PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

CONTAINERS, DOCKER E KUBERNETES: FUNDAMENTOS

Estêvão de Faria Rodrigues, Victor Reis Carlota, Marcus Vinícius Carvalho De Oliveira

Nome professor(es) orientador(es): Daniel Pereira

efrodrigues@sga.pucminas.br, vcarlota@sga.pucminas.br,
marcus.oliveira.1320200@sga.pucminas.br, …, [3]email@professor_orientador

Resumo

Este trabalho aborda os conceitos fundamentais e a aplicação prática de containers, Docker, Docker Compose e Kubernetes, tecnologias essenciais para o desenvolvimento moderno e a entrega contínua de software. Apresenta-se uma revisão teórica sobre o funcionamento dessas tecnologias, suas vantagens e casos de uso. Na parte prática, é apresentada a execução de uma aplicação conteinerizada servida por um servidor NGINX com conteúdo HTML estático, orquestrada com Docker Compose e posteriormente migrada para Kubernetes, demonstrando de forma visual e objetiva o funcionamento das tecnologias de containers.

Palavras-chave: *Containers; Docker; Docker-compose; Kubernetes; Orquestração; infraestrutura.*

1. Introdução

A transformação digital e a crescente adoção de metodologias ágeis impulsionaram mudanças significativas na forma como aplicações são desenvolvidas, testadas, entregues e mantidas. Neste contexto, surgem os containers como uma resposta à demanda por ambientes de execução mais leves, portáteis e consistentes. Eles permitem que aplicações sejam empacotadas juntamente com todas as suas

dependências, garantindo que funcionem da mesma maneira em diferentes ambientes e sistemas operacionais.

A adoção de containers viabilizou uma nova forma de pensar infraestrutura de software. Com o Docker, é possível criar e distribuir imagens de aplicações de forma simples e eficiente, enquanto o Docker Compose permite a definição de ambientes completos com múltiplos serviços interligados. Para ambientes maiores e em produção, o Kubernetes fornece uma solução robusta e escalável de orquestração, permitindo automação no deployment, monitoramento, recuperação de falhas, atualização contínua e balanceamento de carga.

Este trabalho apresenta os fundamentos dessas três tecnologias e mostra sua aplicação prática em um cenário simples, mas representativo: a execução de uma aplicação HTML estática servida por NGINX, inteiramente empacotada e executada dentro de containers. A proposta é demonstrar, sem necessidade de escrever código de backend, como as tecnologias se integram para fornecer um ambiente funcional, confiável e escalável. O objetivo é evidenciar a aplicabilidade dos conceitos estudados de forma visual, acessível e tangível.

2. REFERENCIAL TEÓRICO

O avanço da virtualização e a crescente demanda por ambientes de software portáveis e escaláveis fizeram surgir os containers como uma das soluções mais revolucionárias da atualidade. A construção de sistemas confiáveis, reprodutíveis e com rápida implantação passou a ser um requisito cada vez mais comum em empresas e instituições. Nesse contexto, diversas ferramentas surgiram para oferecer suporte a esse novo paradigma. Esta seção apresenta os fundamentos, características e implicações do uso de containers, Docker, Docker Compose e Kubernetes na construção e operação de sistemas modernos.

Antes de tratar dos conceitos das tecnologias que serão exploradas ao longo da pesquisa, é necessário compreender alguns termos técnicos e princípios que sustentam esse novo paradigma. Entre eles, destacam-se: virtualização, isolamento de processos, imagens de container, orquestração e infraestrutura como código. Compreender esses conceitos permite perceber a lógica por trás das soluções analisadas e compreender o motivo de seu crescimento exponencial no mercado de tecnologia.

A virtualização, em seu sentido mais amplo, refere-se à criação de representações abstratas de recursos computacionais físicos. Isso pode incluir servidores, sistemas operacionais, dispositivos de armazenamento e redes. Essa prática permitiu a otimização do uso de recursos de hardware e abriu caminho para a consolidação de servidores, redução de custos e aumento da flexibilidade operacional. Os containers surgem como uma evolução dessa ideia, fornecendo isolamento mais leve e direto para aplicações.

O isolamento de processos é outro conceito essencial. Ele permite que diferentes aplicações ou serviços rodem simultaneamente em um mesmo sistema sem interferência mútua. Nos containers, esse isolamento é alcançado através do uso de namespaces e cgroups, tecnologias nativas do kernel Linux. Isso garante que cada container funcione como se fosse um sistema independente, com seus próprios arquivos, processos e redes, sem o peso de uma máquina virtual completa.

Outrossim, o conceito de orquestração diz respeito à forma como múltiplos containers são coordenados para compor um sistema funcional. À medida que as aplicações crescem em complexidade, torna-se necessário gerenciar escalonamento, disponibilidade, balanceamento de carga, atualizações e monitoramento. É aí que entram ferramentas como o Docker Compose, para ambientes locais ou simples, e o Kubernetes, para ambientes de produção e de grande escala. A compreensão desses elementos proporciona a base para discutir com propriedade os temas abordados neste estudo. os fundamentos, características e implicações do uso de containers, Docker, Docker Compose e Kubernetes na construção e operação de sistemas modernos.

2.1. **Containers**

Containers são estruturas leves de virtualização que isolam processos e aplicações em ambientes padronizados e independentes do sistema operacional host. Eles operam utilizando recursos como namespaces, cgroups e union file systems, compartilhando o kernel do sistema base, o que reduz o consumo de recursos computacionais. Ao contrário das máquinas virtuais, que incluem o sistema operacional completo, os containers compartilham o mesmo núcleo do host, tornando-os muito mais rápidos e leves para iniciar e executar.

Essa leveza faz com que containers sejam ideais para ambientes que exigem escalabilidade dinâmica, permitindo iniciar novas instâncias de forma quase instantânea. Isso é especialmente útil em sistemas baseados em microserviços, onde cada serviço pode ser isolado em seu próprio container, garantindo independência, facilidade de manutenção e substituição de partes da aplicação sem afetar o todo. Essa independência também favorece a modularidade e a reutilização de componentes em diferentes contextos.

Containers também contribuem para a padronização dos ambientes de desenvolvimento, teste e produção. Um dos maiores desafios em projetos de software é a incompatibilidade entre ambientes. Com containers, a aplicação é empacotada com todas as suas dependências, garantindo que funcione da mesma forma em qualquer lugar. Essa característica é fundamental para a adoção de práticas de DevOps, onde a automação do pipeline de entrega contínua exige ambientes previsíveis e estáveis.

É necessário falar também da imutabilidade das imagens de container. Uma vez criada, uma imagem não muda, o que permite versionamento e rastreabilidade total do que está sendo executado. Isso facilita rollback em caso de falhas e aumenta a confiabilidade dos ambientes, além de permitir testes reprodutíveis e consistentes em diferentes estágios do ciclo de vida do software

2.2 **Docker**

Docker é a principal plataforma de containerização e desempenhou papel central na popularização dessa tecnologia. Ele fornece uma interface simplificada para criação, execução e distribuição de containers, além de um ecossistema robusto com ferramentas complementares.

A estrutura básica do Docker se fundamenta em imagens, que são moldes estáticos de containers. Cada imagem é composta por camadas sobrepostas, otimizadas para reutilização e economia de espaço. Essas imagens são criadas a partir de arquivos chamados Dockerfile, que descrevem passo a passo o ambiente necessário para execução de uma aplicação. Um Dockerfile pode incluir a escolha do sistema operacional base, a instalação de dependências, cópia de arquivos, definição de variáveis de ambiente e comandos de inicialização.

O Docker Hub é o repositório oficial da plataforma, permitindo a publicação, o compartilhamento e o versionamento de imagens. Ele contém milhares de imagens oficiais e de terceiros, prontas para uso. Essa facilidade de acesso a ambientes pré-configurados acelera o desenvolvimento e a experimentação de novas tecnologias.

Através de comandos simples, é possível iniciar, parar, listar, inspecionar, remover e logar containers. Esses comandos estão na *Docker CLI;* em conjunto com APIs, esse controle pode ser automatizado e integrado a pipelines CI/CD.

2.3 **Docker Compose**

Embora o Docker por si só já seja poderoso, muitas aplicações reais demandam múltiplos serviços trabalhando em conjunto. Por exemplo, uma API pode depender de um banco de dados, de um serviço de cache e de um serviço de fila. Para lidar com essa complexidade, surgiu o Docker Compose.

Docker Compose permite definir, em um único arquivo *YAML*, todos os serviços que compõem uma aplicação. Esse arquivo especifica as imagens, volumes, redes, variáveis de ambiente, políticas de reinício e dependências entre os containers. Com um simples comando (`docker-compose up`), todo o ambiente é iniciado em segundos.

Um dos grandes benefícios do Compose é sua legibilidade e facilidade de manutenção. Ele permite a criação de ambientes consistentes entre diferentes membros de equipe, ambientes de desenvolvimento, testes e homologação. A reutilização e modularidade dos arquivos YAML também favorecem a escalabilidade do projeto e a organização dos serviços.

Compose ainda oferece suporte a volumes e redes nomeadas, permitindo que dados sejam persistidos entre execuções e que serviços comuniquem-se por nomes amigáveis. Isso elimina a necessidade de descobrir IPs ou gerenciar rotas manualmente.

2.4 **Kubernetes**

Kubernetes é uma plataforma open-source para orquestração de containers em escala. Criada originalmente pela Google, tornou-se um dos principais projetos da Cloud Native Computing Foundation (CNCF) e é amplamente utilizada por empresas de todos os portes. Seu objetivo é automatizar o deployment, escalonamento, balanceamento de carga e gerenciamento de aplicações conteinerizadas em ambientes distribuídos, tornando a operação de sistemas complexos mais previsível e resiliente.

Entre os componentes principais do Kubernetes estão os Pods, que são as menores unidades de execução e podem conter um ou mais containers; os Deployments, responsáveis por manter o estado desejado de uma aplicação em execução; e os Services, que expõem aplicações de forma estável, mesmo que os Pods por trás sejam criados ou destruídos dinamicamente. Outros recursos importantes incluem ConfigMaps, Secrets, Volumes e Ingress, que adicionam ainda mais controle e flexibilidade à configuração e operação das aplicações.

Um dos grandes diferenciais do Kubernetes é sua capacidade de autogerenciamento. Ele é capaz de monitorar o estado das aplicações em tempo real e agir automaticamente para corrigir problemas, reiniciar containers que falharam, escalar réplicas com base em carga e até redistribuir aplicações entre nós do cluster conforme necessário. Isso torna o ambiente altamente disponível e resiliente a falhas, ideal para aplicações críticas.

Ademais, o Kubernetes é extensível e possui uma ampla comunidade de plugins, ferramentas complementares e suporte de nuvens públicas como AWS, Azure e Google Cloud. Ele pode ser utilizado tanto em clusters locais para testes quanto em ambientes corporativos de produção com dezenas ou centenas de nós, mantendo o mesmo conjunto de comandos e conceitos, o que facilita a escalabilidade da operação.

A arquitetura do Kubernetes é baseada em um modelo mestre-escravo, onde um ou mais nós mestre coordenam e controlam os nós de trabalho. Esses nós de trabalho são os responsáveis pela execução efetiva dos containers. Cada componente do cluster possui responsabilidades específicas e comunica-se via APIs internas, garantindo modularidade e separação de responsabilidades. Essa divisão lógica facilita a manutenção e o monitoramento do cluster como um todo.

A configuração do Kubernetes é declarativa, o que significa que o estado desejado da aplicação é definido em arquivos YAML, e o sistema se encarrega de garantir que esse estado seja atingido e mantido. Essa abordagem promove automação e confiabilidade, pois mudanças são aplicadas de forma controlada e rastreável. Em ambientes com alta frequência de alterações, esse comportamento é fundamental para evitar erros manuais e garantir consistência.

Kubernetes oferece suporte nativo a práticas modernas de desenvolvimento, como DevOps, GitOps e integração contínua. Sua integração com ferramentas de monitoramento, logging e alertas facilita a observabilidade de aplicações em produção. Ele também permite rollouts progressivos, canary deployments e rollback automático, o que garante segurança e controle em atualizações frequentes. Esse conjunto de recursos consolida o Kubernetes como peça-chave em arquiteturas nativas da nuvem.

3. EXEMPLO PRÁTICO: SERVIDOR NGINX COM CONTAINERS

A implementação prática deste trabalho buscou demonstrar, de forma objetiva e visual, o funcionamento de containers e sua orquestração por meio de ferramentas como Docker Compose e Kubernetes. Foi desenvolvida uma aplicação estática, contendo um arquivo HTML com elementos visuais e interativos, que foi servida por um servidor web NGINX empacotado em um container. O foco principal foi a estrutura da infraestrutura e sua capacidade de ser replicada, isolada e escalável, mesmo sem uso de lógica de aplicação ou backend.

A primeira etapa consistiu na criação da imagem do container, baseada em uma imagem oficial do NGINX, adaptada para servir o conteúdo HTML personalizado. Essa imagem foi configurada com um arquivo Dockerfile contendo instruções para substituir a configuração padrão do NGINX e adicionar os arquivos estáticos. A imagem resultante serviu como base para os testes com Docker Compose, onde o ambiente foi orquestrado e executado localmente, expondo a aplicação na porta 8080 para acesso via navegador.

A definição do ambiente via Docker Compose demonstrou a clareza e simplicidade da ferramenta ao permitir a descrição completa do serviço em um arquivo YAML. Foram definidos nome do container, política de reinício, mapeamento de volumes e portas, entre outros parâmetros. A arquitetura demonstrada com Docker Compose permitiu visualizar como múltiplos serviços poderiam ser compostos facilmente em um ambiente local de testes, reforçando o papel dessa ferramenta no processo de desenvolvimento e prototipação.

Em seguida, o projeto foi migrado para o Kubernetes, com o intuito de validar sua escalabilidade e robustez em um ambiente de orquestração mais avançado. Os arquivos `nginx-deployment.yaml` e `nginx-service.yaml` foram criados para representar, respectivamente, a definição do Pod com sua imagem e o serviço que expõe a aplicação. A escolha pelo tipo de serviço NodePort permitiu acesso à aplicação pela porta 30080, simulando um cenário de produção local.

A experiência prática com Kubernetes mostrou a complexidade adicional envolvida, mas também os benefícios que ele oferece. Foi possível observar como o Kubernetes lida com o ciclo de vida dos Pods, como eles são criados, agendados e mantidos em estado desejado. O comportamento autônomo do cluster, ao reagir a alterações nos recursos, evidenciou a maturidade da plataforma para ambientes críticos e escaláveis.

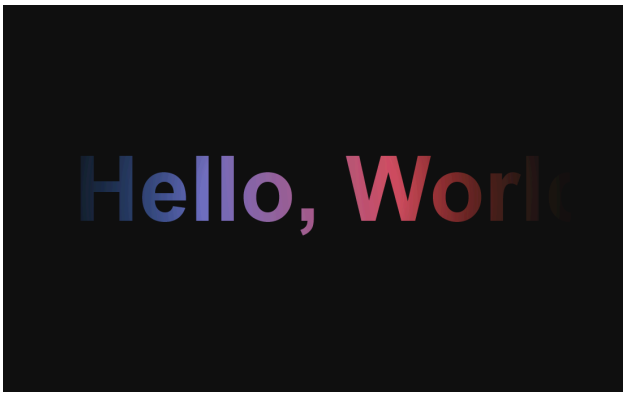

Figura 1. *Interface da aplicação HTML estática servida via container Docker.*

4. **CONCLUSÃO**

O desenvolvimento deste trabalho permitiu compreender a fundo o funcionamento e a importância das tecnologias de containers e orquestração na engenharia de software moderna. A partir do estudo e da aplicação prática de ferramentas como Docker, Docker Compose e Kubernetes, foi possível visualizar na prática como sistemas leves, portáteis e escaláveis podem ser criados com facilidade, mesmo sem a necessidade de código-fonte ou lógica de aplicação envolvida. A adoção de containers e a

automação do ambiente representam um avanço significativo em relação aos modelos tradicionais de implantação e execução de software.

Além disso, a realização do projeto com uma aplicação HTML estática demonstrou que, mesmo em cenários simples, é possível explorar os principais conceitos de isolamento, portabilidade, versionamento e escalabilidade. O uso de NGINX em container servindo conteúdo estático revelou como as práticas modernas de infraestrutura podem ser aplicadas de forma incremental e pedagógica. Essa abordagem facilita o aprendizado e evidencia a importância de entender a base da arquitetura moderna antes de partir para sistemas mais complexos e distribuídos.

A transição entre Docker Compose e Kubernetes permitiu avaliar as diferenças entre ambientes de desenvolvimento e produção, bem como os desafios e benefícios associados a cada ferramenta. Enquanto o Compose favorece agilidade e simplicidade local, o Kubernetes oferece controle, resiliência e escalabilidade para operações em larga escala. Com esse conhecimento, desenvolvedores e engenheiros estão mais preparados para tomar decisões conscientes na escolha das ferramentas mais adequadas aos seus contextos, promovendo soluções mais estáveis, eficientes e sustentáveis ao longo do tempo.

A compreensão adquirida ao longo deste trabalho reforça a relevância do domínio dessas tecnologias não apenas para o desenvolvimento de aplicações modernas, mas também para sua operação contínua em ambientes heterogêneos. O conhecimento sobre containers e orquestração se tornou uma habilidade essencial, sendo cada vez mais requisitada em equipes que buscam agilidade, automação e resiliência operacional. Isso reflete a transição de paradigmas tradicionais de infraestrutura para abordagens orientadas a serviços, escaláveis e automatizadas.

Ademais, o aprendizado prático proporcionado pela implementação de um projeto funcional, ainda que simples, contribui para a consolidação dos conceitos e a familiaridade com os fluxos de trabalho reais. A exposição às etapas de construção de imagens, definição de serviços e migração entre ferramentas reforça a capacidade crítica e técnica para lidar com desafios de infraestrutura no mercado atual. Ao dominar essas ferramentas, o profissional está apto a colaborar em projetos de qualquer porte, adotando práticas sustentáveis de entrega e operação de software.

REFERÊNCIAS

KELSEY, Hightower; BURNS, Brendan; BEDA, Joe. Kubernetes: Up & Running. O'Reilly Media, 2022. DOCKER INC. Documentação oficial. Disponível em: https://docs.docker.com KUBERNETES. Documentação oficial. Disponível em: https://kubernetes.io/docs NGINX. Docker Hub. Disponível em: https://hub.docker.com/_/nginx KOMPOSE. Kompose Tool. Disponível em: https://kompose.io/