IPS Instituto Politécnico de Setúbal
Escola Superior de Ciências Empresariais

ELSA NUNES GODINHO

MONITORIZAÇÃO DE MICROSERVIÇOS NA EMPRESA HOVIONE

Fonte: Hovione

Relatório de caso de estudo do Mestrado em Gestão de Sistemas de Informação

Professor Doutor Vítor Barbosa

Outubro 2023

Dedicatória

Dedico este trabalho aos meus filhos Afonso e Vicente, ao Tiago e a toda a família que me apoiou neste desafio.

Aos meus colegas da Hovione, em especial ao Andrea Costigliola, Pedro Jorge e ao João Penas que tiveram sempre disponíveis para esclarecer todas as minhas dúvidas.

Também gostaria de dedicar este trabalho ao meu orientador Vítor Barbosa, que me encorajou e apoiou durante o percurso com a sua sabedoria que foi fundamental nesta jornada.

Por último, mas não menos importante, dedico este trabalho à minha colega e amiga Marta Matos, cujo apoio e camaradagem tornaram este percurso muito mais agradável.

Agradecimentos

Este trabalho intitulado "*A monitorização de microserviços na Hovione*" foi encarado desde o início como um desafio e apresentando-se como um tema que marca a atualidade e o futuro.

Um agradecimento especial à Hovione, que me permite todos os dias trabalhar com o propósito de salvar vidas.

Resumo

Os microserviços são um estilo de arquitetura de software que consiste num conjunto de serviços independentes que comunicam entre si através de APIs. Na sua implementação identificam-se várias vantagens como flexibilidade, manutenção e independência. No entanto, também se apresentam vários desafios, uma vez que se pode tornar numa arquitetura muito complexa, e pode dificultar a monitorização e manutenção de sistemas. A comunicação é outro dos grandes desafios apresentados, uma vez que é necessário garantir que as API's são compatíveis entre si.

A monitorização dos microserviços ajuda a identificar, analisar e resolver problemas nos microserviços de um modo mais célere, por forma a minimizar qualquer impacto nos seus utilizadores. A monitorização é feita através de métricas, tais como tempo de resposta, taxa de erros, uso de recursos e consumo de rede. A monitorização realiza-se com apoio de ferramentas e plataformas que ajudam a recolher e analisar os dados dos microserviços.

Com a implementação destas métricas, ferramentas e plataformas é possivel a deteção de problemas atempadamente que evitam problemas de desempenho, disponibilidade e confiabilidade, e evita, ou reduz, interrupções nos serviços. A monitorização também pode ajudar a identificar oportunidades de melhoria, otimizando o código, ajustando a configuração ou atualizando os recursos. Para além de que a monitorização pode ajudar a identificar e corrigir problemas que podem causar falhas, o que ajuda a reduzir o tempo de inatividade e assim aumentar a satisfação do cliente. Com uma monitorização adequada são fornecidos dados importantes para tomada de decisão, onde podem ser identificadas tendências, otimizar recursos e ou desenvolver novos produtos e serviços.

Este trabalho faz um enquadramento teórico dos microserviços, e de como é feita a sua monitorização, recorrendo à literatura e apresenta um estudo de caso que explora a monitorização dos microserviços na empresa portuguesa Hovione.

Palavras-chave: monitorização de microserviços; análise de microserviços; arquitetura de microserviços.

Abstract

Microservices are a software architecture style that consists of a collection of independent services that communicate with each other through APIs.

Several advantages have been identified in their implementation, such as flexibility, maintainability, and independence. However, several challenges are also presented, since it can become a very complex architecture, and can make it difficult to monitor and maintain systems. Communication is another major challenge, since it is necessary to ensure that the APIs are compatible with each other.

Microservice monitoring helps to identify, analyze, and resolve problems in microservices more quickly, in order to minimize any impact on their users. Monitoring is done through metrics, such as response time, error rate, resource usage, and network consumption. Monitoring is carried out with the support of tools and platforms that help collect and analyse microservice data.

With the implementation of these metrics, tools, and platforms, it is possible to detect problems early on that avoid performance, availability, and reliability problems, and avoid, or reduce, service interruptions. Monitoring can also help identify opportunities for improvement, by optimizing code, adjusting configuration, or updating resources. In addition, monitoring can help identify and correct problems that can cause failures, which helps to reduce downtime and thus increase customer satisfaction. With proper monitoring, important data is provided for decision-making, where trends can be identified, resources can be optimized, or new products and services can be developed.

This work provides a theoretical framework for microservices, and how they are monitored, using literature and presents a case study that explores the monitoring of microservices at the Portuguese company Hovione.

Keywords: *monitoring microservices; analysis microservices; microservices architecture.*

"Ao criar microserviços, estamos a criar sistemas mais resistentes e escaláveis, mas também a adicionar complexidade. É importante encontrar um equilíbrio entre a complexidade necessária para suportar a escala e a simplicidade necessária para manter a compreensão e a manutenção do sistema."

Sam Newman,
"Building Microservice"

Índice

Indice de Figuras

Lista de abreviaturas, siglas e acrónimos

API – *Application programming interface*

CI/CD - C*ontinuous integration and continuous delivery/continuous deployment*

DevOps - M*ethodology in the software development and IT industry*

ERP - Enterprise resource planning

EVS - *Environmental Services*

GMP - *Good Manufacturing Practices*

LIMS - *Laboratory Information Management System*

MSA - *Microservice architecture*

SOA – *Service-oriented architecture*

1. Introdução

O presente relatório constitui o resultado final da unidade Curricular de Dissertação ou Trabalho de Projeto ou Relatório de Estágio do Mestrado de Gestão de Sistemas de Informação, frequentado pela estudante na Escola Superior de Ciências Empresariais do Instituto Politécnico de Setúbal.

O tema base do trabalho de mestrado foi a monitorização de microserviços. A escolha do tema baseou-se na crescente adoção de microserviços na indústria de desenvolvimento de software e no interesse da estudante em aprofundar o conhecimento sobre a gestão e monitorização de microserviços.

O estudo da monitorização de microserviços apresentado neste relatório é complementado com um estudo de caso da empresa portuguesa Hovione, com o objetivo de partilhar como a mesma utiliza e monitoriza os microserviços e fazer uma análise crítica sobre as boas práticas face à literatura.

Ainda na introdução é apresentada a questão de partida e os objetivos propostos. No capítulo I, Enquadramento Teórico, serão apresentados os conceitos e o estado da arte relacionada com microserviços e a sua monitorização. No capítulo II é feita a apresentação das metodologias utilizadas neste trabalho. O terceiro capítulo é composto pela apresentação do estudo de caso e discussão dos resultados. O trabalho termina com a secção das conclusões e trabalho futuro.

A arquitetura de microserviços, adotada por muitas empresas, consiste na criação de vários serviços independentes (componentes de software), isto é, que não afetam o funcionamento de outros serviços, se este forem atualizados ou substituídos (desde que disponibilizem e/ou consumam os mesmos dados), permitindo maior agilidade no desenvolvimento, implementação e monitorização de novas funcionalidades.

Empresas como a Amazon, Netflix e o Linkedin usam a arquitetura de microserviços no desenvolvimento de grandes aplicações na *Cloud* como também em pequenos serviços que podem ser desenvolvidos, testados, escalados, operados e atualizados de modo independente (Villamizar et al., 2016).

Neste projeto procura-se analisar como estão a ser monitorizados e analisados os microserviços na empresa portuguesa Hovione.

Pretende-se responder às seguintes questões:

1. Qual é a relevância na utilização de microserviços dentro da Hovione?
2. Como quantificar a utilização de microserviços?
3. Como é feita a gestão e monitorização dos microserviços na Hovione?

A indústria farmacêutica tem uma grande responsabilidade com a saúde dos seres humanos, assegurando o bem-estar da população e contribuindo consideravelmente para a qualidade de vida da sociedade (Escária et al., 2016).

Os desafios com que se deparam as empresas farmacêuticas na implementação de novas tecnologias são mais complexos pela natureza do seu negócio, quando comparados com outras indústrias (Escária et al., 2016).

Compostos por vários módulos, os sistemas comunicam entre si, e cada um desses componentes pode ser implementado e escalado de forma independente, o que torna árduo identificar problemas que possam surgir.

A monitorização tem como principal objetivo ajudar a identificar problemas de desempenho, segurança e disponibilidade em todo o ambiente de microserviços, incluindo infraestrutura, rede e aplicações. É importante que a monitorização dos microserviços esteja implementada para identificar e corrigir problemas precocemente, para que melhore a capacidade de resposta do sistema, otimize o uso de recursos e ajude a garantir que o sistema esteja sempre disponível.

Como primeiro objetivo macro deste projeto, definiu-se a recolha e estudo de literatura relacionada com microserviços e a sua gestão. Compreender o seu funcionamento, a sua evolução histórica, vantagens e desvantagens, como podem ser monitorizados e as vantagens dessa mesma monitorização.

Como segundo objetivo macro deste projeto, definiu-se realizar um estudo de caso, na empresa Hovione, onde a estudante tinha conhecimento que já eram utilizados microserviços, de modo a fazer um levantamento, em contexto real e numa empresa de grande dimensão, de qual a utilização que é feita dos microserviços e como esses microserviços são monitorizados. Este estudo de caso permitirá fazer uma análise crítica ao cenário observado, por comparação com a informação obtida da literatura.

2. Enquadramento Teórico

O enquadramento teórico presente neste trabalho contempla uma secção de contextualização da utilização de microserviços e a evolução das arquiteturas dos sistemas de informação, na secção seguinte é explorado o conceito de microserviços e, por último, é abordada a monitorização dos microserviços.

2.1. Contextualização e Evolução

A transformação digital tem sido entendida como um processo que permite o aperfeiçoamento de uma atividade através do uso de ferramentas tecnológicas. Permitindo uma mudança estrutural nas organizações de modo que deixem de perspetivar a tecnologia como um recurso pontual para que a materializem no quotidiano. O uso da tecnologia contribui para melhorar o desempenho, aumentar o alcance, desenvolver estratégias e garantir melhores resultados (Samartinho & Barradas, 2020).

Essa transformação ocorre igualmente na indústria farmacêutica, num mercado em crescimento, com grandes oportunidades, e, como é evidente, com grandes desafios.

“Extrair microserviços de sistemas monolíticos é a próxima forma evolutiva do desafio original de decompor sistemas. Parnas (2017) esteve entre os primeiros a investigar sistematicamente os critérios que deveriam ser usados ao decompor sistemas em módulos” (Mazlami, Cito & Leitner, 2017). No entanto, nem todos os sistemas monolíticos devem migrar para uma arquitetura de microserviços, segundo Newman (2015), obter o domínio errado de um serviço pode ser caro, e componentes excessivamente acoplados podem ser piores do que ter um único sistema monolítico. Ou seja, esta conversão pode ser lenta e cara, pode acabar por ser um enorme desafio.

A arquitetura de microserviços tornou-se muito popular porque, no que respeitava às arquiteturas monolíticas tradicionais, não havia resposta às necessidades de escalabilidade e rápido ciclo de desenvolvimento, de acordo com Lewis & Fowler (2014a).

A interdependência entre módulos dificulta a separação completa. Ao migrar um sistema, é crucial ter informações atualizadas de forma a garantir que os sistemas permaneçam funcionais e não percam quaisquer funcionalidades.

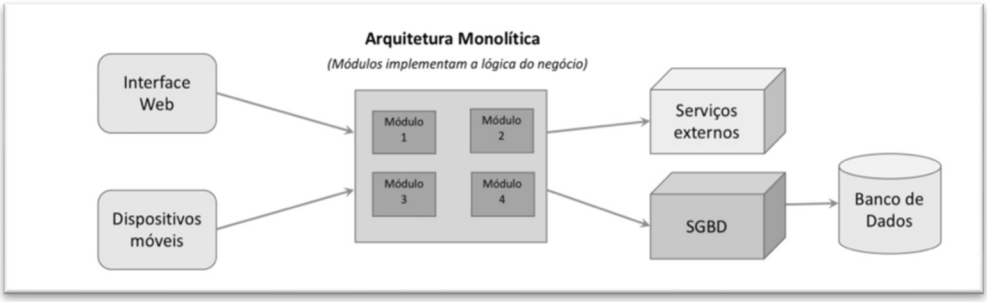


Figura 1- Exemplo de uma arquitetura monolítica

Fonte: Richardson (2015)

Os microserviços apresentam-se como alternativa à abordagem monolítica, cujo exemplo podemos observar na **Error! Reference source not found.**, que é apresentada como um desafio à medida que o sistema cresce de tamanho e de complexidade. A falta de modularidade pode levar a problemas de manutenção, dificuldades na implementação de alterações e problemas de escalabilidade (Richardson, 2015).

Como alternativa, a arquitetura de microserviços veio responder a esses desafios, no entanto, segundo Carrasco, Bladel & Demeyer (2018), "a migração de uma aplicação de um monólito para uma arquitetura de microserviços é uma tarefa complexa, demorada e propensa a erros, que requer o uso de ferramentas que facilitem e orientem o processo de decomposição."

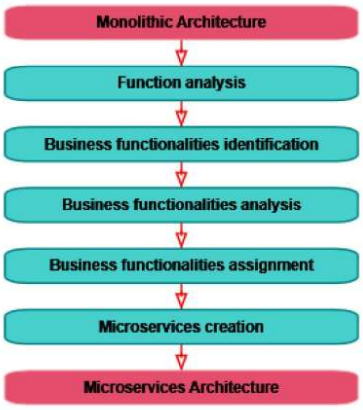


Figura 2 - Migração de Arquitetura Monolítica para Arquitetura de Microserviços

Fonte: De Lauretis (2019)

Como as funcionalidades de negócio são essenciais para estas migrações, uma estratégia de migração proposta por De Lauretis (2019) é dividida em cinco fases, ilustradas na **Error! Reference source not found.**. As cinco fases são:

1) **Análise funcional** - extrair e analisar todas as funções que compõem o monólito. As funções são assim analisadas para extrair dados como taxa de uso e tamanho (em termos de linhas de código). As funções também podem ser modificadas: se uma função for grande, ela pode ser dividida em várias subfunções; e se duas ou mais funções são muito semelhantes, elas podem ser aglutinadas numa única.
2) **Identificação das funcionalidades do negócio** - a identificação de todas as funcionalidades do negócio, num monólito não é uma tarefa simples e pode ser ainda necessário questionar o negócio sobre funcionalidades adicionais.
3) **Análise das funcionalidades do negócio** - as funcionalidades de negócios identificadas na etapa anterior são analisadas, a fim de extrair dados como taxa de uso e outras informações estatísticas.
4) **Atribuição das funcionalidades de negócios** - no caso de haver mais do que uma funcionalidade de negócio com o mesmo âmbito, poderão ser aglutinados num único microserviço, ou caso uma funcionalidade de negócio seja mais utilizada, pode fazer sentido separar em mais do que um microserviço.
5) **Criação de microserviços** - por fim, os microserviços são criados e as funcionalidades de negócios são desenvolvidas dentro dos microserviços. A arquitetura de microserviços está pronta para ser implementada e testada.

Não obstante, a adoção de uma arquitetura de microserviços, como o exemplo apresentado na **Error! Reference source not found.**, também apresenta os seus desafios, e um dos mais impactantes é a necessidade de monitorizar esses microserviços.

A monitorização e os testes ajudam a obter uma maior compreensão do sistema e a melhorar a qualidade em todo o processo de desenvolvimento. No entanto, os sistemas de microserviços são cada vez mais complexos e necessitam de monitorização e testes mais intensivos do que outro tipo de sistemas (por exemplo: SOA e monolíticos) (Rajput, 2018).

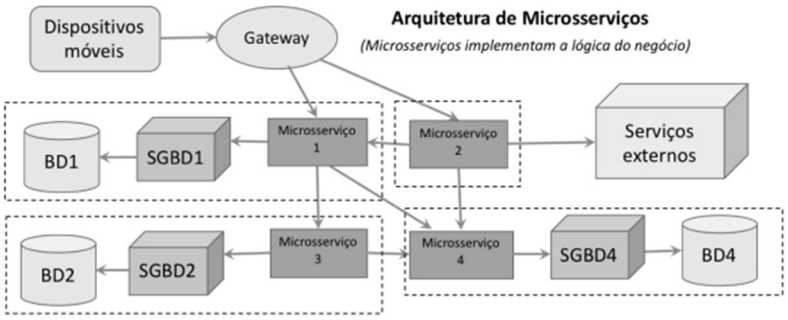


Figura 3 - Exemplo de arquitetura de microserviços
Fonte: (Villaça, Pimenta & Azevedo, 2018)

Segundo Newman (2015), “a transição de uma arquitetura monolítica para microserviços requer uma profunda análise e planeamento. Cada componente do monolítico deve ser avaliada para determinar se é adequado para se tornar um microserviço autônomo".

Na transição de uma arquitetura monolítica para uma de microserviços, um dos primeiros passos é avaliar a possibilidade de se separar em módulos ou em funcionalidades, isto é, microserviços.

Após o desenvolvimento dos microserviços, será necessário efetuar testes independentes e em conjunto com os sistemas já existentes.

Depois de implementado e integrado com o sistema, que normalmente envolve o uso de APIs[1], Gateways[2] e outras tecnologias de comunicação, o sistema monolítico é progressivamente decomposto (Dragoni et al., 2017).

Dragoni et al. (2017) sustentam que "a chave para uma transição bem-sucedida para a arquitetura de microserviços é garantir que cada serviço seja solto e acoplado apenas através de *APIs* bem definidas. Isso permite que cada serviço seja desenvolvido, implementado e escalado de forma independente."

Já os autores Lewis & Fowler (2014a) alertam que esta transição “não é uma panaceia e deve ser abordada com cautela. Embora possa trazer benefícios

[1] API – Application programming interface

[2] Gateway - a network node equipped for interfacing with another network that uses different communication protocols.

significativos em termos de escalabilidade e desenvolvimento independente, também apresenta desafios no que diz respeito à complexidade da gestão e monitorização de muitos serviços independentes."

Uma proposta de abordagem a esta problemática será a última secção deste capítulo, a monitorização de microserviços.

A monitorização de microserviços é um tópico muito discutido e é visto como essencial para garantir a "saúde" do sistema, identificar e resolver problemas rapidamente e obter *insights* sobre o comportamento do sistema em tempo real. Além disso, pode fornecer informações valiosas que podem ser usadas para otimizar o desempenho do sistema e melhorar a experiência do utilizador (Dragoni et al., 2017).

A natureza dinâmica dos sistemas de microserviços precisa de infraestrutura de monitorização para diagnosticar e relatar erros, falhas e problemas de desempenho (Waseem et al, 2021).

Consequentemente as métricas, práticas e ferramentas são usadas para detetar, corrigir e agir em problemas nos microserviços.

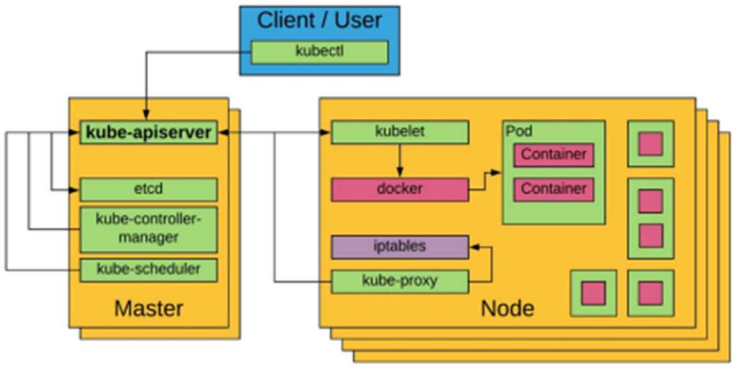


Figura 4 - Arquitetura Kubernetes
Fonte: https://dzone.com/storage/attachments/14131598-dzone-kubernetes-bundle.pdf

Tecnologias de Containers[3], como o Docker, por exemplo, e plataformas de orquestração, como o Kubernetes, exemplificado na Figura 4, facilitam a criação, a

[3] Containers - is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another.

implementação e a gestão de microserviços em larga escala, e foram essenciais na sua popularização (Khan,207).

"Um container é um código encapsulado que contém apenas o sistema operativo, bibliotecas e dependências necessárias para executar a aplicação. Ao criar um container, é criado um executável que pode ser executado em qualquer máquina. Os containers são semelhantes às máquinas virtuais (VMs), porém exigem menos recursos e são mais rápidos. Antes dos containers, transferir código para outros computadores era um problema muito mais complicado. Os containers eliminaram esse problema ("escrito uma vez e executado em qualquer lugar"). A plataforma mais conhecida e utilizada para a containerização é chamada Docker" (Golis, Dakić & Vranić, 2022).

Newman (2015) enfatiza que a "containerização ajuda a criar sistemas mais previsíveis, isolados e seguros, e é uma parte essencial da modernização das aplicações num modelo de microserviços."

A orquestração automatiza a implementação, a gestão, a escala e a rede dos *containers*, e é fundamental para as empresas que precisem de implementar e gerir centenas de milhares de *hosts* e *containers*, e é também base para vários sistemas e *frameworks*, como *serverless*[4]. Os orquestradores de *containers* geralmente oferecem os seguintes recursos principais: controlo do limite de recursos, programação de tarefas, balanceamento de carga, verificação de integridade, tolerância a falhas e escalonamento automático (Casalicchio, 2018).

A Portainer é uma das mais versáteis plataformas de gestão de containers que simplifica a adoção segura de containers com uma velocidade notável. Permite a gestão em segurança de aplicações em containers no Docker e Kubernetes (Portainer, s.d.).

A platafoma permite gerir containers Docker de forma simples e intuitiva através de uma interface web, é universal o que significa que não está vinculado a uma tecnologia ou fornecedor, permite trabalhar em várias tecnologias e faz transição para o Kubernetes se necessário. Integra facilmente com as ferramentas

4 Serverless Framework - is a cloud computing application development and execution model that enables developers to build and run application code without provisioning or managing servers or backend infrastructure.

de CI/CD, controlos de acesso, por exemplo, o que facilita a incorporação do Portainer no ambiente de desenvolvimento já existente.

O Portainer oferece ainda recursos de segurança, como integração de registo privado e modelos de provisionamento de *cluster*, para ajudar a proteger os *containers.*

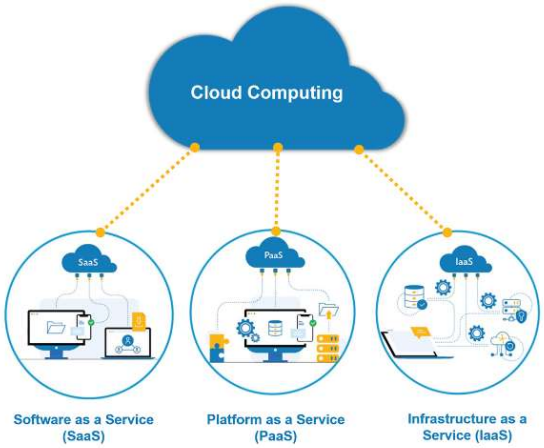


Figura 5 - Cloud Computing
Fonte: https://development.asia/explainer/why-cloud-computing-key-enabler-digital-government

A *Cloud Computing* [5] representada na Figura 5 também desempenha um papel crucial, pois oferece a elasticidade e a escalabilidade necessárias para aproveitar ao máximo os microserviços. "A *Cloud Computing* e os microserviços são as duas faces da mesma moeda. Ambos facilitam a construção de aplicações escaláveis num mundo que exige cada vez mais flexibilidade e resiliência. A *cloud* fornece a infraestrutura, enquanto os microserviços fornecem a arquitetura." (Villamizar et al., 2016.)

Newman (2015) faz referência de que a computação em nuvem oferece a capacidade de implementar microserviços em larga escala, aproveitando recursos como escalabilidade automática e equilíbrio de carga. Isso permite que as organizações respondam rapidamente às mudanças do pedido.

[5] Cloud Computing - Cloud computing is the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user. Large clouds often have functions distributed over multiple locations, each of which is a data center.

Consequentemente, os microserviços beneficiam enormemente da *Cloud*. Eles podem ser escalados de forma independente, aproveitando a elasticidade da nuvem, e podem ser desenvolvidos e implementados rapidamente usando serviços e ferramentas baseadas na *Cloud"* (Richardson, 2015).

Richardson (2018) descreve que sistemas de transmissão de dados, como o RabbitMQ ou o Kafka oferecem uma maneira robusta e escalável para os serviços comunicarem entre si, especialmente em sistemas de microserviços onde a comunicação síncrona pode ser impraticável ou indesejável.

A monitorização e observabilidade, são essenciais para manter a qualidade e a confiabilidade em sistemas baseados em microserviços. Eles fornecem as ferramentas para entender, diagnosticar e corrigir problemas rapidamente, antes que afetem os utilizadores finais (Saman, 2017; Dragoni et al, 2017).

Stevens (2020) apresenta ainda a observabilidade como crucial numa arquitetura de microserviços devido à sua natureza distribuída. É necessário ter visibilidade sobre o estado de todos os serviços e como eles interagem entre si, para entender o estado geral do sistema.

2.2. Microserviços

Os microserviços são o resultado da evolução contínua das práticas e tecnologias de arquitetura de software. “Uma abordagem à arquitetura de aplicações que desmembra uma aplicação numa coleção de serviços menores e independentes, que são desenvolvidos, implementados e escalados de forma independente" (Newman, 2015).

Cada microserviço é um componente de software que pode ser desenvolvido, implementado, operado e dimensionado de forma independente de outros microserviços no sistema. Cada microserviço é responsável por uma funcionalidade específica e tem uma *interface* bem definida que outros microserviços usam para interagir com ele (Richardson, 2018).

É importante salientar que o termo "microserviços" foi designado entre 2011-2012 durante um workshop de arquitetos de software para descrever este estilo arquitetónico que estavam a observar (Lewis & Fowler, 2014a). Dali em diante, a abordagem de microserviços tem vindo a evoluir.

Ainda sem uma definição concreta, mas como contraste às arquiteturas de software monolíticas, foram apresentados como aplicações pequenas, com apenas uma responsabilidade (funcional, não funcional ou requisito multifuncional) que pudesse ser implementado, dimensionado e testado de forma independente. Estes suportam agilidade, porque podem ser desenvolvidos e modificados de forma independente, utilizando mais do que uma linguagem de programação.

Uma arquitetura de microserviços foi apresentada como "*distributed application where all its modules are microservices*" (Baškarada et al., 2020).

A arquitetura orientada a serviços (SOA)[6] foi uma das primeiras tentativas de dividir aplicações monolíticas em componentes menores, mais fáceis de gerir e reutilizáveis. No entanto, a SOA muitas vezes tornava-se complexa e pesada devido ao uso de Enterprise Service Buses (ESB)[7]. O ESB fornece a todos os serviços de infraestrutura de comunicação e mediação necessários para permitir a interação entre serviços que são implementados em múltiplas e, muitas vezes, heterogéneas, plataformas de TI (Chappell, 2004).

Assim, os microserviços podem ser vistos como uma iteração mais leve e mais pragmática da SOA.

As características comuns da arquitetura de microserviços incluem a separação em componentes, via serviços. A aplicação está dividida em serviços independentes que funcionam em diferentes processos, não partilham recursos e comunicam entre si via mensagens leves (Baškarada et al., 2020).

"Na arquitetura de microserviços, os serviços são pequenos, independentes e podem ser alterados sem ter impacto no sistema como um todo. Isso permite uma evolução independente e rápida das funcionalidades" (Newman, 2015).

Cada microserviço que compõe uma arquitetura de microserviços, é simples, no entanto a arquitetura global pode atingir elevados níveis de complexidade. Uma vez que, cada microserviço pode ser implementado com tecnologia diferente e por

[6] SOA - arquitetura de software que permite a criação de aplicações alinhadas com a estratégia de negócios, criando um conjunto de serviços reutilizáveis que correspondem às necessidades do negócio. Estes serviços comunicam entre si, passando dados de um serviço para outro, ou coordenando alguma atividade entre serviços (Erl, 2007).

[7] ESB – It represents a software architecture for distributed computing, and is a special variant of the more general client-server model, wherein any application may behave as server or client.

diferentes equipas, é incontornável utilizar a metodologia *DevOps* [8]na gestão do projeto como um todo (Chen, 2018).

Os microserviços são construídos para determinadas funcionalidades do negócio e são desenvolvidos e implementados de forma independente por um conjunto de mecanismos automáticos. A necessidade de gestão centralizada destes serviços é bastante reduzida. Cada serviço pode ser desenvolvido na sua própria linguagem de programação e utilizar a sua própria tecnologia de armazenamento.

Nesta arquitetura, cada serviço pode ser alterado independentemente dos outros. Isso significa que se for necessário adicionar novas funcionalidades ou fazer alterações às existentes, pode fazer-se num serviço específico sem afetar o restante sistema. Permite assim que as equipas de desenvolvimento se adaptem rapidamente às mudanças, consoante as necessidades do negócio.

"Os microserviços permitem escalar componentes individuais de uma aplicação de acordo com as suas necessidades específicas, proporcionando uma melhor utilização dos recursos e uma maior eficiência operacional." (Richardson, 2018)

Tal como o negócio muda, também a tecnologia avança. Uma arquitetura de microserviços permite experimentar novas tecnologias num serviço específico sem afetar o restante da aplicação/sistema, o que pode tornar mais fácil adotar novas tecnologias e adaptar-se às mudanças nas tendências tecnológicas, uma vez que os "microserviços oferecem a flexibilidade de experimentar novas tecnologias e paradigmas de programação num contexto limitado, mitigando o risco associado à adoção de novas tecnologias." (Lewis & Fowler, 2014a)

Outra característica dos microserviços é a escalabilidade, isto é, a capacidade de uma arquitetura se adaptar e/ou crescer de acordo com a exigência do serviço sem prejudicar o desempenho. Numa "arquitetura de microserviços, os serviços

[8] Metodologia *DevOps – Metodologia de organização do trabalho que integra o desenvolvimento com a exploração do sistema. Utiliza mecanismos automáticos para o desenvolvimento, colocação em produção e monitorização do trabalho.*

podem escalar de forma independente para atender aos pedidos de carga de trabalho." (Richardson, 2018)

A escalabilidade pode verificar-se através da adição de mais recursos, como memória e armazenamento, mas tendo em conta as limitações do hardware denominada por escalabilidade vertical ou na adição de mais máquinas ou instâncias do serviço à infraestrutura, distribuindo a carga de trabalho entre várias instâncias do mesmo serviço, conhecida como escalabilidade horizontal. (Dragoni et al., 2018)

A flexibilidade tecnológica, é também um dos benefícios dos microserviços, que apresenta várias vantagens para o desenvolvimento e para a manutenção de aplicações, permitindo que as equipas de desenvolvimento sejam autónomas e que possam selecionar as melhores tecnologias e/ou ferramentas para o serviço que está a criar, sendo evidentemente mais eficiente e autónomo. Esta flexibilidade facilita ainda os testes de novas tecnologias e abordagens, sem comprometer o funcionamento do sistema. Adaptando ainda as melhores práticas de cada linguagem ou tecnologia escolhida, melhorando assim a qualidade de código e a sua manutenção (Dragoni et al., 2018).

Com efeito, é importante destacar os desafios resultantes do uso de várias tecnologias e linguagens, que pode aumentar a complexidade do sistema, e obrigar ao uso de protocolos de comunicação para a integração e comunicação entre si.

Manter um conjunto diversificado de tecnologias e frameworks exige que haja uma governança apertada que governe, padronize e mantenha o ambiente de desenvolvimento coeso. Com a adoção de diferentes tecnologias, podem gerar-se inconsistências no design e na implementação dos serviços, e consequentemente afetar a experiência do utilizador e a qualidade do sistema. Portanto, é importante que as empresas adotem uma governança de sistemas de forma a garantir a coesão e a qualidade do sistema, permitindo a adaptação a longo prazo da arquitetura de microserviços (Indrasiri & Siriwardena, 2018).

Na arquitetura de microserviços, a resiliência refere-se à capacidade de um sistema ser construído de forma a recuperar rapidamente de falhas e ainda assim continuar a funcionar de forma eficiente. O isolamento de falhas, permite que mesmo que um serviço falhe ou enfrente problemas de performance, os outros

serviços continuem a funcionar normalmente. A arquitetura de microserviços permite a implementação de estratégias de tolerância a falhas, como circuit breakers e timeouts, que ajudam a evitar que falhas num serviço se propaguem para outros serviços. A recuperação dos microserviços é mais rápida, através do reinício ou da substituição de um serviço com falha, minimizando assim os impactos (Richardson, 2018).

Com diferentes equipas, pode trabalhar-se em diferentes serviços, ou seja, as aplicações podem ser desenvolvidas em paralelo, consequentemente acelerando o seu desenvolvimento. Significa ainda que as atualizações, as correções de bugs ou novas funcionalidades possam ser lançadas para um serviço específico, o que efetivamente irá acelerar o tempo de lançamento de novos recursos e correções. Contudo, a velocidade do desenvolvimento pode ser afetada devido a vários fatores, tais como a complexidade da comunicação entre os serviços, necessidade de coordenação entre equipas, com a autoridade no desenvolvimento de software, com os padrões de projeto e com a complexidade de gerir os dados distribuídos. É assim crucial gerir adequadamente esses desafios para garantir um desenvolvimento eficiente e eficaz. A gestão de várias bases de código, pipelines de implementação e infraestruturas separadas, traduz-se em complexidade operacional.

Como os serviços precisam de comunicar entre si, essas " interações entre os serviços podem ser complexas... [e] essas interações ocorrem através de redes que são inerentemente não confiáveis." (Richardson, 2018)

A consistência de dados entre serviços é também um desafio em sistemas de microserviços. "Numa arquitetura de microserviços, cada serviço tem o seu próprio banco de dados para garantir a separação livre de ligações. Isso pode resultar em desafios de consistência de dados." (Newman, 2015)

Para aproveitar ao máximo a arquitetura de microserviços, uma organização deve exigir uma mudança cultural significativa dentro da empresa, pois as equipas precisam trabalhar de forma mais colaborativa e coordenada para manter a aplicação como um todo. "A adoção de microserviços não deve ser feita sem um alto grau de maturidade em técnicas de entrega contínua [...] uma organização deve

estar pronta para investir em práticas de automação de entrega e teste, ou corre o risco de aumentar drasticamente o tempo de entrega." (Lewis & Fowler, 2014a).

Com maior eficiência nos sistemas de software, reduzem-se os custos de infraestrutura como também o risco de interrupções de serviço relacionadas com a capacidade.

A capacidade de gestão independente contribui para melhorar a eficiência e também reduz a necessidade de paragens programadas.

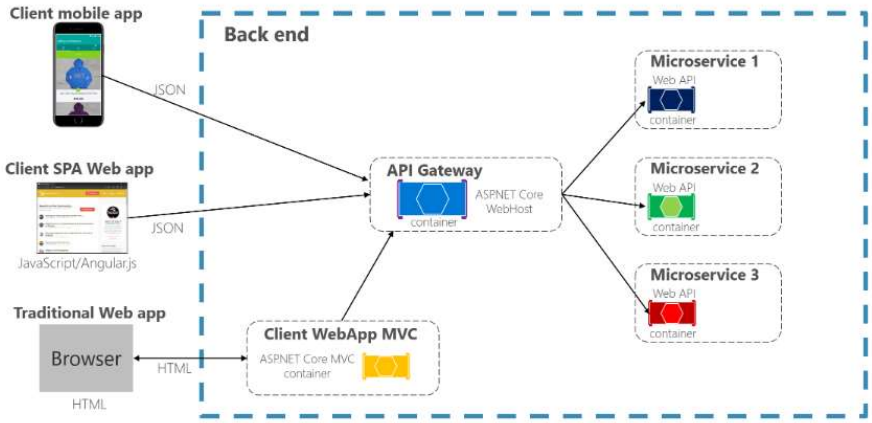


Figura 6 - API Gateway
Fonte: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern

Os padrões de design são estratégias e soluções práticas para lidar com desafios mais recorrentes nesta arquitetura.

A *API Gateway*, representada na Figura 6, é o padrão que fornece um único ponto de entrada para um conjunto de microserviços, responsável por rotear as solicitações para os microserviços apropriados. Como também lidar com questões como gestão de carga, *caching*[9], autenticação e autorização.

A *API Gateway* atua como porta de entrada em todas as solicitações de cliente para os microserviços no *back-end,* encapsula a complexidade interna do sistema de microserviços e fornece uma API unificada para os clientes.

[9] *Caching - the process of storing data in a cache, which is a temporary storage area that facilitates faster access to data with the goal of improving application and system performance.*

Richardson (2018), descreve o papel da *API Gateway* como "responsável por solicitações de roteamento para os serviços apropriados. E pode também fornecer funcionalidades adicionais, como autenticação, *insights, e cache* de resposta. A implementação de tais funcionalidades na camada do *gateway* reduz a complexidade dos serviços, que não precisam lidar com essas preocupações."

A *API Gateway* desempenha um papel crucial ao abstrair a complexidade da arquitetura de microserviços, centraliza algumas funcionalidades comuns, como autenticação e controle de taxa de resposta, que seriam difíceis de implementar de forma consistente em todos os serviços individuais.

No entanto, a *gateway* pode tornar-se num ponto único de falha se não for adequadamente escalado e protegido para além de que pode ser difícil de gerir se a lógica *gateway* se tornar muito complexa.

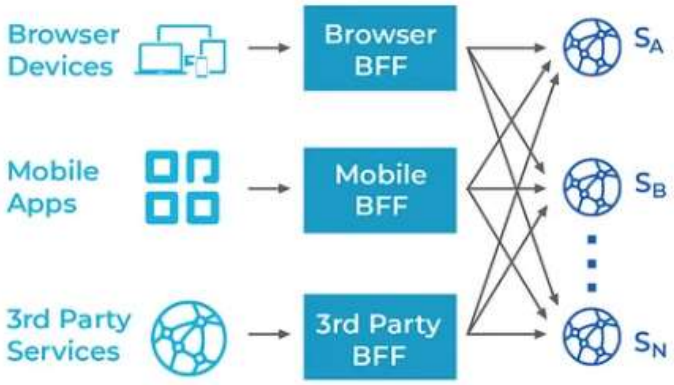


Figura 7 - Padrão BFF
Fonte: BFF

Por outro lado, Newman (2015) assegura que com o Padrão *Backends For Frontends (*BFF), são criados *backends* específicos para diferentes interfaces de utilizador. Portanto, se existirem três *interfaces* de utilizador diferentes, existem três BFFs correspondentes (conforme representado na figura 7). Ter um *backend* personalizado para cada *interface* de utilizador pode simplificar muito a *interface* de utilizador em si. Além disso, a lógica que seria duplicada em várias *interfaces* de utilizador é agora mantida num único local.

O BFF é assim uma resposta à necessidade de ter uma *interface* de utilizador otimizada para cada cliente, tendo em consideração as suas características e

necessidades específicas. Cada BFF é dedicado a um tipo específico de cliente, e pode fornecer uma API otimizada para esse cliente.

Uma das desvantagens deste padrão é a duplicação de código, já que vários BFFs podem precisar de lógica semelhante. No entanto, o benefício de ter uma experiência de utilizador otimizada e a redução da complexidade do cliente, geralmente superam essa desvantagem.

Para lidar com a falha de serviço, o padrão de *Circuit Breaker*, apresentado na Figura 8, procura interromper as solicitações por um período de tempo, permitindo que o serviço recupere.

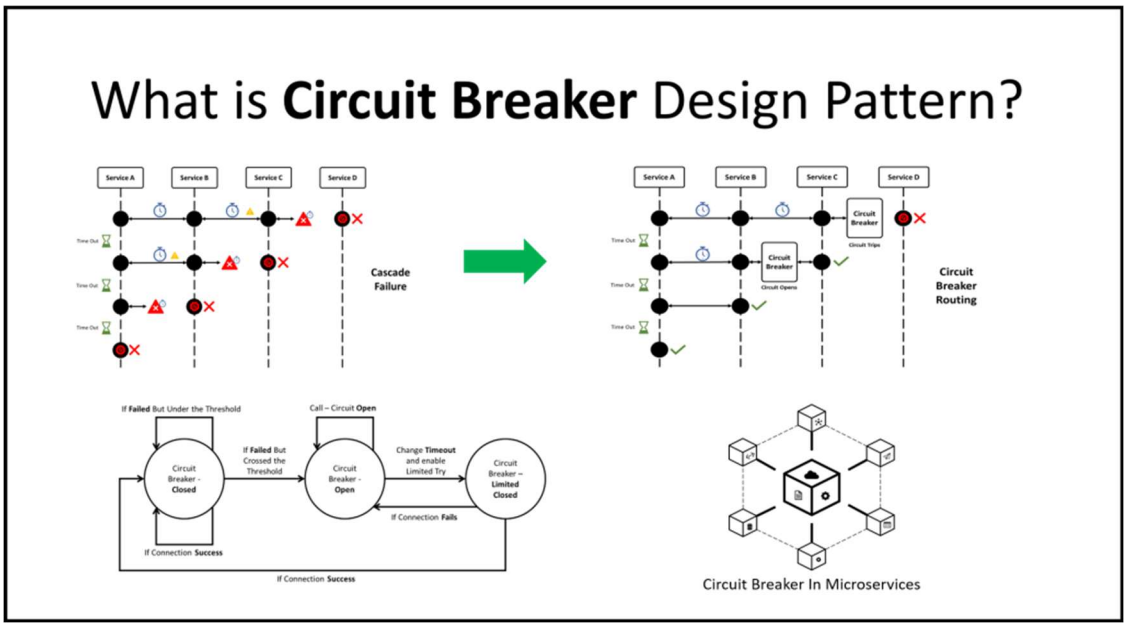


Figura 8 – Circuit Breaker design
Fonte: https://medium.com/javarevisited/what-is-circuit-breaker-in-microservices-a94f95f5e5ae

Nygard (2007), reforça que "O padrão *Circuit Breaker* é uma forma de prevenir automaticamente chamadas para um sistema quando ele está inativo".

Para garantir resiliência e tolerância de falha num sistema, este padrão é um dos mais essenciais. Assim que um microserviço falhe, o *Circuit Breaker* interrompe as solicitações por um período de tempo, permitindo que o serviço recupere. Após um tempo predeterminado, o *Circuit Breaker* permite quantidades limitadas de solicitações passem antes de assumir que o serviço está saudável novamente.

Encarado como uma ferramenta essencial para evitar falhas num serviço, e que se propaguem para outros serviços e resultem numa falha em cascata, que pode

levar a que todo o sistema fique inoperante. Contribui desta forma, para a construção de sistemas mais resilientes e tolerantes a falhas.

O grande desafio deste padrão é a definição de quando abrir e fechar o circuito e como lidar com a recuperação do serviço.

"Os microserviços normalmente são implementados num ambiente elástico e dinâmico onde o número de instâncias de um serviço e a sua localização podem mudar rapidamente. Isso apresenta um desafio: como é que os clientes de um serviço descobrem a localização das suas instâncias?" Richardson (2018) apresenta a solução que passa pela necessidade de ter o *Service Discovery* na arquitetura de microserviços.

O *Service discovery* é um componente essencial para qualquer aplicação de microserviços porque a localização de um microserviço não é atribuída na fase do design. Além disso pode ser implementado num ambiente *cloud*, o que significa que os serviços podem ser realocados e replicados em sistemas de produção. (Al-Debagy, 2018)

Existem diferentes maneiras de implementar o *Service Discovery*, exemplificado na figura 9, consoante o nível de automação e do ambiente em que os microserviços estão a ser executados. Algumas das opções comuns incluem o uso de um registo de serviço, como o *Eureka* da Netflix, ou o uso de um sistema de orquestração de *containers*, como o *Kubernetes*, que tem *Service Discovery* integrado. (Montesi, 2016)

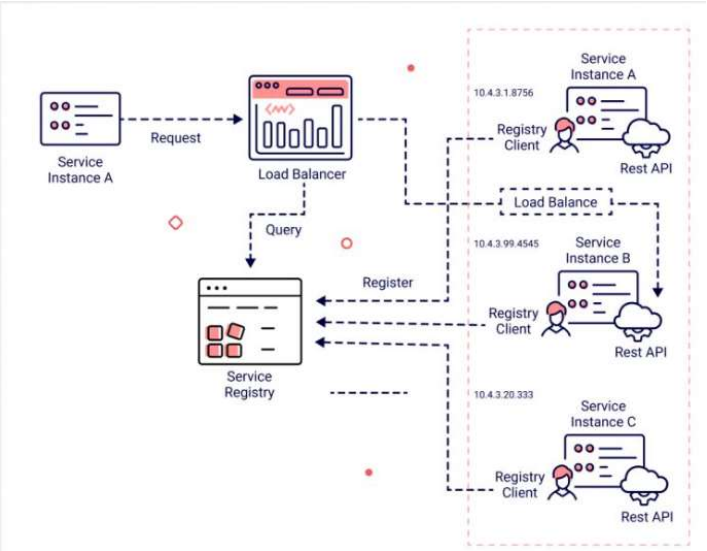


Figura 8 – Service Discovery
Fonte: Middleware

Garantir que as localizações dos serviços estejam sempre atualizadas e lidar ainda com a possibilidade de que o próprio *Service Discovery* possa falhar, são alguns dos desafios que podem ser superados com práticas adequadas de *design* e operação.

Um padrão essencial, na arquitetura de microserviços, de acordo com Richardson (2018), é o *log* centralizado e monitorização na arquitetura "para entender o comportamento do sistema e diagnosticar problemas, e é necessário para correlacionar *logs* de diferentes serviços (...) essencial para uma arquitetura de microserviços eficaz."

Centralização, agregação de logs e monitorização fornece à equipa uma visão geral completa do status do sistema, permitindo que eles ajam proativamente em comportamentos suspeitos e com falhas (Mazzara, 2021).

Para garantir robustez dos microserviços é crucial que cada serviço tenha o seu repositório de dados privado. Partilhar bases de dados entre serviços viola o princípio do desacoplamento e aumenta as hipóteses de colisões de esquema e falhas em cascata (Richardson, 2018).

Por esta razão, o padrão *Database per Service* é fundamental para garantir a independência entre os serviços, permitindo que cada um evolua de forma independente. Existem várias maneiras diferentes de manter os dados privados de um serviço como o *Private-tables-per-service*, cada serviço possuiu um conjunto de tabelas que só podem ser acedidas através daquele serviço, o *Schema-per-service*, onde cada serviço tem o esquema de base de dados privada e o *Database-server-per-service* que tem o seu próprio servidor de base de dados (Messina 2016).

"If you are working in an organization that places lots of restrictions on how developers can do their work, then microservices may not be for you."

Sam Newman, Building Microservice

2.3. Monitorização de Microserviços

Em arquiteturas de microserviços, a monitorização é fundamental. Ajuda a entender o comportamento do sistema, encontrar problemas e melhorar a confiabilidade e a performance do sistema. (Newman, 2015)

A monitorização de uma arquitetura de microserviços envolve a recolha de dados para cálculo de métricas que fornecem uma visão do desempenho e "saúde" de um serviço. Exemplos de métricas são: a taxa de requisições/pedidos, o tempo de resposta, a utilização de recursos e taxas de erros. (Richardson, 2018)

Os *logs* registam detalhes do que está a acontecer num serviço, são uma fonte fundamental de dados para a monitorização e diagnóstico de anomalias (Newman, 2015). A monitorização distribuída que "é uma técnica essencial para a depuração e otimização de microserviços." (Basiri et al., 2016)

A monitorização de microserviços pode ser complexa devido à natureza distribuída e dinâmica dos microserviços. Pode ser difícil correlacionar eventos numa rede de serviços e entender o comportamento do sistema como um todo. (Villamizar et al., 2018)

A monitorização tem por objetivo auxiliar na deteção e diagnostico de problemas rapidamente. Se um (micro)serviço falhar ou começar a apresentar comportamento anormal, a monitorização permite identificar o problema e agilizar a resolução do mesmo. (Newman, 2015)

A monitorização deve fornecer *insights* sobre o desempenho dos serviços, permitindo identificar otimizações para melhorar a eficiência e a velocidade do serviço ou de todo o sistema (Richardson, 2018). Permite, assim, uma melhor compreensão do comportamento do sistema como um todo. É particularmente importante num sistema distribuído, onde o comportamento global do sistema é o resultado de interações entre serviços individuais (Thones, 2015). Sendo contínua, a monitorização, é essencial para garantir a confiabilidade e a disponibilidade de um sistema de microserviços. Ela permite que as equipas identifiquem e corrijam problemas antes que eles afetem os utilizadores (Basiri, et al., 2016).

São apresentados vários desafios na monitorização de microserviços, devido à sua natureza distribuída e dinâmica. A correlação de dados de serviços distintos,

pode ser um desafio, pois cada serviço pode ter a sua própria base de código, infraestrutura e padrões de *log* (Newman, 2015).

Também a visibilidade pode ser um desafio na monitorização de microserviços. Como os microserviços são distribuídos e operam em isolamento, pode dificultar uma visão completa e consistente do sistema como um todo (Richardson, 2018).

A deteção de anomalias num ambiente de microserviços é também um desafio. As anomalias podem ser subtis e difíceis de detetar sem uma compreensão profunda do comportamento normal do sistema, isto é, em que medida o comportamento está fora do normal (Basiri, et al., 2016).

Por sua vez, a monitorização distribuída, considerado um desafio crítico, pois implica monitorizar uma requisição/pedido à medida que ela se move através de diferentes serviços, o que pode ser complexo e difícil de implementar (Villamizar et al., 2018).

Dependendo das necessidades específicas da arquitetura, umas mais passíveis de implementar que outras, são utilizadas na monitorização dos microserviços, ferramentas, técnicas e soluções (Waseem et al., 2021).

Existem muitas ferramentas de monitorização disponíveis, tanto de código aberto como comerciais, que podem ser usadas para recolher, armazenar e analisar dados de monitorização (Newman, 2015). São exemplos, enquanto soluções open source: *Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana);* e soluções comerciais: *Datadog e New Relic*.

O Prometheus (Turnbull, 2018; Brazil, 2018), representado na Figura 9, é um sistema de monitorização e alerta *open-source*, muito popular no ecossistema de microserviços, que recolhe métricas (medidas) de microserviços enviando solicitações HTTP e armazena os resultados numa base de dados de séries temporais.

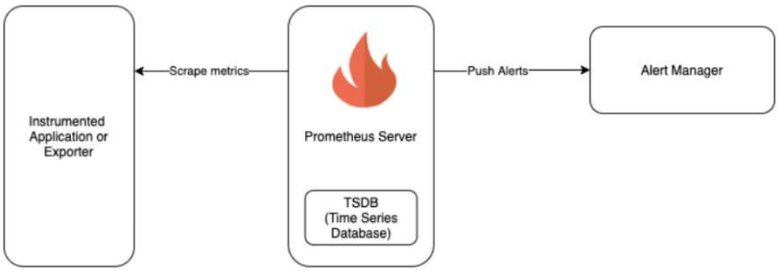


Figura 9 - Arquitetura Prometheus
Fonte: https://prometheus.io/

No que diz respeito ao *Grafana* (Chakraborty & Kundan, 2021), exemplificada na Figura 10, é uma plataforma *open-source* de visualização e análise de dados que permite criar *dashboards* personalizados para exibir as métricas recolhidas pelo *Prometheus* ou outras fontes de dados.

Figura 10 - Dashboard Grafana
Fonte: Grafana

A *Stack ELK (Elasticsearch, Logstash, Kibana*) é outra solução muito popular para recolher, armazenar e analisar *logs* de aplicações e dados de métricas (Lahmadi & Beck, 2015). *Elasticsearch* é o mecanismo de captura e análise, *Logstash* é o que recolhe e processa os dados, e *Kibana* é a *interface* de visualização.

A monitorização dos *logs* é uma técnica comum para monitorizar microserviços. Ela envolve a recolha e análise de *logs* de cada serviço. Ferramentas como *Logstash* e *Fluentd* podem ser usadas para recolher *logs,*

enquanto *Elasticsearch* e *Kibana* podem ser usadas para armazenar e analisar esses registos (Thones, 2015).

Existe ainda uma ferramenta de monitorização distribuída *open-source*, desenvolvida pela *Uber*, que permite analisar as transações entre microserviços e identificar problemas de desempenho. Essa ferramenta é a *Jaeger,* representada na figura 11, que também ajuda a entender o fluxo de chamadas entre microserviços e a identificar problemas de desempenho.

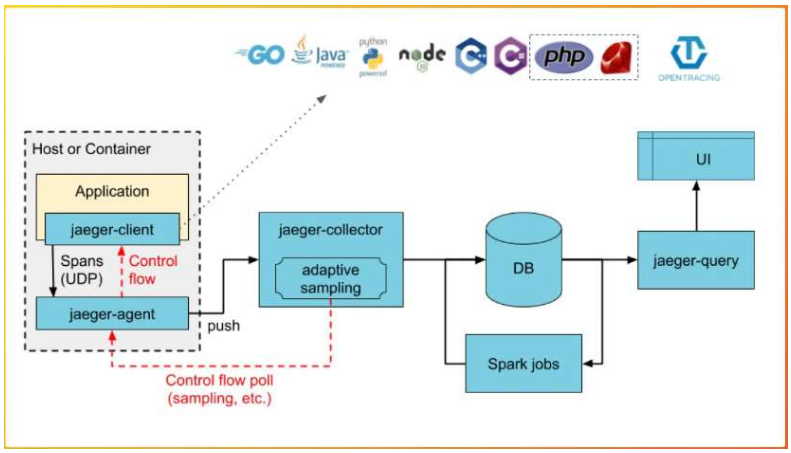


Figura 11 - Jaeer Arquitetura
Fonte: Jaegertracing

A *Datadog* é uma plataforma comercial de monitorização e análise de desempenho baseada em *SaaS (Software as a Service)*[10] que oferece integração com várias tecnologias e serviços, facilitando a recolha e visualização de métricas de microserviços (https://www.datadoghq.com/). A *New Relic* também funciona como *SaaS* é permite a recolha e análise de métricas de desempenho, rastreamento distribuído e monitorização de *logs* (https://newrelic.com/).

A monitorização de *logs* é parte essencial da observabilidade em sistemas de microserviços. Sem uma compreensão clara do que está a acontecer em cada serviço, a capacidade de operar e manter o sistema de forma eficiente é significativamente prejudicada (Sridharan, 2018).

[10] *SaaS - Software-as-a-Service (SaaS), is a cloud based software delivery model that allows end users to access software applications over the internet. With a SaaS model, the software is hosted on remote servers, maintained and updated by the service provider, and made available to customers via web browsers, mobile apps and APIs. salesforce*

Sendo que a monitorização com base em *logs* é uma parte fundamental na monitorização de microserviços, devem efetivamente ser seguidas as boas práticas referidas na literatura:

- Centralizar os *logs* de todos os serviços. Isso permite uma visão unificada e facilita a correlação de eventos entre serviços (Newman, 2015);
- A padronização dos *logs*, que inclui o "formato dos *logs* e os níveis de *log*" (Richardson, 2018);
- Ter "*log*s estruturados que podem ser facilmente analisados e pesquisados. Isso é especialmente útil num ambiente de microserviços, onde pode haver uma grande quantidade de *logs*" (Villamizar et al., 2018);
- Da mesma forma que a monitorização em tempo real dos *logs* pode ajudar a detetar problemas rapidamente e a responder a eles de maneira oportuna. (Thones, 2015)

Segundo Basiri et al. (2016), a retenção dos *logs* deve ser considerada. É preciso equilibrar a necessidade de manter os *logs* para diagnóstico e conformidade com as considerações de custo e privacidade.

A monitorização de desempenho e gestão de aplicações existe também na *Dynatrace* (Forsberg, 2019) que oferece monitorização distribuída, análise de métricas e monitorização de logs para microserviços. Existe ainda o *Istio,* um service *mesh*[11] que fornece recursos de monitorização, permite o rastreamento e observabilidade para microserviços num ambiente de containers, como *Kubernetes* (Calcote & Butcher, 2019). Outra solução *mesh* é a *Consul* que também pode ser usada para monitorizar a saúde dos microserviços e fornecer informações sobre o estado do sistema (Kysow, 2022).

Além de monitorizar os microserviços, também é importante monitorizar o sistema operativo e a infraestrutura. Ferramentas como *cAdvisor e Node Exporter* podem ser usadas a recolher dados de todo o sistema (Villamizar et al., 2018).

11 Mesh - is a mechanism for managing communications between the various individual services that make up modern applications in a microservice-based system.

A capacidade de monitorizar efetivamente o comportamento do sistema é um dos principais desafios na arquitetura de microserviços (Richardson,2018).

A monitorização de *logs* num ambiente de microserviços é essencial para manter a visibilidade e a eficiência operacional. Os *logs* são a fonte da verdade em sistemas distribuídos. Eles fornecem um registo imutável de eventos que podem ser usados para entender o comportamento do sistema, identificar problemas e otimizar o desempenho (Kleppmann, 2017).

3. Objetivos e Metodologias

Considerando os objetivos do estudo, que são detalhados na primeira secção deste capítulo, destaca-se a importância da seleção adequada do instrumento de pesquisa para atingir esses objetivos, que "é crucial para garantir a validade e a confiabilidade dos resultados do estudo", de acordo com Babbie (2015).

Uma das partes mais cruciais de um estudo/trabalho de investigação diz respeito às abordagens e técnicas metodológicas adotadas ao longo do processo. Dessa forma, este capítulo abrange dois pontos: primeiramente focado nos objetivos e, posteriormente nos métodos e técnicas aplicadas.

3.1. Objetivos

Conforme referido na introdução, o objetivo geral deste trabalho consiste no estudo da monitorização dos microserviços numa empresa da indústria farmacêutica, a Hovione.

Tem por base o conceito de monitorização de microserviços que "envolve observar e controlar os microserviços de uma aplicação para avaliar a sua condição, desempenho e funcionamento. Isso inclui recolher métricas e *logs* dos microserviços, rastrear as interações entre eles, e analisar esses dados para detetar e diagnosticar problemas. A monitorização é crucial para garantir a disponibilidade e o desempenho de uma aplicação de microserviços, e para facilitar a identificação e resolução de problemas." (Richardson,2018).

Procurou-se alcançar dois grandes objetivos:

O primeiro objetivo, pretende construir conhecimento sobre o que são microserviços, a razão da sua utilização (vantagens e desafios) e as implicações nos sistemas.

No segundo objetivo, pretende entender como está a ser analisada a *performance* dos microserviços na empresa analisada, isto é que ferramentas, técnicas e métricas são usadas para monitorização de microserviços.

De acordo com Newman (2015), a monitorização de microserviços não é apenas uma boa prática, é uma necessidade absoluta. Dada a natureza distribuída e dinâmica das arquiteturas de microserviços, sem monitorização adequada, as

empresas correm o risco de enfrentar problemas de desempenho, indisponibilidades e falhas que podem ser difíceis de detetar e resolver. A monitorização permite que as empresas mantenham um olhar atento sobre seus microserviços, identifiquem problemas rapidamente e tomem medidas corretivas antes que esses problemas afetem os utilizadores finais ou o negócio como um todo.

O mesmo autor defende que na arquitetura de microserviços, a medição do desempenho é um componente crítico que permite a deteção de problemas em tempo real e a otimização contínua da eficiência do sistema. A medição do tempo de resposta, taxa de erros e uso de recursos para cada serviço permite uma visão holística do desempenho.

A análise de *performance* envolve a recolha de métricas de desempenho de cada serviço e a análise dessas métricas para identificar obstáculos e oportunidades de otimização. Sem uma análise de desempenho eficaz, as organizações correm o risco de operar sistemas ineficientes que não conseguem atender aos requisitos dos usuários. (Richardson,2018)

3.2. Métodos e Técnicas

No presente trabalho académico, sendo que o âmbito foi definido, deu-se o início com uma revisão da literatura e análise documental. Estas atividades formam um papel importante no estudo, pois ajudam a responder às questões colocadas sobre o assunto em análise.

Segundo Browen (2009), a análise documental é um método de recolha de dados sistemático e objetivo que descreve os fenômenos de interesse. Esta análise permite ao investigador fazer inferências sobre os fenômenos a partir de dados textuais.

No que concerne à revisão da literatura, Oliveira (2007) diz que corresponde a uma modalidade de estudo e de análise de documentos de domínio científico, sendo a sua principal finalidade o contato direto com documentos relativos ao tema em estudo.

No âmbito da investigação de natureza qualitativa, a técnica de *Focus Group* é uma das metodologias de investigação mais utilizadas nas ciências sociais (Bloor, 2001).

Os focus group são um método de investigação dirigido à recolha de dados; localiza a interação na discussão do grupo como a fonte dos dados; e, reconhece o papel ativo do investigador na dinamização da discussão do grupo para efeitos de recolha dos dados. (Krueger & Casey, 2009)

Com o objetivo de fundamentar a análise do tema, podemos afirmar que este trabalho utiliza metodologias qualitativas.

A escolha por um estudo de caso, nomeadamente em investigação qualitativa fundamenta-se nas potencialidades de exploração, descrição e compreensão de acontecimentos ou fenómenos complexos, configurados por múltiplos factores (Yin, 2009; Merriam, 1998; Stake, 2003), que um estudo de caso pode oferecer.

Assim, Yin (2009) define estudo de caso “com base nas características do fenómeno em estudo e com base num conjunto de características associadas ao processo de recolha de dados e às estratégias de análise dos mesmos”.

O estudo focalizado e em profundidade que os estudos de caso exigem (Yin, 2009; Merriam, 1998; Stake, 2003) e, no caso das metodologias qualitativas, um plano de investigação aberto e flexível, cuja seleção das dimensões a trabalhar e interpretar é emergente e, em consequência, progressiva, levam a considerar estes estudos como requerendo algum tempo. De acordo com Stake (2003), requerem um tempo relativamente alargado: “no local, particularmente no contacto com actividades e operações relativas ao caso, refletindo, revendo significados”.

Na explicitação de Yin (2009), o estudo de caso responde às questões de investigação “porquê” e “como”, o que facilita a compreensão dos fenómenos sociais, pela análise particularizada do contexto situacional. Para Stake (1978), é “um estudo de um sistema delimitado, que dá ênfase à unidade e globalidade desse sistema, mas concentra a atenção nos aspetos que são relevantes para o problema de investigação, num dado tempo”.

A metodologia baseou-se então em “analisar como é a Hovione faz a monitorização dos seus microserviços”, através do uso da técnica de Focus Group aplicada a diferentes grupos de Stakeholders dentro da Hovione, da area do Digital

(Anexo I) sendo os participantes convidados a discutir como a monitorização está a ser desenvolvida e analisada e em "entender como está a ser analisada a performance dos microserviços nos sistemas de informação da Hovione".

Os colegas da administração de sistemas e os *developers* da equipa do Digital (Anexo I) disponibilizaram-se para partilhar o seu conhecimento sobre os microserviços e monitorização de microserviços, sendo que são os especialistas com maior *know-how* sobre o tema na empresa.

Com uma abordagem pouco estruturada pretendeu-se entender e explorar o conceito, com perguntas abertas para a exploração do tema e obter respostas mais elaboradas. Foi assim possível identificar alguns microserviços internos, entender as atividades em curso, como também as planeadas para o futuro, no que diz respeito às práticas de monitorização dos microserviços.

Outra informação, por confidencialidade, não foi possível apresentar no estudo de caso.

4. Apresentação do estudo de caso

4.1. A empresa Hovione

O estudo de caso deste trabalho académico foi feito numa empresa portuguesa que tem como propósito salvar vidas, a Hovione, empresa muito conhecida no apoio ao combate à COVID-19. Neste estudo de caso, foram levantadas e analisadas as técnicas e métodos de monitorização dos microserviços que a empresa utiliza no seu dia-a-dia para dar resposta ao seu negócio.

A empresa portuguesa fundada em 1959, é especializada na área da ciência da saúde e tem como principal negócio a investigação, desenvolvimento e produção de produtos farmacêuticos de última geração, ou seja, substâncias ativas (API) que compõem os medicamentos.

São vários os artigos que estão disponibilizados, no site da empresa, que nos apresentam a sua evolução como empresa, e onde partilha todas as suas conquistas ao longo dos seus 64 anos.

A Hovione tem um forte foco na inovação e qualidade. Com uma equipa de químicos, engenheiros e farmacêuticos apaixonados pela ciência e inovação, a empresa é uma das maiores investidoras em Investigação e Desenvolvimento (I&D) na indústria farmacêutica portuguesa. Além disso, é o maior empregador privado de doutorados em Portugal, de acordo com o Jornal de Negócios (2021).

A empresa conduz atividades de investigação e desenvolvimento em Portugal, empregando mais de duzentos técnicos e cientistas. Possui cinco fábricas localizadas em Portugal, Estados Unidos da América, Irlanda, Macau e China, e possui escritórios em Hong Kong, Japão, Suíça e Índia. A Hovione emprega mais de 3000 colaboradores, incluindo 300 investigadores, e possui uma capacidade de produção de mais de 1300m³. Os seus produtos são exportados para os mercados mais exigentes do mundo.

A Hovione tem duas áreas principais de atuação: a produção de produtos genéricos, incluindo corticosteroides, antibióticos e agentes de contraste, e a produção sob contrato para outras empresas (outsourcing). Os seus principais clientes são empresas farmacêuticas, e as exigências e necessidades são altamente complexas.

À medida que a empresa cresce, é necessário que os seus sistemas de informação acompanhem esse crescimento. Portanto, a Hovione tem investido em tecnologia e sistemas de informação ao longo dos anos, conforme a figura 12, com o objetivo de impulsionar a inovação e apoiar as operações. Isso é essencial para atender às procuras do mercado e garantir um alto nível de eficiência e qualidade nos seus processos.

Figura 12 - Sistemas de Informação Hovione
Fonte: Hovione

Com o objetivo de implementar uma solução global, capaz de responder aos requisitos de todas as suas localizações e fazendo uso de um sistema único, a Hovione implementou o ERP (SAP) em 1999, que assegura toda a atividade da empresa, em todos os sites, com diferentes módulos, constituindo uma solução verdadeiramente global e com gestão centralizada num único sistema.

Em 2001, implementou o Sistema de Gestão de Informação Laboratorial (LIMS), um sistema de software usado para gerir e monitorizar as operações de laboratório, dados e amostras.

Em 2006, a implementação de um projeto de *Customer Relationship Management* (CRM), surgiu com a necessidade de aumentar a equipa de vendas, de concentrar e partilhar informação, de conseguir maior coordenação e de obter rastreabilidade das oportunidades de negócio. Toda a documentação, tal como as COPs (*Corporate Operating Procedures*), SOPs (*Standard Operating Procedures*), especificações, planos de amostragem, métodos analíticos, desvios, controlo de alterações e formação são controlados eletronicamente através dos sistemas de informação em vigor, os quais cumprem com as normas em vigor para sistemas eletrónicos.

Todos os sistemas da empresa impõem a atuação em conformidade com os procedimentos estabelecidos de acordo com as *Good Manufacturing Practices* (GMP's). Em conjunto controlam toda a gestão fabril, asseguram a conformidade com todos os procedimentos de qualidade estabelecidos, assim como a própria qualidade dos processos e dos produtos. (Hovione, s.d.)

Com a implementação de vários sistemas de automação nas suas operações de produção, a empresa garante a eficiência e precisão do processo de fabricação de produtos farmacêuticos e químicos. Esses sistemas automatizados também permitem um maior controle dos processos e dados, reduzindo os erros humanos e aumentando a qualidade dos produtos.

A empresa é ainda conhecida pela sua expertise em tecnologia '*spray-drying*'[12] e engenharia de partículas, e recentemente investiu em tecnologia de ponta para produção contínua de comprimidos, colocando-a na vanguarda da indústria farmacêutica.

4.2. A utilização e monitorização de microserviços na empresa

A presente secção tem como objetivo detalhar aspetos relevantes a nível da arquitetura de microserviços que é utilizada na infraestrutura da Hovione, e a sua monitorização que garante a qualidade e estabilidade do sistema.

A empresa utiliza a metodologia *DevOps* na criação e implementação de microserviços, que são aplicações/ funcionalidades do negócio.

Os microserviços desenvolvidos seguem as *best practices* definidas pela comunidade para garantir que todo o código fonte seja versionado, garantindo integridade em todas as novas versões. Assim garante a imutabilidade nos *deployments* que são efetuados e garante que o que foi testado foi entregue, seguindo as normas Good Manufacturing Practices (GMP).

[12] *Spray drying is a continuous process that can be used for multiple applications. Over the last 19 years, the technology has been used for the production of solid dispersions and is the fastest growing platform to overcome the solubility issues related to oral drugs, it is also relevant to inhalable particles, specialty excipients and for the isolation of thermally labile products. Hovione.*

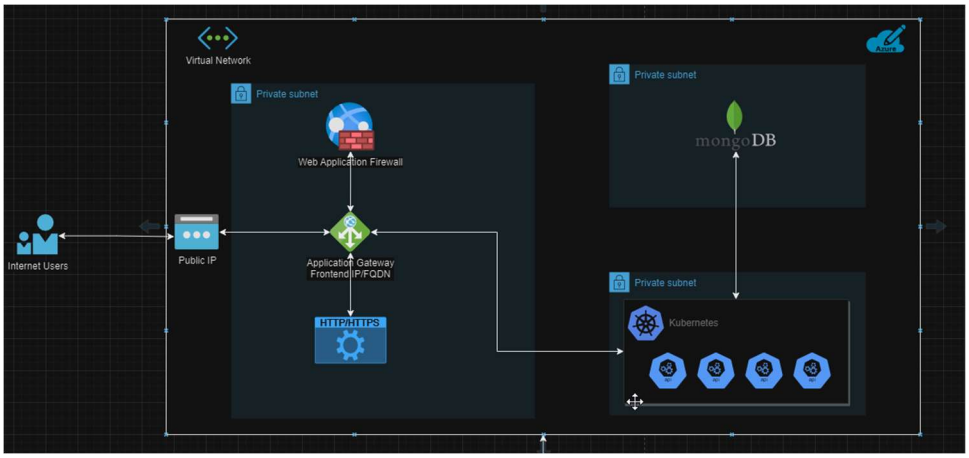


Figura 13 - Exemplo de uma arquitetura na Cloud Hovione

Fonte: Hovione

A Hovione possui aproximadamente 25 microserviços ativos, cada um com uma finalidade específica para atender clientes ou utilizadores. Para encaminhar os pedidos, a empresa utiliza a API Gateway, conforme ilustrado na figura 13.

Sempre que existe acesso a recursos (base dados ou servidor do ERP, por exemplo), na empresa, é prática criar uma *subnet*, por tipo de recursos (por exemplo, por base de dados).

Criar *subnets* ajuda a organizar e controlar o tráfego de forma mais eficiente. Além disso, permite reforçar a segurança do ambiente, separando recursos sensíveis, como bases de dados, de recursos de acesso público, como servidores *web.* E, de acordo com as políticas de segurança, garantem a segurança ao nível de infraestrutura.

Na Hovione, os microserviços invocam/acedem a vários servidores, tais como servidor de Laboratory Information Management System (LIMS), base de dados, Enterprise resource planning (ERP) e Xnet.

A infraestrutura da empresa é baseada na tecnologia Docker, que permite a criação e gestão de *containers*. Para gerir essa infraestrutura, representada na figura 16, a empresa utiliza o software *Portainer Enterprise,* que oferece uma interface gráfica para gerir *containers*, além de permitir a gestão por linha de comandos.

O acesso aos *Portainers* de gestão é feito via *web*. Esta solução dá a possibilidade de visualizar os *logs* de cada *container*, em tempo real e com histórico

relativamente grande (3 a 6 meses), serve também para ter uma monitorização dos *containers*, que em caso de falha é enviada uma notificação via email.

A infraestrutura, representada na figura 14 está dividida em dois ambientes: sand/desenvolvimento e validação/produção. A gestão de cada ambiente é completamente segregada, de forma a evitar erros e garantir a estabilidade do sistema.

No ambiente de *sand*/desenvolvimento, os *developers* podem testar e validar novas funcionalidades. Esta segregação de ambientes ajuda a garantir a qualidade e a estabilidade do sistema, permitindo que os *developers* testem novas funcionalidades sem afetar o ambiente de produção. Além disso, a segregação garante a segurança do sistema, evitando que erros ou vulnerabilidades num ambiente afetem o outro.

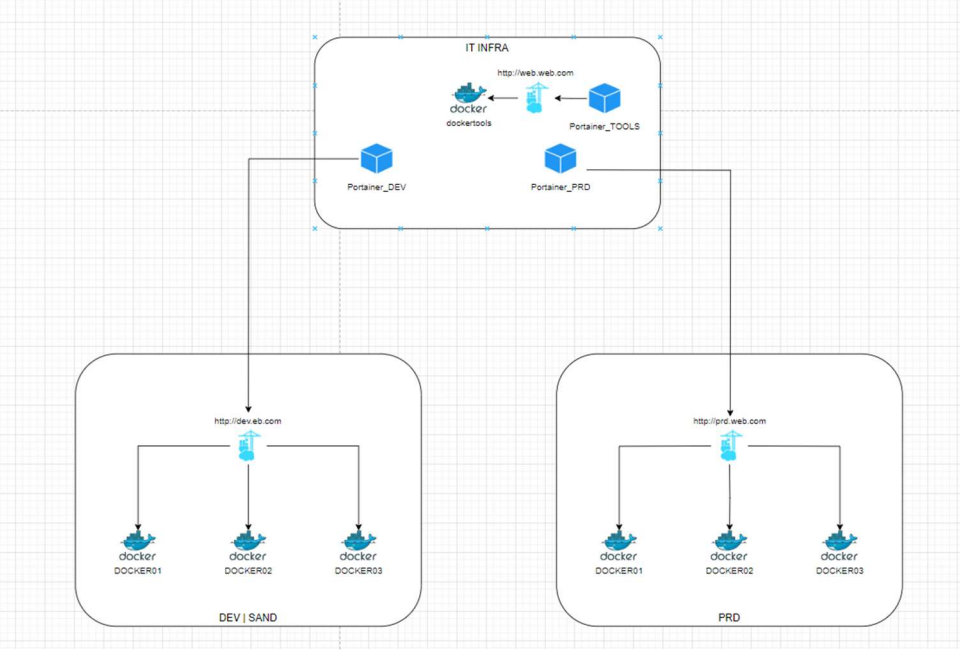


Figura 14 - Arquitetura de Infraestrutura da Hovione

Fonte: Hovione

A utilização de *Portainer* e *Docker* é altamente flexível e escalável, permitindo a execução eficiente e isolada de aplicações. A combinação do *Docker* com o *Portainer* oferece uma maneira robusta de orquestrar e gerir *containers* em qualquer ambiente, a partir de um único servidor local até a um *cluster* de servidores em nuvem.

Em termos de Kubernetes, a empresa tem a infraestrutura em Azure, utilizando o serviço Azure Kubernetes (AKS), esta solução dá a flexibilidade e escalabilidade, quer vertical, significa que é possível aumentar a capacidade de processamento dos *containers* (*nodes*[13] *ou pods*[14]) adicionando mais recursos, quer horizontal, adicionando mais *containers,* praticamente imediata e completamente agnóstica à aplicação.

A solução AKS permite uma gestão automatizada das tarefas operacionais tais como as atualizações e *deploy* de novas versões aplicacionais sem disrupção de disponibilização do serviço/aplicação.

Em Azure, o AKS é um serviço como todos os demais sendo a sua integração fácil e rápida *(straighforward),* retirando essa complexidade das equipas de desenvolvimento.

A empresa utiliza duas ferramentas para controle de versões: Argo CD e Jenkins.

O Argo CD é uma ferramenta de *Continuous Delivery* (CD) que permite a implementação automatizada de aplicações em *Kubernetes*, utiliza uma abordagem *GitOps*[15], que consiste em gerir a infraestrutura como código, armazenando as configurações num repositório *Git*[16] e aplicando-as automaticamente ao *cluster Kubernetes*. O Argo CD é fácil de instalar em qualquer *cluster Kubernetes* e oferece recursos avançados de monitorização e gestão de aplicações. O Jenkins, por sua vez, é uma ferramenta de integração contínua (CI)

13 Nodes - Um Node é uma máquina de trabalho em Kubernetes, que pode ser uma máquina virtual ou física. Cada Node é gerenciado pelo plano de controle do Kubernetes e contém os serviços necessários para executar os Pods. O Node é responsável por executar os containers e fornecer recursos como CPU, memória e armazenamento. O Kubernetes controla automaticamente a distribuição dos Pods pelos Nodes, levando em consideração os recursos disponíveis em cada Node.

14 Pod: Um Pod é a menor unidade implementável em Kubernetes. Ele é composto por um ou mais containers, que compartilham recursos de armazenamento e rede. Cada Pod é agendado num Node e permanece lá até ser encerrado ou excluído. Os Pods são projetados para suportar vários processos cooperativos (como containers) que formam uma unidade coesa de serviço. Os containers num Pod são automaticamente co-localizados e co-agendados na mesma máquina física ou virtual no cluster.

15 GitOps é um padrão de implementação contínua para aplicativos nativos em nuvem, que se concentram numa experiência centrada em developers para operação de infraestrutura.

16 Git - é um sistema de controlo de versões distribuído, utilizado para gerir arquivos de diversos tipos e também uma ferramenta para colaboração de equipas para geriro código fonte de uma aplicação.

que permite a automação do processo de desenvolvimento, teste e implementação de aplicações, é muito utilizado para desenvolver e implementar aplicações em ambientes *On-Premises*. O Jenkins é muito flexível e personalizável, permitindo a integração com uma ampla variedade de ferramentas e plataformas.

Em resumo, a empresa utiliza o Argo CD e o Jenkins para controle de versões. O Argo CD é utilizado para *deploy* na *cloud*, enquanto o Jenkins é utilizado para *deployment* de microserviços no *Docker On-Premises.* Cada ferramenta é voltada para um ambiente específico (*Kubernetes ou Docker On-Premises*) e oferece recursos avançados para automação do processo de implementação.

O Argo CD é integrado com o AKS por forma a automatizar ainda mais os fluxos de trabalho de entrega contínua e implementações. Ambas as ferramentas têm funcionalidades complementares e podem trabalhar em conjunto para proporcionar uma experiência de implementação mais robusta e automatizada. A empresa utiliza esta ferramenta porque permite que tarefas manuais sejam adicionadas ao *Workflow* (WF), gere automaticamente de acordo com as ações criadas no WF, como por exemplo, executar testes de carga, ligação, como também utilizar ferramentas de segurança.

No controlo de versões, a empresa utiliza também o repositório Azure, que oferece integrações com o *GitHub*[17], para acelerar o desenvolvimento de software, automatizar fluxos de trabalho de código para a nuvem e melhorar a segurança da aplicação.

No que se refere à gestão de WF (*commit, push, pull, merge,* etc[18]), representado na figura 15, é utilizada a plataforma *DevOps Azure.*

17 GitHub é uma plataforma e serviço baseado em nuvem para desenvolvimento de software e controle de versão usando o Git.

18 Commit: É a operação de salvar as alterações feitas num ou mais arquivos num repositório Git. O commit é uma operação local e não afeta o repositório remoto.

Push: É a operação de enviar as alterações feitas num repositório local para um repositório remoto. O push atualiza o repositório remoto com as alterações feitas no repositório local.

Pull: É a operação passar as alterações feitas num repositório remoto para um repositório local. O pull atualiza o repositório local com as alterações feitas no repositório remoto.

Merge: É a operação de combinar as alterações feitas em diferentes branches num único branch. O merge é necessário quando duas ou mais pessoas trabalham no mesmo arquivo e fazem alterações conflitantes.

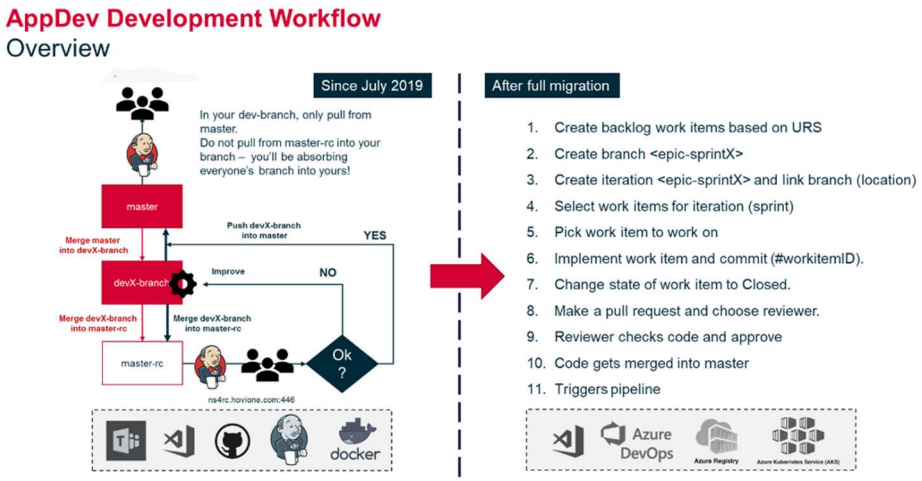


Figura 15 - AppDev Development Workflow

Fonte: Hovione

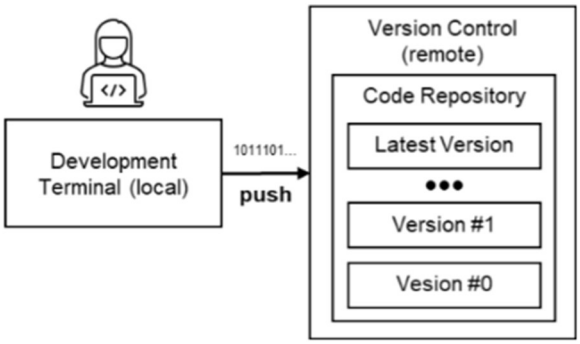


Figura 16 - Controlo de versões

Fonte: Hovione

O s*oftware code version contro*l que podemos ver na figura 16, é um componente crítico no desenvolvimento, que permite às equipas rastrear as alterações ao código fonte ao longo do tempo.

Ao usar um sistema de controlo de versões, como o *Entreprise* GitHub ou *Microsoft Azure DevOps Repos,* ou aproveitando os recursos de controle de versão da plataforma de desenvolvimento (por exemplo ferramentas de desenvolvimento SAP), os *developers* podem manter um histórico de alterações de código, colaborar no código com outros developers e rastrear/resolver problemas que surjam durante o desenvolvimento.

Adicionalmente, o controle de versões pode ainda melhorar a qualidade do código, o que permite aos *developers* rever as suas alterações (por exemplo

usando solicitações *pull*) e detetar erros antes que sejam incluídos no código principal.

Figura 17 - Exemplo Kuma de monitorização de um microserviço

Fonte: Hovione

Recentemente a Hovione começou a testar a obtenção de métricas através de um serviço *mesh*, o *Kuma (open source)*, de acordo com o exemplo da figura 17, que fornece recursos avançados de observabilidade, segurança e gestão de tráfego para aplicações, que são informações recolhidas e expostas para monitorizar e analisar o desempenho, a confiabilidade e a saúde da infraestrutura e das aplicações.

Algumas das métricas que estão em análise são:

- Latência: Que mede o tempo que leva para uma solicitação ser processada de um serviço para outro. Ajudam a identificar por exemplo, a *performance* geral da comunicação entre os serviços.
- Taxa de Erros: Mede a taxa de solicitações que falharam ao serem processadas, indicando problemas ou falhas nos serviços.
- Taxa de Sucesso: Mede a taxa de solicitações que foram processadas com sucesso, indicando a eficácia do serviço em responder às solicitações.
- Taxa de Requisições: Mede a taxa de solicitações enviadas para um determinado serviço, ajudando a identificar padrões de tráfego e carga no serviço.
- Taxa de Tráfego: Mede a quantidade de tráfego em bytes ou pacotes que estão a ser enviados ou recebidos pelos serviços.
- Uso de Recursos: Mede a utilização de recursos, como CPU, memória, entre outros, pelos serviços.

- Contagem de Requisições: Conta o número total de solicitações processadas por um serviço.

A empresa ainda não tem uma ferramenta transversal para gestão de tráfego de microserviços, foi apenas criado através da ferramenta *Kuma,* e encontra-se numa fase de análise e testes para utilização no futuro.

Na figura 18, podemos observar a ferramenta a monitorizar um microserviço.

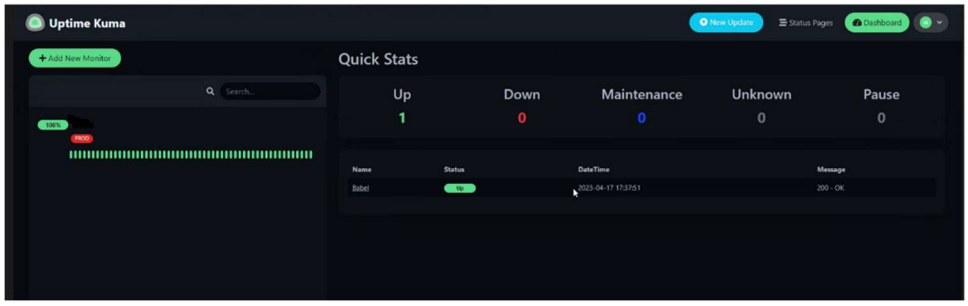


Figura 18 - Exemplo Kuma de monitorização de um microserviço

Fonte: Hovione

No que diz respeito à monitorização, na empresa, esta é realizada através do PRTG (*Paessler Router Traffic Grapher*), ilustrada na figura 19 que é uma ferramenta de monitorização de rede, que pretende ajudar a monitorizar e gerir redes, servidores, dispositivos e aplicações, fornecendo informações detalhadas sobre o desempenho, a disponibilidade e o status dos ativos de IT em tempo real.

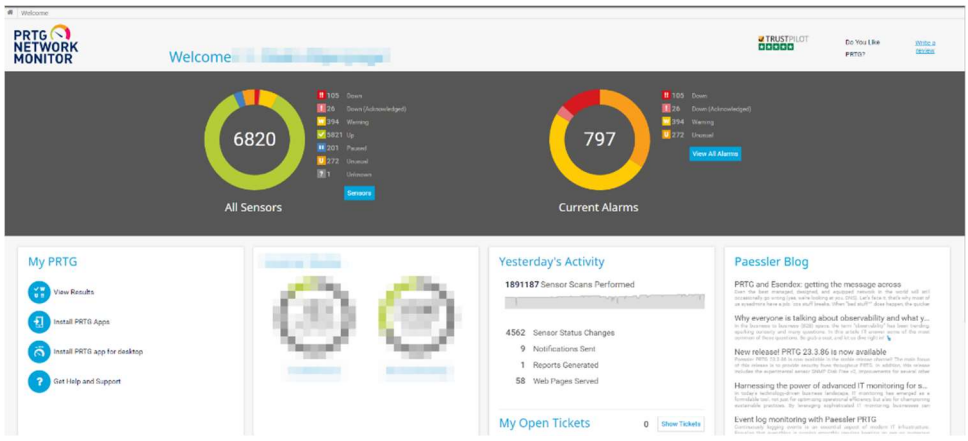


Figura 19 - PRTG

Fonte: Hovione

Existe ainda um *Portainer* que faz a monitorização dos *containers*, que apresenta o estado, as configurações da conta, permite aceder a *logs*, ver o IP, como também as suas ligações e status das configurações.

5. Conclusões

A implementação, desenvolvimento e monitorização de microserviços assumem um papel central nos ambientes contemporâneos de desenvolvimento de *software.* Neste ecossistema em constante evolução, onde as exigências de qualidade, disponibilidade e eficiência são prementes, os microserviços emergem como uma abordagem arquitetónica destinada a responder a essas necessidades com destreza.

A monitorização de microserviços desempenha um papel vital, permitindo às empresas observar de perto o estado dos seus microserviços, recolher métricas essenciais e rastrear interações complexas. Sem esta supervisão constante, a tarefa de manter o desempenho e a disponibilidade dos microserviços torna-se hercúlea, se não impossível. Por outro lado, a análise de desempenho, quando realizada com competência, fornece perceções valiosas para otimizar os sistemas, assegurando que estes atendam às expetativas dos utilizadores.

Não podemos ignorar o facto de que, para dar suporte a esta arquitetura de microserviços, várias ferramentas e tecnologias avançadas, como *Kubernetes, Docker, e Azure DevOps,* tornaram-se amplamente adotadas. Estas ferramentas automatizam processos, simplificam implementações, e são cruciais para manter a integridade dos sistemas.

No entanto, os desafios não podem ser subestimados, especialmente em ambientes distribuídos. A natureza descentralizada de registos e eventos pode tornar a deteção de problemas um quebra-cabeças complexo, exigindo o desenvolvimento de ferramentas e práticas específicas para correlacionar eventos entre serviços.

As métricas-chave, como latência, taxa de erros, utilização de recursos e taxa de tráfego, emergem como faróis orientadores. Fornecem perceções essenciais sobre o desempenho dos microserviços, capacitando a tomada de decisões fundamentadas na otimização.

Neste cenário de evolução constante, as empresas estão a explorar tecnologias avançadas, como serviços *mesh*, para aprimorar a observabilidade e segurança dos seus microserviços.

À medida que a tendência de adotar arquiteturas de microserviços continua a ganhar impulso, várias razões fundamentais impulsionam esta migração. A escalabilidade granular permite às empresas otimizar partes específicas das suas aplicações, aumentando o desempenho e a eficiência. Além disso, a manutenção simplificada e a facilidade de atualização de componentes individuais reduzem consideravelmente os riscos e interrupções operacionais.

A agilidade é uma virtude inestimável num mundo onde a velocidade de desenvolvimento e implementação de funcionalidades é crucial. As equipas de desenvolvimento podem trabalhar de forma independente em microserviços, encurtando o ciclo de desenvolvimento.

A arquitetura de microserviços alinha-se perfeitamente com a crescente procura por sistemas que operam em diversos locais e dispositivos. Além disso, a facilidade de adoção de tecnologias emergentes e a resiliência inerente a falhas são fatores que impulsionam esta tendência. Os microserviços também permitem a escalabilidade horizontal, sendo capazes de lidar com variações de carga de trabalho, o que é crucial em aplicações sujeitas a picos de tráfego.

No entanto, a adoção de microserviços não é uma solução universal. Os desafios de gerir um grande número de serviços e garantir uma comunicação eficaz entre eles não podem ser subestimados. Cada empresa deve considerar cuidadosamente se esta arquitetura é apropriada para os seus casos de uso específicos e se está disposta a investir nas ferramentas e práticas necessárias para enfrentar esses desafios.

No presente caso de estudo, foi possível identificar que a empresa desenvolve internamente microserviços para clientes internos e externos, que segue as melhores práticas e que estas garantem as normas GMP a que estão obrigados na sua atividade, que adota tecnologias avançadas e procura responder aos desafios tanto no desenvolvimento como na monitorização dos microserviços.

Uma parte das mais importantes na arquitetura de microserviços, a monitorização, fundamental para garantir a satisfação dos utilizadores, identificar problemas e otimizar o desempenho do sistema.

Os benefícios da monitorização dos seus microserviços, tais como escabilidade, resiliência, aumento de produtividade, flexibilidade por exemplo, corroboram com os apresentados na revisão de literatura.

A monitorização dos microserviços faz parte de um trabalho futuro que a empresa já deu início, mas, que ainda tem um longo caminho pela frente. Seria interessante analisar os resultados relativos à monitorização, procurar uma ferramenta adequada que permita ter uma visão completa do sistema, que neste momento ainda se encontra como exploratória.

É fundamental investir em soluções de monitorização para garantir a qualidade e eficiência, a empresa deve definir claramente os objetivos que quer atingir com a monitorização, definir as métricas a monitorizar, escolher ferramentas, configurar alertas e analisar os dados recolhidos de forma que se identifiquem possíveis problemas e otimizar o desempenho dos microserviços.

5.1. Limitações do estudo

A indústria farmacêutica é um domínio de extrema sensibilidade devido à natureza altamente confidencial dos dados envolvidos. Em virtude das estritas diretrizes de confidencialidade e segurança de informações, a divulgação de detalhes específicos sobre a estratégia de monitorização de microserviços usada na empresa, torna-se inviável no âmbito deste trabalho.

No entanto, é imperativo sublinhar a significativa relevância da monitorização, no seu contexto. A empresa adotou uma abordagem meticulosa em relação à monitorização, priorizando pilares essenciais, como a segurança das informações, a conformidade rigorosa e a aprimorada eficiência operacional. A monitorização é projetada para assegurar a eficácia nas operações críticas, abrangendo desde a gestão otimizada de recursos e carga de trabalho até a ágil deteção de anomalias e a melhoria contínua do desempenho.

Este cenário de alta complexidade, no qual a sensibilidade dos dados e as necessidades de conformidade regulatória estão em constante equilíbrio com os técnicos a monitorização de microserviços desempenha um papel central, capacitando a empresa a alcançar novos patamares de excelência.

5.2. Trabalho futuro

Com o aparecimento de novas tecnologias e práticas de desenvolvimento de *software*, é essencial considerar a integração de tecnologias emergentes na estratégia de monitorização, particularmente quando o número de microserviços aumenta.

Este trabalho cria espaço para algum trabalho futuro, nomeadamente, estudar as melhores ferramentas de monitorização, inclui a exploração de soluções baseadas em inteligência artificial e automação avançada para aprimorar a deteção de anomalias, a resposta a eventos e a otimização do desempenho.

Este estudo, nomeadamente a revisão da literatura e o levantamento de tecnologias utilizadas na gestão e monitorização dos microserviços, será partilhado com a empresa, o que pode levar a exploração de novas soluções em futuros desenvolvimentos.

Bibliografia

Ahmadvand, M., & Ibrahim, A. (2017). Requirements reconciliation for scalable and secure microservice (de)composition. Proceedings - 2016 IEEE 24th International Requirements Engineering Conference Workshops, REW 2016, 68–73. https://doi.org/10.1109/REW.2016.14

Al-Debagy, O., & Martinek, P. (2018). A comparative review of microservices and monolithic architectures. In *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)* (pp. 000149-000154). IEEE.

Babbie, E. (2015). The Practice of Social Research, Boston. MA: Cengage Learning.

Basiri, A., Behnam, N., De Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., & Rosenthal, C. (2016). Chaos engineering. *IEEE Software*, *33*(3), 35-41.

Baškarada, S., Nguyen, V., & Koronios, A. (2020). Architecting Microservices: Practical Opportunities and Challenges. Journal of Computer Information Systems, 60(5), 428–436. https://doi.org/10.1080/08874417.2018.1520056

Bloor, M., Frankland, J., Thomas, M. & Robson, K. (2001). Focus groups in social research. London: Sage.

Bowen, G. A. (2009). Document Analysis as a Qualitative Research Method. Qualitative Research Journal, 9(2), 27-40.

Brazil, B. (2018). *Prometheus: Up & Running: Infrastructure and Application Performance Monitoring*. " O'Reilly Media, Inc.".

Calcote, L., & Butcher, Z. (2019). *Istio: Up and running: Using a service mesh to connect, secure, control, and observe*. O'Reilly Media.

Carrasco, A., Bladel, B. V., & Demeyer, S. (2018). Migrating towards microservices: migration and architecture smells. In Proceedings of the 2nd International Workshop on Refactoring (pp. 1-6).

Casalicchio, E. (2019). Container orchestration: A survey. *Systems Modeling: Methodologies and Tools*, 221-235.

Chakraborty, M., & Kundan, A. P. (2021). Grafana. In *Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software* (pp. 187-240). Berkeley, CA: Apress.

Chappell, D. A. (2004). *Enterprise service bus: Theory in practice*. O'Reilly Media, Inc.

Chen, L. (2018). Microservices: architecting for continuous delivery and DevOps. In *2018 IEEE International conference on software architecture (ICSA)* (pp. 39-397). IEEE.

De Lauretis, L. (2019). From monolithic architecture to microservices architecture. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* (pp. 93-96). IEEE.

Rajput, D. (2018). *Hands-On Microservices - Monitoring and Testing*. Packt.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, 195-216.

Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R., & Safina, L. (2018). Microservices: How to make your application scale. In *Perspectives of System Informatics: 11th International Andrei P. Ershov Informatics Conference, PSI 2017, Moscow, Russia, June 27-29, 2017, Revised Selected Papers 11* (pp. 95-104). Springer International Publishing.

Erl, T. (2008). *SOA: principles of service design*. Prentice Hall.

Escária, V., Rodrigues, H., Mateus, A., Costa, D., Lopes, F., Ferreira, R. P., & Gouveia, S. (2016). Estudo de Diagnóstico Estratégico da Indústria Farmacêutica em Portugal (Relatório Final).

Forsberg, V. (2019). *Automatic anomaly detection and root cause analysis for microservice clusters.*

Golis, T., Dakić, P., & Vranić, V. (2022). Creating Microservices and using infrastructure as code within the CI/CD for dynamic container creation. In *2022 IEEE 16th International Scientific Conference on Informatics (Informatics)* (pp. 114-119). IEEE.

Hovione. (s.d.). About Hovione. https://www.hovione.com/about-hovione

Indrasiri, K. & Siriwardena, P. (2018). Microservices Governance. In: Microservices for the Enterprise. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-3858-5_6

Jornal de Negócios. (2021). Hovione é o maior empregador privado de doutorados em Portugal, https://www.jornaldenegocios.pt/negocios-iniciativas/premios-exportacao---internaci/detalhe/hovione-e-o-maior-empregador-privado-de-doutorados-em-portugal

Khan, A. (2017). Key characteristics of a container orchestration platform to enable a modern application. *IEEE cloud Computing*, *4*(5), 42-48.

Kleppmann, M. (2017). *Designing Data-Intensive Applications*. O'Reilly Media

Krueger, R. A. & Casey, M. A. (2009) Focus groups: A pratical guide for applied research (4th Ed.). Thousand Oaks, California: Sage.

Kysow, L. (2022). *Consul: Up and Running*. " O'Reilly Media, Inc.".

Lahmadi, A., & Beck, F. (2015, June). Powering monitoring analytics with elk stack. In *9th international conference on autonomous infrastructure, management and security (aims 2015)*.

Lewis, J., & Fowler, M. (2014a). " Microservices." martinfowler.com

Lewis, J., & Fowler, M. (2014b). Microservices: a definition of this new architectural term. MartinFowler. com, 25(14-26), 12.

Oliveira, M.M. (2007). Como fazer pesquisa qualitativa. Petrópolis: Vozes

Mazlami, G., Cito, J., & Leitner, P. (2017). Extraction of Microservices from Monolithic Software Architectures. Paper presented at the 2017 IEEE International Conference on Web Services (ICWS).

Mazzara, M., Dragoni, N., Bucchiarone, A., Giaretta, A., Larsen, S. T., & Dustdar, S. (2021). Microservices: Migration of a mission critical system. *IEEE Transactions on Services Computing*, *14*(5), 1464-1477.

Merriam, S. B. (1998). *Qualitative Research and Case Study Applications in Education. Revised and Expanded from" Case Study Research in Education."*. Jossey-Bass Publishers

Messina, A., Rizzo, R., Storniolo, P., Tripiciano, M., & Urso, A. (2016). The database-is-the-service pattern for microservice architectures. In *Information Technology in Bio-and Medical Informatics: 7th International Conference, ITBAM 2016, Porto, Portugal, September 5-8, 2016, Proceedings 7* (pp. 223-233). Springer International Publishing.

Montesi, F., & Weber, J. (2016). Circuit breakers, discovery, and API gateways in microservices. *arXiv preprint arXiv:1609.05830*.

Newman, Sam. (2015) "Building Microservices: Designing Fine-Grained Systems." O'Reilly Media, 2015.

Nygard, M. T. (2007). Release It!: Design and Deploy Production-ready Software: Pragmatic Bookshelf.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12), 1053-1058.

Pethuru Raj; Skylab Vanga; Akshita Chaudhary, "Kubernetes Architecture, Best Practices, and Patterns," in Cloud-native Computing: How to Design, Develop, and Secure Microservices and Event-Driven Applications, IEEE, 2023, pp.49-70, doi: 10.1002/9781119814795.ch3.

Portainer. (s. d.) Portainer: Docker and Kubernetes Management Platform. https://www.portainer.io/

Richardson, Chris. (2015). "Microservices: The essential practices first goes here." NGINX, Inc, 2015.

Richardson, Chris. (2018). "Microservice Patterns: With examples in Java." Manning Publications., 2018

Sam. "Building Microservices: Designing Fine-Grained Systems." O'Reilly Media

Saman, B. (2017). Monitoring and analysis of microservices performance. *Journal of Computer Science and Control Systems*, *10*(1), 19.

Samartinho, J., & Barradas, C. (2020). A Transformação Digital e Tecnologias da Informação em tempo de Pandemia. Revista UI_IPSantarém, 8(4), 1–6.

Silva, I. S., Veloso, A. L., & Keating, J. B. (2014). Focus group: Considerações teóricas e metodológicas. Revista Lusófona de educação, (26), 175-189

Stake, Robert E. (2003). Case Studies, in N. Denzin e Y. Lincoln (edits.), Strategies of Qualitative Inquiry. California: SAGE. (2ª edição), p. 134-164.

Stake, R. E. (1978). The case study method in social inquiry. *Educational researcher*, *7*(2), 5-8.

Stake, R. (2009). A arte da investigação com estudos de caso. 2. ed. Lisboa: Fundação Calouste Gulbenkian.

Stevens, R. (2020). "Observability in Action: Effective Log Management for Microservices." IT Pro

Sridharan, Cindy. "Distributed Systems Observability". O'Reilly Media, 2018

Thönes, J. (2015). Microservices. *IEEE software*, *32*(1), 116-116.Thones, Jochen. "Microservices." IEEE Software, 2015

Turnbull, J. (2018). *Monitoring with Prometheus*. Turnbull Press.

Villaça, L. H., Pimenta Jr, A. F., & Azevedo, L. G. (2018). Construindo aplicações distribuídas com microsserviços. *Tópicos em Sistemas de Informação: Minicursos XV Simpósio Brasileiro de Sistemas de Informação. SBC*.

Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., ... & Lang, M. (2016). Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (pp. 179-182). IEEE.

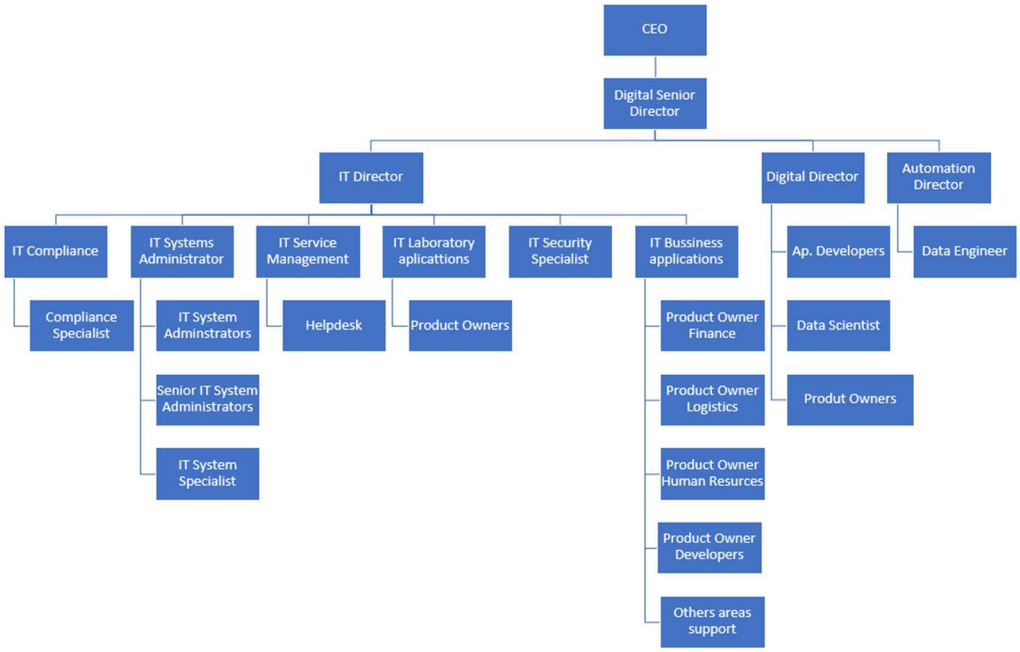
Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. (2015, 21-25 Sept. 2015*). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud*. Paper presented at the 2015 10th Computing Colombian Conference (10CCC).

Yin, R. K. (2009). *Case study research: Design and methods* (Vol. 5). sage.

Waseem, M., Liang, P., Shahin, M., Di Salle, A., & Márquez, G. (2021). Design, monitoring, and testing of microservices systems: The practitioners' perspective. Journal of Systems and Software, 182, 111061.

Anexos

Anexo I – Organigrama Digital Team



Fonte: Elaboração própria