UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Lucas Cerutti Sergio

Utilização de Microsserviços em Ambientes de Desenvolvimento

Uberlândia, Brasil

2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Lucas Cerutti Sergio

Utilização de Microsserviços em Ambientes de Desenvolvimento

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Professor Ronaldo Castro de Oliveira

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2024

Agradecimentos

Gostaria de agradecer aos meus pais Carla e Laércio por todo o apoio, confiança, tempo e ensinamentos de vida, sem vocês eu não seria capaz de chegar aonde estou, vocês são meu norte, meu mundo, minha vida. Um dia, espero ser capaz de retribuir tudo que me forneceram. A minha namorada Bruna por estar sempre ao meu lado em tudo, por me apoiar em todas as minhas decisões, por ser minha inspiração e por estar presente em todas as minhas conquistas. Sem você, eu jamais sonharia em estar aqui, você foi minha motivação em continuar e não largar meu sonho, obrigado por tudo. Gostaria também de agradecer a todos meus amigos e familiares que sempre me incentivam, que me acompanham em toda jornada, fazem parte da minha construção como pessoa, que estão sempre à disposição e torcendo para meu sucesso, em especial minhas irmãs Eduarda e Marina. Por fim, expresso minha sincera gratidão ao meu orientador Ronaldo Oliveira pela sua excepcional habilidade como educador e pelo apoio que me foi oferecido para conclusão deste projeto.

"Não há mentira pior do que uma verdade mal compreendida por aqueles que a ouvem." - Henry James

Resumo

Com o crescimento contínuo no desenvolvimento de software e os investimentos em novas tecnologias, bem como a adoção de métodos ágeis e a melhoria nos serviços de entrega, emerge a arquitetura de microsserviços. Essa estrutura utiliza ferramentas de gerenciamento, capazes de realizar a orquestração de ambientes *clusters*. Com meta de construir uma estrutura *cluster* Kubernetes e pesquisas sobre arquitetura monolítica, é possível ter uma percepção aprofundada e fornecer informações relevantes sobre a temática. Além de trazer conceitos sobre o que é um servidor, virtualização, contêineres, ferramentas de gerenciamento, aprofundar nos conceitos de Kubernetes e entender sobre a arquitetura tradicional monolítica. Após toda a construção do ambiente de microsserviço e observar as comparações com a arquitetura monolítica, é possível trazer informações valiosas como aspectos de escalabilidade, alta disponibilidade, utilização de contêineres, implementações e a gerência de cada arquitetura. Fornecendo um balanço de qual modelo arquitetônico é mais eficientes em determinada situação, proporcionando aos desenvolvedores entender sobre a melhor arquitetura para seus sistemas e projetos.

Palavras-chave: *Cluster* Kubernetes, Utilização de microsserviços, Arquitetura monolítica e microsserviços, Gerenciamento de contêineres, Arquitetura Kubernetes.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
CPU	*Central Processing Unit*
DCMI	*Data Center Infrastructure Monitoring*
DNS	Sistema de Nomes de Domínio
GB	*Gigabyte*
HD	*Hard Disk*
HPA	*Horizontal Pod Autoscale*
SO	Sistema Operacional
TCC	Trabalho de Conclusão de Curso
TI	Tecnologia da Informação
VM	*Virtual Machine*

Sumário

1 Introdução

Nos tempos atuais, é possível perceber de forma significativa o crescimento de desenvolvimento de softwares e empresas investindo cada vez mais em tecnologias novas, implementações de sistemas inovadores, ambientes mais integrados e estáveis. O mundo atual está mais conectado, em que tudo envolve maior praticidade e agilidade e, ao se tratar do desenvolvimento de um software, não é diferente (FOWLER, 2019). Sob essa perspectiva, precisa-se de rapidez na construção de um software, facilidade na manutenção, boa compreensão e também de um alto serviço de entrega. Ao construir um ambiente que possa atender as necessidades de mercado, é importante conhecer algumas arquiteturas, tais como a monolítica e microsserviços, pois são alternativas que podem solucionar as exigências do cenário atual do mercado de tecnologias. Portanto, neste trabalho, serão investigados os benefícios, os desafios e as melhores práticas ao se utilizar um modelo arquitetônico.

Adotar uma arquitetura em um projeto de software requer inúmeras observações, assim como também é importante entende-las. Embora não seja um conceito novo, mas em alta nos últimos anos, se têm a arquitetura de microsserviços, que consiste em uma estrutura software onde são moldados uma aplicação e outros componentes independentes, sendo que cada um deles torna-se capaz de realizar atividades específicas, o que permitem serem alterados de forma independente, diferenciando do modelo tradicional de arquitetura monolítica, no qual sua implementação de códigos vem de uma única base, ou seja, suas alterações são feitas de forma conjunta (JUSTINO, 2018). Assim, ao utilizar o microsserviço, a aplicação pode ser desenvolvida de maneira modular, ou seja, em blocos de subsistemas, o que proporciona uma flexibilidade e escalabilidade para o ambiente, principalmente se comparado ao modelo monólito (AWS, 2023).

A estrutura de microsserviços em ambientes de desenvolvimento oferece uma série de vantagens, o que facilita a implementação de novos recursos, o isolamento de falhas, testes e as atualizações mais ágeis. Além disso, a utilização desse tipo de serviço baseia-se no conceito de DevOps, uma prática ágil que permite desenvolvimento em ciclos mais curtos e com respostas mais rápidas em relação às mudanças de requisitos de desenvolvimento, sendo uma área em grande crescimento e promovendo mais relevância à utilização de microsserviços (JUSTINO, 2018); (REDHAT, 2023). Dessa forma, por mais que a utilização de microsserviços seja um facilitador, têm suas desvantagens ao carregar consigo uma série de desafios ao desenvolver essa estrutura, como gerenciar sua complexidade de vários serviços, manter seus dados de maneira sólida, lidar com indisponibilidades de serviços e infraestrutura complexa (MENDES, 2021). Sendo assim, pode promover questionamentos aos profissionais e estudantes da área de tecnologia da informação se ambientes com

arquitetura monolítica valem a pena migrar para utilização de microsserviços.

Portanto, o trabalho tem como intuito construir um pequeno ambiente de desenvolvimento utilizando microsserviços, analisar o estudo de caso, mostrar uma implementação bem sucedida, trazer seus desafios, vantagens, desvantagens em relação à arquitetura monolítica. Ao fazer isso, tem-se como base trazer informações relevantes para empresas e profissionais da área de tecnologia da informação (TI) que querem implementar o modelo de microsserviço em seus projetos.

1.1 Objetivos

O objetivo deste trabalho é fornecer uma estrutura e pesquisas a respeito da utilização de microsserviços em ambientes de desenvolvimento, podendo ser comparado a outros modelos de arquitetura e trazer informações sobre a temática.

1.1.1 Objetivos específicos

Com base no objetivo geral, tem-se os objetivos específicos que se pretendem alcançar com este trabalho, elencados abaixo:

- **Analisar exemplos reais e estudos de caso**: Analisar exemplos de uso e casos práticos relevantes, a fim de identificar facilidades e dificuldades enfrentadas na implementação de uma nova arquitetura.
- **Analisar os principais ganhos proporcionados pelos microsserviços**: Observar e avaliar as principais vantagens que os microsserviços proporcionam em ambientes de desenvolvimento, como escalabilidade, alta disponibilidade e monitoramento de recursos.
- **Analisar dificuldades na implementação de microsserviços:** Explorar os problemas enfrentados ao optar por microsserviços, abordando temas como custos indiretos e recursos necessários para sua construção.
- **Contribuir com conhecimento:** Com o compartilhamento de conhecimento na área, fornecendo conclusões, recomendações, benefícios e desvantagens.

Para ter um melhor resultado dos objetivos, foi utilizado manuais oficiais na construção do projeto afim de corroborar com o melhor entendimento e enriquecer as informações obtidas para uma compreensão mais detalhada sobre a utilização de microsserviços com Kubernetes. Os objetivos para serem alçados contaram de forma significativa com os trabalhos de Dragoni et al. (2017), Freire (2021), Lima e Santos (2019), Lopes (2021), Souza et al. (2023) e com as documentações da Canonical Ltd (2024) e The Kubernetes Authors (2023).

1.2 Justificativa

Este Trabalho de Conclusão de Curso (TCC) tem como justificativa o aumento significativo da utilização da arquitetura de microsserviços em ambientes de desenvolvimento de softwares nas empresas (NADAREISHVILI et al., 2016). Logo, torna-se importante a realização deste estudo para melhor compreensão dos benefícios, desafios e práticas baseadas na sua utilização, trazendo dados relevantes para os profissionais de TI e às empresas que buscam aderir a essa arquitetura e, assim, não se prender a sistemas legados que utilizam arquitetura monolítica.

Dessa maneira, foram listadas abaixo algumas razões que justificam a realização do tema:

1. Relevância do tema: Alto destaque nos tempos atuais, sendo uma alternativa mais atrativa com relação ao tradicional modelo de monólitos, principalmente ao falar sobre agilidade, escalabilidade e manutenção.

2. Contribuição de conhecimento: Este trabalho pretende expandir os conhecimentos sobre o tema, detalhar melhor o entendimento na área de arquitetura de software e trazer as vantagens e desvantagens de utilizar microsserviços.

3. Alta demanda: Com o aumento de sistemas que utilizam sistemas distribuídos, além da alta escalabilidade, as empresas estão à procura da implementação dos microsserviços, com mais ênfase para compreender o uso dessa arquitetura.

Em resumo, exemplificar aos profissionais de TI e para as empresas a importância da utilização de microsserviços em ambientes de desenvolvimento, além de propagar conhecimento para o meio acadêmico e profissional.

1.3 Metodologia

O estudo foi construído com base nos conhecimentos adquiridos a partir da busca ativa de referenciais teóricos atualizados sobre o assunto abordado, que foram obtidos através de uma revisão de literatura em bases de dados acadêmicas como Google Acadêmico e o Portal de periódicos da Capes. Além de fontes de conhecimento fornecido pelas empresas *Amazon* e *Red Hat*.

A partir dessa primeira etapa, foi realizado um estudo de caso em um pequeno ambiente de desenvolvimento utilizando microsserviços. O estudo de caso incluiu várias etapas: construção de máquinas virtuais, instalação do Kubernetes, habilitação de uma aplicação E-commerce, habilitação do DNS, configuração do Load Balance e estruturação de um ambiente de *clusters*. Este ambiente permite alta disponibilidade e escalabilidade,

demonstrando a distribuição de carga em tempo real com base no uso do processamento, identificado como Unidade Central de Processamento (CPU).

1.4 Organização do Trabalho

- **Capítulo 2 - Referencial Teórico**: apresenta fundamentos teóricos que corroboram para a melhor compreensão do trabalho, com a inclusão de tecnologias como Kubernetes, máquinas virtuais,contêineres e ambiente *cluster*. Além do mais, demonstra como funciona um servidor, esclarece a definição de virtualização, microsserviços e arquitetura monolítica. Por fim, retrata os trabalhos correlatos que serviram de inspiração e base para construção desse TCC.
- **Capítulo 3 - Criação do ambiente**: Descreve o processo de criação das máquinas virtuais e de configurações do Kubernetes. Exibe a aplicação utilizada para o teste, o balanceamento de carga, a alta disponibilidade em funcionamento e a alta escalabilidade estruturada. Por conseguinte, compara a arquitetura monolítica com a de microsserviço.
- **Capítulo 4 - Conclusões**: Este tópico traz os benéficos de se utilizar a arquitetura de microsserviço,s mas sem descartar a existência da arquitetura monolítica para determinados projetos.

2 Referencial Teórico

Neste capítulo, será explorado o estudo dedicado ao tema exibindo todo o conhecimento adquirido ao longo da pesquisa.

2.1 Conceitos Fundamentais

Nesta seção, foram abordados todos os conhecimentos gerais obtidos que foram fundamentais para o desenvolvimento do projeto, trazendo conhecimentos sobre o que é um servidor e um pouco dos seus conceitos, máquinas virtuais (*virtual machine - VM*), *hypervisor*, microsserviços, gerenciamento de contêineres, no qual terá foco sobre a ferramenta Kubernetes, escalabilidade horizontal e automação de infraestrutura. Logo em seguida, serão apresentados trabalhos correlatos.

2.1.1 Servidor

O servidor corresponde a uma máquina ou um programa, que tem como compromisso a centralização de atividades em relação a uma rede de computadores (clientes) (CACIATO, 2009). Um servidor pode ser "responsável por fornecer "serviços" aos clientes, tais como aplicações, recursos e arquivos (SOUZA, 2006). Possui um hardware mais robusto, mais potente que um computador comum. No que tange a sua importância, são criados para suportar uma maior carga de trabalho, utilizando o máximo do hardware disponível a fim de evitar falhas de desempenho (SOUZA, 2006).

2.1.2 Servidores físicos

O servidor físico pode ser conceituado como um dispositivo que possui todos os softwares, assim como sistemas operacionais necessários para uma organização. Sendo compreendido como o elo que executa a conexão entre inúmeros computadores. Conexão essa que possibilita que diferentes indivíduos possam ter acesso a determinados arquivos, bem como a inúmeras informações inseridas nos vários mecanismos que estão ligados ao servidor físico (BALTIERI, 2017). Para facilitar o entendimento, os servidores são fundamentais para uma organização, são eles que fornecem os recursos para que as aplicações funcionem, proveem algo para alguém/cliente.

Quando se fala em servidor físico, significa que ele pode ser palpável, há um hardware realizando as operações, consequentemente, há consumo de energia, necessidade de equipamentos de segurança, prevenção contra incêndios, refrigeração, pisos elevados, *racks* para suporte, controle de acesso, sensores de fumaça, analistas especializados responsáveis,

entre outros aspectos, conhecido como *DCMI- Data Center Infrastructure Monitoring*, ou seja, ficam em locais especializados (BARROSO; HöLZLE, 2013). Na figura 1, é apresentado um modelo de Data Center, onde se localiza um servidor físico.

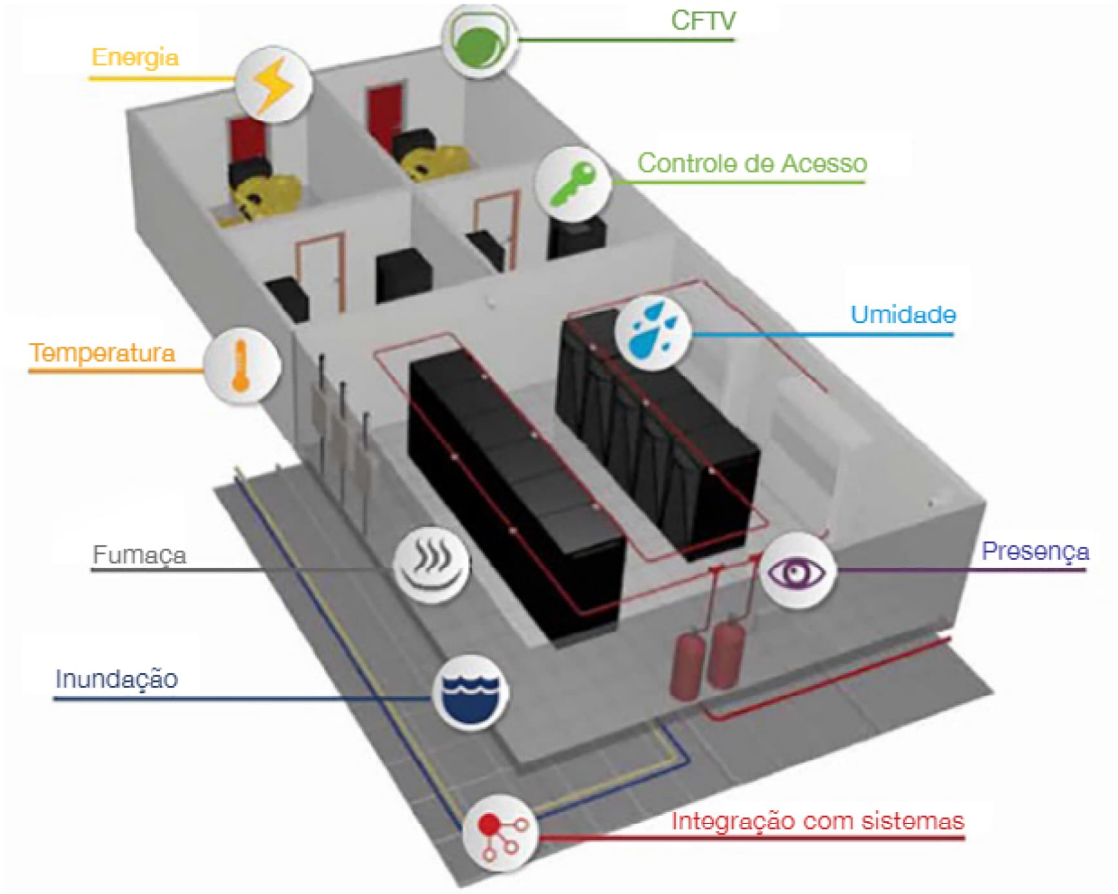


Figura 1 – Modelo de monitoramento em um sistema DCIM. Fonte: Extraída de (FILHO, 2016).

2.1.3 Conceito de virtualização

A virtualização corresponde a um processo de combinações de software e hardware, tendo como propósito a criação de dispositivos ou ambientes virtuais, concentrando em um único equipamento físico (FABIAN et al., 2006). De maneira geral, as máquinas virtuais disponibilizam serviços similares a uma máquina física. No entanto, Figueredo e Lucca (2017) ressaltam que:

> Máquinas ou ambientes virtuais só existem logicamente [...]. Com isso, é possível executar diversas máquinas sobre um mesmo hardware, o que aumenta o aproveitamento computacional e, consequentemente, diminui o consumo energético e infraestrutura(FIGUEREDO; LUCCA, 2017).

De acordo com NATÁRIO (2011), a virtualização é vista como um método que permite a configuração e execução de diferentes programas, serviços, assim como sistemas operacionais distintos em apenas um único dispositivo físico. O hardware, por exemplo, pode ser simulado em um único componente, dentre eles: roteadores, servidores e até mesmo *switches*. Em outras palavras, a virtualização pode ser conceituada como a estruturação de um ambiente virtual que presume um ambiente real. "Em uma linguagem mais técnica, é a abstração de uma camada física em múltiplas divisões lógicas"(CARISSIMI, 2008).

2.1.4 Máquina virtual

A máquina virtual (*Virtual Machine – VM*) pode ser observada como um "computador dentro de outro", isto é, um software de simulação inserido dentro de outro dispositivo real, que, por sua vez, gera uma máquina simulada de forma virtual, até mesmo o processo, as memórias, entre outros recursos, atuam de forma virtual. Vale ressaltar que esse processo tem início após a instalação de um programa de simulação no computador físico, possibilitando que o sistema seja executado para operar as atividades que seriam feitas se fosse máquina física (CARISSIMI, 2008).

2.1.5 Tipos de virtualização

Inicialmente, vale destacar que a "Virtualização" concentra no pensamento em que diferentes sistemas rodam em um mesmo dispositivo, no entanto, esse é somente um tipo de virtualização. No geral, a virtualização de hardware destaca-se como a mais conhecida, mesmo que não seja a única (ROCHA, 2013). A tabela 1 apresenta os três modelos mais conhecidos.

Virtualização de desktops	É um conceito parecido com o de virtualização de servidores, ou seja, vários desktops virtuais são executados em um único servidor físico. Neste tipo de virtualização, os usuários não executam seus aplicativos e sistemas operacionais localmente. É adotado o modelo cliente-servidor, no qual todas as aplicações, processos e dados são mantidos de forma centralizada.
Virtualização de aplicações	É uma forma de virtualização que permite que aplicações sejam executadas, em máquina local ou virtual, sem serem instaladas no dispositivo, ou seja, a aplicação roda centralizada e sem a necessidade de atualização e manutenção em diversos pontos.
Virtualização de servidores (Hypervisor)	É o principal tipo de virtualização, o mais utilizado e comum entre os três. Nesse modo de virtualização um sistema operacional funcional (Hypervisor) é configurado sobre o hardware, substituindo completamente o sistema operacional padrão (Windows Server, Linux) e funcionando como uma camada de virtualização, proporcionando, através da sua tecnologia, a criação de várias máquinas virtuais.

Tabela 1 – Tipos de Virtualização. Fonte: Extraída de (FIGUEREDO; LUCCA, 2017);(CARISSIMI, 2008)adaptado.

2.1.6 Servidores virtuais

Quando se fala em servidor virtual, o mesmo corresponde a um meio decorrente de novas tecnologias que abrange o processo de virtualização. Tendo como princípio um modelo de dispositivo virtual que disponibiliza uma grande capacidade de processamento, assim como espaço de armazenamento e um sistema robusto para o sistema operacional. Os servidores virtuais têm como característica operar no ambiente de um servidor físico. Somado a isso, ele possui um meio de funcionamento através de uma base de TI (PICCINI, 2020).

2.1.6.1 Funcionamento do servidor virtual

O processo que envolveu o funcionamento do servidor virtual assemelha-se muito ao servidor físico. Com uma grande disponibilidade de recursos para softwares, assim como aplicações, além de configurações e *scripts*. No que se refere ao servidor virtual, o mesmo possui recursos focados em virtualizar o hardware, isto é, seu objetivo é promover a separação de recursos e criar servidores virtuais variados (FABIAN et al., 2006).

Logo após ocorrer a separação de recursos, o sistema passa exercer suas atribuições de forma independente, não apresentando qualquer ligação entre si. Através dessas aplicações, os servidores virtuais possibilitam que inúmeras plataformas, assim como sistema operacional, funcionem (executem) de forma simultânea, englobando um único hardware, de maneira independente (PICCINI, 2020).

Em relação a vantagens oferecidas pelo servidor virtual, a maior está associada à liberdade disponibilizada aos clientes. Com isso, quem adquire esse serviço tem como recursos controlá-lo de acordo com suas preferências. Outro benefício está relacionado à possibilidade de instalar e configurar inúmeros programas, tais como: banco de dados, *scripts*, além de outros sistemas (PICCINI, 2020).

2.1.7 Microsserviços

Microsserviços são um tipo de arquitetura de software amplamente utilizada para construir aplicações. Nos últimos anos, essa abordagem ganhou popularidade, especialmente entre os programadores, devido à sua flexibilidade, que possibilita a utilização de diferentes linguagens, tecnologias e reutilização de códigos. Isso representa uma mudança significativa em relação ao antigo método de desenvolvimento de software monolítico. Dessa forma, quando há uma escolha por microsserviços, percebe-se que cada parte que abrange o software apresenta uma conectividade efetiva, podendo evitar que qualquer tipo de erro mude partes de determinado programa, que, por sua vez, causaria uma instabilidade englobando todo o sistema (FOWLER, 2019).

2.1.7.1 Contêineres

Consistem em uma tecnologia que vem transformando a maneira como são executadas as operações de TI. Entende-se que os contêineres são blocos de espaços que podem ser divididos por uma ferramenta, exemplo Docker ou Kubernetes, que podem compartilhar o mesmo sistema operacional (VILLAÇA; JR; AZEVEDO, 2018).

Abaixo, podemos notar a arquitetura de uma máquina virtual[2] e um contêiner[3]:

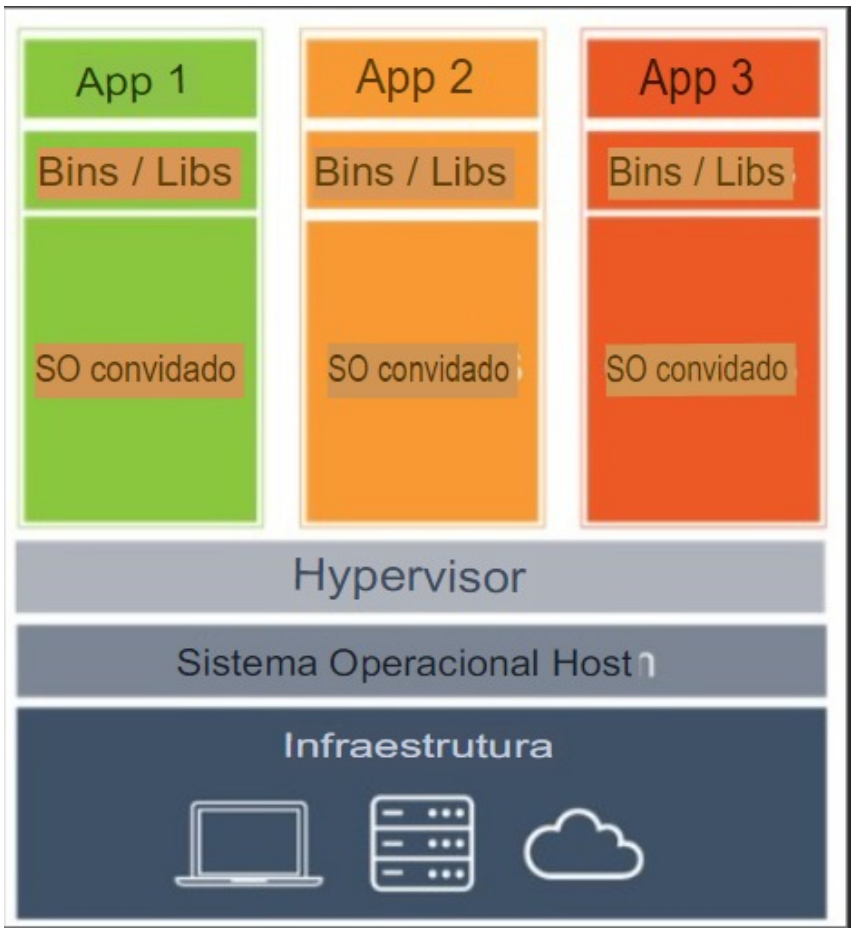


Figura 2 – Arquitetura de uma máquina virtual - Traduzida. Fonte extraída de (SANTHOSH, 2017).

Na Figura 2, tem-se uma infraestrutura física e um sistema operacional executando neste servidor.Este servidor possui um *hypervisor* responsável por gerenciar as máquinas convidadas, chamadas de VM. Os sistemas operacionais de cada VM são individuais, possuindo suas próprias bibliotecas, aplicativos e arquivos.

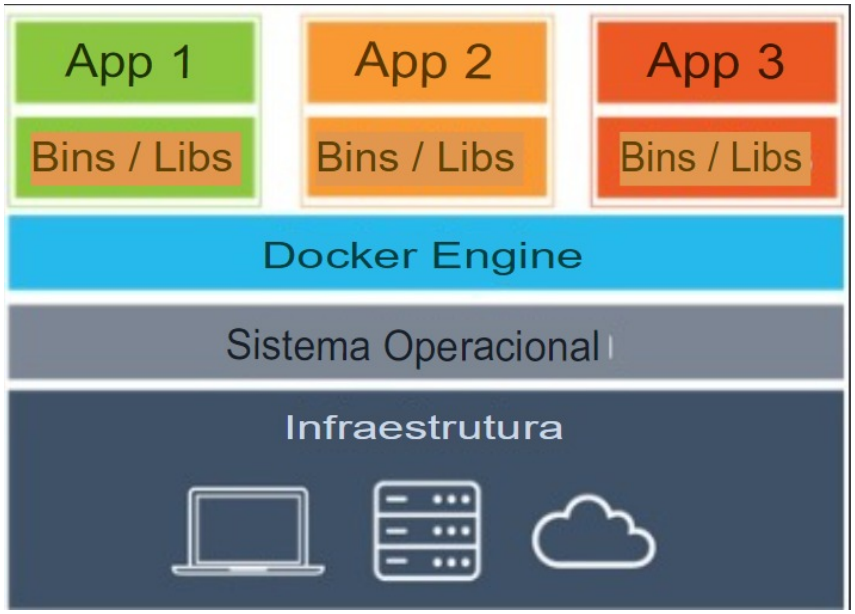


Figura 3 – Arquitetura de um Contêiner - Traduzida. Fonte extraída de (SANTHOSH, 2017).

A presente abordagem (Figura 3) destaca como vantagem o uso de menos recursos computacionais, tendo em vista que cada contêiner não possui o próprio sistema operacional, responsabilidade do Docker Engine fornece-los. Com isso, as ações são vistas como mais eficientes em termos de recursos e, somado a isso, oferecem portabilidade, com a capacidade de executar em uma instância na nuvem ou até mesmo em um serviço específico (LIMA, 2022).

De acordo com Harada (2021), os principais benefícios referentes a microsserviços e contêineres são:

- **Menor necessidade de recursos**: As aplicações são mais leves por não haver um sistema operacional dedicado, o que permite que sejam capazes de executar em serviços de nuvem ou servidores dedicados.
- **Independência entre aplicativos e infraestrutura**: Seus serviços são encapsulados de forma individual, podendo focar em atividades de forma separada, adaptando facilmente a mudanças.
- **Otimização dos processos com foco nas aplicações**: Isso permite que os desenvolvedores possam trabalhar de forma mais eficiente, escalável e ágil.
- **Criação de módulos reutilizáveis**: Os microsserviços são criados para realizar uma função única. Cada serviço comunica-se por interfaces, podendo ser alocado em diferentes partes do sistema, o que permite que o módulo também possa ser compartilhado para outros projetos.

2.1.8 Gerenciamento de Contêineres

Atualmente, ferramentas como Kubernetes, Docker Swarm e Apache Mesos são responsáveis por auxiliar os desenvolvedores a gerenciar com contêineres, capazes de organizar os ambientes que utilizam microsserviços, tornam as implementações mais fáceis, assim como também são capazes de realizar escalonamento e automação de atividades(Red Hat, 2019).

2.2 Ferramentas de Virtualização

É crucial familiarizar-se com as ferramentas empregadas na criação de um ambiente virtual. Embora todas tenham o objetivo comum de gerenciar uma arquitetura de microsserviços, elas possuem suas próprias otimizações e formas de aproveitar os recursos de hardware do *host*. A seguir, apresentam-se ferramentas que podem ser utilizadas para a virtualização.

2.2.1 *Hypervisor*

De acordo com Rocha (2013), o Monitor de máquina Virtual (*Virtual Machine Monitor - VMM*) consiste em um software que está localizado na camada que envolve o hardware físico e os dispositivos virtuais. Sua principal função é gerenciar os recursos que envolvem os sistemas operacionais das máquinas virtuais aos mecanismos de hardware da máquina central, ou seja, os recursos como CPU, memória e armazenamento de cada VM são cedidos pela máquina física. É importante ressaltar que cada mecanismo virtual apresenta os seus próprios recursos, dessa forma, o papel do *hypervisor* é administrar os acessos referentes a cada máquina virtual, assim como os recursos do *host*, dividindo os recursos precisos para cada dispositivo virtual que roda no mesmo, com o objetivo de fazer com que esses trabalhem de maneira independente dos demais.

Na Figura 4, é possível notar que, o processador, memória RAM e internet são gerenciados pelo *hypervisor* e compartilhado para outras três máquinas.

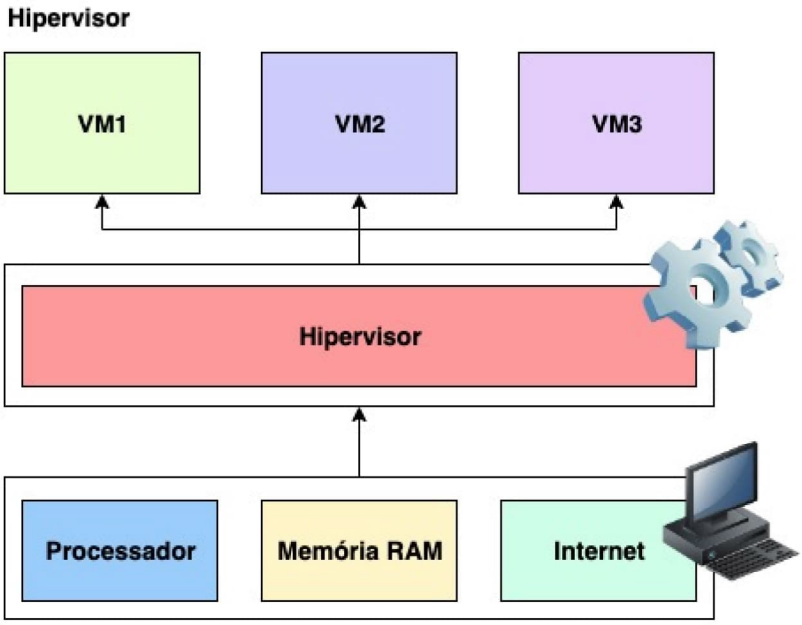


Figura 4 – *Hypervisor*. Fonte: Figura extraída de (COMAR, 2024).

2.2.2 *Hyper-V*

O *Hyper-V* é uma ferramenta desenvolvida pela empresa **Microsoft**, e sua função é realizar virtualização de hardware. Com ela, é possível criar máquinas virtuais que podem ser executadas no seu próprio espaço, ou seja, podem executar inúmeras máquinas de forma individual e ao mesmo tempo. Isso permite construir ambientes de nuvem privados, aumentar a capacidade de uma infraestrutura virtual e contribui para um ambiante de desenvolvimento mais eficiente (FINN et al., 2011).

2.2.3 VMware vSphere

Essa ferramenta foi desenvolvida pela **VMware Inc.**.E, a partir de sua documentação oficial fornecida pela Broadcom (2024), entende-se que sua função é virtualizar os recursos computacionais, armazenamento e rede, sendo capaz de criar um ambiente de data center totalmente virtualizado, com performances e rendimento. O software possui dois componentes principais: o *ESXi*, responsável por criar e executar as máquinas virtuais, ou seja, um modelo de *hypervisor*. Já o *vCenter Server* é um serviço de gerenciamento de *host* que está conectado em uma mesma rede, faz o agrupamento de todos os recursos dos *host*, o que permite realizar balanceamento de carga e monitorar o desempenho do ambiente.

2.2.4 Oracle VM Virtualbox

O Oracle VM Virtualbox é uma ferramenta *open-source*, ou seja, possui seu código fonte disponibilizado e licenciado para realizar atividades para quaisquer finalidades. Ele

foi desenvolvido pela **Oracle Corporation**, sua função é construir máquinas virtuais a nível de software, é uma ferramenta capaz de suportar plataformas de host com imagens de sistemas operacionais diferentes. Em outras vertentes, existe a possibilidade de realizar a virtualização em nível de hardware (Oracle Corporation, 2024).

2.2.5 Multipass

Foi desenvolvido pela **Canonical** e tem como objetivo permitir criar e gerenciar instâncias de máquinas virtuais. O Multipass faz utilização de um *Hypervisor* tipo 2, realiza interação com o sistema para objetar informações que estão ocultas, ou seja, faz a execução no hospedeiro principal e utiliza do *hypervisor* do próprio sistema operacional, no caso do Windows, o *Hyper-V*. Por ser uma ferramenta leve, torna-se ideal para desenvolver ambientes locais de forma rápida e eficiente, tornando-se ótima escolha para o projeto deste trabalho (Canonical Ltd, 2024).

2.3 Visão Geral do Kubernetes

O Kubernetes é uma plataforma de código aberto para automações, implementações, escalabilidade e gerenciamento de contêineres, em que sua ideia é facilitar a organização dos ambientes, como exemplo, microsserviços. Com suas ferramentas, o Kubernetes fornece estruturas de execução para sistemas distribuídos de forma robusta, garantindo seu funcionamento de forma adequada, assim como também é capaz de gerenciar contêineres e implementação de aplicativos (Google Cloud, 2024).

Conforme a documentação oficial de kubernetes The Kubernetes Authors (2023), ele pode oferecer: balanceamento de carga, orquestração de armazenamento, autocorreção, execuções em lote, possibilidades de extensões e muito mais.

2.3.1 Arquitetura do Kubernetes

Um *cluster*, que é uma estrutura de servidores interconectados trabalhando em harmonia, fornece o ambiente perfeito para a implementação do Kubernetes, também conhecido como *cluster* Kubernetes. O Kubernetes é dividido em duas partes distintas: o plano de controle e os nós de computação. Os nós, tanto físicos quanto virtuais, formam a base operacional do *cluster* e cada um desses nós executam pods, que consistem em contêineres relacionados, representando a unidade de execução no Kubernetes. Esses pods, ao agruparem contêineres, facilitam a gestão eficiente de recursos, proporcionando uma abordagem flexível e escalável para a execução de aplicativos no ambiente do Kubernetes (FREIRE, 2021).

Na Figura 5, é possível enxergar o plano de controle (*Master Node*) e os nós de computação (*Worker Nodes*).

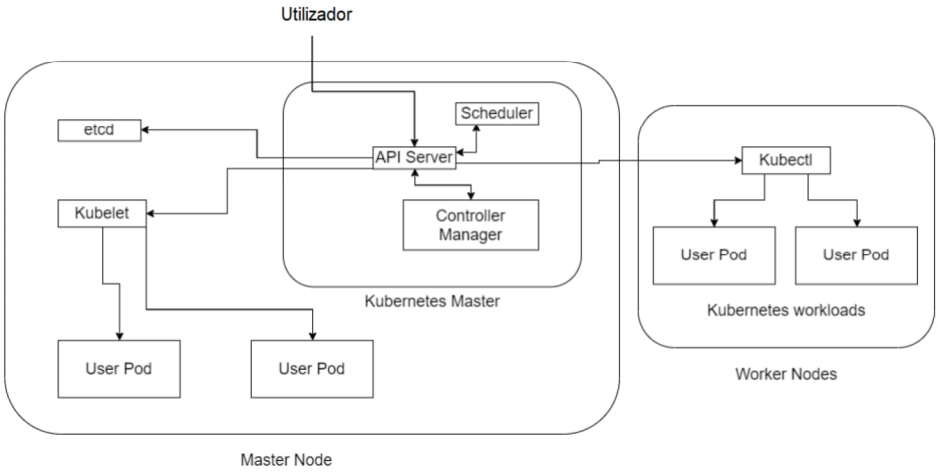


Figura 5 – Arquitetura do Kubernetes. Fonte extraída de (FREIRE, 2021).

2.3.1.1 Plano de Controle - *cluster*

É importante dizer que o plano de controle realiza interações com as máquinas a fim de garantir que o *cluster* seja executado da maneira que se espera, ou seja, como foi configurado (The Kubernetes Authors, 2023). A Figura 5 inclui alguns componentes do *cluster* Kubernetes que desempenham papéis essenciais na coordenação, agendamento, gerenciamento de recursos e armazenamento de dados de configuração, são eles:

- ***API Server***: Responsável por fornecer a interface *RESTfull*, que permite as interações de usuários e componentes com o *cluster*, validar as solicitações recebidas e retornar a solicitação de forma positiva ou não dependendo dos permissionamentos, realizar comunicações com os outros componentes, *Scheduler*, o *Controller Manager* e o etcd, para orientar as operações do *cluster*.

- ***Scheduler***: Capaz de armazenar dados dos recursos para cada nó, capaz de validar se os contêineres podem ser implementados dependendo da integridade do *cluster*. Essa integridade é feita com base nas demandas do pod, como CPU ou memória, solicitando o mais apropriado para realização de tarefas, pod ou serviço, garantindo a utilização dos recursos.

- ***Controller Manager***: Através do servidor de uma Interface de Programação de Aplicação (API), faz a observação do estado desejado do *cluster* com o estado atual.Caso o estado atual e o desejado não forem correspondentes, o controlador realiza correções e direciona o status do objeto para o desejado.

- ***ETCD***: É um banco de dados. Armazena as configurações e o estado do *cluster*, é ele quem permite a interface *RESTfull* acessar e alterar dados armazenados do Kubernetes.

- ***Kubelet***: Responsável por receber as especificações do pod por meio do Servidor API, executando e realizando ações para garantir que estejam executando de forma íntegra.

2.3.1.2 Nós de computação - *cluster*

Para compreender adequadamente a segunda parte, que trata dos *clusters*, é crucial primeiro entender o conceito de um “nó”. Em termos simples, um “nó” é uma referência a um dispositivo ou máquina, seja ele físico ou virtual, que tem a capacidade de enviar, receber ou encaminhar dados para pelo menos um outro dispositivo. Cada um desses dispositivos possui seu próprio endereço de identificação único. Esta interconexão de dispositivos é o que constitui uma rede de computadores (TANENBAUM; WETHERALL, 2010). Em sistemas distribuídos, como *cluster* de servidores, o "nó" representa um servidor individual que integra uma rede de servidores de tamanho incontável, onde cada um pode realizar atividades distintas que, no fim, comuniquem-se para realizar operações complexas. A Figura 6 ilustra os nós formando uma rede distribuída, interagindo com um equipamento de armazenamento.

Figura 6 – Sistema de *Cluster* - Traduzido. Fonte extraída de (REHMAN, 2022).

Na arquitetura do Kubernetes, há um responsável por realizar o agrupamento de contêineres, que são os chamados **"pod"**. Os pods realizam a junção de contêineres de forma com que eles consigam compartilhar recursos computacionais e rede local. Esses recursos são ligados em um sistema *cluster*, conforme Figura 7, tornando-o um sistema distribuído robusto(*Fig.6*). Resumindo, os pods são capazes de fornecer implementações e gerenciamento de aplicativos em ambientes de contêineres (Red Hat, 2017).

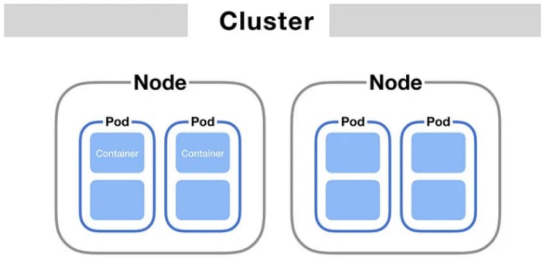


Figura 7 – Estrutura de Pods. Fonte extraída de (HABBEMA, 2023).

A representação da Figura 7, representa duas máquinas que fazem parte de uma estrutura *cluster* (Fig.6), responsável por conectar os *nodes*, na tradução livre "nó". Os nós, são responsáveis por executar os pods que estão orquestrados, fazendo com que as aplicações, recursos de rede, armazenamento, memória e CPU, sejam compartilhados a fim de manter um serviço com alta performance. É importante mencionar que, os pods são responsáveis por gerenciar o ciclo de vida do contêiner.

Por fim, sobre nós de computação de um *cluster*, especificamente do Kubernetes, conhecido como serviços de proxy do kube ou somente **kube-proxy**, um facilitador de serviços de rede do Kubernetes, capaz de controlar o tráfego de rede para os serviços criados no *cluster*, é ele quem se comunica com os pods e faz com que os serviços sejam expostos ao restante do mundo (BELAMARIC; HOCKIN, 2021).

O kube-proxy realiza o balanceamento de carga, em inglês, *load balance*, ou seja, faz a distribuição do tráfego de rede que está sendo direcionado a um pod, assim, permite uma alta disponibilidade das aplicações. Para facilitar, é importante entender que o kube-proxy possui relevância significativa na arquitetura do Kubernetes, pois são responsáveis por balancear a carga, realizar comunicação entre os pods e garantir a segurança dos serviços que estão sendo expostos para o mundo.

2.3.2 *Domain Name System(DNS)* para Serviços e Pods

O *Domain Name System*, que, na tradução livre, é chamado de Sistema de Nomes de Domínio, tem como função nomear os endereços de IP para nomes de domínios. Em resumo, o Tanenbaum e Wetherall (2010) trazem como base que o DNS é:

> [...] a criação de um esquema hierárquico de atribuição de nomes baseado no domínio e de um sistema de bancos de dados distribuídos para implementar esse esquema de nomenclatura(TANENBAUM; WETHERALL, 2010).

Saber e compreender por completo o sistema DNS exigem trabalhos e estudos específicos. Neste trabalho, com foco na utilização do Kubernetes, é importante entender que o ele cria registros DNS para que os pods e os serviços definidos no *cluster* recebam os nomes DNS. Exemplo, um serviço chamado "frontend"rodando no *cluster* Kubernetes, têm um *namespace* "frontend_namespace", o Kubernetes cria um registro DNS, permitindo os pods utilizarem este nome para referenciar o serviço, fazendo com que as comunicações ocorram pelos nomes criados e não por IP (The Kubernetes Authors, 2023).

2.3.3 *Horizontal Pod Autoscale - HPA*

No Kubernetes , o *Horizontal Pod Autoscale*, que em português é chamado de Dimensionador Automático de Pods Horizontal, é um recurso que tem como finalidade corrigir de forma automática o número de réplicas de um conjunto de pods dependendo da sua carga de atividade. Essas “réplicas” representam o número de pods com conteineres idênticos em execução. Ao criar o grupo de pods, é possível especificar o número de réplicas que o Kubernetes deve executar. (PACHECO, 2023).

Por exemplo, se a aplicação for composta por um único contêiner, as réplicas do pod mostrarão que várias cópias desse contêiner vão estar executando com diferentes nós do *cluster*. Da mesma forma, se a aplicação for composta por vários contêineres (por exemplo, um *front-end* e um *back-end*), as réplicas do pod trarão evidências de que várias cópias desses contêineres serão implantadas e gerenciadas pelo Kubernetes.

Sabendo disso, o HPA é capaz de monitorar métricas, como, o consumo de CPU, sendo capaz de regular o número dos pods a fim de manter as aplicações com os recursos necessários com base na demanda. Assim que tem um aumento de trabalho, o HPA aumenta o número de réplicas dos pods para que o *cluster* seja capaz de manter a performance e eficiência da aplicação que está sendo executada. Da mesma forma, quando a carga de trabalho diminui, os números dessas réplicas também são descartados para não haver consumo excessivo (PACHECO, 2023).

Será mostrado no Capítulo 3 um exemplo prático desta escalabilidade horizontal acontecendo de forma automática com o HPA.

2.3.4 *Vertical Pod Autoscale - VPA*

A escalabilidade vertical é uma propriedade do Kubernetes, porém, funciona de maneira distinta ao HPA. Abordado na seção 2.3.3, o HPA aumenta o número de réplicas

de pods para lidar com o aumento de demandas. Já o VPA, que também utiliza os recursos de CPU e memória de um pod, faz o ajuste de acordo com a necessidade, permitindo um dimensionamento adequado das aplicações sem a necessidade de aumentar o número de pods (BEVILAQUA, 2023).

Os sistemas que fazem a utilização do VPA como medida de escalabilidade, pode haver limitações. Exemplo, o VPA solicita a utilização de mais recursos do que os disponíveis no nó, isso faz com que o pod que esteja em execução não receba esses recursos e cause problemas de desempenho. Essa limitação acontece, pois, em um determinado momento, não é possível mais adicionar CPU e memória em um nó, por haver dependência de espaços disponíveis no hardware para adição dessas métricas e também seu custo financeiro. Portanto, a escalabilidade vertical depende desses espaços físicos, tornando-a finita (PINTO, 2022).

É importante destacar que o HPA e o VPA não devem ser configurados para trabalhar em conjunto, já que utilizam a mesma métrica para medir a escalabilidade, o que pode gerar conflitos e causar problemas específicos no sistema, como a falta de controle da escalabilidade. Ambas são ferramentas importantes para realizar operações de escalabilidade em um ambiente Kubernetes, mas cada seus pontos positivos e negativos. A escolha em utilizar uma das métricas depende do das necessidades do ambiente que esta sendo desenvolvido.

2.4 Arquitetura Monolítica

Diferente de uma arquitetura Kubernetes, a arquitetura monolítica possui um modelo de aplicação que é aplicado em um único programa, ou seja, banco de dados, interface do usuário e funcionalidades são feitos em uma única unidade. Hoje em dia, este tipo de arquitetura é associado a softwares legados, mas é uma forma errada de pensar, visto que ela, por mais antiga que seja, ainda pode ser estruturada em ambientes novos. Todavia, é interessante conhecer os diferentes métodos arquitetônicos a fim de evitar futuros problemas, principalmente quando o projeto tem um potencial de crescimento (LUCENA et al., 2021).

Com base nos conhecimentos obtidos nos artigos de Lucena et al. (2021), Garcia e Pagani (2021), Mendes (2021) e Garcia e Pagani (2021), Júnior (2017), é possível concluir que esse tipo de arquitetura ainda é utilizado em contextos variáveis, principalmente em pequenos projetos.

Ainda com base nos artigos, o que se pode concluir sobre a arquitetura monolítica é que ela possui três camadas-chave:

- **Interface do Usuário**: Por meio de uma interface gráfica ou linha de comando são

feitas as interações com o usuário capazes de trazer um retorno visual a repeito do progresso e o estado da aplicação.

- **Camada de Aplicação**: É nesta camada que a lógica de negócio da aplicação localiza-se, podendo fazer solicitações, cálculos, execuções e coordenar a interação entre os diferentes componentes da aplicação.
- **Dados**: Capaz de acessar e efetuar a manipulação de dados armazenados no banco de dados ou em outros sistemas que contêm dados, possuindo métodos de recuperação, criação, atualização e exclusão de dados.

É possível concluir também que, na arquitetura monolítica, há diferentes características.

Simplicidade: Possui um desenvolvimento mais simples e menos complexo de se entender, visto que a aplicação está em uma única base.

Desenvolvimento e Implantação facilitada: Dado ao fato de os componentes estarem juntos, seu desenvolvimento e implantação da aplicação monolítica tornam-se mais fáceis em relação às arquiteturas distribuídas, como o Kubernetes.

Limitação na escalabilidade: Há uma limitação nas aplicações monolíticas pois todas as partes da aplicação são escaladas juntas, fazendo com que haja uma dificuldade em gerenciar os picos de tráfego e dimensionamento das cargas de forma eficiente.

Acoplamento forte: Como a arquitetura monolítica é integrada, seus componentes de aplicação geralmente são acoplados de forma rígida, deixando as mudanças e atualizações mais difíceis e arriscadas.

Essa arquitetura, por mais antiga que seja, ainda possui seu diferencial, mas isso não descarta as mudanças que as empresas buscam em fazer devido às vantagens em termos de escalabilidade, disponibilidade, manutenção e flexibilidade que arquitetura de microsserviços pode oferecer.

2.5 Trabalhos Correlatos

O artigo de Dragoni et al. (2017) discute toda a ideia e evolução do microsserviço, informa suas origens, como é visto atualmente e como pode desempenhar no futuro. Em seu artigo, traz as dificuldades, assim como também as ideias inovadoras da arquitetura de microsserviços trouxeram em relação aos sistemas monolíticos tradicionais, destacando os benefícios de modularidade, escalabilidade e flexibilidade. No artigo, conclui-se que

os microsserviços estão se destacando como ideias revolucionárias, mas que, no fim, são realmente benéficas, porém que ainda há suas fraquezas. Seu trabalho, assim como este, contribui para o crescimento de informações a respeito das implementações do microsserviço.

O trabalho de Freire (2021) realiza a implementação de dois ambientes, um que utiliza Kubernetes e outro, Docker Swarm, duas ferramentas que estão em alta em se tratando de gerenciamento de contêineres com intuito de avaliar o desempenho das ferramentas, mostrando seus pontos fortes e fracos e qual das ferramentas é mais adequada para cada tipo de ambiente de *cluster*.

Os autores Lima e Santos (2019) realizaram, em seu artigo, os conceitos de microsserviços como uma arquitetura moderna, discutindo seus benefícios com relação à arquitetura monolítica tradicional, assim como o trabalho de Dragoni et al. (2017), abordando temas como escalabilidade, manutenibilidade e flexibilidade. O artigo também traz desafios que foram encontrados quando se utilizam microsserviços, que foram gerenciamento de dados distribuídos e comunicação de serviços. Os autores trazem em seu trabalho, com base em estudos teóricos e exemplos práticos, conclusões sobre microsserviços.

3 Criação do ambiente

Neste capítulo, é mostrado todo o processo de criação do ambiente virtual, tornando-se um sistema *cluster* que faz a utilização do Kubernetes.

3.1 Introdução

Este capítulo mostra os recursos das máquinas, instalações e seus eventuais erros e soluções tomadas para contornar os problemas, aplicação utilizada, configurações feitas e por fim um ambiente em seu funcionamento completo. Ao construir um ambiente, é importante saber suas configurações de hardware e as versões de softwares datadas, portanto, neste capítulo, são mostradas todas as configurações e versões das ferramentas. É importante ressaltar que todos os referenciais teóricos abordados no capítulo 2 foram cruciais para a construção do projeto, assim como as documentações fornecidas pelas empresas responsáveis por cada ferramenta que foi utilizada no projeto.

3.1.1 Especificações de Hardware e softwares do ambiente

Na Tabela 2, estão todas as especificações de hardware que estão sendo utilizados como provedor para o ambiente virtual. Essas configurações são de uma máquina intermediária com requisitos computacionais suficientes para suportar outras três máquinas virtuais com recursos computacionais mínimos para execução de atividades simples de um usuário convencional, sendo então capaz de simular o ambiente que foi criado.

Host	
CPU	AMD Ryzen™ 7 6800H CPU @3.2GHz, 8 núcleos, 16 processadores lógicos
Memória	Slot 1 DDR5-4800 (2400 MHz) 16GB Slot 2 DDR5-4800 (2400 MHz) 8 GB
Disco	SM2P41C3 NVMe ADATA 512GB
Placa Mãe	Dell Inc. 0HWH6N .4XG8XW3.BRCMM002BO00DC.A02
Sistema Operacional	Windows 11 Home Single Language Versão 23H2

Tabela 2 – Especificações do Host

Agora, na Tabela 3, nota-se as configurações das máquinas criadas para construir o ambiente virtual. Com base nas configurações de um computador comum, foram criadas três máquinas com as mesmas configurações, projetadas para funcionar como ambiente de *cluster*, portanto compartilham seus recursos para realizar tarefas e processos de dados com intuito de melhorar a performance do ambiente, que, por sua vez, passa a ser um ambiente distribuído com três nós.

Máquinas Virtuais	
CPU	AMD Ryzen™ 7 6800H CPU @3.2GHz, 8 núcleos, 16 processadores lógicos
Memória	Multipass Memória 4GB
Disco	Multipass Hard Disd 40GB
Placa Mãe	Multipass Placa Mãe
Sistema Operacional	Ubuntu 16.04 LTS

Tabela 3 – Especificações das 3 Máquinas Virtuais

3.2 Instalação do Multipass

Para realizar a instalação e configuração do Multipass, foi necessário instalar o VirtualBox para que ele fizesse a função do *hypervisor*, para que, assim, fosse possível realizar o gerenciamento das máquinas. Ao iniciar o processo de instalação, havia a opção de utilizar o *hyper-V*, mas, devido à versão do **host** ser o Windows Home Single, essa opção estava desabilitada, então, utilizou-se o virtualbox na instalação. Para a utilização do Multipass, agora instalado, faz-se necessário utilizar um interpretador de linha de comando, neste trabalho, o **PowerShell**. Sabendo disto, ao se iniciar comando `multipass start` no PowerShell, a seguinte mensagem retornava:

Erro:
start failed: The Hyper-V Windows feature is disabled. Please enable by using the following command in an Administrator PowerShell and reboot:
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All

Para contornar o erro, foi necessário habilitar o *Hyper-V* em recursos do Windows, mas, devido ao *host* estar com uma versão simples do Windows, essa opção não aparece de forma natural, sendo necessário executar um script, para que fosse possível essa alteração (Figura 8).

```
pushd "%~dp0"
dir /b %SystemRoot%\servicing\Packages\*Hyper-V*.mum >hyper-v.txt
for /f %%i in ('findstr /i . hyper-v.txt 2^>nul')
do dism /online /norestart /add-package:"%SystemRoot%\servicing\Packages\%%i"
del hyper-v.txt
Dism /online /enable-feature /featurename:Microsoft-Hyper-V -All /LimitAccess /ALL
pause
```

Figura 8 – Script para habilitar o Hyper-V em Windows Home Single. Fonte: do autor

Após a execução do script, o Windows reconheceu a existência do *Hyper-V* e foi possível habilitar essa função (Figura 9).

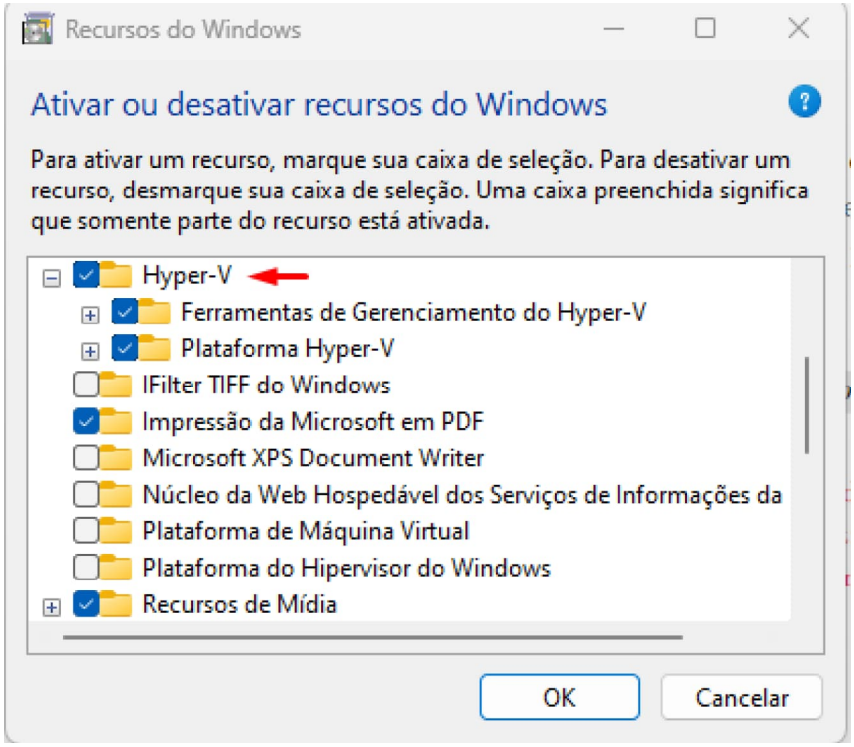


Figura 9 – Habilitando Hyper-V no Windowns. Fonte: do autor

Após as execuções das Figuras 8 e 9, foi necessário alterar onde eram armazenadas as imagens das máquinas virtuais do Multipass na máquina do *host*. O Multipass foi pausado, e realizou-se a troca do **local.drive**, que recebia o virtualbox como armazenador de imagens, passando então a ser o *Hyper-V* quem recebe as imagens criadas no Multipass. Após isso, foi iniciado novamente o processo da ferramenta Multipass (Figura 10).

```
PS C:\Users\cerut> multipass get local.driver
virtualbox
PS C:\Users\cerut> multipass version
multipass   1.12.2+win
multipassd  1.12.2+win
PS C:\Users\cerut> multipass stop --all
PS C:\Users\cerut>
PS C:\Users\cerut>
PS C:\Users\cerut> multipass set local.driver=hyperv
PS C:\Users\cerut>
PS C:\Users\cerut>
PS C:\Users\cerut>
PS C:\Users\cerut> multipass start --all
PS C:\Users\cerut>
PS C:\Users\cerut>
PS C:\Users\cerut>
PS C:\Users\cerut> multipass get local.driver
hyperv
PS C:\Users\cerut>
```

Figura 10 – Alteração do caminho local.driver. Fonte: do autor.

3.3 Criação das Máquinas Virtuais

Para construção das máquinas virtuais no Multipass, foi utilizado o manual feito pela empresa Canonical Ltd (2024), cuja especificações estão presentes na Tabela 3. Ao executar o comando abaixo (Figura 11), a primeira máquina é construída. É repetido o processo duas vezes para a construção das demais.

multipass launch lts –name (nome_a_escolher) –memory 4G –disk 40G –cpus 8 –network Ethernet

Figura 11 – Comando Multipass - Criação de máquina. Fonte: do autor

As três máquinas criadas (Figura 12) ainda são independentes umas das outras, são máquinas que não possuem interface gráfica e com configurações computacionais idênticas.

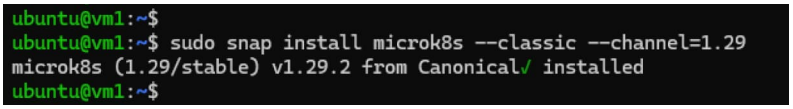
```
PS C:\Users\cerut> multipass list
Name                    State             IPv4             Image
vm1                     Running           172.31.218.87    Ubuntu 22.04 LTS
                                          192.168.100.137
vm2                     Running           172.31.215.106   Ubuntu 22.04 LTS
                                          192.168.100.138
vm3                     Running           172.31.212.70    Ubuntu 22.04 LTS
                                          192.168.100.139
```

Figura 12 – Lista de Máquinas Virtuais Criadas. Fonte: do autor.

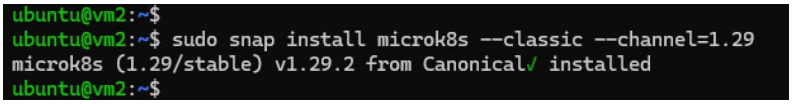
3.4 Instalação do Kubernetes - Microk8s

Para criar um *cluster* na arquitetura do Kubernetes, precisa-se instalar o Kubernetes em todas as máquinas e, para isso, foi utilizado também o manual da empresa Canonical Ltd (2024), mas agora voltado para construção de *cluster*. O **Microk8s** é uma distribuição do

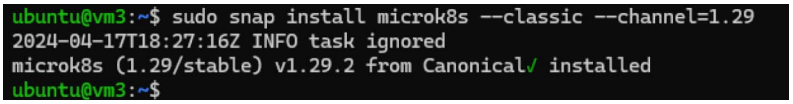
Kubernetes leve e de fácil instalação, seu foco é para desenvolvimento de pequenos projetos, como deste trabalho. Na Figura 13, destacado pela cor verde, nota-se o nome "ubuntu@vm1", "ubuntu@vm2" e "ubuntu@vm3", o que identifica cada máquina. Individualmente é feita a instalação do Microk8s, que, no caso, requer apenas digitar o comando no próprio terminal da máquina virtual.



(a) Máquina Virtual 1 - VM1



(b) Máquina Virtual 2 - VM2



(c) Máquina Virtual 3 - VM3

Figura 13 – Microk8s Instalado individualmente. Fonte: do autor.

3.5 Configurando o Ambiente

Nesta sessão, são feitas todas as configurações do ambiente virtual, transformando-o em *cluster* com arquitetura Kubernetes, sendo assim, as três máquinas da Figura 13 formam um sistema distribuído capaz de realizar distribuição de carga, escalabilidade, alta disponibilidade, entre outras vantagens que serão abordadas.

O projeto trata de um ambiente de desenvolvimento que utiliza microsserviços e o Kubernetes fazendo a orquestração, portanto, inicialmente, é preciso escolher o nó principal para o projeto, que, neste caso, será a 'vm1' (Figura 13a). Para que essa VM possa realizar as atividades sem restrições, o usuário existente nela, chamado de **ubuntu**, passa a ser um administrador, permitindo que ele construa e configure o sistema (Figura 14). O processo de liberação de usuário também ocorre nas demais máquinas.

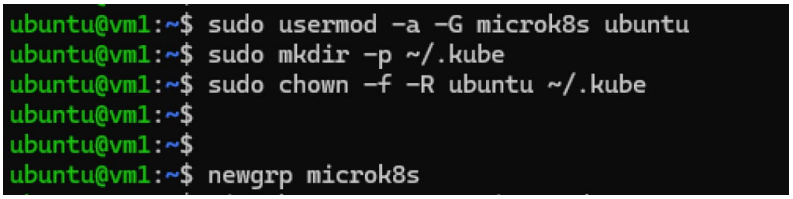


Figura 14 – Liberação de acesso. Fonte do autor.

3.5.1 Configuração da Aplicação E-Commerce

Quando se constrói um ambiente com arquitetura de microsserviços, é esperado que exista ao menos uma aplicação em desenvolvimento ou já desenvolvida. Na estrutura deste projeto, foi adotada uma aplicação que qualquer pessoa com acesso a plataforma GitHub possa utilizar, neste caso, é um *E-Commerce* (Figura 23).

O motivo no qual foi escolhido essa arquitetura, é simples. Ela representa como vários módulos, de linguagens diferentes, se comunicam entre si para entregar um projeto. Exemplo, o microsserviço *front-end*, foi desenvolvido na linguagem Go, que ao contrário do microsserviço *cartservice*, onde se armazena os itens selecionados para compra, foi desenvolvido em C#, ou seja, é possível trabalhar com diferentes estruturas de códigos de forma independentes, o que difere do que acontece em uma estrutura monolítica, que consiste em apenas uma linguagem padrão para todo o projeto.

O papel do Kubernetes na aplicação *E-Commerce*, é ser capaz de fornecer uma infraestrutura para orquestrar, implantar, escalar e gerenciar todos os microsserviços presentes, a fim de garantir que executem de forma eficiente e ser possível realizar a alta escalabilidade e disponibilidade. Este *E-Commerce* também permite gerenciar facilmente as dependências entre os microsserviços.

Após entender os motivos que levaram a escolha da aplicação utilizada, da-se inicio a sua configuração. O Multipass ao cria uma imagem Linux, ele constrói um sistema operacional (SO) que contém, por padrão, a versão de um software que é capaz de permitir a reprodução do repositório da plataforma GitHub, chamado de *Git* (Figura 15).

```
ubuntu@vm1:~$ git --version
git version 2.34.1
ubuntu@vm1:~$
```

Figura 15 – Software Git. Fonte: do autor.

Na pasta raiz do sistema, a cópia é realizada através do comando `git clone`, que só se torna possível com o software Git da Figura 15.

Na Figura 16, é possível notar a existência de uma nova pasta chamada **"microsservices-demo"**. Nesta pasta, é feita a implantação do **e-commerce**, fazendo com que, de fato, a aplicação funcione no ambiente e passe a ser possível acessá-la de forma local (Figura 17).

Figura 16 – Clone do Repositório GitHub. Fonte: do autor

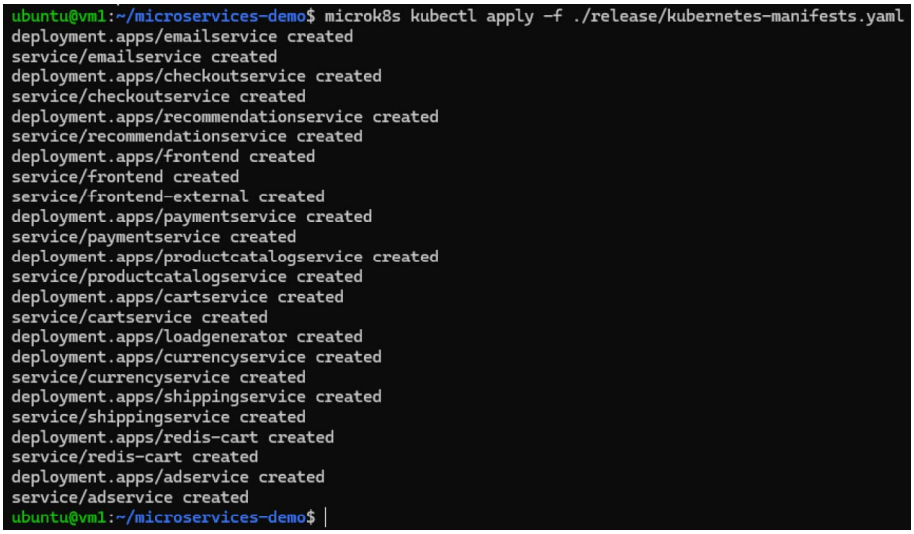


Figura 17 – Implementação da Aplicação. Fonte: do autor.

Com a implementação realizada, significa que o Kubernetes está gerenciando os pods, ou seja, existe um ambiente que está isolado, executando com um grupo de contêineres. A validação dos pods é feita pelo comando *microk8s kubectl get pods*, trazendo como resultado a Figura 18, que, ao mostrar no campo **READY**, que na tradução livre significa "pronto", os números 1/1 significam que existe um contêiner dento do pod e que esse contêiner está pronto para receber informações.

```
ubuntu@vm1:~/microservices-demo$ microk8s kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
adservice-cf48dc6df-jq58v                1/1     Running   0          27m
cartservice-657b544cf4-ql59x             1/1     Running   0          27m
checkoutservice-7644b45df9-tlp4t         1/1     Running   0          27m
currencyservice-7df78c99-56tlh           1/1     Running   0          27m
emailservice-5855bdc465-9sxg9            1/1     Running   0          27m
frontend-695bcc595f-2qfk8                1/1     Running   0          27m
loadgenerator-788d5978c4-99l74           1/1     Running   0          27m
nginx-7854ff8877-2jj6l                   1/1     Running   0          36m
paymentservice-57f58b8dc7-lfsm2          1/1     Running   0          27m
productcatalogservice-5b9d97869f-qgz5m   1/1     Running   0          27m
recommendationservice-7f9cff7b77-zgc8s   1/1     Running   0          27m
redis-cart-bf5c68f69-dgktv               1/1     Running   0          27m
shippingservice-7556967db5-dhdb6         1/1     Running   0          27m
ubuntu@vm1:~/microservices-demo$
```

Figura 18 – Grupo de Contêineres. Fonte: do autor

3.5.2 Habilitando o DNS

Por padrão, ao se criar uma máquina virtual através do Multipass, seu DNS vem desabilitado, acredita-se que isso aconteça devido ao fato de que sua construção é simples, então alguns recursos vêm desabilitados para que essa imagem criada seja rápida e leve, o que permite uma maior flexibilidade para configurar o DNS da maneira que se achar necessário.

Na arquitetura do Kubernetes, é essencial que o serviço esteja funcionando para garantir que os pods comuniquem-se com outros e que os serviços possam ser expostos para o mundo, é isso o que faz os aplicativos como o *e-commerce* funcionarem da maneira correta. De maneira simples, foi executado o comando *microk8s enable dns* para cada máquina (Figura 19).

```
ubuntu@vm1:~$ microk8s enable dns
Infer repository core for addon dns
Addon core/dns is already enabled
ubuntu@vm1:~$
```

(a) Máquina Virtual 1 - VM1

```
ubuntu@vm2:~$ microk8s enable dns
Infer repository core for addon dns
Addon core/dns is already enabled
ubuntu@vm2:~$
```

(b) Máquina Virtual 2 - VM2

```
ubuntu@vm3:~$ microk8s enable dns
Infer repository core for addon dns
Addon core/dns is already enabled
ubuntu@vm3:~$
```

(c) Máquina Virtual 3 - VM3

Figura 19 – Habilitando DNS nas Máquinas. Fonte: do autor.

3.5.3 Habilitando *Load Balance*

Com a implementação do *e-commerce* realizada e os pods em execução (Figura 18), é preciso disponibilizar a aplicação para o exterior, isso significa permitir que outras pessoas acessem o conteúdo que foi desenvolvido e implementado, por meio de outras redes que não pertencem ao *cluster*. Neste trabalho, apenas para fins educacionais, foi realizada a configuração do *Load Balance* (2.3.1.2) e disponibilizado o acesso do ambiente apenas de forma local.

No *cluster* Kubernetes, existe um balanceador de carga chamado **MetalLB** que condiciona os serviços de *cluster* kubernetes a exporem dados fora da rede local do *cluster*. Ele fornece uma solução para a camada de rede do Kubernetes, disponibilizando serviços fora do *cluster*. Portanto, no Kubernetes, para habilitar o metallb, o comando `microk8s enable metallb` foi utilizado (Figura 21) e foi aplicada uma "faixa de endereços de IP"que estão dentro do intervalo de IPs disponíveis na rede da máquina virtual 1.

Para saber o intervalo de IPs disponíveis, o comando `ip a` é utilizado. No campo em destaque da Figura 20, é possível notar o IP da vm1 (Figura 12) e a notação /24. Isso significa que os primeiros 24 bits do endereço são dedicados à rede. Um endereço IPV4 possui 32 bits, sabe-se que 24 são parte da rede, então $32 - 24$ resultam em 8, portanto os 8 bits restantes ($2^8 = 256$) são dedicados aos *hosts* na rede, ou seja, o IP pode variar de 0 a 255, totalizando em 256 IPs disponíveis para se colocar no *load balance*. No entanto, é importante notar que isso não é um limite fixo. A máscara de sub-rede é configurável e, dependendo de como ela é definida, pode permitir um número maior ou menor de hosts. Por exemplo, se a máscara tiver menos de 24 bits, haverá mais bits disponíveis para os hosts, permitindo um maior número de endereços IP. Além disso, o uso do endereço 192.168.x.x permite até aproximadamente 65 mil hosts, demonstrando a flexibilidade e a capacidade de escala dos endereços IPV4.

```
ubuntu@vm1:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 52:54:00:9d:46:61 brd ff:ff:ff:ff:ff:ff
    inet 172.31.218.87/20 metric 100 brd 172.31.223.255 scope global dynamic eth0
       valid_lft 81895sec preferred_lft 81895sec
    inet6 fe80::5054:ff:fe9d:4661/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 52:54:00:2a:5f:21 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.137/24 metric 200 brd 192.168.100.255 scope global dynamic eth1
       valid_lft 81897sec preferred_lft 81897sec
    inet6 fe80::5054:ff:fe2a:5f21/64 scope link
       valid_lft forever preferred_lft forever
6: vxlan.calico: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN group default
    link/ether 66:e4:7a:3c:b2:3c brd ff:ff:ff:ff:ff:ff
    inet 10.1.225.0/32 scope global vxlan.calico
       valid_lft forever preferred_lft forever
    inet6 fe80::64e4:7aff:fe3c:b23c/64 scope link
       valid_lft forever preferred_lft forever
7: cali17dac1a3698@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netns cni-1b3fe148-9f76-8e16-f127-f3a10bd74abb
    inet6 fe80::ecee:eeff:feee:eeee/64 scope link
```

Figura 20 – Faixa de Endereços de IP. Fonte: do autor

```
ubuntu@vm1:~$ microk8s enable metallb
Infer repository core for addon metallb
Enabling MetalLB
Enter each IP address range delimited by comma (e.g. '10.64.140.43-10.64.140.49,192.168.0.105-192.168.0.111'): 192.168.100.140-192.168.100.141
Applying Metallb manifest
customresourcedefinition.apiextensions.k8s.io/addresspools.metallb.io created
customresourcedefinition.apiextensions.k8s.io/bfdprofiles.metallb.io created
customresourcedefinition.apiextensions.k8s.io/bgpadvertisements.metallb.io created
customresourcedefinition.apiextensions.k8s.io/bgppeers.metallb.io created
customresourcedefinition.apiextensions.k8s.io/communities.metallb.io created
customresourcedefinition.apiextensions.k8s.io/ipaddresspools.metallb.io created
customresourcedefinition.apiextensions.k8s.io/l2advertisements.metallb.io created
namespace/metallb-system created
serviceaccount/controller created
serviceaccount/speaker created
clusterrole.rbac.authorization.k8s.io/metallb-system:controller created
clusterrole.rbac.authorization.k8s.io/metallb-system:speaker created
role.rbac.authorization.k8s.io/controller created
role.rbac.authorization.k8s.io/pod-lister created
clusterrolebinding.rbac.authorization.k8s.io/metallb-system:controller created
clusterrolebinding.rbac.authorization.k8s.io/metallb-system:speaker created
rolebinding.rbac.authorization.k8s.io/controller created
secret/webhook-server-cert created
service/webhook-service created
rolebinding.rbac.authorization.k8s.io/pod-lister created
daemonset.apps/speaker created
deployment.apps/controller created
validatingwebhookconfiguration.admissionregistration.k8s.io/validating-webhook-configuration created
Waiting for Metallb controller to be ready.
error: timed out waiting for the condition on deployments/controller
MetalLB controller is still not ready
deployment.apps/controller condition met
ipaddresspool.metallb.io/default-addresspool created
l2advertisement.metallb.io/default-advertise-all-pools created
MetalLB is enabled
ubuntu@vm1:~$
```

Figura 21 – Habilitando Metallb do Kubernetes. Fonte: do autor.

Com metallb habilitado (Figura 21), utiliza-se do comando *microk8s kubectl get svc* para validar os serviços no *cluster* Kubernetes e nele é possível verificar que o *load balance* está habilitado e endereçado no IP desejado. Em destaque, na Figura 22, encontra-se o endereço IP que dá acesso externo para aplicação *e-commerce*, mas, neste caso, só será acessado de forma local.

```
ubuntu@vm1:~$ microk8s kubectl get svc
NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)          AGE
adservice               ClusterIP      10.152.183.248   <none>           9555/TCP         96m
cartservice             ClusterIP      10.152.183.117   <none>           7070/TCP         96m
checkoutservice         ClusterIP      10.152.183.103   <none>           5050/TCP         96m
currencyservice         ClusterIP      10.152.183.75    <none>           7000/TCP         96m
emailservice            ClusterIP      10.152.183.176   <none>           5000/TCP         96m
frontend                ClusterIP      10.152.183.33    <none>           80/TCP           96m
frontend-external       LoadBalancer   10.152.183.51    192.168.100.140  80:30569/TCP     96m
kubernetes              ClusterIP      10.152.183.1     <none>           443/TCP          108m
paymentservice          ClusterIP      10.152.183.149   <none>           50051/TCP        96m
productcatalogservice   ClusterIP      10.152.183.87    <none>           3550/TCP         96m
recommendationservice   ClusterIP      10.152.183.104   <none>           8080/TCP         96m
redis-cart              ClusterIP      10.152.183.34    <none>           6379/TCP         96m
shippingservice         ClusterIP      10.152.183.189   <none>           50051/TCP        96m
ubuntu@vm1:~$
```

Figura 22 – IP de Acesso da Aplicação. Fonte: do autor.

Após finalizar os passos anteriores, a aplicação pode ser acessada através do IP:192.168.100.140 (Figura 23). Com isso, tem-se um sistema executando com apenas um nó (Figura 24), o qual possui um pod orquestrando um contêiner, igual ao que é ilustrado na Figura 7.

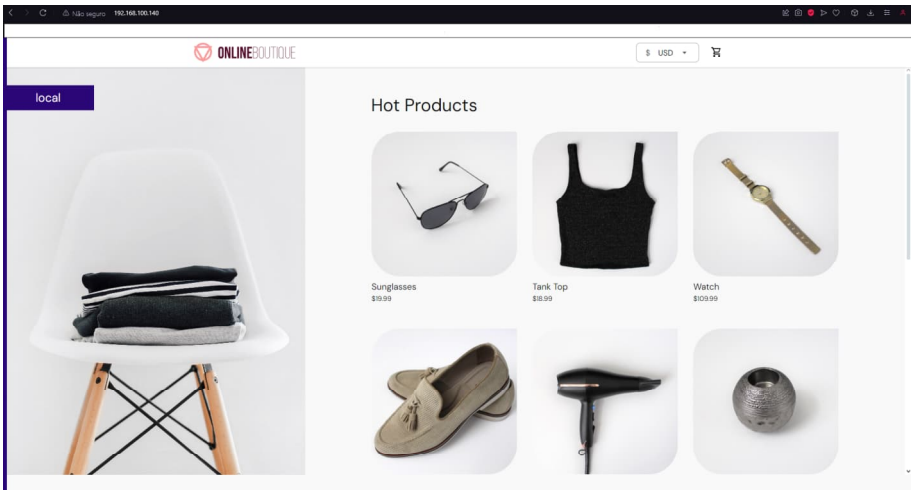


Figura 23 – *E-commerce. Fonte: do autor*

```
ubuntu@vm1:~$ microk8s kubectl get nodes
NAME   STATUS   ROLES    AGE    VERSION
vm1    Ready    <none>   145m   v1.29.2
ubuntu@vm1:~$
```

Figura 24 – Apenas um Nó sendo Gerenciado. Fonte: do autor

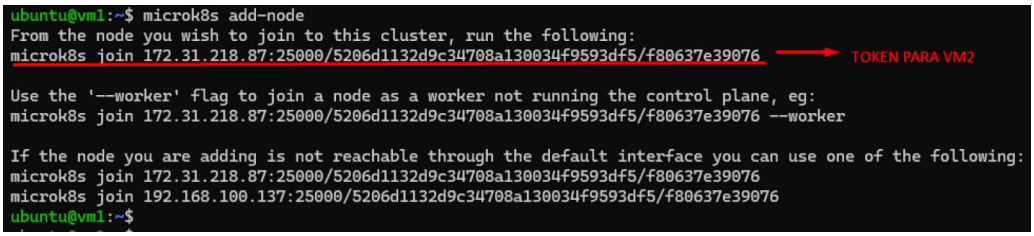
3.5.4 Estruturando o Sistema *Cluster*

Com a VM1 estruturada da maneira necessária, é preciso criar o sistema *cluster*, ou seja, fazer com que as máquinas trabalhem de forma conjunta, somar seus recursos computacionais. Para isso, uma das máquinas deve fornece um "**token**"para que as demais tornem-se nós trabalhadores. Assim como decidido neste projeto, a VM1 é dada como o nó principal e possui os demais nós, VM2 e VM3, podendo trabalhar em conjunto e permitindo que haja uma tolerância a falhas, significa que, com mais nós trabalhando,

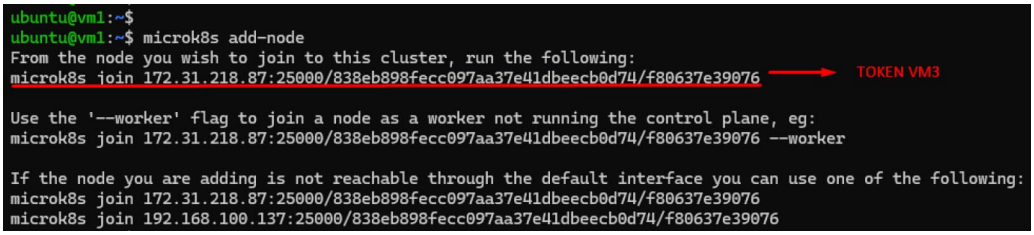
o *cluster* Kubernetes torna-se resistente a possíveis quedas ou perda de dados, o que contribui para alta disponibilidade, pois se um dos nós estiver com problemas, os demais irão assumir.

Além de contribuir com a alta disponibilidade, ao estruturar o *cluster*, ele permite que ocorra a escalabilidade, distribuindo a carga de trabalho de maneira satisfatória, e escalonar os pods de forma automática quando necessário, fazendo com que o ambiente, ao sofrer picos de demandas, possa lidar em fornecer recursos para manter o desempenho da aplicação, conhecido como balanceamento de carga.

Na máquina principal, foi utilizado o comando `microk8s add-node`, que permite ter acesso a um **token**, uma espécie de "chave"que permite que outro nó seja integrado ao nó principal, então, para integrar a VM2 e VM3, foram geradas duas chaves. Esse processo pode ser notado nas Figuras 25 e 26.

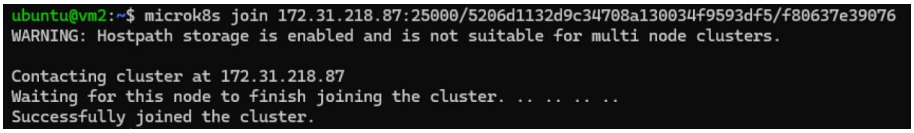


(a) Token gerado para Máquina Virtual 2 - VM2

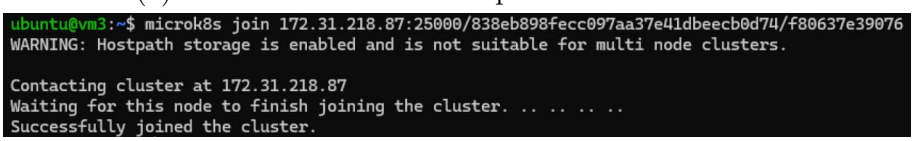


(b) Token gerado para Máquina Virtual 3 - VM3

Figura 25 – Gerando o Token. Fonte: do autor.



(a) Token habilitado na Máquina Virtual 2 - VM2



(b) Token Habilitado na Máquina Virtual 3 - VM3

Figura 26 – Habilitando o Token. Fonte: do autor.

Fonte: do autor

Executando o comando `microk8s kubectl get nodes`, havia apenas um nó, ilustrado na Figura 24. Agora, ao repetir esse comando, é possível notar a existência de três nós

que estão envolvidos em um sistema *cluster* Kubernetes (Figura 27). É importante saber que agora o ambiente está estruturado igual à Figura 6.

```
ubuntu@vm1:~$ microk8s kubectl get nodes
NAME   STATUS   ROLES    AGE   VERSION
vm1    Ready    <none>   16h   v1.29.2
vm2    Ready    <none>   24m   v1.29.2
vm3    Ready    <none>   23m   v1.29.2
ubuntu@vm1:~$
```

Figura 27 – Nós do *Cluster* Kubernetes. Fonte: do autor.

3.6 Adição de Métricas

Agora que se tem um sistema *cluster*, toda operação executada na VM1 é feita de forma automática nas demais VMs, ou seja, ao adicionar um tipo de métrica em uma, as outras também terão. Por padrão, foi utilizada a VM1 como principal para realizar as configurações e todas as outras atividades de construção de ambiente, mas, para o projeto, ela é trabalhadora, assim como as demais, ou seja, ela não é um nó *master*.

Para o Kubernetes realizar o monitoramento de métricas, foi utilizado o *Horizontal Pod Autoscaler (HPA)*, falado na seção 2.3.3. Sua aplicação é simples, porém é importante definir qual métrica é desejada para escalar os pods, no caso do projeto, usou-se a CPU, que é um componente responsável por executar os processos de uma máquina. Quando se está medindo o usa da CPU, significa que o HPA monitora o consumo de processadores. Para que a escalabilidade horizontal aconteça, deve-se também fornecer um limite de consumo dessa CPU, sabendo disso, o primeiro passo para adição das métricas foi utilizar o comando `microk8s enable metrics-server` para habilitar o *metrics-server*, um componente crucial para coletar métricas de recursos do *cluster*. Essas métricas, agora visíveis, são utilizadas pelo HPA para que ele possa dimensionar a escalabilidade dos pods de forma automática.

Com as métricas habilitadas, o HPA precisa saber as configurações que irão fornecer a escalabilidade. Portanto, foi necessário especificar o nome da implementação realizada, a métrica que está sendo monitorada e os valores mínimos e máximos de réplicas que serão utilizados para a escalabilidade. Para isso, foi executado o comando `microk8s kubectl autoscale deployment frontend –cpu-percent=30 –min=1 –max=10`. Este comando está informando que, se deve monitorar o *deployment frontend* e baseado na carga que ele recebe, no caso, 30% da CPU, com um limite de 1 até 10 pods, ele deve realizar a escalabilidade. Essas configurações de monitoramento foram escolhidas para que seja possível demonstrar uma distribuição de carga ocorrendo de forma mais rápida e eficiente

para o projeto, caso estes números fossem maiores, para fins didáticos, ocasionariam em uma demora significativa de tempo de execução até que ocorre uma escalabilidade.

Com o comando *microk8s kubectl get hpa* é possível ter a certeza de que o HPA está monitorando, seu retorno trará o nome do frontend, os *targets* de CPU que foram definidos e o máximo e mínimo de pods que podem estar executando simultaneamente.

3.7 Na Prática

Com o *cluster* Kubernetes funcionando, métricas habilitadas para o HPA e aplicação e-commerce executando, agora é possível realizar a distribuição de carga de forma automática em tempo real, alta disponibilidade em caso de queda em um dos nós e, por fim, ver o alto consumo de CPU. Em resumo, podem-se colocar as seguinte situações:

- **Alta disponibilidade**: Caso a aplicação e-commerce falhe, por exemplo, por falta de conexão, o Kubernetes estará monitorando o estado desse serviço e os pods que estavam executando. Então, ao notar que um deles parou, o seu gerenciamento faz a restauração dos pods em um dos seus nós, no caso do projeto, VM2 e VM3, para que mantenha todos os contêineres funcionais e não haja necessidade de uma interação humana.
  No projeto, foi desabilitado a VM1 no Multipass pelo comando *multipass stop vm1* , o que significa que, no sistema *cluster*, uma máquina foi desligada de forma inesperada. Para que a aplicação não deixasse de fornecer para o mundo externo, a VM2 ou VM3 assumem os novos pods criados. A vm que assume depende de qual está mais apropriada para assumir os recursos que estão sendo utilizados.

- **Escalabilidade Horizontal**: Suponha que a aplicação está sofrendo um número alto de acessos por algum motivo. O e-commerce está com produtos em promoções, por exemplo, o que significa que está ocorrendo um número alto de consumo de recursos, necessitando da realização de um aporte para que o serviço continue entregando com qualidade.
  O Kubernetes implementado no projeto é capaz de mostrar essa situação de alto consumo. Para simular um alto número de acessos, foi utilizado o comando das figuras 28 e 29

```
microk8s kubeclt run -i –tty load-generator –rm –image=busybox:1.28 –restart=Never – /bin/sh -c
```

Figura 28 – Comando (Parte 1). Fonte: do autor

```
"while sleep 0.01; do wget -q -0- http://frontend; done"
```

Figura 29 – Comando (Parte 2). Fonte: do autor.

Detalhando o que acontece nas figuras 28 e 29:

1. *microk8s kubeclt* - comando da distribuição Kubernetes que interage com *cluster*;
2. *run* - inicialização do contêiner;
3. *-i e –tty* - Mantém a entrada padrão aberta (stdin) e a associa a um terminal no contêiner;
4. *load-generator* - nome que foi dado ao pod;
5. *–rm* - Remove o contêiner quando ele é finalizado
6. *–image=busybox:1.28* - Usa uma imagem "busybox:1.28"para criar o contêiner. O BusyBox é uma imagem leve do Linux que contém uma variedade de utilitários de linha de comando;
7. *–restart=Never* - Indica que o contêiner não deve ser reiniciado automaticamente
8. */bin/sh -c "while sleep 0.01; do wget -q -0- http://frontend; done"* - Comando executado dentro do contêiner. Ele executa um loop infinito que faz uma solicitação HTTP para "http://frontend"a cada 0,01 segundos.

Com base no comando executado, enquanto o contêiner estiver executando, o e-commerce ficará atualizando infinitamente a página (Figura 23), realizando um alto número de carga, obrigando o HPA a distribuir recursos à medida que se alcança 30% de CPU.

- **Consumo CPU**: Com o sistema executando em alta performance, é sempre importante monitorar o consumo do ambiente que foi desenvolvido, que, para este projeto, é possível monitorar o consumo de CPU em tempo real enquanto a carga está sendo gerada.
  Para monitorar, faz-se uso do comando `watch -n1 microk8s kubectl get hpa`, que permite a visualização a todo momento do consumo da CPU, no qual, ao atingir mais de 30%, a carga é redistribuída para a próxima VM, aumentando o número de réplicas, ou seja, aumentando o número de pods *frontend*. Algo semelhante ao que ocorre na Figura 30.

```
C:\WINDOWS\system32>kubectl get hpa pressure-api-hpa --watch
NAME                REFERENCE                                  TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
pressure-api-hpa    Deployment/pressure-api-deployment   4%/50%    1         5         1          75m
pressure-api-hpa    Deployment/pressure-api-deployment   4%/50%    1         5         1          75m
pressure-api-hpa    Deployment/pressure-api-deployment   59%/50%   1         5         1          76m
pressure-api-hpa    Deployment/pressure-api-deployment   59%/50%   1         5         2          76m
pressure-api-hpa    Deployment/pressure-api-deployment   31%/50%   1         5         2          76m
pressure-api-hpa    Deployment/pressure-api-deployment   31%/50%   1         5         2          77m
pressure-api-hpa    Deployment/pressure-api-deployment   31%/50%   1         5         2          77m
pressure-api-hpa    Deployment/pressure-api-deployment   31%/50%   1         5         2          77m
```

Figura 30 – Consumo de CPU. Fonte: extraída de (RAI, 2022)

Em vermelho, na figura 30, o número 59% representa que a CPU atingiu mais de 30% e, assim, realizou a criação de mais uma réplica, destacada pela cor verde. Por

se tratar de uma imagem representativa, o valor esperado pelo hpa para criar uma nova replica é acima de 50%.

Por fim, temos uma estrutura completa de um ambiente feito com arquitetura de Kubernetes, sendo possível notar seu potencial com relação à praticidade, à agilidade, ao desempenho que é capaz de fornecer, além de segurança em poder isolar os recursos do *cluster*, a fim de evitar falhas de serviços.

3.8 Vantagens e Desvantagens do Kubernetes

Com base na construção do sistema *cluster* Kubernetes para utilização de microsserviços em ambientes de desenvolvimento, podem-se concluir algumas observações. O Projeto foi eficaz em evidenciar sua utilização na prática, trazendo uma base para entender como funciona a aplicação que utiliza Kubernetes e sabendo da existência do modelo monolítico. Foi construído uma estrutura para mostrar vantagens e desvantagens do Kubernetes com a arquitetura monolítica, a fim de descrever com maior clareza as suas características. É importante ressaltar que as informações sobre o monólito foram obtidas de trabalhos correlatos e as do Kubernetes com base nos estudos de caso e construção do projeto.

1. **Vantagens**

   a) **Escalabilidade**

   - Kubernetes:
     - O Kubernetes permite escalar horizontalmente, adicionando ou removendo contêineres conforme necessário, tornando-o ideal para lidar com picos de tráfego e cargas variáveis
   - Arquitetura Monolítica:
     - Escala verticalmente, seus recursos são compartilhados, seus módulos são executados de forma independente e isso faz com que o serviço seja escalado como um todo e não de forma localizada (LOPES, 2021).

   b) **Alta Disponibilidade**

   - Kubernetes:
     - Com os nós do *cluster* estruturado torna-se automática a recuperação de falhas, o Kubernetes garante que os aplicativos continuem disponíveis, mesmo em caso de problemas de hardware ou software.
   - Arquitetura Monolítica:
     - A alta disponibilidade pode ser algo mais custoso, pois geralmente requer configurações complexas de balanceamento de carga e redundância de servidores (SOUZA et al., 2023).

c) **Modularidade**

- Kubernetes:
  - A arquitetura de microsserviços utilizando Kubernetes permite que os aplicativos sejam divididos em componentes independentes, facilitando a manutenção e o desenvolvimento ágil.
- Arquitetura Monolítica:
  - A modularidade está intimamente ligada dentro de apenas um aplicativo, com o aumento de número de softwares, aumenta o número de módulos o que consequentemente torna mais complexo as dependências e dificulta a manutenção e atualização do ambiente(LOPES, 2021).

d) **Implementação Contínua**

- Kubernetes:
  - Com ferramentas integradas, o Kubernetes pode simplificar o processo de atualização e entrega de aplicativos, promovendo uma cultura de desenvolvimento ágil e iterativo.
- Arquitetura Monolítica:
  - A arquitetura monolítica não possui muita flexibilidade, seu processo é lento e com alto risco de falhas em cascata, gerando alto tempo para implementações, mas não descarta sua eficiência(SOUZA et al., 2023).

e) **Orquestração de Contêineres**

- Kubernetes:
  - O Kubernetes gerencia automaticamente o ciclo de vida dos contêineres, distribuindo e balanceando as cargas de trabalho de forma eficiente, garantindo a disponibilidade e o desempenho dos aplicativos.
- Arquitetura Monolítica:
  - Sem suporte nativo para contêineres, a orquestração de aplicativos monolíticos é incomum ou até inexistente, mas é possível implementar mas pode gerar desafios e futuros problemas no ambiente (SOUZA et al., 2023).

f) **Sobrecarga de CPU**

- Kubernetes:
  - A distribuição de cargas de trabalho entre os nós do *cluster* ajuda a evitar a sobrecarga de CPU, permitindo que as aplicações sejam dimensionadas conforme necessário.
- Arquitetura Monolítica:

  - A sobrecarga de CPU pode ser um problema, especialmente em sistemas monolíticos, onde uma única instância de aplicativo pode monopolizar recursos (JUSTINO, 2018).

2. **Desvantagens**

   a) **Complexidade**

   - Kubernetes:
     - A utilização de contêineres, quando se trata de ambientes de larga escala, exige uma especialização sobre Kubernetes e/ou ferramentas de gerenciamento de contêineres, assim como também é necessário um grande conhecimento em redes, infraestrutura em nuvem e sistemas distribuídos, deixando a curva de aprendizado mais íngreme.
   - Arquitetura Monolítica:
     - Na arquitetura monolítica, por possuir uma estrutura simplificada, exige menos habilidade técnicas e conhecimento específicos, se comparado a utilização de outras arquiteturas. Além de, abordar apenas uma linguagem de programação(GARCIA; PAGANI, 2021).

   b) **Custo indireto**

   - Kubernetes:
     - A execução de um *cluster* Kubernetes requer recursos adicionais, que incluem capacidade computacional, armazenamento e largura de banda de rede, o que pode ocorrer um aumento nos custos operacionais e complexidade da infraestrutura física ou virtual.
   - Arquitetura Monolítica:
     - A implantação de aplicativos monolíticos em um único servidor ou máquina virtual geralmente requer menos recursos e oferece uma eficiência de uso de recursos mais alta. Quando trata-se de ambiente mais simples, a complexidade de uma infraestrutura é de menor escala(SOUZA et al., 2023).

   c) **Desempenho**

   - Kubernetes:
     - A comunicação com o Kubernetes é feito com protocolos, utilização de API's e necessidade de um ambientes com alta taxa de transferência entre os nós, fazendo com que qualquer situação não programada, como problemas de rede, levar ocorrência de alta latência.
   - Arquitetura Monolítica:

  - O desempenho costuma ser mais consistente, pois sua comunicação é feita de forma direta, ou seja, dentro da própria aplicação (MENDES, 2021).

d) **Dependência de Infraestrutura**

- Kubernetes:
  - Para executar o Kubernetes, é necessário provisionar e gerenciar uma infraestrutura de *cluster* de contêineres, incluindo servidores, redes, armazenamento e serviços em nuvem, o que pode aumentar a complexidade e a dependência de fornecedores externos.
- Arquitetura Monolítica:
  - Diferente do Kubernetes, a arquitetura monolítica pode ser implantada em uma ampla variedade de ambientes, desde servidores locais até nuvens públicas, e requer menos dependências externas para operar, por isso, o Kubernetes pode ser uma desvantagem se comparado a monolítica. (MENDES, 2021).

3.9 Desafios e Obstáculos do Projeto

O desenvolvimento do ambiente com a utilização do Kubernetes trouxe alguns desafios e obstáculos singulares, superados à medida que o projeto foi avançando. Primeiramente, ao iniciar o projeto, foram utilizados recursos da empresa Amazon, que era o aluguel de uma máquina virtual da própria empresa, onde foi feito o sistema *cluster* Kubernetes, mas este tipo de serviço gerava um custo financeiro alto e por se tratar de um projeto acadêmico, foi mais viável a realização do projeto de forma local.

Quando se iniciou o projeto localmente, foi inquestionável a necessidade de um upgrade na máquina inicial, o que exigiu uma máquina com as configurações contidas na Tabela 2. Isso ocorreu devido a necessidade de criar três máquinas que tivessem uma configuração aceitável para a construção de um sistema *cluster*.

Com a nova máquina em mãos, foi necessário corrigir o erro citado na seção 3.2, o que representou um grande desafio. Encontrar comunidades que dessem uma visão ou solução do caso não foi fácil, as documentações oficiais não foram encontradas, nem meios de contornar a situação.

Outro problema ocorreu ao criar um sistema *cluster* (Seção 3.5.4), as três máquinas/nós passavam a interagir entre si, mas, após algum tempo, essas interações perdiam-se e o serviço Kubernetes ficava indisponível sem motivo aparente. Inicialmente, acreditase que o problema ocorria quando se fazia o teste de alta escalabilidade. Ao forçar a máquina, estava havendo estouro de memória de disco, ou seja, HD, pois, no começo,

estavam reservados apenas 10GB de espaço. Após mais estudos, encontrou-se no site da Microsoft uma inspiração do verdadeiro problema. Imagina-se que, ao desligar a máquina física, as máquinas do Multipass estavam perdendo as configurações do temporizador, impossibilitando a utilização correta das máquinas criadas novamente. Esse problema ocorria sempre que o *cluster* era adicionado em um novo nó, problema contornado com o não desligamento da máquina física.

No mais, não ocorreram maiores problemas, boa parte das documentações para a construção do ambiente, que foram abordados neste trabalho, estavam com informações relevantes e com exemplos similares, se não iguais, que facilitaram a construção da arquitetura Kubernetes de uma forma simplificada.

4 Conclusão

Neste trabalho de conclusão de curso, foram expostos os conceitos de microsserviços como um método de arquitetura para desenvolvimento de software, com ideia de propagar conhecimento sobre o Kubernetes e uma nova opção além da arquitetura tradicional. Ao longo do projeto, foi possível analisar as vantagens e desvantagens dessa abordagem em comparação com a tradicional arquitetura monolítica, sendo possível destacar os principais requisitos a serem considerados para garantir qual dos dois modelos pode ser melhor para utilizar no ambiente.

Os microsserviços emergiram como uma alternativa viável à abordagem monolítica, oferecendo benefícios significativos em termos de escalabilidade, disponibilidade, implantação da aplicação facilitada, segurança a falhas e orquestração de contêineres. Conclui-se que os pontos-chave que impulsionaram a adoção dos microsserviços são justamente a sua capacidade de escalar horizontalmente, adicionar e remover serviços independentemente e sua facilidade em realizar atualizações de pontos de serviços específicos sem a necessidade de atualizar todo ambiente.

No entanto, é possível notar que os microsserviços não são uma solução universal e acompanha desafios únicos. Seu aspecto operacional robusto, com curva de aprendizado elevado e a dependência de uma infraestrutura com poder computacional maior são aspectos a serem considerados ao optar por essa arquitetura. Também é necessário considerar que sua comunicação com outros serviços e sua gestão a respeito da consistência dos dados, em um ambiente distribuído, trazem consigo desafios que podem exigir abordagens específicas.

É possível frisar, com base no trabalho, que, ao utilizar a arquitetura monolítica ou microsserviço, devem-se considerar as necessidades e características específicas de cada projeto, não há uma solução única que se aplique a todas as situações. Portanto, é fundamental que sejam avaliados de forma minuciosa todos os prós e contras de cada estrutura antes de tomar a decisão, seja para começar um ambiente novo ou a migração do monólito para microsserviço.

Avaliando o cenário atual e as perspectivas do futuro, a tendência é que os ambientes de desenvolvimento continuem evoluindo para se tornarem mais ágeis, flexíveis e escaláveis. Os microsserviços desempenham um papel fundamental neste cenário porque fornecem uma arquitetura modular e distribuída que pode se adaptar às mudanças nas necessidades dos negócios.

Uma última conclusão, seja qual for a escolha arquitetônica adotada, é essencial manter um foco contínuo na entrega de valores e isso é fundamental para todos os envolvidos no projeto. O sucesso da aplicação desenvolvida não está apenas na escolha da tecnologia

ou arquitetura, mas na capacidade de fornecer soluções eficazes e de alta qualidade que atendam às necessidades do mercado.

4.1 Contribuições

Com base no que foi apresentado neste trabalho, as principais contribuições que se destacam são:

1. **Comparação entre Arquitetura Monolítica e Microsserviços**: Feito uma análise das vantagens e desvantagem da utilização de microsserviços se comparado a arquitetura monolítica. Ajudando os desenvolvedores e profissionais do ramo a tomarem decisões de qual arquitetura é mais adequada para o projeto, pois com os pontos fortes e fracos sobre microsserviços, permite uma compreensão mais completa a respeito da sua estrutura, desenvolvimento e prática que permite uma compreensão melhor do modelo arquitetônico.

2. **Entendimento da Arquitetura Microsserviços**: Ao explorar as características do microsserviço o trabalho ajuda a aumentar o conhecimento sobre ele, podendo beneficiar estudantes, profissionais de TI e pesquisadores.

3. **Exploração da Orquestração de Contêineres**: Feito um levantamento sobre orquestração de contêineres em ambientes de microsserviços, onde destaca-se fortemente o papel do Kubernetes neste contexto, fornecendo uma boa compreensão da ferramenta e quais tecnologias necessárias para gerenciar com eficiente o sistema *cluster* criado.

4. **Análise da Complexidade**: O trabalho trouxe alguns temas a respeito da complexidade operacional e seus desafios, sendo capaz de ajudar aqueles que buscam avaliar as compensações que envolvem a adoção de uma arquitetura em comparação a outra.

4.2 Trabalhos Futuros

O processo de criar um ambiente de desenvolvimento que faz a utilização de microsserviços seguiu com o padrão desejado, com construção de um ambiente *cluster* Kubernetes capaz de fornecer alta disponibilidade, escalabilidade e otimização de recursos. Porém é válido ressaltar que, ainda existem pontos que podem ser aprofundados e melhor explorados.

Um exemplo, explorar maneiras de otimizar a implementação e o gerenciamento de microsserviços para melhorar o seu desempenho, segurança e até mesmo a escalabi-

lidade, podendo trazer ferramentas e novas práticas de automação para facilitar uma implementação continua, promovendo ainda mais a cultura de DevOps.

Ainda sobre maneiras de otimização, uma área promissora para trabalhos futuros é a escalabilidade e resiliência dos microsserviços. É possível explorar novas técnicas e aprimoramentos que facilitem o balanceamento de carga de forma estratégica, melhorem a distribuição de tráfego e ofereçam meios mais eficazes de monitorar falhas, de modo que essas falhas possam ser implementados métricas e detecções proativas de problemas nos microsserviços, visando tornar a arquitetura ainda mais automatizada e eficiente.

Por fim, um trabalho interessante a ser feito é sobre o gerenciamento de dados e armazenamento duradouro, ou seja, investigar estratégias para lidar com desafios relacionados ao gerenciamento de dados em ambientes de microsserviços, exemplo, consistências, replicações e sincronizações de dados.

Referências

AWS. **O que são microsserviços? | AWS**. AWS, 2023. Disponível em: <https://aws.amazon.com/pt/microservices/>. Acesso em: 08 Jan. 2024. Citado na página 10.

BALTIERI, N. V. **Um experimento comparativo entre computação em nuvem e servidores físicos**, 2017.60 f. trabalho de conclusão de curso (curso de tecnologia em segurança da informação). Faculdade de Tecnologia de Americana, 2017. Citado na página 14.

BARROSO, L. A.; HöLZLE, U. **The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines**. San Rafael, CA: Morgan & Claypool Publishers, 2013. Citado na página 15.

BELAMARIC, J.; HOCKIN, T. **Networking and Kubernetes: A Layered Approach**. Newton, Massachusetts, EUA: O'Reilly Media, 2021. Citado na página 25.

BEVILAQUA, B. Suporte à federação de brokers mqtv via microsserviços: gerenciamento dinâmico da topologia da federação. Universidade Federal da Fronteira Sul, 2023. Disponível em: <https://rd.uffs.edu.br/bitstream/prefix/6573/1/BEVILAQUA.pdf>. Citado na página 27.

Broadcom. **VMware vSphere Documentação**. Irvine,Califórnia, 2024. Disponível em: <https://docs.vmware.com/br/VMware-vSphere/index.html>. Acesso em: 25 Março 2024. Citado na página 21.

CACIATO, L. E. Virtualização e consolidação dos servidores do datacenter. **Centro de Computaçao da Universidade Estadual de Campinas–Sao Paulo,(23)**, 2009. Citado na página 14.

Canonical Ltd. **Microk8s Documentationl**. Londres, Reino Unido, 2024. Disponível em: <https://microk8s.io/docs/tutorials>. Acesso em: 25 Dez. 2022. Citado 2 vezes nas páginas 11 e 33.

Canonical Ltd. **Multipass Documentationl**. Londres, Reino Unido, 2024. Disponível em: <https://multipass.run/docs>. Acesso em: 25 Dez. 2022. Citado 2 vezes nas páginas 22 e 33.

CARISSIMI, A. Virtualização: da teoria a soluções. **Minicursos do Simpósio Brasileiro de Redes de Computadores–SBRC**, v. 2008, p. 173–207, 2008. Disponível em: <https://jvasconcellos.com.br/wp-content/uploads/2012/01/cap4-v2.pdf>. Citado 2 vezes nas páginas 6 e 16.

COMAR, E. **Virtualização**. IXCSoft, 2024. Disponível em: <https://wiki.ixcsoft.com.br/pt-br/Servidores/Virtualiza%C3%A7%C3%A3o>. Acesso em: 28 Abril 2024. Citado 2 vezes nas páginas 5 e 21.

DRAGONI, N.; GIALLORENZO, S.; LAFUENTE, A. L.; MAZZARA, M.; MONTESI, F.; MUSTAFIN, R.; SAFINA, L. Microservices: yesterday, today, and tomorrow. **Present**

and ulterior software engineering, Springer, p. 195–216, 2017. Disponível em: <https://arxiv.org/pdf/1606.04036>. Citado 3 vezes nas páginas 11, 28 e 29.

FABIAN, P.; PALMER, J.; RICHARDSON, J.; BOWMAN, M.; BRETT, P.; KNAUERHASE, R.; SEDAYAO, J.; VICENTE, J.; KOH, C.-C.; RUNGTA, S. Virtualization in the enterprise. **Intel Technology Journal**, v. 10, n. 3, p. 227–242, 2006. Disponível em: <https://search.ebscohost.com/login.aspx?direct=true&db=bsx&AN=22445029&site=eds-live>. Citado 2 vezes nas páginas 15 e 17.

FIGUEREDO, L. R. R.; LUCCA, G. d. S. D. Virtualização e docker. **Revista Vincci-Periódico Científico do UniSATC**, v. 2, n. 2, p. 152–179, 2017. Disponível em: <https://revistavincci.satc.edu.br/index.php/Revista-Vincci/article/view/89>. Citado 3 vezes nas páginas 6, 15 e 16.

FILHO, M. F. Conceitos e infraestrutura de datacenters: livro digital. **Palhoça: Unisul Virtual**, 2016. Citado 2 vezes nas páginas 5 e 15.

FINN, A.; LOWNDS, P.; LUESCHER, M.; FLYNN, D. **Windows Server 2012 Hyper-V Installation and Configuration Guide**. [S.l.]: John Wiley & Sons, 2011. v. 47. Citado na página 21.

FOWLER, S. J. **Microsserviços prontos para a produção: Construindo sistemas padronizados em uma organização de engenharia de software**. [S.l.]: Novatec Editora, 2019. Citado 2 vezes nas páginas 10 e 17.

FREIRE, J. E. L. **Orquestração de Containers Usando Kubernetes e Docker Swarm**. Dissertação (Mestrado) — Universidade da Beira Interior (Portugal), 2021. Disponível em: <https://ubibliorum.ubi.pt/bitstream/10400.6/11091/1/7913_17373.pdf>. Citado 5 vezes nas páginas 5, 11, 22, 23 e 29.

GARCIA, G. B.; PAGANI, C. Do monolítico ao microserviço. **Instituto Federal de Educação, Ciência e Tecnologia de São Paulo**, Campus Hortolândia – SP – Brasil, Maio 2021. Citado 2 vezes nas páginas 27 e 47.

Google Cloud. **O que é o Kubernetes?** Google Cloud, 2024. Disponível em: <https://cloud.google.com/learn/what-is-kubernetes?hl=pt-br>. Acesso em: 17 Fev. 2024. Citado na página 22.

HABBEMA, H. **Kubernetes (K8S)**. Medium, 2023. Disponível em: <https://medium.com/@habbema/kubernetes-k8s-11cdce9e3a33>. Acesso em: 10 Março 2024. Citado 2 vezes nas páginas 5 e 25.

HARADA, E. **O que diferencia o uso de Containers e microsserviços? - TecMundo**. Tecmundo, 2021. Disponível em: <https://www.tecmundo.com.br/software/209032-diferencia-o-uso-containers-microsservicos.htm>. Acesso em: 28 Dez. 2022. Citado na página 19.

JÚNIOR, O. A. **Arquitetura de micro serviços: uma comparação com sistemas monolíticos**. Dissertação (Trabalho de Conclusão de Curso) — Universidade Federal da Paraíba, 2017. Disponível em: <https://repositorio.ufpb.br/jspui/bitstream/123456789/3235/1/OAJ14062017.pdf>. Citado na página 27.

JUSTINO, Y. d. L. **Do monolito aos microsserviços: um relato de migração de sistemas legados da Secretaria de Estado da Tributação do Rio Grande do Norte**. Dissertação (Mestrado) — Brasil, 2018. Citado 2 vezes nas páginas 10 e 47.

LIMA, L. R.; SANTOS, R. M. dos. Microsserviços: Arquitetura de software moderna. **Revista de Informática Aplicada**, v. 26, n. 1, 2019. Citado 2 vezes nas páginas 11 e 29.

LIMA, M. d. A. **Análise de soluções de rastreamento open source no contexto de aplicações baseadas em microsserviços. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação)**. Dissertação (B.S. thesis) — Universidade Federal de Pernambuco, Recife, 2022. Disponível em: <https://www.cin.ufpe.br/~tg/2022-1/tg_CC/tg_mal3.pdf>. Citado na página 19.

LOPES, T. R. **Método de migração de sistemas monolíticos legados para a arquitetura de microsserviços**. Dissertação (Trabalho de Conclusão de Curso) — Universidade de Brasília, 2021. Disponível em: <http://www.realp.unb.br/jspui/bitstream/10482/41178/1/2021_TaylorRodriguesLopes.pdf>. Citado 3 vezes nas páginas 11, 45 e 46.

LUCENA, M. A. et al. **Revisão sistemática da literatura: os desafios encontrados na migração de uma arquitetura monolítica para uma orientada a microsserviços.** Dissertação (Trabalho de Conclusão de Curso) — Universidade Federal de Campina Grande, 2021. Disponível em: <http://dspace.sti.ufcg.edu.br:8080/xmlui/bitstream/handle/riufcg/24979/MARIANA%20ARA%C3%9AJO%20LUCENA%20-%20TCC%20ARTIGO%20CI%C3%8ANCIA%20DA%20COMPUTA%C3%87%C3%83O%202021.pdf?sequence=1&isAllowed=y>. Citado na página 27.

MENDES, I. S. **Arquitetura monolítica vs microsserviços: uma análise comparativa**. Dissertação (Trabalho de Conclusão de Curso) — Universidade de Brasília – UnB, 2021. Disponível em: <https://bdm.unb.br/bitstream/10483/30715/1/2021_IasminSantosMendes_tcc.pdf>. Citado 3 vezes nas páginas 10, 27 e 48.

NADAREISHVILI, I.; MITRA, R.; MCLARTY, M.; AMUNDSEN, M. **Microservice architecture: aligning principles, practices, and culture**. Newton, Massachusetts, EUA: "O'Reilly Media, Inc.", 2016. Citado na página 12.

NATÁRIO, R. **Paravirtualização Explicada [2011]**. 2011. Disponível em: <https://redes-e-servidores.blogspot.com/2011/11/para-e-um-prefixo-de-origem-grega-que.html>. Acesso em: 08 Dez. 2022. Citado na página 16.

Oracle Corporation. **VirtualBox User Manual**. Santa Clara, Califórnia, EUA, 2024. v. 7. Disponível em: <https://download.virtualbox.org/virtualbox/UserManual.pdf>. Acesso em: 28 Dez. 2022. Citado na página 22.

PACHECO, D. R. P. **Escalabilidade horizontal automática de serviços utilizando o componente HPA do Kubernetes**. Dissertação (Trabalho de Conclusão de Curso) — Instituto Federal do Espírito Santo, 2023. Disponível em: <https://repositorio.ifes.edu.br/bitstream/handle/123456789/4416/TCC___Diego_Perez___Disponibilidade_com_Kubernetes_2023.pdf?sequence=1&isAllowed=y>. Citado na página 26.

PICCINI, M. **O que são servidores virtuais, como eles funcionam e qual sua importância? - Tigra Consult**. São Paulo, 2020. Disponível em: <https://tigraconsult.com.br/2020/10/12/servidores-virtuais/>. Acesso em: 09 Dez. 2022. Citado na página 17.

PINTO, D. F. F. **Plataforma de partilha de bicicletas altamente escalável através de uma arquitetura de micro-serviços**. Tese (Doutorado) — superior de tecnologia e gestão politécnico do porto, 2022. Disponível em: <https://recipp.ipp.pt/bitstream/10400.22/22502/1/DM_DanielPinto_MEI_2022.pdf>. Citado na página 27.

RAI, R. **Practical Introduction to Kubernetes Autoscaling Tools with Linode Kubernetes Engine**. The Cloud Blog, 2022. Disponível em: <https://thecloudblog.net/post/practical-introduction-to-kubernetes-autoscaling-tools-with-linode-kubernetes-engine/>. Acesso em: 22 Abril 2024. Citado 2 vezes nas páginas 5 e 44.

Red Hat. **O que é um pod no Kubernetes?** Red Hat, 2017. Disponível em: <https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes-pod>. Acesso em: 15 Nov. 2023. Citado na página 25.

Red Hat. **O que é orquestração de containers**. Red Hat, 2019. Disponível em: <https://www.redhat.com/pt-br/topics/containers/what-is-container-orchestration#orquestraÃğÃčo-de-containers-nas-empresas>. Acesso em: 15 Jan. 2024. Citado na página 20.

REDHAT, E. **Microsserviços | O que são microsserviços?** RedHat, 2023. Disponível em: <https://www.redhat.com/pt-br/topics/microservices#o-que-s%C3%A3o-microsservi%C3%A7os>. Acesso em: 08 Jan. 2024. Citado na página 10.

REHMAN, J. **O que é sistema clusterizado com exemplo?** IT Release, 2022. Disponível em: <https://www.itrelease.com/2022/01/what-is-clustered-system-with-example/>. Acesso em: 10 Março 2024. Citado 2 vezes nas páginas 5 e 24.

ROCHA, V. **Tipos de virtualização**. TI Especialistas - Desenvolvendo ideias, 2013. Disponível em: <https://www.tiespecialistas.com.br/tipos-de-virtualizacao/>. Acesso em: 08 Dez. 2022. Citado 2 vezes nas páginas 16 e 20.

SANTHOSH, R. **What is Docker, Difference between Docker and VM, Installation of Docker and its usage**. Techglimpse, 2017. Disponível em: <https://techglimpse.com/docker-installation-tutorial-centos/>. Acesso em: 05 Jan. 2023. Citado 3 vezes nas páginas 5, 18 e 19.

SOUZA, F. N. d. **Avaliação de desempenho de servidores de aplicação utilizando redes de Petri**. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2006. Citado na página 14.

SOUZA, W. D. C. et al. **Guia de orientações na migração de sistemas de informação do ambiente monolítico para ambiente de microsserviços na computação em nuvem em microempresas de software**. Dissertação (Trabalho de Conclusão de Curso) — Universidade Nove de Julho, 2023. Disponível em: <https://bibliotecatede.uninove.br/bitstream/tede/3245/2/Wilians%20Douglas%20Conde%20Souza.pdf>. Citado 4 vezes nas páginas 11, 45, 46 e 47.

TANENBAUM, A. S.; WETHERALL, D. J. **Computer Networks**. Londres, Reino Unido: Pearson Education, 2010. Citado 3 vezes nas páginas 24, 25 e 26.

The Kubernetes Authors. **Kubernetes Documentation**. Kubernetes, 2023. Disponível em: <https://kubernetes.io/docs/home/>. Acesso em: 17 Nov. 2023. Citado 4 vezes nas páginas 11, 22, 23 e 26.

VILLAÇA, L.; JR, A. F. P.; AZEVEDO, L. G. Construindo aplicações distribuídas com microsserviços. **Tópicos em Sistemas de Informação: Minicursos XV Simpósio Brasileiro de Sistemas de Informação. SBC**, 2018. Citado na página 18.