Instalação, Configuração, Utilização e Análise de Desempenho de Contentores Docker em ambientes Local e Cloud

João Claudio Paco e Ndukula Selo Afonso

Todos Assistentes de Investigação em Engenharia Informática da Universidade Kimpa-Vita em Angola

***Abstract*—Este trabalho explora a instalação, configuração, utilização e análise de desempenho de contentores Docker em ambientes local e *cloud*. Com o crescimento do uso de *containers* no desenvolvimento e *deploy* de aplicações, é crucial entender as diferenças de desempenho e escalabilidade entre os dois ambientes. Utilizando uma aplicação como exemplo, em python (geração de números primos menor a 1000), comparamos a performance local usando o terminal vs code e o Docker desktop, e a performance ao nível do *Azure* (Azure CLI, ACR, ACI). Nossos resultados destacam as vantagens e desvantagens de cada abordagem, fornecendo *insights* valiosos para desenvolvedores e arquitetos de sistemas.**

Palavras-chave— cloud, ambientes, performance, local, *desempenh**o.***

I. INTRODUCTION

Nos últimos anos, os *containers* Docker têm se tornando uma tecnologia essencial no desenvolvimento de *software*, permitindo que aplicações sejam empacotadas com todas as suas dependências em um único pacote executável. Esta abordagem garante consistência entre os ambientes de desenvolvimento, teste e produção, simplificando o processo de *deploy* e reduzindo o tempo de configuração.

A tecnologia de *containers* oferece inúmeros benefícios, como portabilidade, isolamento e eficiência de recursos. No entanto, a decisão de onde executar esses *containers*- em um ambiente local ou na *cloud*- pode impactar significativamente o desempenho, a escalabilidade e a gestão da aplicação. Este trabalho busca explorar essas diferenças, utilizando uma aplicação simples como estudo de caso para comparar a instalação, configuração e desempenho de *containers* Docker em ambos os ambientes.

A escolha entre executar *containers* localmente ou na *cloud* é uma decisão crítica que pode afetar não apenas o desempenho, mas também a escalabilidade, a segurança e os custos operacionais de uma aplicação. Entender essas diferenças, é essencial para desenvolvedores, engenheiros e arquitetos de sistemas, que buscam otimizar suas infraestruturas e garantir uma experiência de usuário consistente e eficiente.

O Docker simplifica o processo de criação, *deploy* e execução de aplicações em *containers*, oferecendo uma plataforma leve e eficiente para o desenvolvimento. *Em um ambiente local, os desenvolvedores têm controle total sobre os recursos e a configuração do sistema,* o que pode resultar em menor latência e maior eficiência para algumas aplicações. Por outro lado, *o cloud computing* oferce vantagens significativas em termo de escalabilidade e gerenciamento de recursos. Plataformas como *Azure Container Instances* (ACI) permite que os *containers* sejam executados em uma infraestrutura altamente disponível e escalável, com gestão simplificada e integração com outros serviços de *cloud*.

Neste trabalho, utilizamos uma aplicação simples escrita em python, com operações de execução das ferramentas *benchmarks* tais como *geekbench* e *sysbench*, para realizar testes de desempenho e comparar os resultados entre os ambientes local e *cloud*. Nossa análise inclui métricas de tempo de resposta, uso de CPU e memória, e escalabilidade, fornecendo uma visão clara das vantagens e desvantagens de cada abordagem.

II. APLICAÇÃO EXEMPLO, ARQUITETURA, ALGORITMOS, MÉTODOS, PROTOCOLOS, RESULTADOS ANALÍTICOS E EXEMPLO ILUSTRADO

A. *Aplicação Exemplo*

A aplicação exemplo é um *script Python* desenvolvido no Visual Studio Code (VS CODE) que executa ferramentas de *benchmark* como *Geekbench* para avaliar o desempenho da CPU e *Sysbench* para avaliar a performance da memória. Esta aplicação será containerizada usando Docker e implementada em ambientes local e *cloud* (*Azure Container Instances* - *ACI*). O objetivo é comparar o desempenho da aplicação nesses diferentes ambientes.

B. *Arquitetura*

A arquitetura da aplicação é simples, composta por:

1. **Container Docker:** Executando *o script Python* e as ferramentas de benchmark.
2. **Ambientes de Execução:**
   - **Local:** Um laptop ou desktop com Docker instalado.
   - **Cloud:** Azure container Instances (ACI), para execução na nuvem.

A figura abaixo ilustra a Arquitetura da aplicação:

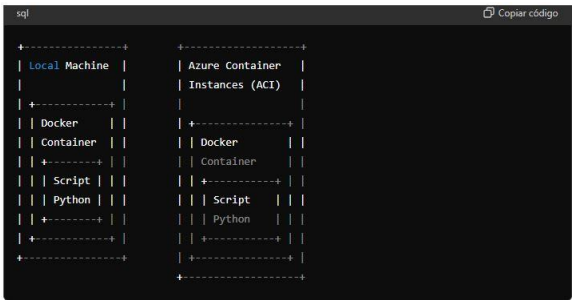


Figura 1: Arquitetura do sistema implementado

C. *Algoritmos e Métodos*

1. **Script Python:**
   - O *script Python* executa comandos de *benchmark* utilizando bibliotecas como subprocess paara chamar *Geekbench e Sysbench.*
   - Captura e armazena os resultados dos benchmarks para análise posterior.

2. **Dockerfile:**
   - Define a imagem Docker que inclui todas as dependências necessárias para executar o *script e os benchmarks*.
   - Dockerfile instala *Python*, *Geekbench* e copia o script *Python* para o container.

3. **Execução do script:**
   - **Local:**
     - *Docker build*
     - *Docker run*
   - **Cloud:**
     - Configuração do *Azure Container Registry* (ACR) e upload da imagem Docker.
     - Criação de uma instância do *Azure Container Instances* (ACI) para executar a imagem Docker.

D. *Protocolos*

Os *benchmarks* utilizam ferramentas padrão da indústria:

- *Geekbench:* avalia o desempenho da CPU, executando testes de processamento intensivo.
- *Sysbench:* avalia o desempenho da memória, executando testes de leitura e escrita.

III. CONFIGURAÇÃO EXPERIMENTAL

A. *Arquitetura do Sistema Implementado*

Para comparar o desempenho de contentores Docker em ambientes local e *cloud*, implementamos a arquitetura ao lado ilustrada.

B. *Detalhes de Instalação*

B.1. Ambiente local:

1. Download e instalação do Dokcer desktop
2. Construção da imagem Docker usando o IDE do Ms vs code:

2.1 Criar um *Dockerfile* com o seguinte conteúdo:

✓ *Script para Geekbench:*

```
FROM ubuntu:latest

# Instalar dependências
RUN apt-get update && apt-get install -y \
    wget \
    python3 \
    python3-pip \
    unzip

# Baixar e instalar o Geekbench
RUN wget https://cdn.geekbench.com/Geekbench-5.4.1-Linux.tar.gz && \
    tar -xzvf Geekbench-5.4.1-Linux.tar.gz && \
    mv Geekbench-5.4.1-Linux /opt/geekbench

# Copiar o aplicativo Python para o contêiner
COPY app2.py /app/app2.py
WORKDIR /app

# Comando padrão para rodar o Geekbench
CMD ["/opt/geekbench/geekbench5"]
```

✓ *Script para Sysbench*

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y sysbench python3

COPY . /app
WORKDIR /app

CMD ["sh", "-c", "python3 app2.py & sysbench --test=memory run"]
```

2.2 *criar o script para a aplicação:*

```
import timeit

def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

def generate_primes(limit):
    primes = []
    for num in range(2, limit + 1):
        if is_prime(num):
            primes.append(num)
    return primes

def benchmark():
    setup_code = """
from __main__ import generate_primes
"""
    stmt = "generate_primes(1000)"
    times = timeit.repeat(stmt, setup=setup_code, repeat=3,
number=100)
    print(f"Tempo de execução: {min(times)}")

if __name__ == "__main__":
    generator = generate_primes(1000)
    print(generator)
    benchmark()
```

2.3 Construir a imagem Docker:

✓ *Teste cpu pelo Gekkbench:*

- *docker build –taag geekbench-python-benchmark .*

✓ ***Teste memória pelo sysbench***

- *docker build –tag sysbench-python-bechmark .*

2.4 Executar o contentor

✓ ***Teste cpu peo Geekbench***

- *docker run -it geekbemch-python-bechmark*

✓ ***Teste memóroa pelo Sysbench***

- *docker run -it sysbench-python-benchmark*

B.2. Ambiente cloud:

1. Instalação do Azure CLI e criação do ACR:

1.1 Download e instalar o Azure CLI
1.2 Login no Azure:

- *az login*

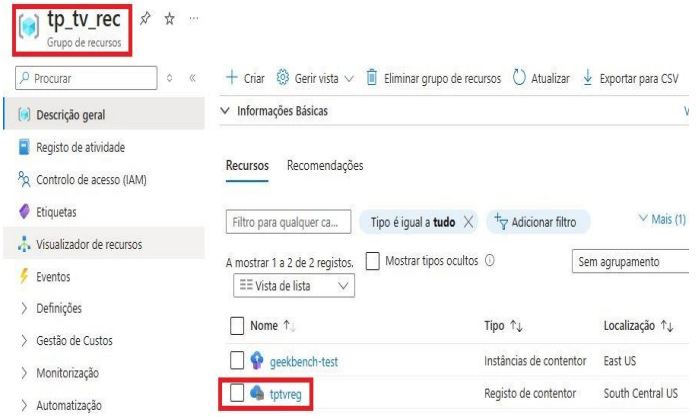
1.3 Criar um grupo de recurso:

- *az group create –name tp_tv_rec –location eastus*

1.4 Criar uma instância do ACR:

- *az acr create –resource tp_tv_rec –name tptvreg Standard –location southcentralus*

1.5 Resultado no cloud Azure:



2. Enviar a imagem Docker para ACR:

2.1 Fazer login no ACR:

- *docker login tptvreg.azurecr.io*

2.2 Taggear a imagem Docker

✓ ***Do Geekbench :***

- *docker tag geekbench-python-benchmark tptvreg.azure.io/geekbemch-python-bechmark:latest*

✓ ***Do Sysbench***

- *docker tag sysbench-python-benchmark tptvreg.azure.io/sysbemch-python-bechmark:latest*

2.3 Enviar a imagem para ACR:

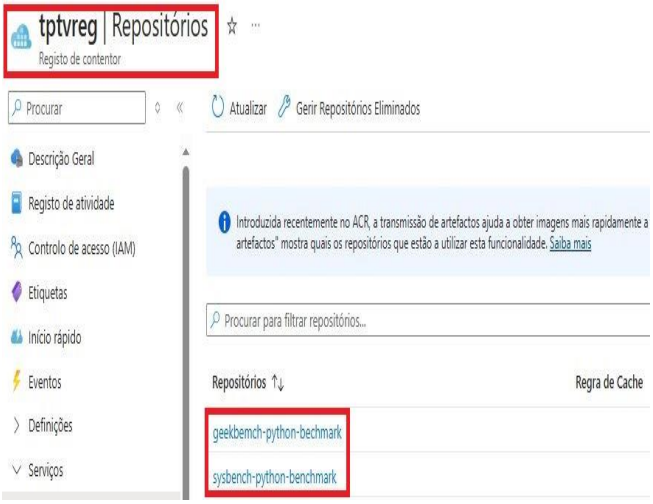
✓ ***Do Geekbench:***

➢ *docker push tptvreg.azurecr.io/geekbench-python-benchmark:latest*

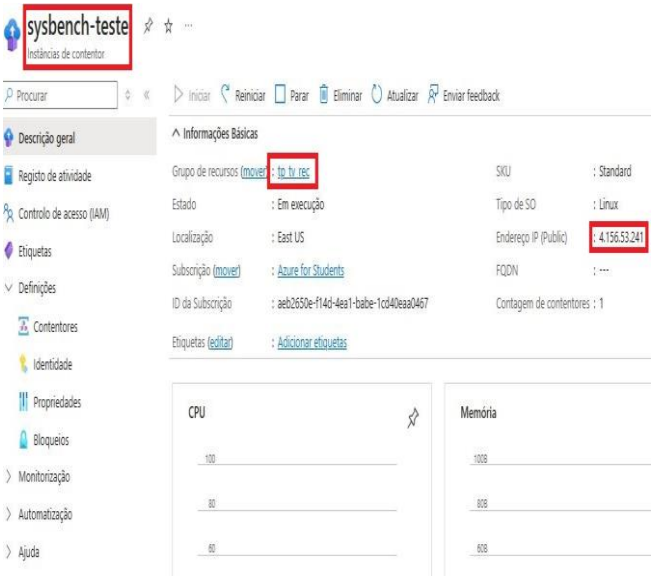
✓ ***Do Sysbench:***

➢ *docker push tptvreg.azurecr.io/sysbench-python-benchmark:latest*

✓ **Resultado no *cloud Azure***:

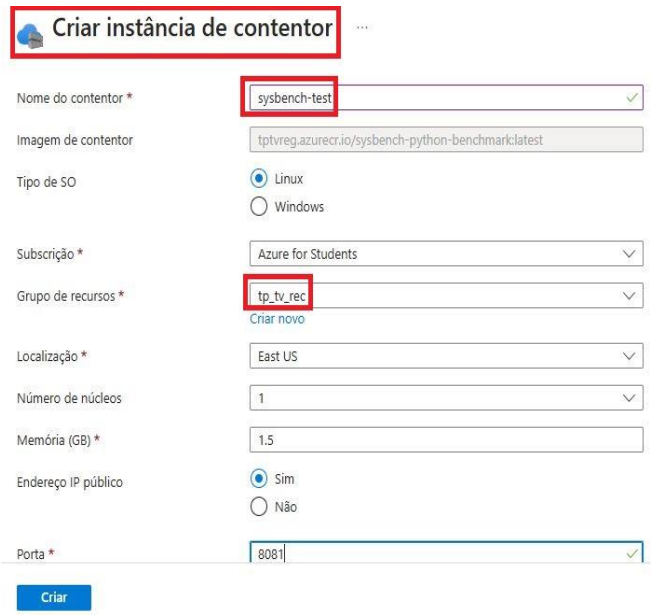


b) Instância criada



2.4 Criar a instância do ACI

a) Configuração:



IV. RESULTADO EXPERIMENTAL

A. *Benchmarks e conjunto de dados utilizado*

1. *Ferramentas benchmark*

- ✓ *Geekbench:* Ferramenta de benchmarking que mede o desempenho da CPU.
- ✓ *Sysbench:* Ferramenta de benchmarking que mede o desempenho da memória.

2. *Conjunto de dados*

Para os *benchmarks*, utilizamos os testes padrão das ferramentas *Geekbench* e *Sysbench*.

Estes testes incluem:

- ✓ *Geekbench*: Avaliação da performance da CPU através de uma série de testes computacionais,
- ✓ *Sysbench:* Avaliação da performance da memória através de testes de leitura e escrita.

3. *Execução das imagens Docker:*

3.1 No ambiente local:

- ✓ *Commandos:*

- ➢ *docker run -it geekbench-python-benchmark*
- ➢ *docker run -it sysbech-python-benchmark*

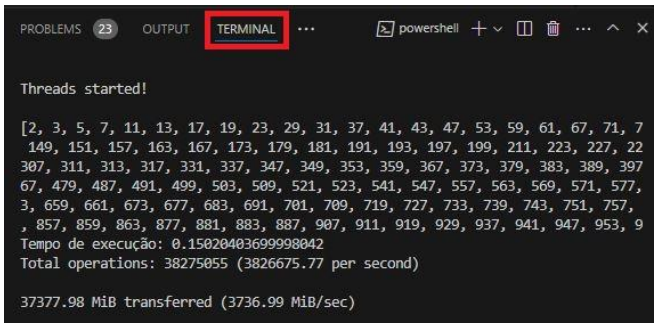
- ✓ *Resultados:*

- o *Geekbench*

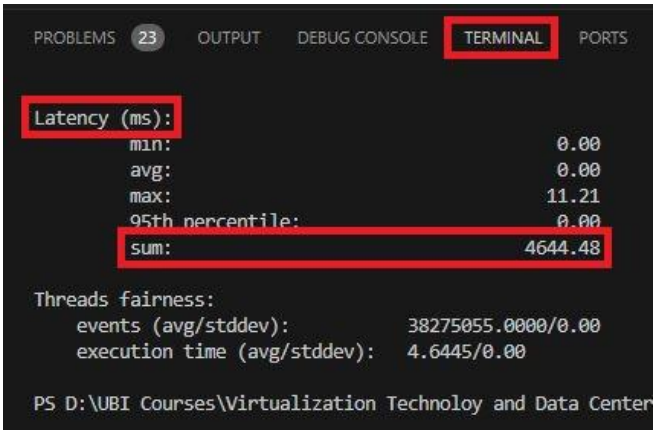
  - *Resultado em single e multi core*

- o *Sysbench*

  - *Execução do script python: Gereção dos número primos*



  - *Exibição do teste da latência por sysbench*

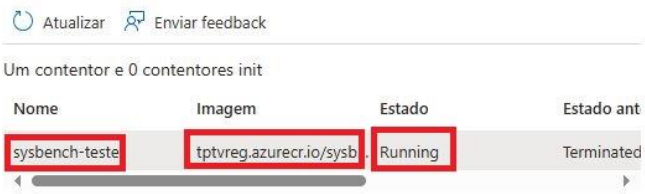


3.2 No ambiente cloud Azure:

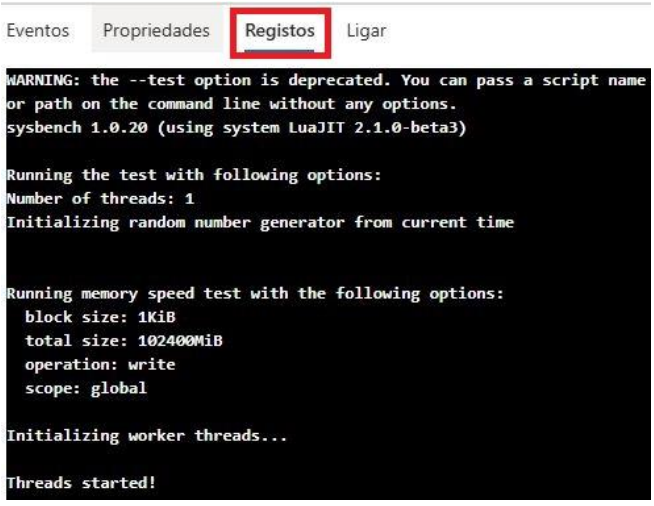
- o *Geekbench*

- o *Sysbench*

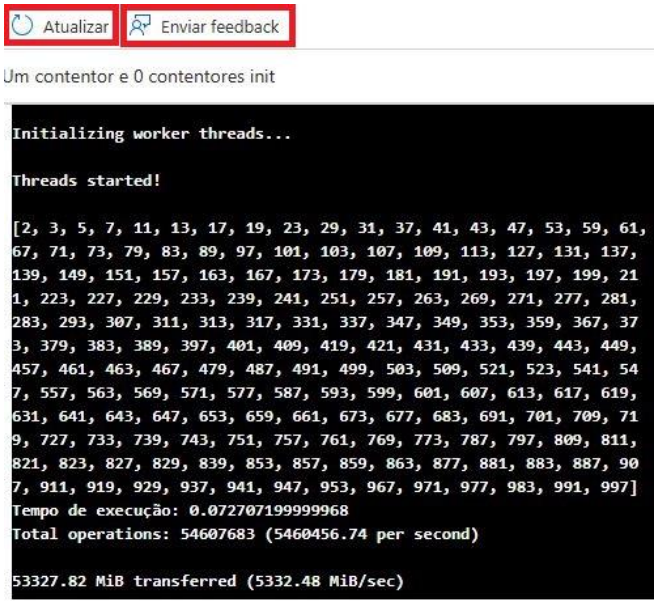
- ✓ *Container em curso de execução*



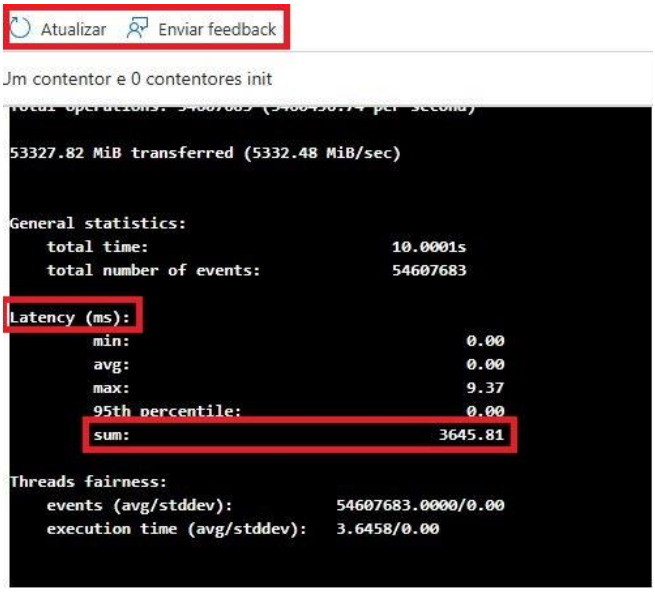
- ✓ *Container em curso de execução*



✓ *Execução de script python: geração números primos*



✓ *Exibição do teste da latência por sysbench*



3.3 Analise comparative de desempenho : Container local vs cloud

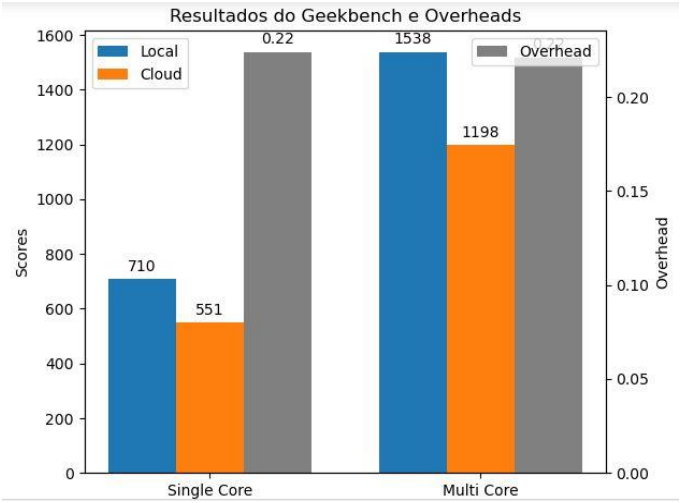
a) Teste CPU (Geekbench)

✓ *Teste CPU e cálculo overhead*

Local		**Cloud**		**Overhead**	
Single core	*Multi core*	*Ingle core*	*Multi core*	*Single core*	*Multi core*
710	*1538*	*551*	*1198*	0,22394	0,22106

Tabela 1: Teste CPU e overhread

✓ Gráfico teste CPU e overhead



✓ *Interpretação:*

Overhead do Single Core:

- Este valor de 0.223943661971831 indica que o desempenho de um único núcleo ***dentro do contentor em cloud é aproximadamente 22.39% inferior ao desempenho do contentor em nível local***.

Overhead do Multi Core:

- Este valor de 0.2210663198959688 indica que o desempenho de múltiplos núcleos ***dentro do contentor em cloud é aproximadamente 22.11% inferior ao desempenho do contentor ao nível local***.

Isso reflete a diminuição do desempenho devido à sobrecarga introduzida pela execução em um ambiente *cloud.*

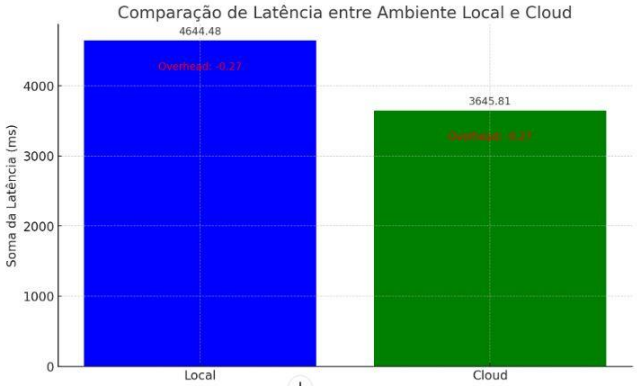
b) Teste memória (Sysbench)

✓ Teste latência e cálculo do overhread

Latency teste (ms)			
Teste	**local**	**Cloud**	**Overhead**
min	0.00	0.00	
avg	0.00	0.00	
max	11.21	9.37	
sum	**4644.48**	**3645.81**	**-0,2739**

Tabela 2: Teste latência e cálculo do overhread

✓ Gráfico do resultado do teste realizado



✓ Interpretação:

O gráfico acima mostra a comparação da soma da latência entre o ambiente local e o ambiente cloud

- No ambiente local, a soma da latência é de 4644,48 ms.
- No ambiente clpous, a soma da latÊncia é de 3645.81 ms

O overhead calculado é de aproximadaamente -27.39%, indicando que a latÊncia no ambiente *cloud* é significativamente menor do que no ambiente local

CONCLUSÕES E REFERENCIAS

Em ambientes locais, o Docker proporciona um ambiente isolado que replica fielmente a produção, facilitando o desenvolvimento e a resolução de problemas. Já em ambientes cloud. O Docker maximiza a utilização dos recursos. Melhora a escalabilidade e facilita a gestão de grandes volumes de trabalho.

Ao analisar o desempenho dos contentores Docker, observaou-se que. Embora haja uma pequena sobrecarga de recursos em comparação com a execução direta no hardware, os benefícios em termos de portabilidade, consistência e eficiência operacional superam significativamente essas desevantagens. Alem disso, as praticas recomendadas de otimização de imagens Docker e a utilização de ferramentas de orquestração, como ACR, contribuem para mitigar os impactos no desempenho.

Referencias

1. Dua, R., Raja, A. R., & Kahadin, D. (2014). Virtualization vs Containerization to Support Paas, 2014 IEEE, INternational Conference on Cloud Engeneering
2. Zhang, X, & Ravichandran, R. (2017). Docker Container vs Virtual Machines: A performance Comparison, 2017 IEEE, International Conference on cloud Computing Technology and Science (CloudCom), 342-350. DOL: 10.1109/CloudCom.2017.53