INSTITUTO FEDERAL GOIANO - CAMPUS CERES
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
EMELLY MARINHO PEREIRA SILVA

A EVOLUÇÃO DA ARQUITETURA DE SERVIDORES: COMPARAÇÃO ENTRE ARQUITETURA MONOLÍTICA E MICROSSERVIÇOS

CERES - GO
2023

EMELLY MARINHO PEREIRA SILVA

A EVOLUÇÃO DA ARQUITETURA DE SERVIDORES: COMPARAÇÃO ENTRE ARQUITETURA MONOLÍTICA E MICROSSERVIÇOS

Trabalho de curso apresentado ao curso de Sistemas de Informação do Instituto Federal Goiano – Campus Ceres, como requisito parcial para a obtenção do título de bacharel em Sistemas de Informação, sob orientação do Prof. MSc. Roitier Campos Gonçalves .

CERES - GO

2023

EMELLY MARINHO PEREIRA SILVA

A EVOLUÇÃO DA ARQUITETURA DE SERVIDORES: COMPARAÇÃO ENTRE ARQUITETURA MONOLÍTICA E MICROSSERVIÇOS

Trabalho de curso apresentado ao curso de Sistemas de Informação do Instituto Federal Goiano – Campus Ceres, como requisito parcial para a obtenção do título de bacharel em Sistemas de Informação, sob orientação do Prof. MSc. Roitier Campos Gonçalves .

Banca Examinadora

MSc. Roitier Campos Gonçalves
Instituto Federal Goiano Campus Ceres

MSc. Rangel Rigo
Instituto Federal Goiano Campus Ceres

Hugo de Moura Campos
Instituto Federal Goiano Campus Ceres

Aprovada em _____/_____/_____

Emelly Marinho Pereira Silva
A EVOLUÇÃO DA ARQUITETURA DE SERVIDORES: COMPARAÇÃO ENTRE ARQUITETURA MONOLÍTICA E MICROSSERVIÇOS / Emelly Marinho Pereira Silva. – Ceres, 6 de dezembro de 2023-
35 p. : il. (algumas color.) ; 30 cm.

Orientador: MSc. Roitier Campos Gonçalves

Monografia (Bacharelado) – Instituto Federal Goiano Campus Ceres,
6 de dezembro de 2023.
1. Docker. 2. Kubernetes. 2. Servers. I. MSc. Roitier Campos Gonçalves . II. Instituto Federal Goiano Câmpus Ceres. IV. A EVOLUÇÃO DA ARQUITETURA DE SERVIDORES: COMPARAÇAO ENTRE ARQUITETURA MONOLÍTICA E MICROSSERVIÇOS OS

MOTIVAÇÃO

A motivação para escrever este trabalho baseia-se na conclusão do meu curso e na vontade de contribuir para minha formação e crescimento profissional e acadêmico. Como profissional de TI, tenho interesses nas transformações que a cultura DevOps vem trazendo para o mundo empresarial e nas ferramentas que impulsionam essa mudança. Explorar e estudar o impacto dessas transformações e destacar as ferramentas utilizadas pelo mercado atualmente é uma oportunidade gratificante e enriquecedora para mim.

RESUMO

A evolução dos computadores ocorreu de forma rápida e com mudanças constantes. O mesmo, com os servidores, responsáveis por gerenciar as informações da web. Inicialmente, eles eram máquinas locais, mas com o tempo se transformaram em grandes data centers, capazes de conectar pessoas ao redor do mundo. Com o aumento de usuários, surgiram novas tecnologias, como, por exemplo: máquina virtuais e containers com o objetivo de otimizar o uso de hardware dos servidores. Os servidores monolíticos tornaram-se limitados devido ao alto consumo de hardware e, para contornar esse problema, surgiram as máquinas virtuais, cujo objetivo eram segregar serviços da aplicação web para compartilhar os recursos de máquina do servidor físico. No entanto, como as Maquinas Virtuais demandavam muitos recursos hardware, surgiram os containers, que oferecem uma abordagem mais simples, compartilham o kernel do servidor com não simulam um Sistema Operacional completo como as máquinas virtuais. Com o advento dos containers, tornou-se necessário o desenvolvimento de novas tecnologias para orquestrar e gerenciá-los. Portanto, esta pesquisa tem, como objetivo, uma revisão bibliográfica para comparar a evolução da arquitetura de servidores, destacando as principais ferramentas utilizadas para melhorar o desempenho do hardware. Assim, o entendimento dessas tecnologias é fundamental para disseminar o conhecimento e aplicá-lo em novas aplicações web e servidores legados.

Palavras-chave: docker. kubernetes. arquitetura. monolítico. máquinas virtuais

ABSTRACT

The evolution of computers has occurred rapidly with constant changes. The same goes for servers, responsible for managing web information. Initially, they were local machines, but over time, they transformed into large data centers capable of connecting people around the world. With the increase in users, new technologies emerged, such as virtual machines and containers, with the aim of optimizing the use of server hardware. Monolithic servers became limited due to high hardware consumption, and to overcome this problem, virtual machines emerged, which aimed to segregate web application services to share the physical server's resources. However, as Virtual Machines demanded significant hardware resources, containers emerged, offering a simpler approach by sharing the server's kernel and not simulating a complete operating system like virtual machines. With the advent of containers, it became necessary to develop new technologies to orchestrate and manage them. Therefore, this research aims to provide a literature review to compare the evolution of server architecture, highlighting the main tools used to improve hardware performance. Understanding these technologies is crucial for disseminating knowledge and applying it in new web applications and legacy servers.

Keywords: docker. kubernetes. containers. architecture. monolithic. virtual machines

LISTA DE ABREVIATURAS E SIGLAS

ARPANET	Advanced Research Projects Agency
CPU	Central Processing Unit
IP	Internet Protocol
RAM	Randon Access Memory
TCP	Transmission Control Protocol
VMs	Virtual Machines

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Desde os anos de 1960 até o início do século XXI, era comum o uso de servidores com arquitetura monolítica para disponibilizar sites na internet. De acordo com (GOS; ZABIEROWSKI, 2020), a aplicação monolítica é um servidor no qual é comum ter todas as funcionalidades de uma aplicação funcionando em um único servidor, como, por exemplo, o código de aplicação, banco de dados (responsável por armazenar as informações fornecidas pelos usuários), *front-end* (interface gráfica para o usuário interagir com o site) entre outros componentes para o funcionamento de um serviço web. Porém, surgiram limitações nesse modelo de arquitetura de servidores, como, por exemplo: altos custos de manutenção, atualização do sistema (por se tratar de um sistema integrado e complexo), a falta de escalabilidade (habilidade de uma infraestrutura de se adaptar a mudanças no fluxo de dados conforme o crescimento da empresa), e com um número crescente de usuários sugiu a necessidade cada vez mais recorrente de criar sistemas eficientes para a disponibilização de serviços web (GOS; ZABIEROWSKI, 2020). Esses foram os principais motivos para a busca por mudanças e a criação de tecnologias que se adéquem melhor a essas situações.

Com as complicações do uso de uma única máquina para a aplicação web, o conceito de virtualização foi introduzido. A virtualização utiliza software para criar uma camada de abstração sobre o hardware de um computador, permitindo que os elementos de hardware de um único computador - processadores, memória, armazenamento e outros - sejam divididos em múltiplos computadores virtuais, comumente chamados de máquinas virtuais (DIAS; CORREIA; MALHEIROS, 2021). Com o modelo de virtualização, cada aplicação web poderia ser designada em uma máquina virtual diferente, como o código do site e o banco de dados. Contudo, as máquinas ainda possuíam limitações devido ao sobrecarregamento dos servidores, sendo necessário a aquisição de mais recursos hardware, uma vez que cada máquina virtual era uma simulação de um sistema completo (NISSENBAUM, 2001).

De acordo com Dias, Correia e Malheiros (2022), as máquinas virtuais podem sofrer com um desempenho reduzido devido à necessidade de emulação de hardware e à camada de abstração entre a máquina virtual e o hardware físico subjacente. Esses fatores podem desencadear um consumo significativo de energia no servidor, como apresentado por (SRIVASTAVA; JOSHI, 2023), no qual os servidores web virtualizados obtiveram um consumo de 40% a mais do que os não virtualizados. Além disso,

> [...] as máquinas virtuais possuem um desempenho instável devido o processo de inicialização leva muito tempo para ser executado e as máquinas virtuais não conseguem resolver dificuldades como capacidade de gerenciamento, atualizações de software e integração/entrega contínua (POTDAR et al. 2020).

Esses motivos foram os principais fatores para a busca por mudanças e recursos mais eficientes. Assim, a segregação das aplicações web deveriam continuar a separar e a organizar esses serviços dentro do servidor, mas com uma tecnologia que não exigisse de forma tão brusca os recursos do servidor (POTDAR et al., 2020). Desta forma, ao invés de simular um computador completo, a solução foi virtualizar apenas os recursos necessários para um determinado serviço (POTDAR et al., 2020). A grande diferença foi compartilhar o Kernel com o hardware, atuando como intermediário entre o hardware e o software. No Kernel do host, os binários e as bibliotecas específicas do aplicativo web podem ser processados por meio do carregamento na memória, alocação de recursos e início da execução, tornando a execução mais rápida (POTDAR et al., 2020).

O processamento de dados diretamente no kernel evita a sobrecarga de simulação de um sistema completo, o que contribui para a velocidade de execução da aplicação em containers (NGUYEN et al., 2020). A tecnologia dos *containers* que consiste em um modelo de virtualização mais leve que a anterior que, ao invés de simular todo um sistema, apenas virtualiza o que é necessário para uma parte da aplicação (POTDAR et al., 2020), tornando uma abordagem mais eficiente que pode otimizar o desempenho geral da aplicação. Posteriormente, em meados de 2013, criou-se uma plataforma de conteinerização para isolar ambientes e aplicações chamada *Docker* (KUMAR; KAUR, 2022). Essa tecnologia foi capaz de segregar uma aplicação de sua infraestrutura, garantindo uma melhor flexibilidade na configuração do servidor por meio da utilização de *containers*. De acordo com POTDAR et al. (2020):

> [...] observa-se que os contêineres do *Docker* têm melhor desempenho sobre a VM em todos os testes, pois a presença da camada QEMU na máquina virtual a torna menos eficiente que os containers *Docker*.

Potdar et al. (2020) apresentou a nível de hardware as vantagens do uso da conteinerização com bases nas diferenças de virtualização, observando que os contêineres do *Docker* possuem melhor desempenho sobre a VM (*Virtual Machines*) em todos os testes uma vez que a presença da camada QEMU (*Quick Emulator*) na máquina virtual - um virtualizador de hardware para testes - a torna menos eficiente que os *containers Docker*. Nesse contexto, diferentemente

das primeiras versões da arquitetura de servidores que apresentavam uma abordagem monolítica na configuração do software, agora, com a segregação desses serviços encapsulados com conteiners se obtem um servidor com arquitetura de software com microsserviços, que consiste, justamente em segregar uma aplicação web ou mobile em pequenos serviços cada um com sua função no servidor (SELIVORSTOVA et al., 2021). Conforme a complexidade dos servidores cresciam e, mesmo com o uso de *containers*, ainda surgiram complicações para lidar com uso de muitos *containers* dentro de um servidor, foi necessário implementar uma nova tecnologia que automatizasse o gerenciamento de um grande fluxo de *containers* (KUMAR; KAUR, 2022). Assim, em 2014, foi lançado o *Kubernetes*, responsável por gerenciar fluxos de *containers* e escalonar o fluxo de carga conforme a necessidade (de quem), sendo responsável pelo gerenciamento dos *containers* (RENSIN, 2015). De acordo com Nguyen et al. (2020),

> [...] no Kubernetes, um dos recursos mais importantes é o escalonamento automático, pois permite que aplicativos e serviços sejam executados de forma resiliente sem a necessidade de intervenção humana.

Mesmo com o escalonamento automático e com toda a automação de infraestrutura, também surgiram diversas ferramentas de gerenciamento que visaram simplificar e aprimorar a administração de ambientes de forma simples e visual (GOLI et al., 2020). Em 2015, a empresa Portainer desenvolveu uma plataforma de gerenciamento de contêineres (portainer.io), oferecendo uma interface de usuário intuitiva para gerenciar clusters (Conjunto de servidores) do *Docker* (MAHDAVI-HEZAVEH; DREMANN; WILLIAMS, 2021)). Com suas diversas funcionalidades, o Portainer permite que os administradores visualizem, gerenciem e monitorem contêineres, imagens, redes e volumes de forma centralizada, fornecendo uma visão detalhada dos recursos e estatísticas de desempenho, oferecendo uma solução versátil para simplificar a administração e o monitoramento de ambientes *Docker*.

Neste trabalho, serão avaliados os impactos de diferentes formas na gestão de serviços e operações de empresas, junto com melhores práticas para a disponibilização de software em um servidor. Logo, será analisada a transição de mudanças na forma de arquitetura de servidores e demais tecnologias, será explorado estudos de caso e pesquisas que evidenciem os benefícios obtidos com a utilização de ferramentas para melhorar o desempenho de hardware dos servidores, como a redução de tempo e esforço na administração, a simplificação da implantação de aplicativos web e a melhoria na visibilidade e controle do ambiente. Portanto, este estudo pretende fornecer uma análise comparativa globalizante entre os servidores monolíticos e os

servidores atuais com *containers*, destacando suas vantagens e principais diferenças, contribuindo para o conhecimento da evolução na forma da configuração de servidores.

As temáticas apresentadas nesta pesquisa serão abordadas de forma técnica ao longo do trabalho, explicitando os detalhes sobre as tecnologias e seus usos. Por fim, espera-se fornecer informações sobre os tipos de servidores existentes, sua evolução e seus impactos na gestão de serviços, contribuindo para a automação dos processos de implantação, escalabilidade e administração de contêineres e clusters. Além disso, espera-se fornecer uma análise dos principais impactos e mudanças de um tipo de servidor para o outro, permitindo uma visão consciente e alinhada com tecnologias inovadoras para a construção de uma arquitetura de servidor alinhada com ferramentas de software atuais.

1.1 Objetivo Geral

Comparar servidores monolíticos e servidores que usam microsserviços a nível de uso de hardware e otimização de software com base na literatura científica, apresentar as principais alterações e vantagens na forma de organizar uma infraestrutura de servidor com tecnologias atuais.

1.2 Objetivos específicos

Investigar a arquitetura e funcionamento de servidores monolíticos, visando suas características, limitações e desafios em relação à escalabilidade, manutenção e flexibilidade, explorando as formas de conteinerização como tecnologias com funções de segmentar as aplicações em diversos serviços e sua maior flexibilidade.

Apresentar ferramentas e tecnologias utilizadas no contexto do uso de máquinas virtuais, *containers* com o uso do *Docker*, *Kubernetes* e ferramentas de gerenciamento de ambiente com interface, como, por exemplo, o Portainer.

Avaliar a eficácia e benefícios oferecidos pela adoção de *containers* em comparação com servidores monolíticos, como escalabilidade, flexibilidade, rápida implantação e atualização de serviços e diferenças de desempenho e gastos entre cada tipo de arquitetura.

2 JUSTIFICATIVA

A internet é a forma mais rápida de se conectar, obter e enviar informações para outras pessoas, onde todas as informações são armazenadas em servidores complexos espalhados pelo mundo, chamados de Data Centers (ABBOTT; FISHER, 2015). Com um grande fluxo de informações, é preciso acompanhar a evolução tecnológica para, assim, estar situado com as melhores práticas de desenvolvimento de software e ferramentas utilizadas no mercado e garantir as melhores ferramentas para as aplicações web de uma empresa ou projeto.

Portanto, criar sistemas em servidores que não se adéquem às tecnologias atuais pode gerar diversos empecilhos ou perdas para as empresa de tecnologias, como no caso da rede social *Friendster's*, fundada em 2002 para usuários dos Estados Unidos da América, era uma plataforma de rede social que antecedeu ao *Facebook* e tinha algumas semelhanças com ele. Ambas as plataformas permitiram que os usuários criassem perfis, se conectassem com amigos e compartilhassem conteúdo como fotos e mensagens (CORRY, 2022). Com o tempo, ele ganhou popularidade e expandiu sua base de usuários, principalmente no Sudeste Asiático. Porém, a plataforma enfrentou problemas técnicos de escalabilidade, o que impediu seu crescimento e sucesso. Esses desafios incluíram problemas de capacidade de hardware do servidor devido ao grande número de usuários e ao alto volume de *uploads* de imagens. Com isso, a rede social enfrentou vários desafios e o *Facebook* acabou superando o *Friendster* em popularidade e base de usuários, levando ao declínio e eventual encerramento do *Friendster* em 2018 (CORRY, 2022).

Conforme ilustrado, configurar um servidor sem planejamento e não seguir boas práticas de automação de infraestrutura de servidor pode ser prejudicial para uma organização. Esse exemplo mostra uma das causas de implementar uma arquitetura de servidor mais atual para evitar prejuízos catastróficos por conta do uso de tecnologias de arquitetura de servidores ultrapassadas. Para suportar a complexidade dos softwares atuais e reduzir o consumo excessivo de hardware dessas máquinas, foram criadas ferramentas de configuração para servidores com o objetivo de atender novos desafios (MAHDAVI-HEZAVEH; DREMANN; WILLIAMS, 2021). Com isso, entender quais são essas ferramentas e mudanças é fundamental para tomar decisões mais eficientes em relação a criação de novos sistemas, impactando na qualidade de novos softwares de maneira positiva.

3 METODOLOGIA

Este estudo consistiu em uma pesquisa bibliográfica exploratório sobre servidores monolíticos e máquinas virtuais, relacionados às tecnologias *Docker, Kubernetes e Portainer,* destacados suas principais características e explorando suas principais funcionalidades. Assim, realizou-se pesquisas em documentações oficiais de cada plataforma e em artigos científicos para apresentar o funcionamento de cada tecnologia e seus principais casos de utilização para uma aplicação web e servidores, evidenciando suas vantagens e impactos fornecidos para o usuário, como, por exemplo, usabilidade e escalabilidade.

Com isso em pauta, foi fornecido um comparativo entre os tipos de configuração de um servidor monolítico e um servidor configurado com base em *container*, analisando o impacto desses gerenciadores na gestão de serviços, incluindo estudos de caso e pesquisas que evidenciem os benefícios obtidos com sua utilização.

Para a coleta de informações dos artigos científicos, foram usadas as bases de dados Google Acadêmico, *Mendeley, Scielo e IEEE Xplore, ScienceDirect*, além de documentações oficiais de cada plataforma. Foram utilizadas as palavras-chave: *container, server, monolithic, virtual machine, Docker, kubernetes, portainer, architecture, Evolution*. Considerou-se como critérios de seleção dos artigos os estudos que formularam comparativos entre os dois tipos de servidores abordados, monolítico e com o uso de microsserviços e apresentaram as vantagens de usar tecnologias atuais para a configuração de servidores em relação aos métodos tradicionais.

Essa abordagem permitiu uma análise comparativa dos aspectos de escalonamento entre os servidores monolíticos e os servidores baseados em *containers Docker*, gerenciado pelo Portainer. Através da comparação desses dois ambientes, foi possível entender sobre a eficiência operacional e escalabilidade alcançadas com a utilização de *containers Docker* em comparação com o tradicional servidor monolítico.

Além de contribuir para a disseminação do conhecimento sobre o assunto, seus gerenciadores e destacar a evolução na forma de configuração de servidores, os resultados obtidos a partir dessa análise contribuirão para o avanço e sintetização do conhecimento na área de arquiteturas de servidores e auxiliarão na tomada de decisões sobre a escolha da abordagem mais adequada para implantação e escalonamento de serviços na construção de websites.

4 REVISÃO DE LITERATURA

4.1 Servidores monolíticos

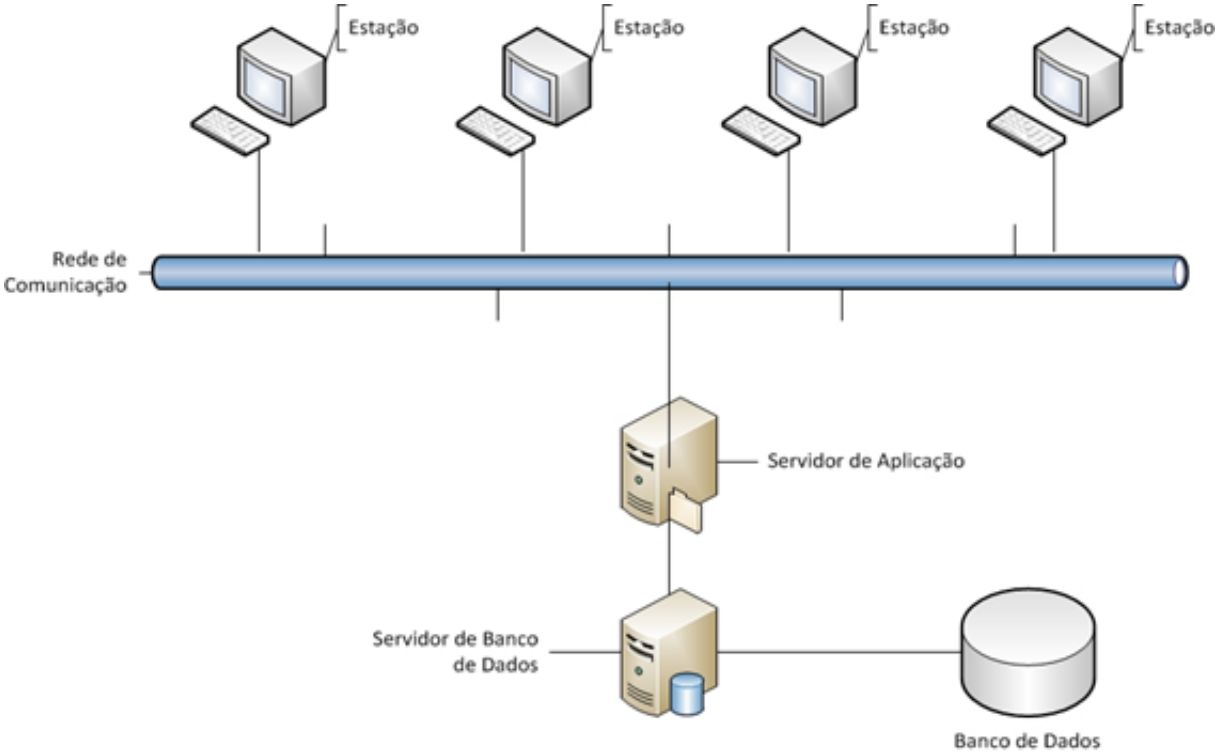
Criada no final da década 1960, a ARPANET (*Advanced Research Projects Agency*) foi uma das primeiras redes de computadores que usavam o protocolo TCP/IP (*Transmission Control Protocol/Internet Protocol*) para fins acadêmicas nos estados unidos, dando início ao desenvolvimento da internet (HAFNER; LYON, 1998). Nessa época era utilizado um servidor de arquitetura monolítica, que tinha como principal característica ser um servidor que compartilhava os recursos de hardware diretamente com outros serviços para o funcionamento de uma aplicação web, como por exemplo a base de dados. De acordo com Villamizar et al. (2015)

> [...] em aplicativos monolíticos todos os serviços são desenvolvidos em uma única base de código compartilhada entre vários desenvolvedores, quando esses desenvolvedores desejam adicionar ou alterar serviços, é preciso garantir que todos os outros serviços continuem funcionando.

Um servidor monolítico é um tipo de arquitetura de servidor em que todos os componentes e funcionalidades de um aplicativo são totalmente integrados em uma única pilha de software (BIGGS; LEE; HEISER, 2018), ele segue uma abordagem tradicional e centralizada em que todo o aplicativo é executado em uma única instância de servidor. Isso significa que todos os componentes de processamento, armazenamento de dados e interface do usuário são combinados em uma única unidade. Os servidores monolíticos são conhecidos por sua simplicidade e facilidade de implantação, pois não exigem sistemas distribuídos complexos (BIGGS; LEE; HEISER, 2018). No entanto, eles podem ser limitados em termos de escalabilidade e flexibilidade, pois quaisquer alterações ou atualizações no aplicativo exigem a modificação de todo o monólito (FAN; MA, 2017).

A configuração de servidores envolvia a instalação e configuração manual de sistemas operacionais e softwares em máquinas físicas. Com isso, o aplicativo ou serviço exigia um servidor dedicado, como apresentado na Figura 1 (VILLAMIZAR et al. 2015). Nessa imagem é apresentado um servidor com duas máquinas físicas, no qual o "Servidor de aplicação" é onde o código da aplicação web está armazenada e o "Servidor de banco de dados" são os dados que serão recebidos pelos usuários e serão armazenados. Neste caso, a rede de comunicação é responsável por fazer os usuários acessarem essa aplicação por meio da "estação" que são os computadores.

Figura 1 – Macro esquematização Servidores monolíticos



Disponível em: https://static.imasters.com.br/wp-content/uploads/2013/10/bdd-1.png. Acesso em: 2 de junho. 2023

Com base nisso, mesmo que este tipo de arquitetura se apresentasse simples e de fácil configuração, a longo prazo, a atualização do código se tornou difícil aos desenvolvedores (GOLI et al. 2020). Além disso, conforme TAPIA et al. (2020), o desempenho das aplicações monolíticas é afetado quando a quantidade de dados a ser processada aumenta ou excede um certo nível de capacidade. Logo, o modelo de arquitetura monolítica se demonstrou pouco eficaz conforme a complexidade dos sistemas e o número de usuários expandiam. No mesmo período, surgiram as máquinas virtuais com o intuito de melhorar o desempenho dos servidores e compartilhar os processos de uso da máquina de forma paralela (RANDAL, 2020)

4.2 A virtualização de computadores, funcionalidades e limitações

As máquinas virtuais, *virtual machines* (VMs), que são simulações de um computador dentro de um computador físico, trouxeram uma abordagem mais eficiente para a configuração de servidores (PORTNOV, 2012). As VMs permitiam a criação de ambientes isolados e independentes em um único servidor físico. Assim, como cada VM continha um sistema operacional completo, incluindo aplicativos e serviços, isso permitia a consolidação de vários

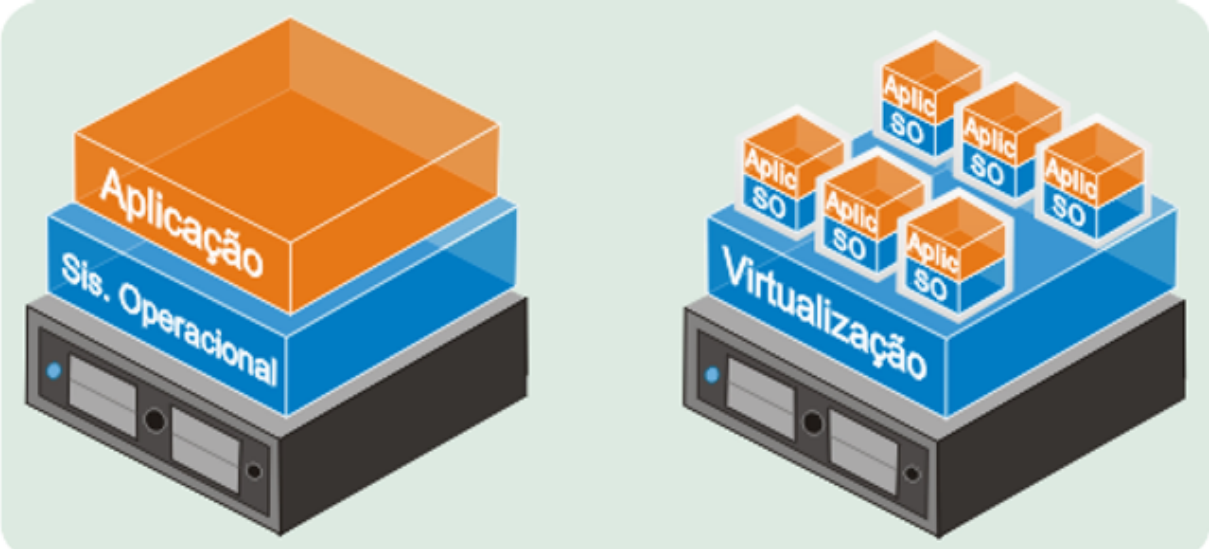
servidores virtuais em uma única máquina física, proporcionando uma melhor utilização de hardware.

De acordo com SHROPSHIRE, 2014, o Kernel

> [...] atua como intermediários entre as máquinas virtuais e o hardware subjacente. Para compartilhar recursos com as máquinas virtuais hospedadas, ele abstrae o hardware do sistema e gerencia as solicitações de uso.

A Figura 2 apresenta as mudanças de uso de hardware, no qual é possível observar um servidor onde toda sua aplicação é armazenada em um único servidor (à esquerda), e o kernel, um intermediário entre o hardware e o software, que é responsável por executar as funções do sistema operacional e controlar os recursos do servidor. Segundo Randal (2020), um dos objetivos fundamentais de adicionar multiprogramação a hardware e sistemas operacionais no final da década de 1950 era melhorar o desempenho por meio da utilização mais eficiente dos recursos disponíveis compartilhando-os em processos paralelos.

Figura 2 – Diferença operacional de VMs e Containers.

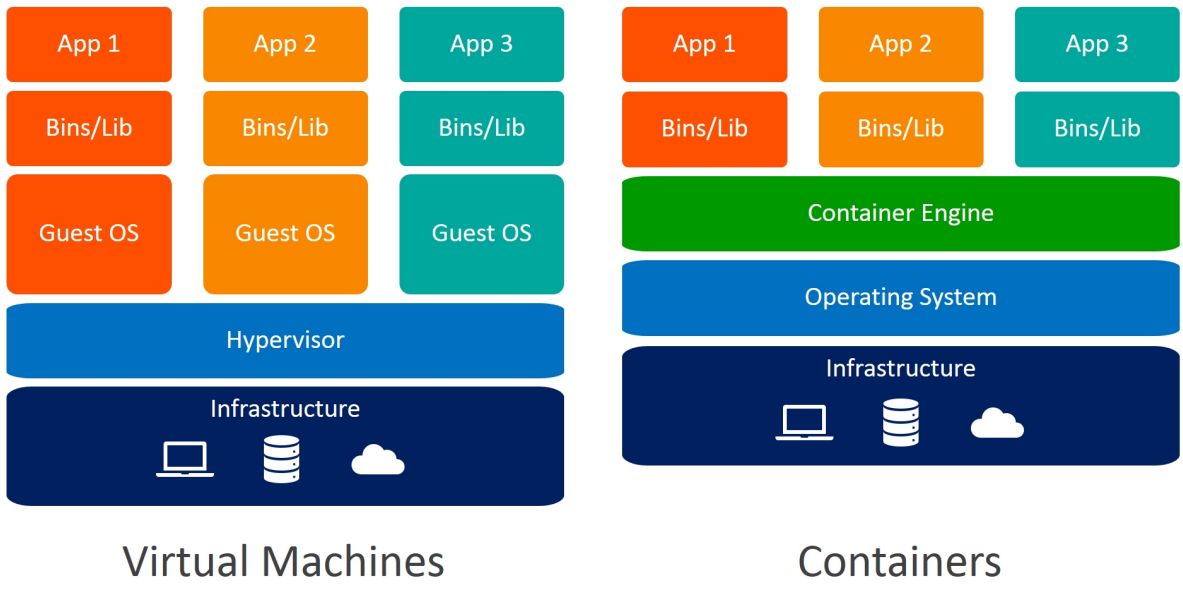


Disponível em: Acesso em: 2 de junho. 2023

https://www.teleco.com.br/imagens/tutoriais/tutorialdatacenter1figura03.jpg

E em seguida, é possível observar na Figura 2 um exemplo de servidor com virtualização (à direita), onde cada funcionalidade e serviço de uma determinada aplicação é isolada das demais, com seus próprios recursos alocados e sistema operacional. O kernel, neste caso, é compartilhado de forma limitada com cada VM, por conta do hypervisor, que é um software de isolamento para o kernel onde torna possível o uso de outros sistemas operacionais dentro de uma máquina como mostra a Figura 3, tornando mais fácil a configuração e manutenção desses servidores (JOY, 2015) .

Figura 3 – Diferença de aplicações entre VMs e Containers



Disponível em: https://images.contentstack.io/v3/assets/blt300387d93dabf50e/bltb6200bc085503718/5e1f209a63d1b6503160c6d5/containers-vs-virtual-machines.jpg. Acesso em: 2 de junho. 2023

Mesmo com essas melhorias do uso dos recursos das máquinas, as VMs ainda possuíam algumas limitações, como o alto consumo de recursos devido à duplicação de sistemas operacionais completos. Além disso, o tempo de inicialização das VMs e a sobrecarga de gerenciamento eram consideráveis FELTEN et al. (2015), por exemplo, demonstrou que a máquina virtual leva mais tempo para processar até mesmo uma única solicitação. Com esses desafios, foi necessário tomar novas medidas para melhorar o desempenho dos servidores.

Assim, foi necessário criar outra tecnologia que atendesse melhor às necessidades das novas mudanças, chamada de *containers*, que se caracterizam pela virtualização de softwares e não uma simulação completa de sistemas, como as máquinas virtuais (MIELL; SAYERS, 2019). De acordo com JOY (2015), a conteinerização, como mostrado, ainda na Figura 3 é uma abordagem na qual é possível executar muitos processos de forma isolada, que utiliza apenas um kernel para múltiplos ambientes isolados e compartilha o sistema operacional hospedeiro e incluindo apenas as dependências necessárias para os aplicativos.

4.3 O fundamento dos containers Docker

Os *containers* se tornaram populares em 2013 pelo *Docker*, uma plataforma disponível para o uso dessa tecnologia, disponível através do endereço website https://www.docker.com.

A plataforma do *Docker* mudou a maneira como os serviços são configurados e implantados. Segundo Mouat (2015), os objetivos fundamentais das VMs e dos *containers* são diferentes, sendo o propósito de uma VM emular completamente um ambiente estrangeiro, enquanto o *container* tornar as aplicações portáteis e autocontidas.

Assim, ao contrário das VMs, os *containers* compartilham o kernel do sistema operacional do host, o que resulta em uma menor sobrecarga e maior eficiência de recursos. Cada *container* contém apenas os componentes necessários para a execução do serviço, como bibliotecas que são subprogramas de um sistema para automação no desenvolvimento de software, o que os torna mais leves e rápidos de inicializar (JOY, 2015). Com os containers, a configuração dos servidores se tornou muito mais ágil, pois:

> [...] os containers compartilham não apenas recursos físicos, mas também o sistema operacional e as bibliotecas de suporte, enquanto as VMs tradicionais baseadas em hipervisor oferecem apenas uma abstração no nível do hardware (WAN et al. 2018, p.2).

Portanto, criar um ambiente conteinerizado é mais leve pois cada serviço da aplicação web estará com recursos de hardware específicos para uma determinada função dentro da aplicação web, com isso, usar *containers* proporciona uma maior consistência e pode evitar problemas de dependências ou má alocação do uso de hardware. Assim, a implantação de aplicativos pode ser simplificada, pois os *containers* são capazes de serem construídos uma vez e implantados em qualquer ambiente compatível com o *Docker*. Além disso, a escalabilidade é facilitada, uma vez que é possível executar vários *containers* em um único host ou distribuí-los em vários hosts (WAN et al. 2018a). Ao utilizar *containers*, empresas e desenvolvedores ganham benefícios como a padronização de ambientes e a portabilidade de aplicativos. Portanto, criar ambientes com propósitos para desenvolvimento, teste de software ou aplicação web de produção podem ser facilmente replicados uma vez que a aplicação web já estará com suas funções de serviços padronizadas decorrente ao uso dos *containers* (FELTER et al. 2015).

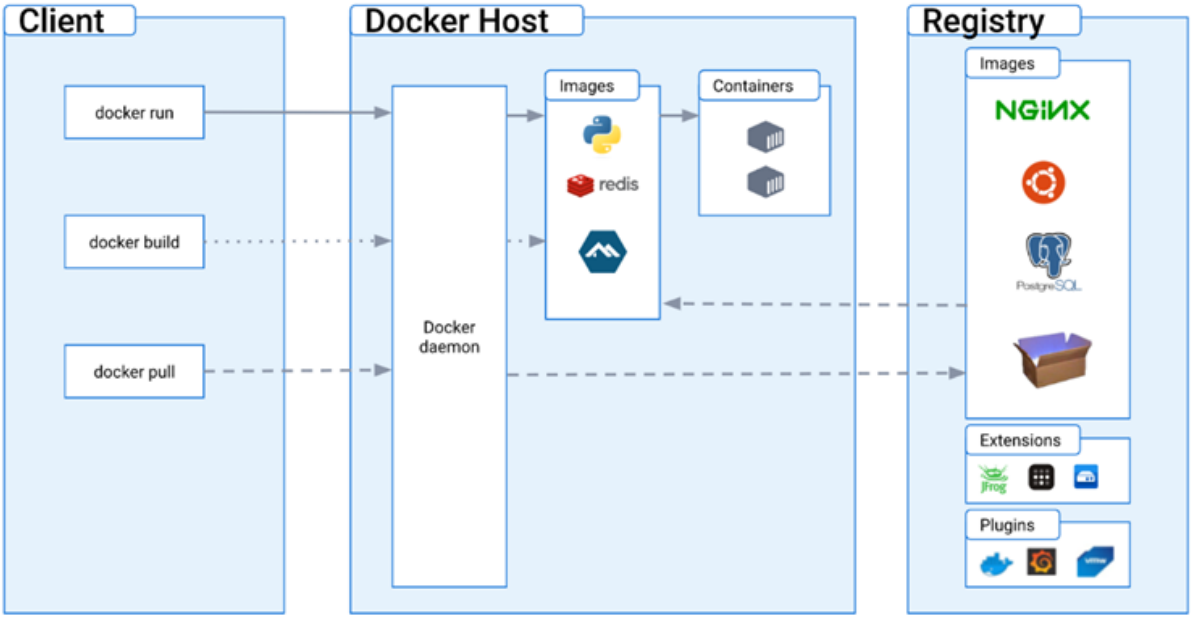
4.4 Ambiente com container e suas funcionalidades

O *Docker* trouxe uma nova abordagem para a implantação de serviços, permitindo que os aplicativos e suas dependências sejam empacotados em *containers* independentes (JARAMILLO; NGUYEN; SMART, 2016). Esses *containers* são leves, portáteis e isolados, o que facilita a implantação consistente e a execução de serviços em diferentes ambientes (RANDAL, 2020). Neste ponto, é importante explorar as principais funcionalidades do *Docker*,

como a construção de imagens que são a base para o aplicativo da ferramenta que o usuário deseja usar no servidor, a criação de *containers* e a gerência de volumes, onde os dados dos containers são armazenados, além de discutir os benefícios que essa tecnologia traz para a gestão de serviços (CHUNG et al., 2016)

A Figura 4, a seguir, apresenta a arquitetura padrão do *Docker* desde a parte de comando, até imagens e ferramentas utilitárias. O Cliente, neste caso, é a parte de linha de comando onde os usuários podem interagir com a ferramenta como criar, executar, parar ou excluir *containers*, construir imagens, gerenciar redes e volumes, entre outras operações (SINGH; SINGH, 2016)

Figura 4 – *Arquitetura do Docker*



Disponível em: https://docs.docker.com/assets/images/architecture.svg . Acesso em: 2 de junho. 2023

Docker host pode ser uma máquina física ou uma máquina virtual, que possui o ambiente *Docker* instalado (MOUAT, 2015). Ele fornece os recursos necessários suportar a execução dos *containers*, como CPU *(Central Processing Unit)*, que é o componente central de processamento do computador; Memória RAM *(Randon Access Memory)*, a memória primária que é responsável por armazenar temporariamente instruções do processador; armazenamento, que se trata de uma memória a longo prazo e responsável por armazenar os dados do computador; e rede, uma infraestrutura responsável por fazer a conexão entre computadores e servidores (SWARNALATHA; SHANTHI, 2014). Já o *Docker Engine* é responsável por garantir

o isolamento entre os *containers*, atribuindo recursos apropriados a cada um e garantindo que eles não interfiram uns nos outros (SYED; FERNANDEZ, 2017)

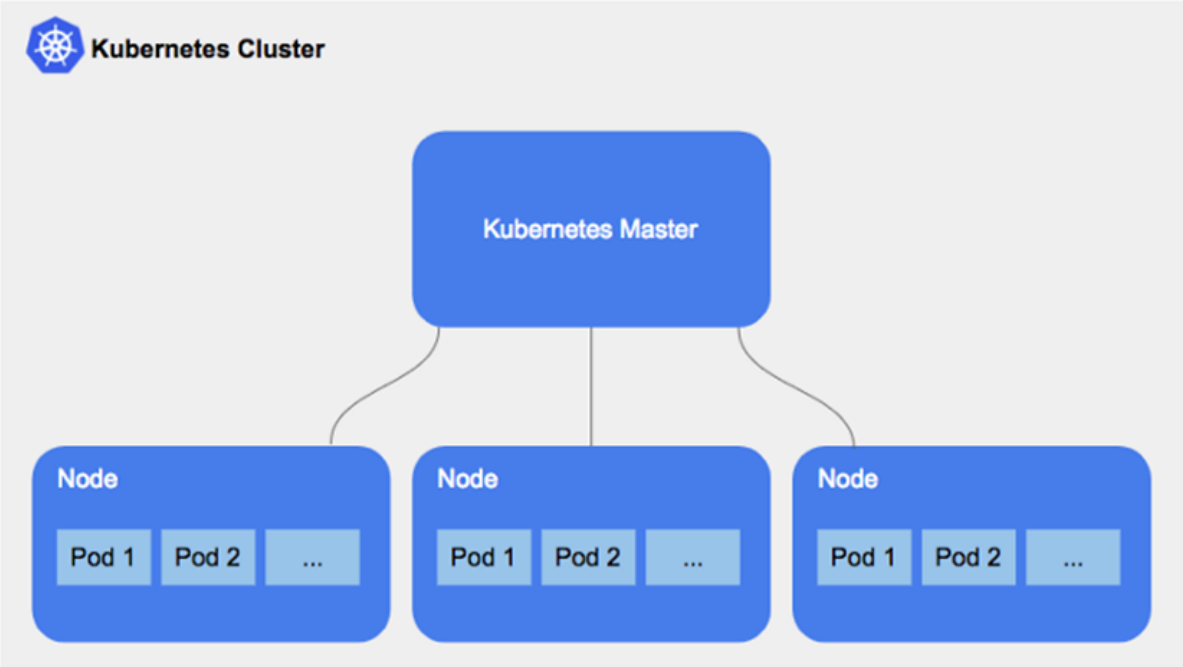
Por outro lado, o *Docker Registry* é uma plataforma de repositórios que viabiliza o armazenamento e compartilhamento de imagens *Docker*, que são unidades executáveis leves e isoladas, contendo todos os componentes necessários para a execução de um software específico (SANTOS et al., 2018). Já para registrar e usar essas imagens, é usado o *Docker Hub*, que são repositórios online que permitem que os usuários criem suas próprias imagens localmente e as disponibilizem para uso público ou privado, em que a plataforma pode ser acessada em https://hub.docker.com.

Porém, conforme a adoção do uso dos containers, surgiu a necessidade de gerenciar esses *containers* em grande escala. Assim, em 2014, a Google lançou os Kubernete como uma ferramenta de orquestração de containers em escala, disponível em https://kubernetes.io/pt-br/. De acordo com Menouer (2021), a capacidade inerente do *Kubernetes* de ser portátil e expansível, juntamente com sua otimização para todas as infraestruturas e aplicações, justifica o seu uso em vez de *containers* isolados.

4.5 O uso do Kubernetes como orquestrador de containers

Um cluster *Kubernetes* é um sistema para implantar e gerenciar *containers Docker* que hospeda microsserviços (PONISZEWSKA-MARAŃDA; CZECHOWSKA, 2021). Diferentemente dos servidores com arquitetura monolítica, o *cluster* é um conjunto de computadores que trabalham juntos, mas que formam um sistema único. A Figura 5 mostra essa distribuição dentro do servidor. Primeiramente, a máquina *Kubernetes Master* é responsável por ordenar o funcionamento dos *Nodes*, que são as máquinas *Workers* responsáveis por armazenar os *Pods* (que pode ter entre um ou mais *containers*). Assim, caso um *Node* fique indisponível para os usuários, o *Node Master* irá escalar outro node com a aplicação (THURGOOD; LENNON, 2019).

Figura 5 – *Arquitetura de cluster Kubernetes.*



Disponível em: https://medium.com/@tomerf/so-you-want-to-configure-the-perfect-db-cluster-inside-a-kubernetes-cluster-a4d2c26aca7a. Acesso em: 15 de setembro. 2023

Tendo em mente o funcionamento base da ferramenta *Kubernetes*, é preciso destacar as funcionalidades e funcionamento do *Kubernetes*, com o uso de arquivos do tipo YAML (*YAML Ain't Markup Language*), criando estruturas para a configuração do servidor. Esses arquivos permitem a definição declarativa de dados de forma estrutural dos aplicativos e determinam o comportamento do *Kubernetes* em relação ao cluster e aos *containers* (NGUYEN et al., 2020). Essa arquitetura distribuída composta pelo *Kubernetes Master* e *Nodes Workers*, juntamente com suas funcionalidades avançadas, proporciona automação e escalabilidade.

> [...] Ao utilizar o Kubernetes para o agendamento de containers, é possível otimizar a alocação de recursos, melhorar a utilização de CPU, memória e disco, reduzir o consumo de energia, equilibrar a carga entre os nós e melhorar o tempo de resposta (MENOUER, 2021).

A tecnologia de *containers*, aliada a ferramentas de orquestração como o *Kubernetes*, proporciona uma abordagem mais flexível, eficiente e escalável para o gerenciamento de serviços. A adoção de *containers* tem impactado positivamente empresas de diferentes setores, permitindo o desenvolvimento ágil, a implantação rápida de aplicações, a redução de custos operacionais e a melhoria na experiência do usuário.

4.6 A importância dos gerenciadores de infraestrutura

A configuração de servidores evoluiu de um processo manual e demorado para um ambiente automatizado, no qual a infraestrutura pode ser tratada como código e os serviços podem ser implantados de maneira rápida e consistente (MAHDAVI-HEZAVEH; DREMANN; WILLIAMS, 2021). Essa transformação tem implicações significativas no mundo empresarial, impulsionando a inovação, facilitando a colaboração entre equipes de desenvolvimento e operações e possibilitando a entrega contínua de valor aos clientes (MENOUER, 2021). A compreensão desse contexto é fundamental para que as empresas se mantenham competitivas e adotem abordagens modernas de gestão de serviços, aproveitando ao máximo os benefícios oferecidos pelos *containers* e suas tecnologias associadas (KITHULWATTA et al., 2022). Contudo, conforme a complexidade do sistema aumenta, é preciso buscar formas de organizar todos esses recursos com o uso de tecnologias que possam automatizar os processos para criação de servidores consistentes.

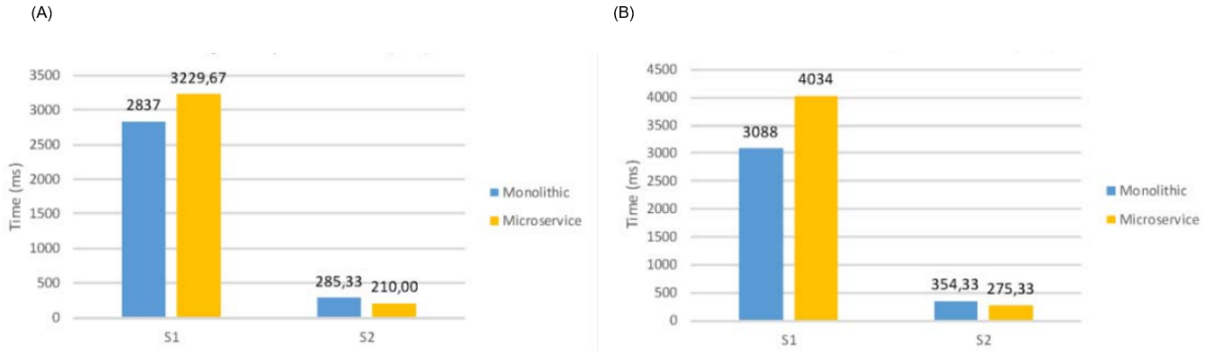
> [...] Apresentamos um padrão para um Gerenciador de containers que descreve esses componentes, juntamente com as funções necessárias desse gerenciador (SYED; FERNANDEZ, 2017).

As tecnologias usadas são os gerenciadores de ambiente dos servidores, que fornecem uma página visual e interativa dos recursos usados em um determinado servidor como, por exemplo, os recursos de hardware utilizados pelas aplicações, as imagens dos *container*, acesso aos containers e gerenciamento em geral do servidor (ZHOU et al., 2021). Existem muitas variedades de plataformas que podem ser utilizadas para gerenciar os servidores, como a ferramenta Portainer, por exemplo, uma plataforma gratuita que pode ser acessada em https://www.portainer.io, e traz todos os recursos necessários para o gerenciamento de um servidor com o uso de *container*.

5 DISCUSSÃO

Na pesquisa conduzida por Villamizar et al. (2015), em que foram realizados testes comparativos entre dois modelos de arquitetura de servidores, os resultados apresentaram a redução de custos e diferenças de escalabilidade de um servidor monolítico e outro como o uso de microsserviços. A Figura 6 apresenta o tempo de resposta médio das aplicações sob diferentes níveis de carga no gráfico (A), enquanto no gráfico (B) exibe o *throughput* (número de requisições que uma aplicação consegue suportar) das aplicações sob diferentes níveis de carga. Os resultados mostraram que a arquitetura de microserviços teve um desempenho melhor do que a abordagem monolítica em termos de tempo de resposta e *throughput*.

Figura 6 – *(A)Tempo médio de resposta (B)Tempo médio de resposta de 90%.*



Fonte: Villamizar et al. (2015)

Especificamente, ainda de acordo com Villamizar et al. (2015), a arquitetura de microserviços teve um tempo de resposta três vezes mais rápido do que a abordagem monolítica, considerando um cenário de 1000 usuários por minuto durante 10 minutos, sendo capaz de lidar com uma carga maior e com um *throughput* mais alto. Esses resultados sugerem que a arquitetura de microserviços pode ser mais eficiente e escalável do que a abordagem monolítica. Rathore e Rajavat (2022) também comparou o desempenho entre arquitetura monolítica e arquitetura de microsserviços, no qual os resultados mostraram que, em relação a produtividade, a arquitetura de microsserviços superou a monolítica em 22%, com um tempo de resposta de pico de 28% e o uso médio de CPU de cerca de 49,26% menor no uso de arquitetura com microsserviços do que o servidor de arquitetura monolítica.

Outro ponto importante a se considerar em uma infraestrutura são os custos, devido sua importância para as empresas de tecnologia. Villamizar et al. (2015), por exemplo, apresentou resultados sobre gastos entre cada tipo de arquitetura de servidor, em que foi possível obter

dados em relação à diferença de preços em cada tipo de arquitetura, como base na utilização dos mesmos serviços, aplicação e recursos de hardware. O servidor com a arquitetura de microsserviços proporcionou a redução de 17% com gastos em infraestrutura, como mostram as Tabelas 1 e 2. A grande questão dessa implementação de tipo de arquitetura de servidor é que, na pesquisa, tanto a arquitetura com microsserviços, quanto a monolítica, estavam utilizando serviços de nuvem da AWS. Perém, a diferença de configuração para o modelo de microsserviços impactou o desempenho da aplicação e os gastos como um todo.

Tabela 1 – *Custo de infraestrutura de implantação monolítica*

Serviço	Custo por hora (USD)	Quantidade por mês	Custo por mês
Aplicação Web 1 EC2 instance c4.2xlarge.	$0.464	720	$334.08
Web app1ication. 1 RDS instance db.m3.medium com Single-AZ.	$0.090	720	$64.80
Total custo por mês			**$398.88**

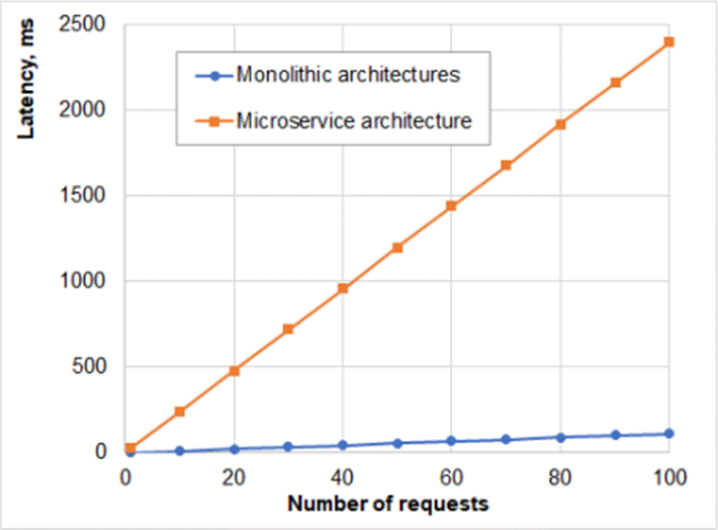
Fonte: Villamizar et al. (2015)

Tabela 2 – *Custo de infraestrutura de implantação com microserviços*

Serviço	Custo por hora (USD)	Quantidade por mês	Custo por mês
Microserviço µS1. 1 EC2 instancia c4.xlarge.	$0.232	720	$167.04
Microserviço µS1. 1 RDS instancia db.m3.medium com Single-AZ.	$0.090	720	$64.80
Microserviço µS2. 1 RDS instancia m3.medium	$0.067	720	$48.24
Gateway. 1 instancia m3.medium.	$0.067	720	$48.24
Total custo por mês			**$328.32**

Fonte: Villamizar et al. (2015)

Figura 7 – *Tempo de resposta por requisição.*

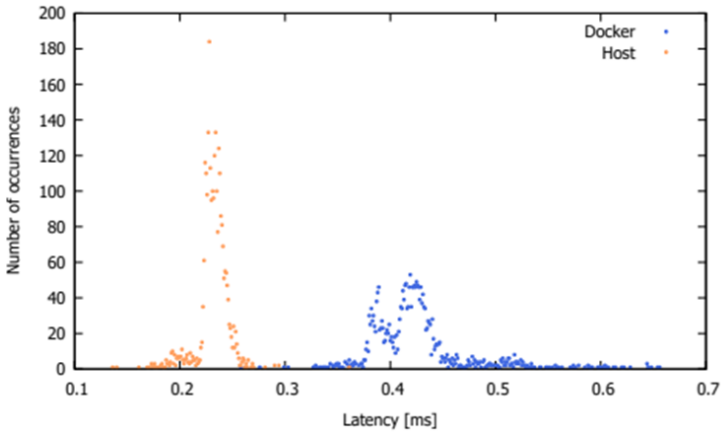


Fonte: Selivorstova et al. (2022).

Sobre a capacidade de resposta por requisição de usuários, Selivorstova et al. (2021), comparou a eficiência de resposta entre servidores monolíticos e servidores com microserviços ao testar o tempo de requisição de 100 usuários simultâneos, revelou que servidores de arquitetura monolítica teve um atraso de 24 ms em relação a servidores de microsserviços, como apresentado na Figura 7. Testar essa capacidade em um servidor em relação ao número de requisições dos usuários é de suma importância, uma vez que é possível detectar se o serviço estará disponível mesmo em momentos com muitos acessos, como, por exemplo, durante a Black Friday. Caso o contrário, uma empresa que aborda configurações antigas para os seus serviços pode comprometer a experiência dos usuários para a aplicação.

Com os benefícios de um modelo de arquitetura sobre o outro esclarecidos, é crucial compreender as discrepâncias no desempenho entre o uso de Máquinas Virtuais (VMs) e *containers*. Esta análise torna-se essencial na fase de desenvolvimento de uma aplicação, visto que pode suscitar dúvidas significativas. Um estudo realizado por Kononowicz e Czarnul (2022) observou que a sobrecarga do *Docker* em relação ao uso de máquinas virtuais em um servidor variou de 7.59% a 15,30%, como apresentado na Figura 8. Com base nisso, é possível afirmar que o uso de *containers*, em relação às máquinas virtuais, torna-se mais vantajoso, uma vez que apresenta maior leveza e não emula o sistema operacional completo, mas sim apenas o necessário para a execução de um específico serviço .

Figura 8 – *Latência de comunicação entre nós para Docker e Host..*



Fonte: Kononowics et al. (2022).

A decisão da utilização de *containers* traz benefícios sobre o desempenho do hardware, como, por exemplo, redução de custos, já que os componentes das máquinas serão bem mais aproveitados, não sendo necessário aumentar a capacidade do hardware de forma tão recorrente, Wan et al. (2018):

> [...] Ao permitir o compartilhamento no sistema operacional, bem como apoiar as bibliotecas, a virtualização baseada em *container* oferece uma grande oportunidade para reduzir o custo de implantação de aplicativos e melhorar a experiência do usuário final.

Um estudo apresentado por Zhao e Han (2019). mostrou que o *Docker* consumiu entre 10 a 15% de CPU e memória a menos em comparação às VMs, como evidenciado na Tabela 3. Essa redução de custo deve-se ao fato de que uma vez que cada parte do software terá sua virtualização, questões como: recursos de hardware, configurações de rede e dependências de bibliotecas estarão organizados separadamente, logo, trará mais eficiência de execução para o container (RATHORE; RAJAVAT, 2022).

Tabela 3 – *Teste de performace de CPU.*

Size	LDA	Align	Native(Gflops)	Docker(Gflops)	KVM(Gflops)
1000	1000	4	9.3437	9.0949	3.7868
2000	2000	4	27.1717	23.3068	11.0375
5000	5008	4	44.2333	39.2364	19.1695
10000	10003	4	44.0367	54.2655	24.2219
15000	15006	4	50.0541	48.3416	27.3552
18000	18008	4	56.7938	52.0565	21.4054
20000	20026	4	60.0415	68.2376	28.0256
22000	22048	4	61.4552	69.0096	29.0231
25000	25003	4	64.2644	64.5307	30.3508
26000	26004	4	62.4321	64.6455	30.3265
27000	27000	4	63.5444	63.7645	30.4153
30000	30007	1	64.4905	65.1906	30.4954
35000	35000	1	63.4128	63.1944	30.6245
40000	40000	1	64.1271	65.1352	30.6256
45000	45000	1	65.3508	65.3176	30.6345

Fonte: Zhao et al. (2019).

Mesmo com a utilização de *containers*, seu gerenciamento ainda é necessário, já que em um grande ambiente pode apresentar centenas de serviços simultâneos. Um dos orquestradores mais populares é o *Kubernetes*. Em um estudo recente, Dimolitsas et al. (2022). mostrou que a estrutura de escalonamento automático alcançou uma redução de 9% no consumo de energia e que, além de tornar a aplicação auto escalável, houve redução de gastos. Além disso, de acordo com Ruíz et al. (2022), um dos principais benefícios da infraestrutura baseada em *containers* é a capacidade de fornecer escalabilidade dinâmica e elasticidade rápida.

Logo, a adoção de uma infraestrutura baseada em conteinerização e orquestração se mostra promissora para lidar com a demanda atual de aplicações em constante evolução, possibilitando maior eficiência operacional (WAN et al., 2018). Assim, toda essa estrutura alinhada a um orquestrador de *containers* gera um melhor desempenho da aplicação para o usuário final.

Com base nos estudos apresentados, é possível afirmar que os servidores que adotam a abordagem de conteinerização e microsserviços estão mais preparados para lidar com o atual cenário de grande fluxo de carga e constantes mudanças em um sistema. Ao usar *containers*, os servidores podem obter maior flexibilidade e escalabilidade automática para atender às demandas

dos usuários e das empresas. Isso se deve à sua capacidade de provisionar e dimensionar recursos de forma dinâmica, permitindo que os *containers* se adaptem às variações de carga de maneira eficiente.

6 CONSIDERAÇÕES FINAIS

Em suma, a adoção de tecnologias como *Docker, Kubernetes e Portainer*, pode proporcionar uma transformação significativa na forma como as empresas configuram e gerenciam seus serviços. Essas tecnologias oferecem maior agilidade, escalabilidade e eficiência operacional, permitindo a implantação rápida e consistente de aplicações em diferentes ambientes. Além disso, a plataforma *Portainer* facilita a administração e visualização do servidor, oferecendo interfaces intuitivas e recursos avançados de gerenciamento. Com o uso de toda essas tecnologias apresentadas, as empresas podem alcançar benefícios como a redução de custos, a melhoria na colaboração entre equipes e a entrega contínua de valor aos clientes, proporcionando uma melhor qualidade na entrega de aplicações web.

Essas tecnologias impulsionam a inovação, aprimorando a eficiência operacional e permitindo uma maior flexibilidade e escalabilidade nos negócios. É essencial que as organizações estejam atentas a essas transformações, adotando abordagens modernas de gestão de serviços e aproveitando ao máximo os benefícios oferecidos por essas tecnologias. Ao compreender o impacto dessas ferramentas, as empresas estarão mais preparadas para enfrentar os desafios do mercado atual e impulsionar o crescimento e o sucesso de seus negócios.

Portanto, este estudo teve como objetivo contribuir para uma visão aprimorada sobre a evolução da arquitetura de servidores e dos benefícios resultantes dessa transformação, tais como a agilidade no desenvolvimento, a otimização do aproveitamento do hardware e a flexibilidade no escalonamento de recursos para aplicações. Ao aprofundar a compreensão nesse campo, buscou-se fornecer informações que permitiram apreciar o impacto dessas mudanças na infraestrutura tecnológica e no desenvolvimento de soluções computacionais. Portanto, considerando todo o percurso da evolução de arquitetura de servidores, desde as fazes iniciais com a arquitetura monolítica, até a solução com o uso de de VMs para o encapsulamento de serviços e posteriormente a criação de containers para diminuir a sobrecarga de processamento de hardware, até as formas mais complexas de gerenciamento com orquestradores Kubernes e plataformas visuais de gerenciamento de ambiente de infraestrutura como portainer. Por meio dessa análise evolutiva, é possível promover uma compreensão mais abrangente dos benefícios e das oportunidades proporcionadas pela evolução na arquitetura de servidores, e assim, contribuindo para o avanço do conhecimento da área.

REFERÊNCIAS

ABBOTT, M. L.; FISHER, M. T. **The art of scalability: Scalable web architecture, processes, and organizations for the modern enterprise**. [S.l.]: Addison-Wesley Professional, 2015.

BIGGS, S.; LEE, D.; HEISER, G. The jury is in: Monolithic os design is flawed: Microkernel-based designs improve security. In: **Proceedings of the 9th Asia-Pacific Workshop on Systems**. [S.l.: s.n.], 2018. p. 1–7.

CHUNG, M. T. et al. Using docker in high performance computing applications. In: IEEE. **2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)**. [S.l.], 2016. p. 52–57.

CORRY, F. Why does a platform die? diagnosing platform death at friendster's end. **Internet Histories**, Taylor & Francis, v. 6, n. 1-2, p. 31–47, 2022.

DIAS, A. H.; CORREIA, L. H.; MALHEIROS, N. A systematic literature review on virtual machine consolidation. **ACM Computing Surveys (CSUR)**, ACM New York, NY, v. 54, n. 8, p. 1–38, 2021.

DIMOLITSAS, I. et al. Ahp4hpa: An ahp-based autoscaling framework for kubernetes clusters at the network edge. In: IEEE. **GLOBECOM 2022-2022 IEEE Global Communications Conference**. [S.l.], 2022. p. 2566–2571.

FAN, C.-Y.; MA, S.-P. Migrating monolithic mobile application to microservice architecture: An experiment report. In: IEEE. **2017 ieee international conference on ai & mobile services (aims)**. [S.l.], 2017. p. 109–112.

GOLI, A. et al. Migrating from monolithic to serverless: A fintech case study. In: **Companion of the ACM/SPEC International Conference on Performance Engineering**. [S.l.: s.n.], 2020. p. 20–25.

GOS, K.; ZABIEROWSKI, W. The comparison of microservice and monolithic architecture. In: IEEE. **2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)**. [S.l.], 2020. p. 150–153.

HAFNER, K.; LYON, M. **Where wizards stay up late: The origins of the Internet**. [S.l.]: Simon and Schuster, 1998.

JARAMILLO, D.; NGUYEN, D. V.; SMART, R. Leveraging microservices architecture by using docker technology. In: IEEE. **SoutheastCon 2016**. [S.l.], 2016. p. 1–5.

KITHULWATTA, W. et al. Docker containerized infrastructure orchestration with portainer container-native approach. In: IEEE. **2022 3rd International Conference for Emerging Technology (INCET)**. [S.l.], 2022. p. 1–6.

KONONOWICZ, T.; CZARNUL, P. Performance assessment of using docker for selected mpi applications in a parallel environment based on commodity hardware. **Applied Sciences**, MDPI, v. 12, n. 16, p. 8305, 2022.

KUMAR, M.; KAUR, G. An empirical study of containerized mpi and gui application on hpc in the cloud. In: IEEE. **2022 2nd International Conference on Innovative Sustainable Computational Technologies (CISCT)**. [S.l.], 2022. p. 1–6.

MAHDAVI-HEZAVEH, R.; DREMANN, J.; WILLIAMS, L. Software development with feature toggles: practices used by practitioners. **Empirical Software Engineering**, Springer, v. 26, p. 1–33, 2021.

MENOUER, T. Kcss: Kubernetes container scheduling strategy. **The Journal of Supercomputing**, Springer, v. 77, n. 5, p. 4267–4293, 2021.

MIELL, I.; SAYERS, A. **Docker in practice**. [S.l.]: Simon and Schuster, 2019.

MOUAT, A. **Using Docker: Developing and deploying software with containers**. [S.l.]: "O'Reilly Media, Inc.", 2015.

NGUYEN, T.-T. et al. Horizontal pod autoscaling in kubernetes for elastic container orchestration. **Sensors**, MDPI, v. 20, n. 16, p. 4621, 2020.

NISSENBAUM, H. How computer systems embody values. **Computer**, IEEE, v. 34, n. 3, p. 120–119, 2001.

PONISZEWSKA-MARAŃDA, A.; CZECHOWSKA, E. Kubernetes cluster for automating software production environment. **Sensors**, MDPI, v. 21, n. 5, p. 1910, 2021.

POTDAR, A. M. et al. Performance evaluation of docker container and virtual machine. **Procedia Computer Science**, Elsevier, v. 171, p. 1419–1428, 2020.

RANDAL, A. The ideal versus the real: Revisiting the history of virtual machines and containers. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 53, n. 1, p. 1–31, 2020.

RATHORE, N.; RAJAVAT, A. Evaluating the performance of monolithic and microservices architectures in an edge computing environment. **International Journal of Fog Computing (IJFC)**, IGI Global, v. 5, n. 1, p. 1–18, 2022.

RENSIN, D. **Kubernetes**. [S.l.]: O'Reilly Media, Incorporated, 2015.

RUÍZ, L. M. et al. Autoscaling pods on an on-premise kubernetes infrastructure qos-aware. **IEEE Access**, IEEE, v. 10, p. 33083–33094, 2022.

SANTOS, E. A. et al. How does docker affect energy consumption? evaluating workloads in and out of docker containers. **Journal of Systems and Software**, Elsevier, v. 146, p. 14–25, 2018.

SELIVORSTOVA, T. V. et al. Analysis of monolithic and microservice architectures features and metrics. , , 2021.

SINGH, S.; SINGH, N. Containers & docker: Emerging roles & future of cloud technology. In: IEEE. **2016 2nd international conference on applied and theoretical computing and communication technology (iCATccT)**. [S.l.], 2016. p. 804–807.

SRIVASTAVA, M. K.; JOSHI, V. K. Efficient consolidation of vms systems in the cloud to reduce energy use. In: IEEE. **2023 10th International Conference on Computing for Sustainable Global Development (INDIACom)**. [S.l.], 2023. p. 1510–1514.

SYED, M. H.; FERNANDEZ, E. B. The container manager pattern. In: **Proceedings of the 22nd European Conference on Pattern Languages of Programs**. [S.l.: s.n.], 2017. p. 1–9.

THURGOOD, B.; LENNON, R. G. Cloud computing with kubernetes cluster elastic scaling. In: **Proceedings of the 3rd International Conference on Future Networks and Distributed Systems**. [S.l.: s.n.], 2019. p. 1–7.

VILLAMIZAR, M. et al. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In: IEEE. **2015 10th Computing Colombian Conference (10CCC)**. [S.l.], 2015. p. 583–590.

WAN, X. et al. Application deployment using microservice and docker containers: Framework and optimization. **Journal of Network and Computer Applications**, Elsevier, v. 119, p. 97–109, 2018.

ZHAO, H.; HAN, Z.-y. Application of docker container in intelligent traffic cloud. In: SPRINGER. **Green Intelligent Transportation Systems: Proceedings of the 8th International Conference on Green Intelligent Transportation Systems and Safety**. [S.l.], 2019. p. 635–643.

ZHOU, N. et al. Container orchestration on hpc systems through kubernetes. **Journal of Cloud Computing**, SpringerOpen, v. 10, n. 1, p. 1–14, 2021.

INSTITUTO FEDERAL
Goiano

Repositório Institucional do IF Goiano - RIIF Goiano
Sistema Integrado de Bibliotecas

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR PRODUÇÕES TÉCNICO-CIENTÍFICAS NO REPOSITÓRIO INSTITUCIONAL DO IF GOIANO

Com base no disposto na Lei Federal nº 9.610/98, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia Goiano, a disponibilizar gratuitamente o documento no Repositório Institucional do IF Goiano (RIIF Goiano), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, em formato digital para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IF Goiano.

Identificação da Produção Técnico-Científica

[ ] Tese	[ ] Artigo Científico
[ ] Dissertação	[ ] Capítulo de Livro
[ ] Monografia – Especialização	[ ] Livro
[x] TCC - Graduação	[ ] Trabalho Apresentado em Evento

[ ] Produto Técnico e Educacional - Tipo: _____

Nome Completo do Autor: Emelly Marinho Pereira Silva
Matrícula: 2019103202030121
Título do Trabalho: A evolução da arquitetura de servidores: comparação entre arquitetura monolítica e microsserviços

Restrições de Acesso ao Documento

Documento confidencial: [x] Não [ ] Sim, justifique: _____

_____

Informe a data que poderá ser disponibilizado no RIIF Goiano: 06/12/23
O documento está sujeito a registro de patente? [ ] Sim [x] Não
O documento pode vir a ser publicado como livro? [ ] Sim [x] Não

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O/A referido/a autor/a declara que:

1. o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;
2. obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autor/a, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia Goiano os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;
3. cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia Goiano.

Ceres - Goiás , 06/12/23 .
Local Data

Emelly Marinho
_____
Assinatura do Autor e/ou Detentor dos Direitos Autorais

Ciente e de acordo:

Kaitan Campos Gonçalves
_____
Assinatura do(a) orientador(a)

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA GOIANO

ATA DE DEFESA DE TRABALHO DE CURSO

Ao(s) 20 dia(s) do mês de novembro do ano de dois mil e 2023, realizou-se a defesa de Trabalho de Curso do(a) acadêmico(a) EMELLY MARINHO PEREIRA SILVA, do Curso Bacharelado em Sistemas de Informações, matrícula 2019103202030121, cujo título é A EVOLUÇÃO DA ARQUITETURA DE SERVIDORES: COMPARAÇÃO ENTRE ARQUITETURA MONOLÍTICA E MICROSSERVIÇOS. A defesa iniciou-se às 19h horas e 20 minutos, finalizando-se às 21 horas e 15 minutos. A banca examinadora considerou o trabalho **APROVADO** com média **8,9** no trabalho escrito, média **9,9** no trabalho oral, apresentando assim média aritmética final de **9,4** pontos, estando o(a) estudante **APTO** para fins de conclusão do Trabalho de Curso.

Após atender às considerações da banca e respeitando o prazo disposto em calendário acadêmico, o(a) estudante deverá fazer a submissão da versão corrigida em formato digital (.pdf) no Repositório Institucional do IF Goiano – RIIF, acompanhado do Termo Ciência e Autorização Eletrônico (TCAE), devidamente assinado pelo autor e orientador.

Os integrantes da banca examinadora assinam a presente.

(Assinado Eletronicamente)
Roitier Campos Gonçalves

(Assinado Eletronicamente)
Hugo de Moura Campos

(Assinado Eletronicamente)
Rangel Rigo

Documento assinado eletronicamente por:

- Rangel Rigo, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 04/12/2023 16:00:54.
- Hugo de Moura Campos, TECNICO DE LABORATORIO AREA, em 04/12/2023 11:16:55.
- Roitier Campos Goncalves, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 04/12/2023 11:06:16.

Este documento foi emitido pelo SUAP em 04/12/2023. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifgoiano.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 553254
Código de Autenticação: 068fb666a3