UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM REDES DE COMPUTADORES

PAULO ARAGÃO DE AZEVEDO NETO

ANALISE DE DESEMPENHO DE CONTÊINERES LXC E DOCKER EM AMBIENTE VIRTUAL

QUIXADÁ

2025

PAULO ARAGÃO DE AZEVEDO NETO

ANALISE DE DESEMPENHO DE CONTÊINERES LXC E DOCKER EM AMBIENTE VIRTUAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Redes de Computadores do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Redes de Computadores.

Orientador: Prof. Dr. João Marcelo Uchôa de Alencar.

QUIXADÁ

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A988a Azevedo Neto, Paulo Aragão de.
Análise de desempenho de contêineres LXC e Docker em ambiente virtual / Paulo Aragão de Azevedo Neto. – 2025.
44 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Redes de Computadores, Quixadá, 2025.
Orientação: Prof. Dr. João Marcelo Uchôa de Alencar.

1. Contêineres. 2. Computação em Nuvem. 3. Benchmarking. 4. Analise de Desempenho. I. Título.
CDD 004.6

PAULO ARAGÃO DE AZEVEDO NETO

ANALISE DE DESEMPENHO DE CONTÊINERES LXC E DOCKER EM AMBIENTE VIRTUAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Redes de Computadores do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Redes de Computadores.

Aprovada em: 01/08/2025.

BANCA EXAMINADORA

Prof. Dr. João Marcelo Uchôa de Alencar (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Antonio Rafael Braga
Universidade Federal do Ceará (UFC)

Prof. Dr. Michel Sales Bonfim
Universidade Federal do Ceará (UFC)

À minha família, por sempre acreditarem em mim e me apoiarem ao longo da minha jornada acadêmica. Mãe, sua paciência nos momentos difíceis foi fundamental para que eu seguisse em frente. Pai, sua confiança de que eu seria capaz foi uma fonte constante

AGRADECIMENTOS

A minha família, por todo o amor, apoio e por acreditarem em mim em cada etapa dessa jornada. Nada disso seria possível sem vocês.

À minha mãe, Cintia Arruda Lima, pelo amor incondicional, pelo exemplo de força e pela presença constante nos momentos difíceis. Sua dedicação e carinho foram fundamentais em cada etapa desta jornada.

Ao meu pai, Paulo Aragão de Azevedo Filho, por me ensinar o valor do esforço, da disciplina e da honestidade. Sua confiança em mim sempre foi um incentivo silencioso, mas poderoso, para seguir em frente.

Aos meus amigos, Anderson Moura, Antonio Cêsar, David Machado e Felipe Pedrosa, pelo companheirismo, incentivo constante e por tornarem essa caminhada mais leve e significativa.

Ao Prof. Dr. João Marcelo Uchôa de Alencar, pela orientação durante o trabalho.

RESUMO

Este estudo tem como objetivo comparar o desempenho de duas tecnologias de conteinerização, Docker e LXC, em um ambiente de computação em nuvem na Amazon Web Services (AWS). A pesquisa busca identificar qual dessas tecnologias oferece melhor eficiência para execução de tarefas, utilizando ferramentas de benchmarking como Sysbench e FIO para avaliar o desempenho da CPU, memória e disco I/O. O trabalho aborda conceitos de conteinerização, Docker, LXC, AWS e ferramentas de benchmarking, além de descrever a configuração do ambiente virtual na AWS e a aplicação das ferramentas de benchmarking. Os resultados obtidos serão comparados e analisados para determinar qual tecnologia oferece melhor desempenho em termos de estabilidade e facilidade de uso.

Palavras-chave: Conteineres; Computação em Nuvem; Benchmarking; Analise de Desempenho.

ABSTRACT

This study aims to compare the performance of two containerization technologies, Docker and LXC, in a cloud computing environment on Amazon Web Services (AWS). The research seeks to identify which of these technologies offers better efficiency for task execution, using benchmarking tools such as Sysbench and FIO to evaluate CPU, memory, and disk I/O performance. The work covers concepts of containerization, Docker, LXC, AWS, and benchmarking tools, and describes the configuration of the virtual environment on AWS and the application of benchmarking tools. The obtained results will be compared and analyzed to determine which technology offers better performance in terms of stability and ease of use.

Keywords: Containers; Cloud computing; Benchmarking; Performance Analysis.

SUMÁRIO

1 INTRODUÇÃO

A virtualização é uma tecnologia que utiliza o mesmo servidor para hospedar múltiplas instâncias virtuais (Bhattacharya; Mittal, 2023). Atualmente, os serviços de computação em nuvem são amplamente disponibilizados, facilitando o gerenciamento e o monitoramento de diversos ambientes. Uma de suas possíveis aplicações é a criação de laboratórios virtuais, que podem ser utilizados por professores e alunos para realizar experimentos que normalmente exigiriam uma infraestrutura com alto custo de *hardware*.

A virtualização permite a criação de múltiplos serviços em uma única máquina física, o que resulta em uma alta utilização do espaço físico, baixo desempenho e tempo de inicialização elevado. Nesse contexto, surgiram as tecnologias de virtualização leve, como forma de sanar essas deficiências (Potdar *et al.*, 2020).

Tecnologias de virtualização e conteinerização são amplamente utilizadas por desenvolvedores para a criação de servidores e armazenamento de dados, buscando maior desempenho e disponibilidade, de forma que o gerenciamento e a operação desses recursos sejam mais simples. As tecnologias de virtualização tradicionais exigiam uma alta demanda das máquinas físicas, o que comprometia o desempenho. As tecnologias de virtualização leve (*containers*) vêm para solucionar essas limitações.

A função dos *containers* é fornecer um ambiente de execução genérico, independentemente da infraestrutura utilizada (Ruan *et al.*, 2016). Por conta disso, a conteinerização facilita o uso dessas ferramentas por estudantes que desejam realizar tarefas com objetivos acadêmicos ou testes recorrentes.

Identificar quais tecnologias serão utilizadas em um experimento é de vital importância para garantir que os resultados estejam dentro do esperado, além de prevenir imprevistos durante a implementação do projeto. Dessa forma, este trabalho busca auxiliar professores e estudantes na escolha da tecnologia de virtualização leve que apresenta melhor desempenho em um ambiente de computação em nuvem na Amazon (*AWS*).

O verdadeiro poder do Docker está no fato de que, embora ele possa substituir alguns aspectos das ferramentas mais tradicionais, geralmente é compatível com elas ou tem sua eficiência aumentada quando combinado a elas também (Kane; Matthias, 2015, p.5). Tendo em vista as vantagens da virtualização, tecnologias de conteinerização como Docker e LXC surgiram para facilitar a execução de *software* em ambientes virtuais isolados, com a premissa de oferecer melhor desempenho aos usuários.

Diante desse cenário, e com o intuito de identificar o desempenho entre as tecnologias de virtualização, tem-se a seguinte questão de pesquisa: qual o método mais eficiente para realizar operações de virtualização em *containers*?

Para responder a essa pergunta, serão elaborados diversos testes no ambiente virtual da Amazon (*AWS*), onde os cenários serão configurados por meio de um *script*, desde a criação das instâncias até a execução dos *benchmarks* de desempenho, comparando as tecnologias de conteinerização (Docker e LXC) a fim de identificar qual delas oferece maior estabilidade e facilidade de uso.

1.1 Objetivos

1.1.1 Objetivo Geral

Este trabalho tem como objetivo realizar uma avaliação de desempenho entre virtualizações feitas por meio de *containers* em plataformas de computação em nuvem. O foco é descobrir qual dessas tecnologias apresenta maior capacidade para a execução de atividades com fins educacionais.

1.1.2 Objetivos Específicos

- Planejar os experimentos que serão executados
- Identificar tecnologias de *Benchmark* a serem utilizadas.
- Identificar quais tipos de instâncias serão utilizadas

1.2 Organização do trabalho

- Capítulo 2: Apresenta a fundamentação teórica, detalhando as tecnologias e plataformas que serão utilizadas na pesquisa.
- Capítulo 3: Mostra trabalhos que abordam objetivos semelhantes e as métricas adotadas para medição de desempenho.
- Capítulo 4: Detalha a metodologia utilizada para o desenvolvimento do trabalho.
- Capítulo 5: Detalha os resultados obtidos na realização do trabalho.
- Capítulo 6: Apresenta as conclusões do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos fundamentais para o trabalho proposto. A seção 2.1 explica o conceito de conteinerização e o compara à forma como os contêineres são executados. A seção 2.2 apresenta uma breve introdução sobre a plataforma Docker, explicando seus conceitos de conteinerização para criação e gerenciamento de ambientes isolados. A seção 2.3 mostra o LXC, outra forma de conteinerização que terá seu desempenho comparado com o Docker. A seção 2.4 apresenta a plataforma de serviços de computação em nuvem AWS, assim como os serviços de virtualização disponibilizados por ela. A seção 2.5 define quais ferramentas de *benchmark* serão utilizadas no trabalho e quais métricas serão coletadas e comparadas a fim de medir o desempenho das tecnologias.As seções 2.6 e 2.7 apresentam as ferramentas de coleta e visualização Prometheus e Grafana.

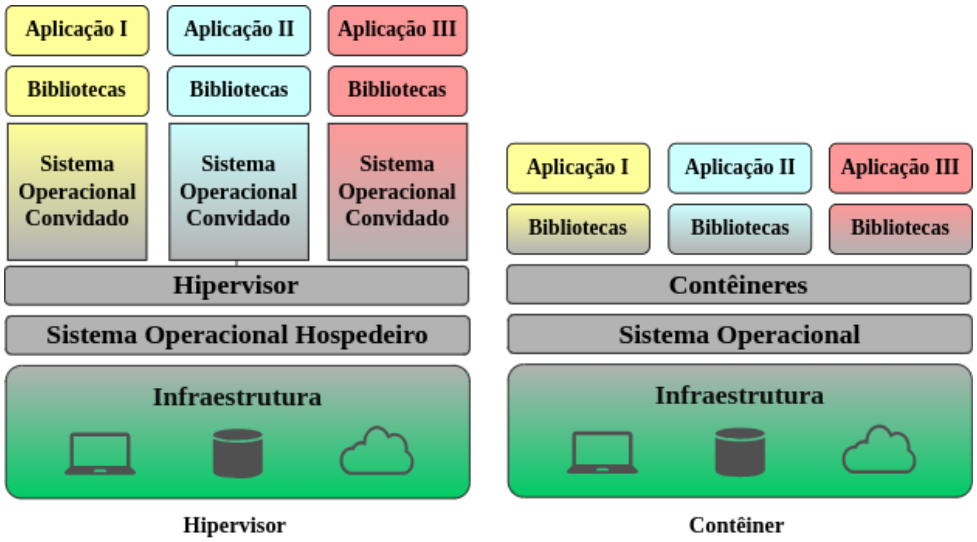
2.1 Conteinerização

A conteinerização é uma tecnologia que permite executar aplicativos e suas dependências em um ambiente isolado, chamado de contêiner. Esse processo garante que o *software* funcione de maneira consistente, independentemente do ambiente em que está sendo executado, como diferentes servidores ou sistemas operacionais.

A conteinerização pode ser classificada entre contêineres de aplicação e contêineres de sistema. Os contêineres de aplicação (por exemplo, Docker) são projetados para encapsular uma única tarefa em uma imagem padrão para distribuir aplicativos de maneira eficaz. Para ser mais específicos, os contêineres de aplicação simplificam um contêiner tanto quanto possível para um processo único para executar microsserviços. No entanto, os contêineres de sistema (por exemplo, LXC) são projetados para fornecer sistemas operacionais totalmente funcionais com os serviços mais utilizados. De certa forma, os contêineres de sistema são como uma máquina virtual, mas com um design mais leve (Ruan *et al.*, 2016).

Ao contrário da virtualização tradicional, onde se cria uma máquina virtual completa com seu próprio sistema operacional, os contêineres compartilham o *kernel* do sistema hospedeiro, o que os torna mais leves e eficientes. Isso reduz o consumo de recursos e melhora o desempenho, tornando a execução e o gerenciamento de múltiplas instâncias de contêineres mais ágeis. A figura 1 demonstra a diferença de estrutura entre uma máquina virtual tradicional com hipervisores e uma feita através de contêineres.

Figura 1 – VM x Contêiner



Fonte: Elaborado pelo Autor

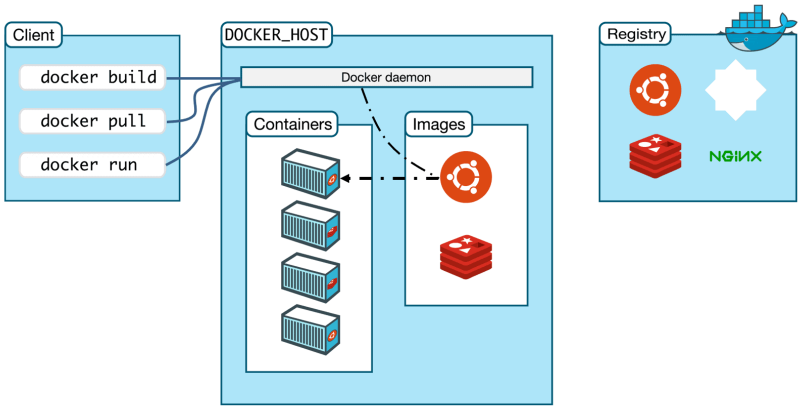
2.2 Docker

O Docker é uma plataforma de código aberto para a criação, implantação, gerenciamento e execução de aplicativos que podem ser organizados em forma de contêineres, tecnologia que combina o aplicativo, as dependências relacionadas e as bibliotecas do sistema organizadas para construir na forma de um contêiner (Potdar *et al.*, 2020).

A criação de ambientes isolados fornecida pelo Docker permite que as aplicações sejam executadas com mais eficiência em diferentes sistemas operacionais ou ambientes em nuvem. A conteinerização aumenta a portabilidade e a escalabilidade das aplicações, permitindo que os desenvolvedores tenham mais controle e facilidade de gerenciamento.

O Docker possui diversas ferramentas para o auxílio da criação de ambientes virtuais através de contêineres. Dentre elas, discutiremos nas subseções seguintes o Docker Engine, o Docker Compose , Cadvisor e o Docker Hub.

Figura 2 – Arquitetura Docker



Fonte: k21academy, 2022

2.2.1 Docker Engine

O Docker Engine é uma tecnologia de conteinerização cliente-servidor instalada na máquina hospedeira para construir contêineres de aplicações. Ela é composta por:

- Docker Daemon: Responsável pela construção, execução e distribuição dos contêineres;
- Docker REST API: Interface que permite interação com os contêineres;
- Docker CLI: O meio de comunicação com o Daemon através de linhas de comando;

2.2.2 Docker Compose

O Docker Compose é uma ferramenta utilizada para definir e executar aplicações com múltiplos contêineres. O Compose simplifica o controle das aplicações, facilitando o gerenciamento de serviços. Utilizando um arquivo de configuração YAML, é possível especificar as configurações de vários contêineres, o que inclui imagens, volumes, redes, variáveis de ambiente e dependências.

Além disso, ele elimina a necessidade de gerenciar manualmente múltiplos comandos para iniciar contêineres interdependentes, reduzindo erros e economizando tempo. Também suporta a escalabilidade, permitindo aumentar ou reduzir o número de instâncias de um serviço com facilidade, o que é útil para testar o comportamento de sistemas em diferentes cargas de trabalho.

2.2.3 Docker Hub

O Docker Hub funciona como um repositório público para compartilhamento e gerenciamento de imagens Docker, onde os desenvolvedores podem hospedar imagens de suas aplicações, possuindo assim acesso facilitado às imagens. O Docker Hub economiza tempo ao fornecer imagens pré-criadas que podem ser utilizadas como base para novas aplicações.

2.2.4 Cadvisor

O cAdvisor (*Container Advisor*) é uma ferramenta de monitoramento de contêineres desenvolvida pelo Google. Ele coleta, processa e exporta métricas em tempo real sobre o desempenho de contêineres, como uso de CPU, memória, rede e sistema de arquivos. O cAdvisor é integrado nativamente ao Docker e a outros ambientes de contêineres, fornecendo uma interface *web* simples para visualização dos dados.

Essa ferramenta é especialmente útil para administradores de sistemas e desenvolvedores que precisam monitorar o consumo de recursos de seus contêineres. Além disso, o cAdvisor pode ser integrado com ferramentas como Prometheus e Grafana para armazenamento e visualização avançada de métricas, ajudando na identificação de gargalos e otimização de desempenho em ambientes baseados em contêineres.

2.3 LXC

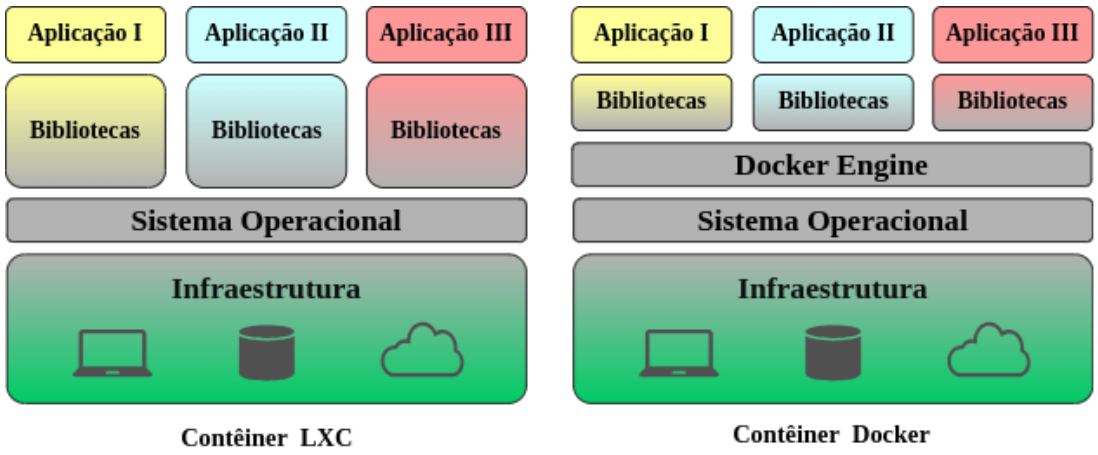
LXC (Linux Contêineres) é uma tecnologia de virtualização em nível de sistema operacional que utiliza *namespaces* e *cgroups* para isolar e gerenciar recursos como CPU, memória e rede. Os autores em Ruan *et al.* (2016) destacam que LXC permite criar e gerenciar contêineres leves e eficientes, oferecendo um desempenho próximo ao nativo em ambientes de nuvem. Este tipo de contêiner é particularmente adequado para fornecer uma infraestrutura de sistema completo dentro de um ambiente isolado, minimizando o *overhead* em comparação com máquinas virtuais tradicionais e facilitando a execução de serviços e aplicativos de forma segura e eficiente.

LXC e Docker são tecnologias de virtualização baseadas em contêineres, mas diferem no foco e no nível de abstração. O LXC é projetado para criar contêineres de sistema, oferecendo um ambiente semelhante a uma máquina virtual, onde é possível executar um sistema operacional completo, com maior controle sobre o isolamento do sistema. Ele é mais adequado para

administradores de sistemas que precisam de ambientes robustos e flexíveis. Por outro lado, o Docker é voltado para contêineres de aplicação, sendo ideal para empacotar e executar aplicações isoladas com suas dependências em contêineres leves e rápidos.

O LXC é uma forma de conteinerização que utiliza *namespaces* e *cgroups* do *kernel* Linux para isolar processos e controlar o uso de recursos. Assim como o Docker, o LXC proporciona um ambiente isolado capaz de criar, gerenciar e executar aplicações sem que haja interferências entre si. Tal isolamento também proporciona maior segurança para as aplicações. A figura 3 mostra como a tecnologia de conteinerização do LXC é estruturada em comparação ao Docker.

Figura 3 – LXC x Docker



Fonte: Elaborado pelo Autor

2.3.1 LXD

LXD é uma extensão da tecnologia de contêineres LXC que adiciona uma camada de gerenciamento e funcionalidades adicionais. Enquanto o LXC se concentra na criação e gerenciamento de contêineres em nível de sistema operacional, o LXD visa simplificar a administração de contêineres, proporcionando uma interface mais amigável por meio de uma API REST e uma CLI mais intuitiva. Os autores em Silva *et al.* (2024) discutem que o LXD integra funcionalidades avançadas como migração ao vivo de contêineres, controle refinado de recursos e suporte a *snapshots*, o que o torna uma solução ideal para ambientes corporativos e para gerenciar grandes números de contêineres de maneira mais eficiente e segura.

2.3.2 LXD Exporter

O LXD Exporter é um componente que coleta métricas de contêineres e máquinas virtuais gerenciadas pelo LXD (*Linux Container Daemon*) e as disponibiliza em um formato compatível com o Prometheus. Ele atua como um intermediário, extraindo dados como uso de CPU, memória, disco e rede dos contêineres LXD e transformando-os em métricas que podem ser consumidas por sistemas de monitoramento.

Esse *exporter* é essencial para quem utiliza o LXD em ambientes de produção, pois permite a integração com ferramentas como Prometheus e Grafana para monitoramento em larga escala. Com ele, é possível acompanhar o desempenho de múltiplos contêineres, identificar problemas de alocação de recursos e garantir a estabilidade de ambientes baseados em LXD.

2.4 AWS

Amazon Web Services (AWS) é uma plataforma de computação em nuvem oferecida pela Amazon. Ela fornece um ambiente para que os desenvolvedores criem, testem e gerenciem aplicações através de ambientes virtuais.

Dentre os diversos serviços fornecidos pela plataforma, temos dois principais a serem abordados neste trabalho, sendo eles o *Amazon Elastic Compute Cloud* (EC2) e o *Amazon Elastic Container Service* (ECS).

2.4.1 Amazon EC2

O serviço EC2 da AWS é uma plataforma popular de computação em nuvem. Ele permite que os clientes paguem proporcionalmente ao uso dos recursos. O EC2 se refere a cada VM como uma instância.

Como essas instâncias ou VMs não possuem armazenamento constante, os dados são perdidos quando elas são encerradas. Por isso, é sempre recomendado usar o EC2 em conjunto com o *Amazon Elastic Block Store* (EBS) ou o S3, que oferecem armazenamento durável para as instâncias do EC2 (Jubury, 2022).

Esta ferramenta é responsável por boa parte das funcionalidades da plataforma AWS, sendo nela onde os desenvolvedores podem:

- Hospedar Aplicações.
- Desenvolver e Testar Ambientes.
- Processar Dados.

Além da flexibilidade em poder configurar minuciosamente suas instâncias, também é muito simples escalar recursos. A Amazon fornece diversas zonas de disponibilidade para a hospedagem das instâncias, o que garante um melhor desempenho.

2.4.2 Amazon ECS

O Amazon ECS é um serviço de gerenciamento de contêineres disponibilizado pela AWS para facilitar a implementação de aplicações em contêineres Docker. Com o ECS, é possível definir as configurações de execução dos contêineres, assim como o monitoramento e um ajuste de escala automática de acordo com a demanda do usuário.

2.5 Benchmarks

Benchmarks são ferramentas com testes padronizados feitas para avaliar o desempenho de sistemas, softwares e hardwares. Eles fornecem métricas que permitem a comparação da eficiência, velocidade e capacidade de processamento das tecnologias em um ambiente controlado. Neste trabalho, as ferramentas de *benchmark* a serem utilizadas são Sysbench e o *Flexible IO Tester* (Fio).

2.5.1 Sysbench

O Sysbench é uma ferramenta de *benchmarking* multiplataforma que permite aos administradores de sistemas avaliar o desempenho do hardware e do sistema operacional, simulando cargas de trabalho específicas. Ele pode testar a CPU, a memória, o I/O de disco e a performance de banco de dados, entre outras métricas.

Em Mavridis e Karatza (2021), os autores utilizam benchmarks como Sysbench e Fio para avaliar o desempenho de CPU, disco e memória em tecnologias como Kata *Containers*, gVisor e Nabla. O objetivo é analisar o comportamento dessas *runtimes* de contêineres *sandboxed*, comparando seu isolamento e eficiência em cenários de computação em nuvem *multitenant* e arquiteturas *serverless*.

2.5.2 Fio

A benchmark Fio (*Flexible I/O Tester*) é uma ferramenta de teste de desempenho de E/S (entrada/saída) amplamente utilizada para avaliar o desempenho de discos rígidos, SSDs, sistemas de armazenamento em nuvem e outros dispositivos de armazenamento. O Fio permite simular diferentes cargas de trabalho, como leitura sequencial, escrita aleatória e operações mistas com alto nível de personalização.

A combinação de Sysbench e Fio permite uma análise abrangente de desempenho em ambientes multitenant, como demonstrado por (Viktorsson *et al.*, 2020).

2.6 Prometheus

O Prometheus é uma ferramenta de monitoramento e alerta de código aberto amplamente utilizada para coletar, armazenar e analisar métricas de sistemas e aplicações. Desenvolvido originalmente pela *SoundCloud*, o Prometheus foi projetado para oferecer visibilidade em tempo real do desempenho e comportamento de infraestruturas de TI. Ele armazena dados em um *banco de dados próprio baseado em séries temporais*, permitindo a consulta eficiente por meio de sua poderosa linguagem de consulta, o *PromQL*.

Com suporte para coleta de métricas através de *"exporters"* e *"targets"*, o Prometheus é altamente eficaz em arquiteturas modernas baseadas em *microsserviços e sistemas distribuídos*. Além disso, ele oferece funcionalidades integradas de *alertas* que ajudam equipes a responder rapidamente a incidentes. A capacidade de operar de forma autônoma e seu modelo de dados flexível tornam o Prometheus uma escolha ideal para empresas que buscam monitorar a performance de suas aplicações e garantir a disponibilidade de seus serviços.

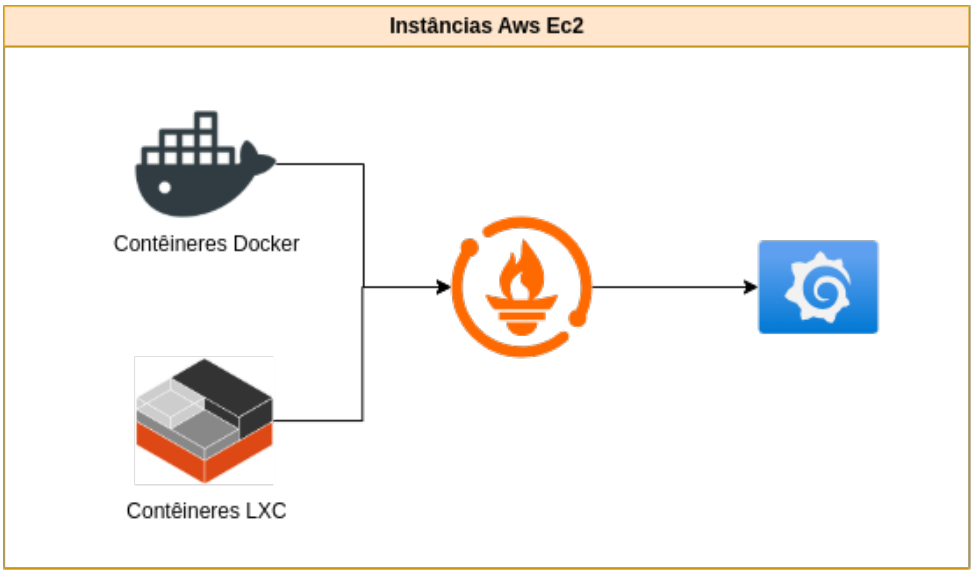
EmGedia e Perigo (2018), os autores utilizaram o Prometheus para coletar as métricas dos containers em conjunto com *exporters* como o Cadvisor e o *node exporter*. Depois das métricas serem coletadas e devidamente armazenadas elas poderiam ser visualizadas no Grafana.

2.7 Grafana

O Grafana é uma poderosa plataforma de visualização de dados projetada para transformar métricas brutas em gráficos interativos e *dashboards* intuitivos. Ele permite que equipes monitorem a performance de sistemas e aplicações em tempo real, criando painéis personalizados que facilitam a interpretação de informações complexas. Compatível com diversas fontes de dados, como *Prometheus, InfluxDB, Elasticsearch e MySQL*, o Grafana se destaca por sua flexibilidade e capacidade de integrar diferentes sistemas de monitoramento em um único ambiente visual.

Além da visualização, o Grafana oferece recursos avançados, como *alertas configuráveis*, compartilhamento de *dashboards* e suporte a consultas dinâmicas, permitindo que usuários personalizem completamente a experiência de monitoramento. Sua interface amigável e altamente configurável o torna uma escolha popular entre *equipes de DevOps, SREs e administradores de sistemas*, ajudando a identificar problemas rapidamente e garantir a estabilidade das infraestruturas críticas. A Figura 4 exemplifica como seria um ambiente envolvendo as tecnologias prometheus e grafana na AWS.

Figura 4 – Prometheus Grafana



Fonte: Elaborado pelo Autor

3 TRABALHOS RELACIONADOS

Neste capítulo serão apresentados alguns trabalhos relacionados à análise de desempenho de tecnologias de conteinerização, sejam elas através de aplicações como Docker e LXC ou até mesmo em ambientes de desenvolvimento em nuvem, como a AWS (ECS).

3.1 Performance Evaluation of Docker Container and Virtual Machine

No artigo Potdar *et al.* (2020), é afirmado que tecnologias de virtualização leve, como a conteinerização, surgiram como forma de atender à alta demanda de técnicas rápidas de virtualização para desenvolvimento. O artigo apresenta motivos para o uso da conteinerização, sendo eles o tamanho das máquinas virtuais , a redução de desempenho quando múltiplas máquinas são operadas ao mesmo tempo e o longo tempo necessário para a inicialização das máquinas.

Quadro 1 – Máquinas Virtuais versus Contêineres Docker

Característica	**Máquinas Virtuais**	**Contêineres Docker**
Nível de Isolamento	*Hardware*	Sistema Operacional
Sistema Operacional	Separado	Compartilhado
Tempo de Inicialização	Longo	Curto
Uso de Recursos	Maior	Menor
Imagens Pré-construídas	Difíceis de encontrar e gerenciar	Disponíveis no Docker Hub
Imagens Personalizadas	Difíceis de construir	Fáceis de construir
Tamanho	Maior (contém SO completo)	Menor (apenas o necessário)
Mobilidade	Fácil migração para novo *host*	Destruídos e recriados ao invés de movidos
Tempo de Criação	Mais longo	Questão de segundos

Fonte: Adaptado de Potdar et al. (2020)

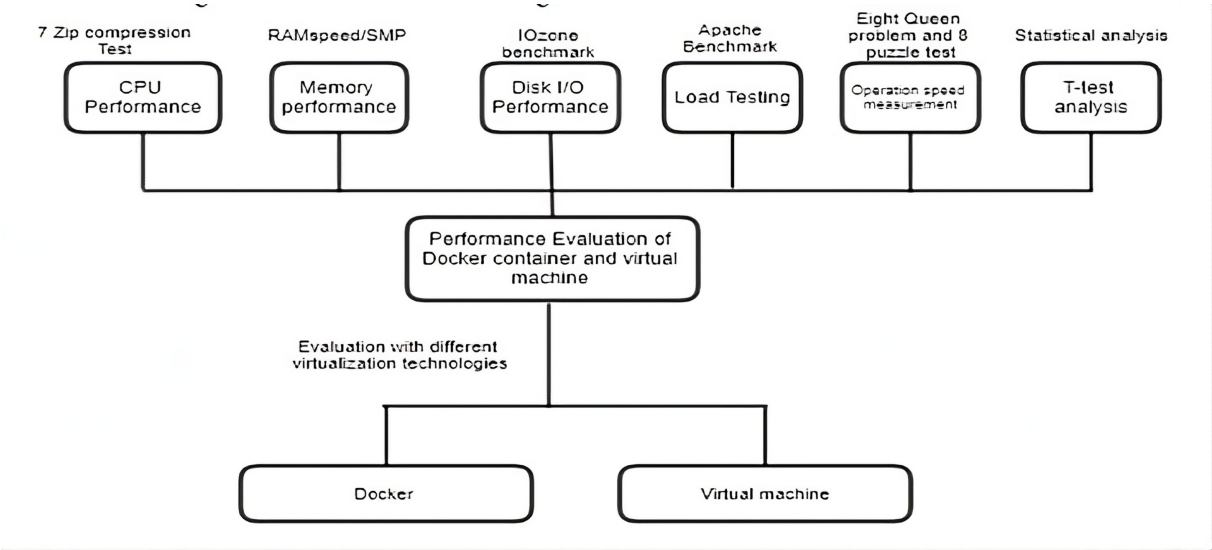
Entrando mais a fundo sobre a tecnologia Docker, os autores explicam e diferenciam as máquinas, virtuais e contêineres. Como visto no Quadro 1, Máquinas virtuais tradicionais são executadas no hardware da máquina nativa, enquanto os contêineres são executados no sistema operacional (SO) o que faz com que eles possuam um tempo de execução menor, assim como pouco gasto de memória.

Na implementação do artigo são detalhadas as medidas de cinco métricas para medir o desempenho entre máquinas virtuais tradicionais à base de hipervisores e máquinas virtualizadas através de contêineres utilizando a plataforma Docker. As métricas em questão são:

- Desempenho da CPU;
- Desempenho da Memoria;
- Desempenho do Disco I/O;
- Teste de Carga;
- Medição da Velocidade de Operação;

Para a medição de desempenho entre *KVM(Kernel virtual machine)* e Docker, foram utilizadas três ferramentas de *benchmarking*, sendo elas Sysbench , Phoronix , Apache *benchmark*. As medições foram realizadas num servidor tendo como sistema operacional nativo o Windows 10, as máquinas a serem comparadas em cada tecnologia foram instaladas e configuradas com Ubuntu 16.04 para o procedimento dos testes. A figura 5 demonstra como está organizada a estrutura do trabalho.

Figura 5 – Ferramentas de benchmarking



Fonte: Potdar *et al.* (2020)

Como conclusão, os autores determinaram, após a análise dos testes, que o Docker teve um desempenho superior em comparação com a máquina virtual tradicional, demonstrando melhor eficiência em todos os testes anteriormente mencionados.

3.2 Performance comparison between container-based and VM-based services

No artigo Salah *et al.* (2017) descreve uma análise comparativa entre contêineres no Amazon ECS (EC2 *Container Service*) e máquinas virtuais (VMs) no Amazon EC2. Os autores realizam uma análise de desempenho focada em três principais métricas, sendo elas:

- Vazão (*Throughput*);
- Tempo de Resposta;
- Utilização de CPU;

Os autores destacam como a computação em nuvem se tornou preferencial para a construção de aplicações, trazendo benefícios como maior resiliência, escalabilidade e eficiência no uso de recursos.

Na documentação da AWS indica que os contêineres no ECS (*Elastic Container Service*) são executados sobre VMs do EC2, em vez de hardware *bare-metal*, o que motivou os autores a investigar essa configuração e suas implicações de desempenho.

Para a realização do experimento, os autores configuraram as instâncias utilizando o Amazon ECS na região de Tóquio. Em seguida, foram realizados três cenários de teste, sendo eles:

- Cenário 1: Um único serviço web em uma instância contêiner e uma estrutura EC2 equivalente com Apache, ambas com instâncias de tipo t2.small.
- Cenário 2: Dois serviços web em contêineres separados rodando em portas diferentes e hospedados na mesma instância contêiner, replicado em uma estrutura similar no EC2.
- Cenário 3: Três serviços web rodando simultaneamente em uma instância contêiner com portas adicionais configuradas, replicado também em uma instância EC2.

A ferramenta de medição escolhida foi o JMeter, requisições de clientes foram simuladas para registrar a vazão e o tempo de resposta. A CPU foi monitorada utilizando a ferramenta *sar* no Linux.

Como conclusão, o estudo revela que, surpreendentemente, os serviços em contêineres no ECS apresentam desempenho inferior aos serviços em VMs no EC2.

3.3 Toward Optimal Virtualization: An Updated Comparative Analysis of Docker and LXD Container Technologies

Em Silva *et al.* (2024) os autores realizam a comparação entre tecnologias de contêinerização de aplicação (Docker) e de sistemas (LXD) em cenários de virtualização. A pesquisa tem como objetivo quantificar o desempenho e as limitações de cada tecnologia, desse modo auxiliando a escolha de soluções ideais para as respectivas necessidades de cada organização ou empresa.

As seguintes métricas foram coletadas pelos autores :

- Desempenho da CPU
- Desempenho do Disco I/O
- Vazão

Os experimentos foram realizados em dois sistemas de hardware: um laptop com Intel Core i7 e um *desktop* com AMD Ryzen 7, ambos com 16 GB de RAM e SSD NVMe. Os sistemas operacionais utilizados foram Windows 10 e distribuições Linux (Garuda Linux e Arch Linux). Para avaliar o desempenho, foram usados os *benchmarks* Geekbench 6 (processamento), CrystalDiskMark (I/O de armazenamento) e iPerf3 (comunicação de rede) .

Os testes revelaram que os *containers* Docker e LXD apresentam desempenho maduro e baixa sobrecarga em comparação aos ambientes nativos. O Docker teve melhor desempenho em cenários de aplicações leves e de núcleo único, enquanto o LXD destacou-se em cenários de múltiplos processos. Ambos são adequados para diferentes demandas de virtualização, com o LXD favorecendo aplicações de maior complexidade e o Docker sendo ideal para cenários de CI/CD e *microservices*.

3.4 A Performance Study of Containers in Cloud Environment

Os autores em Ruan *et al.* (2016) comparam o desempenho de tecnologias de conteinerização Docker e LXC. Para isso, foram conduzidos diversos testes através de *benchmarks* para obter as métricas de :

- Desempenho da CPU
- Desempenho do Disco I/O
- Latência de Rede

Os resultados mostraram que contêineres de sistema (LXC) são mais adequados para

suportar cargas de trabalho de I/O em comparação com contêineres de aplicação (Docker), devido à latência de I/O mais alta associada ao Docker. Além disso, executar contêineres diretamente no *bare metal* resulta em uma degradação severa do desempenho de disco e rede, enquanto contêineres em máquinas virtuais introduzem latências adicionais. O estudo sugere que, para aplicações de alto desempenho que necessitam de baixo *overhead*, os contêineres de sistema são preferíveis. A avaliação do serviço público de contêineres (ECS e GKE) revelou que ECS oferece melhor desempenho que GKE, indicando que plataformas de nuvem poderiam obter melhor desempenho ao rodar contêineres diretamente no bare metal.

3.5 Quadro Comparativo

O Quadro 2 abaixo apresenta a comparação entre o que foi feito em cada trabalho apresentado na seção anterior e este trabalho.

Quadro 2 – Comparação entre este trabalho e estudos relacionados

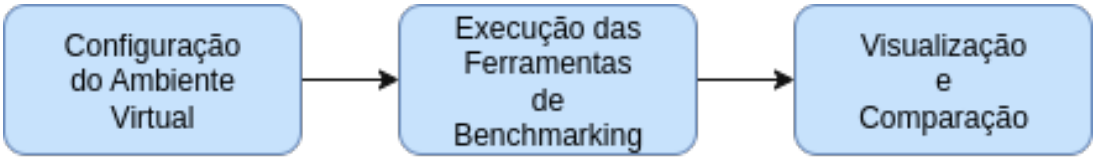
Critério	**Este Trabalho**	**Potdar et al. (2020)**	**Salah et al. (2017)**	**Ruan et al. (2016)**	**Silva et al. (2024)**
AWS	Sim	Não	Sim	Sim	Sim
Docker	Sim	Sim	Sim	Sim	Sim
LXC	Sim	Não	Sim	Não	Não
Desempenho CPU	Sim	Sim	Sim	Sim	Sim
Desempenho Memória	Sim	Sim	Não	Sim	Sim
Desempenho do Disco I/O	Sim	Sim	Sim	Não	Não

Fonte: Elaborado pelo Autor

4 METODOLOGIA

Este capítulo descreve a metodologia utilizada para o desenvolvimento do trabalho. Ele abrange a configuração do ambiente virtual na AWS, a aplicação das ferramentas de *benchmarking* e a visualização e comparação dos dados obtidos. A Figura 6 demonstra a sequência entre esses passos.

Figura 6 – Diagrama Metodológico

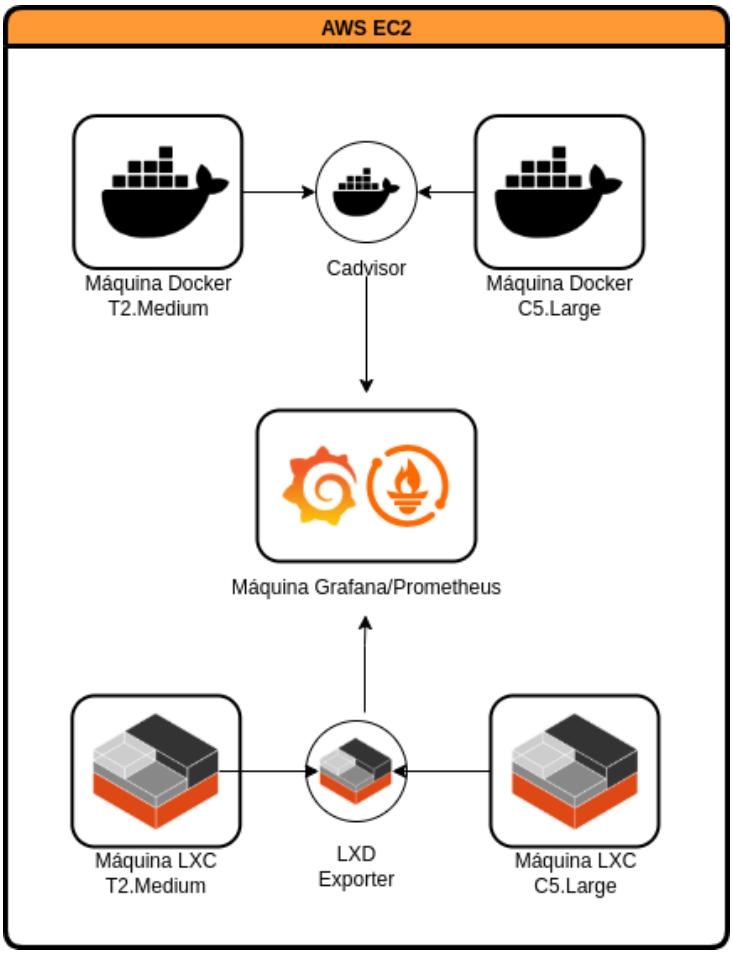


Fonte: Elaborado pelo Autor

4.1 Configuração do ambiente virtual

Os testes de desempenho propostos neste trabalho serão realizados na plataforma virtual da Amazon (AWS), onde serão configuradas cinco instâncias através do Amazon EC2, com sistemas operacionais Ubuntu 22.04; uma instância será responsável pela coleta dos dados através do Prometheus e os respectivos *exporters* de cada tecnologia de contêineres e, posteriormente, os dados serão visualizados pelo Grafana. Os tipos de instâncias que serão utilizados para as quatro máquinas restantes serão T2.Medium e C5.large; ambas possuem 2 vCPUs e 4 GB de memória RAM, porém a C5.large oferece desempenho consistente e alto poder de processamento, sendo melhor para cargas de trabalho intensivas e estáveis. Ao contrário da T2.Medium, a C5.large tem rede mais rápida (10 Gbps) e CPU dedicada. Após a configuração, as instâncias terão instaladas as tecnologias Docker e LXC. A Figura 7 traz uma ilustração de como o ambiente será configurado.

Figura 7 – Ambiente Virtual



Fonte: Elaborado pelo Autor

4.2 Ferramentas de Benchmarking

Como visto anteriormente, para medir o desempenho entre os contêineres Docker e LXC serão utilizadas ferramentas de *benchmarking* específicas para cada aspecto do desempenho, sendo eles:

- Desempenho da CPU;
- Desempenho da memória;
- Desempenho do disco I/O.

Em relação à configuração dos contêineres, foram utilizadas imagens de Ubuntu 22.04 disponibilizadas por padrão em cada tecnologia.

4.2.1 Desempenho da CPU

Para a medição do desempenho da CPU, será utilizada a ferramenta de *benchmarking* Sysbench. As subseções abaixo demonstram quais testes serão realizados e como serão executa-

dos. Após a realização dos testes, as ferramentas retornarão métricas de tempo de execução e operações por segundo, que serão utilizadas posteriormente para análise gráfica.

4.2.1.1 Teste de CPU com Sysbench

O *Sysbench* será aplicado para realizar um teste de CPU baseado em cálculos de números primos, avaliando o desempenho da máquina. O teste será realizado com um valor de 20.000 para identificação de números primos e será executado com 4 *threads*, com o objetivo de avaliar a capacidade da máquina em processar grandes volumes de dados. Abaixo está o comando para realizar o teste:

```
$ docker ou lxc exec "$containerid" sysbench cpu --cpu-max-
   prime=20000 --threads=4 --time=$testduration run
```

As variáveis *containerid* e *testduration* representam, respectivamente, um indicador de qual *container* deve ser testado e por quanto tempo o teste deverá ser executado. Neste trabalho, o tempo padrão utilizado é de 120 segundos.

4.2.2 Desempenho da Memória

A avaliação do subsistema de memória foi conduzida por meio da ferramenta **Sysbench**, amplamente utilizada em *benchmarks* sintéticos por sua flexibilidade e precisão na simulação de cargas reais de trabalho. O objetivo do teste foi mensurar a taxa de transferência de leitura e escrita em blocos contínuos de memória, o que permite inferir a eficiência do sistema na manipulação intensiva de dados em RAM.

Os testes foram executados em ambientes isolados, utilizando tanto *containers* Docker quanto *containers* LXC, com o propósito de comparar o desempenho da memória sob diferentes tecnologias de virtualização em nível de sistema operacional. O comando utilizado para a realização do teste foi o seguinte:

```
docker ou lxc exec "$container_id" sysbench memory
--memory-block-size=1K
--memory-total-size=10G
--time=$testduration run
```

Descrição dos parâmetros:

- `containerid`: Representa o identificador do *container* em que o teste foi executado.
- `memory-block-size`: Define o tamanho dos blocos de memória utilizados em cada operação (1 kilobyte).
- `memory-total-size`: Indica o volume total de dados a ser processado durante o teste (10 gigabytes).
- `time=testduration`: Determina o tempo máximo de execução do teste, em segundos, controlado por uma variável de ambiente.

Durante a execução, o `Sysbench` realiza leituras e escritas sequenciais na memória, utilizando múltiplos blocos em paralelo, de modo a simular cenários realistas de carga computacional. Como saída, a ferramenta fornece métricas como a **taxa de transferência** (em MiB/s), **número de operações por segundo** e **latência média por operação**, que são essenciais para a análise comparativa entre os diferentes ambientes de *container*.

A aplicação do mesmo procedimento tanto em instâncias Docker quanto em LXC visa identificar variações de desempenho decorrentes das diferenças estruturais entre os dois modelos de virtualização, contribuindo para uma análise mais precisa da eficiência de cada abordagem no gerenciamento de memória.

4.2.3 Desempenho do Disco I/O

A avaliação do desempenho de entrada e saída de dados em disco (I/O) foi realizada por meio da ferramenta **FIO** (Flexible I/O Tester), amplamente utilizada em ambientes acadêmicos e profissionais para simular cargas de trabalho realistas em dispositivos de armazenamento.

Os testes foram executados em ambientes isolados utilizando tecnologias de containerização distintas, sendo selecionados *containers* **Docker** ou **LXC** de acordo com a configuração da máquina em análise. O comando utilizado para a realização do teste foi:

```
lxc ou docker exec "$container" -- fio
--name=disk_test
--rw=randrw
--bs=4k
--size=100M
--runtime=$TEST_DURATION
--time_based
--output-format=json
```

Descrição dos parâmetros utilizados:

- `container`: Identificador do container onde o teste foi executado (Docker ou LXC).
- `name=disktest`: Nome atribuído à tarefa de teste, utilizado para organização dos resultados.
- `rw=randrw`: Define o padrão de acesso como leitura e escrita aleatórias, simulando cargas reais e imprevisíveis de I/O.
- `bs=4k`: Especifica o tamanho do bloco de leitura e escrita como 4 kilobytes.
- `size=100M`: Determina o tamanho total do arquivo de teste como 100 megabytes.
- `runtime=testduration`: Define a duração do teste, em segundos, controlada por variável de ambiente.
- `timebased`: Indica que o teste será executado por tempo (e não até consumir todo o arquivo).
- `output-format=json`: Gera a saída dos resultados em formato JSON, facilitando a análise e integração com ferramentas de visualização.

Durante a execução, o `FIO` realiza operações de leitura e escrita aleatória com blocos de 4 KB, uma configuração que simula o comportamento típico de aplicações que realizam múltiplas operações simultâneas em sistemas de arquivos. A saída do teste inclui métricas detalhadas, tais como:

- **Taxa de IOPS (*Input/Output Operations Per Second*)**: Quantidade de operações por segundo realizadas em leitura e escrita.
- ***Throughput***: Volume de dados processados por segundo, geralmente em KB/s ou MB/s.
- **Latência**: Tempo médio de resposta por operação, reportado em microssegundos (µs).

A utilização do FIO em diferentes tecnologias de *conteiner* permite comparar o

impacto que cada ambiente isolado exerce sobre o desempenho de disco, fornecendo subsídios técnicos para decisões sobre a escolha da plataforma mais adequada em cenários que exigem alto desempenho de I/O.

4.3 Visualização e Análise dos Resultados

Após a execução dos testes, os resultados são coletados pelo Prometheus e visualizados em *dashboards* no Grafana, permitindo uma análise detalhada do desempenho comparativo entre Docker e LXC. Essa abordagem facilita a identificação de padrões, gargalos e variações significativas entre os dois ambientes de *container*.

Para uma análise estatística mais aprofundada, os dados são processados em Python utilizando as bibliotecas *pandas* para manipulação dos dados, *matplotlib*e *seaborn* para visualização. Essas ferramentas possibilitam a geração de gráficos comparativos, como séries temporais de consumo de recursos e distribuições de desempenho, auxiliando na detecção de tendências e *outliers*.

A fim de validar a significância estatística das diferenças observadas, foi aplicado um teste T de *Student* para amostras independentes. O teste foi escolhido por sua adequação a comparações entre duas médias quando os dados seguem uma distribuição aproximadamente normal e as variâncias podem ser consideradas homogêneas. O tamanho da amostra, composta por 100 medições para cada tecnologia, garante um poder estatístico suficiente para detectar diferenças relevantes com um nível de confiança de 95%.

A escolha de 100 amostras para cada tecnologia buscou equilibrar precisão e praticidade. Esse número é suficiente para garantir resultados confiáveis com boa margem de confiança, reduzindo os efeitos do acaso nas comparações. Além disso, permite aplicar testes estatísticos com segurança, assegurando que as conclusões reflitam diferenças reais de desempenho. Durante a análise, não foram identificados outliers significativos, o que reforça a consistência dos dados coletados.

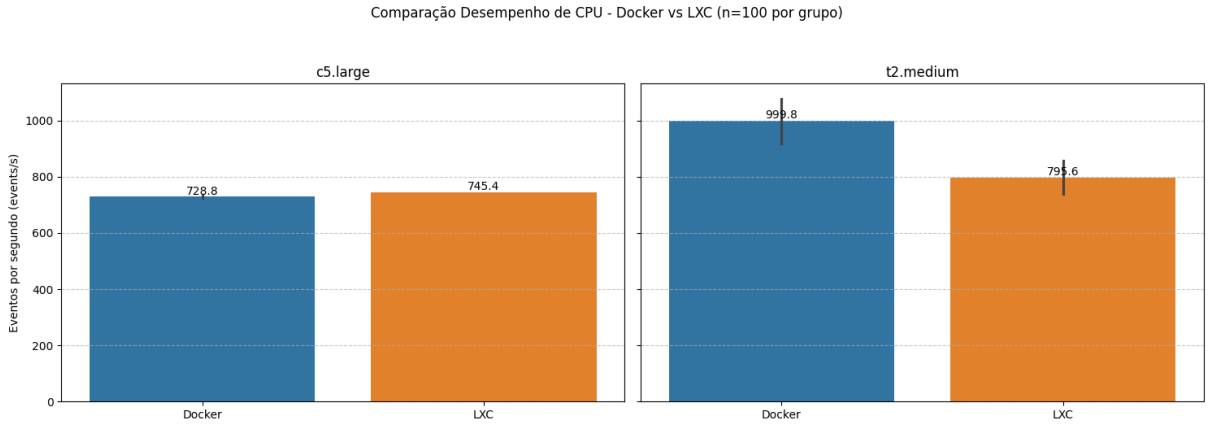
Os resultados são consolidados em gráficos comparativos e tabelas estatísticas, destacando métricas como tempo de resposta, uso de CPU e memória, além da significância das diferenças encontradas. Essa metodologia assegura uma avaliação robusta e baseada em evidências, permitindo conclusões confiáveis sobre o desempenho relativo do Docker e do LXC nos cenários testados.

5 RESULTADOS

5.1 Análise do Desempenho de CPU

Nesta seção, são apresentados os resultados dos testes de desempenho de CPU, executados em containers Docker e LXC. Para uma análise abrangente, os testes foram realizados em dois tipos distintos de instância: `c5.large` e `t2.medium`. Cada tipo de instância representa um ambiente com características específicas de processamento e modelos de alocação de recursos de CPU, que influenciam diretamente a performance. Foram coletadas 100 amostras por configuração (combinação de tecnologia e tipo de instância), com o objetivo de comparar o desempenho dessas tecnologias de virtualização leve sob diferentes condições de infraestrutura.

Figura 8 – Comparação de Desempenho de CPU - Docker vs LXC por Tipo de Instância.



Fonte: Elaborado pelo Autor

A Figura 8 apresenta a comparação do desempenho de CPU entre as tecnologias Docker e LXC para os tipos de instância `c5.large` e `t2.medium`. Os resultados, expressos em eventos por segundo, revelam diferenças tanto no desempenho quanto na sua consistência:

- **Instância `c5.large`:**
  - **Docker:** Obteve um desempenho médio de **728,8 eventos/s**. A barra de desvio padrão, de tamanho reduzido, indica uma performance extremamente consistente e estável nesse ambiente.
  - **LXC:** Registrou um desempenho médio de **745,4 eventos/s**, muito semelhante ao Docker em `c5.large`.

  As instâncias Amazon EC2 C5 oferecem alto desempenho a um custo acessível, com uma baixa relação preço/desempenho computacional, sendo ideais para cargas de traba-

lho intensivas como HPC, processamento em lote, codificação de vídeo e inferência de aprendizado de máquina (Djordjevic *et al.*, 2025). Ambas as tecnologias de contêiner demonstraram desempenho semelhante e notavelmente estável, com pouca variabilidade.

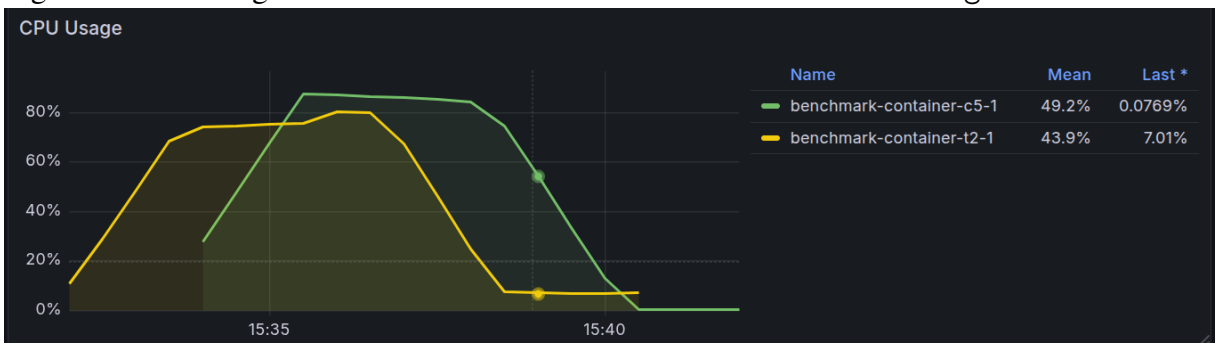
- **Instância `t2.medium`:**
  - **Docker:** Alcançou um desempenho médio de **993,8 eventos/s**, o maior desempenho observado em todos os testes. No entanto, o alto desvio padrão indica uma variabilidade considerável nos resultados, sugerindo flutuações em torno da média.
  - **LXC:** Obteve um desempenho médio de **791,6 eventos/s**. Embora inferior ao Docker nesta instância, ainda é superior ao seu desempenho na `c5.large`. O LXC também apresentou variabilidade, embora aparentemente menor do que a do Docker nessa instância.

  A família T de instâncias oferece performance de CPU de linha de base com capacidade de "burst"— isto é, picos temporários de desempenho — conforme necessário. Essa arquitetura é ideal para workloads de uso geral com utilização moderada de CPU (Amazon Web Services, 2025). Assim, as `t2.medium` mostram desempenho variável, com o Docker explorando melhor essa capacidade de expansão, embora com menor previsibilidade em comparação ao LXC ou às instâncias `c5.large`.

5.1.1 Análise do Uso de CPU por Tecnologia e Tipo de Instância

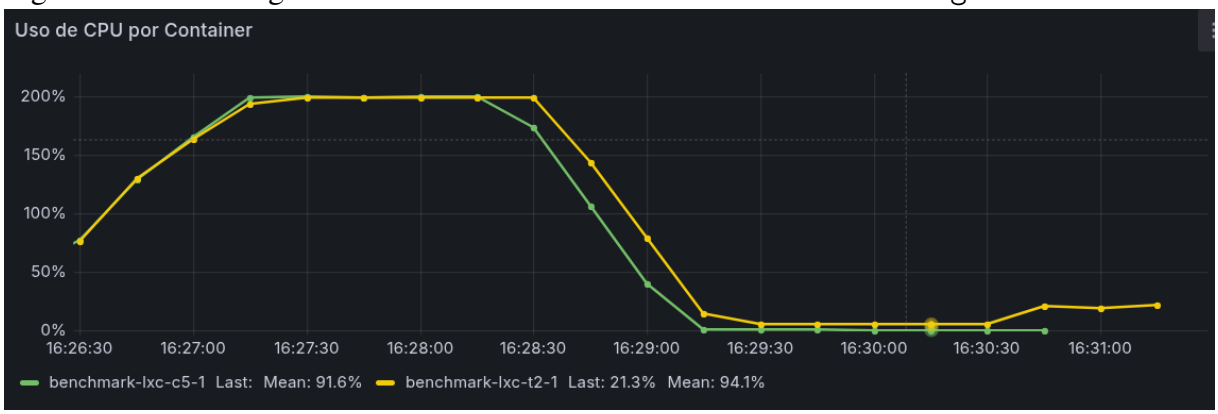
Para complementar a análise de desempenho em eventos por segundo, foi realizado o monitoramento do uso de CPU durante os benchmarks. Esse monitoramento ajuda a compreender como cada tecnologia (Docker e LXC) e tipo de instância (`c5.large` e `t2.medium`) utilizam os recursos de processamento disponíveis.

Figura 9 – Porcentagem de Uso de CPU - Docker em `t2.medium` e `c5.large`



Fonte: Elaborado pelo Autor

Figura 10 – Porcentagem de Uso de CPU - LXC em `t2.medium` e `c5.large`



Fonte: Elaborado pelo Autor

Uso de CPU pelos Containers Docker

A Figura 9 mostra o uso de CPU dos *containers* Docker nas instâncias `t2.medium` (linha azul) e `c5.large` (linha verde). Nota-se que ambos atingem cerca de **80%** de uso de CPU. Esse padrão sugere que, na configuração utilizada, ou por limitações do Docker no uso de *cgroups* e alocação de CPU, os *containers* utilizaram efetivamente apenas um vCPU, mesmo quando havia múltiplos núcleos disponíveis.

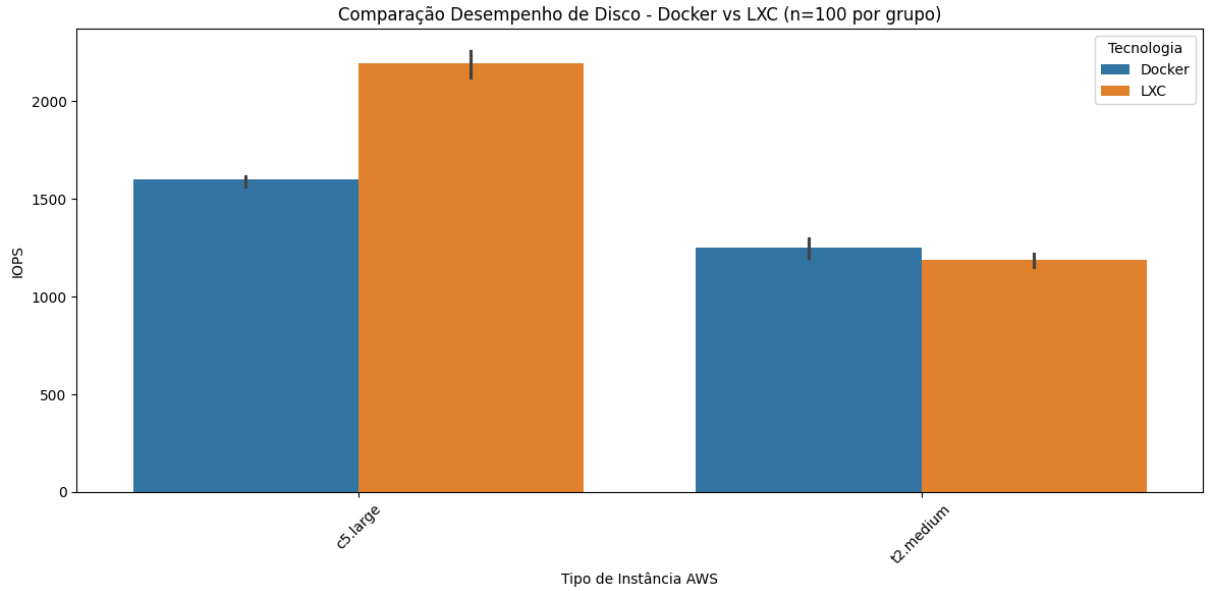
Uso de CPU pelos Containers LXC e Múltiplos Núcleos

Em contraste, a Figura 10 revela que os *containers* LXC alcançaram picos sustentados de até **200%** de uso de CPU, tanto na `c5.large` (linha verde) quanto na `t2.medium` (linha amarela). Esse valor indica que o LXC conseguiu utilizar dois vCPUs de forma simultânea. Em sistemas com múltiplos núcleos, é comum a porcentagem de uso da CPU somar o consumo de cada núcleo — por isso, 200% representa a utilização de dois núcleos. Isso mostra que o LXC escalou melhor, aproveitando toda a capacidade da instância.

5.2 Análise do Desempenho de E/S de Disco

Após a análise de CPU, esta seção avalia o desempenho de Entrada/Saída (E/S) de disco, medido em Operações de Entrada/Saída por Segundo (IOPS). O objetivo é comparar o comportamento do Docker e do LXC sob cargas de trabalho intensivas de disco, em diferentes tipos de instância.

Figura 11 – Comparação de Desempenho de Disco - Docker vs LXC por Tipo de Instância AWS.



Fonte: Elaborado pelo Autor

Resultados do Desempenho de Disco por Tipo de Instância

- **Instância** `c5.large`**:**
  - **Docker:** Apresentou média de **1600 IOPS**, com boa consistência.
  - **LXC:** Obteve média de **2200 IOPS**, superando o Docker, com desvio padrão maior, mas mantendo boa estabilidade.

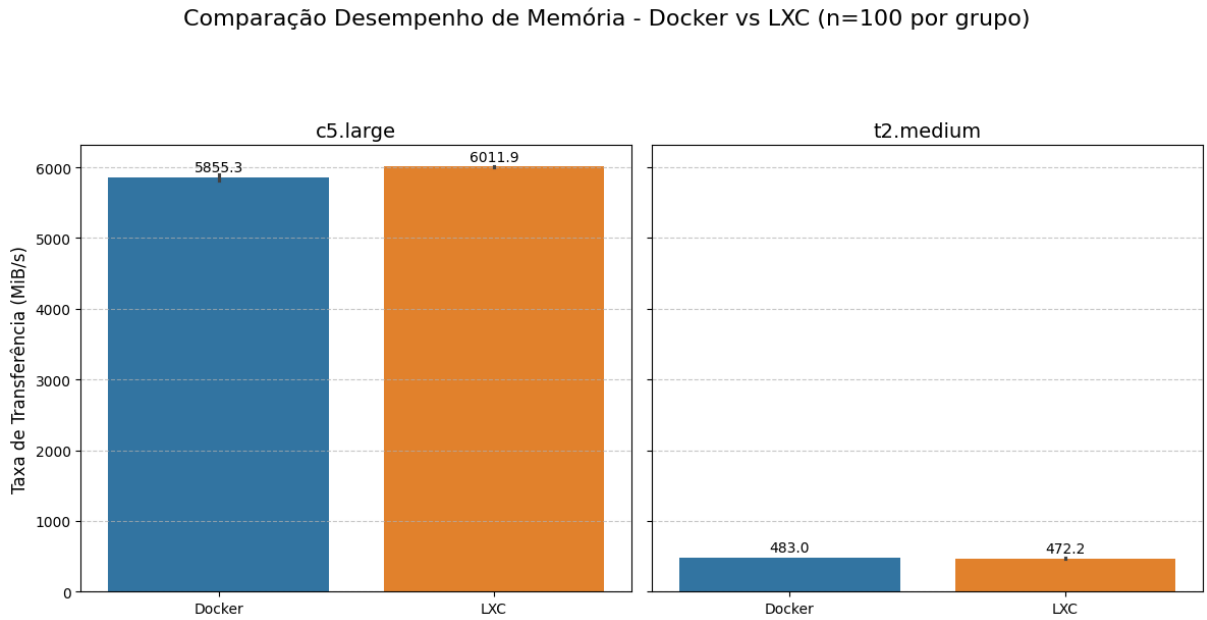
  Aqui, o LXC mostra desempenho superior em E/S de disco, mesmo em uma instância voltada para computação com recursos equilibrados.
- **Instância** `t2.medium`**:**
  - **Docker:** Média de **1250 IOPS**.
  - **LXC:** Média de **1200 IOPS**, praticamente equivalente.

  O desempenho de disco nas `t2.medium` é consideravelmente inferior em comparação às `c5.large`, e as diferenças entre Docker e LXC são insignificantes. Isso sugere que o principal limitador está nas características da própria instância — como a menor capacidade de I/O — e não nas tecnologias de contêiner em si.

5.3 Análise do Desempenho de Memória

Nesta seção, é analisado o desempenho de memória, medido em taxa de transferência (MiB/s). Assim como nas métricas anteriores, foram coletadas 100 amostras por configuração.

Figura 12 – Comparação de Desempenho de Memória - Docker vs LXC por Tipo de Instância AWS.



Fonte: Elaborado pelo Autor

Resultados do Desempenho de Memória por Tipo de Instância

- **Instância** `c5.large`**:**
  - **Docker:** Média de **5855,3 MiB/s**, com alta consistência.
  - **LXC:** Média ligeiramente superior, de **6011,9 MiB/s**.

  Ambas as tecnologias demonstram excelente desempenho de memória. A diferença entre elas é pequena, indicando que ambas são eficientes nesse aspecto.
- **Instância** `t2.medium`**:**
  - **Docker:** Média de **483,0 MiB/s**.
  - **LXC:** Média de **472,2 MiB/s**.

  O desempenho aqui é substancialmente menor que nas `c5.large`. A diferença entre as tecnologias é praticamente irrelevante, e os resultados refletem as limitações da instância.

5.4 Teste de Hipotese

A Tabela 1 mostra o resultado do teste T de Studant realizado, informando se houve ou não diferença estatiscamente relevante nos resultados obtidos.

Tabela 1 – Resultados estatísticos dos testes de desempenho

Testes	Diferença	Valor P	Diferença Significativa
CPU_T2.Medium	194,57	0,0002	Sim
CPU_C5.Large	-17,72	0,0000	Sim
MEM_T2.Medium	10,46	0,0091	Sim
MEM_C5.Large	-155,79	0,0000	Sim
DISK_T2.Medium	54,69	0,0768	Não
DISK_C5.Large	-611,35	0,0000	Sim

Fonte: Elaborado pelo Autor

5.5 Discussão dos Resultados

- **Eficiência de CPU e Alocação de Recursos:** A análise de desempenho da CPU revelou um aspecto notável do Docker: sua alta eficiência por núcleo. Apesar de o monitoramento no Grafana indicar que os *containers* Docker utilizavam cerca de 80% de um único vCPU (sem ultrapassar os 100%), o desempenho obtido em "eventos por segundo"foi altamente competitivo — e, na instância `t2.medium`, inclusive superior ao do LXC. Em contrapartida, os *containers* LXC conseguiram utilizar os dois vCPUs disponíveis, atingindo até 200% de uso de CPU. Isso sugere que o Docker, por meio de otimizações internas e do seu modelo de isolamento, é capaz de extrair mais desempenho por núcleo, mesmo que o uso simultâneo de múltiplos núcleos pareça limitado por padrão.
- **Desempenho de E/S (Disco e Memória): Predominância da Infraestrutura:** Para cargas de trabalho intensivas em E/S — tanto de disco quanto de memória —, a pesquisa demonstrou que o fator mais determinante para o desempenho é a capacidade da própria instância da AWS.
  - Nas instâncias `t2.medium`, que possuem uma base de desempenho limitada para E/S, tanto Docker quanto LXC apresentaram resultados significativamente mais baixos. As diferenças entre as tecnologias foram mínimas — e, no caso do disco, estatisticamente não significativas (p=0,0768). Isso indica que, em ambientes mais restritos, a escolha da tecnologia de contêiner tem impacto limitado.
  - Já nas instâncias `c5.large`, com maior capacidade de E/S, ambas as tecnologias alcançaram desempenhos superiores. Nesse contexto, o LXC apresentou vantagem estatisticamente significativa em operações de disco (diferença de -611,35 IOPS, p=0,0000) e memória (diferença de -155,79 MiB/s, p=0,0000), o

que pode indicar uma sobrecarga menor ou maior de eficiência em aproveitar recursos de alto desempenho.

– **Impacto da Variabilidade:**

A análise da variabilidade, especialmente nos testes de CPU com instância `t2.medium`, evidenciou o comportamento "*burstable*"desse tipo de instância. O Docker conseguiu atingir picos impressionantes de desempenho, mas com um desvio padrão elevado (407,34), o que pode ser prejudicial em aplicações que exigem baixa latência e resultados mais previsíveis. Já nas instâncias `c5.large`, os resultados foram mais estáveis para ambas as tecnologias.

6 CONCLUSÕES

Este trabalho teve como objetivo principal comparar o desempenho e o comportamento de alocação de recursos entre duas tecnologias de contêineres leves — Docker e LXC — em diferentes tipos de instâncias da nuvem AWS (`t2.medium` e `c5.large`). Os testes envolveram medições de CPU, E/S de disco e E/S de memória, com coleta sistemática de 100 amostras por configuração, e análise estatística com teste T de Student.

Com base nos resultados obtidos, não é possível afirmar categoricamente que Docker ou LXC é superior de forma geral. O desempenho variou de acordo com o tipo de recurso analisado e o tipo de instância utilizada. A Tabela 1 evidencia que, embora a maioria das comparações tenha revelado diferenças estatisticamente significativas, essas diferenças não foram unilaterais nem consistentes em favor de uma única tecnologia.

Portanto, a escolha entre Docker e LXC deve considerar o perfil específico da carga de trabalho, o tipo de recurso crítico (CPU, memória, disco) e as características da infraestrutura subjacente. O Docker demonstrou maior eficiência por núcleo em situações limitadas a um vCPU, enquanto o LXC teve melhor aproveitamento de múltiplos núcleos e desempenho superior em E/S em ambientes mais robustos.

Em síntese, os dados indicam que **nenhuma das tecnologias é universalmente melhor que a outra**. Cada uma possui vantagens específicas dependendo do contexto de execução, e decisões informadas devem considerar não apenas *benchmarks* isolados, mas também os requisitos operacionais e limitações de infraestrutura do ambiente real de aplicação.

Além do valor técnico, este trabalho também tem relevância prática para o contexto educacional. Ao comparar o desempenho de Docker e LXC com foco na economia de recursos computacionais, busca-se fornecer subsídios para que professores possam escolher, de forma mais embasada, a tecnologia mais adequada ao utilizar contêineres em sala de aula. Em ambientes educacionais com infraestrutura limitada, essa decisão pode impactar diretamente na eficiência e na viabilidade de práticas laboratoriais. Assim, os resultados apresentados contribuem não apenas para o avanço do conhecimento técnico, mas também para a otimização do uso de recursos no ensino de computação em nuvem e tecnologias de virtualização leve.

REFERÊNCIAS

Amazon Web Services. **Instâncias de desempenho com capacidade de expansão**. 2025. Disponível em: https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGuide/burstable-performance-instances.html. Acesso em: 22 jul. 2025.

BHATTACHARYA, M.; MITTAL, H. Exploring the performance of container runtimes within kubernetes clusters. **International Journal of Computing**, v. 22, p. 509–514, 12 2023.

DJORDJEVIC, B.; KRALJEVIC, N.; JANJIC, K.; RISTIC, N. Testing the performance of ssd disk technology on aws architecture. In: INTERNATIONAL SCIENTIFIC CONFERENCE – SCIENCE AND HIGHER EDUCATION IN FUNCTION OF SUSTAINABLE DEVELOPMENT (SED 2025), 14., 2025, Belgrade, Serbia. **Proceedings[...]**. Belgrade, Serbia: Academy of Technical and Art Applied Studies Belgrade, 2025. p. 1–9.

GEDIA, D.; PERIGO, L. Performance evaluation of sdn-vnf in virtual machine and container. In: CONFERENCE ON NETWORK FUNCTION VIRTUALIZATION AND SOFTWARE DEFINED NETWORKS (NFV-SDN), 2018., 2018, Verona, Italy. **Proceedings[...]**. Verona, Italy: IEEE, 2018. p. 1–7.

JUBURY, M. A. **Measure the data transfer performance between containers**. Dissertação (Mestrado) – Aalto University, 2022.

KANE, S. P.; MATTHIAS, K. **Docker**: Up running: Shipping reliable containers in production. 1st. ed. [*S. l.*]: O'Reilly Media, 2015. 200 p. 5. ISBN 978-1-491-91757-2.

MAVRIDIS, I.; KARATZA, H. Orchestrated sandboxed containers, unikernels, and virtual machines for isolation-enhanced multitenant workloads and serverless computing in cloud. **Concurrency and Computation: Practice and Experience**, Wiley Online Library, v. 33, n. 19, p. e6365, 2021. Disponível em: https://doi.org/10.1002/cpe.6365. Acesso em: 10 jul. 2024.

POTDAR, A. M.; D G, N.; KENGOND, S.; MULLA, M. M. Performance evaluation of docker container and virtual machine. **Procedia Computer Science**, v. 171, p. 1419–1428, 2020. ISSN 1877-0509. Third International Conference on Computing and Network Communications (CoCoNet'19). Disponível em: https://www.sciencedirect.com/science/article/pii/S1877050920311315. Acesso em: 20 fev. 2024.

RUAN, B.; HUANG, H.; WU, S.; JIN, H. A performance study of containers in cloud environment. In: INTERNATIONAL CONFERENCE ON SERVICE COMPUTING (SCC 2016), 10., 2016, Zhangjiajie, China. **Proceedings[...]**. Zhangjiajie, China: Springer International Publishing, 2016. p. 343–356. ISBN 978-3-319-49178-3.

SALAH, T.; ZEMERLY, M. J.; YEUN, C. Y.; AL-QUTAYRI, M.; AL-HAMMADI, Y. Performance comparison between container-based and vm-based services. In: CONFERENCE ON INNOVATIONS IN CLOUDS, INTERNET AND NETWORKS (ICIN), 20., 2017, Paris, France. **Proceedings [...]**. Paris, France: IEEE, 2017. p. 185–190.

SILVA, D.; RAFAEL, J.; FONTE, A. Toward optimal virtualization: An updated comparative analysis of docker and lxd container technologies. **Computers**, v. 13, n. 4, 2024. ISSN 2073-431X. Disponível em: https://www.mdpi.com/2073-431X/13/4/94. Acesso em: 20 maio 2024.

VIKTORSSON, W.; KLEIN, C.; TORDSSON, J. Security-performance trade-offs of kubernetes container runtimes. In: INTERNATIONAL SYMPOSIUM ON MODELING, ANALYSIS, AND SIMULATION OF COMPUTER AND TELECOMMUNICATION SYSTEMS (MASCOTS), 28., 2020, Nice, France. **Proceedings [...]**. Nice, France: IEEE, 2020. p. 1–4.

ANEXOS

Figura 13 – Script de Benchmarks Docker.

```bash
#!/bin/bash

# Configurações
OUTPUT_DIR="/home/ubuntu/benchmark_results"
mkdir -p "$OUTPUT_DIR"
CSV_FILE="$OUTPUT_DIR/docker_benchmark_$(date +%Y%m%d_%H%M%S).csv"

# Cabeçalho do CSV
echo "Teste,Container,Parametros,Resultado,Timestamp" > "$CSV_FILE"

# Menu de escolha
echo "Escolha o teste:"
echo "1) CPU (sysbench)"
echo "2) Disco (fio)"
echo "3) Memória (sysbench)"
echo "4) Todos"
read -p "Opção (1-4): " TEST_CHOICE

# Parâmetros
read -p "Número de containers: " NUM_CONTAINERS
read -p "Duração (segundos): " TEST_DURATION

# Funções de benchmark
benchmark_cpu() {
    local container_id=$1
    echo "Running CPU test in $container_id..."
    docker exec "$container_id" sysbench cpu --cpu-max-prime=20000 --threads=4 --time=$TEST_DURATION run | \
    grep "events per second:" | awk '{print $4}' | \
    xargs -I {} echo "CPU,$container_id,prime20000 {} events/s,$(date +%Y-%m-%d_%H:%M:%S)" >> "$CSV_FILE"
}

benchmark_disk() {
    local container_id=$1
    echo "Running Disk test in $container_id..."
    docker exec "$container_id" fio --name=disk_test --rw=randrw --bs=4k --size=100M --runtime=$TEST_DURATION --time_based --output-format=json | \
    jq '.jobs[0].read.iops_mean, .jobs[0].write.iops_mean' | \
    xargs printf "DISK,$container_id,randrw %.0f/%.0f IOPS,$(date +%Y-%m-%d_%H:%M:%S)\n" >> "$CSV_FILE"
}

benchmark_mem() {
    local container_id=$1
    echo "Running Memory test in $container_id..."
    docker exec "$container_id" sysbench memory --memory-block-size=1K --memory-total-size=10G --time=$TEST_DURATION run | \
    grep "MiB/sec" | awk '{print $4}' | \
    xargs -I {} echo "MEM,$container_id,{} MiB/s,$(date +%Y-%m-%d_%H:%M:%S)" >> "$CSV_FILE"
}

# Execução principal
for i in $(seq 1 "$NUM_CONTAINERS"); do
    CONTAINER_ID="benchmark-container-$i"
    docker run -d --name "$CONTAINER_ID" ubuntu:22.04 tail -f /dev/null
    docker exec "$CONTAINER_ID" apt-get update > /dev/null
    docker exec "$CONTAINER_ID" apt-get install -y sysbench fio jq > /dev/null

    case "$TEST_CHOICE" in
        1) benchmark_cpu "$CONTAINER_ID" ;;
        2) benchmark_disk "$CONTAINER_ID" ;;
        3) benchmark_mem "$CONTAINER_ID" ;;
        4)
            benchmark_cpu "$CONTAINER_ID"
            benchmark_disk "$CONTAINER_ID"
            benchmark_mem "$CONTAINER_ID"
        ;;
    esac

    docker rm -f "$CONTAINER_ID" > /dev/null
done

echo "Benchmark concluído! Arquivo CSV gerado em:"
echo "$CSV_FILE"
```

Fonte: Elaborado pelo Autor

Figura 14 – Script de Benchmarks LXC.

```
#!/bin/bash

# Configurações
OUTPUT_DIR="/home/ubuntu/benchmark_results"
mkdir -p "$OUTPUT_DIR"
CSV_FILE="$OUTPUT_DIR/lxc_benchmark_$(date +%Y%m%d_%H%M%S).csv"

# Cabeçalho do CSV
echo "Teste,Container,Parametros,Resultado,Timestamp" > "$CSV_FILE"

# Menu de escolha
echo "Escolha o teste:"
echo "1) CPU (sysbench)"
echo "2) Disco (fio)"
echo "3) Memória (sysbench)"
echo "4) Todos"
read -p "Opção (1-4): " TEST_CHOICE

# Parâmetros
read -p "Número de containers: " NUM_CONTAINERS
read -p "Duração (segundos): " TEST_DURATION

# Funções de benchmark
benchmark_cpu() {
    local container=$1
    echo "Running CPU test in $container..."
    lxc exec "$container" -- sysbench cpu --cpu-max-prime=20000 --threads=4 --time=$TEST_DURATION run | \
    grep "events per second:" | awk '{print $4}' | \
    xargs -I {} echo "CPU,$container,prime20000 {} events/s,$(date +%Y-%m-%d_%H:%M:%S)" >> "$CSV_FILE"
}

benchmark_disk() {
    local container=$1
    echo "Running Disk test in $container..."
    lxc exec "$container" -- fio --name=disk_test --rw=randrw --bs=4k --size=100M --runtime=$TEST_DURATION --time_based --output-format=json | \
    jq '.jobs[0].read.iops_mean, .jobs[0].write.iops_mean' | \
    xargs printf "DISK,$container,randrw %.0f/%.0f IOPS,$(date +%Y-%m-%d_%H:%M:%S)\n" >> "$CSV_FILE"
}

benchmark_mem() {
    local container=$1
    echo "Running Memory test in $container..."
    lxc exec "$container" -- sysbench memory --memory-block-size=1K --memory-total-size=10G --time=$TEST_DURATION run | \
    grep "MiB/sec" | awk '{print $4}' | \
    xargs -I {} echo "MEM,$container,{} MiB/s,$(date +%Y-%m-%d_%H:%M:%S)" >> "$CSV_FILE"
}

# Execução principal
for i in $(seq 1 "$NUM_CONTAINERS"); do
    CONTAINER_NAME="benchmark-lxc-$i"
    lxc launch ubuntu:22.04 "$CONTAINER_NAME"
    sleep 10  # Espera inicialização
    lxc exec "$CONTAINER_NAME" -- apt-get update > /dev/null
    lxc exec "$CONTAINER_NAME" -- apt-get install -y sysbench fio jq > /dev/null

    case "$TEST_CHOICE" in
        1) benchmark_cpu "$CONTAINER_NAME" ;;
        2) benchmark_disk "$CONTAINER_NAME" ;;
        3) benchmark_mem "$CONTAINER_NAME" ;;
        4)
            benchmark_cpu "$CONTAINER_NAME"
            benchmark_disk "$CONTAINER_NAME"
            benchmark_mem "$CONTAINER_NAME"
        ;;
    esac
    lxc delete --force "$CONTAINER_NAME" > /dev/null
done

echo "Benchmark concluído! Arquivo CSV gerado em:"
echo "$CSV_FILE"
```

Fonte: Elaborado pelo Autor