode ser a única fonte de informação para identificar uma falha passada, isto porque é extremamente difícil replicar o estado de um sistema com o objetivo da repetição da falha em estudo. Sendo assim, engenheiros de *software* utilizam as mensagens de *logs* para identificar as falhas existentes [23]. Contudo, o processo de obter e tratar *logs* é dispendioso em tempo e recursos, para além disto, é necessário filtrar a informação guardada em ficheiros de *logs* de modo a limitar a quantidade de informação guardada e consequentemente facilitar a leitura e identificação de uma falha por parte de um humano.

2.3.2 Monitorização

Monitorização é o processo de recolha de métricas sobre as operações de uma aplicação de software. As métricas recolhidas podem representar as condições de *software* ou *hardware*, sendo assim podem ser retiradas as seguintes métricas: códigos *HTTP* dos pedidos, taxa de utilização do *CPU* ou memória utilizada.

O conceito de monitorização pode ser associado à identificação de problemas de software recorrendo a *logging* ou *tracing*. No entanto, os três conceitos: monitorização, *logging* e *tracing* representam diferentes ideias. A Monitorização está ligada à atividade de diagnostico de problemas para alertar os desenvolvedores de um anormal funcionamento do sistema. Uma ferramenta de monitorização tanto pode representar o sistema em função de métricas recolhidas em *run-time*, como pode representar as tendências das métricas para um intervalo de tempo escolhido.

A Monitorização em tempo real recolhe métricas do estado recente e atual do sistema permitindo aos desenvolvedores uma rápida reacção aos problemas existentes.

A monitorização de dados que representem o comportamento do sistema num intervalo de tempo passado permitem a análise da variação das métricas com o objetivo de

identificar uma tendência de comportamento. Através desta análise de tendências é possível identificar possíveis falhas existentes e consequentemente melhorar a performance do sistema. Com o objetivo de facilitar e automatizar o processo de monitorização estão disponíveis ferramentas de monitorização automática. No capitulo 3 são analisadas algumas destas aplicações.

2.3.3 Tracing distribuído

A identificação da origem de uma falha que ocorra num software implementado numa arquitetura em microsserviços está relacionada com o rastreamento de pedidos entre vários *containers*. *Tracing* distribuído permite aos utilizadores fazerem o rastreamento de um pedido num sistema de software distribuído através de múltiplas aplicações, serviços ou base de dados em diferentes *containers*. Um pedido de um serviço pode despoletar uma série de outros pedidos a outros serviços e chamadas à base de dados. Numa arquitetura em monólito o rastreamento desse pedido é bastante directo. No entanto, para a identificação da causalidade e dependência de um serviço numa arquitetura em microsserviços, é necessário atribuir um identificador para que seja possível refazer a cronologia das transações causadas por um pedido.

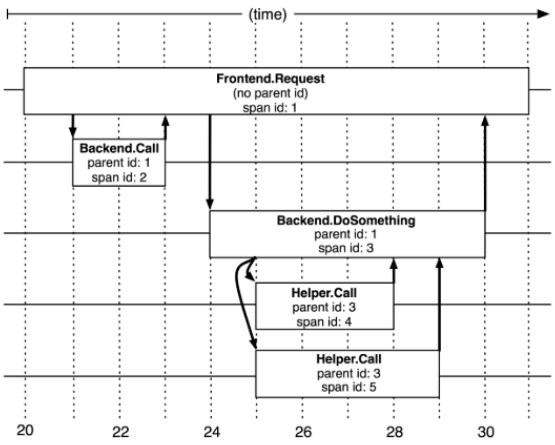

Figura 2.5: Árvore de *spans* de um *trace* [24]

Para o rastremento de um pedido, é atribuído um *id* único a essa transação, sendo chamada de *trace* e representa a transação completa. Por sua vez, cada transação pode ser composta por vários trabalhos a serem desempenhados, como um pedido a um serviço ou um pedido a uma base de dados. Por essa razão, cada *trace* é composto por vários *spans*.

A figura 2.5 representa a distribuição de *spans* para um único *trace*, este é composto por uma árvore de *spans*, a cada *span* é atribuído o *id* do seu respetivo trace, um *span* sem o *id* do pai é chamando de *root span*.

A instrumentação do código para a implementação de um sistema de *tracing* é realizada por determinadas ferramentas que especificam como preencher cada parâmetro dos *traces* e *spans*. OpenTracing [25] e OpenCensus [26] são *APIs* que estandardizam a implementação de *traces* numa aplicação. Os *traces* recolhidos de aplicações que usem os dois standars são representados visualmente em ferramentas como o Zipkin [27] ou Jaeger [28].

Cada *span* representa um estado do sistema e de acordo com OpenTracing, cada

span contem o seguinte conjunto de informações [29]:

- Nome da operação.
- Tempo de início e de fim do *span*.
- *Tags*, são anotações definidas pelo utilizador para permitir a consulta, a filtração e entendimento dos dados, por exemplo: “http:status_code"ou "db.instance”.
- *Logs*, são anotações usadas para guardar informações de *debug* extras às *tags*.
- Um *SpanContext*, é constituído por um conjunto de *ids* relacionados com *spans* vizinhos, *id* do pai ou *id* do *trace*.

A figura 2.6 é um *span* especificado segundo OpenTracing [29], o *span* é uma consulta a uma base de dados. Do *span* apresentado é possível retirar algumas informações, como as seguintes: a instância ao qual foi feito o pedido, a base de dados *mysql* acessível no porto 3306 do *host* local, a *query* da consulta, o *log* produzido e alguns identificadores relativos ao *trace* e *span*.

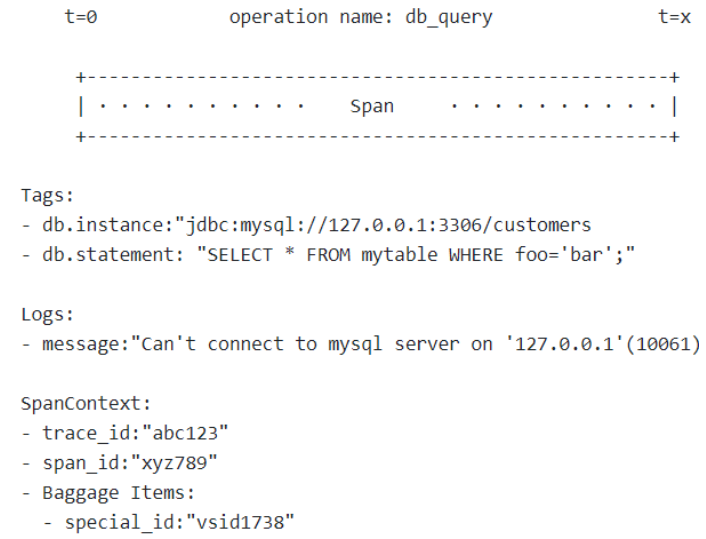

Figura 2.6: Span de *query* a uma base de dados

2.4 Grafos

No contexto da monitorização e *tracing* distribuído, a representação da estrutura dos serviços e das suas dependências é realizada a partir de grafos. Na verdade, os grafos são utilizados para representar problemas de diversas áreas de estudo. Devido à importância no contexto de monitorização de microsserviços em *containers*, os grafos são analisados de seguida.

Um grafo G é “ um par ordenado (V(G),E(G)) que consiste num conjunto V(G) de vértices e um conjunto E(G), disjunto de V(G), de arestas, ambos os conjuntos com uma função de incidência ψG, que associa cada aresta com um par não ordenado de vértices de G”.

Existem variâncias possíveis num grafo, as arestas podem ser não direcionais como no grafo G ou arestas com uma direção especifica. Para alem disso, as arestas podem ter

um peso e também existe a possibilidade de haver múltiplas arestas para o mesmo par de vértices.

Para o grafo H da figura 2.7 podemos definir:

$$G = V(G), E(G))$$
$$V(G) = v0, v1, v2, v3, v4, v5$$
$$E(G) = e1, e2, e3, e4, e5, e6, e7, e8, e9, e10$$

ψG é definido por:

$$\psi G(e1) = v1v2 \quad \psi G(e2) = v2v3 \quad \psi G(e3) = v3v4$$
$$\psi G(e4) = v4v5 \quad \psi G(e5) = v5v6 \quad \psi G(e6) = v1v0$$
$$\psi G(e7) = v2v0 \quad \psi G(e8) = v3v0 \quad \psi G(e9) = v4v0 \quad \psi G(e10) = v5v0$$

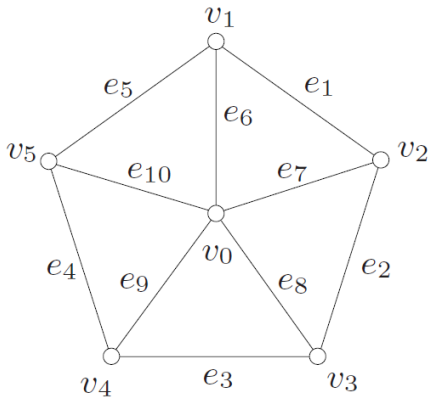

Figura 2.7: Grafo G

Um grafo de microsserviços representa diversas dependências entre os serviços. Um microsserviço é representado como um vértice do grafo e às suas arestas podem corresponder diversas métricas do sistema, como as seguintes:

- Número de pedidos durante um intervalo de tempo.
- Relação de dependência de dois serviços, se existir uma chamada entre dois serviços, a sua interdependência é representada por uma aresta entre os nós respetivos.
- Número de diferentes tipos de pedidos entre os dois serviços.

2.5 Base de dados de grafos

As bases de dados de grafos são bases de dados destinadas a armazenar informação com estrutura de um grafo. As tradicionais bases de dados relacionais são usadas como meio de armazenamento de informação de contextos em que a informação contida em cada entidade assume elevada importância. No entanto, existem variados contextos com diferentes prioridades no armazenamento de informação.

Um modelo de base de dados de grafos é aplicado em contextos em que a interligação entre as entidades tem uma importância semelhante ou superior ao seu conteúdo. Este tipo de base de dados foi desenvolvida com base na teoria de grafos e faz uso dos conceitos de nós, vértices e ligações.

Uma base de dados de grafos faz uso do modelo de grafo para armazenar e relacionar a informação, por conseguinte, cada entidade pode estar relacionada uma outra por um vértice. Este modelo permite descrever informação interrelacionada de um forma mais natural para o utilizador, uma vez que a estrutura dos dados é visível nas ligações e nós da base de dados [30].

Para além da estrutura de dados ser definida com o objectivo de optimizar o armazenamento de informação com entidades muito relacionadas, as bases de dados de grafos utilizam estruturas de armazenamento de grafos e algoritmos específicos para a realização de operações sobre a informação [30]. De entre as múltiplas bases de dados disponíveis podem-se destacar as seguintes:

- Neo4j
- Microsoft Azure Cosmos DB
- ArangoDB
- OrientDB

2.6 Base de dados de séries temporais

De acordo com o modelo de dados utilizado pelas base de dados de séries temporais, uma série temporal inclui dados sobre a variável de tempo e informação relativa à própria série temporal, por exemplo, um identificador da série temporal. Como exemplo de uma série temporal surge a variação temporal do preço de uma acção num dia, para este exemplo, a informação relativa à série temporal pode ser uma identificador da empresa, o seu nome, ou médias de valor da ação para um determinado intervalo de tempo.

Uma base de dados de séries temporais é construída especificamente para gerir métricas ou eventos que são relacionados com a variável tempo. Desta forma, uma base de dados que siga um modelo de séries temporais tenta otimizar a capacidade de medir e analisar as mudanças ocorrentes aos longo do tempo.[31]

A figura 2.8 representa o armazenamento de múltiplas séries temporais para o mesmo identificador temporal. Como de pode ser pelos valores de cada métrica, uma séries temporal é um conjunto de medições de uma métrica que, ao longo do tempo, representa a sua variação. Este tipo de estrutura de dados ajuda na leitura de dados de forma mais eficiente, em vez de fazer a leitura de cada métrica individualmente, como as métrias estão relacionadas pela variável tempo, é suficiente ler apenas um conjunto de dados identificados pela mesma variável. [32]

Como exemplos de bases de dados de séries temporais surgem os seguintes:

- InfluxDB
- *Prometheus*
- *RRDtool*

- *TimeScale*

Timestamp	Metric 1	Metric 2	Metric 3	Metric 4	Metric 5
2019-03-28 00:00:01	765	873	124	98	0
2019-03-28 00:00:02	5876	765	872	7864	634
2019-03-28 00:00:03	234	7679	98	65	34
2019-03-28 00:00:04	345	3	598	0	7345

Figura 2.8: Esquema de armazenameto de dados numa base de dados de séries temporais

Capítulo 3

Estado de arte

3.1 Ferramentas de monitorização de performance

As ferramentas de monitorização de performance, também conhecidas como "APMs" devido à sua tradução para inglês "*Application Performance Monitoring*", tem como objetivo a monitorização continua de uma aplicação, isto é, o acompanhamento continuo e automático da performance de uma aplicação, de forma a identificar tendências, anomalias e possíveis problemas. Para este objetivo, é essencial a recolha de métricas de utilização da aplicação, sendo a principal métrica recolhida por este tipo de ferramentas o tempo de resposta dos pedidos. Esta métrica permite analisar o estado de carga da aplicação e consequentemente analisar a experiência de utilização por parte dos utilizadores, experiência esta relacionada com disponibilidade e performance da aplicação. De acordo com as funcionalidades implementadas nas diferentes ferramentas, o resultado disponibilizado à equipa de desenvolvimento pode ser a apresentação das métricas de forma sintetizada para facilitar a sua análise, ou pode ser realizado um processamento extra com o objetivo de identificar as fontes de problemas.

As ferramentas de monitorização automática de performance têm como abreviatura "APM", a abreviatura pode corresponder ao termo "*Application performance monitoring*" ou "*Application performance measurement*", esta dualidade de definições das ferramentas conduz a duas diferentes interpretações. O termo "*Application performance monitoring*" está relacionado com a monitoria constante da aplicação. Por outro lado, o termo "*Application performance measurement*" está relacionado com a criação de métricas por parte das ferramentas. Ambos os termos podem-se aplicar à maioria das ferramentas de monitorização automática, dependendo do objetivo da implementação das mesmas e das principais características da própria ferramenta. Nas secções seguintes é documentada a revisão às principais ferramentas deste contexto.

Numa aplicação de monitorização de performance podem-se identificar 4 áreas de controlo de uma aplicação, sendo assim uma aplicação de controlo de performance divide-se em [33]:

- Monitoria *top-down*, a monitoria *top-down* é focada na experiência e utilização por parte dos utilizadores. Uma monitoria deste tipo pode fazer uso do espelhamento do tráfego nos portos da rede ou de mecanismos de rastreamento de transações na aplicação. A combinação das duas técnicas de monitorização da rede permitem avaliar a disponibilidade dos serviços e providenciar uma astração da saúde da aplicação durante períodos de carga elevada.

- Monitoria *Bottom-up*, a monitoria *bottom-up* também é conhecida como monitoria da infraestrutura, está relacionada com a recolha centralizada de métricas da aplicação e com a correlação de eventos.
- Gestão de falhas, a gestão de falhas é uma das principais funcionalidades de uma aplicação de monitoria de performance. A interligação da monitoria *top-down* e *bottom-up* permite a identificação das falhas na estrutura da aplicação.
- Recolha de reportamento de métricas, uma aplicação de monitoria de performance recolhe dados não processados e após algum tipo de processamento, quer seja com a finalidade de relacionar métricas ou de facilitar a apresentação das mesmas, são apresentados à equipa de desenvolvedores.

A figura 3.1 representa os 4 pilares de monitoria de aplicações de monitoria de performance anteriormente enumerados, a figura 3.1 foi apresentada por Larry Dragich.

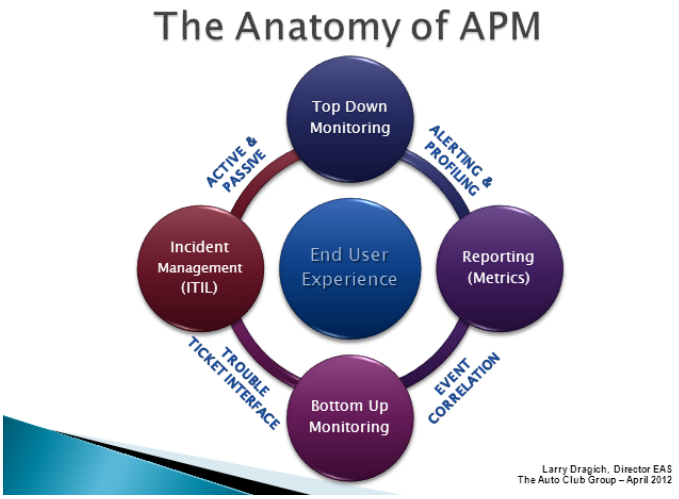

Figura 3.1: Anatomia de aplicações de monitoria de performance [33]

Analisando algumas aplicações de monitoria de performance disponíveis é possível definir algumas funcionalidades implementadas na maioria destas aplicações. As funcionalidades de seguida enumeradas fazem parte das tendências das várias funcionalidades disponíveis nas várias aplicações, são essenciais para garantir um correto *feedback* às equipas que usam este tipo de aplicações, para facilitar a manutenção do seu sistema. Desta forma as principais funcionalidades características de uma aplicação de monitoria de performance são [33]:

- Analisar a performance de pedidos e transações, a análise de performance de cada transação e pedido é a principal métrica necessária no controlo de uma aplicação. Esta métrica permite identificar que pedidos são mais frequentemente utilizados e também identificar um tempo de resposta alto aos mesmos.
- Analisar a performance de base de dados e de outras dependências, um atraso na resposta a um pedido por dever-se a dependências da aplicação, é essencial analisar os possíveis atrasos relativos a transações a dependências como a base de dados ou outro serviço externo.
- Identificar linhas de código correspondente a uma transação, a análise da aplicação ao nível do código identifica que métodos e funções têm um pior desempenho.

- Apresentar métricas de *hardware*, como utilização de cpu, memoria, etc.
- Permitir a coleta e apresentação e métricas personalizadas pela equipa de desenvolvimento.
- Gestão centralizada de *logs*, após a identificação da causalidade de uma falha com o auxílio da ferramenta, os *logs* relativos ao intervalo de tempo correspondente podem conter informações essenciais para a resolução da falha.
- Alerta de falhas da aplicação, uma atempada alerta das falhas encontradas pela aplicação permite reduzir o tempo necessário para recuperar ao estado normal do sistema.

Devido à natureza comercial das várias aplicações analisadas, a sua revisão é baseada em revisões alheias e em vídeos tutorias encontrados *online*. O preço de aquisição de um serviço de monitoria automática é baseado no número de servidores, na quantidade de informação e no número de serviços da arquitetura. Sendo assim, e mais uma vez baseada na revisão das ferramentas por terceiras pessoas, o preço varia entre os 20 dólares por mês até rondar os 3500 dólares por mês. Os preços praticados pelas ferramentas de monitoria automática, juntamente com a falta de uma arquitetura de testes implementada num serviço *cloud* como *Amazon EC2*, dificulta o teste das ferramentas de forma a validar a informação encontrada *online*. As principais fontes de informação existentes são os *websites* criados pelas mesmas, por essa razão é necessário validar as funcionalidades através de fontes de informação independentes, como vídeos independentes de revisão das ferramentas.

3.2 New Relic

New Relic é uma conhecida aplicação de monitoria de performance existente desde 2008 e é bastante utilizada em diversos países. *New relic* analisa em tempo real dados de utilização dos serviços *web* e não *web*, isto é, monitoriza a experiência de utilização do sistema em monitoria bem como as camadas lógicas que suportam o *front-end* disponível aos clientes [33]. Apesar das funcionalidades concedidas pela aplicação, estas estão acessíveis mediante o pagamento do seu serviço. As principais funcionalidades são as seguintes [33]:

- Recolha de métricas de tempo de resposta, taxa de transferência, taxas de erros, etc.
- Análise da performance de serviços externos.
- Análise das transações, o utilizador pode fazer pesquisas sobre o tempo de cada transação.
- Histórico de *deployments* e comparação entre os mesmos.
- Suporte a aplicação moveis.

Das características da aplicação podem-se destacar a simplicidade em instalar e implementar a aplicação e a apresentação dos dados intuitiva.

3.3 DynaTrace

DynaTrace foi lançada em 2006, é uma das mais conhecidas e usadas aplicações de monitoria de performance. Para além das funcionalidades de monitoria de sistemas

implementados de acordo com uma arquitetura em monólito, a aplicação *DynaTrace* permite a monitoria de aplicações distribuídas em serviços *cloud* [33]. O desencadeamento da monitoria de um sistema por parte de *DynaTrace* é a implementação de um agente que automaticamente descobre os serviços e ambiente de execução da aplicação em análise. As principais funcionalidades são as seguintes [33]:

- Monitorização ao nível do código, permite identificar as funções e partes do código envolvidas numa falha ou atraso na resposta a um pedido.
- Suporte a *.NET* e *Java*.
- Resolução de problemas proativa e sem envolver o utilizador.
- Automática descoberta do ambiente do sistema.

As características realçantes do ponto de vista dos utilizadores são: fácil instalação e implementação da ferramenta e a criação de um ambiente de trabalho simples e intuitivo.

3.4 Appdynamics

AppDynamics é uma aplicação de monitoria de performance de soluções de software existente desde 2008 e faz parte da *Cisco software business*. Os serviços disponibilizados por a aplicação são agregados em 3 pacotes distintos e comercializados através de uma mensalidade, os três pacotes de serviços são: Monitorização da experiência do utilizador fina (EUEM), aplicação de monitoria de performance (APM) e monitorização da infraestrutura, base de dados e análise da aplicação [34]. As principais funcionalidades são as seguintes [33]:

- Envio de alertas de uma falha critica baseando-se numa *baseline* definida.
- Resolução de problemas de performance analisando cada linha de código do software em monitoria.
- Usando um sistema de alertas e *feedback* por parte da equipa de manutenção do *software*, *Appdynamics* automaticamente descobre os padrões normais e de falha para cada métrica.
- Suporte a várias linguagens.

A implementação da aplicação não é complicada, fácil de configurar e acrescenta pouca latência na aplicação. Das características da aplicação podem-se destacar a sua compatibilidade com linguagens e soluções de desenvolvimento de software, bem como a sua facilidade de implementação.

3.5 Instana

Instana é uma outra aplicação de monitoria de performance, *Instana* monitoriza diferentes tipos de arquitetura incluindo sistema distribuídos em microsserviços. A monitoria de software distribuído é suportada por *tracing* nos serviços existentes, desta forma cada sistema é analisado independentemente da sua arquitetura e tipo de implementação, quer seja *bare metal*, virtualização ou em *containers* [33]. As principais funcionalidades são as seguintes [33]:

- Monitorização de arquiteturas distribuídas em *containers*.
- Elevada compatibilidade com linguagens de programação.
- Autodescobrimento dos serviços.
- Fácil configuração.

A implementação da aplicação *Instana* é realizada pela instalação de um agente, este descobre os serviços existentes de forma a iniciar a monitoria do sistema.

3.6 Comparação de ferramentas

Como forma de sintetizar a comparação de aplicações de monitorização de performance de sistemas de software, é apresentada na tabela 3.1. Na tabela é analisado um conjunto de parâmetros para cada aplicação de monitorização de performance analisadas anteriormente com a adição de uma aplicação extra. Os fatores analisados são os seguintes:

- Preço, o preço de subscrição mensal é um fator decisivo na escolha de uma aplicação de monitorização de performance, como consequência é analisado o preço de cada aplicação.
- Linguagens suportadas, análise à compatibilidade da solução.
- Suporte a arquiteturas distribuídas, nomeadamente a microsserviços.
- Tipo de *Deployment*, o tipo de implementação e a sua facilidade é fulcral para uma competitiva aplicação de monitoria de performance.
- Análise de problemas.

Ferramenta	Preço (1 mês)	Linguagens suportadas	microsserviços	Deployment	Análise de Problemas
New Relic	25-200 $ por servidor	Go, Java, .NET, Node.js, PHP, Python, Ruby.	Containers são analisados sob a forma de hosts, cada microsserviço é representado num mapa de serviços.	Deployment por instalação de agente.	análise de problemas baseada em exceções da execução do código apresentadas sob a forma de gráfico ao longo do tempo.
DynaTrace	Não indica preço.	Maioria das linguagens e tecnologias.	Microsserviços são automaticamente descobertos por um agente. Para a visualização a plataforma auxilia-se de uma topologia actualizada a tempo real.	Instalação de agente que descobre automaticamente todos os processos.	A causa e a evolução de um problema no sistema é apresentada numa árvore de dependências com suporte de uma análise da correlação de vários eventos.
Appdynamics	230 $ por servidor.	Maioria das linguagens e tecnologias.	Microsserviços representados sob a forma de grafo para um determinado período de tempo. É disponibilizada a opção de comparação do grafo com a *baseline* para análise das métricas de performance.	Deployment por agente.	Através de inteligência artificial é estabelecido um padrão para as métricas de performance. Estes padrões servem de comparação para a deteção de erros bem como a sua origem.
Instana	75 $	Maioria das linguagens e tecnologias.	Inclui grafo de microsserviços.	Análise dos processos por instalação de agente.	análise de problemas suportada por inteligência artificial.
Oracle Enterprise Manager Cloud Control 13c	*Pack* inicial gratuito embora contenha *packs* com preço fixo.			Deployment por agente.	Análise da raiz dos problemas através de análise de métricas e de *logs*.

Tabela 3.1: Comparação APMs

3.7 Ferramentas de visualização de grafos

A visualização correta de um grafo cumpre um importante papel na visualização de múltiplos tipo de dados, dados estes que são apresentados sobre a forma de grafo. Um grafo bem representado apresenta corretamente e de forma intuitiva os nós do mesmo bem como as suas ligações, para esta correta visualização, existem algoritmos que formam a estrutura do grafo de acordo com algumas métricas, no seguimento, vão ser abordados alguns destes algoritmos. As ferramentas de visualização de grafo de seguida analisada, fazem uso deste tipo de algoritmos como forma de melhorar a performance no tempo de processamento e para melhorar a estrutura do grafo.

A construção de grafo dever seguir um conjunto de requisitos como forma de facilitar uma correta leitura. Se seguida são abordados os 4 requisitos [35].

3.7.1 Legibilidade

A legibilidade garante que o grafo é fácil de ler, as principais características de um grafo que satisfazem o requisito de legibilidade são as seguintes[35]:

- Número de cruzamentos, o número de cruzamentos é o número de arestas sobrepostas no grafo, o número de arestas sobrepostas influência a intuição na perceção da informação presente no grafo. Um grafo sem arestas sobrepostas é mais intuito. No entanto, existem situações em que não é possível garantir o não cruzamento das arestas, pelo que é minimizado este número.

- Área do grafo, a área envolvente ao grafo influencia a perceção da informação do grafo, um grafo com uma área envolvente elevada diminui a intuitividade do grafo.

- Formato das arestas, o formato das arestas influencia a capacidade de um leitor de das seguir e de perceber a ligação entre os nós. Uma aresta com um formato simples e com um número de dobras e de mudanças de direção reduzido facilita a visualização do grafo.

- Tamanho das arestas, o tamanho das arestas num grafo tem a tendência a ser o mais reduzido possível, também a uniformidade do tamanho das arestas é importante, pelo que a estrutura dos nós no grafo deve uniformizar o comprimento das arestas.

3.7.2 Conformidade

O grafo deve conformar-se ao contexto do grafo, isto é, podem ser impostas alguma características ao grafo fruto do contexto do mesmo. As características podem ser as seguintes [35]:

- Ordem horizontal dos nós, por exemplo, o nó A tem de aparecer à esquerda do nó B.

- Ordem vertical dos nós.

- Distância entre nós, por exemplo, o nó A tem de aparecer perto do nó B.

3.7.3 Controlabilidade

O requisito de controlabilidade é semelhante ao requisito de conformidade, neste caso, as limitações do grafo são impostas pelo utilizador e não podem ser implementadas no código, de igual forma ao requisito anterior, as mesmas limitações se aplicam a este requisito [35].

3.7.4 Eficiência

A eficiência está relacionada com o tempo de processamento do grafo. O processamento de um número elevado de nós e de arestas pode demorar minutos, por esta razão, o requisito de eficiência é fundamental neste tipo de algoritmos.

3.7.5 Técnicas de visualização de grafos

Os algoritmos de visualização de grafos têm como objetivo a visualização, mas cumprindo os requisitos anteriormente descritos. De entre as várias técnicas implementadas pelos algoritmos, são destacadas as 3 seguintes [35]:

- *The Sugiyama algorithm*, os nós do grafo são dispostos em camadas, o *flow* do grafo é representado do topo para as camadas inferiores. Os objetivos do algoritmo são reduzir o número de cruzamentos de arestas, manter as arestas o mais direitas possível e a distribuição uniforme dos nós pelo espaço disponível.
- *TRIP*, a técnica *TRIP* foi inventada por kamada em 1989 e consiste num algoritmo de 3 fases. A primeira consiste na análise dos nós e no estabelecimento das relações lógicas entre eles. A segunda consiste na transformação dos objetos lógicos para geométricos e das transformações lógicas para geométricas. A terceira e última fase consiste na transformação dos objetos e relações para uma figura/imagem.
- Técnicas de mola, a construção do grafo em forma de figura tem por base o estabelecimento de relações de força entre os nós, forças de atracão entre nós com arestas e força de afastamento para nós sem nenhuma relação. O objetivo final é a construção de uma configuração com as forças entre os nós anuladas.

Na tabela 3.2 é apresentada a avaliação das diferentes técnicas no cumprimento dos requisitos de construção visual de um grafo anteriormente descritos [35].

Técnica	**Legibilidade**	**Conformidade**	**Controlabilidade**	**Eficiência**
The Sugiyama algorithm	Excelente	Pobre	Pobre	Bom
TRIP	Pobre	Excelente	Excelente	Pobre
Técnicas de mola	Médio	Médio	Médio	Médio

Tabela 3.2: Cumprimento dos requisitos de visualização de grafos

Analisando as técnicas de visualização de grafos, pode-se definir uma preferência na escolha da ferramenta a utilizar para a construção automática de um grafo. A partir da tabela 3.2, pode-se identificar a técnica de mola como a mais equilibrada em relação as características analisadas na tabela. Nas secções seguintes são apresentadas algumas das ferramentas de construção analisadas, na sua analise as técnicas de construção do grafo têm um importante interesse.

3.7.6 Ferramentas *Standalone* de construção de grafos

As ferramentas existentes para a visualização de grafos apresentam características diferentes entre cada uma. De entre as principais características e que mais influenciam a sua escolha é o tipo de aplicação. Existem dois tipos de aplicações, um primeiro tipo que são aplicações *standalone*, e um segundo que funcionam como *frameworks* ou *addons* a um proje to.

Tom Sawyer Perspectives

Tom Sawyer Perspectives é uma ferramenta de visualização de dados, ajuda na análise de grandes quantidades de dados. Para alem de permitir a visualização de dados,

nomeadamente de grafos, inclui também ferramentas de consulta de base de dados, construção de modelos de processo de negócio (BPM) e a construção de diagramas MBSE, *model-based systems engineering*.

Esta ferramenta suporta dois módulos de funcionamento, o *designer* e o *previwer*. O modulo *designer* permite ao utilizador definir esquemas de dados, fontes de informação de dados, regras e pesquisas. O modulo *previwer* é usado para ver a informação adicionada visualmente. A informação a visualizar é carregada utilizando um integrador utilizando *queries*, para além diste tipo de integração, também pode ser integrada informação utilizando *REST*, *JSON*, *XML*, texto, *SQL*, excel e *RDF*.

Gephi

Gelphi é uma plataforma de visualização de grafos grátis para usar, apenas é cobrado um valor para ter acesso a repositórios privados. Uma das suas principais características é a possibilidade de geração de de grafos com 50000 nós. *Gelphi* utiliza algoritmos resultantes das técnicas anteriormente descritas, nomeadamente algoritmos baseados em técnicas de mola. A ferramenta *Gelphi* tem suporte para grafos dinâmicos em hierárquicos e permite também ao utilizador mudar o aspecto do grafo alterando certas definições [36].

Após o utilizador fazer a implementação do grafo, é lhe possibilitada a consulta de certas métricas, como a centralidade da proximidade dos nós, diâmetro do grafo, coeficiente de agrupamento a detecção de caminhos mais curtos entre grafos.

3.7.7 Framework de construção de grafos

As ferramentas de seguidas analisadas de seguida são *frameworks* adaptáveis a um projeto do utilizador, sendo compatíveis com *javascrip* ou outro tipo de linguagem de programação. De seguida algumas deste tipo de ferramentas são analisadas.

Vis.js

Vis.js é uma livraria de visualização de grafo compatível com *javascript* que permite a construção de grafos com uma grande quantidade de informação. Esta ferramenta para além de permitir a gestão e visualização de grafos também opera sobre outro tipo de dados, como *timelines*, *datasets*, *networks* e gráficos a 2 e 3 dimensões.

Para cada tipo de formato de dados, a plataforma oferece vários tipos de customização, como [37]:

- Estilo e etiqueta de cada nó.
- Tipo de arestas
- Tipo de disposição dos nós, por exemplo uma disposição hierárquica.
- Adição de eventos relativos a clicks.

Um exemplo de um grafo produzido pela ferramenta *vis.js* é apresentado na figura 3.2.

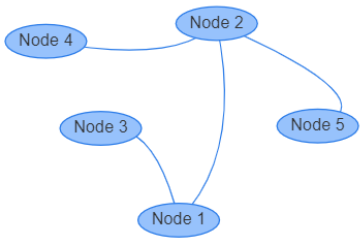

Figura 3.2: Grafo produzido pela ferramenta a *Vis.js*

Graphviz

A ferramenta/*framework* graphviz é *opensource* permite a visualização de informação relacionada com grafos ou com algum tipo de rede. A ferramenta faz uso de uma sintaxe de preenchimento da informação das relações dos nós e constrói o relativo grafo e cria o resultado em diversos formatos [38], entre os quais: imagens, *SVG* para páginas *web*, *Postscript* para incluir em *pdfs* e também existe a possibilidade e resultado ser apresentado num *browser* de grafos.

A ferramenta possibilita a customização de cores, fontes de letra, *layout* dos nós ou estilo das arestas. Para além destas características, a ferramenta oferece vários tipos de formatos de grafos para o utilizador escolher de acordo com o contexto da sua informação, nomeadamente[39]:

- Formato "dot", produz uma hierarquia de grafos direcionados, este algoritmo procura manter as arestas na mesma direção e evitar o cruzamento das mesmas.
- Formato "neato", produz um grafo utilizando o algoritmo Kamada-Kawai, isto é, através de forçar de repulsa e de aproximação utilizadas em técnicas de mola, a função de energia do sistema é minimizada.
- Formato "fdp", é semelhante ao formato "neato" mas em vez de reduzir a função de energia, trabalha diretamente com as forças.

3.7.8 D3.js

A ferramenta *D3.js* é desenvolvida em *javascript* e permite a criação dinâmica de componentes visuais para navegadores web. De forma a ser possível criar de modo facilitado as componentes por parte do utilizador, a ferramenta faz uso de gráficos vetoriais escaláveis (*SVG*), funcionalidades de *HTML5* e também de *CSS*.

A criação baseada na ferramenta *d3js* de um grafo, faz uso de técnicas de mola. O uso das técnicas de mola permite a customização da forma de construção do grafo, isto é, a adição de força em certos sentidos ou a sua remoção permite a construção de grafos com formas variadas.

A ferramenta *D3.js* [40] permite a construção de uma série de componentes visuais, tais como grafos, gráficos de barras, calendários, diagramas de cordas entre outros, através de [41] é possível identificar todos as componentes visuais que a ferramenta permite criar.

No contexto do currente estágio, é de interesse analisar a forma como esta ferramenta cria um determinado grafo. A figura 3.4 representa um grafo criado com recurso à ferramenta, este exemplo surge na documentação da ferramenta [41]

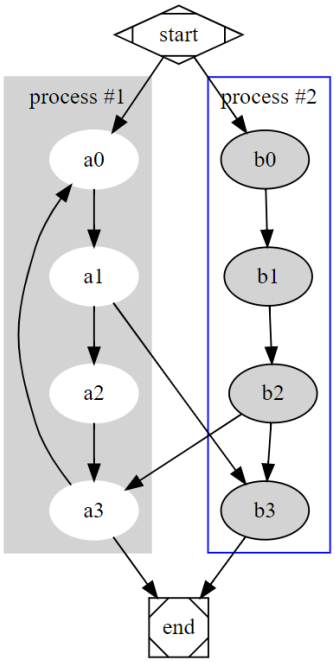

Figura 3.3: Exemplo de grafo produzido por *Graphviz* em formato "dot"

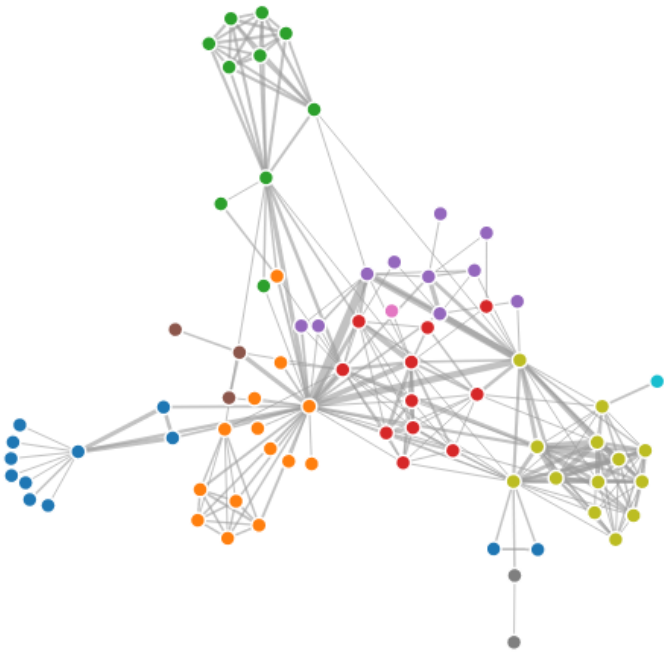

Figura 3.4: Exemplo de grafo produzido pela ferramenta *d3.js*

Para da informação já apresentada sobre a ferramenta, é importante referir a especificação de entrada e saída de dados para a construção de uma qualquer componente visual. Após algum estudo sobre a mesma, foi determinado que as componentes visuais são representadas sobre a forma de um *SVG* interligado a uma componete *DOM* de *html5*, este facto é de elevada importância para a escolha de uma solução a adotar devido à possibilidade de manipulação da informação e das componetes visuais após a construção de um grafo.

3.7.9 Conclusões

Existem dois tipos de ferramentas que criam ou ajudam um utilizador a criar as componentes visuais na forma de um grafo e relativas a um conjunto de informação do utilizador. As ferramentas *standalone*, apesar da sua complexidade, não se adequa ao trabalho prático realizado, por outro lado, as *frameworks* ou livrarias que suportam uma linguagem de programação surgem como uma melhor solução a adotar.

Após o teste da maior parte das ferramentas analisadas nas secções anteriores, pode-se concluir que a solução que melhor se adapta ao trabalho prático é a ferramenta *d3.js*. Os fatores que motivaram esta solução é o formato de dados criados, facilidade de utilização,documentação existente e também o tipo de metodologia utilizada, neste caso, uso de técnicas de mola.

Capítulo 4

Solução

A solução caracterizada pelos requisitos e arquitetura de seguida tem como objetivo principal a construção de componentes visuais que facilitam a percepção das dependências dos serviços do sistema a analisar.

A necessidade do desenvolvimento da ferramenta de visualização documentada neste relatório foi identificada durante o semestre anterior ao início deste presente estágio, à data de Fevereiro de 2019, pelo estudante de doutoramento André Bento na sua dissertação de mestrado. Parte do trabalho prático do colega consistiu na recolha de informação em *traces* de um sistema real, assim, foi identifica a lacuna no mercado de uma ferramenta que permite uma visualização visual da arquitetura do sistema relativo aos *traces* recolhidos.

Desta forma, a solução construída durante os dois semestres da dissertação tenta combater a lacuna de mercado anteriormente encontrada. Nas próximas secções são apresentados os requisitos e arquitetura da plataforma desenvolvida.

4.1 Requisitos Funcionais

A definição dos requisitos funcionais é essencial na fase inicial de qualquer projeto. A especificação dos requisitos auxilia todos os envolvidos durante as fases seguintes da implementação. Nesta secção são apresentados os requisitos funcionais associados a um cenário do sistema, cada cenário pode incluir um ou mais requisitos funcionais.
Um cenário consiste num conjunto de ações do utilizador que desencadeia uma resposta visível por parte do sistema. Para cada cenário são definidos os seguintes componentes: nome do cenário, desencadeamento, pré-condições, caminho, resposta e condições excepcionais.

4.1.1 Escolha do intervalo de amostragem dos dados

Requisitos Funcionais:

- **REQF-1:** O utilizador deve poder escolher um intervalo de tempo para a amostragem dos dados (Alta prioridade).

Cenário	Escolha do intervalo de amostragem dos dados
Desencadeamento	O utilizador seleciona um intervalo de tempo na barra cronológica.
Pré-Condições	Em todas as páginas existe uma barra cronológica que possibilita escolher o intervalo de tempo para a amostragem dos dados. O utilizador seleciona uma página de apresentação de informação.
Caminho	1. O utilizador seleciona uma das seguintes *tabs*: grafo de dependências, *workflows*, *workflows* de erros ou diferença do grafo de dependências. 2. O utilizador seleciona na barra cronológica o intervalo de tempo pretendido. 3. O sistema apresenta a informação relativa ao intervalo de tempo selecionado.
Resposta	É apresentada a informação correspondente ao intervalo de tempo selecionado
Condições excepcionais	Não existem dados correspondentes ao intervalo de tempo selecionado. O utilizador pode abandonar a operação a qualquer momento

Tabela 4.1: Escolha do intervalo de amostragem dos dados

4.1.2 Realçar *workflows* de erros

Cenário	Realçar *workflows* de erro
Desencadeamento	O utilizador seleciona a funcionalidade de apresentar *workflows de erros na barra de navegação*
Pré-Condições	O utilizador seleciona um intervalo de tempo para amostragem dos dados
Caminho	1. O utilizador seleciona a funcionalidade de apresentar *workflows de erros na barra de navegação* 2. O utilizador seleciona um intervalo de tempo para amostragem dos dados
Resposta	O sistema apresenta no grafo de dependências os *workflows* com algum tipo de erro, realçados no grafo.
Condições excepcionais	Não existem *workflow* com erros para o intervalo de tempo selecionado O utilizador abandona a operação a qualquer momento.

Tabela 4.2: Realçar *workflows* de erros

Requisitos Funcionais:

- **REQF-2:** O sistema deve apresentar os *workflows* onde exista algum erro. (Média prioridade).

4.1.3 Identificar instâncias de um serviço

Requisitos Funcionais:

- **REQF-3:** O sistema deve apresentar as instâncias correspondentes ao serviço escolhido (Alta prioridade).

4.1.4 Identificar dependências das instâncias de um serviço

Requisitos Funcionais:

Cenário	Identificar instâncias de um serviço
Desencadeamento	O utilizador seleciona um serviço no grafo de dependências
Pré-Condições	O utilizador acede à página do grafo de dependências.
Caminho	1. O utilizador acede à página do grafo de dependências. 2. O utilizador escolhe o intervalo de tempo para a amostragem dos dados. 3. O utilizador seleciona um serviço.
Resposta	O sistema apresenta para o serviço selecionado as suas instâncias sob a forma de árvore.
Condições excepcionais	O utilizador abandona a operação a qualquer momento.

Tabela 4.3: Identificar instâncias de um serviço

Cenário	Identificar dependências das instâncias de um serviço
Desencadeamento	O utilizador seleciona um serviço no grafo de dependências
Pré-Condições	O utilizador seleciona um intervalo de tempo de amostragem de dados.
Caminho	1. O utilizador acede à página do grafo de dependências. 2. O utilizador escolhe o intervalo de tempo para a amostragem dos dados. 3. O utilizador seleciona um serviço.
Resposta	O sistema apresenta as dependências entre cada instância e os serviços restantes.
Condições excepcionais	O utilizador abandona a operação a qualquer momento.

Tabela 4.4: Identificar dependências das instâncias de um serviço

- **REQF-4:** Caso o utilizador assim escolha, o sistema deve apresentar as ligações dependências entre as instâncias de um serviço e os restantes serviços (Média prioridade).

4.1.5 Proporção de pedidos recebidos para cada instância do serviço

Cenário	Proporção de pedidos recebidos para cada instância do serviço
Desencadeamento	O utilizador seleciona um serviço no grafo de dependências
Pré-Condições	O utilizador seleciona um intervalo de tempo de amostragem de dados.
Caminho	1. O utilizador acede à página do grafo de dependências. 2. O utilizador escolhe o intervalo de tempo para a amostragem dos dados. 3. O utilizador seleciona um serviço. 4. O utilizador escolhe uma instância
Resposta	O sistema apresenta a proporção de pedidos tratados para a instância selecionada, em relação ao total de pedidos tratados pelo serviço.
Condições excepcionais	O utilizador abandona a operação a qualquer momento.

Tabela 4.5: Proporção de pedidos recebidos para cada instância do serviço

Requisitos Funcionais:

- **REQF-5:** O sistema deve apresentar para cada instância a proporção de pedidos recebidos em relação ao número total de pedidos recebidos pelo serviço (Baixa Prioridade).

4.1.6 Tipos de código *HTTP* para cada serviço e instância

Cenário	Tipos de código *HTTP* para cada serviço e instância
Desencadeamento	O utilizador seleciona um serviço no grafo de dependências
Pré-Condições	O utilizador seleciona um intervalo de tempo de amostragem de dados.
Caminho	1. O utilizador acede à página do grafo de dependências. 2. O utilizador escolhe o intervalo de tempo para a amostragem dos dados. 3. O utilizador seleciona um serviço ou instância.
Resposta	O sistema apresenta para cada serviço ou instância os tipos de códigos *HTTP* e o número de ocorrências para cada.
Condições excepcionais	O utilizador abandona a operação a qualquer momento.

Tabela 4.6: Tipos de código *HTTP* para cada serviço e instância

Requisitos Funcionais:

- **REQF-6:** O sistema deve apresentar para cada instância e serviço o número dos diferentes códigos *HTTP* existentes e o número de pedidos recebidos e enviados (Baixa Prioridade).

4.1.7 Comparação de grafos de dependências

Cenário	Comparação de grafos de dependências
Desencadeamento	O utilizador seleciona a opção de comparação de grafos de dependência na barra de navegação.
Pré-Condições	O utilizador seleciona dois intervalos de tempo para a comparação de grafos.
Caminho	1. O utilizador acede à página de comparação de grafo de dependências. 2. O utilizador escolhe os intervalos de tempos para a comparação dos grafos.
Resposta	O sistema apresenta um grafo que representa a diferença de pedidos e serviços entre os dois grafos. A diferença na arquitetura de serviços dos dois grafos é apresentada pela coloração dos serviços que foram eliminados ou acrescentados. As arestas dos grafos representam a diferença entre o número de pedidos.
Condições excepcionais	O utilizador seleciona intervalos de tempo Incompatíveis. O utilizador abandona a operação a qualquer momento.

Tabela 4.7: Comparação de grafos de dependências

Requisitos Funcionais:

- **REQF-7:** O utilizador pode escolher comparar o grafo de dependências para dois intervalos de tempo escolhidos (Alta prioridade).

- **REQ-8:** O sistema apresenta a diferença entre grafos de dependências para dois intervalos de tempo escolhidos através de um novo grafo (Alta prioridade).

- **REQ-9:** O sistema apresenta, no grafo de comparação, a diferença entre o número de pedidos para todas a ligações entre os serviços (Alta prioridade).

Cenário	Número *workflows*
Desencadeamento	O utilizador seleciona a opção de *workflows* na barra de navegação
Pré-Condições	O utilizador seleciona o intervalo de tempo para a amostragem dos dados.
Caminho	1. O utilizador acede à página de *workflows* 2. O utilizador escolhe o intervalo de tempo para a amostragem dos *workflows* 3. O utilizador escolhe o tipo de *workflows* a apresentar.
Resposta	O sistema apresenta os diferentes *workflows* diferentes para o intervalo de tempo escolhido, os serviços são identificados de acordo com o tipo de *workflow* escolhido, *HTTP*, nome do serviço e *End-point*.
Condições excepcionais	O utilizador abandona a operação a qualquer momento.

Tabela 4.8: Comparação de grafos de dependências

4.1.8 Número de *workflows*

Requisitos Funcionais:

- **REQF-10:** O sistema deve apresentar para um intervalo de tempo os diferentes tipos de workflows sobre a forma de árvore (Média prioridade).

- **REQF-11:** O sistema deve apresentar para um intervalo de tempo *workflows* do tipo: *HTTP*, service name e End-point (Média prioridade).

4.1.9 Detalhes de um *workflow*

Cenário	Detalhes de um *workflow*
Desencadeamento	O utilizador seleciona um *workflow* na página de amostragem dos diferentes *workflows*
Pré-Condições	O utilizador seleciona o intervalo de tempo para a amostragem dos dados.
Caminho	1. O utilizador acede à página de *workflows* 2. O utilizador escolhe o intervalo de tempo para a amostragem dos *workflows* 3. O utilizador escolhe o tipo de *workflows* a apresentar. 4. O utilizador seleciona um *workflow*.
Resposta	O sistema apresenta os detalhes do *workflow* escolhido e dá acesso aos traces correspondentes ao respectivo workflow..
Condições excepcionais	O utilizador abandona a operação a qualquer momento.

Tabela 4.9: Detalhes de um *workflow*

Requisitos Funcionais:

- **REQF-12:** O sistema deve dar aceso ao utilizador e para cada *workflow* os traces respetivos.

4.1.10 Pesquisa de serviço ou instância

Requisitos Funcionais:

Cenário	Pesquisa de serviço ou instância
Desencadeamento	Utilizador faz uso da caixa de pesquisa
Pré-Condições	
Caminho	1. O utilizador acede à página do grafo de dependências. 2. O utilizador escolhe o intervalo de tempo para a amostragem dos dados. 3. Preenche o campo de pesquisa e o tipo de pesquisa. 4. O utilizador confirma a ação.
Resposta	O sistema apresenta uma lista com os serviços ou instâncias de acordo com a pesquisa do utilizador.
Condições excepcionais	O utilizador abandona a operação a qualquer momento.

Tabela 4.10: Pesquisa de serviço ou instância

- **REQF-13:** O sistema deve permitir a pesquisa de serviços ou instâncias de serviços e ordenados por: número de pedidos recebidos, número de vizinhos no grafo, número de diferentes tipos de pedidos (Prioridade baixa).

4.2 Priorização dos requisitos funcionais

A cada requisito funcional foi atribuída uma das seguintes prioridades: Alta, média e baixa prioridade. A classificação da prioridade de cada requisito funcional foi escolhida de acordo com a sua importância no contexto do problema, sendo assim, um requisito de alta prioridade apresenta informação bastante pertinente para o utilizador.

Prioridade	**Requisito**
Alta prioridade	REQF-1 REQF-3 REQF-7 REQ-8 REQ-9
Média prioridade	REQF-2 REQF-4 REQF-10 REQF-11 REQF-12 REQF-13
Baixa prioridade	REQF-5 REQF-6

Tabela 4.11: Priorização dos requisitos funcionais

4.3 Requisitos de qualidade

Atributos de qualidade são uma parte constituinte dos *drivers* arquiteturais de um sistema. Os drivers arquiteturais de um sistema levantam as perguntas "Como" e "Porquê" que influenciam as escolhas na arquitetura do mesmo, os *drives* arquiteturais são constituídos pelos requisitos funcionais, requisitos de qualidade e restrições técnicas ou de negócio.

Para além das funcionalidades de um sistema, os atributos de qualidade são características extra e são dos *drivers* arquiteturais mais difíceis de identificar e definir.

Os requisitos de qualidade são identificados na tabela 4.12 que representa a árvore de utilidade, cada atributo de qualidade pode corresponder a mais do que um driver arquitetural (ASR). Cada ASR é qualificado com duas letras e de acordo com a sua importância para a definição da arquitetura e com a sua importância para o negócio, respectivamente. A notação utilizada é a seguinte: alta (H), média (M), baixa (L).

Atributo de qualidade	**Refinamento do atributo**	**ASR**
Usabilidade	Apresentação de serviços, QA1	O utilizador acede a uma das páginas de apresentação da estrutura dos serviços, num ambiente de normal execução, o sistema deve apresentar numa única vista do ecrã 30 serviços.
	Agrupamento de serviços, QA2	O utilizador acede a uma das páginas de apresentação dos serviços, numa situação de apresentação de um número superior a 30 serviços, o sistema deve agrupar os serviços de modo a facilitar a visualização do grafo.
	Intuição da realização de tarefas, QA3	Um utilizador pretende realizar facilmente todas as tarefas. O sistema permite ao utilizador realizar as tarefas mais frequentes em menos de 3 *clicks* a partir de qualquer página da plataforma.
	Memorização da interface QA4	Um utilizador pretende realizar qualquer tarefa sem qualquer assistência. O utilizador, após uma utilização de 10 minutos da plataforma, consegue navegar e realizar todas as tarefas disponíveis sem dificuldade.
Compatibilidade	Documentação sobre a interface de partilha de informação, QA5	O utilizador deseja utilizar a plataforma para visualizar a sua arquitetura, o sistema fornece documentação sobre a interface necessária para que o sistema tenha acesso aos dados do utilizador.
Performance	Rapidez no carregamento dos grafos, QA6	O utilizador acede a uma das páginas de apresentação dos serviços e escolhe um intervalo de tempo para a amostragem dos dados, numa situação de apresentação de 30 serviços, o sistema apresenta a disposição dos serviços no grafo em menos de 3 segundos.

Tabela 4.12: Árvore de utilidade dos requisitos funcionais

4.3.1 QA1

Apresentação de serviços, de forma a permitir a visualização de 30 serviços é necessário recorrer a algoritmos de disposição de nós em grafos, como por exemplo o *force-directed graph drawing*, *spectral layout* ou *layered graph drawing*. Os algoritmos anteriormente enumerados asseguram uma correta disposição dos nós do grafo, de modo a rentabilizar o espaço de ecrã disponível e facilitar a visualização do mesmo. Para uma fácil e intuitiva percepção da arquitetura de microsserviços representada no respectivo grafo, as ferramentas de construção de grafos fazem uso de algoritmos de disposição dos nós de um grafo pelo espaço disponível.

4.3.2 QA2

Agrupamento de serviços, foi definido que um grafo com um número de nós superior a 30, necessita de um agrupamento de serviços de forma a limitar a quantidade de informação apresentada ao utilizador numa única vista. Os serviços são agrupados pelo número de ligações entre os vizinhos, isto é, os serviços de um conjunto apresentam um grande número de interligações. O algoritmo *Tarjan* [42] deteta componentes fortemente ligadas, o que possibilita o agrupamento de serviços para o contexto do atributo de qualidade 2.

4.3.3 QA3

Intuição da realização de tarefas, O sistema fornece aos utilizadores a possibilidade da realização de qualquer tarefa principal em menos de 3 *clicks* a partir da qualquer página da plataforma. A presença de uma barra de navegação na lateral, em conjunto a janela de pesquisa acessível do lado direito de todas das vistas na plataforma facilitam navegação por parte dos utilizadores.

4.3.4 QA4

Memorização da interface, uma interface de utilizador intuitiva diminui a carga de informação que é necessário memorizar pelo utilizador para que este navegue rapidamente pela plataforma.

4.3.5 QA5

Documentação sobre a interface de partilha de informação, a plataforma consome um serviço que expõe uma interface *REST* para a partilha dos dados sobre os *traces* a serem analisados. Apesar de o sistema ser desenvolvido com base na informação recolhida no trabalho realizado durante a dissertação do colega André Bento, o objetivo final é a possibilidade de qualquer utilizador visualizar os serviços da sua plataforma. A documentação sobre o esquema dos dados para a partilha de informação é essencial para a compatibilidade da aplicação com sistemas personalizados do utilizador.

4.3.6 QA6

Rapidez no carregamento dos grafos, um dos fatores mais limitativos da plataforma é o processamento dos nós do grafo para a sua posterior visualização. Devido à existência de uma barra cronológica que permite a escolha do intervalo de tempo, é necessário garantir a rapidez no processamento do grafo para manter a suavidade na apresentação dos dados ao utilizador. Após vários testes realizados sobre a ferramenta *graphviz* concluiu-se que para um número de nós inferior a 30 e a média de arestas de cada nó 15, está garantido um tempo de processamento inferior a 3 segundos.

4.4 Arquitetura

A solução apresentada nesta secção é uma plataforma web que permite a visualização da informação resultante de *traces* sob a forma vários grafos.

O *input* da aplicação é a informação relativa aos grafos dos *traces* do utilizador. Esta informação é recebida através da utilização de uma interface *REST* desenvolvida pelo utilizador, o objetivo da dissertação é desenvolver uma plataforma que consuma uma interface externa, para isso, é fornecida documentação sobre as especificações da mesma e o formato dos dados a enviar, em *JSON*. O processamento relativo à transformação dos dados para o formato especificado na arquitetura é responsabilidade do utilizador.

Após a aplicação receber os dados em JSON, estes são guardados numa base de dados *mysql* como forma de facilitar o seu futuro acesso, ou seja, a base de dados surge como uma cache para os grafos do utilizador. Como a informação é recebida num formato *JSON*, esta pode ser guardada numa base de dados regular, consequentemente, não é necessário utilizar uma base de dados de grafos. O utilizador, através da aplicação *Web* que por sua vez encaminha os pedidos ao modulo de gestão dos mesmos. Caso a informação relativa ao grafo desejado se encontre em *cahche*, este é devolvido e com auxílio de uma ferramenta de construção de grafos, é construído o grafo em modo visual. Caso não exista a informação do pedido, é realizada uma chamada à interface da ferramenta externa e construído de igual forma o grafo com essa informação.

Os atributos definidos anteriormente, requisitos funcionais e de qualidade, definem a arquitetura da plataforma. Na representação da arquitetura é usado o modelo C4 proposto por Simon Brown [43]. Segundo o *C4 Model* são definidos 4 diferentes diagramas que representam diferentes vistas da arquitetura, os modelos definidos são os seguintes: diagrama de contexto, diagrama de contentores, diagrama de componentes e diagrama de diagrama de código. As secções seguintes apresentam os 3 primeiros diagramas bem como a explicação de algumas decisões pertinentes.

4.4.1 Diagrama de Contexto

O diagrama de contexto é essencial para entender o contexto geral da plataforma e a interação entre os módulos que compõem o sistema. A figura 4.2 apresenta ao diagrama de contexto da plataforma de visualização.

A plataforma de visualização recebe pedidos de um utilizador, esta por sua vez obtém dados da ferramenta de recolha de informação sobre traces desenvolvida pelo colega André Bento.

4.4.2 Diagrama de Contentores

O diagrama de contentores apresenta uma visão mais concreta da plataforma. Cada módulo do diagrama representa uma unidade capaz de ser executada ou de guardar dados, como nas base de dados [43]. A partir do diagrama de contexto, o modulo da ferramenta de visualização é divido em dois, o *Back-End* e a *Web-app*. A *Web-app* apresenta uma interface ao utilizador e este pode fazer pedidos na mesma. Cada pedido é redirecionado para o *Back-end* , este processa-o e se necessário é realizado um pedido de informação à ferramenta de recolha de dados de traces, através de um pedido *HTTP* à *REST API*. Também é lida e escrita informação nas duas base de dados, uma para guardar estatísticas de utilização e

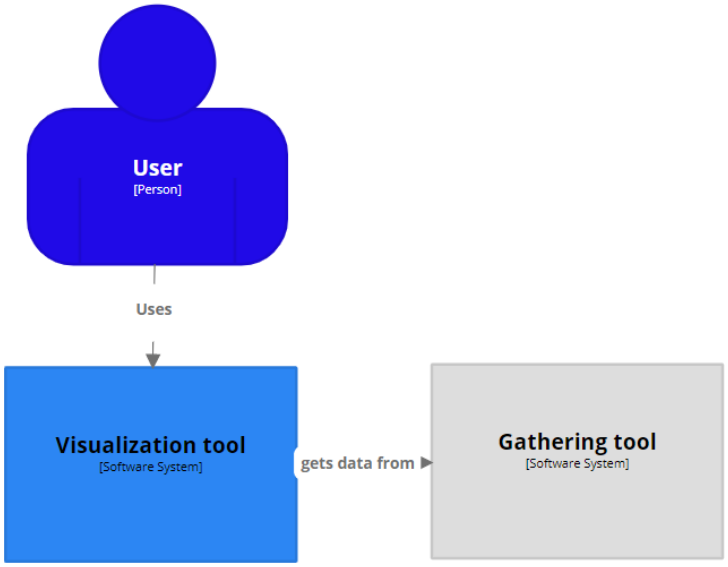

Figura 4.1: Diagrama de contexto

uma outra que serve como cache de informação sobre os grafos. A base de dados de cache de informação de grafos é importante para reduzir o tempo de processamento dos grafos, com o objetivo de cumprir o requisito de qualidade número 6.

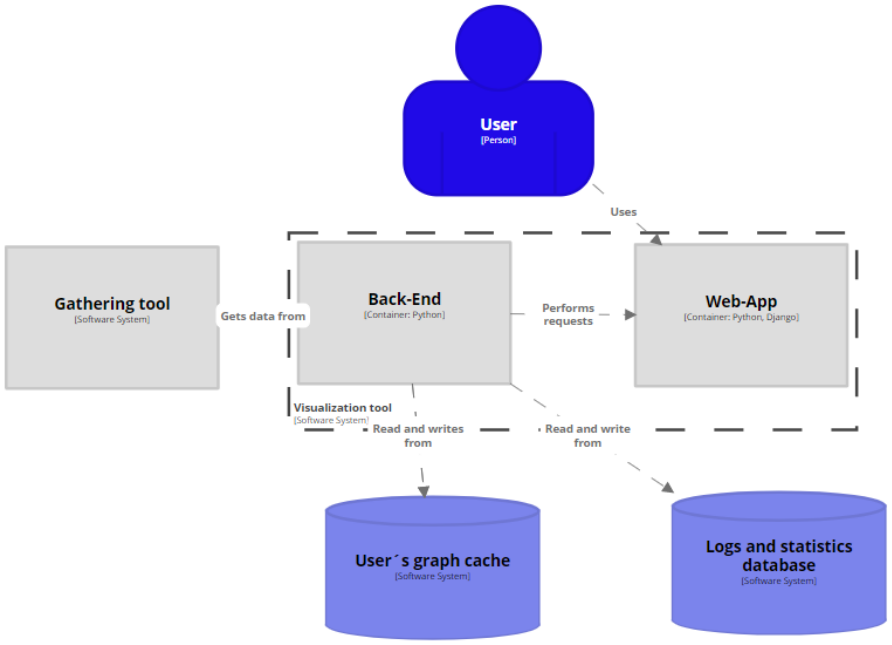

Figura 4.2: Diagrama de contentores

4.4.3 Diagrama de Componentes

O diagrama que mais especifica a arquitetura, dos diagramas propostos por Simon Brown [43], é o diagrama de componentes. Seguindo a lógica dos diagramas anteriores, este diagrama faz um *zoom* no diagrama de contentores de forma a decompor cada modulo

em blocos estruturalmente independentes. No diagrama é também possível identificar as relações de independência entre cada módulo e são de igual forma especificadas as tecnologias de cada bloco e detalhes de implementação.

A diferença entre o diagrama de contentores e o diagrama de componentes apresentado na figura 4.3 é o maior detalhe do módulo de *Back-end*, o resto dos módulos não sofrem alterações nos dois diagramas. A linguagem de programação escolhida para desenvolver a plataforma é *Python*, para além disso são usadas *frameworks* que facilitam o desenvolvimento, como no caso de *Django*. As razões da escolha de *Python* como linguagem de programação são a sua facilidade de utilização e a sua grande popularidade.

As constituintes do diagrama de componentes são as seguintes:

- User
- Web-App, a aplicação *web* expõe as funcionalidades ao utilizador, segundo o pedido do utilizador, a aplicação faz pedidos ao modulo *Process requests from the Web-App*, que por sua vez reponde com a informação desejada em JSON. No caso de um pedido de um qualquer tipo de grafo, a *Web-App* recebe a informação sobre a estrutura do grafo, isto é, sobre o posicionamento de cada nó no ecrã, também através de uma resposta *HTTP* e com esquema de dados em *JSON*.

  A aplicação *web* é desenvolvida em *python* e em topo da *framework Django*.
- Gathering Tools
- *Process requests from the Web-App*, este modulo recebe pedidos *HTTP* da *Web-App*. Dependendo do pedido, outras ações são desencadeadas por parte do módulo. No caso do recebimento de um pedido para a amostragem de um grafo, é verificada a existência do mesmo na base de dados de cache, numa não existência do mesmo na cache, é realizado um pedido ao módulo que se conecta com a ferramenta de recolha de grafos a partir de *traces*.
- *Acess data from gathering tool*, numa situação de necessidade de adicionar informação formatada em forma de grafo à plataforma, este módulo faz pedidos à ferramenta de recolha de informação sobre *traces*. Os pedidos à ferramenta externa são do tipo *HTTP* com um esquema de dados em JSON.
- *Statistics and logs*, o modulo é acedido pelo modulo principal e orquestrador, o *Process requests from the Web-App*. Este modulo é responsável por operações de leitura e de escrita na base de dados de estatísticas e *logs*, os dados são acedidos através de operações de persistência dos mesmos.
- *Manage cache*, de igual forma ao modulo de acesso à base de dados de estatísticas e *logs*, este modulo realiza operações de persistência de dados sobre a base de dados de cache. Este módulo é acedido pelo gestor principal de pedidos.
- *Generate graph structure*, com a utilização da *framework Graphviz*, a estrutura do grafo é calculada, para além desta função, este modulo é responsável por agrupar previamente os nós se necessário e utilizando algoritmos como o *tarjan*.
- *Logs and statistics database*, o modulo *statistics and logs* recebe dados estatísticos e *logs* da utilização da plataforma, este escreve e lê este tipo de informação nesta base de dados *mysql*.

- *User's Graph cache*, a base de dados de grafos em *cache* é uma base de dados *mysql*, esta por sua vez guarda a informação sobre grafos anteriormente requisitados como forma de melhorar o desempenho da plataforma, os grafos são guardados na base de dados com as coordenadas de cada nó para evitar repetir o processamento do grafo. A base de dados é acedida pelo modulo de gestão da mesma.

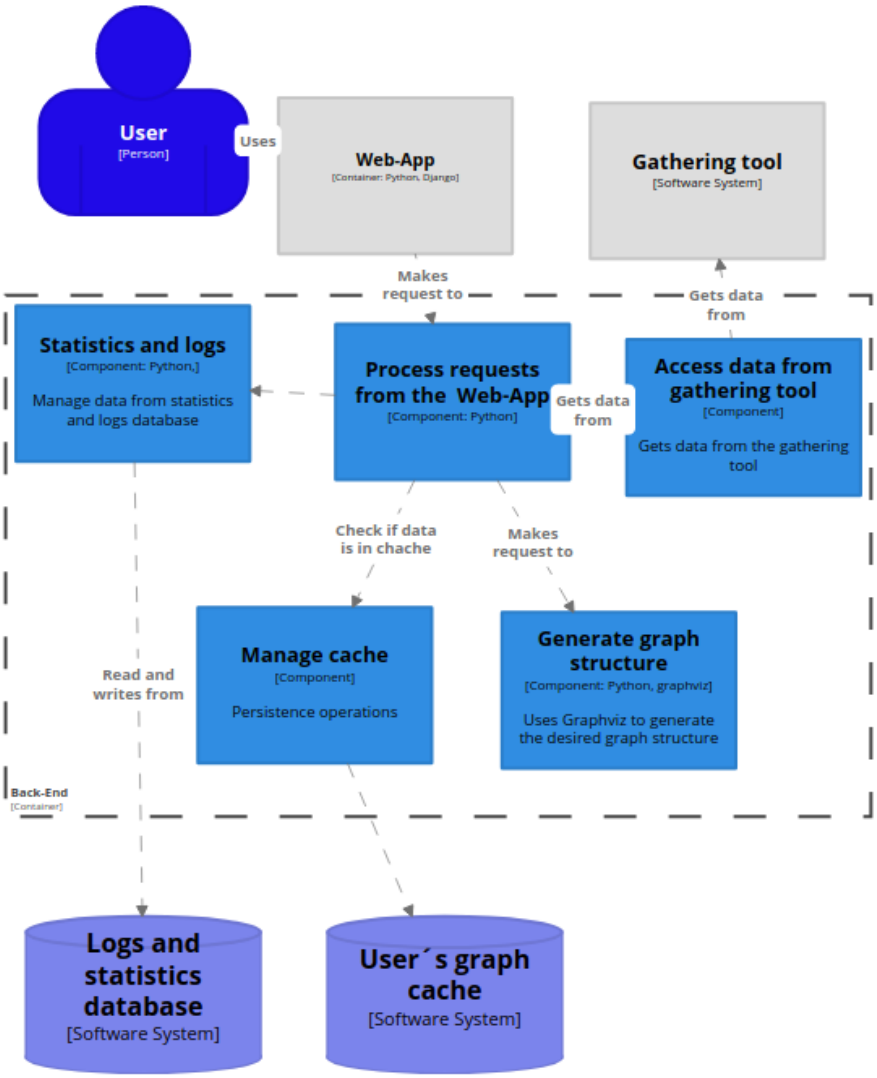

Figura 4.3: Diagrama de componentes

4.5 Protótipos de baixa fidelidade

O objetivo principal da tese é desenvolver e documentar a plataforma de visualização de arquiteturas em microsserviços, numa fase antecessora à implementação é essencial a criação de algum tipo protótipo, de forma a testar o design da plataforma. Existem dois tipos distintos de protótipos que podem ser classificados pela sua fidelidade, protótipos de alta e baixa fidelidade. Um protótipo de baixa fidelidade não envolve o desenvolvimento de um design próximo do final, mas é um conjunto do conceitos e ideias expressas sobre a forma de um modelo que os permita testar. Contrariamente, o prototipo de alta fidelidade

aparenta e funciona de forma muito semelhante ao produto final desejado. Os protótipos desenvolvidos e apresentados das secções seguintes são protótipos de baixa fidelidade.

4.5.1 Grafo de dependências

A página que apresenta o grafo de dependências, figura 4.4, é acessível através do menu de navegação do lado esquerdo do ecrã. Para a amostragem do grafo do dependências o utilizador necessita de selecionar o intervalo de tempo através da barra cronológica horizontal ao ecrã, é fornecida a possibilidade de o utilizador arrastar o intervalo escolhido ou alterar comprimento do intervalo selecionando apenas um limite do mesmo.
Após a escolha do intervalo de tempo e do dia correspondente a esse intervalo é apresentado o grafo das dependências dos serviços. Cada nó representa um serviço e cada aresta representa uma relação de dependências entre dois serviços, isto é, cada aresta representa o número de diferentes pedidos realizados na direcção da aresta e entre os respetivos serviços no intervalo de tempo previamente selecionado.

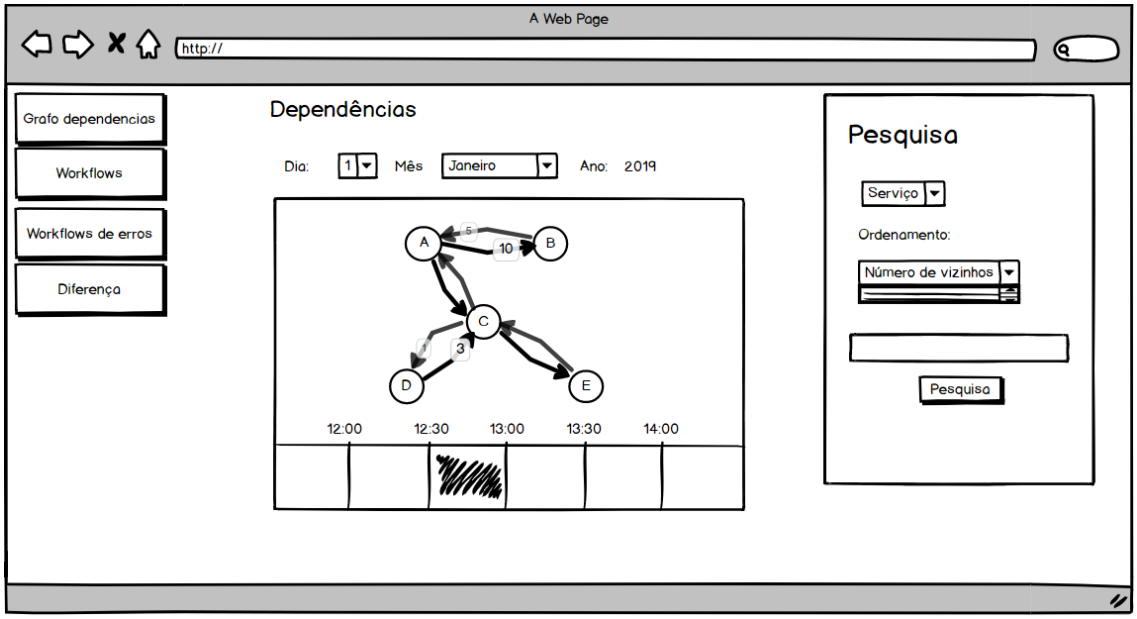

Figura 4.4: Página do grafo de dependências

É também de notar a possibilidade de fazer uma pesquisa por serviços ou instâncias na caixa de pesquisa fixa no lado direito. O utilizador pode pesquisar por serviços ou instâncias de acordo com uma ordenação pelas seguintes métricas: número de pedidos recebidos, número de vizinhos no grafo, número de diferentes tipos de pedido. A resposta da plataforma ao pedido de pesquisa é apresentada na figura 4.5, no exemplo da figura são ordenados os serviços ou instâncias pelo número de vizinhos.

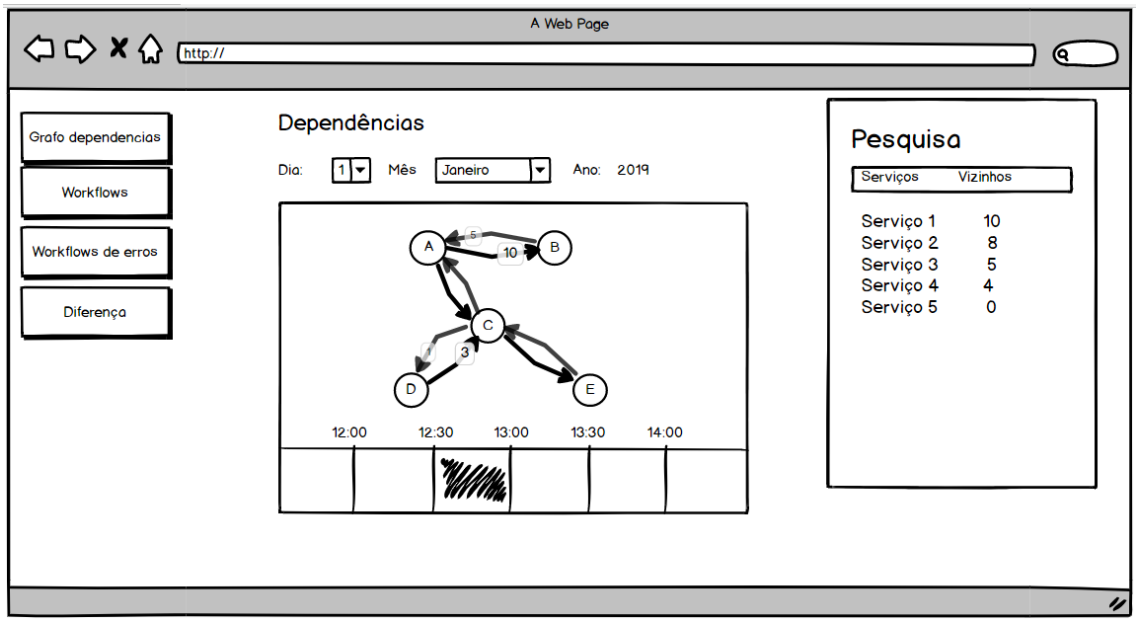

Figura 4.5: Página de uma pesquisa

4.5.2 Detalhes de um serviço

A partir da página do grafo de dependências de serviços, o utilizador pode fazer um *click* sobre um serviço do grafo, são apresentadas algumas métricas sobre o mesmo, nomeadamente: Número de vizinhos, número de tipos de pedidos, número total de pedidos, etc.

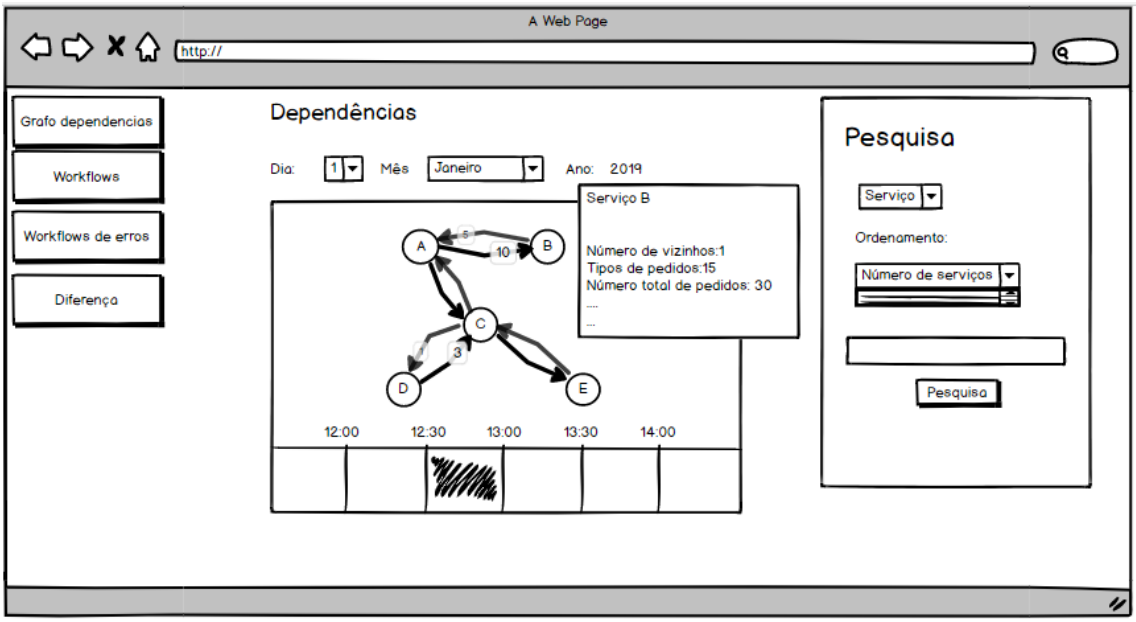

Figura 4.6: Detalhes de um serviço

4.5.3 Instâncias de um serviço

Em semelhança à página de detalhes dos serviços, a página da figura 4.11 é acessível pela página do grafo de dependências da figura 4.4. A ação que desencadeia a apresentação

da vista das instâncias é um duplo *click* no serviço pretendido.

As replicas do serviço do exemplo do protótipo são apresentadas sob a forma de árvore, para cada réplica é possível obter informações extra da mesma. As informações apresentadas são semelhantes às de um serviço, como na figura 4.6 com a adição da proporção de pedidos recebidos em relação ao número total de pedidos recebidos pelo serviço.

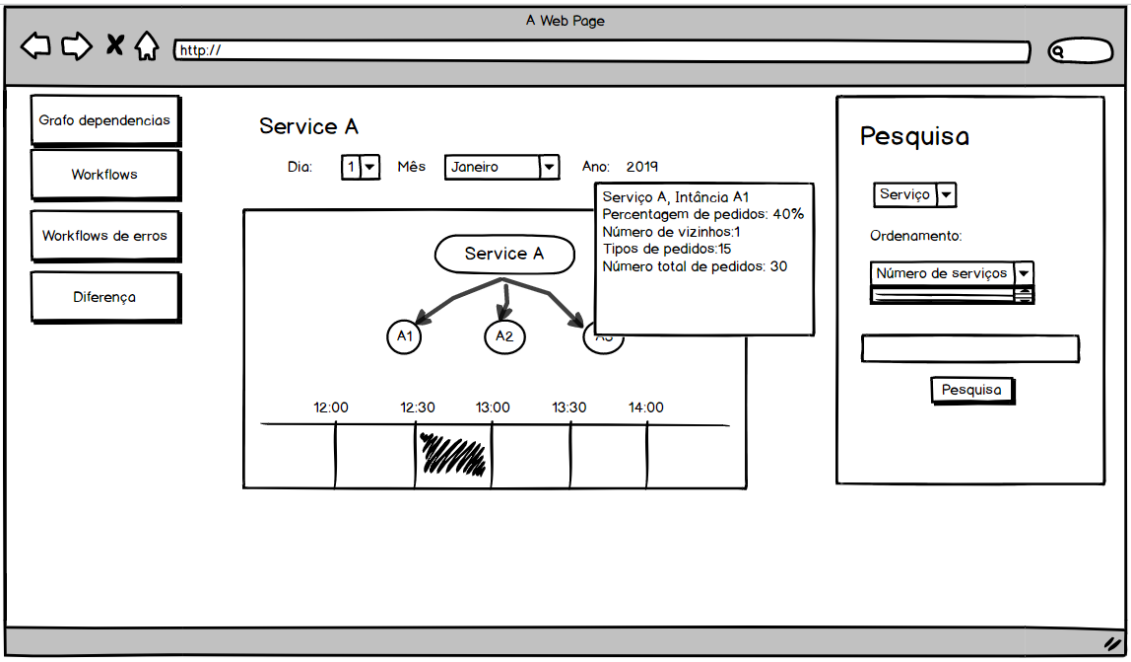

Figura 4.7: Instâncias de um serviço

4.5.4 Tipos de *flows*

A página de *flows* é acessível através do menu de navegação do lado esquerdo. Ao utilizador, após selecionar o intervalo de tempo desejado e o dia correspondente, são apresentados todos os diferentes *flows* existentes no intervalo de tempo selecionado. A funcionalidade de escolha do nome identificador do serviço permite ao utilizador escolher entre o nome do serviço, o *HTTP url* ou *end-point* do serviço. Na figura 4.9 a informação relativamente ao *flow* é facilitada ao utilizador, nomeadamente os *identificadores* dos traces pertencentes ao respectivo *flow*.

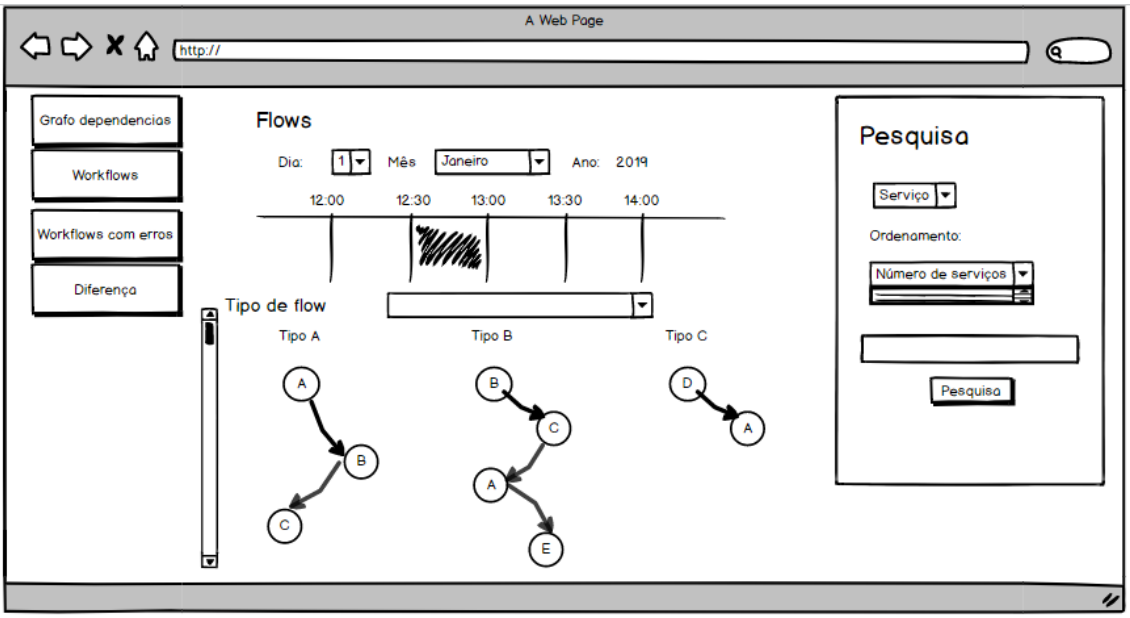

Figura 4.8: Tipos de *flows*

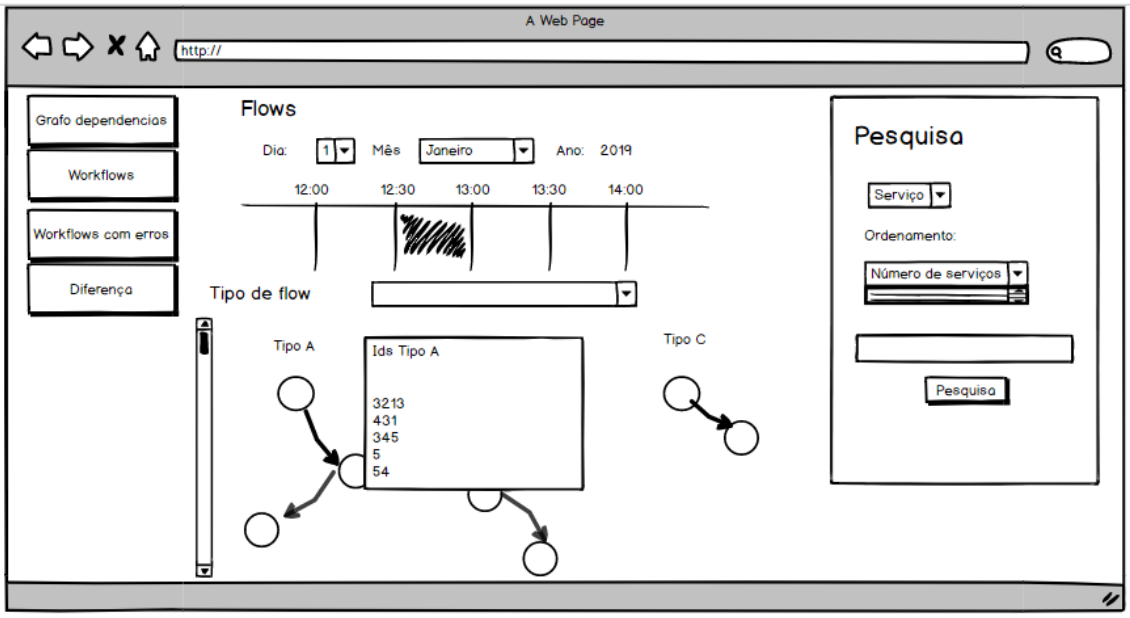

Figura 4.9: Detalhe de um serviço de um *flow*

4.5.5 *Flows* com falhas

O protótipo da figura 4.10 é a página de apresentação de *flows* com algum tipo de falha, em semelhança às páginas de apresentação de *flows* das secções anteriores, figura 4.8 e figura 4.9, é necessário a escolha do intervalo de tempo e do dia correspondente.

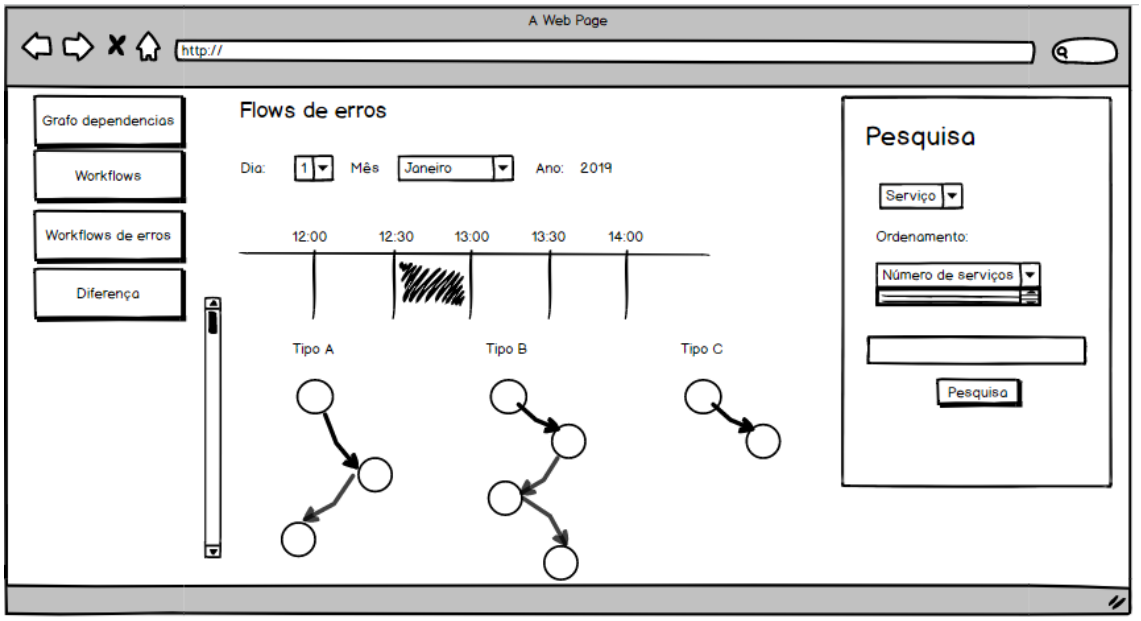

Figura 4.10: Flows com falhas

4.5.6 Comparação de grafos de dependência

De forma a cumprir os requisitos 7,8 e 9 é desenvolvida a funcionalidade de comparação da arquitetura dos microsserviços correspondente a dois intervalos de tempo diferentes. Após o utilizador selecionar os intervalos de tempo para a comparação, é apresentado num máximo de 3 segundos o grafo da diferença para os dois grafos.
Cada aresta é a diferença entre o número de pedidos dos serviços correspondentes nos dois intervalos de tempo selecionados. A operação pode devolver um resultado negativo devido à uma possível redução do número de pedidos. A cor associada a alguns nós representa o aparecimento ou desaparecimento dos serviços entre os dois intervalos de tempo, isto é, um serviço com cor vermelha deixou de ser chamado na transição para o intervalo de tempo posterior, o contrário acontece com um serviço de cor verde.

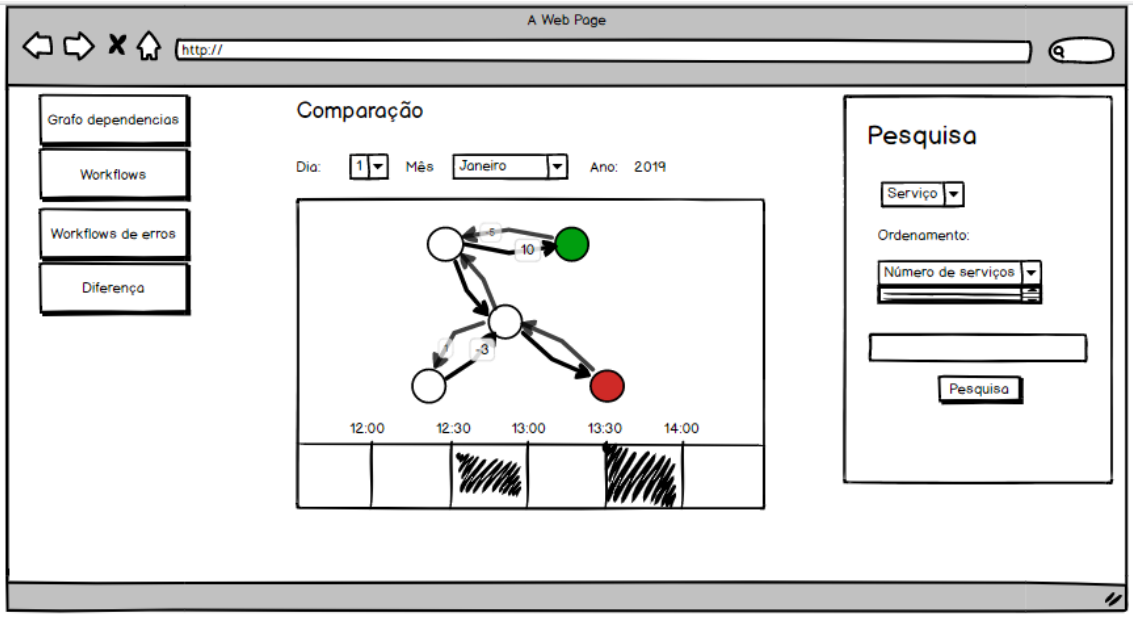

Figura 4.11: Diferença entre grafos

Capítulo 5

Implementação

Durante o segundo semestre do estágio foram desenvolvidos os requisitos estudados durante o primeiro semestre. Este capítulo descreve a parte prática do presente estágio.

A implementação da ferramenta de visualização de microsserviços pode-se dividir em dois trabalhos distintos, mas essenciais para o correto funcionamento da totalidade da plataforma. Assim sendo, foi desenvolvida a própria ferramenta que apresenta visualmente a arquitetura dos microsserviços e também, um servidor *REST* que cria e disponibiliza a informação que irá ser visualizada na ferramenta.

Nas secções seguintes, são apresentadas as soluções encontradas para os problemas identificados durante o período de desenvolvimento da ferramenta, mas também, são descritas as técnicas de integração dos vários módulos desenvolvidos.

5.1 Arquitetura implementada

A figura 5.1 representa o diagrama da arquitetura da solução final. Da arquitetura fazem parte duas compontes, um gerador de informação e a plataforma de visualização.

O gerador de informação simula grafos de dependências entre serviços de forma aleatória e fornece-os à plataforma de visualização. O gerador de informação expõe uma interface para a comunicação com a plataforma de visualização ou com outro tipo de sistemas que, no futuro, utilizem o gerador. De modo a providenciar informação, o gerador é responsável pela criação de dois tipos de dados diferentes. Em primeiro lugar, é criada a informação que constitui os grafos de dependências entre serviços ao longo do tempo. Em segundo lugar, é criada a informação relativa aos pedidos entre os vários serviços. De forma a armazenar persistentemente a informação, são usadas duas bases de dados distintas, uma base de dados desenvolvida para guardar grafos, a *ArangoDB*, e uma base de dados de séries temporais, a *InfluxDB*. A plataforma de visualização é diretamente usada pelo utilizador e faz uso das várias componentes apresentadas no diagrama da figura 5.1 para a construção de componentes visuais sobre a informação recolhida com recurso ao servidor *REST* desenvolvido. O utilizador é fornecido com uma série de funcionalidades que permitem, de um modo interativo, uma percepção das dependências dos serviços e instâncias ao longo do tempo.

Neste capitulo, são abordadas as partes constituintes do conjunto composto pelo gerador de informação e a plataforma de visualização me mais promenor.

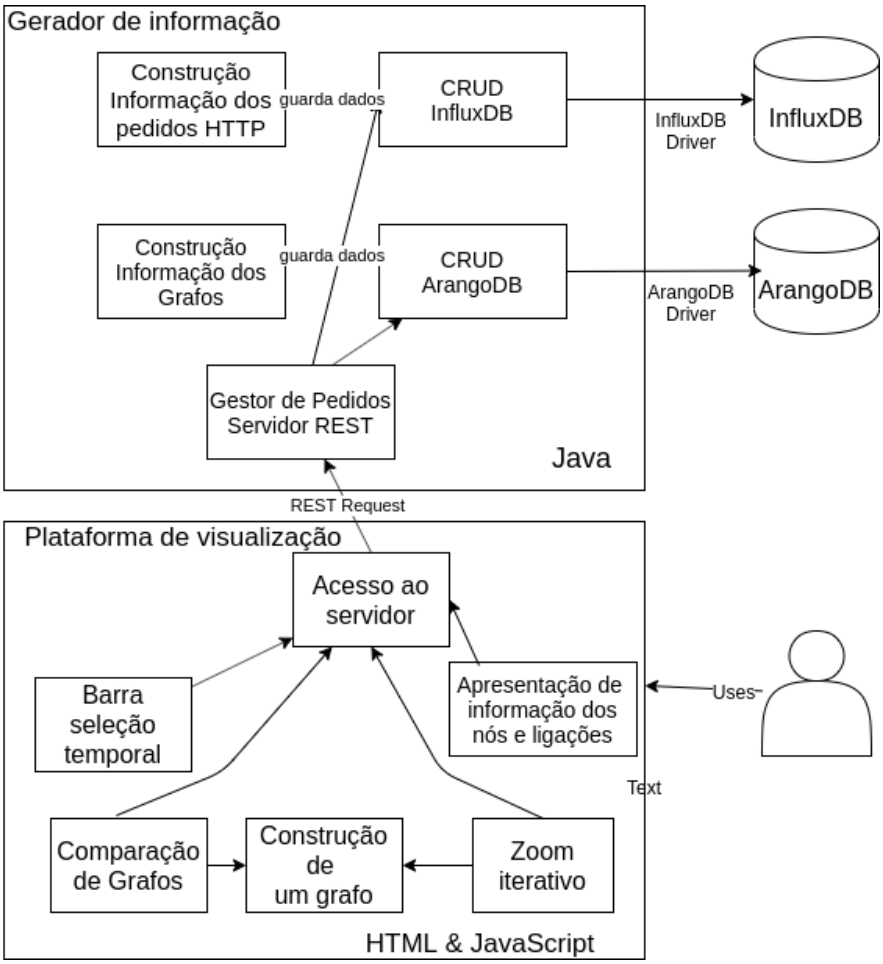

Figura 5.1: Arquitetura implementada

5.2 Especificação dos dados

Ao longo da implementação da ferramenta, surgiu a necessidade de definir uma especificação de dados de entrada da plataforma. Esta necessidade surgiu pelos seguintes fatores:

- Integração da plataforma de visualização da plataforma com o servidor de criação de dados.
- Estruturação da informação com o propósito de facilitar a visualização dos dados por parte do desenvolvedor para plataforma.
- Necessidade de normalizar a informação recebida. O desenvolvimento da plataforma foi desencadeado pelo estágio, no entanto, a plataforma de visualização de microserviços tem como objetivo final de permitir a fácil utilização da plataforma qualquer utilizador, a utilização da plataforma por um utilizador é facilitada pela leitura da especificação da informação de entrada da plataforma, desta forma, independentemente da tecnologia utilizada para extrair os dados do sistema implementado numa arquitetura em microserviços, uma serialização dos dados para especificação da plataforma permite a utilização da plataforma.

O formato de troca de informação entre o servidor de disponibilização da informação

e a plataforma de visualização é um ficheiro *JSON*, *Javascript Object Notation*. Como concorrência a esta tecnologia surge a notação *XML*. Devido à maior facilidade de visualização por parte de um ser humano de um ficheiro *JSON* em comparação com um ficheiro *XMl*, da maior compatiblidade da notação *JSON* com *Javascript* e do facto da maior rapidez de serialização de um ficheiro *JSON*, esta notação foi escolhida para o desenvolvimento do trabalho prático. Por outro lado, a maior experiência na utilização de notação *JSON* foi um fator muito determinante para esta escolha.

Existem três formatos de dados utilizados na integração entre a plataforma de visualização e o servidor de disponibilização de dados. Nas seguintes secções, cada uma das especificações é apresentada e analisada. É de notar que a especificação dos dados de entrada na ferramenta de visualização é muito semelhante ao formato dos dados guardados nos dois tipos de base de dados, nomeadamente, na base de dados de grafos, *ArangoDB* e na base de dados de series temporais, *InfluxDB*.

5.2.1 Informação sobre os grafos

```
{
 "nodes":
  [
   {
    "name": "instance0",
    "date": 0,
    "serviceName": "service4"
   },
   {
    "name": "instance1",
    "date": 0,
    "serviceName": "service3"
   }
  ]
}
```

Figura 5.2: Exemplo de um ficheiro *JSON* para os nós

A informação sobre as instâncias de um serviço é guardada num ficheiro *JSON* semelhante ao ficheiro em cima apresentado. Na aplicação de uma arquitetura numa plataforma de gestão de microserviços, por exemplo, *kubernets* ou *docker swarm*, cada serviço é replicado em instâncias com o objetivo de garantir o tempo de resposta do serviço dentro de certos parâmetros. Isto posto, cada instância de um serviço é identificada pelo parâmetro único *name*, é associado um nome de serviço, *serviceName* e também um identificador para o intervalo de tempo correspondente à informação, *date*.

A construção do grafo de arquitetura da aplicação em estudo é realizada a partir da informação, transmitida à ferramenta de visualização, presente num ficheiro *JSON* semelhante ao ficheiro em cima apresentado. A unidade mais básica na construção do grafo é a instância de um serviço, portanto, as ligações entre os serviços são representadas através das ligações entre várias instâncias de dois serviços. O ficheiro *JSON* em cima apresentado apenas representa a informação de uma ligação no grafo, entre uma instância fonte, *source*, e uma distância destino, *target*.

```
{
 "links":
  [
   {
    "source": "instance1",
    "target": "instance22"
   },
   {
    "source": "instance5",
    "target": "instance10"
   },
   {
    "source": "instance5",
    "target": "instance6"
   },
  ]
}
```

Figura 5.3: Exemplo de um ficheiro *JSON* para as ligações entre nós

A informação relativa aos diferentes pedidos *HTTP*, relacionados com a ligação do ficheiro *JSON* anterior, é apresentada num ficheiro à parte. Este é analisado de seguida.

5.2.2 Informação de pedidos *HTTP*

O ficheiro *JSON* anterior apresenta a informação das ligações entre a *instance1* e *instance22*, e relativa a um intervalo de tempo, identificado pelo parâmetro *timestamp*. A cada intervalo de tempo está relacionada uma lista de ligações, parâmetro *edge_influx_list*. Cada elemento da lista de ligações representa, entre duas instâncias, a informação dos pedidos *HTTP* agrupados de acordo com o seu código. Para cada tipo de pedido é associado o número de pedidos realizados durante o intervalo de tempo, *numberRequests*, e a média de tempo em milissegundos, *meanTime*. No exemplo, são representados os códigos 200 e 444.

5.3 Gerador de informação

No presente capítulo, capitulo 5.3, é analisado o gerador aleatório de informação desenvolvido a par do desenvolvimento da ferramenta de visualização.

Com base no trabalho prático anteriormente desenvolvido pelo grupo de trabalho orientado pelo professor orientador Filipe Araújo, surgem duas opções para a origem da informação para o desenvolvimento e teste da plataforma. Em primeiro lugar surge a possibilidade de reaproveitar o esforço realizado pelo aluno de doutoramento, André Bento, durante a sua dissertação de mestrado e desenvolver um servidor *REST* que disponibilizasse a informação por ele trabalhada. A segunda opção é a possibilidade de gerar dados de acordo com a especificação definida, no entanto de cariz arbitrária. Desta forma, é possível testar uma diversidade de informação gerada de acordo a intenção do desenvolvedor, e testar um maior número de casos de teste.

```
{
 "timestamp": 1577979022,
 "edge_influx_list":
   [
    {
     "from": "instance1",
     "target": "instance22",
     "HTTPCodesList":
      [
       {
        "code": "200 OK",
        "numberRequests": 73,
        "meanTime": 5
       },
       {
         "code": "444 Connection Closed Without Response",
         "numberRequests": 9,
         "meanTime": 58
           }
          ]
       }
  ]
}
```


Figura 5.4: Exemplo de um ficheiro *JSON* para a informação dos pedidos *HTTP*

A ferramenta de visualização tem como objetivo a visualização dos dados relativos à aplicação e às dependências que existem entre seus serviços. Os dados têm como fonte o utilizador que quer utilizar a aplicação, e pressupõe que a informação é retirada a partir de uma análise aos *traces* da aplicação. No entanto, a forma como uma aplicação é instrumentada para a obtenção dos dados diverge de aplicação para aplicação, devido à especificação utilizada, *openCensus* ou *openTracing*, e às escolhas realizadas por cada programador durante a instrumentação de uma aplicação. Desta forma, uma grande disparidade no formato de especificação pode ocorrer entre duas aplicações distintas. Para contornar este problema, a definição da especificação dos dados permite que qualquer utilizador após uma análise e transformação da informação relativa à sua aplicação em estudo, utilize a plataforma de visualização desenvolvida no corrente estágio.

Um dos problemas encontrados numa fase inicial da parte prática do estágio foi a obtenção da informação para suportar o desenvolvimento da aplicação, quer para guiar os objetivos principais, quer para testar as funcionalidades.

O mote para a criação do estágio descrito no documento foi uma falha existente no mercado na altura da dissertação de mestrado do colega André Bento. De forma resumida, durante o trabalho prático desenvolvido pelo colega, uma série de informação retirada de um conjunto de *traces* foi analisada e algumas métricas retiradas. No entanto, uma das dificuldades apontadas é a dificuldade de visualização da dependência entre os vários serviços.

Garantida uma uniformidade da informação de entrada, foi também definida uma interface à qual a ferramenta faz pedidos *HTTP* para a obtenção da informação, necessária para a visualização dos dados. Desta forma, e em conjunto com o gerador de informação

é também disponibilizado um servidor *REST*, criado de acordo com as especificações da ferramenta de visualização respeitando a especificação dos pedidos *HTTP* da plataforma. o funcionamento conjunto do gerador de informação e o servidor *REST* garantem o fornecimento de toda a informação à plataforma.

Nas secções seguintes, cada uma das partes constituintes do gerador de informação e do servidor *REST* são analisadas.

O desenvolvimento do gerador de informação sofreu um processo iterativo ao longo dos meses, devido à necessidade de alteração do seu funcionamento após discussão entre o grupo de trabalho. Por conseguinte, em primeiro lugar seriam desenvolvidas todas a componentes necessárias para a disponibilização da informação à plataforma, posteriormente a plataforma seria desenvolvida. Ambas as, anteriormente referidas, partes do trabalho prático foram desenvolvidas em simultâneo.

5.3.1 Funcionalidades do gerador de informação

O gerador de informação desenvolvido é uma ferramenta, desenvolvida na linguagem *Java*, que cria informação aleatória e simula a informação gerada a partir de dados provenientes de *traces* de uma aplicação instrumentada para tal efeito.

De forma a simular a informação que possa ser comparada à proveniente da recolha de *traces*, é realizado um processo iterativo de alteração da informação construída em cada iteração. Após a escolha de um intervalo de tempo por parte do utilizador, o gerador de informação cria a informação relativa ao grafo de serviços correspondente ao instante inicial, os serviços e as suas ligações são criadas como forma de um grafo. Criada a informação inicial, um ciclo de iterações é realizado como forma de simular a alteração da informação ao longo do tempo de vida do sistema. É de notar que, a alteração da informação corresponde ao aparecimento ou remoção de um serviço e/ou à adição ou remoção das ligações entre dois serviços. Outra característica que adiciona diversidade à informação criada é a possibilidade de não existir qualquer alteração entre iterações, como consequência, o intervalo de tempo correspondente à iteração é desprezado e uma nova iteração toma lugar.

Após serem realizadas todas as iterações de alteração de informação, ver figura 5.5, surge como resultado das operações realizadas a seguinte informação:

- Conjunto de datas correspondes a instantes que representam alguma alteração na informação.
- Conjunto de serviços e ligações para os intervalos de tempo em que a informação é mantida.

Toda a informação criada no gerador de informação é armazenada em duas bases de dados distintas para posterior consulta. Os detalhes da utilização das bases de dados são abordados numa secção seguinte.

O gerador de informação providencia ao utilizador duas opções para a customização da construção do grafo ao longo do tempo:

- Geração de um conjunto de informação que simula os pedidos entre os serviços de um sistema, para um intervalo de tempo escolhido pelo utilizador. A informação retirada dos *traces* do utilizador que, ocorrem durante um intervalo de tempo escolhido pelo

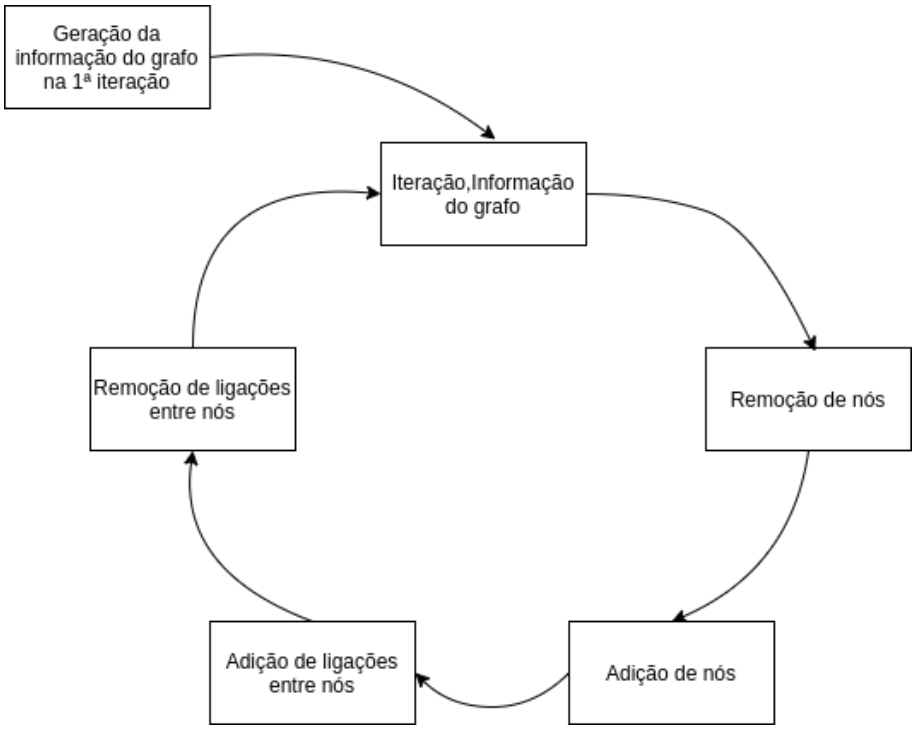

Figura 5.5: Ciclo iterativo de geração de um grafo ao longo do tempo

mesmo, é simulada no gerador de informação. Um dos requisitos básicos para a geração de informação é a escolha do intervalo de tempo, isto é, a escolha das datas de início e de fim da informação e do intervalo de tempo mínimo para a iteração dos dados.

- Escolha da probabilidade de alteração de informação ao longo do intervalo de tempo escolhido. Após a definição do intervalo de tempo, o utilizador define as probabilidades de alteração da informação de iteração para iteração. O utilizador pode escolher as seguintes probabilidades: probabilidade de adição de um novo serviço, probabilidade de remoção de um serviço, probabilidade de adição de um pedido entre serviços não ligados anteriormente e, probabilidade da remoção de todos os pedidos entre dois serviços.

5.3.2 Base de dados de grafos, ArangoDB

Após cada iteração de geração de informação, a informação relativa à estrutura do grafo de dependências de serviços, é armazenada permanentemente. Estes dados são guardados numa base de dados para o efeito, a *ArangoDB* [44]. A base de dados de grafos permite guardar dados relativos a um grafo, de tal forma que, as operações de escrita e leitura são otimizadas para o efeito.

Um dos requisitos necessários para armazenar a informação na base de dados é a sua instalação, neste caso, foi instalada a base de dados *ArangoDB* na versão *3.4.8*. Para além da instalação da base de dados, para que seja possível realizar chamadas à base de dados a partir do código em *Java*, foi também adicionado o *driver* de acesso correspondente à base de dados para projetos desenvolvido na linguagem *Java* [45].

De modo a manter o formato dos dados ao longo de todas as partes constituintes do trabalho prático, é maximizada a semelhança do formato de armazenamento e do for-

mato de transmissão dos dados em *JSON*. Desta forma, na base de dados é armazenada informação num formato semelhante ao ficheiro *JSON* que define um nó, no entanto, com o objetivo de manter a data correspondente ao grafo, um identificador referente à data é guardado em conjunto com cada coleção de nós e de ligações entre nós. Este identificador permite a consulta de coleções de nós e ligações filtrados por um intervalo de tempo.

As funcionalidades desenvolvidas para as operações de interação com a base se dados são:

- Criação de uma coleção de nós para um determinado intervalo de tempo, a coleção criada permite o armazenamento de nós.
- Escrita de um nó numa coleção.
- Leitura de um nó identificado por nome e intervalo de tempo, cada nó é armazenado numa coleção de acordo com o seu identificador temporal, assim, é possível identificar um nó pelo seu nome e pelo identificador temporal.
- Criação de uma colecção de ligações, de igual forma às coleções de armazenamento de nós, as coleções de ligações relacionam uma ligação a um intervalo de tempo.
- Escrita de uma nova ligação.
- Leitura de uma ligação filtrada por nó de origem e de destino.
- Leitura das ligações correspondes a um determinado nó.

5.3.3 Base de dados de séries temporais, InfluxDB

A informação que constitui os pedidos entre os serviços é armazenada numa série temporal, a base de dados *InfluxDB* [46]. Em auxílio da informação relativa à estrutura do grafo, é também guardada toda a informação para cada tipo de pedido realizado no sistema, a informação guardada nesta bases de dados segue o formato de dados dos ficheiro *JSON* de pedidos *HTTP*.

Devido ao facto de a informação dos pedidos entre cada serviço depender directamente da variável tempo, é necessário utilizar uma base de dados que facilite o armazenamento de informação com estas características. Na situação da informação relativa aos pedidos, existe sempre um *timestamp* associado a toda a informação, assim, pode ser descrita a informação dos pedidos *HTTP* entre os serviços como uma sequência das mesmas medições ao longo da variável tempo. .

No caso da base de dados de séries temporais foi necessário utilizar um *driver* que gestão de acesso à base de dados. No caso deste trabalho prático, foi utilizado um cliente para a *InfluxDb*,na linguagem *Java* [47].

A ligação entre dois serviços é armazenada na base de dados e identificada pelo identificador temporal. Para além deste, o identificador do serviço de origem e de destino é também um identificador de pesquisa. Em adição às operações de pesquisa são também disponibilizadas no gerador de informação as funções de escrita de ligações entre serviços.

5.3.4 Servidor REST

Como a ferramenta de visualização apenas apresenta dados de forma visual, esta faz pedidos constantes a um servidor a implementar pelo utilizador da plataforma. No caso

do trabalho prático, o gerador de informação armazena os dados nas bases de dados após a sua criação para que seja possível visualizar os dados. Numa fase inicial do estágio foi desenvolvido, com o auxilio da *frameWork Jersey*, um servidor *REST* para a consulta de informação.

O servidor *REST* disponibiliza uma interface de funções de consulta de informação das bases de dados. Assim, o servidor *REST* consegue servir informação constante à plataforma de visualização quando *online.*

Na tabela seguinte são analisadas todas as funcionalidades representadas na interface.

Função	**Descrição**	***Input***	***Output***
init	Desencadeia a criação de nova informação por parte do gerador de informação		Verificador de sucesso
getConfig	Usada após a geração de informação para obter configurações necessárias para posterior visualização dos dados		*Array* que contem todas as datas que correspondem à alteração de uma qualquer informação.
date	Usada para obter a informação relativa à constituição do grafo de serviços para um determinado intervalo de tempo	Identificador da data	Conjunto dos ficheiro *Json* para os nós e ligações
dateInflux	Usada para obter a informação relativa aos pedidos entre serviços para um determinado período de tempo	Identificador para o intervalo de tempo	Ficheiro *Json* para a informação dos pedidos *HTTP* entre serviços
end	Função que desencadeia o encerramento do servidor		

Tabela 5.1: Funcionalidades disponibilizadas pelo servidor *REST*

5.4 Ferramenta de visualização

A ferramenta de visualização disponibiliza uma série de funcionalidades que permitem ao utilizador visualizar a informação coletada pelo utilizador de alguma forma, idealmente coletada a partir da análise de traces.

Apesar de a plataforma ter sido desenvolvida partindo do pressuposto da utilização das tecnologias de *tracing*, a especificação da informação de entrada anteriormente analisada possibilita a visualização de qualquer informação que cumpra os requisitos necessários para preencher a especificação. Cada funcionalidade principal da plataforma é uma tenta-

tiva de resolver os problemas de complexidade associado à informação a representar. Nas secções seguintes é realizada uma análise sobre cada uma das funcionalidades, os métodos de desenvolvimento associados a cada uma, bem como os problemas solucionados por cada funcionalidade.

Para uma melhor análise da ferramenta, esta pode-se dividir em várias constituintes. Cada uma das partes que compõem a ferramentas auxilia-se de diferentes *frameworks* e soluções, cada uma das seguintes secções representa uma destas mesmas partes da plataforma.

A plataforma de visualização é uma aplicação *Web* desenvolvida em *HTML*, *CSS*, *JavaScript* e também com auxílio da *framework Bootstrap*. A principal razão pela escolha do desenvolvimento da ferramenta na linguagem *JavaScript* é a maior compatibilidade das ferramentas de construção automática da disposição dos nós de um grafo. Desta forma, como a ferramenta utilizada para auxiliar a construção dos grafos de dependências é baseada em *JavaScript*, optou-se por usar esta linguagem como base para todo o desenvolvimento.

5.4.1 Funcionalidades da ferramenta

A plataforma de visualização tem como foco a construção de grafos interativos com base em informação disponibilizada por um servidor, neste caso por um servidor de geração de informação aleatório no caso do trabalho prático desenvolvido. A construção dos grafos de dependências permite a análise de algumas diferentes características da informação. Portanto as funcionalidades que se podem destacas são as seguintes:

- Exploração dos conjuntos de instâncias e serviços
- Apresentação da diferença entre grafos num intervalo de tempo.
- Exploração do grafo das instâncias agrupadas por serviços
- Exploração do grafo das instâncias.
- Apresentação da informação relativa a cada serviço, instância ou ligação entre os mesmos.

5.4.2 Construção dos grafos

Devido à dificuldade inerente à construção total de um grafo, isto é, definir a posição de cada nó de forma a minimizar a sobreposição de nós e links e maximizar a ordem lógica do mesmo, é necessário auxiliar o trabalho prático com uma ferramenta que construa automaticamente um grafo.

O posicionamento dos nós do grafo e das suas ligações, é definido pela ferramenta D3 [40]. Pode-se definir a ferramenta D3 como uma biblioteca para a linguagem *JavaScript* que permite relacionar dados arbitrários a um objeto *DOM*. A estes documentos é possível aplicar transformações de forma a criar tabelas relativas à informação. Principalmente, é possível criar gráficos que representam e facilitam a analise da informação.

A construção dos grafos foi realizada pelo módulo de representação de grafos da ferramenta *D3* que realiza cálculos matemáticos sobre forças entre nós. Simulando cada

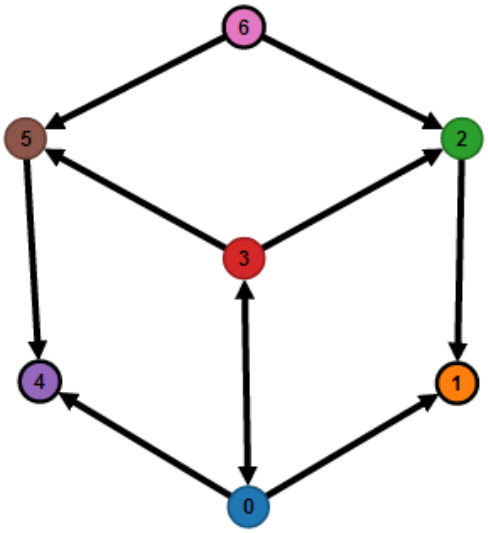

Figura 5.6: Forças simuladas na ferramenta *D3*

nó como uma partícula e cada ligação entre nó como uma força de repulsa ou atração, é possível construir um grafo com representação lógica e de leitura eficiente.

Usando o exemplo da imagem 5.6, consultada na documentação da ferramenta *D3* [48] , é possível estudar como as forças simuladas numa ligação entre grafos permitem a construção lógica de um grafo. Estudando o nó número 6, cujos vizinhos são os nós número 5 e 2, este nó afasta os seus vizinhos para uma certa distância. No entanto existe também uma força de atracão que impede estes se afastem para além de uma determinda distância dendente da força aplicada. Para além das forças direccionadas a cada um dos nós, existe também uma força global que centra os serviços na imagem e mantém o grafo com dimensões adequadas.

Após algum breve estudo teórico e prático sobre as possibilidades de customização na construção do grafo, foram definidas as forças e condições a aplicar na criação dos grafos da ferramenta. Assim, foram definidas as seguintes condições na simulação da ferramenta *D3*:

- Força de atracão ao centro da imagem.
- Força de repulsa das laterais da imagem
- Força atracão de nós interligados.
- Força de repulsa de nós que impede a colisão de nós.
- Tempo da simulação
- Velocidade da simulação
- Tamanho base dos nós

As condições de simulação do grafo na ferramenta *D3* são dependentes entre si, uma vez que, para uma construção suave de um grafo que seja fácil de compreender por um humano, é necessário ajustar iterativamente as condições até se atingir o equilíbrio desejado.

De modo a usar a ferramenta de simulação de forças diretas entre nós, é necessária uma transformação dos dados para a especificação de entrada da ferramenta, ver ficheiro *JSON* 5.4. Para alem da especificação dos dados ser diferente em comparação à usada na ferramenta de visualização, também é necessário uma serialização para informação em formato JSON. Esta normalização dos dados permite que a toda a simulação da construção do grafo seja totalmente feita pela plataforma de visualização.

```
{
  "nodes":
        [
            {"id": "instance1", "service": "service1"},
            {"id": "instance2", "service": "service1"},
        ]
    "links":
        [
            {"source": "instance1", "target": "instance2"},
        ]
}
```

Ficheiro *JSON* 5.1: Exemplo de um ficheiro *JSON* de entrada da ferramenta D3

Após a construção total do grafo, são várias as ações desencadeadas. É adicionado um evento de *click* do rato para os nós e ligação para a apresentação de informação relativa ao objeto clicado.É adicionada uma funcionalidade de zoom, arrastamento dos nós, e por fim o realçamento de um nó e seus vizinhos no caso de o rato estar por cima de um nó.

Para que seja possível manipular a informação representada no ecrã, a ferramenta de construção dos grafos representa cada objeto com recurso a SVG, gráficos vetoriais escaláveis. Uma característica de um SVG é a criação simultânea de um objeto DOM bastante personalizável. Aos objetos DOM é possível adicionar eventos, alterar o seu estilo, alterar o seu tamanho de forma a manter a responsividade da ferramenta.

Como resultado da aplicação das funcionalidades da ferramenta *D3*, são construídos vários grafos com diferentes objetivos. No entanto, para a apresentação de qualquer grafo na plataforma é usada a mesma função de visualização, apenas ajustando alguns parâmetros com o propósito de criar um grafo de fácil leitura. Na figura é apresentado um exemplo de um grafo construído na plataforma e que pode representar de forma geral a construção dos diferentes grafos.

5.4.3 Realçamento de um nó

Com recurso às funcionalidades disponibilizadas pela ferramenta *D3* foi desenvolvida a funcionalidade de realçar um nó e seus vizinhos no caso de o cursor passar por cima de um nó. A figura 5.8 é um exemplo desta funcionalidade na plataforma. O rato encontra-se por cima do nó identificado por *instance3*, tal acção leva a ferramenta a aumentar a transparência dos restantes nós, restando apenas os nós em evidencia. Após o cursor deixar de desencadear este evento, todos os nós têm a mesma transparência.

A funcionalidade de realçar os nós de um grafo e seus vizinhos é particularmente importante pois permite identificar facilmente os vizinhos de um nó no caso de alguma anomalia nos tempos de resposta do serviço.

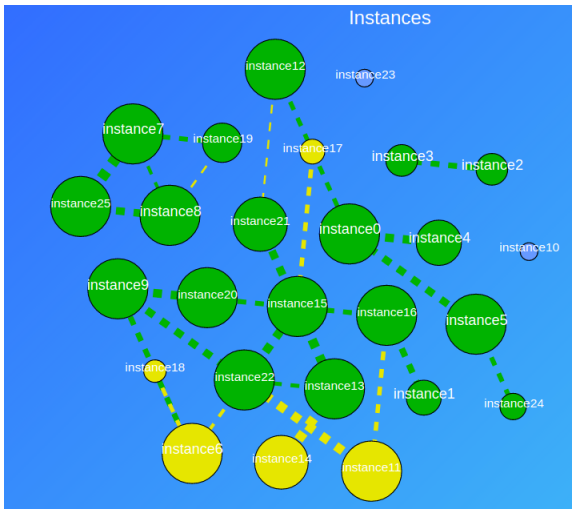

Figura 5.7: Grafo exemplo da plataforma

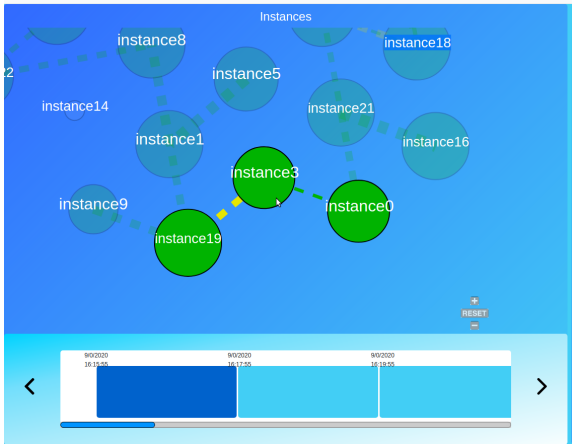

Figura 5.8: Realçamento de um nó

5.4.4 Grafo iterativo

Umas das principais funcionalidades desenvolvidas na plataforma é apresentação de um grafo iterativo, isto é, um grafo que se modifica ao longo da ampliação do grafo com recurso ao deslizamento do rato. A operação de *zoom* com o rato desencadeia a modificação do grafo de acordo com uma ordem lógica, desta forma, em primeiro lugar é apresentado um grafo que representa um conjunto de serviços, em segundo lugar é apresentado o agrupamento de instâncias em serviços e por último são apresentadas todas as instâncias do grafo. A razão fundamental que justifica o desenvolvimento da funcionalidade em questão é a necessidade de agrupar informação numa situação de existência de elevado número de nós. Este problema é, de facto, uma das questões levantadas durante uma fase inicial do estágio. Para solucionar o problema da representação foram desenvolvidas duas técnicas de agrupamento de informação utilizadas no grafo iterativo.

O primeiro agrupamento realizado sobre as instâncias existentes é o agrupamento por serviço, este agrupamento reduz significativamente a quantidade de informação apresentada simultaneamente no ecrã e facilita a perceção das dependências entre serviços e instâncias. O segundo agrupamento realizado na funcionalidade do grafo iterativo é a agrupamento de serviços de acordo com o algoritmo de identificação de componentes fortemente conexas.

Figura 5.9: Ordem de aparecimento dos grafos na funcionalidade de Grafo iterativo

Nas secções seguintes são abordados os grafos apresentados na funcionalidade anteriormente apresentada.

Identificação de serviços fortemente conexos

Sabendo que podem existir situações nas quais o número de serviços e instâncias de serviços é demasiado elevado para a representação total da informação existente, um agrupamento lógico da informação apresenta-se como requisito no desenvolvimento da plataforma. Como a única informação existente que relaciona serviços e instâncias são as suas ligações no grafo, usando esta informação como princípio de agrupamento, a solução encontrada para aplicar um agrupamento lógico dos serviços é a identificação de componentes fortemente conexas.

O algoritmo de Tarjan [49] pressupõe que num grafo existem subgrafos que são fortemente conexos, isto é, qualquer nó do grafo está ligado a todos os restantes nós de forma direta ou indireta. Uma componente fortemente conexa de um grafo é o maior subgrafo fortemente conexo.

O algoritmo de *Tarjan* identifica componentes fortemente conexas de um grafo através da pesquisa em profundidade num grafo, como auxílio à pesquisa em profundidade são mantidas variáveis em cada nó, representando o início de uma componente conexa. A recursão de cada pesquisa no grafo permite identificar a raiz de uma componente. A figura 5.10 representa visualmente a aplicação do algoritmo de *Tarjan* para a identificação de componentes fortemente conexas, é de notar que para cada nó é guardada a informação relativa ao nó raiz da componente e ao nó imediatamente anterior no caminho entre o nó em questão e a raiz da componente. No exemplo da figura, a aplicação do algoritmo de *Tarjan* permitiu agrupar os 8 nós em apenas 3 componentes fortemente conexas.

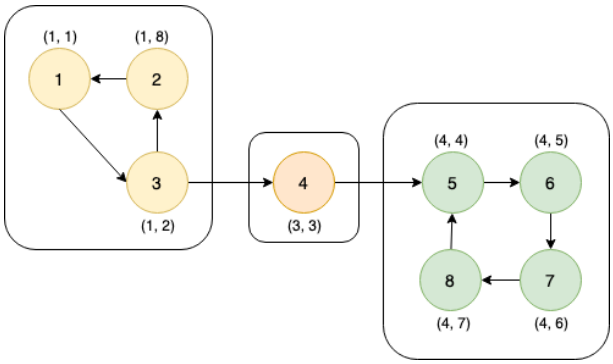

Figura 5.10: Aplicação do algoritmo de *Tarjan* [50]

Para aplicação do algoritmo de identificação de componentes fortemente conexas de *Tarjan* nos grafos mantidos na plataforma é necessária uma adaptação do mesmo com fim

de manipular as classes de dados que representam instâncias, serviços ou agrupamento de serviços.

A adaptação do algoritmo de *Tarjan* ao trabalho prático permite reduzir de forma significante o número de informação a apresentar. Por outro lado, a aplicação do algoritmo só é justificada num eventual número elevado de serviços, desta forma, foi definido um limite inferior de 20 instâncias para o agrupamento de serviços através do algoritmo de *Tarjan*.

A figura 5.11 representa a aplicação do algoritmo de *Tarjan* num grafo de serviços para um determinado intervalo de tempo. Na situação representada através dos grafos da figura, existem 30 instâncias, 10 serviços e 6 agrupamento de serviços. Com recurso ao exemplo da figura e dos testes realizados ao longo do semestre, o rácio entre o número de componentes existentes e número de serviços varia significantemente com a probabilidade de um nó estar ligado a um outro nó, pelo que não existe uma estimativa objectiva. Esta propriedade ocorre devido ao facto de o algoritmo de *Tarjan* identificar conjuntos de grafo fortemente ligados entre si, assim, quanto maior for o número de ligações num grafo, maior é a taxa de agrupamento deste algoritmo.

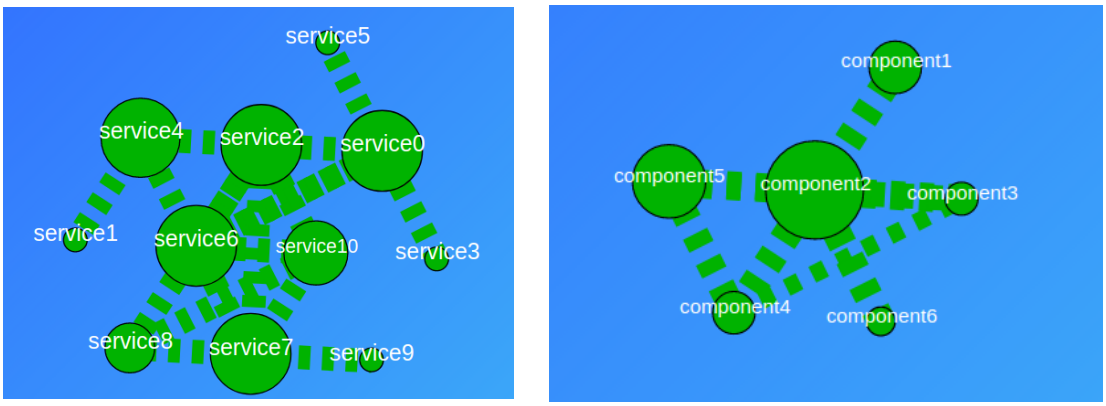

Figura 5.11: Aplicação do algoritmo de *Tarjan* ao grafo de representação de serviços

5.4.5 Comparação de grafos

Um dos requisitos analisado durante o primeiro semestre do estágio e documentado no documento é a possibilidade de comparação de grafos relativos a dois intervalos de tempo diferentes. Numa fase inicial de desenvolvimento da aplicação, este mesmo requisito foi cumprido, como se pode verificar na figura 5.12, a figura representa um grafo de comparação entre a informação relativa a dois intervalos de tempo escolhidos.

A representação da diferença entre dois grafos compreende uma série de passos necessários para a realização da funcionalidade, em primeiro lugar os dois grafos são comparados e são identificadas as diferenças em termos de existência de nós e ligações, após o cálculo da diferença, toda a informação de ambos os grafos é reunida e anotada para posterior representação visual. Na situação em que o utilizador escolhe a funcionalidade de comparação de grafos e também são escolhidos dois intervalos de tempo diferentes, é apresentada a informação relativa a uma diferença lógica entre os grafos. A diferença realizada na presente funcionalidade pode-se representar pela diferença no conjunto de nós e ligações dos intervalos selecionados. Assim, pode-se representar a funcionalidade da comparação de dois grafos distintos pelas seguintes características:

- Nó de coloração verde, o nó não existia no primeiro grafo por ordem cronológica, foi adicionado ao sistema.

- Nó de coloração vermelha, o nó não existe no segundo grafo por ordem cronológica, o nó foi eliminado do sistema.
- Nó de cor azul significa a não alteração entre os dois intervalos.
- Ligação com coloração verde, a ligação foi adicionada ao sistema.
- Ligação de coloração vermelha, a ligação foi eliminada do sistema.
- Ligação de cor branca, não existe nenhuma alteração na ligação

A coloração existente nos nós e ligações do grafo permite ao utilizador uma fácil perceção da evolução do grafo ao longo do tempo e consequentemente uma análise dos grafos por sua parte.

Figura 5.12: Diferença entre de grafos de dois intervalos de tempo

5.4.6 Apresentação da informação de nós e de ligações

Para além da informação representada nos vários da plataforma, ao utilizador é também dada a opção de obter informação extra relativa a um nó ou a uma ligação. Desta forma, pode-se dividir a apresentação de informação extra em duas categorias, a primeira relativa à informação de uma instância, serviço ou conjunto de serviços e a segunda relativa à informação de uma determinada ligação entre nós. É de notar que a informação é apresentada na barra do lado direito da imagem e que, o evento que origina o seu aparecimento é o clique com o botão esquerdo do rato num nó ou numa ligação.

A figura 5.13 é o resultado do clique por parte do utilizador no nó representativo do serviço identificado por *service0*. No caso da ocorrência deste evento, são apresentadas as seguintes informações relativas ao nó em causa:

- Elementos representados pelo nó. Isto é, no caso em que o nó é um serviço, são apresentadas as suas instâncias. Por outro lado, no caso em que o nó é um conjunto de serviços, são apresentados os serviços constituintes do conjunto. É de alguma importância notar que, para cada elemento apresentado após o clique no botão de menu, existe a possibilidade de desdobrar a informação com um máximo de 2 iterações.

- Vizinhos do nó. De forma semelhante ao ponto anterior, são apresentados os nós vizinhos do nó em causa. Para melhor realçar os vizinho de um nó no grafo, estes são realçados sempre que o ponteiro do rato se encontra na caixa de informação de um nó.

- Apresentação de métricas relativas aos pedidos originados pelo nó e aos pedidos direcionados ao nó. As métricas apresentadas são as seguintes: tempo de resposta médio, rácio entre o tempo de resposta médio e a média geral dos tempos de resposta, número total de pedidos, e racio entre o número total de pedidos e a média geral do número de pedidos em cada nó.

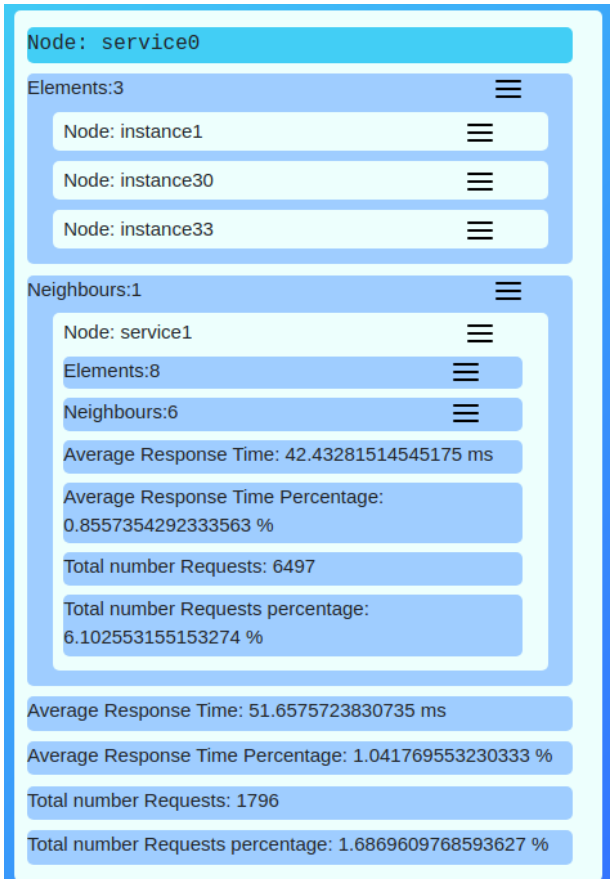

Figura 5.13: Apresentação da informação de um nó

Na figura 5.14 é apresentado o resultado do clique do utilizador numa ligação entre dois nós, neste caso, entre o nó *service12* e o nó *service0*. A área de informação relativa a esta ligação apresenta a seguinte informação:

- Nome da ligação em causa.
- Nó de origem da ligação
- Nó alvo da ligação, a informação criada relativa aos nós dentro das suas respectivas caixas, o nó de origem e nó alvo, pode ser expansível da mesma forma como os elementos pertencentes a um nó ou como os vizinhos de um nó.
- Métricas relativas à ligação, a informação apresentada é muito semelhante às métricas apresentadas na caixa de informação de um nó.
- Pedidos *HTTP* da ligação. É também possível expandir a informação relativa aos pedidos *HTTP*, cada pedido é agrupado pelo código *HTTP* e para cada código é apresentado o número total de pedidos e a média do tempo de resposta geral.

A informação anteriormente descrita é apresentada do lado direito da imagem de forma persistente, isto é, uma barra de deslocamento é adicionada quando a informação ocupa mais que o espaço disponível na secção dedicada para o efeito, uma das razões para a escolha deste tipo de funcionamento é o aumento da interação por parte do utilizador.

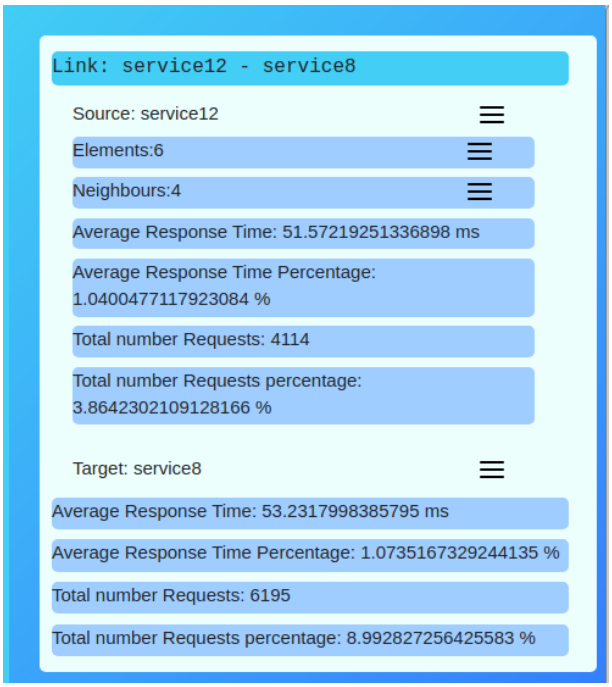

Figura 5.14: Apresentação da informação de uma ligação

5.4.7 Apresentação da informação dos pedidos *HTTP* no grafo

Como forma de facilitar a percepção da importância de um grafo na arquitetura do sistema, foi adicionadas funcionalidades à plataforma. Partindo do exemplo da figura 5.8,é possível visualizar os diferentes tamanhos dos nós do grafo. Esta característica assenta no

facto de que cada nó tem um número de pedidos variados, desta forma, é possível calcular a percentagem de pedidos que têm origem ou destino num nó em especifico. A percentagem calculada permite alterar o tamanho dos nós. Desta forma, definindo um limite superior e inferior de tamanho, um nó com um tamanho reduzido apresenta um baixo número de pedidos *HTTP*, de forma contrária, um nó de tamanho elevado apresenta mais pedidos *HTTP*.

Outra característica visível na figura são as diferentes cores dos nós e das lgações. A cor representa o tempo de resposta médio de um nó ou de uma ligação. Assim, um elemento com cor verde apresenta tempos de resposta médios dentro da média geral, cor amarela representa um tempo de resposta um pouco maior que a média geral, cor vermelha indica um tempo de resposta significativamente maior que a média geral da plataforma.

5.4.8 Barra de selecção de intervalos de tempo

A barra de seleção temporal encontra-se na parte inferior do ecrã e está presente em todas as ações despoletadas pelo utilizador. A figura 5.15 representa a seleção de dois intervalos de tempo, esta opção apenas está disponível na funcionalidade de comparação de grafos. As funcionalidades restantes apresentam informação visual relativa a apenas um intervalo de tempo, desta forma, apenas é possível escolher um intervalo de tempo.

É visível na imagem os diferentes tamanhos de cada intervalo, isto deve-se ao facto da relação proporcional entre o tamanho dos intervalos e do tempo que representam. Desta forma, existe uma perceção do tamanho real dos intervalos de dados. Com fim de representar os intervalos com tamanhos adequados ao tamanho do ecrã e espaço disponível para a barra temporal, foi definido um tamanho mínimo para o menor intervalo de dados. Desta forma, os RESTantes intervalos de tempo são representados por uma secção com tamanho proporcional ao intervalo menor existente.

Com objetivo de melhor a navegabilidade por parte do utilizador, foram adicionados botões e barras de deslizamento para uma fácil seleção do intervalo desejado.

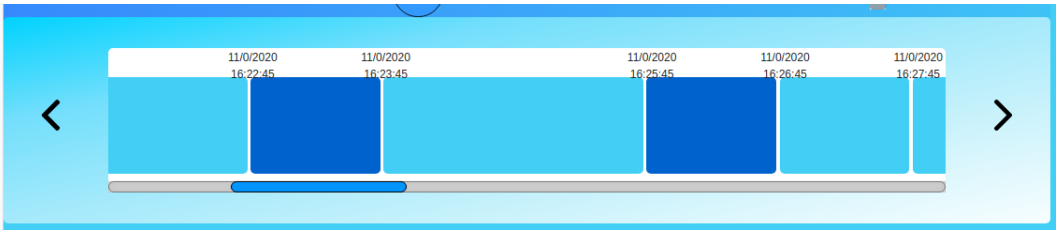

Figura 5.15: Barra de seleção de intervalos de tempo

5.4.9 Funcionalidades extra

Para além das funcionalidades principais descritas nas secções anteriores, foram acrescentadas algumas funcionalidades extra para melhorar a experiência do utilizador, sendo assim, as funcionalidades adicionadas são as seguintes:

- Arrastamento de um nó no grafo, em qualquer grafo apresentado na plataforma, o utilizador por mudar a sua posição no ecrã com auxílio do ponteiro do rato. Esta funcionalidade é particularmente importante para a análise dos grafos pois, existem grafos que são construídos de forma a que não se consiga entender as ligações entre

os nós. Este facto ocorre devido ao elevado número de nós ou ao cruzamento entre ligações.

- Possibilidade de fazer *zoom* nos grafos. Esta funcionalidade foi adicionada numa fase adiantada do trabalho prático. Para um utilizador fazer um *zoom in* ou *zoom out* num grafo, pode optar por duas opções de interação. A primeira é o vento de deslizamento do rato, a segunda são os botões adicionados no canto inferior direito de cada grafo.

5.4.10 Decisões realizadas

A definição da especificação da informação de entrada na plataforma foi realizada de uma forma iterativa ao longo do tempo de desenvolvimento. Isto porque, em discussão com o professor orientador e colegas do grupo de trabalho foram surgindo algumas informações a acrescentar ou a alterar de alguma forma. Assim sendo, algumas decisões de elevada importância para o desenvolvimento do trabalho prático forma realizadas. Podem-se destacar as seguintes:

- Separação da informação sobre os nós, ligações e os pedidos *HTTP*. Esta separação surgiu pela necessidade de simplificar e modelar toda a informação. Assim sendo, separou-se a informação relativa à constituição do grafo das ligações entre várias instâncias e a informação relativa aos pedidos entre as mesmas.

- Unidade básica para a representação no grafo. Os pedidos são representados na ferramenta como uma ligação entre duas instâncias, isto porque, as instâncias de um mesmo serviço não apresentam obrigatoriamente os mesmos pedidos. No entanto, as ligações entre instâncias podem ser agrupadas por serviços. Desta forma, a unidade básica da estrutura do grafo poderia ser o serviço. Atendendo às duas opções anteriormente apresentadas, optou-se por representar as ligações entre serviços como uma ligação entre duas instâncias como forma de não perder nenhum tipo de informação no agrupamento das ligações por serviços.

- Agrupamento dos pedidos *HTTP* por código. A ligação entre duas instâncias representa um conjunto de pedidos individuais com características em com que garantem o agrupamento em instâncias. Os pedidos representados numa única ligação partilham o intervalo de tempo de ocorrência, a instância de origem e de alvo do pedido. Outra característica que permite agrupar pedidos é o tipo de código *HTTP* originado. Assim sendo, como o número de pedidos com o mesmo código pode ser muito elevado, estes são agrupados e é apenas guardada a informação relativa ao número total de pedidos e, o tempo médio de resposta dos mesmos.

Capítulo 6

Conclusão e Trabalho futuro

O estudo dos conceitos fundamentais para a contextualização à teoria relacionada com o trabalho a ser desenvolvido, permitiu adquirir conhecimentos fundamentais para as fases seguintes da dissertação. Dos conceitos estudados, podem-se destacar a containeriazação e a monitorização das aplicações de software. O conceito de monitorização é principalmente importante no contexto do problema pois permite adquirir uma visão distribuída e isolada da implementação de software. As arquiteturas em microsserviços e que irão ser analisadas pela aplicação a desenvolver no segundo, utilizam soluções de containerização na implementação dos vários serviços pertencentes à aplicação. Sendo assim, o estudo deste conceito apresenta-se como bastante importante no contexto do problema. Além do conceito de containerização, o estudo do conceito de microsserviço permite adquirir conhecimentos sobre as arquiteturas monitorizadas pelos *traces*, no entanto, a solução visa permitir a visualização dos *traces* independentemente da sua origem. Assim, o estudo deste contexto é importante, mas não desempenha um papel principal entre os restantes.

A análise a manutenção de aplicações, nomeadamente aos conceitos de *logging*, monitorização e *tracing*, surgiu na necessidade de estudar as principais técnicas de manutenção e de verificação do estado de aplicações de *software*, das técnicas anteriormente enumeradas, a prática de recolher *traces* de aplicações de *software*, permite, principalmente em sistemas distribuídos, o acompanhamento do desencadeamento de pedidos provocados por uma determinada ação. Desta forma, no caso de uma propagação de um erro pelos vários serviços, é possível reverter o desencadeamento de pedidos e encontrar a fonte do mesmo.

Durante o primeiro semestre foi realizado um estudo sobre o estado actual da arte, relativamente às ferramentas de observação da estrutura de uma aplicação desenvolvida utilizando o conceito de microsserviço. Após uma análise a estas ferramentas, pode-se concluir que a sua capacidade de analisar as aplicações utilizando *traces* não é totalmente garantida pelas mesmas. Apesar de algumas das ferramentas de monitorização de performance de aplicações apresentarem a estrutura da arquitetura, após alguma investigação, e acrescentando o facto de a informação encontrada sobre o assunto é bastante escassa, pode-se concluir que o suporte deste tipo de aplicações à análise de *traces* com especificação *OpenTracing* ou *OpenCensus* é escassa na maioria das aplicações.

O estudo das necessidades existentes atualmente relativamente à visualização da informação presente em *traces* desencadeou a definição dos requisitos, estes definem características da plataforma que combatem as lacunas identificas ao longo do estudo teórico. A definição dos requisitos e da arquitetura proposta numa fase prévia à implementação da solução desempenhou um papel de guia ao longo de toda a fase de implementação do trabalho prático.

Numa fase inicial do segundo semestre, o problema da origem dos dados para teste da plataforma foi levantado nas discussões do grupo de trabalho. Posto este problema, duas soluções foram identificadas e as suas características analisas, desta forma podem-se detalhar cada uma das soluções: a primeira solução surge como o uso do trabalho prático desenvolvido pelo colega André Bento durante a sua dissertação de mestrado com fim de fornecer os dados recolhidos no seu trabalho prático à plataforma de visualização. Por outro lado, foi levantada a solução de geração de dados com características semelhantes aos recolhidos na análise de *traces* mas de forma aleatória, uma das vantagens da utilização deste método é a maior possibilidade de divergir os dados. Considerando as duas opções, foi privilegiada a maior diversidade de informação em relação à maior veracidade dos dados recolhidos num trabalho prático anterior. Após a fase de desenvolvimento estar completa, a opção tomada mostrou-se como correta pois a maior elasticidade no tipo de dados fornecidos à ferramenta de visualização foi uma vantagem durante todo o desenvolvimento, uma vez que a ferramenta foi testada numa maior diversidade de dados.

Considerando a opção tomada, durante uma fase inicial do tempo disponível para a implementação, foi desenvolvida uma ferramenta de geração de dados. Esta ferramenta fornece a informação de um sistema distribuído, a principal característica de um sistema desenvolvido de forma distribuída é a comunicação entre serviços independentes entre si e autónomos. Esta característica define o tipo de informação criada no gerador de informação, assim, pode-se estruturar a informação criada e fornecida à plataforma de visualização como um grafo de comunicação entre serviços ou instâncias de serviços.

A existência do gerador de informação distingue a arquitetura da solução implementada em duas componentes interligadas, a comunicação entre o gerador e a plataforma de visualização é gerida por um servidor *REST*, este, define a estrutura de informação que é recebida pela plataforma e garante o fornecimento de dados à plataforma para futura visualização.

A plataforma de visualização desenvolvida é o resultado da combinação do tipo de informação recolhida e das várias componentes que, em conjunto, constituem as funcionalidades da aplicação. O tipo de informação definida na especificação dos dados de entrada, apresentados no capítulo 5, influencia todas as decisões realizadas sobre o tipo de visualização apresentada. Desta forma, a especificação dos dados de entrada da plataforma assume um papel muito importante em todo o desenvolvimento. No entanto, a construção da especificação foi um processo iterativo de alterações nos dados, estas alterações, foram essenciais para uma boa expressividade das componentes visuais desenvolvidas. A plataforma de visualização fornece aos utilizadores funcionalidades que lhe permitem uma interpretação visual e interativa da informação relativa aos seus sistemas. Uma das funcionalidades principais para a o desenvolvimento de todas as restantes é a capacidade de construir um grafo automaticamente. Esta funcionalidade ocupou uma percentagem importante do tempo, mas, refletindo sobre o tempo despendido no seu desenvolvimento, a construção correta da funcionalidade garantiu a possibilidade do desenvolvimento de uma série de outras funcionalidades de forma interativa.

O resultado final do trabalho prático pode ser definido com uma plataforma de visualização de sistema em microsserviços foi construída, para além da plataforma, e também uma aplicação de construção e fornecimento de dados semelhantes aos retirados da análise de *traces* permite testar e utilizar a plataforma com fins de exemplificação das funcionalidades.

Como trabalho futuro, podem ser adicionadas funcionalidades de visualização de *workflows*, como visualização de cada *workflow*, realçamento de *workflows* com tempos de resposta elevados ou com algum problema anotado previamente. Para além destas

funcionalidades, pode ser também aproveitado o trabalho desenvolvido para a visualização de informação com características de grafo, sendo necessário algum tipo de adaptação no trabalho atual.

Esta página foi intencionalmente deixada em branco.

Bibliografia

[1] Rajdeep Dua, A. Reddy Raja e Dharmesh Kakadia. "Virtualization vs containerization to support PaaS". Em: *Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014* (2014), pp. 610–614. DOI: 10.1109/IC2E.2014.41.

[2] *What is a Container? | Docker*. URL: https://www.docker.com/resources/what-container (acedido em 14/06/2019).

[3] Claus Pahl. "Containerization and the PaaS Cloud". Em: *IEEE Cloud Computing* 2.3 (2015), pp. 24–31. ISSN: 23256095. DOI: 10.1109/MCC.2015.51.

[4] *containerization_intro*. URL: https://searchitoperations.techtarget.com/definition/application-containerization-app-containerization (acedido em 02/06/2019).

[5] Antonio Celesti et al. "Exploring Container Virtualization in IoT Clouds". Em: *2016 IEEE International Conference on Smart Computing, SMARTCOMP 2016* Lxc (2016), pp. 1–6. DOI: 10.1109/SMARTCOMP.2016.7501691.

[6] Jeanna Neefe Matthews et al. "Virtualization and Containerization of Application Infrastructure : A Comparison". Em: *Lecture Notes in Computer Science* 1.4 (2014), pp. 1–1. ISSN: 03029743. DOI: 10.1109/CloudCom.2016.94. arXiv: 1606.04036. URL: http://awsdocs.s3.amazonaws.com/ElasticBeanstalk/latest/awseb-dg.pdf{\%}0Ahttp://subs.emis.de/LNI/Proceedings/Proceedings184/139.pdf{\%}0Ahttp://dl.acm.org/citation.cfm?id=1862393{\%}0Ahttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6809342{\%}0Ahttp://.

[7] *container osquestration*. URL: https://www.linux.com/NewS/8-OPEN-SOURCE-CONTAINER-ORCHESTRATION-TOOLS-KNOW (acedido em 08/06/2019).

[8] *What Is Container Orchestration?* URL: https://blog.newrelic.com/engineering/container-orchestration-explained/ (acedido em 08/06/2019).

[9] *Containers vs Virtual Machines: What's The Difference? – BMC Blogs*. URL: https://www.bmc.com/blogs/containers-vs-virtual-machines/ (acedido em 04/06/2019).

[10] *Home - Open Containers Initiative*. URL: https://www.opencontainers.org/ (acedido em 08/06/2019).

[11] Jonathan Baier. *Getting Started with Kubernetes*. Packt Publishing Ltd, 2015.

[12] *Kubernetes: An Overview - The New Stack*. URL: https://thenewstack.io/kubernetes-an-overview/ (acedido em 13/01/2020).

[13] Wilhelm Hasselbring e Guido Steinacker. "Microservice architectures for scalability, agility and reliability in e-commerce". Em: *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings* (2017), pp. 243–246. DOI: 10.1109/ICSAW.2017.11.

[14] *microservices - Explorar - Google Trends*. URL: https://trends.google.pt/trends/explore?q=microservices{\&}geo=PT (acedido em 01/06/2019).

[15] Xabier Larrucea et al. "Microservices". Em: *IEEE Software* 35.3 (2018), pp. 96–100. ISSN: 07407459. DOI: 10.1109/MS.2018.2141030.

[16] Samuel Newman. *Building Microservices @ Squarespace*. O'Reilly Media, 2017, pp. 2–6. ISBN: 9781491950357. URL: https://books.google.com.ec/books?hl=en{\&}lr={\&}id=jjl4BgAAQBAJ{\&}oi=fnd{\&}pg=PP1{\&}dq=Newman+Sam.+Building+microservices{\&}ots={_}AKRUo6XfM{\&}sig=8gdoi9-fLEYBzqpXNWNwhYoErvQ{\#}v=onepage{\&}q=NewmanSam.Buildingmicroservices{\&}f=false.

[17] *Service-oriented architecture (SOA) - Hands-On Microservices - Monitoring and Testing [Book]*. URL: https://www.oreilly.com/library/view/hands-on-microservices-/9781789133608/936cb8b7-b10c-4093-9001-17fddcca0b30.xhtml (acedido em 02/06/2019).

[18] Tasneem Salah et al. "The evolution of distributed systems towards microservices architecture". Em: *2016 11th International Conference for Internet Technology and Secured Transactions, ICITST 2016* (2017), pp. 318–325. DOI: 10.1109/ICITST.2016.7856721.

[19] *Microservices vs. Monolith Architecture - Pixel Point - Medium*. URL: https://medium.com/pixelpoint/microservices-vs-monolith-architecture-c7e43455994f (acedido em 13/01/2020).

[20] Nicola Dragoni et al. "Microservices : yesterday , today , and tomorrow To cite this version : HAL Id : hal-01631455". Em: (2017).

[21] Bob Familiar. *Microservices, IoT, and Azure*. Jan. de 2015. DOI: 10.1007/978-1-4842-1275-2.

[22] Production-ready Microservices e Production-ready Microservices. *Production-Ready Microservices*. 2017. ISBN: 9781491965979.

[23] Benjamin H Sigelman et al. "Dapper, a Large-Scale Distributed Systems Tracing Infrastructure". Em: *Google Technical Report dapper-2010-1* April (2010), p. 14. URL: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/36356.pdf.

[24] *THE CAUSAL AND TEMPORAL RELATIONSHIPS BETWEEN FIVE SPANS IN A DAPPER... | Download Scientific Diagram*. URL: https://www.researchgate.net/figure/The-causal-and-temporal-relationships-between-five-spans-in-a-Dapper-trace-tree{_}fig2{_}239595848 (acedido em 13/01/2020).

[25] *The OpenTracing project*. URL: https://opentracing.io/ (acedido em 15/06/2019).

[26] *OpenCensus*. URL: https://opencensus.io/ (acedido em 15/06/2019).

[27] *OpenZipkin · A distributed tracing system*. URL: https://zipkin.io/ (acedido em 15/06/2019).

[28] *Jaeger: open source, end-to-end distributed tracing*. URL: https://www.jaegertracing.io/ (acedido em 15/06/2019).

[29] *Spans-openTracing*. URL: https://opentracing.io/docs/overview/spans/ (acedido em 15/06/2019).

[30] Renzo Angles. "A comparison of current graph database models". Em: *Proceedings - 2012 IEEE 28th International Conference on Data Engineering Workshops, ICDEW 2012* (2012), pp. 171–177. DOI: 10.1109/ICDEW.2012.31.

[31] Z Y Gao et al. "m [20]". Em: 31.1 (2001), pp. 160–169.

[32] *An Introduction to Time Series Databases | Severalnines*. URL: https://severalnines.com/database-blog/introduction-time-series-databases (acedido em 19/01/2020).

[33] *The Anatomy of APM – 4 Foundational Elements to a Successful Strategy*. URL: https://www.brighttalk.com/webcast/534/46045/the-anatomy-of-apm-4-foundational-elements-to-a-successful-strategy (acedido em 23/06/2019).

[34] *appdynamics*. URL: https://www.pcmag.com/review/338449/appdynamics (acedido em 24/06/2019).

[35] Peter Eades. *Graph Drawing Graph Drawing Algorithmic and Declarative g Approaches*. Rel. téc. URL: http://www.it.usyd.edu.au/{~}peter/scase.pdf.

[36] *gelophi*. URL: https://gephi.org/features/ (acedido em 30/06/2019).

[37] *vis.js customization*. URL: https://visjs.org/network{_}examples.html (acedido em 30/06/2019).

[38] *Output Formats*. URL: https://www.graphviz.org/doc/info/output.html (acedido em 30/06/2019).

[39] *Graphviz - Graph Visualization Software*. URL: https://www.graphviz.org/ (acedido em 19/06/2019).

[40] *D3.js - Data-Driven Documents*. URL: https://d3js.org/ (acedido em 13/01/2020).

[41] *Gallery · d3/d3 Wiki · GitHub*. URL: https://github.com/d3/d3/wiki/Gallery (acedido em 19/01/2020).

[42] Linear Graph Algorithms. "SMJ000146.pdf". Em: 1.2 (1972), pp. 146–160.

[43] *The C4 model for visualising software architecture*. URL: https://c4model.com/ (acedido em 18/06/2019).

[44] *Multi-model highly available NoSQL database - ArangoDB*. URL: https://www.arangodb.com/ (acedido em 13/01/2020).

[45] *GitHub - arangodb/arangodb-java-driver: ArangoDB Java driver*. URL: https://github.com/arangodb/arangodb-java-driver (acedido em 13/01/2020).

[46] *InfluxDB: Purpose-Built Open Source Time Series Database | InfluxData*. URL: https://www.influxdata.com/ (acedido em 13/01/2020).

[47] *GitHub - influxdata/influxdb-java: Java client for InfluxDB*. URL: https://github.com/influxdata/influxdb-java (acedido em 13/01/2020).

[48] *Gallery · d3/d3 Wiki · GitHub*. URL: https://github.com/d3/d3/wiki/Gallery (acedido em 13/01/2020).

[49] *Tarjan's Algorithm to find Strongly Connected Components - GeeksforGeeks*. URL: https://www.geeksforgeeks.org/tarjan-algorithm-find-strongly-connected-components/ (acedido em 13/01/2020).

[50] *Tarjan's Algorithm to find Strongly Connected Components*. URL: https://iq.opengenus.org/tarjans-algorithm/ (acedido em 13/01/2020).