Centro Estadual De Educação Tecnológica Centro Paula Souza
Faculdade De Tecnologia de Indaiatuba
Curso Superior Em Redes De Computadores

Guilherme Gonçalves Milesi

Implementação em Cloud Computing com Kubernetes de software em Microserviço

INDAIATUBA
2025

Centro Estadual De Educação Tecnológica Centro Paula Souza
Faculdade De Tecnologia de Indaiatuba
Curso Superior Em Redes De Computadores

Guilherme Gonçalves Milesi

Implementação em Cloud Computing com Kubernetes de software em Microserviço

Projeto de Trabalho de Graduação do Curso de Redes de Computadores Faculdade Fatec Indaiatuba.

INDAIATUBA
2024

Centro Estadual De Educação Tecnológica Centro Paula Souza
Faculdade De Tecnologia de Indaiatuba
Curso Superior Em Redes De Computadores

Guilherme Gonçalves Milesi

Banca Avaliadora:

Prof. Welligton	**Orientador**
Prof. ipsum lorem	
Profª. ipsum lorem	

Dedicatória

Dedico este trabalho à minha filha, Camille, por ser minha inspiração constante, por seu amor incondicional e por me motivar a ser sempre melhor. À minha mãe, Jane, pela sua dedicação incansável, pela paciência e apoio inabaláveis em todos os momentos, e por acreditar em mim mesmo nos dias mais desafiadores.

Aproveito também para expressar minha profunda gratidão aos professores da FATEC, cuja orientação e conhecimento foram fundamentais para a realização deste trabalho.

Agradecimentos

Agradeço a mim mesmo por nunca desistir, pela perseverança em cada desafio e pela determinação em seguir adiante. Agradeço também a Deus, cuja presença e sabedoria me guiam e aconselham em cada passo desta jornada.

“Quando você deseja uma coisa, todo o Universo conspira para que possa realizá-la”

- O alquimista

Resumo

Este trabalho de conclusão de curso investiga o uso de computação em nuvem e da tecnologia Kubernetes para orquestração de contêiners em uma aplicação baseada em microserviços. O foco do projeto é explorar o ecossistema de computação em nuvem para desenvolver e implantar um software utilizando ferramentas e serviços oferecidos pela AWS. Serão abordados os aspectos culturais e operacionais da filosofia DevOps para apresentar as melhores práticas atuais no desenvolvimento de software, demonstrando a aplicação correta e otimizada da computação em nuvem em um cenário de microserviços. A infraestrutura central utiliza um cluster de máquinas virtuais na AWS (instâncias EC2), orquestrado pelo Kubernetes através do Elastic Kubernetes Service (EKS). A filosofia DevOps será aplicada com o objetivo de agregar agilidade ao projeto, utilizando ferramentas de integração contínua e entrega contínua (CI/CD) para acelerar o desenvolvimento e a implantação.

Palavras-chave: *Computação em Nuvem, Kubernetes, Microserviços, DevOps, AWS, CI/CD, Orquestração de Contêineres.*

Abstract

This thesis investigates the use of cloud computing and Kubernetes technology for container orchestration in a microservices-based application. The project focuses on exploring the cloud computing ecosystem to develop and deploy software using AWS tools and services. It examines the cultural and operational aspects of the DevOps philosophy to present current best practices in software development, demonstrating the correct and optimized application of cloud computing in a microservices environment. The core infrastructure employs a cluster of virtual machines on AWS (EC2 instances), orchestrated by Kubernetes through the Elastic Kubernetes Service (EKS). The DevOps approach is applied to enhance project agility, leveraging continuous integration and continuous delivery (CI/CD) tools to accelerate development and deployment.

Lista de Figuras

Lista de abreviaturas

AWS	Amazon Web Services
CI/C	Continuous Integration / Continuous Deployment
EKS	Elastic Kubernetes Service
IaC	Infrastructure as Code
S3	Simple Storage Service
EC2	Elastic Compute Cloud
API	Application Programming Interface
DNS	Domain Name System
VPC	Virtual Private Cloud
SRE	Site Reliability Engineering
ELB	Elastic Load Balancing
RDS	Relational Database Service
CRD	Custom Resource Definition
API Gateway	Application Programming Interface Gatewa
GitOps	Git Operations

Sumário

Introdução

Nos últimos anos, a computação em nuvem tem se consolidado como uma das tecnologias mais impactantes para o setor de tecnologia da informação, trazendo avanços consideráveis em flexibilidade, escalabilidade e eficiência de recursos. Para Buyya, Broberg e Goscinski (2013), a computação em nuvem representa uma revolução ao permitir que empresas acessem poder computacional de forma flexível, utilizando-o conforme suas demandas e sem a necessidade de infraestrutura física própria. Esse modelo tem se mostrado essencial para o desenvolvimento de soluções inovadoras em um mercado que exige cada vez mais agilidade e adaptação a novas demandas.

A computação em nuvem também possibilita uma significativa melhoria na agilidade operacional das empresas, reduzindo o tempo necessário para lançar novas aplicações e atualizações ao mercado. De acordo com Voorsluys, Broberg e Buyya (2011), essa agilidade é vital para empresas que buscam manter-se competitivas e atender rapidamente às necessidades de seus clientes. A nuvem permite que as empresas possam escalar seus serviços de acordo com o uso real e ajustar seus recursos quase em tempo real, aumentando a eficiência e diminuindo os tempos de resposta.

Contudo, para muitas empresas que desenvolvem software, a implementação de infraestrutura tecnológica própria representa um alto custo de capital (CAPEX). Esses investimentos iniciais são frequentemente necessários para a aquisição de servidores, hardware, software e espaço físico, representando uma barreira financeira significativa, especialmente para empresas menores ou em crescimento. Maréchal (2020) explica que a transição para um modelo baseado em nuvem permite reduzir esses custos iniciais, transferindo-os para um modelo de despesa operacional (OPEX), que, por sua vez, melhora a flexibilidade financeira da empresa e aumenta seu potencial para inovação.

Diante desse cenário, o objetivo deste estudo é investigar como a computação em nuvem, com a tecnologia de orquestração de containers Kubernetes na AWS, pode contribuir para reduzir o CAPEX de empresas de software. Através da implementação de um ambiente em nuvem

gerenciado, como o Elastic Kubernetes Service (EKS) da AWS, o estudo explora como a automação e escalabilidade podem ser otimizadas, proporcionando economia de custos, agilidade e eficiência no desenvolvimento de aplicações baseadas em microserviços.

Este estudo se mostra relevante ao demonstrar como o uso de tecnologias em nuvem pode aumentar a competitividade das empresas no mercado atual. A capacidade de escalar rapidamente e de forma econômica permite que as empresas respondam mais prontamente a novas demandas e inovem em ritmo acelerado, características essenciais para garantir sua relevância e sucesso no mercado. Para Buyya, Broberg e Goscinski (2013), o uso eficiente da nuvem aumenta as possibilidades de expansão e integração de novos recursos, promovendo um ambiente dinâmico e adaptável às necessidades da organização.

A estrutura deste trabalho é composta por quatro capítulos principais. O Capítulo 1, Fundamentação Teórica, fornece uma base sobre os temas de computação em nuvem, orquestração de contêineres com Kubernetes, e práticas DevOps, fundamentada em estudos já estabelecidos na área. O Capítulo 2, Materiais e Metodologias, descreve o processo de pesquisa, com detalhamento do ambiente de estudo e das ferramentas utilizadas. O Capítulo 3, Desenvolvimento e Resultados, apresenta a implementação de um software em arquitetura de microserviços em um cluster Kubernetes na AWS, com foco na automação e escalabilidade. Por fim, o Capítulo 4, Conclusões e Recomendações, discute os resultados obtidos e sugere direções para futuras pesquisas.

CAPÍTULO I

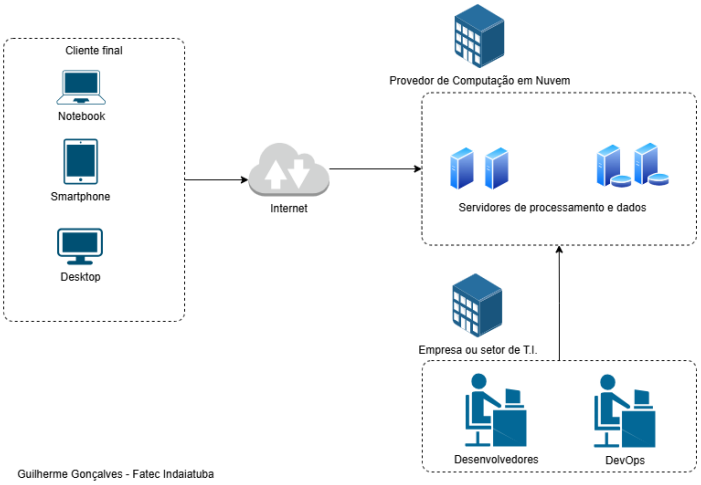
Fundamentação Teórica

Neste capítulo, abordamos os fundamentos das principais tecnologias utilizadas no desenvolvimento de sistemas de microserviços em nuvem, utilizando como referência as práticas DevOps e a infraestrutura da AWS, além do papel do Kubernetes como orquestrador. Esses conceitos são essenciais para entender as práticas e os processos recomendados no desenvolvimento de uma arquitetura resiliente e escalável.

1.1 Computação em Nuvem e Amazon Web Services (AWS).

A computação em nuvem representa um modelo de prestação de serviços de TI no qual os recursos de infraestrutura, como processamento, armazenamento e rede, são fornecidos sob demanda, escaláveis e acessíveis remotamente pela internet. Esse modelo oferece vantagens significativas em termos de flexibilidade, escalabilidade e redução de custos, pois permite que as empresas aluguem recursos de acordo com a necessidade, eliminando investimentos iniciais elevados em hardware e infraestrutura física. Segundo Buyya et al. (2013), um dos pioneiros no campo da computação em nuvem, o modelo de cloud computing não apenas democratiza o acesso a recursos de TI avançados, mas também acelera a inovação ao permitir que organizações de todos os tamanhos experimentem com novos serviços e aplicações sem grandes barreiras de entrada.

Figura 1: Representação do serviço de Computação em Nuvem



A Amazon Web Services (AWS) se destaca entre os provedores de computação em nuvem por seu portfólio abrangente e pela maturidade de seus serviços, que atendem desde pequenas aplicações até soluções empresariais robustas e escaláveis. Dentre os serviços essenciais oferecidos pela AWS para suporte a aplicações ou infraestrutura, os serviços centrais são:

Elastic Compute Cloud (EC2): Serviço de infraestrutura como serviço (IaaS) que fornece capacidade de computação escalável e configurável, permitindo o provisionamento de máquinas virtuais conforme a necessidade. EC2 é a base de muitas aplicações em nuvem, oferecendo flexibilidade e controle para que desenvolvedores ajustem o ambiente de execução.

Simple Storage Service (S3): Um dos serviços mais conhecidos da AWS, o S3 fornece armazenamento de objetos com alta durabilidade e escalabilidade, ideal para armazenar dados de forma segura e econômica. S3 é amplamente utilizado para armazenar logs, backups e arquivos estáticos de aplicações em microserviços.

Relational Database Service (RDS): Serviço gerenciado de banco de dados relacional que suporta várias engines de banco de dados, incluindo MySQL, PostgreSQL e Oracle. O RDS facilita a configuração, operação e escalabilidade de bancos de dados em nuvem, simplificando o gerenciamento de dados críticos das aplicações.

Elastic Load Balancing (ELB): Essencial para distribuições de microserviços, o ELB distribui automaticamente o tráfego de entrada entre várias instâncias de EC2, aumentando a tolerância a falhas e permitindo a escalabilidade dos serviços.

A AWS também oferece o Elastic Kubernetes Service (EKS), um serviço gerenciado que permite a execução de contêineres em Kubernetes, facilitando a orquestração e o gerenciamento de clusters de contêineres. O EKS abstrai grande parte da complexidade de gerenciar o plano de controle do Kubernetes, permitindo que desenvolvedores se concentrarem no deployment e gerenciamento de suas aplicações em contêineres. No contexto de microserviços, o EKS é fundamental para otimizar o gerenciamento e a escalabilidade das aplicações, pois automatiza operações críticas, como escalabilidade horizontal, balanceamento de carga e monitoramento de pods.

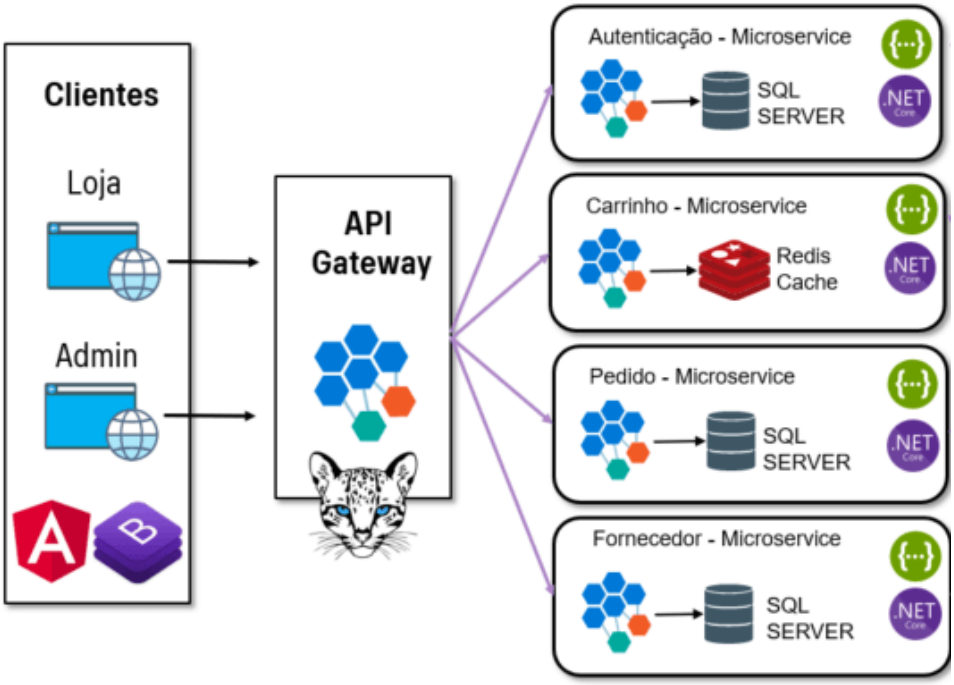
Além disso, para gerenciar as dependências e comunicação entre microserviços, a AWS oferece o Amazon API Gateway, que permite criar, publicar, manter e monitorar APIs em grande escala. Este serviço é especialmente útil para arquiteturas de microserviços, pois permite a criação de interfaces unificadas para que os diferentes serviços possam se comunicar com segurança e eficiência. Outra ferramenta de destaque é o AWS Lambda, que possibilita a execução de código em resposta a eventos, sem a necessidade de provisionar ou gerenciar servidores, uma abordagem altamente eficaz para otimizar recursos e implementar padrões serverless em arquiteturas de microserviços.

Como ressaltado por Voorsluys, Broberg e Buyya (2011), a computação em nuvem não só amplia a capacidade de inovação das organizações, mas também transforma a maneira como aplicações são desenvolvidas, distribuídas e gerenciadas, ao permitir uma abordagem modular e escalável. Em suma, ao combinar os recursos da AWS com práticas de orquestração de contêineres com Kubernetes e a filosofia DevOps, as empresas podem potencializar o desenvolvimento de soluções baseadas em microserviços, com alto grau de automação e resiliência, preparando-se para responder rapidamente às mudanças de mercado.

1.2 Arquitetura de Microserviços e Kubernetes

A arquitetura de microserviços organiza uma aplicação em uma coleção de serviços pequenos e independentes, cada um dedicado a uma função específica, como autenticação, gerenciamento de dados ou processamento de pagamentos. Estes microserviços se comunicam entre si, geralmente por meio de APIs HTTP ou gRPC, o que permite que eles sejam implementados, escalados e atualizados de forma independente. Este modelo de desenvolvimento, descrito por Newman (2015) como um dos mais resilientes para aplicações modernas, é altamente escalável e adaptável, pois permite que cada microserviço funcione de maneira autônoma, isolando possíveis falhas e facilitando o gerenciamento da infraestrutura.

Figura 2: Representação da arquitetura de software em design de microserviço



No entanto, a administração de múltiplos microserviços também introduz desafios complexos, como o gerenciamento do estado, escalabilidade automática, balanceamento de carga, e a monitoração de desempenho de cada serviço. É aí que o Kubernetes, uma plataforma de orquestração de contêineres de código aberto desenvolvida pelo Google, se destaca. O Kubernetes automatiza o deployment, o escalonamento e a manutenção de aplicações containerizadas,

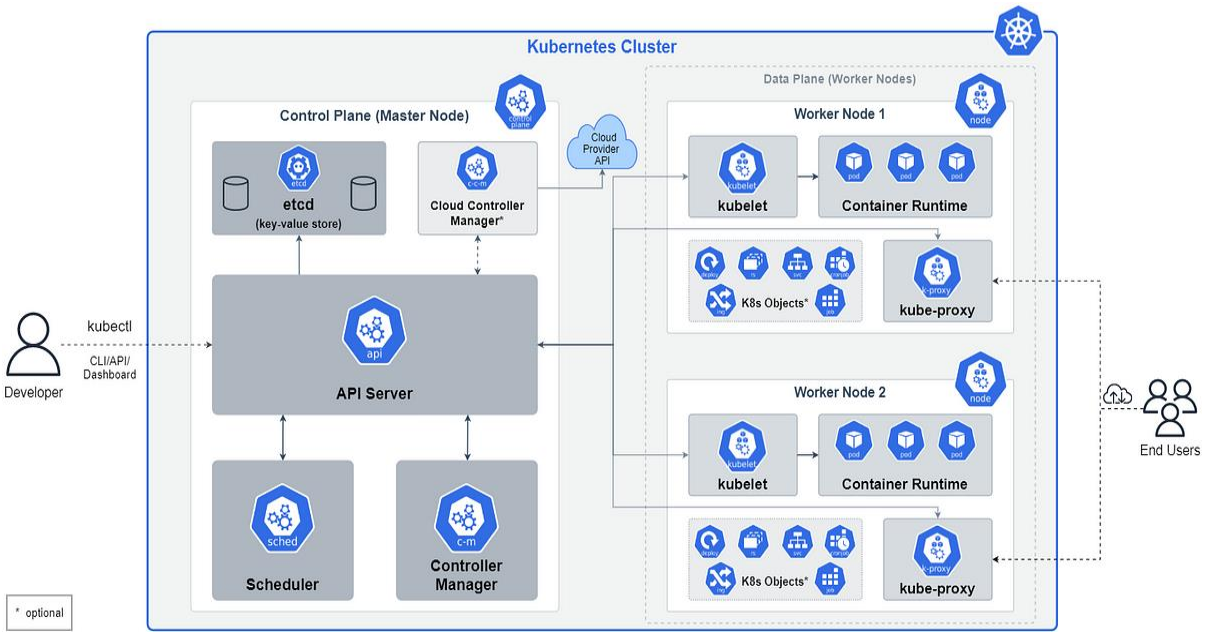
simplificando a gestão de grandes conjuntos de microserviços em produção. Segundo Hightower et al. (2019), o Kubernetes foi projetado para lidar com esses desafios de maneira altamente eficiente, utilizando componentes específicos para cada função.

1.2.1. Componentes da Arquitetura do Kubernetes

O Kubernetes é uma plataforma que opera como um sistema de orquestração de contêineres em um cluster de máquinas virtuais (VMs), facilitando o gerenciamento de aplicações distribuídas. Em um cluster Kubernetes, as VMs servem como nós onde os contêiners são executados e monitorados, formando uma infraestrutura escalável e resiliente. Cada nó no cluster é responsável por hospedar grupos de contêiners, chamados pods, que representam a menor unidade de implantação em Kubernetes.

A arquitetura do Kubernetes é organizada em dois grupos principais de componentes: o plano de controle (control plane) e os nós de trabalho (worker nodes). O plano de controle é responsável por gerenciar todo o cluster, incluindo decisões de agendamento, controle dos estados desejados das aplicações e monitoramento das operações. Já os nós de trabalho executam as cargas de trabalho propriamente ditas, hospedando os contêiners em unidades chamadas pods. Essa divisão permite que o Kubernetes coordene e escale automaticamente as aplicações, proporcionando uma infraestrutura flexível e resiliente para o gerenciamento de contêineres.

Figura 3: Arquitetura interna do Kubernetes



O plano de controle contém os seguintes componentes:

- **API Server:** Atua como o ponto de entrada do Kubernetes, recebendo solicitações para criação, atualização ou exclusão de recursos.
- **etcd:** Banco de dados chave-valor que armazena o estado e as configurações do cluster, garantindo a consistência das informações.
- **Controller Manager:** Controlador que gerencia recursos para manter o estado do cluster conforme o desejado. Ele gerencia, por exemplo, o agendamento de pods em caso de falhas.
- **Scheduler:** Componente responsável por alocar os pods nos nós de trabalho com base nos requisitos de recursos e configurações de afinidade.

Os nodes, responsáveis por processar e alocar a aplicação contém alguns componentes para seu gerenciamento, são três componentes, segue:

- **Kubelet:** Agente que gerencia os pods e se comunica com o plano de controle, garantindo que cada contêiner esteja no estado esperado.
- **Kube-proxy:** Garante que os pods dentro do cluster possam se comunicar, gerenciando as regras de rede e roteamento de tráfego entre os nós.

- **Container Runtime:** Software que executa os contêineres. O Docker é um dos runtimes mais comuns, mas outros como CRI-O e containerd também são suportados.

Cada serviço é executado em contêineres, agrupados em "pods" — a menor unidade de execução do Kubernetes. Os pods são gerenciados em clusters, e o Kubernetes cuida automaticamente do balanceamento de carga e da recuperação de falhas, redirecionando os contêineres para outros nós caso um nó falhe. Esta arquitetura permite escalabilidade e resiliência, dois aspectos essenciais para aplicações modernas baseadas em microserviços.

1.2.2. Kubernetes com Amazon EKS:

Na Amazon Web Services, o Elastic Kubernetes Service (EKS) abstrai a complexidade de configuração e gerenciamento do plano de controle, permitindo que empresas concentrem seus esforços na implementação de aplicações em microserviços, sem a necessidade de manter manualmente os componentes críticos do Kubernetes, como o API Server e o etcd.

O EKS simplifica ainda mais o gerenciamento dos clusters com os seguintes recursos:

- **Gerenciamento Automático do Plano de Controle:** O EKS mantém o plano de controle do Kubernetes, incluindo alta disponibilidade e escalabilidade automática, o que reduz a sobrecarga de operação e oferece um ambiente seguro e atualizado para os clusters.
- **Integração com Serviços de Redes da AWS:** O EKS integra-se com o Elastic Load Balancer (ELB) para distribuir o tráfego de entrada de maneira eficiente, permitindo que os serviços Kubernetes sejam expostos ao tráfego externo por meio de balanceadores de carga configurados automaticamente.

  Opções de Exposição de Aplicações:

  - **LoadBalancer Services:** Kubernetes configura automaticamente um Load Balancer para expor o serviço a tráfego externo, integrando-se ao ELB.
  - **Ingress Controller com AWS Load Balancer Controller:** Configura o Application Load Balancer (ALB) para roteamento avançado e regras de SSL, ideal

para aplicações que precisam de configurações personalizadas de roteamento HTTP.

- **AWS PrivateLink:** Permite expor serviços para acesso privado dentro de uma VPC, aumentando a segurança e mantendo a conformidade com regulamentações de dados.
- **Amazon Route 53**: Serviço de DNS gerenciado da AWS, que pode ser integrado ao EKS para configurar o roteamento do tráfego externo para a aplicação, seja usando domínios personalizados ou configurando políticas de roteamento geográfico e failover.

Ao combinar Kubernetes e AWS EKS, as organizações ganham uma plataforma completa para gerenciar e escalar aplicações em microserviços, com menos esforço de configuração e manutenção. Esse ambiente permite a criação de sistemas altamente resilientes, capazes de responder rapidamente a mudanças e de atender a demandas crescentes.

1.2.3 Kubernetes e addons

Além dos componentes nativos do Kubernetes, um conjunto de ferramentas auxiliares foi desenvolvido para simplificar e automatizar ainda mais as operações em clusters. Algumas dessas ferramentas são conhecidas como "K's" devido ao uso da letra "K" no início de seus nomes. Aqui estão algumas das ferramentas mais relevantes:

- **Karpenter:** Ferramenta de escalabilidade automática que permite ao Kubernetes ajustar dinamicamente os recursos dos nós de acordo com a carga e as necessidades dos pods. Karpenter ajuda a reduzir custos ao provisionar apenas os recursos necessários, integrando-se com provedores de nuvem, como a AWS, para fornecer escalabilidade em tempo real.
- **Kyverno:** Ferramenta de gerenciamento de políticas de segurança para Kubernetes, que permite definir, validar e aplicar regras de conformidade em tempo real no cluster. Com o Kyverno, é possível garantir que os recursos de Kubernetes estejam configurados de acordo com as políticas organizacionais, por exemplo, garantindo o uso de imagens de contêineres seguras ou aplicando práticas de segurança padrão.

- **Kubeflow:** Plataforma de machine learning desenvolvida para Kubernetes, que facilita a execução de pipelines de ML em ambientes de contêineres. Kubeflow simplifica o gerenciamento e a escalabilidade de tarefas de aprendizado de máquina, desde o treinamento até a implantação de modelos em produção.
- **K9s:** Interface de linha de comando interativa que permite monitorar e gerenciar facilmente recursos do Kubernetes. K9s oferece uma visualização prática e intuitiva do cluster, facilitando o acesso e a manipulação de pods, deployments, serviços e outros recursos em tempo real.
- **Kustomize:** Ferramenta de configuração nativa para Kubernetes que permite personalizar arquivos de configuração YAML sem duplicá-los. Com o Kustomize, é possível sobrepor configurações para diferentes ambientes (desenvolvimento, testes, produção) de maneira simples e eficiente, mantendo a estrutura do código DRY (Don't Repeat Yourself).

Essas ferramentas auxiliam na administração, automação e otimização dos clusters Kubernetes, tornando-os mais adaptáveis e seguros, além de facilitarem a implantação e o gerenciamento de aplicações em escala.

1.2.4 Operadores de Kubernetes

Os Operadores de Kubernetes são extensões avançadas que permitem gerenciar recursos e aplicações específicas dentro de um cluster Kubernetes de forma automatizada e personalizada. Inspirados no conceito de "operadores humanos", que realizam tarefas complexas e de manutenção, os operadores de Kubernetes foram projetados para gerenciar o ciclo de vida completo de uma aplicação, incluindo instalação, configuração, escalabilidade e recuperação em caso de falhas.

Funções dos Operadores:

- **Automatização de Tarefas Repetitivas:** Operadores podem executar automaticamente tarefas complexas e repetitivas, como provisionamento de recursos, configuração de backup e recuperação de dados.

- **Gerenciamento do Ciclo de Vida de Aplicações:** Um operador pode garantir que uma aplicação esteja sempre em um estado "desejado", realizando tarefas de monitoramento contínuo e de auto-recuperação em caso de falhas.
- **Customização e Padronização de Recursos:** Operadores permitem definir configurações específicas para aplicações customizadas, facilitando o gerenciamento de aplicações complexas e permitindo que desenvolvedores implementem melhores práticas e políticas de conformidade.

Operadores geralmente são baseados no conceito de **controladores** e utilizam **Custom Resource Definitions (CRDs)** para definir novos tipos de recursos no Kubernetes. Um CRD é um recurso personalizado criado para estender a API do Kubernetes, permitindo que o operador monitore e controle o estado desses novos recursos e garanta sua conformidade com o estado desejado.

Exemplo de Operadores Populares:

- **Prometheus Operator:** Automatiza a criação e o gerenciamento de instâncias do Prometheus, configuração de regras de monitoramento e alertas em clusters Kubernetes.
- **MongoDB Operator:** Facilita o gerenciamento de clusters MongoDB, com automação para escalabilidade, backup, restauração e gerenciamento de usuários.
- **Elasticsearch Operator:** Gerencia e automatiza a implantação de clusters Elasticsearch, permitindo configurações de replicação, ajustes de desempenho e gerenciamento de failover.

Os operadores são amplamente utilizados para simplificar o gerenciamento de aplicações complexas em Kubernetes, oferecendo uma abordagem programática para manutenção e operações contínuas. Essa funcionalidade permite que organizações implementem aplicações resilientes e com maior consistência, eliminando o trabalho manual em operações críticas.

1.3 Terraform: Infraestrutura como código

O Terraform é uma ferramenta desenvolvida pela HashiCorp que permite definir, provisionar e gerenciar infraestrutura de TI por meio de código declarativo. Ao adotar o conceito de Infraestrutura como Código (IaC), o Terraform possibilita que toda a configuração de infraestrutura seja escrita em arquivos de configuração legíveis e versionáveis. Isso agiliza o projeto ao automatizar o processo de provisionamento de recursos, eliminando configurações manuais propensas a erros e garantindo consistência em todos os ambientes (HUMBLE & FARLEY, 2010).

No contexto deste projeto, o uso do Terraform acelera significativamente a implantação da infraestrutura necessária na AWS, sendo possível definir recursos como instâncias EC2, clusters EKS do Kubernetes, balanceadores de carga e outros serviços essenciais. Essa abordagem permite que alterações na infraestrutura sejam aplicadas de forma controlada e repetível, facilitando a escalabilidade e manutenção do ambiente. Além disso, o Terraform oferece suporte a planos de execução que mostram previamente as mudanças que serão aplicadas, aumentando a transparência e previsibilidade das operações (MORRIS, 2016).

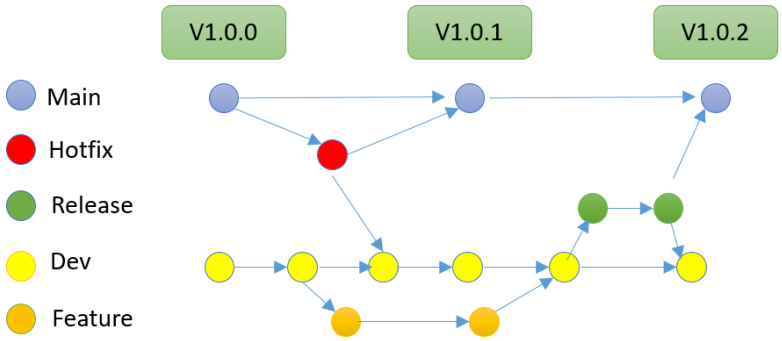
Ao integrar o Terraform com práticas DevOps, a equipe pode incluir o provisionamento de infraestrutura nas pipelines CI/CD, automatizando não apenas o deployment de aplicações, mas também a configuração do ambiente onde elas serão executadas. Isso reduz o tempo de setup para novos ambientes e facilita a recuperação em caso de falhas, aumentando a resiliência do sistema. Em suma, o Terraform como IaC é uma ferramenta vital que agiliza o projeto ao promover automação, consistência e eficiência na gestão de infraestrutura em nuvem.

1.4 Git e versionamento de código

O Git é uma das ferramentas de controle de versão mais utilizadas no desenvolvimento de software moderno, permitindo o gerenciamento eficiente de alterações no código-fonte ao longo do tempo. Segundo Chacon e Straub (2014), o Git facilita a colaboração entre desenvolvedores, proporcionando um histórico detalhado das modificações realizadas, o que é essencial para a manutenção da integridade e da qualidade do código. Além disso, o Git suporta diversas estratégias de branching e merging, permitindo que equipes de desenvolvimento trabalhem de forma paralela e coordenada em diferentes funcionalidades e correções. A adoção do Git como ferramenta de versionamento contribui para a transparência e a responsabilidade dentro das equipes, elementos

fundamentais para a implementação bem-sucedida de arquiteturas de microserviços e práticas de DevOps.

Figura 4: Representação no uso de versionamento de código via branches.



1.4.1 Versionamento e branches

O versionamento com Git envolve a utilização de branches para gerenciar diferentes linhas de desenvolvimento dentro de um projeto. Cada branch representa uma versão isolada do código, permitindo que desenvolvedores trabalhem em novas funcionalidades, correções de bugs ou experimentos sem afetar a versão principal do projeto. Segundo Kim et al. (2016), a estratégia de branching é crucial para manter a organização e a eficiência no desenvolvimento de software, especialmente em ambientes colaborativos e de alta complexidade como os de microserviços. As estratégias mais comuns incluem o uso de branches de feature para desenvolver novas funcionalidades, branches de release para preparar versões estáveis para produção e branches de hotfix para corrigir rapidamente problemas críticos. A adoção de uma estratégia de branching bem definida facilita a integração contínua (CI) e a entrega contínua (CD), permitindo que as equipes implementem e testem alterações de forma rápida e segura. Além disso, o uso de pull requests e revisões de código contribui para a qualidade do software, promovendo a colaboração e a revisão por pares antes que as alterações sejam mescladas à branch principal.

1.4.2 Versionamento em GitOps

GitOps é uma prática que utiliza o Git como a única fonte de verdade para a infraestrutura e as aplicações, integrando o versionamento de código com a automação de processos de deployment. Conforme descrito por Burns et al. (2020), GitOps permite que todas as alterações na infraestrutura sejam feitas através de commits no repositório Git, garantindo rastreabilidade e auditabilidade completas. Essa abordagem facilita a implementação de práticas de Continuous Deployment (CD), onde as atualizações são automaticamente aplicadas ao ambiente de produção após a validação nos estágios de integração contínua. Além disso, GitOps promove a consistência entre ambientes, reduzindo a probabilidade de erros humanos durante o processo de deployment e melhorando a recuperação em caso de falhas. A utilização do GitOps está alinhada com os princípios de infraestrutura como código (IaC), proporcionando uma gestão declarativa e automatizada da infraestrutura em nuvem.

1.5 DevOps: Movimento ágil e papel.

O movimento DevOps surgiu como uma resposta às dificuldades enfrentadas pelas equipes de desenvolvimento de software e operações de TI na entrega rápida e confiável de aplicações. No final dos anos 2000, profissionais como Patrick Debois e Andrew Shafer começaram a discutir a necessidade de maior colaboração entre desenvolvimento e operações. Essa iniciativa ganhou força com a apresentação "Agile Infrastructure" na conferência Agile 2008, culminando no primeiro evento oficial de DevOps, o "DevOpsDays" em 2009 (Hutley, 2018).

A cultura DevOps é baseada na integração de pessoas, processos e tecnologias para melhorar a capacidade de uma organização em entregar aplicações e serviços com alta velocidade e qualidade. Isso envolve a adoção de práticas como integração contínua, entrega contínua, infraestrutura como código e monitoramento constante. O DevOps busca romper silos organizacionais, promovendo uma cultura de colaboração, comunicação e compartilhamento de responsabilidades entre equipes de desenvolvimento e operações (Kim et al., 2016).

Desde sua concepção, o DevOps tem transformado a maneira como as organizações desenvolvem, entregam e mantêm software. Empresas que adotam práticas DevOps relatam melhorias significativas na frequência de deploys, tempos de recuperação mais rápidos e menor

taxa de falhas em mudanças (Forsgren et al., 2018). A evolução contínua das ferramentas e práticas associadas ao DevOps, como contêineres, orquestração e automação, continua a impulsionar a eficiência e a inovação no setor de TI.

1.5.1 Contribuição de um Especialista em DevOps

Um especialista em DevOps atua na interseção entre desenvolvimento de software e operações de TI, sendo responsável por implementar práticas que automatizam e otimizam o ciclo de vida de desenvolvimento. Suas responsabilidades incluem a configuração de pipelines de CI/CD, gerenciamento de infraestrutura como código, monitoramento de sistemas e garantia de que as aplicações sejam escaláveis e resilientes. Além disso, ele promove a cultura DevOps dentro da equipe, incentivando a colaboração e a comunicação eficaz (Humble & Farley, 2010).

Este profissional deve possuir um conjunto diversificado de habilidades técnicas, incluindo conhecimento em linguagens de script, ferramentas de automação, sistemas operacionais e plataformas de nuvem. Também é essencial que tenha habilidades em gerenciamento de configuração, contêineres e orquestração com ferramentas como Docker e Kubernetes. Além das competências técnicas, um especialista em DevOps deve ter excelentes habilidades interpessoais para facilitar a colaboração entre equipes multidisciplinares (Luz & Palma, 2019).

A inclusão de um especialista em DevOps na equipe agrega valor ao acelerar o processo de desenvolvimento e deployment, reduzir erros e aumentar a confiabilidade dos sistemas. Ele atua como um catalisador para a adoção de práticas modernas de desenvolvimento, permitindo que a organização responda rapidamente às mudanças de mercado e às necessidades dos clientes. Isso resulta em maior eficiência operacional, melhor qualidade de software e vantagem competitiva (Leite, 2020).

CAPÍTULO II

Metodologia

Este capítulo apresenta as ferramentas utilizados e as metodologias adotadas para a implementação do projeto em Computação em Nuvem com Kubernetes na AWS de uma aplicação em microserviço gerenciada e apoiada pela filosofia DevOps. A seguir, detalhamos as etapas do desenvolvimento do projeto focando na infraestrutura utilizada e automaçoes em pipelines para a integração contínua e entrega contínua (CI/CD), bem como a integração do ambiente para análise de logs e métricas. Todas as abordagens estão fundamentadas nas melhores práticas da área, conforme descrito por autores renomados como Buyya et al. (2013) e Maréchal (2020).

2.1 Desenvolvimento do Projeto

A implementação do projeto segue as metodologias da filosofia DevOps e Ágil, conforme proposto por Forsgren et al. (2018). A filosofia DevOps promove a colaboração contínua entre as equipes de desenvolvimento e operações, facilitando a automação e a melhoria contínua dos processos. A metodologia Ágil, por sua vez, permite uma adaptação rápida às mudanças e uma entrega incremental de funcionalidades, o que é essencial para projetos que envolvem tecnologias dinâmicas como microserviços e computação em nuvem.

2.1.1 Metodologia Ágil Kanban

Optou-se pela metodologia Ágil Kanban para o desenvolvimento do projeto, devido à sua flexibilidade e foco na visualização do fluxo de trabalho. Kanban permite identificar e eliminar gargalos no processo de desenvolvimento, promovendo uma melhoria contínua. As equipes utilizaram quadros Kanban para monitorar o progresso das tarefas, facilitando a comunicação e a colaboração entre os membros. Essa abordagem está alinhada com as diretrizes de alta eficiência em projetos de TI descritas por Sutherland e Schwaber (2013).

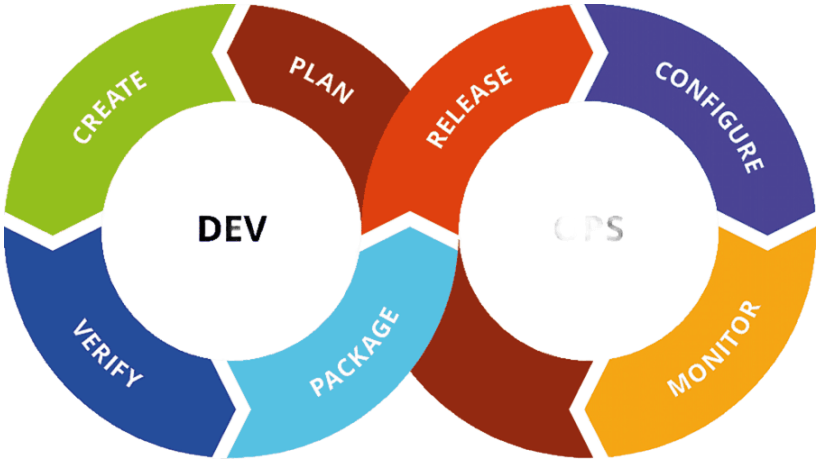
De acordo com Highsmith (2010), metodologias ágeis promovem flexibilidade e entregas contínuas, permitindo ajustes conforme as demandas evoluem. No contexto deste trabalho, o Kanban foi escolhido por ser uma metodologia visual e adaptável, permitindo a visualização do progresso e o gerenciamento eficiente de cada etapa. As equipes utilizaram um quadro Kanban com seis colunas: "Backlog/To Do" (tarefas a serem feitas), “Planejamento/Plan” (análise de requisitos e metrificação em tempo de execução), "Em Progresso/Develop" (tarefas em desenvolvimento ou em progresso), "Em Testes/ Test" (tarefas em fase de validação pelo cliente, homologação ou QA) e "Concluído/Done" (tarefas finalizadas).

Figura 5: Exemplo de metodologia agil Kanban

2.1.2 Filosofia DevOps

A filosofia DevOps permeia todo o desenvolvimento do projeto, integrando as práticas de desenvolvimento e operações para criar um ciclo contínuo de entrega e melhoria. O ciclo DevOps é composto pelas seguintes etapas: Planejar (Plan), Criar (Create), Verificar (Verify), Empacotar (Package), Liberar (Release), Configurar (Configure) e Monitorar (Monitor).

Figura 6: Diagrama da filosofia DevOps e comportamento em atuações.



- **Planejar (Plan)**: Definição de requisitos e planejamento das funcionalidades a serem desenvolvidas, utilizando ferramentas como o Jira para organizar as tarefas.
- **Criar (Create)**: Desenvolvimento do código dos microserviços, seguindo as melhores práticas de programação e padrões de arquitetura.
- **Verificar (Verify)**: Realização de testes automatizados para garantir a qualidade e a funcionalidade do código, utilizando frameworks de teste adequados.
- **Empacotar (Package)**: Containerização das aplicações utilizando Docker, preparando-as para serem implantadas em ambientes de produção.
- **Liberar (Release)**: Implantação das aplicações nos ambientes de produção através das pipelines de CI/CD, garantindo que as novas versões sejam disponibilizadas de forma eficiente.
- **Configurar (Configure)**: Configuração contínua dos ambientes utilizando ferramentas como Terraform, assegurando que a infraestrutura esteja sempre atualizada.
- **Monitorar (Monitor)**: Monitoramento das aplicações e da infraestrutura para identificar e resolver problemas rapidamente, utilizando ferramentas como Prometheus e Grafana.

Essa abordagem cíclica permite uma integração contínua e uma entrega rápida de valor, reduzindo o tempo de resposta às mudanças e melhorando a qualidade do software, conforme destacado por Kim et al. (2016).

2.1.3 DevSecOps: Integração da Segurança nas Pipelines

Além das práticas DevOps tradicionais, o projeto incorpora a filosofia DevSecOps, que integra a segurança em todas as etapas do ciclo de desenvolvimento. DevSecOps garante que a segurança não seja um complemento, mas uma parte fundamental do processo de desenvolvimento, desde o planejamento até a implantação e monitoramento.

Para implementar a segurança nas pipelines de CI/CD, foram utilizadas diversas ferramentas que automatizam a verificação de vulnerabilidades e asseguram a conformidade com as políticas de segurança. Entre as ferramentas adotadas estão:

- **Snyk**: Utilizada para análise de dependências e identificação de vulnerabilidades em bibliotecas e frameworks.
- **OWASP ZAP**: Ferramenta de teste de segurança que realiza varreduras em aplicações web para identificar falhas comuns.
- **SonarQube**: Integrada nas pipelines para análise contínua da qualidade do código e detecção de possíveis vulnerabilidades.
- **Aqua Security**: Utilizada para a segurança de containers, garantindo que as imagens Docker estejam livres de vulnerabilidades antes da implantação.
- **HashiCorp Vault**: Gerencia segredos e credenciais de forma segura, integrando-se às pipelines para fornecer acesso controlado aos recursos sensíveis.

A integração dessas ferramentas nas pipelines de CI/CD assegura que cada etapa do desenvolvimento seja verificada quanto à segurança, prevenindo a introdução de vulnerabilidades e garantindo a integridade das aplicações. Essa abordagem está alinhada com as recomendações de Chappell (2018), que enfatiza a importância de incorporar a segurança de forma contínua no ciclo de vida do desenvolvimento de software.

2.2 Desenvolvimento da Infraestrutura com Terraform

Para configurar e automatizar a infraestrutura na AWS, utilizou-se o Terraform, que permite definir infraestrutura como código (IaC), uma prática que oferece maior controle,

automação e rastreamento de mudanças no ambiente. Segundo Morris (2016), a IaC elimina a necessidade de configurações manuais, garantindo consistência e facilitando a replicação da infraestrutura.

A infraestrutura criada com Terraform incluiu a configuração da rede, armazenamento e clusters Kubernetes no Amazon Elastic Kubernetes Service (EKS). A configuração foi dividida em três arquivos principais:

- **main.tf**: Define os recursos principais da AWS necessários para o projeto, como VPC, subnets, grupos de segurança, instâncias EC2, e o cluster EKS.
- **variables.tf**: Declara as variáveis utilizadas no `main.tf`, permitindo a parametrização e reutilização dos scripts.
- **output.tf**: Define as saídas do Terraform, facilitando o acesso às informações essenciais após o provisionamento dos recursos.

A utilização do Terraform agiliza o processo de configuração da infraestrutura, garantindo consistência e reduzindo a possibilidade de erros manuais, conforme destacado por Maréchal (2020)

2.3 Desenvolvimento das Pipelines CI/CD

Para o desenvolvimento das pipelines de integração e entrega contínuas, utilizou-se o GitHub como repositório central para controle de versão do código e gerenciamento colaborativo das aplicações. As pipelines foram construídas com GitHub Actions, uma plataforma de automação integrada ao GitHub que permite configurar e executar fluxos de trabalho em resposta a eventos de commits ou pull requests. Nas fases de análise, foram empregadas ferramentas de DevSecOps para verificar a segurança e qualidade do código, assegurando que vulnerabilidades e padrões inadequados sejam identificados antes do build. Em seguida, a fase de build compila o código e gera o artefato, que é armazenado em um registry específico para ser reutilizado e versionado conforme necessário. Finalmente, na etapa de deploy, o artefato é entregue ao ambiente de

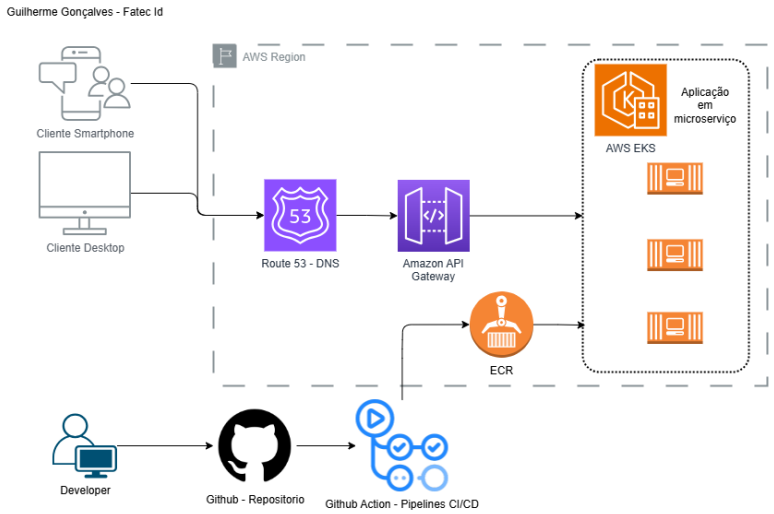
produção ou de teste, proporcionando um ciclo de desenvolvimento seguro, automatizado e eficiente, alinhado com as melhores práticas de DevOps.

As equipes de desenvolvimento, uma focada em Frontend e a outra no Backend, foram organizadas em repositórios separados no GitHub. Essa separação permitiu uma gestão mais eficiente dos projetos e a implementação de pipelines específicas para cada linguagem. Cada repositório foi configurado com as devidas permissões e integrações necessárias para suportar o fluxo de trabalho das equipes, alinhando-se com as práticas recomendadas por Gannon (2019), logo o versionamento de código foi gerenciado utilizando Git para assegurar que todas as modificações sejam registradas de maneira organizada, promovendo a transparência e a responsabilidade dentro das equipes, conforme destacado por Chacon e Straub (2014).

2.3.1 Significado de CI e CD nas Pipelines

As pipelines de CI (Integração Contínua) e CD (Entrega Contínua ou Deploy Contínuo) são fundamentais para a automação e eficiência no processo de desenvolvimento de software. A Integração Contínua envolve a prática de integrar regularmente as alterações de código em um repositório compartilhado, onde são automaticamente testadas e validadas. Isso permite a detecção precoce de erros e a garantia de que o código esteja sempre em um estado funcional.

Figura 8: Representação da integração continua (CI) e entrega continua (CD).



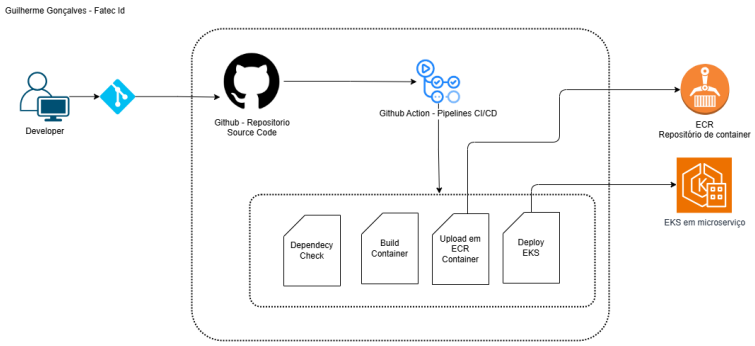
A Entrega Contínua estende a CI ao automatizar a preparação e a disponibilização das alterações de código para a produção. Isso inclui processos como a construção, testes e

implantação, assegurando que as novas funcionalidades possam ser lançadas de forma rápida e confiável. A implementação de CI/CD nas pipelines permite uma entrega ágil de software, reduzindo o tempo de desenvolvimento e aumentando a qualidade das entregas.

2.3.2 Estágios da Pipeline

As pipelines de CI/CD foram configuradas para realizar uma série de estágios que garantem a qualidade e a segurança do software desenvolvido. Cada estágio é projetado para automatizar partes específicas do ciclo de vida do desenvolvimento, promovendo eficiência e consistência.

Figura 9: Representação de estágios de uma pipeline CI/CD



- **Teste de Código Unitário**: Este estágio executa testes automatizados para verificar a funcionalidade individual das unidades de código. Garantir que cada componente funcione corretamente é essencial para a integridade do sistema como um todo.
- **Build com Libraries e Frameworks**: Durante o build, o código é compilado e as dependências necessárias são integradas. Este processo assegura que todas as bibliotecas e frameworks utilizados estão atualizados e compatíveis com o projeto.
- **Teste de Segurança**: Este estágio realiza verificações de vulnerabilidade para identificar e mitigar possíveis riscos de segurança. Ferramentas como Snyk e OWASP ZAP são utilizadas para automatizar a análise de segurança, garantindo que o software esteja protegido contra ameaças conhecidas.
- **Deploy**: Finalmente, o estágio de deploy automatiza a implantação das aplicações nos ambientes de produção. Utilizando ferramentas como GitHub Actions, as alterações aprovadas são distribuídas de forma rápida e segura, reduzindo o tempo de inatividade e melhorando a experiência do usuário.

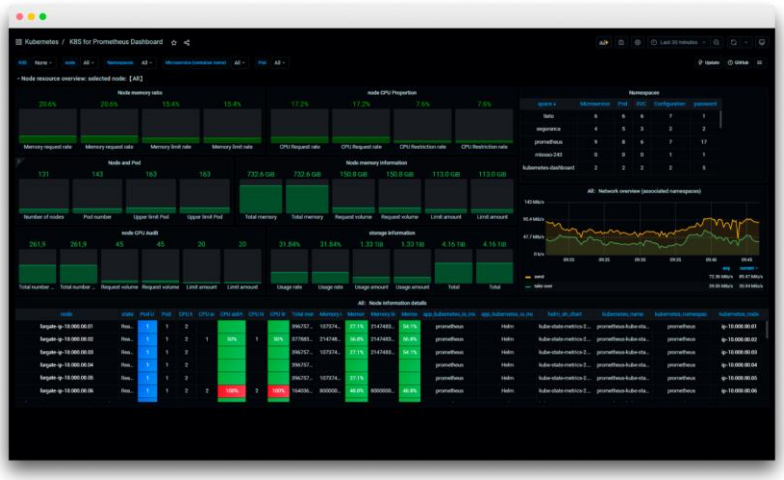
Esses estágios, quando integrados em uma pipeline automatizada, permitem uma entrega contínua de software de alta qualidade, alinhando-se com as melhores práticas de DevOps e garantindo que as aplicações sejam lançadas de maneira eficiente e segura.

2.4 Monitoramento e Observabilidade

2.4.1 Importância do Monitoramento

O monitoramento é uma componente crucial na gestão de sistemas de software modernos, especialmente em arquiteturas de microserviços implantadas em ambientes de nuvem. Ele permite a observação contínua do desempenho e da saúde das aplicações, identificando problemas antes que impactem os usuários finais. Ferramentas de monitoramento fornecem insights valiosos sobre o comportamento do sistema, auxiliando na tomada de decisões informadas para otimizar recursos e melhorar a performance.

Figura 10: Representação de monitoramento para cluster Kubernetes.



2.4.2 Métricas dos Serviços para Nuvem

As métricas são indicadores quantitativos que refletem o desempenho e a eficiência dos serviços em nuvem. Elas incluem dados como tempo de resposta, taxa de erro, uso de CPU e memória, entre outros. Monitorar essas métricas é essencial para garantir que os serviços estejam operando dentro dos parâmetros esperados e para identificar áreas que necessitam de otimização. A coleta e análise contínua dessas métricas permitem uma gestão proativa, evitando problemas de desempenho e garantindo a disponibilidade das aplicações.

2.4.3 Observabilidade e Uso de SRE

A observabilidade vai além do monitoramento, englobando a capacidade de compreender o estado interno de um sistema a partir de suas saídas externas. Ela envolve a coleta e análise de logs, traces e métricas para obter uma visão holística do comportamento do sistema. A prática de Site Reliability Engineering (SRE) utiliza princípios de engenharia para melhorar a confiabilidade e a eficiência dos sistemas, aplicando técnicas de automação e monitoramento avançado. A observabilidade é fundamental para implementar práticas de SRE eficazes, permitindo a identificação rápida e a resolução de incidentes.

2.4.4 Métricas no Uso em Negócios: Análise de UX e Experiência do Usuário

Além das métricas técnicas, é importante considerar as métricas de negócios que avaliam a experiência do usuário (UX) e a satisfação do cliente. Essas métricas incluem dados sobre a interação dos usuários com a aplicação, como tempo de carregamento, facilidade de navegação e taxas de conversão. Analisar essas métricas permite entender como os usuários percebem e utilizam a aplicação, identificando oportunidades para melhorar a UX e aumentar a satisfação do cliente. Integrar essas análises com as métricas técnicas proporciona uma visão completa do desempenho da aplicação, alinhando os objetivos técnicos com os objetivos de negócios.