Projeto Primos-Thread: Integração de Threads, Docker e Kubernetes

Gabriel Vitor Silva
mygabriel0@gmail.com

22 de fevereiro de 2025

Resumo

Este artigo apresenta o projeto **Primos-Thread**, desenvolvido como trabalho de aproveitamento de estudo para a disciplina de *Laboratório de Sistemas Operacionais*. A solução demonstra a integração de **Threads** em Python, **Docker** e **Kubernetes** para resolver um problema real de processamento paralelo, com ênfase na verificação de números primos em intervalos aleatórios. Adicionalmente, são abordadas as dificuldades encontradas, melhorias implementadas, e a motivação para o uso das tecnologias Docker e Kubernetes, bem como o aprendizado adquirido através do curso da Udemy.

Sumário

1 Introdução

O objetivo deste projeto é demonstrar a aplicação prática de conceitos fundamentais em sistemas operacionais, utilizando:

- **Threads:** para executar cálculos de forma paralela;
- **Docker:** para empacotar e distribuir a aplicação de forma portátil;
- **Kubernetes:** para orquestrar e escalar a aplicação.

A aplicação consiste em um script Python que verifica a primalidade de números em intervalos aleatórios usando múltiplas threads. A imagem gerada foi publicada no Docker Hub, permitindo que qualquer usuário execute o projeto em diferentes ambientes, seja localmente ou em um cluster Kubernetes.

2 Tecnologias Utilizadas

2.1 Python & Threads

A linguagem Python foi escolhida devido à sua simplicidade e robustez para manipulação de threads. O uso de threads possibilita a execução de múltiplas operações simultâneas, otimizando o tempo de processamento em tarefas intensivas como a verificação de números primos.

2.2 Docker

O Docker permite empacotar a aplicação em um container, garantindo que ela seja executada de forma idêntica em qualquer ambiente. Essa portabilidade é essencial para a distribuição do projeto e para facilitar o deploy em máquinas de diferentes configurações.

2.3 Kubernetes

Kubernetes é utilizado para orquestrar os containers Docker, permitindo a escalabilidade horizontal da aplicação. Com Kubernetes, é possível criar múltiplas réplicas do container, distribuir a carga e gerenciar automaticamente a disponibilidade da aplicação.

3 Estrutura do Projeto

O projeto está organizado da seguinte forma:

- **primos-thread.py:** Script principal contendo a lógica de verificação de números primos utilizando threads.
- **Dockerfile:** Arquivo para construir a imagem Docker da aplicação.
- **kubernetes/deployment.yaml (Opcional):** Manifest para deploy no Kubernetes (pode ser substituído pelos comandos do `kubectl`).
- **README.md:** Documentação (este artigo em LaTeX pode ser convertido para PDF e incluído no repositório).

Repositório GitHub: `https://github.com/gvs22/primos-thread`

4 Código Fonte em Python

A seguir, apresenta-se o código-fonte principal (`primos-thread.py`) que utiliza threads para realizar o cálculo de números primos:

```python
import threading
import random
import time

# Fun  o para verificar se um n mero    primo
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

# Fun  o que cada thread executar
def check_primes(thread_id):
    start = random.randint(1, 50000)
    end = random.randint(start, start + random.randint(10000, 50000))  # Intervalo aleat rio
    numbers = random.sample(range(start, end), 500)  # Seleciona 500 n meros aleat rios

    print(f"* Thread {thread_id} iniciando | Intervalo: {start}-{end} | Analisando {len(numbers)} n meros.")

    start_time = time.time()
    primes = [n for n in numbers if is_prime(n)]
    execution_time = time.time() - start_time

    sleep_time = random.uniform(0.5, 3.0)  # Simula varia  o de carga
    time.sleep(sleep_time)

    print(f"# Thread {thread_id} finalizou em {execution_time:.2f}s (+ {sleep_time:.2f}s de delay) | {len(primes)} n meros primos encontrados.")
    print(f"@ Thread {thread_id} Primos: {primes}\n")

# Criando e iniciando m ltiplas threads
NUM_THREADS = 6  # N mero de threads a serem criadas
threads = []

print("Iniciando processamento com threads...\n")
for i in range(NUM_THREADS):
    thread = threading.Thread(target=check_primes, args=(i,))
    threads.append(thread)
    thread.start()

# Aguardar todas as threads finalizarem
for thread in threads:
    thread.join()

print("\nTodos os c lculos foram finalizados!")
```

Listing 1: Código em Python - primos-thread.py

5 Docker

5.1 Construção e Publicação da Imagem

Para construir a imagem Docker da aplicação, siga os passos abaixo:

1. **Construir a Imagem:**

```
docker build -t gvs22/primos-thread:v4 .
```

2. **Enviar a Imagem para o Docker Hub:**

```
docker push gvs22/primos-thread:v4
```

5.2 Executando a Imagem em Outra Máquina

Para rodar o projeto em outra máquina:

1. **Puxar a Imagem:**

```
docker pull gvs22/primos-thread:v4
```

2. **Executar o Container:**

```
docker run --rm gvs22/primos-thread:v4
```

6 Kubernetes (Opcional)

Caso deseje orquestrar a aplicação com Kubernetes, os seguintes comandos permitem a implantação sem depender de um arquivo YAML:

6.1 Criando o Deployment

1. **Criar o Deployment:**

```
kubectl create deployment primos-thread --image=gvs22/primos-thread:v4
```

2. **Expor o Serviço (caso necessário):**

```
kubectl expose deployment primos-thread --type=LoadBalancer --port=80
```

3. **Escalar o Deployment:**

```
kubectl scale deployment primos-thread --replicas=4
```

4. **Verificar os Pods e Logs:**

```
kubectl get pods
kubectl logs -l app=primos-thread --follow

```

Observação: Se o arquivo `kubernetes/deployment.yaml` estiver disponível, ele pode ser aplicado com:

```
kubectl apply -f kubernetes/deployment.yaml
```

7 Dificuldades Encontradas e Aprendizados

7.1 Dificuldades Encontradas

Durante o desenvolvimento do projeto, enfrentei diversos desafios, tais como:

- **Gerenciamento de Threads:** Sincronizar a execução e garantir que todas as threads finalizem corretamente.
- **Intervalos Aleatórios:** Definir intervalos que garantissem uma boa distribuição dos números para a verificação de primalidade.
- **Integração com Docker e Kubernetes:** Durante o processo de aprendizagem, me atrapei na criação de muitos containers e imagens Docker. Acabei subindo imagens diferentes e cometendo erros com comandos incorretos, o que gerou certa confusão.
- **Resultados nos Logs:** Os resultados dos cálculos de números primos são exibidos nos logs, exigindo atenção para a análise correta das saídas.

7.2 Melhorias Futuras

- **Otimização do Algoritmo:** Implementar algoritmos mais eficientes para a verificação de números primos.
- **Gerenciamento de Logs:** Integrar ferramentas para centralizar e analisar os logs gerados pelas threads.
- **Automatização do Deploy:** Desenvolver scripts mais robustos para automatizar o processo de build, push e deploy.
- **Monitoramento:** Integrar soluções de monitoramento no ambiente Kubernetes para acompanhar a performance e a escalabilidade da aplicação.

8 Motivação para o Uso do Docker e Kubernetes

8.1 Docker

- **Portabilidade:** Garante que a aplicação funcione da mesma forma em qualquer ambiente que suporte Docker.
- **Isolamento:** Os containers isolam as dependências da aplicação, evitando conflitos com outras aplicações.
- **Facilidade de Distribuição:** Com o Docker Hub, é possível distribuir a imagem facilmente para outros usuários.

8.2 Kubernetes

- **Escalabilidade:** Permite criar múltiplas réplicas da aplicação para lidar com aumentos na demanda.
- **Orquestração:** Gerencia a implantação, escalabilidade e balanceamento de carga dos containers.
- **Resiliência:** Automatiza a recuperação de falhas, garantindo alta disponibilidade da aplicação.

Além disso, o uso de **Minikube** para simulação de clusters locais demonstrou ser uma ferramenta excelente para testes e validação antes da implantação em ambientes de produção.

9 Aprendizado e Referências

Durante o desenvolvimento deste projeto, participei do curso *"Aprenda Docker do básico ao avançado e ainda orquestração com Docker Swarm e Kubernetes!"* na Udemy. O foco do curso foi:

- Aprender a construir e gerenciar containers com Docker.
- Entender a orquestração de containers com Docker Swarm e Kubernetes.
- Utilizar o Minikube para simular clusters Kubernetes localmente.

O curso foi fundamental para ampliar meu entendimento sobre os conceitos de containerização e orquestração, além de despertar meu interesse no potencial do Docker e do Kubernetes.

10 Conclusão

O projeto **Primos-Thread** demonstra de forma prática a integração de threads em Python com as tecnologias Docker e Kubernetes para criar uma aplicação escalável e portátil. As dificuldades enfrentadas, como o gerenciamento de múltiplas imagens Docker e erros nos comandos durante a fase de aprendizado, contribuíram significativamente para o meu crescimento técnico. Os resultados dos cálculos de números primos, exibidos nos logs, evidenciam o funcionamento da aplicação e a eficiência do processamento paralelo. A experiência adquirida e os conhecimentos obtidos no curso da Udemy foram essenciais para superar os desafios e aprimorar a solução proposta.

11 Contato

- **Email:** mygabriel0@gmail.com
- **GitHub:** `https://github.com/gvs22/primos-thread`