INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE PERNAMBUCO

***CAMPUS* BELO JARDIM**

BACHARELADO EM ENGENHARIA DE SOFTWARE

JOSÉ WEVERTON BARROS DA SILVA

CANARY DEPLOYMENT EM KUBERNETES COM ISTIO: um estudo de caso sobre implantações graduais e gerenciamento de tráfego

Belo Jardim - PE
2025

JOSÉ WEVERTON BARROS DA SILVA

CANARY DEPLOYMENT EM KUBERNETES COM ISTIO: um estudo de caso sobre implantações graduais e gerenciamento de tráfego

Trabalho de conclusão de Curso (TCC) apresentado como requisito parcial para obtenção do grau de Ba- charel em Engenharia de Software. Pri- meiro TCC aprovado no curso de Enge- nharia de Software do IFPE - Campus Belo Jardim

Orientador: Me. Elton Bezerra Torres

Belo Jardim - PE
2025

Dados Internacionais de Catalogação - CIP

S586c Silva, José Weverton Barros da.

Canary Deployment em Kubernetes com istio: um estudo de caso sobre implantações graduais e gerenciamento de tráfego / José Weverton Barros da Silva. – Belo Jardim-PE, 2025.

60f.: il.

Trabalho de Conclusão de Curso (Graduação de Bacharelado em Engenharia de Software) – Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco, Campus Belo Jardim - PE, 2025.

Orientador: Profº Me. Elton Bezerra Torres.

Inclui referências.

1. Engenharia de Software. 2.Roteamento de tráfego. 3. Canary Deployment. ntônio da. I. Torres, Elton Bezerra. II. Título. III. Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco.

CDD 005.1068

Ficha catalográfica elaborada pelo Sistema Integrado de Bibliotecas - SIBI/IFPE.
Bibliotecária: Fernanda de Oliveira Freitas Cavalcante CRB-4/PE - 2420

JOSÉ WEVERTON BARROS DA SILVA

CANARY DEPLOYMENT EM KUBERNETES COM ISTIO: um estudo de caso sobre implantações graduais e gerenciamento de tráfego

Trabalho aprovado. Belo Jardim, 26/02/2025.

Elton Bezerra Torres

Professor Orientador

Guilherme Cavalcanti da Silva

Convidado 1

Daniel Leite Viana

Convidado 2

AGRADECIMENTOS

Dedico este trabalho à minha família, à minha mãe, Márcia, à minha irmã, Clarissa, e, em especial, ao meu pai, Regi, que em julho de 2019 me apoiou quando tomei uma decisão completamente incerta. Incerteza essa que está sendo concretizada com este trabalho e que também me tirou daquela realidade.

Ao meu orientador, Elton Torres, que me mostrou o caminho que estou trilhando hoje. Sua orientação vai além deste trabalho; és uma inspiração para mim. Espero retribuir com muito orgulho no futuro e agradeço por me aguentar todo esse tempo.

Ao meu parceiro de orientação e amigo, Kaique Rierickson, foi uma honra compartilhar essa trajetória com todos os momentos e aprendizados que vivemos juntos. És um ás, e, caso precise, conte comigo em qualquer esfera da vida.

A todos os professores do curso com os quais compartilhei a sala de aula e que me deram toda a base de conhecimento para estar onde estou hoje, em especial a Hitalo Silva e Guilherme Cavalcanti.

Ao coordenador do curso, João Almeida, pois nada disso seria possível sem sua persis- tência e perseverança.

Aos meus amigos e pessoas especiais que conheci nessa trajetória, onde presenciei mi- lhares de espaços de tempo indeterminados que me fizeram enxergar que a vida vale a pena ser vivida.

RESUMO

Este trabalho apresenta a implementação e análise de uma estratégia de Canary Deploy- ment utilizando o Istio em um cluster Kubernetes. O Canary Deployment é uma técnica de implantação gradual que permite a liberação de novas versões de software de forma controlada, direcionando inicialmente uma pequena parcela do tráfego para a nova versão enquanto a maior parte permanece na versão estável. Essa abordagem reduz riscos asso- ciados a falhas e permite a validação da nova versão em produção antes de sua adoção completa (NEWMAN, 2015; RICHARDSON, 2018). O estudo foi conduzido com base na configuração de recursos do Istio, como Desti- nation Rules, Virtual Services e Gateways, para gerenciar o roteamento de tráfego entre duas versões de uma aplicação (v1 e v2). Foram explorados diferentes cenários de distri- buição de tráfego, iniciando com 90% para a versão estável (v1) e 10% para a nova versão (v2), evoluindo para uma divisão equilibrada de 50% para cada versão e, finalmente, di- recionando 100% do tráfego para a v2 após sua validação. Em caso de problemas, foi implementado um mecanismo de rollback rápido, redirecionando o tráfego de volta para a versão estável. Para monitorar e avaliar o desempenho das versões, foram utilizadas as ferramentas Prometheus e Kiali. O Prometheus forneceu métricas detalhadas sobre taxa de erro, latên- cia e throughput, enquanto o Kiali ofereceu uma visão gráfica da topologia da aplicação e do fluxo de tráfego entre os serviços. Essas ferramentas permitiram a detecção precoce de problemas e a tomada de decisões baseadas em dados, garantindo a estabilidade do sistema. Os resultados demonstraram que a estratégia de Canary Deployment, aliada ao uso do Istio, é eficaz para a implantação segura e gradual de novas versões em ambientes de mi- crosserviços. A capacidade de ajustar dinamicamente o tráfego e realizar rollbacks rápidos mostrou-se fundamental para minimizar impactos negativos e garantir a confiabilidade do sistema. Além disso, a integração com ferramentas de monitoramento proporcionou visi- bilidade e controle sobre o comportamento da aplicação em tempo real. Por fim, o trabalho sugere direções futuras para aprimorar a implementação, como a automação do processo de Canary Deployment, a integração com ferramentas de análise de logs, a realização de testes de carga mais abrangentes e a expansão para ambientes multi- cluster. Essas melhorias têm o potencial de aumentar ainda mais a eficiência, resiliência e escalabilidade do sistema, consolidando o Canary Deployment como uma prática essencial para a gestão de microsserviços em produção.

Palavras-chave: Canary Deployment, Istio, Kubernetes, Roteamento de Tráfego, Mi- crosserviços, Prometheus, Kiali, Monitoramento, Mecanismo de Rollback, Estabilidade do Sistema.

ABSTRSCT

This paper presents the implementation and analysis of a Canary Deployment strategy using Istio in a Kubernetes cluster. Canary Deployment is a gradual release technique that allows the controlled rollout of new software versions by initially directing a small portion of the traffic to the new version while the majority remains on the stable version. This approach reduces the risks associated with failures and enables the validation of the new version in production before full adoption (NEWMAN, 2015; RICHARDSON, 2018). The study was conducted based on Istio resource configuration, such as Destination Rules, Virtual Services, and Gateways, to manage traffic routing between two versions of an application (v1 and v2). Different traffic distribution scenarios were explored, starting with 90. To monitor and assess the performance of the versions, Prometheus and Kiali tools were used. Prometheus provided detailed metrics on error rates, latency, and throughput, while Kiali offered a graphical view of the application's topology and traffic flow between services. These tools enabled early problem detection and data-driven decision-making, ensuring system stability.The results demonstrated that the Canary Deployment strategy, combined with Istio, is effective for the safe and gradual deployment of new versions in microservices envi- ronments. The ability to dynamically adjust traffic and perform rapid rollbacks proved crucial in minimizing negative impacts and ensuring system reliability. Additionally, the integration with monitoring tools provided visibility and control over the application's behavior in real-time.Finally, the paper suggests future directions for improving the implementation, such as automating the Canary Deployment process, integrating with log analysis tools, con- ducting more extensive load testing, and expanding to multi-cluster environments. These improvements have the potential to further increase the efficiency, resilience, and scalabi- lity of the system, solidifying Canary Deployment as an essential practice for managing microservices in production.

Keywords: Canary Deployment, Istio, Kubernetes, Traffic Routing, Microservices, Prometheus, Kiali, Monitoring, Rollback Mechanism, System Stability.

LISTAS DE ABREVISTURSS

K8s	Kubernetes
CI	Continuous Integration - Integração Contínua
CD	Continuous Deployment - Deploy Contínuo
API	Application Programming Interface
AWS	Amazon Web Services
CRM	Customer Relationship Management
DevOps	Development and Operations
ERP	Enterprise Resource Planning
IaaS	Infrastructure as a Service
IBM	International Business Machines
Istio	Service Mesh para controle de tráfego em microsserviços
LAN	Local Area Network
OCI	Oracle Cloud Infrastructure
PaaS	Platform as a Service
SaaS	Software as a Service
RedHat	Empresa de software open source especializada em soluções de TI
VCN	Virtual Cloud Network
VPN	Virtual Private Network
WAN	Wide Area Network

LISTAS DE FIGURAS

1 INTRODUÇÃO

1.1 Contextualização

A entrega contínua (Continuous Delivery) tornou-se um pilar essencial em ambientes mo- dernos de desenvolvimento de software, permitindo que as equipes liberem novas funcio- nalidades de forma rápida e segura. Em um cenário onde microsserviços são amplamente adotados, a complexidade do gerenciamento de tráfego e a necessidade de garantir a estabi- lidade das implantações tornam-se desafios críticos. Nesse contexto, o Kubernetes emerge como uma plataforma de orquestração de contêineres líder, oferecendo escalabilidade e flexibilidade para gerenciar aplicações distribuídas. No entanto, o Kubernetes, por si só, possui limitações em relação ao controle granular de tráfego, especialmente em cenários de implantações progressivas, como o Canary Deployment (Red Hat, 2023).

É aqui que o Istio, um service mesh, se destaca. O Istio complementa o Kubernetes ao fornecer controle avançado de tráfego, observabilidade e segurança para microsserviços. Ele permite que os desenvolvedores implementem estratégias de implantação como o Canary Deployment de forma eficiente, garantindo que novas versões de uma aplicação sejam testadas com uma pequena porcentagem de usuários antes de uma liberação completa (Istio, 2023).

1.2 Problema de Pesquisa

A implementação de um Canary Deployment eficiente em um ambiente Kubernetes re- quer não apenas a configuração adequada de recursos como Deployments e Services, mas também a integração de ferramentas como o Istio para gerenciar o roteamento de tráfego de forma dinâmica. Além disso, o monitoramento contínuo de métricas e logs é essencial para identificar problemas e garantir a estabilidade da aplicação durante a transição en- tre versões. Diante disso, surge a questão: Como implementar e monitorar um Canary Deployment eficiente usando Istio em um cluster Kubernetes?

1.3 Objetivos

O objetivo geral deste trabalho é demonstrar como configurar e executar um Canary Deployment em um cluster Kubernetes utilizando o Istio. Para isso, os seguintes objetivos específicos foram definidos:

- Configurar um cluster Kubernetes funcional: Preparar um ambiente Kubernetes com os recursos necessários para suportar a implantação de microsserviços.

- Instalar e configurar o Istio para gerenciamento de tráfego: Integrar o Istio ao cluster Kubernetes e configurar regras de roteamento de tráfego para suportar o Canary Deployment.

- Demonstrar o Canary Deployment com análise de métricas e logs: Implementar um exemplo prático de Canary Deployment, utilizando ferramentas de observabilidade para monitorar o desempenho e a estabilidade da aplicação.

1.4 Justificativa

O Canary Deployment é uma estratégia fundamental para minimizar riscos em implan- tações de software, permitindo que novas versões sejam testadas com um subconjunto de usuários antes de uma liberação completa. Essa abordagem reduz o impacto de possíveis falhas e garante uma transição suave entre versões (Martin Fowler, 2023).

O Istio, por sua vez, desempenha um papel crucial nesse processo ao fornecer controle granular de tráfego, permitindo que os desenvolvedores definam regras de roteamento baseadas em pesos, cabeçalhos ou outras propriedades das requisições. Além disso, o Istio facilita a observabilidade, fornecendo métricas detalhadas e logs que ajudam a identificar problemas em tempo real (Istio, 2023).

A combinação de Kubernetes e Istio oferece uma solução robusta para a implementa- ção de Canary Deployments, tornando-a uma escolha relevante para equipes que buscam garantir a confiabilidade e a eficiência de suas implantações em ambientes de microsserviços.

2 FUNDAMENTAÇÃO TEÓRICA

Para entender melhor a implementação e a metodologia, é essencial explicar com mais detalhes alguns termos que estão guiando o trabalho, incluindo as técnicas e métodos utilizados, bem como as ferramentas empregadas no desenvolvimento deste.

2.1 DevOps

O conceito de DevOps emerge como uma resposta à necessidade de maior agilidade e integração entre as equipes de desenvolvimento e operações na tecnologia da informação. A palavra DevOps é uma junção dos termos *"Development"* (Desenvolvimento) e *"Ope- rations"* (Operações), representando uma abordagem que visa unir esses dois segmentos tradicionalmente distintos. Esta integração não se limita a uma fusão de responsabilida- des, mas envolve uma transformação cultural e metodológica profunda que busca otimizar e agilizar todo o ciclo de vida do desenvolvimento de software.

2.1.1 Origem e Filosofia

O termo DevOps foi formalmente introduzido por Patrick Debois em 2009, e sua criação foi influenciada pela crescente adoção de práticas ágeis no desenvolvimento de software. Essas práticas ágeis, por sua vez, têm raízes na filosofia Lean, que preconiza a eliminação de desperdícios e a maximização da eficiência. O pensamento Lean, que tem mais de um século de história, enfatiza a importância de eliminar processos desnecessários e otimizar continuamente as operações para alcançar uma maior agilidade e eficiência (Ebert et al., 2016; Ravichandran et al., 2016).

DevOps é, portanto, um conjunto de práticas, metodologias e técnicas que visam a automação e a integração das atividades relacionadas ao desenvolvimento e operações. Entre os principais benefícios do DevOps estão a redução do ciclo de desenvolvimento do software, a implementação de testes automatizados, atualizações contínuas e uma resposta mais rápida ao feedback dos usuários. Essas práticas contribuem para uma entrega de

software mais rápida e de maior qualidade, alinhando as expectativas dos clientes com a capacidade de resposta das equipes de desenvolvimento e operações (Chatterjee, 2021).

Além disso, o **DevOps** não pode ser reduzido a uma simples função ou cargo técnico; ele também incorpora uma transformação cultural dentro das organizações. Esta trans- formação envolve características como comunicação aberta, alinhamento de incentivos, responsabilidades compartilhadas e confiança mútua entre as equipes. Estes aspectos cul- turais são considerados facilitadores que suportam um conjunto de práticas e processos de engenharia voltados à automação e à integração contínua, fundamentais para a imple- mentação eficaz do **DevOps** (Chatterjee, 2021; Senapathi & Jim Buchan, 2018). Embora a cultura seja um componente essencial, sozinha não define o **DevOps**. Ela serve como uma base para permitir a fluidez e a eficiência dos processos de engenharia.

2.1.2 Definição e Benefícios

Os benefícios dessa abordagem vão além da simples aceleração das entregas. A prática de **DevOps** contribui para uma maior resiliência nas operações, uma redução no tempo de resposta a mudanças e uma melhor gestão de feedback dos usuários, o que permite ajustes contínuos e melhorias no software (Ravichandran et al., 2016). Contudo, os desafios da adoção dessa metodologia ainda são significativos, incluindo a complexidade técnica e as barreiras culturais dentro das organizações.

Portanto, o **DevOps** pode ser entendido como um "conjunto de recursos de processos de engenharia suportados por facilitadores culturais e tecnológicos", onde esses recursos definem os processos que as organizações devem ser capazes de executar de maneira ágil, flexível e eficiente (Senapathi & Jim Buchan, 2018). Essa definição será usada como guia para investigar a implementação do **DevOps** na prática real, reconhecendo que a expe- riência adquirida em contextos práticos é inestimável e poucas pesquisas detalham essas aplicações em profundidade.

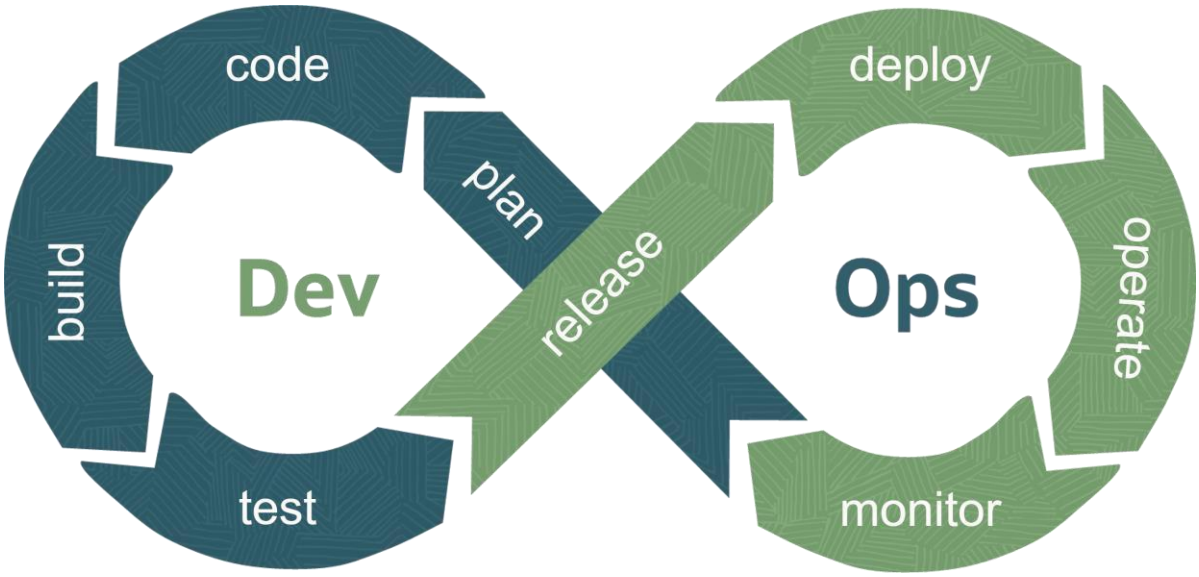
2.1.3 Ciclo DevOps

Segundo Bass et al. (2015) O ciclo DevOps é composto por várias etapas interligadas que visam acelerar o processo desde o desenvolvimento até a entrega do software ao usuário final. Essas etapas incluem:

Cada uma dessas fases apresentadas possuem ferramentas e/ou recursos específicos que priorizam a otimização dos resultados de cada etapa e auxiliam na implementação da estratégia. As quatro primeiras fases dizem respeito ao Dev, a parte da codificação em si:

- **Plan** — Esta etapa é crucial para a definição e coleta de requisitos dos clientes. Envolve a interação contínua com os *stakeholders* para entender suas necessidades e expectativas, e o planejamento de como essas necessidades serão atendidas ao longo do ciclo de desenvolvimento.

Figura 2.1: Fluxo de trabalho contínuo DevOps



(a) Fonte: Oracle Help Center

- **Code** — Nessa fase, os requisitos identificados durante o planejamento são transformados em código utilizando linguagens de programação específicas. É comum que sejam realizadas interações frequentes com sistemas de versionamento de código-fonte, e testes unitários são realizados durante a compilação para garantir a qualidade dos artefatos gerados.

- **Build** — Após o desenvolvimento, o código é submetido a testes automatizados para verificar se atende aos critérios de qualidade e funcionalidade estabelecidos. Esses testes podem ocorrer tanto durante quanto após o desenvolvimento.

- **Test** — A fase de teste envolve a validação do artefato gerado em um ambiente de testes que simula o ambiente de produção. O objetivo é garantir que o software atenda a todos os requisitos dos clientes antes da sua implementação no ambiente produtivo.

- **Implementação** — Nesta etapa, o software é publicado no ambiente de produção. Esta etapa pode envolver a configuração de ambientes e a realização de ajustes finais para garantir uma implantação suave.

- **Operação**: Uma vez que o software está em produção, é necessário manter e melhorar o ambiente onde ele opera. Isso inclui o planejamento de escalonamento automático, a gestão de módulos e dependências, e a garantia de que o sistema esteja sempre disponível e funcional.

- **Monitoramento**: A última etapa do ciclo DevOps é o monitoramento contínuo do software em produção. Isso envolve a coleta de logs e a geração de relatórios de per-

formance para identificar e corrigir falhas rapidamente, além de fornecer informações valiosas para o planejamento do próximo ciclo de desenvolvimento.

2.1.4 Componentes e Práticas

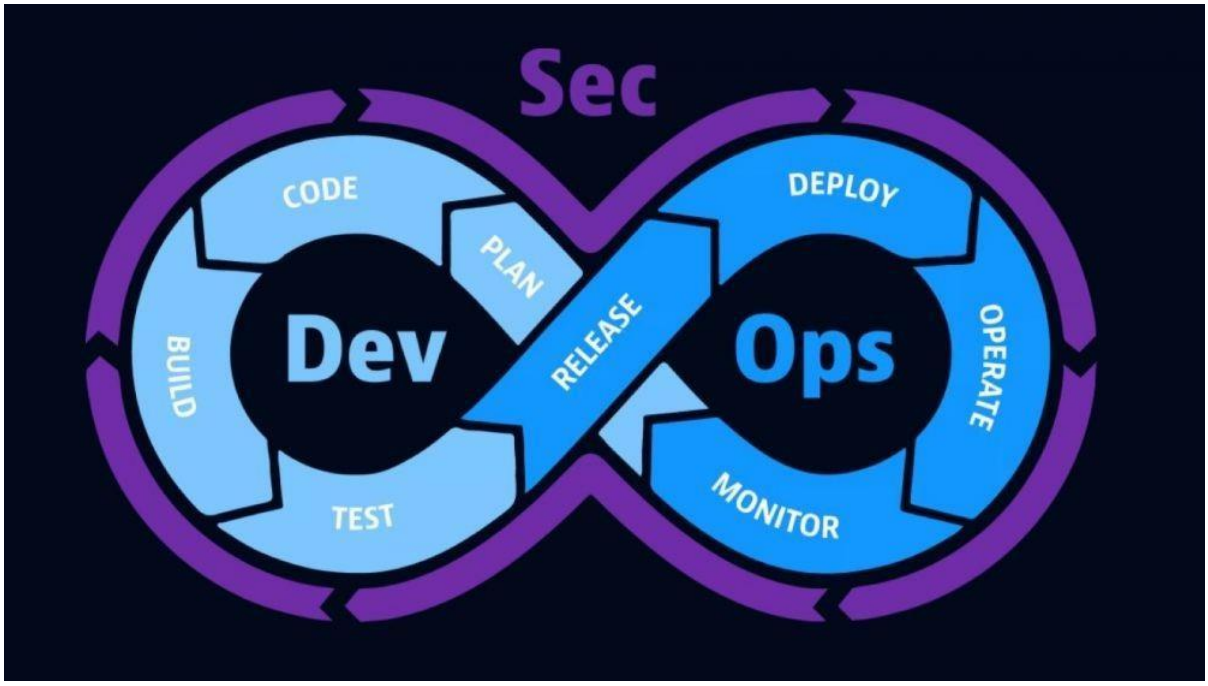
A implementação bem-sucedida do DevOps depende da integração de vários componentes fundamentais. Esses componentes são essenciais para a criação de uma cultura DevOps eficaz e incluem (Wiedemann et al., 2019):

- **Componente Cultural** — Promover um ambiente de confiança entre os membros da equipe, incentivar o aprendizado contínuo e a abertura para mudanças são aspectos cruciais. A cultura DevOps deve valorizar a colaboração e a disposição para a melhoria contínua.
- **Componente de Automação** — A automação é central para a prática do DevOps, principalmente na criação de pipelines para Integração e Entrega Contínua. Isso permite a execução automática de tarefas repetitivas e a manutenção de um fluxo de trabalho eficiente.
- **Componente de Simplificação** — Visa melhorar e otimizar os processos de desenvolvimento e operações, evitando gargalos e melhorando a eficiência. A simplificação ajuda a criar um ambiente mais ágil e adaptável.
- **Componente de Medição** — Inclui a medição e o monitoramento das métricas do sistema, bem como dos indicadores de desempenho. A capacidade de medir e avaliar o desempenho é essencial para identificar áreas de melhoria e garantir que os objetivos sejam alcançados.
- **Componente de Compartilhamento de Conhecimento** — Incentiva a comunicação e o compartilhamento de conhecimento entre os membros da equipe e entre diferentes setores. A colaboração e a troca de informações são fundamentais para resolver problemas e promover a inovação. produção, é necessário manter e melhorar o ambiente onde ele opera. Isso inclui o planejamento de escalonamento automático, a gestão de módulos e dependências, e a garantia de que o sistema esteja sempre disponível e funcional.

Entre as práticas populares do DevOps estão a Integração Contínua (CI), a Entrega Contínua (CD), a Infraestrutura como Código (IaC), o uso de Microsserviços e a Centralização de Logs (AWS, 2021). Essas práticas impactam positivamente a cultura organizacional e os processos de desenvolvimento e entrega de software, permitindo uma resposta mais ágil às necessidades do mercado e melhorando a qualidade do software.

2.2 DevSecOps

Figura 2.2



(a) Fonte: Dynatrace

O conceito de **DevSecOps** é uma evolução do **DevOps**, com o objetivo de integrar práticas de segurança diretamente no ciclo de vida do desenvolvimento de software. Ao contrário das abordagens tradicionais, nas quais a segurança era tratada como uma etapa final, o **DevSecOps** preconiza a inclusão da segurança desde o início do processo de de- senvolvimento, garantindo que tanto a aplicação quanto a infraestrutura sejam protegidas continuamente. Além disso, ele visa automatizar barreiras de segurança de forma a não interromper o fluxo ágil de desenvolvimento, que é a essência do DevOps (Red Hat, 2023). Para atingir essas metas, a escolha das ferramentas adequadas é essencial. Por exemplo, integrar um **ambiente de desenvolvimento integrado (IDE)** com funcionalidades de segurança permite que os desenvolvedores identifiquem vulnerabilidades em tempo real, corrigindo-as antes que causem problemas no ambiente de produção. No entanto, o **DevSecOps** vai além de apenas implementar novas ferramentas; ele se baseia em uma mudança cultural que busca integrar as equipes de segurança no ciclo de desenvolvimento desde o início, promovendo colaboração entre desenvolvimento, operações e segurança. Isso permite uma abordagem mais ágil e eficiente para o gerenciamento de riscos, evitando que
a segurança se torne um gargalo no processo (Red Hat, 2023).

A importância do **DevSecOps** se torna ainda mais evidente à medida que as empresas produzem software em um ritmo cada vez mais acelerado e em ambientes de

TI cada vez mais complexos. Tecnologias nativas da nuvem, como microsserviços, containers e arquiteturas sem servidor, oferecem flexibilidade e escalabilidade, mas também introduzem novos desafios de segurança. Políticas de segurança estáticas já não são suficientes para esses ambientes dinâmicos, e a probabilidade de falhas de segurança aumenta conforme as arquiteturas de software se tornam mais complexas (Elastic, 2023).

Além disso, a necessidade de entregar software mais rapidamente em um cenário de crescente preocupação com a segurança cibernética exige que o DevOps se adapte, tornando o **DevSecOps** essencial. Ao integrar a segurança diretamente no ciclo de desenvolvimento, as empresas conseguem mitigar riscos mais rapidamente, garantindo tanto a conformidade quanto a segurança proativa de seus produtos. Isso não apenas otimiza o processo de desenvolvimento, mas também sinaliza aos clientes que a segurança é uma prioridade central, o que pode aumentar a confiança e a satisfação do cliente, refletindo positivamente nos resultados da empresa (Elastic, 2023).

2.3 Containers vs Maquinas Virtuais

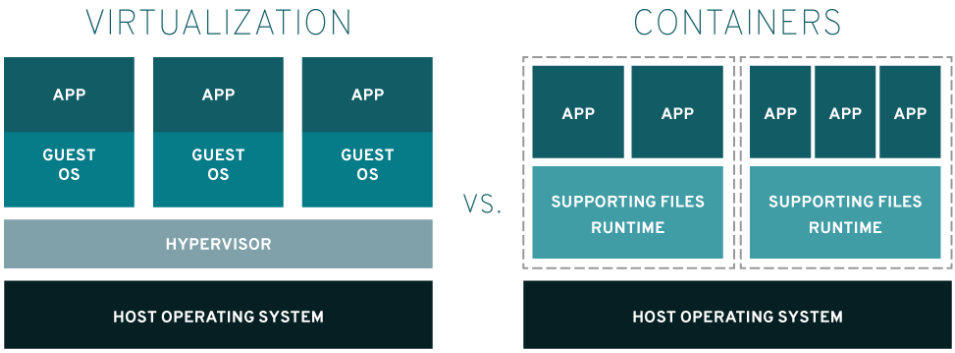
Os Containers e máquinas virtuais (VMs) são métodos para encapsular ambientes de com- putação. Ambas incluem diversos componentes de TI e mantêm esses ambientes separados do restante do sistema. A diferença essencial entre as duas reside nos elementos que são isolados, influenciando a capacidade de escalonamento e a portabilidade de cada método. "Embora containers e VMs tenham características distintas e únicas, eles são semelhantes no fato de ambos melhorarem a eficiência de TI, fornecerem portabilidade de aplicações e aprimorarem o DevOps e o ciclo de vida do desenvolvimento de software "(IBM Cloud Team, I. C. (2021)).

Um container é uma unidade padrão de software que empacota o código e todas as suas dependências para que a aplicação seja executada de maneira rápida e confiável de um ambiente de computação para outro. Uma imagem de container Docker é um pacote de software leve, autônomo e executável que inclui tudo o que é necessário para executar uma aplicação: código, runtime, ferramentas do sistema, bibliotecas do sistema e configurações. "As imagens de containers se tornam containers em tempo de execução e, no caso dos containers Docker, as imagens se tornam containers quando são executadas no Doc- ker Engine. Disponível tanto para aplicações baseadas em Linux quanto em Windows, o software containerizado sempre funcionará da mesma maneira, independentemente da infraestrutura. Containers isolam o software de seu ambiente e garantem que ele funcione uniformemente, apesar das diferenças, por exemplo, entre desenvolvimento e staging." (Docker Inc. (2020). What is a Container?).

As VMs por sua vez é um computador, assim como um computador físico, executa um sistema operacional e aplicações. A máquina virtual é composta por um conjunto de arquivos de especificações e configurações e é suportada pelos recursos físicos de um

host. Cada máquina virtual possui dispositivos virtuais que oferecem a mesma funcionalidade que o hardware físico, além de benefícios adicionais em termos de portabilidade, gerencia- bilidade e segurança. Uma máquina virtual consiste em vários tipos de arquivos que você armazena em um dispositivo de armazenamento suportado. Os arquivos principais que compõem uma máquina virtual são o arquivo de configuração, o arquivo de disco virtual, o arquivo de configuração da NVRAM e o arquivo de log (Holbrook, J. (2022). What is a Virtual Machine. In VMware Virtualization Fundamentals).

Figura 2.3: Virtualization vs Containers



(a) https://www.redhat.com/rhdc/managed-files/styles/wysiwyg$_f$ *ull$_w$idth/private/virtualization−vs − containers$_t$ransparent.png.webp*?*itok = LQxFhKKx*

2.4 Kubernetes e Orquestração de Contêineres

Nos últimos anos, a contêinerização emergiu como uma técnica revolucionária no desen- volvimento de software. As práticas tradicionais, que dependiam de ambientes de virtu- alização complexos e configurações de sistemas locais, mostraram-se limitadas frente ao crescimento da computação em nuvem e às demandas por maior escalabilidade. Nesse contexto, o Docker se destacou como uma tecnologia que simplifica o empacotamento, a distribuição e a execução de aplicações em contêineres. Contêineres são unidades de software que encapsulam o código de uma aplicação, suas dependências e configurações, tornando-as portáveis e independentes de um sistema operacional específico, assegurando que aplicações operem de forma consistente em qualquer ambiente, seja de desenvolvi- mento, teste ou produção (Red Hat, 2022).

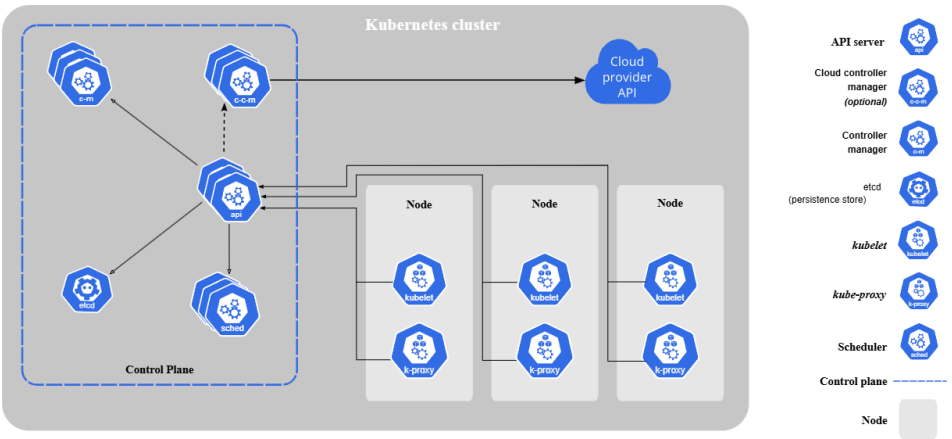
À medida que o uso de contêineres cresceu, surgiu a necessidade de um sistema que fosse além do empacotamento e execução de contêineres isolados. Em cenários de larga escala, como sistemas distribuídos e arquiteturas de microsserviços, torna-se essencial gerenciar o ciclo de vida dos contêineres, escalar suas instâncias e monitorá-las de maneira centralizada. O Kubernetes, plataforma de código aberto originalmente desenvolvida pelo Google, atende a essas necessidades ao oferecer recursos de orquestração para contêineres.

O Kubernetes permite automatizar operações diárias, abstrair a infraestrutura subja- cente e monitorar a integridade dos serviços em execução. Comandos nativos da plata- forma permitem aos operadores implementar políticas de gerenciamento automatizado, garantindo que as aplicações estejam sempre disponíveis e operando conforme o desejado (Google Cloud, s.d.). Além disso, a plataforma oferece suporte a funcionalidades como a reinicialização automática de contêineres que falham, a distribuição de cargas de trabalho de forma equilibrada e o ajuste de escala conforme a demanda.

Apesar de serem frequentemente comparados, Docker e Kubernetes são tecnologias complementares e não concorrentes. Enquanto o Docker é amplamente utilizado para a criação e o empacotamento de contêineres, o Kubernetes se destaca por seu papel na orquestração desses contêineres em ambientes de produção. O Docker facilita a portabi- lidade, uma vez que encapsula todas as dependências e configurações necessárias em um único pacote, que pode ser executado em qualquer sistema compatível com contêineres. No entanto, o Docker por si só não oferece soluções nativas para o gerenciamento de múltiplos contêineres em grande escala.

O Kubernetes, por outro lado, é essencial para o gerenciamento, monitoramento e escalabilidade de contêineres em produção. Com ele, as equipes de desenvolvimento e operações podem gerenciar cargas de trabalho em clusters complexos, assegurando que os aplicativos estejam sempre disponíveis e respondendo de forma eficiente às demandas. Em outras palavras, o Kubernetes age como um "capitão"que coordena a execução dos contêineres e garante sua disponibilidade e performance contínua (Google Cloud, s.d.).

Figura 2.4: Components of Kubernetes



(a) Fonte: https://kubernetes.io/docs/concepts/overview/components/

A implementação do Kubernetes traz diversos benefícios, que vão desde a simplificação de operações até a confiabilidade no ambiente de produção. Entre os principais benefícios da adoção dessa tecnologia, destacam-se:

- Automação das Operações: O Kubernetes possui comandos e ferramentas que per-

mitem automatizar tarefas comuns, como a implantação de novos contêineres, reinício automático em caso de falhas e balanceamento de carga. Isso reduz o trabalho manual e aumenta a eficiência das operações.

- Abstração de Infraestrutura: O Kubernetes abstrai a infraestrutura subjacente, permitindo que os desenvolvedores se concentrem nas aplicações sem se preocupar com detalhes específicos da configuração do ambiente.
- Monitoramento Contínuo da Integridade: A plataforma executa verificações constantes para monitorar a integridade dos serviços e contêineres, garantindo que somente os componentes em pleno funcionamento estejam disponíveis para os usuários.

Embora os benefícios do Kubernetes sejam amplamente reconhecidos, sua adoção apre- senta desafios específicos. Implementar uma solução de orquestração robusta requer conhe- cimentos técnicos especializados e um planejamento detalhado da arquitetura do sistema. Além disso, a configuração inicial do Kubernetes pode ser complexa, especialmente em am- bientes de produção que envolvem redes distribuídas e políticas avançadas de segurança. Para pequenas empresas ou desenvolvedores individuais, o investimento inicial de tempo e recursos pode ser uma barreira, justificando a análise cuidadosa do custo-benefício antes de sua implementação (IBM, s.d.).

Docker e Kubernetes, embora possuam papéis distintos, juntos oferecem uma solução poderosa para o desenvolvimento e implantação de aplicações modernas. Enquanto o Docker garante portabilidade e consistência, o Kubernetes assegura escalabilidade e resiliência, criando uma base sólida para arquiteturas baseadas em microsserviços e sistemas distribuídos. A sinergia entre essas tecnologias permite não só um desenvolvimento ágil, mas também um ambiente de produção confiável, onde as operações automatizadas e o monitoramento contínuo da integridade são fundamentais para a entrega de soluções de alta qualidade.

2.5 Computação em nuvem

A computação em nuvem representa um avanço significativo na forma como os recursos de tecnologia da informação (TI) são gerenciados e provisionados. Ela permite o acesso a recursos de computação, como servidores, armazenamento e bancos de dados, através da internet, possibilitando que empresas e usuários finais utilizem recursos computacionais sob demanda, sem a necessidade de investimentos em infraestrutura física (Rittinghouse; Ransome, 2017). Essa abordagem não apenas reduz custos iniciais, mas também elimina a necessidade de treinamento de funcionários para gerenciar novos sistemas ou licenciar softwares adicionais (Knorr; Gruman, 2008).

Todas as nuvens, independentemente de seu tipo, compartilham a característica fun- damental de extrair, agrupar e compartilhar recursos escaláveis em uma rede. Elas viabilizam a execução de cargas de trabalho, utilizando um conjunto de tecnologias que frequentemente inclui um sistema operacional, plataformas de gerenciamento e interfaces de programação de aplicações (Apis). Adicionalmente, é comum a implementação de aplicações de virtualização e automação, que aumentam a eficiência e a oferta de recursos em todos os tipos de nuvem (Ghahramani; Zhou; Hon, 2017; Red Hat, 2023).

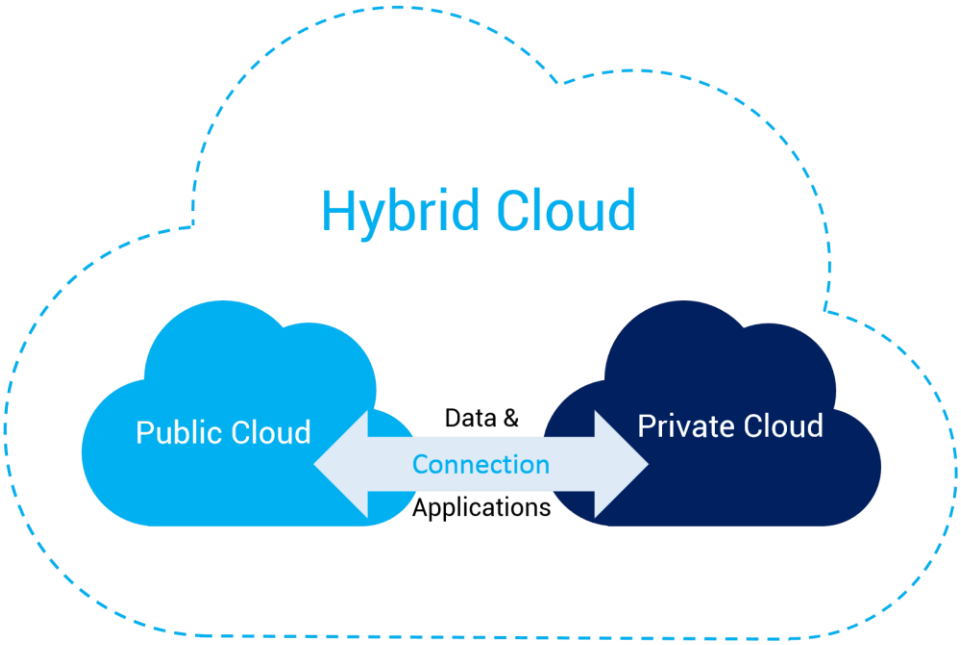
2.5.1 Diferenças entre os Tipos de Nuvem

Historicamente, a distinção entre nuvens públicas, privadas, híbridas e multiclouds era definida por critérios de local e propriedade. No entanto, essa abordagem se tornou mais complexa com a evolução das tecnologias de nuvem.

2.5.2 Diferenças entre os Tipos de Nuvem

Historicamente, a distinção entre nuvens públicas, privadas, híbridas e multiclouds era definida por critérios de local e propriedade. No entanto, essa abordagem se tornou mais complexa com a evolução das tecnologias de nuvem.

Figura 2.5: Serviços de nuvem



(a) Fonte: Alibaba Cloud

Nuvem Pública:

As nuvens públicas são ambientes que utilizam uma infraestrutura de TI não pertencente ao usuário final, sendo operadas por grandes provedores como Alibaba Cloud,

Amazon Web Services (Aws), Google Cloud, Ibm Cloud e Microsoft Azure. Tradicionalmente, esses serviços eram oferecidos em data centers off-premises, mas atualmente muitos provedores também oferecem soluções que utilizam infraestrutura on-premise dos clientes, tornando as distinções de local e propriedade obsoletas. (Red Hat, 2023).

Nuvem Privada:

As nuvens privadas são ambientes dedicados a um único usuário final e geralmente operam atrás de firewalls. A definição de nuvem privada não se limita mais a infraes- truturas on-premise; muitas organizações agora criam nuvens privadas em data centers alugados off-premises. Dessa forma, as regras tradicionais de local e propriedade tornam- se menos relevantes, refletindo uma evolução nas práticas de implementação de nuvens privadas (Red Hat, 2023).

Nuvem Híbrida:

Uma nuvem híbrida é um ambiente de TI que combina múltiplos outros ambientes conectados por redes locais (Lans), redes de longa distância (Wans), redes privadas virtuais (Vpns) ou Apis. As características de uma nuvem híbrida podem variar, mas geralmente incluem uma combinação de nuvens públicas e privadas, permitindo que as aplicações se movam entre diferentes ambientes interconectados. Para que um sistema de TI seja considerado híbrido, deve haver uma consolidação de recursos que permita escalabilidade sob demanda, com uma plataforma integrada de gerenciamento e orquestração (Red Hat, 2023).

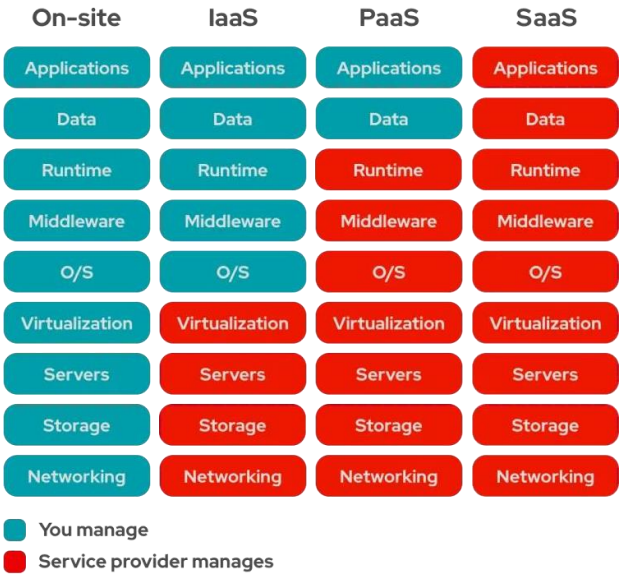
A escalabilidade é uma das melhorias mais notáveis que a computação em nuvem ofe- rece em relação às infraestruturas on-premise. Ela permite que as empresas aumentem ou diminuam rapidamente os recursos computacionais disponíveis para atender à demanda em constante mudança. Essa flexibilidade, além de resultar em um custo de manutenção menor, também proporciona ganho de desempenho e uma melhor disponibilidade de re- cursos. Além disso, a mobilidade dos serviços em nuvem facilita o acesso e a utilização de dados de qualquer lugar, o que se tornou um fator crucial para as operações empresariais modernas (Rittinghouse; Ransome, 2017).

Essas distinções ressaltam a flexibilidade e a adaptabilidade que as soluções de com- putação em nuvem oferecem às empresas, permitindo que escolham a melhor abordagem para suas necessidades operacionais e de segurança.

2.5.3 Modelos de Nuvem

A computação em nuvem revolucionou a maneira como as empresas utilizam e geren- ciam seus recursos de TI. Ela permite o acesso sob demanda a serviços de infraestrutura, plataformas de desenvolvimento e softwares completos por meio da internet, eliminando a necessidade de gerenciar esses elementos localmente. Dentre os principais modelos de serviços em nuvem, destacam-se o IaaS (Infrastructure as a Service), PaaS (Platform as a Service) e SaaS (Software as a Service), cada um com

Figura 2.6: Serviços de nuvem



(a) FONTE: Redhat

diferentes níveis de responsabilidade e controle por parte do usuário.

IaaS (Infrastructure as a Service)

O IaaS, ou Infraestrutura como Serviço, é o modelo mais básico de computação em nuvem, oferecendo recursos de infraestrutura, como servidores virtuais, armazenamento, redes e capacidade de processamento. Nesse modelo, o usuário é responsável por gerenciar o sistema operacional, os dados e os aplicativos, enquanto o provedor de serviços cuida da manutenção da infraestrutura subjacente, como hardware e redes. Dessa forma, as empresas têm maior controle e flexibilidade, podendo configurar seus próprios ambientes de TI de acordo com suas necessidades específicas (IBM, s.d.).

Por exemplo, uma empresa pode usar IaaS para hospedar seus servidores virtuais e ter total controle sobre o sistema operacional, middleware e aplicativos, sem precisar lidar com o gerenciamento do hardware. Isso possibilita uma escalabilidade eficiente, já que os recursos podem ser ajustados conforme a demanda do negócio aumenta ou diminui.

PaaS (Platform as a Service)

Já o PaaS, ou Plataforma como Serviço, é um modelo que fornece uma plataforma com- pleta para o desenvolvimento, teste e implantação de aplicativos na nuvem. O objetivo do PaaS é simplificar o trabalho dos desenvolvedores, eliminando a necessidade de gerenciar a infraestrutura subjacente, como servidores e sistemas operacionais,

permitindo que eles se concentrem exclusivamente na criação e manutenção de aplicativos (Google Cloud, s.d.).

Esse modelo é particularmente vantajoso para equipes de desenvolvimento que preci- sam de um ambiente flexível e escalável para criar e testar seus aplicativos sem se preocu- par com a manutenção de hardware. As empresas podem reduzir o tempo de lançamento de novos produtos, pois todo o ambiente de desenvolvimento é fornecido e gerenciado pelo provedor de serviços. Além disso, o PaaS permite a integração com serviços de banco de dados, ferramentas de monitoramento e escalabilidade automática, conforme as necessi- dades de cada projeto.

SaaS (Software as a Service)

Por fim, o SaaS, ou Software como Serviço, oferece softwares completos acessíveis via internet. Nesse modelo, o usuário final não precisa se preocupar com a instalação, manu- tenção ou atualização do software, pois tudo é gerenciado pelo provedor do serviço. Isso permite que as empresas usem soluções prontas para diversas necessidades, como geren- ciamento de relacionamento com clientes (CRM), sistemas de gestão empresarial (ERP) ou ferramentas de comunicação (TOTVS, s.d.).

O SaaS tem se tornado o modelo preferido para muitas organizações, pois oferece conveniência e reduz custos operacionais, uma vez que não é necessário investir em in- fraestrutura para rodar o software. Além disso, os fornecedores de SaaS garantem que as soluções estejam sempre atualizadas e seguras, com correções de bugs e novos recursos sendo implementados regularmente.

Os modelos IaaS, PaaS e SaaS são exemplos claros de como a computação em nuvem tem transformado a forma como as empresas gerenciam e utilizam seus recursos de TI. Cada modelo oferece um nível diferente de controle e flexibilidade, atendendo a diversas ne- cessidades organizacionais. Empresas que buscam maior controle podem optar pelo IaaS, enquanto aquelas focadas em desenvolvimento podem preferir o PaaS. Já o SaaS é ideal para empresas que desejam utilizar soluções prontas e totalmente gerenciadas.

2.6 Continuous Integration, Continuous Delivery e Con- tinuous Deployment

Esta seção explica brevemente os conceitos de Continuous Integration, Continuous Deli- very e Continuous Deployment.

Continuous Integration (CI)

De acordo com Shahin, Ali Babar, & Zhu (2017), Continuous Integration (integração contínua) é uma boa prática de DevOps que consiste em garantir uma rotina de com- pilação e testes automatizada que é executada sempre que algum código é mesclado

na branch principal do repositório. Dessa forma, as equipes de desenvolvimento mitigam os desafios relacionados à integração de código. RedHat (2020) explica que uma CI bem-sucedida ocorre quando novas mudanças no código de uma aplicação são desenvolvidas, testadas e consolidadas regularmente em um repositório compartilhado. Essas mudanças são então validadas através da criação automática da aplicação, com testes automatizados, geralmente de unidade e integração, para garantir que as mudanças não corrompam a aplicação.

Os custos associados à adoção da prática de CI por um time de desenvolvimento incluem:

- A necessidade de escrever testes automatizados para cada nova feature, melhoria ou correção de bug;
- A necessidade de um servidor de integração contínua que monitore constantemente a branch principal do repositório e execute os testes sempre que um novo commit for feito nesta branch (Shahin, Ali Babar, & Zhu, 2017).

Continuous Delivery (CD)

Continuous Delivery (entrega contínua) é a prática de automatizar os testes e o envio das mudanças de código para um repositório, de forma que posteriormente elas possam ser lançadas em produção pelo time de operações ou por alguém com esse nível de permis- são. Essa prática é uma extensão da integração contínua, já que os testes automatizados também são uma parte fundamental da CD (Shahin, Ali Babar, & Zhu, 2017).

A Amazon (2021) explica que a entrega contínua expande a integração contínua, im- plantando todas as alterações de código em um ambiente de teste ou ambiente de produção após o estágio de criação. RedHat (2020) complementa explicando que, após realizar a automação de compilação e os testes de unidade e integração na CI, a entrega contínua automatiza o lançamento desse código validado em um repositório.

Os custos para a adoção da prática de CD por um time de desenvolvimento incluem:

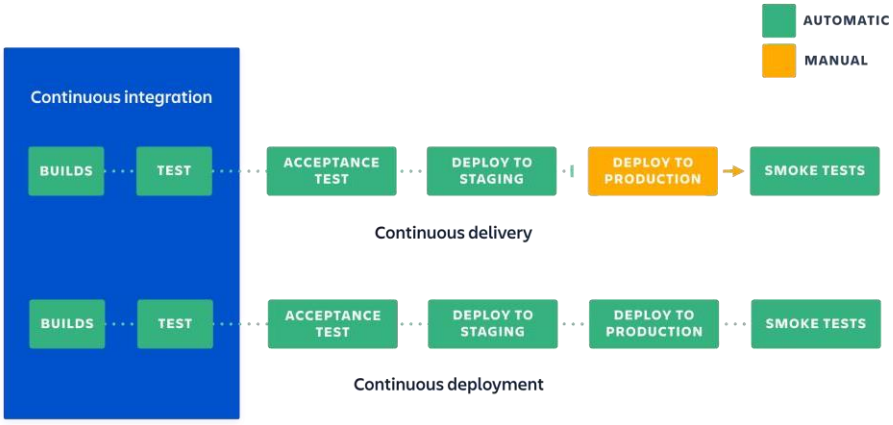
- Uma boa implementação da prática de CI;
- A implementação de algum botão capaz de disparar um processo de implantação automática em ambiente de produção;
- A implementação de feature flags, para que features específicas possam ser acessíveis em produção apenas quando completamente desenvolvidas (Atlassian, 2020).

Continuous Deployment (CD)

Continuous Deployment (Implantação Contínua) é praticamente igual à prática de Con- tinuous Delivery, com a diferença de que o processo é inteiramente automatizado. Shahin, Ali Babar, & Zhu (2017) explicam que, enquanto no Continuous Delivery há um comando manual para o deploy ocorrer, no Continuous Deployment, a compilação, testes e implan- tação são realizados sempre que um commit é feito na branch principal, sem intervenção humana. RedHat (2020) afirma que a implantação contínua automatiza o lançamento de compilações prontas para produção em um repositório de códigos.

Deploy contínuo, como explica Faraday (2019), é uma prática de desenvolvimento de software na qual cada alteração de código passa por todo o pipeline e é colocada em produção automaticamente, resultando em muitas implantações de produção todos os dias. Isso depende muito da automação otimizada dos testes realizados.

Figura 2.7: Pipeline de CI/CD



(a) FONTE: Atlassian

2.7 Estrategias de Deploy

As estratégias de deploy são cruciais para assegurar a entrega de software de maneira segura, eficiente e com o mínimo impacto no ambiente de produção. No contexto de prá- ticas modernas, como DevOps, essas estratégias permitem maior flexibilidade e agilidade no processo de entrega contínua, ao mesmo tempo em que reduzem os riscos de falhas em ambientes críticos (RedHat, 2022).

Deploy Recriado

Uma abordagem tradicional, embora simples, é o deploy recriado, no qual a aplicação antiga é completamente substituída por uma nova versão após a interrupção do serviço. Esse método, apesar de direto, apresenta a desvantagem de gerar períodos de indisponi- bilidade e pode não ser viável para sistemas que demandam alta disponibilidade (AWS, 2022).

Blue-Green

Por outro lado, estratégias como o blue-green deploy se destaca por mitigar riscos durante a transição. O modelo blue-green envolve a manutenção de dois ambientes de produção idênticos: o ambiente atual (blue), que atende aos usuários, e um novo ambiente (green), onde a nova versão é implementada e testada. Após a validação, todo o tráfego é redireci- onado para o ambiente green, tornando-o a nova versão em produção (Fowler, 2010). Essa abordagem garante que o ambiente antigo permaneça como fallback, permitindo rollback instantâneo em caso de falhas.

A principal vantagem do blue-green deploy está na simplicidade do rollback e na garantia de que a transição para a nova versão ocorre de forma transparente para os usuários. Entretanto, essa estratégia exige infraestrutura duplicada, o que pode gerar custos adicionais, especialmente em sistemas com recursos computacionais elevados ou grandes bancos de dados que precisam ser sincronizados entre os ambientes (Amazon, s.d.).

Canary Deploy

Entre as várias abordagens, o deploy canário destaca-se como uma das mais eficazes para mitigar riscos durante a implementação de novas versões. Inspirada no uso de canários em minas de carvão para detectar gases perigosos antes que os trabalhadores fossem afetados, essa estratégia envolve liberar a nova versão para um subconjunto pequeno e controlado de usuários antes de expandi-la para a base total. Essa liberação gradual permite validar a estabilidade da aplicação em um ambiente real enquanto minimiza o impacto de eventuais falhas (Martin, 2011).

A principal vantagem do deploy canário está na capacidade de realizar testes no mundo real, com dados e interações autênticas de usuários. Por exemplo, em aplicações de grande escala, como plataformas de e-commerce, um subconjunto pequeno e representativo de usuários pode experimentar a nova versão sem que toda a base seja impactada. Caso sejam detectados problemas, o rollback pode ser realizado rapidamente, limitando o alcance das falhas (Fowler, 2010).

Para implementar um deploy canário, é essencial que a infraestrutura suporte rote- amento granular de tráfego. Ferramentas como Kubernetes, Aws Elastic Load Balancer e Istio oferecem mecanismos para direcionar porcentagens específicas de tráfego para

diferentes versões da aplicação. Além disso, práticas de monitoramento contínuo são indispensáveis, pois ajudam a identificar anomalias em métricas como latência, consumo de recursos ou taxas de erro em tempo real, fornecendo dados para tomada de decisão durante o rollout (Young, 2020).

Comparado a outras estratégias, como o blue-green deploy ou o rolling upgrade, o canary deploy proporciona um equilíbrio ideal entre controle e eficiência. Enquanto o blue-green requer ambientes duplicados, o que pode ser custoso, e o rolling upgrade opera bem em atualizações contínuas, o canary deploy destaca-se em cenários onde é fundamental testar mudanças com um risco extremamente controlado antes de atingir toda a base de usuários (Amazon, s.d.).

Embora seja altamente vantajoso, o deploy canário também apresenta desafios. A configuração do roteamento de tráfego e o gerenciamento de estados entre versões podem ser complexos, especialmente em sistemas que não foram projetados para operações distribuídas. Apesar disso, sua capacidade de identificar problemas em fases iniciais e seu impacto limitado no ambiente produtivo tornam essa estratégia uma escolha popular para organizações que priorizam confiabilidade e experiência do usuário (RedHat, 2022).

Rolling Upgrade

O Rolling Upgrade é uma estratégia de deploy que visa minimizar o tempo de inatividade e os riscos durante a atualização de sistemas em produção. Nessa abordagem, a aplica- ção é dividida em várias instâncias, e a atualização ocorre de forma gradual, com cada instância sendo atualizada individualmente enquanto as demais continuam operacionais. Essa técnica é especialmente útil em sistemas que exigem alta disponibilidade, como pla- taformas de e-commerce, serviços online e sistemas críticos de negócios (Amazon, s.d.; RedHat, 2022).

O processo de Rolling Upgrade começa com a preparação da nova versão em um ambiente de staging. Em seguida, cada instância do ambiente de produção é retirada do pool de servidores, atualizada e reintegrada após validação. Esse processo é repetido até que todas as instâncias estejam na nova versão. A principal vantagem dessa estratégia é a minimização do tempo de inatividade, já que o serviço permanece ativo durante a maior parte do processo. Além disso, problemas podem ser detectados e corrigidos antes que todas as instâncias sejam atualizadas, reduzindo riscos e facilitando o rollback em caso de falhas (Aws, 2022; Martin, 2011).

No entanto, o Rolling Upgrade apresenta desafios, como a complexidade de geren- ciar a atualização gradual em sistemas com muitas instâncias ou dependências críticas. A coexistência temporária de diferentes versões do software pode causar problemas de compatibilidade, exigindo cuidados adicionais. Além disso, o processo pode ser mais lento comparado a outras estratégias, como o Blue-Green Deploy, especialmente em sistemas de grande escala (Fowler, 2010; RedHat, 2022).

3 METODOLOGIA

3.1 O Estudo de Caso

O estudo de caso desenvolvido neste trabalho utiliza um ambiente de Kubernetes com Istio como plataforma para orquestração de contêineres e gerenciamento de tráfego. Foi escolhida uma aplicação real, composta por um conjunto de replicas da aplicação, visando simular um sistema de produção com múltiplas versões de software por meio da adoção da estratégia de Canary Deployment. A metodologia foi dividida em etapas claras e replicáveis, conforme descrito a seguir.

3.1.1 Preparação do Ambiente

O primeiro passo consistiu na preparação do ambiente que será usado para desenvol- vimento e testes. Para isso, um cluster Kubernetes foi configurado, com a escolha de recursos e configurações baseada nas necessidades da aplicação a ser implantada. O clus- ter foi provisionado utilizando ferramentas do provedor de nuvem pública Oracle Cloud. Em seguida, o Istio foi instalado e configurado no cluster Kubernetes para controlar o roteamento de tráfego entre as versões da aplicação.

O Kubernetes é uma plataforma de orquestração de contêineres amplamente utilizada para automação da implantação, escalonamento e operação de aplicativos em contêineres. Ele fornece os mecanismos essenciais para a execução de estratégias de Canary Deploy- ment, permitindo a gestão eficiente das versões das aplicações e a distribuição do tráfego entre diferentes releases.

O Istio foi instalado e configurado no cluster Kubernetes para gerenciar o roteamento de tráfego entre as diferentes versões da aplicação, desempenhando um papel fundamental na implementação de estratégias como o Canary Deployment. Como uma solução de ser- vice mesh, o Istio oferece controle granular sobre o tráfego de rede, permitindo a divisão de tráfego entre versões da aplicação de forma segura e eficiente. Ele também fornece funcionalidades avançadas, como balanceamento de carga, autenticação mútua entre serviços, monitoramento e coleta de métricas, que são essenciais para garantir a estabilidade e a segurança durante a transição entre versões. Ao integrar-se ao Kubernetes, o Istio facilita a implementação de estratégias de deploy modernas, como o Canary Deployment, permitindo que uma pequena porcentagem do tráfego seja direcionada para a nova versão da aplicação enquanto a maioria dos usuários continua a utilizar a versão estável. Essa abordagem reduz riscos e permite a validação da nova versão em um ambiente real antes de sua liberação completa (ISTIO, 2023; REDHAT, 2022).

Para simular o tráfego de usuários e testar o roteamento entre as versões, foi utiliza-

do o curl, ferramenta de linha de comando que permite enviar requisições HTTP para os serviços e verificar a resposta de diferentes versões da aplicação. Esse teste ajudou a garantir que a divisão do tráfego estava ocorrendo conforme o esperado, com a porcentagem definida sendo corretamente distribuída entre as versões estável e nova da aplicação.

3.1.2 Implementação do Canary Deployment

Após a configuração do ambiente, a arquitetura do Canary Deployment foi implementada utilizando o Istio. Inicialmente, o tráfego foi redirecionado exclusivamente para a versão estável da aplicação. A partir desse ponto, uma fração do tráfego foi direcionada para a versão canário, utilizando regras de roteamento baseadas em pesos definidas no Istio. A porcentagem de tráfego direcionada para a versão canário foi gradualmente aumentada à medida que a nova versão era validada, garantindo uma transição segura e controlada.

3.1.3 Coleta e Análise de Métricas

Durante o processo de implantação, foram coletadas e analisadas métricas de desempenho para avaliar o comportamento da aplicação. As métricas incluíram:

- Tempo de resposta: Medido para comparar o desempenho entre a versão estável e a versão canário.
- Taxa de erros: Verificação de erros ou falhas ocorridas durante o tráfego direcionado para a versão canário.
- Taxa de sucesso: Percentual de requisições bem-sucedidas, comparando ambas as versões para garantir estabilidade e confiabilidade.

Para a coleta dessas métricas, foram utilizadas ferramentas de observabilidade integradas ao Istio, como Prometheus e Kiali, além de logs gerados pelo Kubernetes.

3.1.4 Avaliação dos Resultados

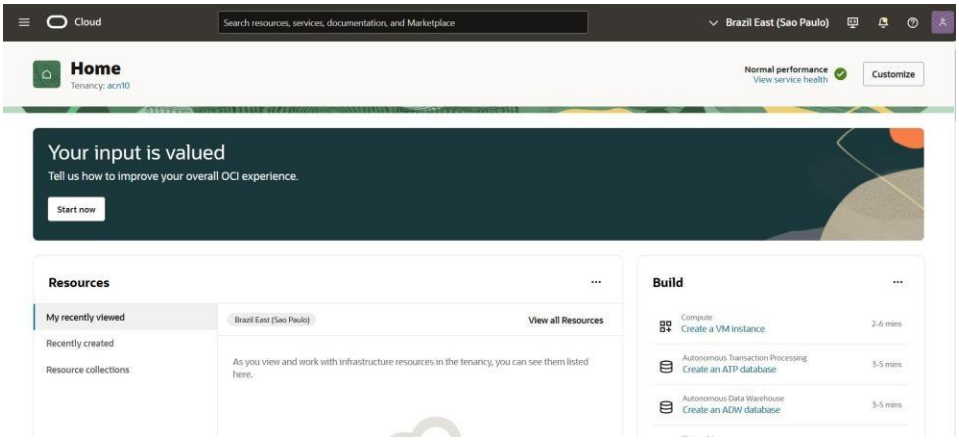
Por fim, os resultados obtidos foram avaliados com base nas métricas coletadas e no comportamento geral da aplicação durante o Canary Deployment. A análise incluiu a identificação de possíveis problemas, a eficácia da estratégia de roteamento de tráfego e a validação da nova versão da aplicação. Com base nessa avaliação, foram propostas melhorias e boas práticas para a implementação de Canary Deployments em ambientes de produção.

4 DESENVOLVIMENTO

4.1 Configuração do Cluster no OKE

O Oracle Kubernetes Engine (OKE) é um serviço gerenciado que permite a criação e operação de clusters Kubernetes na Oracle Cloud Infrastructure (OCI), proporcionando escalabilidade, segurança e integração nativa com serviços da Oracle.

Figura 4.1: Interface Oracle Cloud



(a) FONTE: Própio autor

Pré-requisitos

Antes da criação do cluster, foram atendidos os seguintes pré-requisitos:

- Conta na Oracle Cloud Infrastructure (OCI) com permissões adequadas.
- Instalação e configuração do OCI CLI (Oracle, 2024).
- Instalação do kubectl, ferramenta para gerenciamento do Kubernetes.

Processo de Configuração do Cluster

A criação do cluster Kubernetes foi realizada por meio do console da Oracle Cloud In- frastructure (OCI), conforme os seguintes passos:

Autenticação na Oracle Cloud via CLI Para configurar o ambiente local e acessar a infraestrutura da Oracle, foi utilizado o seguinte comando: *oci setup config* Esse comando gera o arquivo */.oci/config*, contendo as credenciais necessárias.

Processo de Configuração do Cluster

- Acesse Developer Services > Kubernetes Clusters.
- Clique em Create Cluster.
- Escolha a opção Quick Create ou Custom Create.
- Defina o nome do cluster e a região desejada.
- Selecione o tamanho do cluster (exemplo: três nós com 2 CPUs e 4 GB de RAM cada).
- Escolha uma Virtual Cloud Network (VCN) existente ou crie uma nova.
- Finalize a configuração e aguarde a criação do cluster.

Configuração do kubectl para acessar o cluster

Após a criação do cluster, a configuração foi realizada com o seguinte comando:

Figura 4.2: Instalação istio

```
jwbs@cloudshell:~ (sa-saopaulo-1)$ oci ce cluster create-kubeconfig --cluster-id ocid1.cluster.oc1.sa-saopaulo-1.aaaaaaaaj47jnepdqp3u2xc2x32u46wi5vdxgtqwgum2ckc6tclht6
l6effa --file $HOME/.kube/config --region sa-saopaulo-1 --token-version 2.0.0  --kube-endpoint PUBLIC_ENDPOINT
```

(a) FONTE: Própio autor

Verificação do acesso ao cluster

Para confirmar que o cluster foi configurado corretamente, foi executado o seguinte co- mando: *kubectl get nodes*

4.2 Instalação e Configuração do Istio

Com o cluster Kubernetes configurado, o próximo passo foi a instalação do Istio, uma service mesh que oferece controle avançado de tráfego, segurança e observabilidade para microsserviços.

Figura 4.3: Verificação dos nodes do cluster.

```
jwbs@cloudshell:~ (sa-saopaulo-1)$ kubectl get nodes
NAME          STATUS   ROLES   AGE   VERSION
10.0.10.222   Ready    node    27d   v1.31.1
10.0.10.228   Ready    node    27d   v1.31.1
jwbs@cloudshell:~ (sa-saopaulo-1)$
```

(a) FONTE: Própio autor.

Processo de Instalação do Istio

A instalação do Istio foi realizada utilizando a ferramenta istioctl, seguindo os passos abaixo:

- Download da versão mais recente do Istio:

Figura 4.4: Download istio.

```
jwbs@cloudshell:~ (sa-saopaulo-1)$ curl -L https://istio.io/downloadIstio | ISTIO_VERSION=1.24.2 TARGET_ARCH=x86_64 sh -
```

(a) FONTE: Própio autor.

- Adição do Istioctl ao PATH:

Figura 4.5: PATH istio.

```
jwbs@cloudshell:~ (sa-saopaulo-1)$ export PATH=$PWD/bin:$PATH
```

(a) FONTE: Própio autor.

- Instalação do Istio no cluster: A instalação foi realizada com o seguinte comando:

Figura 4.6: Instalação istio.

```
jwbs@cloudshell:~ (sa-saopaulo-1)$ istioctl install --set profile=demo
```

(a) FONTE: Própio autor.

- Verificação da instalação do Istio Para garantir que os componentes do Istio foram instalados corretamente, foi utilizado o comando: *kubectl get pods -n istio-system*

4.2.1 Habilitação da Injeção Automática de Sidecar

O Istio utiliza proxies Envoy como sidecars para gerenciar o tráfego entre os serviços. O Envoy é um proxy de serviços de alto desempenho. No contexto do Istio, ele atua como um

Figura 4.7: Verificação istio.

```
jwbs@cloudshell:~ (sa-saopaulo-1)$ kubectl get pods -n istio-system
NAME                                    READY   STATUS    RESTARTS   AGE
istio-egressgateway-55c8fb4dfc-f6242    1/1     Running   1          2d4h
istio-ingressgateway-54bb98fb97-82kcz   1/1     Running   1          2d4h
istiod-74d89d7dc5-gdgrl                 1/1     Running   1          2d4h
```

(a) FONTE: Própio autor.

intermediário entre os serviços dentro do service mesh, oferecendo funcionalidades avançadas como roteamento inteligente, segurança, balanceamento de carga e observabilidade. A ativação da injeção automática desses sidecars garante que todas as aplicações dentro do namespace marcado utilizem os recursos do Istio. Quando essa injeção está habilitada, um proxy Envoy é automaticamente adicionado a cada pod, interceptando todo o tráfego de entrada e saída. Isso permite a aplicação de políticas de segurança, como autenticação mútua (mTLS), além de recursos como circuit breaking, retries e rate limiting. Dessa forma, o uso do Envoy no Istio facilita a comunicação entre os serviços, proporcionando maior controle e visibilidade sem exigir alterações no código das aplicações.

Marcação do namespace para injeção automática de sidecar. Para permitir que os pods no namespace da aplicação recebam automaticamente os sidecars do Istio, foi executado o seguinte comando:

Figura 4.8: Injesção Sidecar.

```
jwbs@cloudshell:~ (sa-saopaulo-1)$ kubectl label namespace default istio-injection=enabled
```

(a) FONTE: Própio autor.

4.2.2 Implantação da Versão Estável

A primeira etapa da implementação consiste em implantar a versão estável da aplicação (v1). Essa versão será a base para o Canary Deployment, servindo a maior parte do tráfego inicialmente.

No caso deste exemplo, a aplicação é uma landing page de restaurante, desenvolvida para apresentar informações como menu, horários de funcionamento, localização e formas de contato. A página também pode incluir um sistema de reservas e avaliações de clientes.

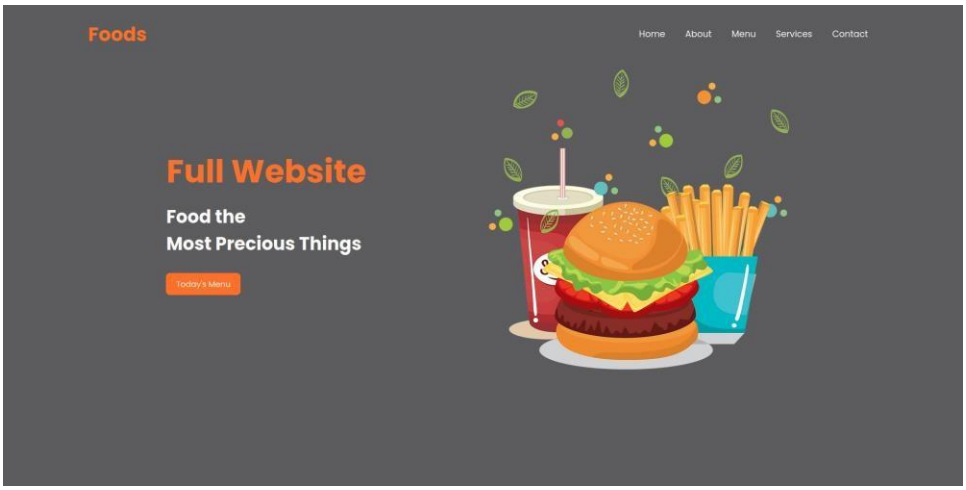


Figura 4.9: Página Inicial da aplicação utilizada.

Essa versão inicial (v1) garante que os usuários tenham uma experiência estável e confiável antes da introdução progressiva de novas funcionalidades ou atualizações através do Canary Deployment. Deployment da Versão v1:

```
io.k8s.api.apps.v1.Deployment (v1@deployment.json)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: appfood-v1
  namespace: appfood
spec:
  replicas: 3
  selector:
    matchLabels:
      app: appfood
      version: v1
  template:
    metadata:
      labels:
        app: appfood
        version: v1
    spec:
      containers:
        - name: appfood
          image: wev6/appfood:v1
          ports:
            - containerPort: 80
          resources:
            requests:
              cpu: "250m"
              memory: "256Mi"
            limits:
              cpu: "500m"
              memory: "512Mi"

```

Figura 4.10: Arquivo YAML Deployment Estável.

O deployment foi aplicado com o comando *kubectl apply -f deployment-v1.yaml*:

```
jwbs@cloudshell:~ (sa-saopaulo-1)$ kubectl apply -f deployment-v1.yaml
```

Figura 4.11: Comando deployment

Para expor a aplicação, foi criado um service que direciona o tráfego para os pods da versão v1. O arquivo YAML utilizado foi:

```
io.k8s.api.core.v1.Service (v1@service.json)
apiVersion: v1
kind: Service
metadata:
  name: appfood
  namespace: appfood
spec:
  selector:
    app: appfood
  ports:
  - port: 80
    targetPort: 80
```

Figura 4.12: Arquivo YAML.

4.3 Configuração do Canary Deployment

O controle de tráfego no Istio é realizado por meio de dois principais recursos: Destination Rules e Virtual Services.

4.3.1 Definição das Destination Rules

As Destination Rules criam identificadores para diferentes versões do serviço, permitindo que o roteamento seja feito de forma segmentada. No Arquivo YAML abaixo, foram criados dois subsets (v1 e v2), permitindo que as Virtual Services redirecionem tráfego para cada versão conforme necessário.

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: appfood-destination
  namespace: appfood
spec:
  host: appfood
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

Figura 4.13: Arquivo YAML Destination Rules.

4.3.2 Configuração do Gateway e VirtualService

Para permitir o acesso externo à aplicação, foram configurados um Gateway e um Virtu- alService no Istio. O arquivo YAML utilizado foi:

Gateway

O Gateway define como o tráfego externo entra na malha de serviços do Istio. Ele é responsável por configurar o ponto de entrada (por exemplo, uma porta HTTP/HTTPS) para o tráfego externo.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: appfood-gateway
  namespace: appfood
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

Figura 4.14: Arquivo YAML Gateway.

Virtual Service

O VirtualService define as regras de roteamento para o tráfego que entra no Gateway. Ele especifica como o tráfego deve ser direcionado para os diferentes subsets (versões) da aplicação.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: appfood
  namespace: appfood
spec:
  hosts:
  - "*"
  gateways:
  - appfood-gateway
  http:
  - route:
    - destination:
        host: appfood
        subset: v1

```

Figura 4.15: Arquivo YAML Virtual Service.

4.3.3 Implantação da Nova Versão

Foi criado um deployment para a versão v2 da aplicação. O arquivo YAML abaixo foi utilizado:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: appfood-v2
  namespace: appfood
spec:
  replicas: 3
  selector:
    matchLabels:
      app: appfood
      version: v2
  template:
    metadata:
      labels:
        app: appfood
        version: v2
    spec:
      containers:
        - name: appfood
          image: wev6/appfood:v2-amd64
          ports:
            - containerPort: 8080
          resources:
            requests:
              cpu: "250m"
              memory: "256Mi"
            limits:
              cpu: "500m"
              memory: "512Mi"

```

Figura 4.16: Arquivo YAML Virtual Service.

4.3.4 Ajuste Progressivo do Tráfego para a Nova Versão

O ajuste do tráfego no Canary Deployment ocorre de forma gradual, permitindo a valida- ção da nova versão (v2) sem impactar toda a base de usuários. Para isso, serão explorados diferentes cenários de distribuição de tráfego entre as versões.

Cenário 1: Implantação Inicial do Canary

Neste primeiro momento, a nova versão v2 recebe apenas 10% do tráfego, enquanto v1 ainda responde por 90% das requisições. O objetivo é testar a estabilidade da nova versão com um volume reduzido. O arquivo YAML abaixo foi utilizado:

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: appfood
  namespace: appfood
spec:
  hosts:
  - "*"
  gateways:
  - appfood-gateway
  http:
  - route:
    - destination:
        host: appfood
        subset: v1
      weight: 90
    - destination:
        host: appfood
        subset: v2
      weight: 10

```

Figura 4.17: Arquivo YAML Canary 90/10.

Cenário 2: Aumento Progressivo do Tráfego

Se a versão v2 apresentar um comportamento estável no primeiro cenário, avançamos para um ajuste progressivo, distribuindo o tráfego de forma equitativa (50% v1 / 50% v2).

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: appfood
  namespace: appfood
spec:
  hosts:
  - "*"
  gateways:
  - appfood-gateway
  http:
  - route:
    - destination:
        host: appfood
        subset: v1
      weight: 50
    - destination:
        host: appfood
        subset: v2
      weight: 50
```

Figura 4.18: Arquivo YAML Canary 50/50.

Cenário 3: Promoção Total da Nova Versão

Se a versão v2 se comportar conforme o esperado, a distribuição de tráfego pode ser ajustada para 100% na nova versão.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: appfood
  namespace: appfood
spec:
  hosts:
  - "*"
  gateways:
  - appfood-gateway
  http:
  - route:
    - destination:
        host: appfood
        subset: v1
      weight: 0
    - destination:
        host: appfood
        subset: v2
      weight: 100
```

Figura 4.19: Arquivo YAML Canary 0/100.

Cenário 4: Detecção de Problemas e Rollback

Caso a nova versão v2 apresente degradação no desempenho, aumento na latência ou falhas acima do limite aceitável, um rollback imediato deve ser realizado. O Istio permite essa reversão rapidamente, direcionando 100% do tráfego de volta para v1, garantindo a estabilidade do serviço.

O monitoramento contínuo com Prometheus e Kiali facilita a detecção precoce de problemas, permitindo a tomada de decisão baseada em métricas como taxa de erro (5xx), tempo de resposta e consumo de recursos.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: appfood
  namespace: appfood
spec:
  hosts:
  - "*"
  gateways:
  - appfood-gateway
  http:
  - route:
    - destination:
        host: appfood
        subset: v1
      weight: 100
    - destination:
        host: appfood
        subset: v2
      weight: 0
```

Figura 4.20: Arquivo YAML Canary 100/0

5 MONITORAMENTO E TESTES DE PERFORMANCE

Após a implementação dos cenários de ajuste progressivo de tráfego, é essencial realizar um monitoramento aprofundado e executar testes de performance para garantir que a nova versão (v2) esteja operando conforme o esperado. Nesta seção, detalharemos o uso das ferramentas Prometheus e Kiali para monitoramento.

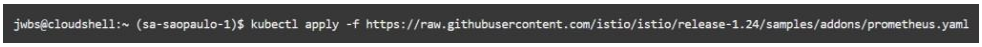
5.1 Prometheus e Kiali

5.1.1 Configuração do Prometheus

O Prometheus é uma ferramenta de monitoramento e sistema de séries temporais open- source que coleta e armazena métricas numéricas em tempo real, ele é capaz de fazer requisições periódicas a endpoints HTTP expostos por aplicações e serviços monitorados. As métricas coletadas podem ser exportadas para alguma ferramenta de observabilidade como o Kiali, possibilitando análises detalhadas ao longo do tempo. O Istio fornece uma instalação básica do Prometheus para uma configuração rápida. Para a instalação o co- mando abaixo foi executado:

- Download da versão mais recente do Prometheus:

Figura 5.1: Download prometheus



(a) FONTE: Própio autor

5.1.2 Configuração do Kiali

O Kiali é uma interface de observabilidade desenvolvida especificamente para ambientes Istio, fornecendo visualizações detalhadas sobre o tráfego de rede, topologia dos serviços e métricas de desempenho em tempo real. Sua principal função é auxiliar na compreensão

do comportamento dos microserviços, permitindo a detecção rápida de falhas, gargalos e anomalias na comunicação entre os pods. Para integrá-lo:

- Download da versão mais recente do Kiali:

Figura 5.2: Download Kiali

```
jwbs@cloudshell:~ (sa-saopaulo-1)$ kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.24/samples/addons/kiali.yaml
```

(a) FONTE: Própio autor

5.1.3 Visualização com Prometheus e Kiali

Com o Prometheus e o Kiali configurados, é possível monitorar detalhadamente o com- portamento das versões v1 e v2 da aplicação.

- Consultas no Prometheus: Utilizamos o Prometheus para criar consultas que monitorem métricas específicas, como taxa de erro, latência e throughput das versões implantadas.

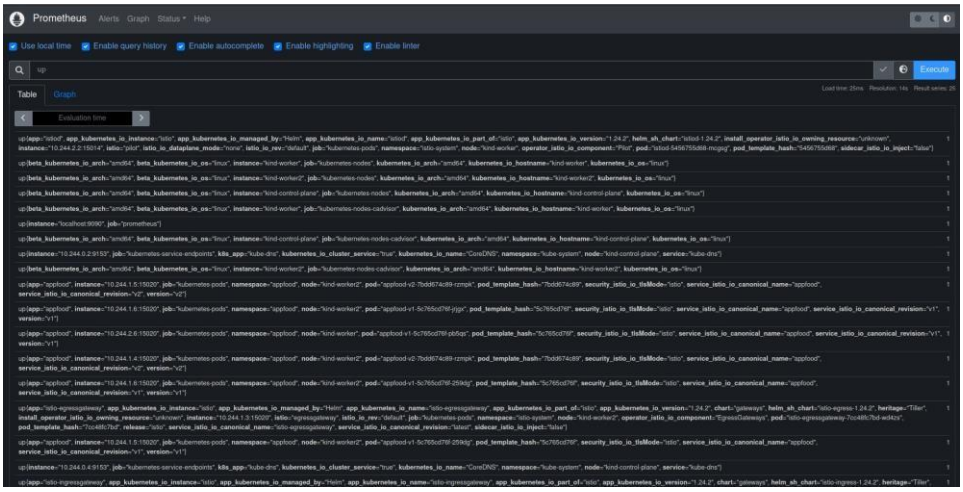


Figura 5.3: Interface Prometheus

- Dashboards do Kiali: O Kiali fornece gráficos que ilustram a topologia do serviço mesh, permitindo visualizar o fluxo de tráfego entre serviços e identificar possíveis gargalos ou falhas.

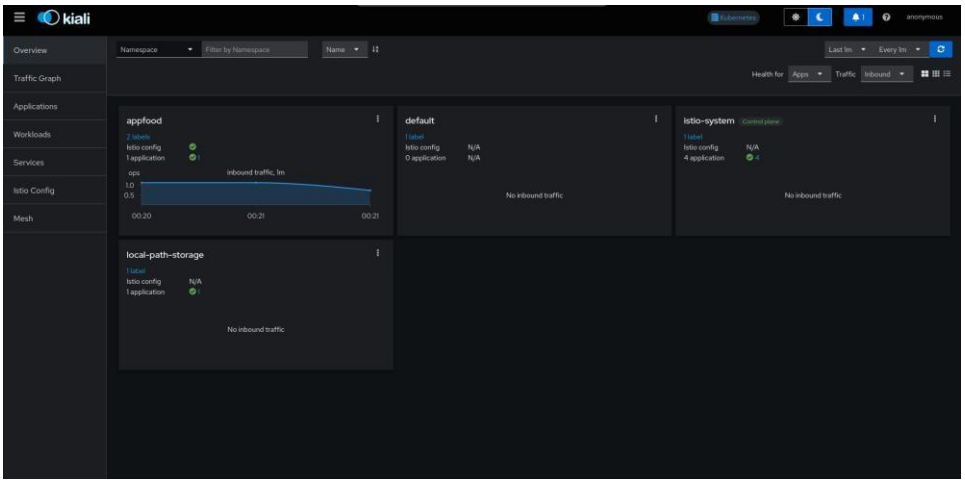


Figura 5.4: Interface Kiali

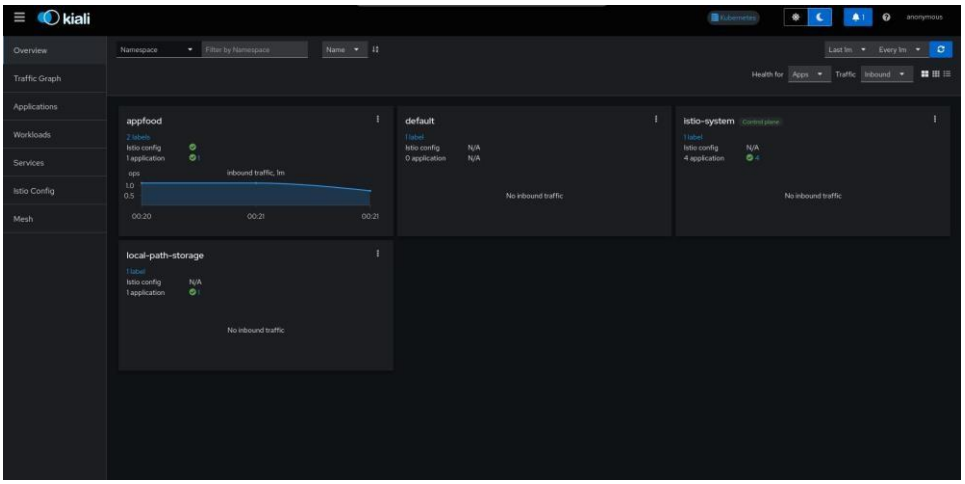


Figura 5.5: Interface Kiali

5.1.4 Visualização da Topologia no Kiali

O Kiali oferece uma visão gráfica e interativa da topologia da aplicação, permitindo que os operadores visualizem como o tráfego flui entre os serviços e suas respectivas versões. Essa capacidade de mapear as conexões e dependências entre os serviços é essencial para identificar gargalos, falhas ou comportamentos anômalos, além de facilitar a depuração e a otimização do sistema.

Figura 5.6: Topologia no Kiali

Na imagem acima, é possível observar de forma detalhada a arquitetura e as conexões entre os principais componentes do ambiente, incluindo o Ingress, os Services, as versões v1 e v2 da aplicação, bem como outros elementos integrantes do ecossistema Istio. O Ingress, que serve como o ponto de entrada para o tráfego externo, está claramente re- presentado, demonstrando como ele recebe as requisições e as encaminha para os Services correspondentes. Esses Services, por sua vez, atuam como abstrações que definem como o tráfego é distribuído entre os Pods que implementam as diferentes versões da aplicação (v1 e v2).

5.1.5 Análise da Distribuição de Tráfego no Kiali

Para avaliar o comportamento da aplicação sob diferentes versões e distribuir o tráfego de maneira controlada, foi utilizada uma estratégia de geração de requisições automatizadas. A simulação foi realizada por meio do comando curl em um loop infinito dentro do cluster, onde uma requisição era enviada para o serviço da aplicação a cada segundo. O script utilizado foi o seguinte:

```
jwbs@cloudshell:~ (sa-saopaulo-1)$ while true; do curl http://appfood.svc.cluster.local; sleep 1; done
```

Figura 5.7: Comando de simulação de requisições

Esse comando executa um loop contínuo (while true), no qual a cada iteração, o curl re- aliza uma requisição HTTP para o serviço da aplicação (http://appfood.svc.cluster.local). Em seguida, o script aguarda um segundo (sleep 1) antes de realizar uma nova requisi- ção. Dessa forma, um tráfego constante foi gerado dentro do cluster, permitindo a análise detalhada da distribuição das requisições entre as versões da aplicação.

No Cenário 1, onde 90% do tráfego ainda está na versão v1 e apenas 10% é redireci- onado para v2, Podemos observar no kiali que o Service principal da aplicação aparece como um nó central, com duas ramificações claramente identificáveis: uma para a v1 e

outra para a v2. A porcentagem nas linhas que conectam o Service às versões reflete o volume de tráfego, onde podemos ver uma leve margem de variação da proporção definida de 90% para 10%, respectivamente. Além disso, as cores das linhas variam de acordo com métricas como latência ou taxa de erro, permitindo uma análise rápida do desempenho de cada versão.

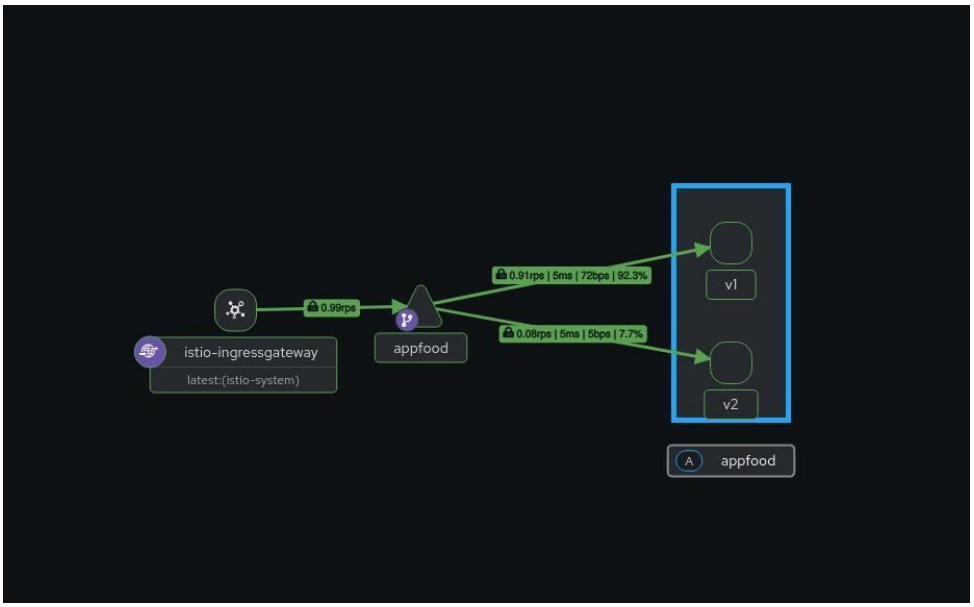


Figura 5.8: Gráfico de Distribuição no Cenário 1

No Cenário 2, onde o tráfego é redistribuído de forma equilibrada, com 50% direcionado para a versão v1 e 50% para a versão v2, podemos visualizar no gráfico de topologia, que Service principal da aplicação ainda aparece como um nó central, mas agora as linhas que conectam o Service às versões v1 e v2 apresentam porcentagens muito próximas, refletindo a divisão igualitária de 50% do tráfego para cada versão. Essa mudança é claramente visível, permitindo que os podemos confirmar que a redistribuição de tráfego foi aplicada conforme o esperado.

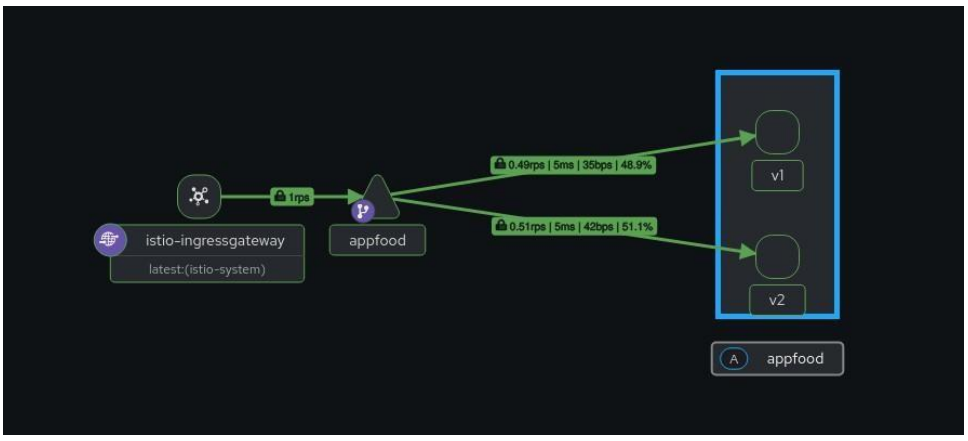


Figura 5.9: Gráfico de Distribuição no Cenário 2

No Cenário 3, onde a versão v2 recebe 100% do tráfego, o Kiali se torna uma ferramenta indispensável para monitorar e garantir que a nova versão esteja operando conforme o esperado. Neste cenário, o gráfico de topologia no Kiali mostra uma mudança significativa em relação aos cenários anteriores. O Service principal da aplicação ainda aparece como um nó central, mas agora há apenas uma linha de conexão, que direciona todo o tráfego para a v2. A ausência de tráfego para a v1 é claramente visível, confirmando que a migração para a nova versão foi concluída com sucesso.

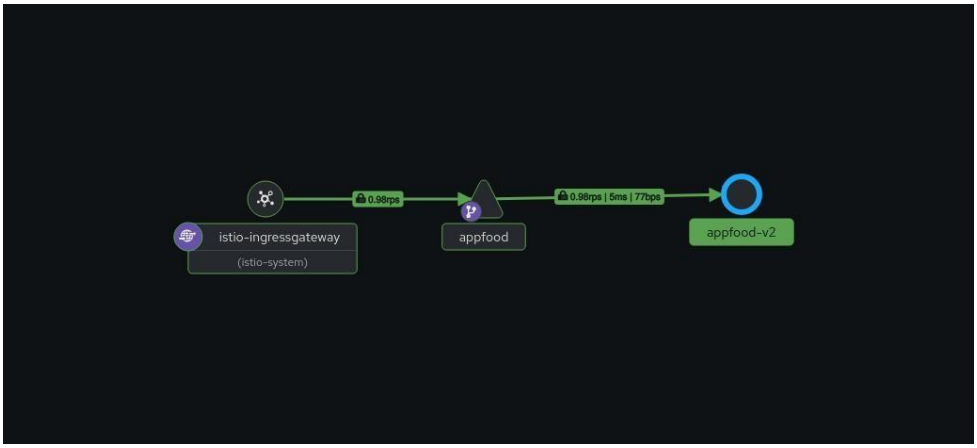


Figura 5.10: Gráfico de Distribuição no Cenário 3

A taxa de erro também é uma métrica crucial a ser monitorada. No Cenário 3, espera- se que a v2, após ter sido testada e validada nos cenários anteriores, apresente uma taxa de erro baixa ou próxima de zero. No entanto, caso ocorra um aumento inesperado na taxa de erro, o Kiali exibe alertas visuais no gráfico, permitindo que a equipe identifique e resolva rapidamente possíveis problemas. Além disso, o Kiali pode mostrar detalhes sobre os tipos de erros que estão ocorrendo, como timeouts, falhas de conexão ou respostas HTTP inadequadas, o que facilita a depuração.

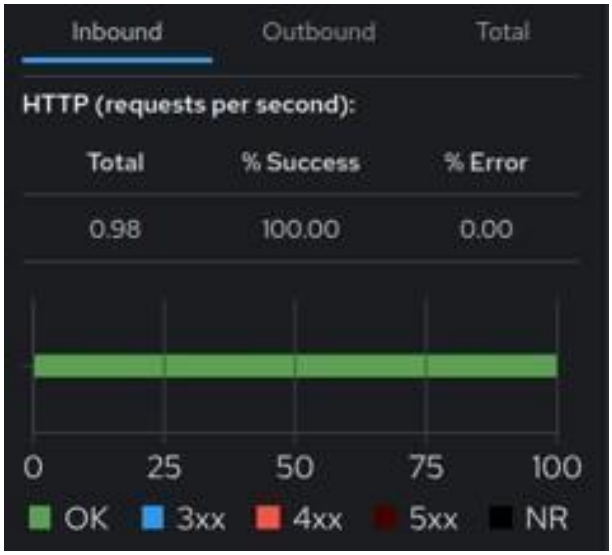


Figura 5.11: Gráfico de Distribuição no Cenário 3

Na imagem acima, após a versão v2 receber 100% do tráfego, o monitoramento con- tínuo realizado no Kiali confirmou que a migração foi bem-sucedida e que a nova versão está operando de forma estável e eficiente. A taxa de erro, que é uma das métricas mais críticas a serem observadas, manteve-se consistentemente baixa, próxima de zero, durante todo o período de monitoramento. Isso indica que a v2, após ter sido testada e validada nos cenários anteriores (Cenário 1 e Cenário 2), está lidando adequadamente com a carga total de tráfego, sem apresentar falhas.

No Cenário 4, simulamos uma situação crítica em que a versão v2, após receber 50% do tráfego no Cenário 2, apresenta um aumento significativo na taxa de erro ou degradação no desempenho. Esse cenário é fundamental para validar a eficácia do mecanismo de rollback e garantir que a aplicação possa retornar rapidamente à versão estável (v1) em caso de problemas.

Para simular um erro na versão v2, foi utilizada uma versão da imagem Docker ine- xistente no registro de contêineres. Essa abordagem foi escolhida para forçar falhas na inicialização dos pods da v2, resultando na indisponibilidade temporária do serviço. O Kiali foi utilizado para monitorar em tempo real o comportamento da aplicação, a fer- ramenta exibiu alertas visuais no gráfico de topologia, indicando que a v2 não estava respondendo adequadamente às requisições.

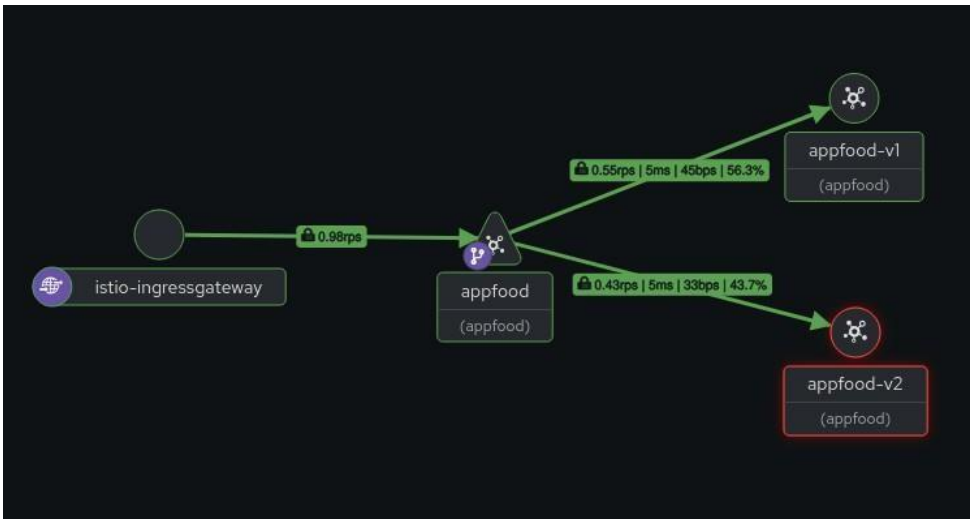


Figura 5.12: Gráfico de Distribuição no Cenário 4 - Erro na v2

Na imagem acima, é possível observar que a v2 começou a apresentar uma taxa de erro elevada (indicada pela cor vermelha na aplicação v2). Essa detecção precoce é crucial para evitar impactos negativos na experiência do usuário e na estabilidade do sistema.

Diante do aumento na taxa de erro e da degradação no desempenho da v2, foi execu- tado o procedimento de rollback, redirecionando 100% do tráfego de volta para a versão estável (v1). Esse ajuste foi realizado por meio da atualização do VirtualService no Istio, que reconfigurou o roteamento de tráfego para direcionar todas as requisições à v1.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: appfood
  namespace: appfood
spec:
  hosts:
  - "*"
  gateways:
  - appfood-gateway
  http:
  - route:
    - destination:
        host: appfood
        subset: v1
      weight: 100
    - destination:
        host: appfood
        subset: v2
      weight: 0
```

Figura 5.13: Arquivo YAML Rollback 100/0

Após a aplicação do rollback, o Kiali confirmou que todo o tráfego foi redirecionado para a v1,garantindo a estabilidade do sistema, conforme ilustrado na imagem abaixo.

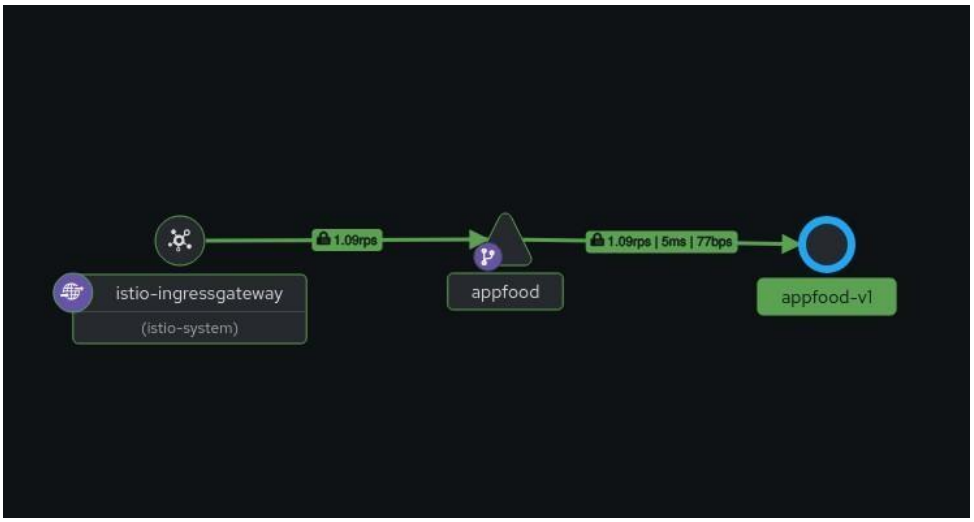


Figura 5.14: Gráfico de Distribuição após Rollback - 100% v1

O Cenário 4 demonstrou a importância de um mecanismo de rollback rápido e eficiente em estratégias de Canary Deployment. A capacidade de detectar problemas em tempo real e redirecionar o tráfego para a versão estável é essencial para minimizar impactos negativos e garantir a continuidade do serviço. O uso do Istio e do Kiali mostrou-se fundamental nesse processo, fornecendo as ferramentas necessárias para monitorar, identificar e corrigir falhas de forma ágil.

Além disso, esse cenário reforçou a necessidade de testes rigorosos e monitoramento contínuo durante a implantação de novas versões. A simulação de falhas e a execução de rollbacks devem fazer parte do planejamento de qualquer estratégia de implantação gradual, garantindo que a equipe esteja preparada para responder rapidamente a qualquer imprevisto.

Por fim, o sucesso do rollback no Cenário 4 valida a robustez da arquitetura proposta e a eficácia das ferramentas utilizadas, consolidando o Canary Deployment como uma prática segura e confiável para a gestão de microsserviços em produção.

6.1 Conclusões

A implementação do Canary Deployment utilizando o Istio demonstrou-se uma aborda- gem eficaz para a implantação gradual de novas versões de serviços em um ambiente de microsserviços. Através da configuração de Destination Rules, Virtual Services e Ga- teways, foi possível controlar de forma precisa o tráfego entre as versões v1 e v2 da apli- cação, garantindo uma transição suave e segura. A estratégia de distribuição progressiva do tráfego, iniciando com 10% para a nova versão e aumentando gradualmente até 100%, permitiu a validação da estabilidade e desempenho da v2 sem impactar significativamente a experiência do usuário.

O uso de ferramentas de monitoramento como Prometheus e Kiali foi fundamental para o sucesso da implementação. O Prometheus forneceu métricas detalhadas sobre o comportamento das versões, enquanto o Kiali ofereceu uma visão clara da topologia da aplicação e do fluxo de tráfego entre os serviços. Essas ferramentas permitiram a detecção precoce de possíveis problemas e a tomada de decisões baseadas em dados, garantindo a confiabilidade do sistema.

Além disso, a capacidade de realizar um rollback rápido em caso de problemas com a nova versão destacou a importância de ter um plano de contingência bem definido. A flexibilidade do Istio em ajustar dinamicamente o roteamento de tráfego mostrou-se uma vantagem significativa, permitindo que a equipe responda rapidamente a qualquer degradação no desempenho ou aumento na taxa de erros.

6.2 Trabalhos Futuros

Embora os resultados obtidos tenham sido positivos, há várias áreas que podem ser ex- ploradas para melhorar ainda mais a implementação do Canary Deployment e o uso do Istio em clusters Kubernetes. Algumas dessas áreas incluem:

- Automatização do Processo de Canary Deployment: A implementação atual requer ajustes manuais nos arquivos YAML para alterar a distribuição de tráfego entre as versões. Um trabalho futuro poderia focar na automatização desse processo, utilizando scripts ou ferramentas de CI/CD para ajustar dinamicamente o tráfego com base em métricas de desempenho e estabilidade.

- Integração com Ferramentas de Análise de Logs: Além do Prometheus e Kiali, a integração com ferramentas de análise de logs, como o ELK Stack (Elasticsearch, Logstash, Kibana), poderia fornecer insights adicionais sobre o comportamento da aplicação, permitindo uma análise mais aprofundada de erros e padrões de uso.

- Testes de Carga e Performance: Realizar testes de carga mais abrangentes para simular cenários de tráfego intenso e identificar possíveis gargalos ou pontos de falha na nova versão. Isso ajudaria a garantir que a aplicação seja capaz de lidar com picos de demanda sem degradação no desempenho.

Em resumo, a implementação do Canary Deployment com Istio mostrou-se uma es- tratégia robusta e eficiente para a implantação de novas versões de serviços em ambientes de microsserviços. No entanto, há um vasto campo de oportunidades para aprimorar e expandir essa abordagem, visando aumentar a automação, a resiliência e a eficiência do sistema como um todo.

REFERÊNCIAS

AMAZON. **Blue/Green Deployments**. s.d. Disponível em: https://aws.amazon.com /blogs/devops/blue-green-deployments-with-aws/. Acessado em: 15/10/2024.

AMAZON. **What is Continuous Delivery?**. 2021. Disponível em: https://aws.amaz on.com/devops/continuous-delivery/. Acessado em: 03/11/2024.

ATLASSIAN. **Continuous Deployment**. 2020. Disponível em: https://www.atlassia n.com/continuous-delivery/continuous-deployment. Acessado em: 20/12/2024.

AWS. **Deployment Strategies**. 2022. Disponível em: https://aws.amazon.com/devop s/deployment-strategies/. Acessado em: 05/01/2025.

AWS. **What is DevOps?**. 2021. Disponível em: https://aws.amazon.com/devops/wh at-is-devops/. Acessado em: 10/02/2025.

BASS, L. et al. **DevOps:** A Software Architect's Perspective. Addison-Wesley Professional, 2015.

CHATTERJEE, S. **DevOps:** A Comprehensive Guide. 2021.

DOCKER INC. **What is a Container?**. 2020. Disponível em: https://www.docker.c om/resources/what-container. Acessado em: 12/10/2024.

EBERT, C. et al. DevOps. **IEEE Software**, v. 33, n. 3, p. 94-100, 2016.

ELASTIC. **DevSecOps:** Integrating Security into DevOps. 2023. Disponível em: https://www.elastic.co/what-is/devsecops. Acessado em: 18/11/2024.

FARADAY. **Continuous Deployment:** a practical guide. 2019. Disponível em: https://www.faraday.io/blog/continuous-deployment-practical-guide. Acessado em: 30/12/2024.

FOWLER, M. **BlueGreenDeployment**. 2010. Disponível em: https://martinfowler.c om/bliki/BlueGreenDeployment.html. Acessado em: 22/01/2025.

FOWLER, M. **Canary Release**. 2023. Disponível em: https://martinfowler.com/bliki /CanaryRelease.html. Acessado em: 14/02/2025.

GHAHRAMANI, M.; ZHOU, M.; HON, C. **Cloud Computing and Its Applications**. In: Proceedings of the 2017 International Conference on Cloud Computing and Big Data Analysis, 2017.

GOOGLE CLOUD. **Kubernetes Overview**. s.d. Disponível em: https://cloud.google .com/kubernetes-engine/docs/concepts/kubernetes-engine-overview. Acessado em: 08/10/2024.

GOOGLE CLOUD. **What is PaaS?**. s.d. Disponível em: https://cloud.google.com/lea rn/what-is-paas. Acessado em: 17/11/2024.

HOLBROOK, J. **What is a Virtual Machine**. In: VMware Virtualization Fundamen- tals, 2022.

IBM. **Kubernetes: Challenges and Benefits**. s.d. Disponível em: https://www.ibm. com/cloud/learn/kubernetes. Acessado em: 29/12/2024.

IBM CLOUD TEAM. **Containers vs Virtual Machines**. 2021. Disponível em: https: //www.ibm.com/cloud/blog/containers-vs-vms. Acessado em: 11/01/2025.

ISTIO. **Istio Documentation**. 2023. Disponível em: https://istio.io/latest/docs/. Acessado em: 19/02/2025.

KNORR, E.; GRUMAN, G. **What Cloud Computing Really Means**. InfoWorld, 2008. Disponível em: https://www.infoworld.com/article/2683784/what-cloud-com puting-really-means.html. Acessado em: 07/10/2024.

MARTIN, R. C. **Continuous Deployment**. 2011. Disponível em: https://blog.clean coder.com/uncle-bob/2011/04/25/ContinuousDeployment.html. Acessado em: 16/11/2024.

NEWMAN, S. **Building Microservices**. 1. ed. [S.l.]: O'Reilly Media, 2015.

RAVICHANDRAN, T. et al. **DevOps:** principles and Practices. In: Proceedings of the 2016 ACM SIGMIS Conference on Computers and People Research, 2016.

RED HAT. **Deployment Strategies**. 2022. Disponível em: https://www.redhat.com/e n/topics/devops/deployment-strategies. Acessado em: 04/01/2025.

RED HAT. **Kubernetes e Canary Deployments**. 2023. Disponível em: https://ww w.redhat.com/en/blog/simple-canary-deployments-using-kubernetes-statefulsets-o n-openshift. Acessado em: 13/02/2025.

RED HAT. **Types of Cloud Computing**. 2023. Disponível em: https://www.redhat .com/en/topics/cloud-computing/public-cloud-vs-private-cloud-and-hybrid-cloud. Acessado em: 21/09/2024.

RED HAT. **What is a Container?**. 2022. Disponível em: https://www.redhat.com/en/topics/containers/what-is-a-container. Acessado em: 06/11/2024.

RED HAT. **What is Continuous Deployment?**. 2020. Disponível em: https://www.redhat.com/en/topics/devops/what-is-continuous-deployment. Acessado em: 24/12/2024.

RED HAT. **What is Continuous Integration?**. 2020. Disponível em: https://www.redhat.com/en/topics/devops/what-is-continuous-integration. Acessado em: 09/01/2025.

RED HAT. **What is DevSecOps?**. 2023. Disponível em: https://www.redhat.com/en/topics/devops/what-is-devsecops. Acessado em: 18/02/2025.

RED HAT. **What is Istio?**. 2022. Disponível em: https://www.redhat.com/en/topics/microservices/what-is-istio. Acessado em: 02/10/2024.

RICHARDSON, C. **Microservices Patterns**. 1. ed. [S.l.]: Manning Publications, 2018.

RITTINGHOUSE, J. W.; RANSOME, J. F. **Cloud Computing:**implementation, Management, and Security. CRC Press, 2017.

SENAPATHI, M.; BUCHAN, J. **DevOps Capabilities, Practices, and Challenges**. In: Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, 2018.

SHAHIN, M.; ALI BABAR, M.; ZHU, L. **Continuous Integration, Delivery and Deployment:** a Systematic Review. In: Proceedings of the 2017 International Conference on Software and System Process, 2017.

TOTVS. **What is SaaS?**. s.d. Disponível em: https://www.totvs.com/blog/saas/o-que-e-saas/. Acessado em: 26/09/2024.

WIEDEMANN, A. et al. **DevOps: A Systematic Literature Review**. In: Proce- edings of the 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2019.

YOUNG, J. **Canary Deployments: A Practical Guide**. 2020. Disponível em: https://www.youngtech.com/blog/canary-deployments-practical-guide. Acessado em: 22/02/2025.