Implementação de uma função sem servidor(Serverless Function) para gerar números primos inferiores a 1.000

Mestrado em Engenharia Informática
Ano Letivo de 2023/2024, 2º Semestre

João Claudio Paco, M13709
Assistente de Investigação em Engenharia Informática da Universidade Kimpa-Vita em Angola

Resumo *(Abstrac)t*—Este relatório explora a implementação de uma função sem servidor para a geração de números primos usando a linguagem python. Utilizando a computação sem servidor com a Plataforma MS Azure, além, examinamos suas capacidades e eficiência em relação a função implementada. O relatório detalha a escolha da Plataforma que é MS Azure, o processo de implementação e *deployment,* seguido por uma análise de desempenho e conclusões baseadas nos resultados obtidos.

Palavras chave (Keywords) — implementação, função, plataforma, azure, número

I. INTRODUCTION (*HEADING 1*)

A computação sem servidor, também conhecida como *serverless computing*, representa uma evolução significativa na forma como aplicativos e serviços são desenvolvidos e executados na nuvem. Este paradigma elimina a necessidade de gerenciar a infraestrutura tradicional de servidores, permitindo que os desenvolvedores se concentrem exclusivamente na lógica e funcionalidade de suas aplicações.

Em um modelo *serverless*, a execução de código é realizada em resposta a eventos, como solicitações *HTTP*, atualizações de banco de dados, filas de mensagens ou outros ocorrências definidas pelo usuário. Esse modelo é comumente implementado através de funções como um serviço (*FaaS*), onde pequenos trechos de código são carregados, executados e escalonados automaticamente conforme necessário.

A principal vantagem da computação sem servidor é a abstração da gestão da infraestrutura. Os provedores de nuvem gerenciam automaticamente a alocação de recurso, escalabilidade, balanceamento de carga, segurança e manutenção, permitindo que os desenvolvedores entreguem aplicações de forma mais rápida e eficiente. Além disso, o modelo de pagamento por utilização, onde se paga apenas pelo tempo de execução do código e os recursos consumidos, toma a computação sem servidor uma opção economicamente atraente para muitas organizações.

Este relatório discute a implementação de uma função sem servidor para gerar números primos, um problema clássico na teoria dos números, abordando o processo de desenvolvimento e *depçloymen*t e a análise de desempenho da solução proposta.

II. COMPUTAÇÃO SEM SEVIDOR E FUNÇÕES COMO UM SERVIÇO

A computação sem servidor, ou *serverless computing,* é uma arquitetura de computação em nuvem onde o provedor de nuvem gerência a execução do código do usuário, alocando dinamicamente os recursos da máquina conforme necessário. Este modelo elimina a necessidade de os desenvolvedores se preocuparem com a infraestrutura subjacente, permitindo-lhes focar na escrita do código e na lógica da aplicação.

A. Computaçºao Sem Servidor

No modelo de computação sem servidor, os recursos de computação são alocados em resposta a eventos e somente durante a execução desses

eventos. Os desenvolvedores escrevem funções pequenas e modulares que são invocadas por vários tipos de eventos,

como solicitações *HTTP*, atualizações de banco de dados, eventos de armazenamento em nuvem ou filas de mensagens. Este modelo de execução é especialmente eficiente para aplicações que apresentam cargas de trabalho variáveis e imprevisíveis, pois os recursos são escalados automaticamente com base na demanda.

Vantagens:

A computação sem servidor e o *FaaS* oferecem várias vantagens significativas:

1. **Redução de custos operacionais:** pagamento apenas pelo tempo de execução e recursos utilizados,
2. **Escalabilidade automáticas:** as funções são escaladas automaticamente para lidar com picos de carga, sem necessidade de configuração manual,
3. **Rápida implementação:** permite uma rápida iteração e implantação de código, acelerando o ciclo de desenvolvimento,
4. **Simplificação da gestão de infraestrutura:** o provedor de nuvem cuida de todas as preocupações, e segurança.

Desafios:

Apesar de suas inúmeras vantagens, a computação sem servidor também apresenta alguns desafios:

1. **Latência de inicialização:** funções inativas podem enfrentar uma latência inicial (*cold star*) ao serem invocadas,
2. **Limitações de tempo de execução:** muitas plataformas impõem limites ao tempo máximo de execução de uma função, o que pode não ser adequado para todas as aplicações,
3. **Complexidade de *debugging*:** depurar e monitorar funções sem servidor pode ser mais complexo devido a natureza distribuída da execução.

B. Funções como Serviço (FaaS)

Funções como um servidor (*FaaS*), é um componente crucial da computação sem servidor. *FaaS* permite que os desenvolvedores implantem funções individuais na nuvem que são executadas de maneiras autônoma e escalável. Essas funções são escritas em diversas linguagens de programação suportadas pelos provedores de nuvem e podem ser acionadas por uma ampla gama de eventos.

Principais características do *FaaS* incluem:

1. **Execução sob demanda:** as funções são executadas somente em resposta aos eventos específicos, economizando recursos quando não estão em uso,
2. **Escalabilidade automática:** as funções escalam automaticamente para atender ao número de eventos recebidos, sem necessidade de intervenção manual,
3. **Modelo de pagamento baseado no uso:** os custos são incorridos apenas durante o tempo de execução das funções, tornando-o um modelo económico para cargas de trabalho esporádicas,
4. **Facilidade de gerenciamento:** os desenvolvedores são liberados da gestão de infraestrutura, permitindo foco total na lógica de negócio.

A computação sem servidor elimina a necessidade de gerenciar servidores físicos ou virtuais, permitindo que os desenvolvedores se concentrem no código. Funções como um Serviço (*FaaS*) é um modelo de execução onde é carregado e executado em resposta a eventos. Esse paradigma é ideal para tarefas esporádicas e de curta duração, como a geração de números primos.

III. PLATAFORMAS PARA COMPUTAÇÃO SEM SERVIDOR

Diversas plataformas oferecem suporte de computação sem servidor, incluindo:

- **AWS Lambada:** oferece uma integração robusta com o ecossistema AWS,
- **Google Cloud Function:** integra-se facilmente com os serviços do Google Cloud,
- **Microsoft Azure Functions:** possui forte integração com o Azure e suporte para diversas linguagens de programação,
- **IBM Cloud Functions:** baseado no Apache Open Whisk, oferece uma alternativa de Código aberto.

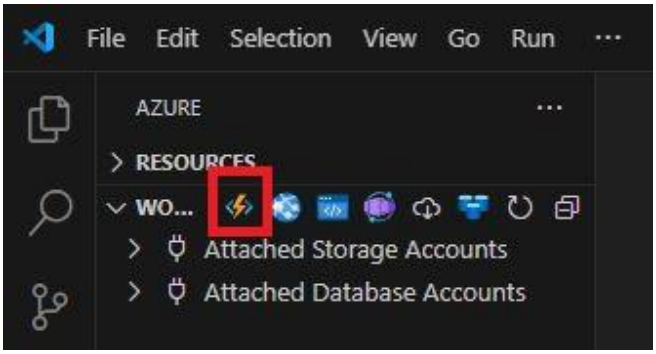
IV. ESCOLHA DA PLATAFORMA, IMPLEMENTAÇÃO E *DEPLOYMENT* DA FUNÇÃO

Para este projeto, escolhemos *Microsoft Azure Function* que oferece uma Plataforma flexível e altamente, escalável, suportando diversas linguagens de programação e permitindo integração com uma ampla gama de serviço Azure. Os códigos da função foram ecritos em linguagem python.

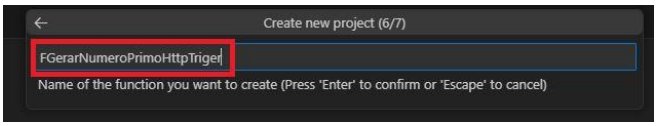
IMPLEMENTAÇÃO DA FUNÇÃO:

A. Ao nível do VS CODE:

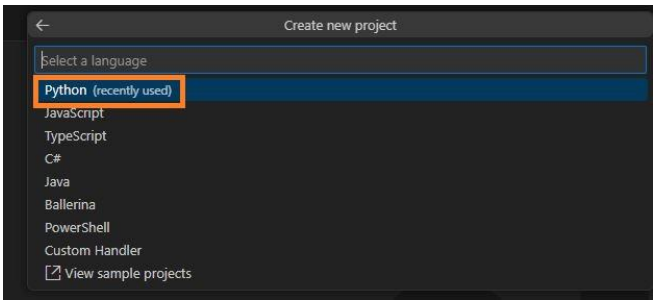
1. Sign in com a conta Azure,
2. No icóno de Azure Function, clicar: "Create new project"



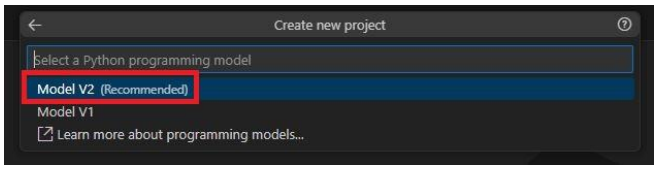
3. Nomear a aplicação de função



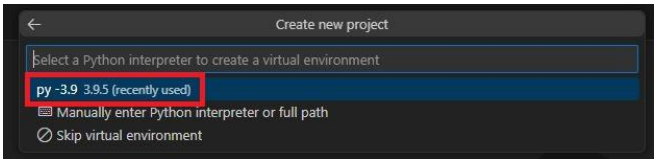
4. Escolher a pilha de runtime: **Python**



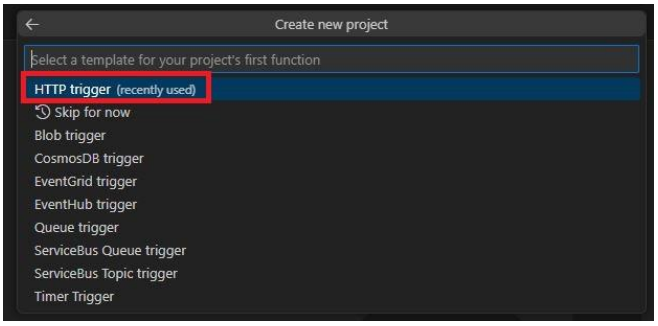
5. Escolher o modelo **:** Model v2



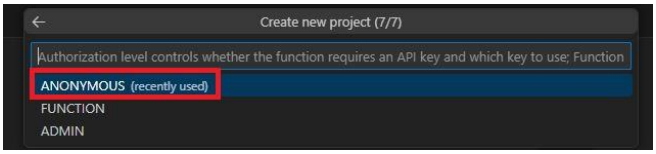
6. Escolher a sua versão :



7. Escolher o modelo de programação:

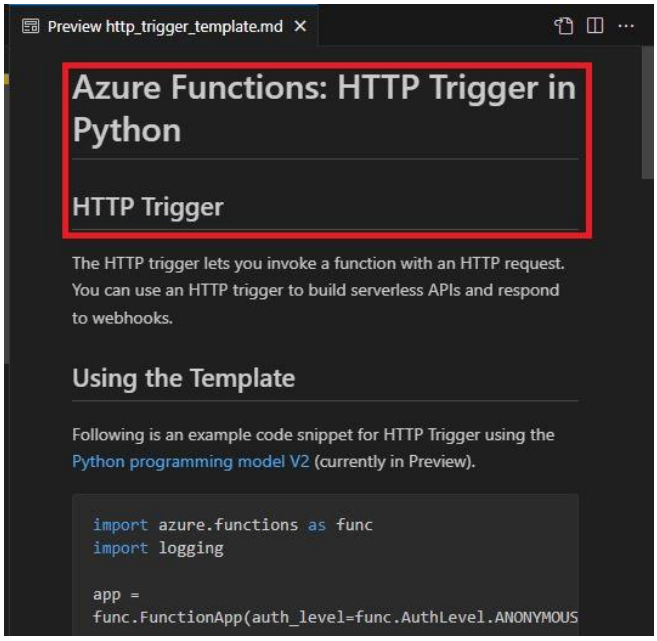


8. Escolher o nível de autorização: *Anonimous*

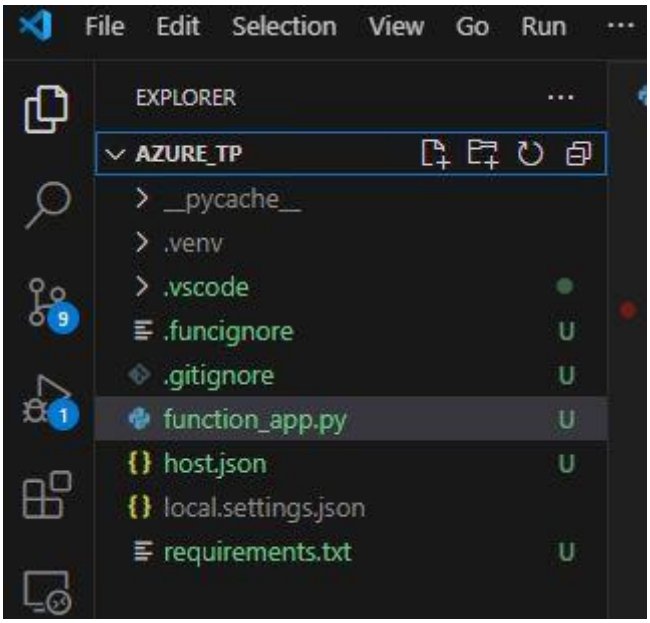


9. Output do processo de criação da função:

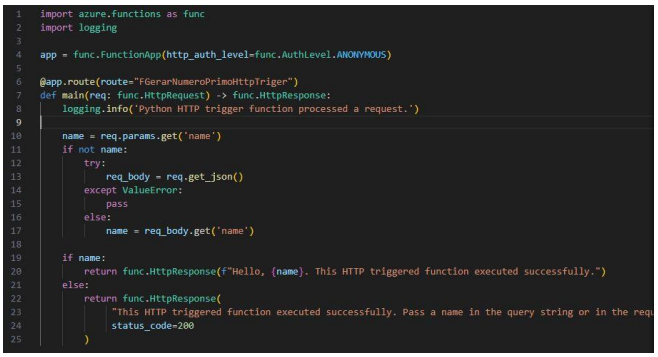
10. Mensagem de Confirmação



11. Os ficheiros gerados para projeto:



12. Os códigos vs code da função :



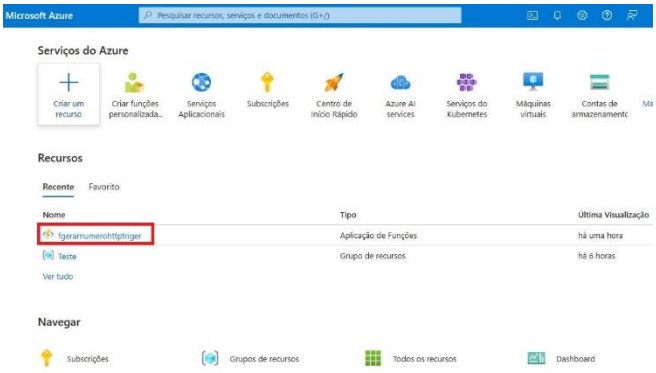
```python
import azure.functions as func
import logging

app = func.FunctionApp(http_auth_level=func.AuthLevel.ANONYMOUS)

@app.route(route="FGerarNumeroPrimoHttpTriger")
def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('Python HTTP trigger function processed a request.')

    name = req.params.get('name')
    if not name:
        try:
            req_body = req.get_json()
        except ValueError:
            pass
        else:
            name = req_body.get('name')

    if name:
        return func.HttpResponse(f"Hello, {name}. This HTTP triggered function executed successfully.")
    else:
        return func.HttpResponse(
             "This HTTP triggered function executed successfully. Pass a name in the query string or in the requ
             status_code=200
        )
```

13. Os códigos Azure Application da função : gereção de números primos

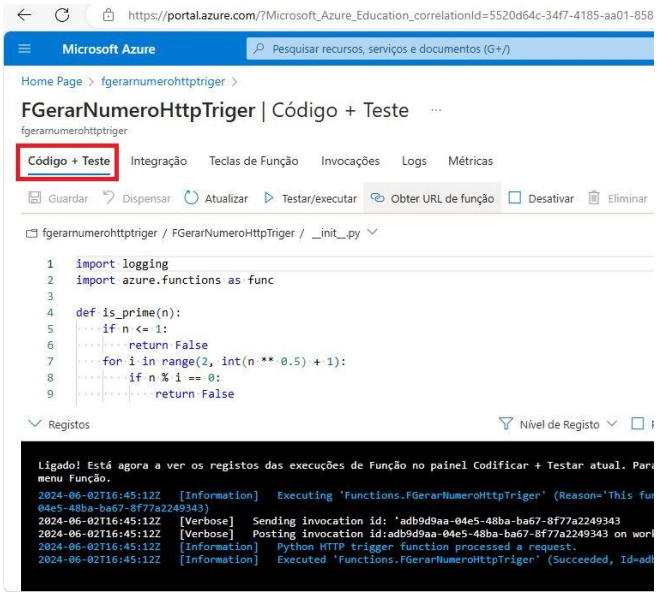
```python
import logging
import azure.functions as func

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 main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('Python HTTP trigger function processed a request.')
```

B. Ao nível do portal de AZURE

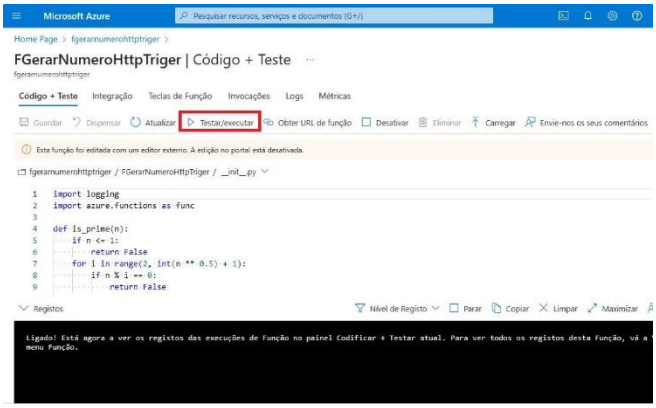
1. Interface das funções Azure: função criada



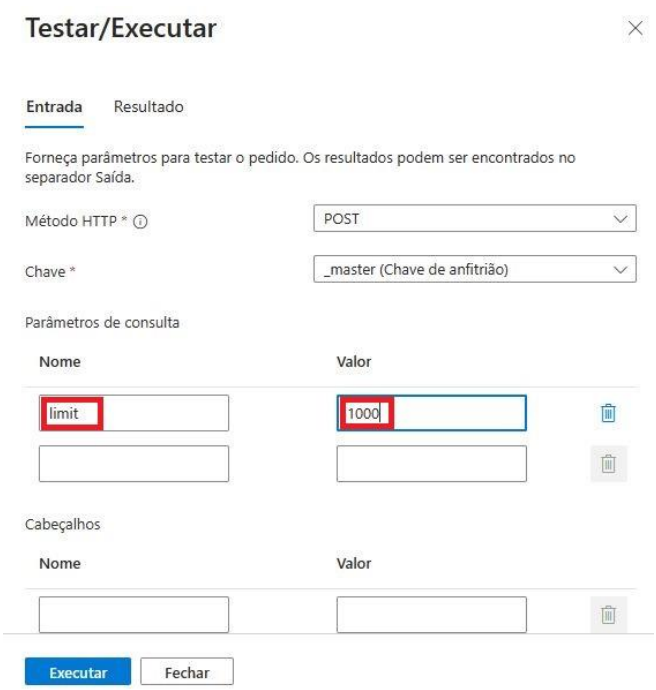
2. Interface “Código + Test” da função criada



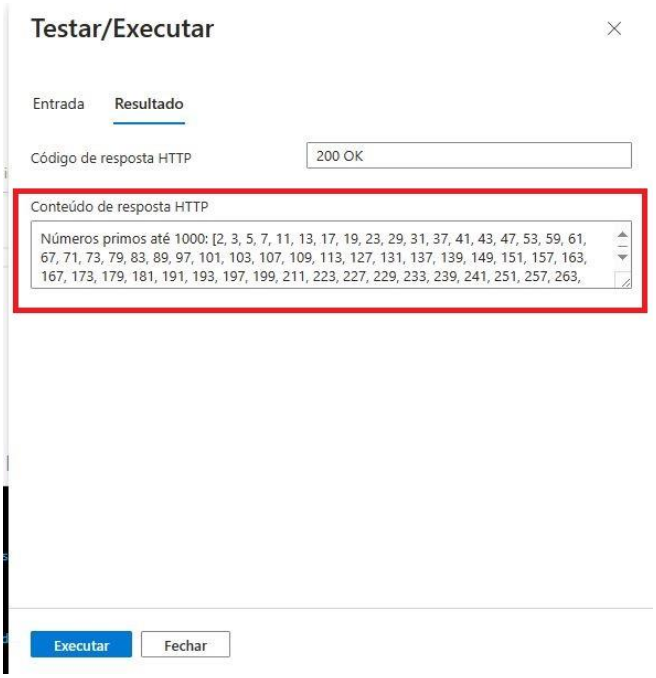
3. Interface “Testar e Executar” da função criada



4. Teste e Execução

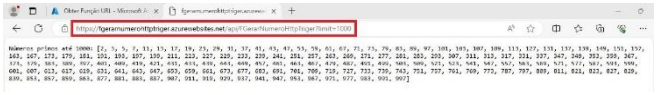


5. Interface: Resultado de execução da função



6. Interface: URL Gerado do resulatado

7. Execução da função por url:



8. Url :

https://fgerarnumerohttptriger.azurewebsites.net/api/FGerarNumeroHttpTriger?

V. ANÁLISE DE DESEMPENHO

A análise de desempenho da função sem servidor foi realizada medindo a latência em utilização concurrente da função e o could starts relative a mesma.

V.1 A latência

A latência de uma plicação se refere ao tempo que leva para uma solicitação feita por um cliente (por exemplo, um navegador web ou um aplicativo) ser processada e receber uma resposta do servidor. Em outras oalavras, é o tempo total entre a emissão deuma solicitação e o recebimento da resposta correspondente.

V.1.1 O resultado da primeira execução da função

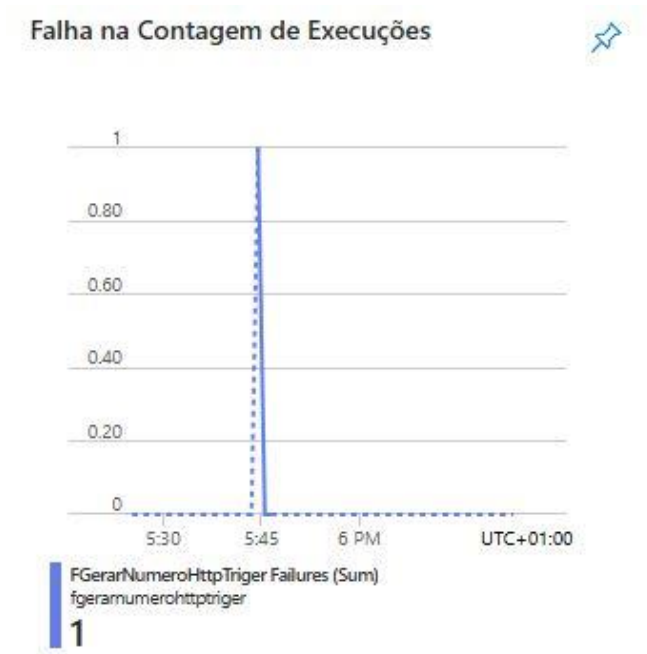
1º) Contagem de Execuçºao Total:

Interpretação: A função foi executada um total de 6 vezes no período de medição específico (5:45)

2º) Contagem de Execução Realizada com èxito

Interpretação: das 6 execuções totais, 5 foram concluídas com sucesso. Isso significa que a função completou sua tarefa conforme o esperado em 5 dessas execuções.

3º) Falha na Contagem de Execução



Interpretação: houve 1 falha na execução da função. Isso significa que uma das 6 execuções não foi concluída com sucesso devido a um erro ou problema.

4º) Resumo e nterpretação geral:

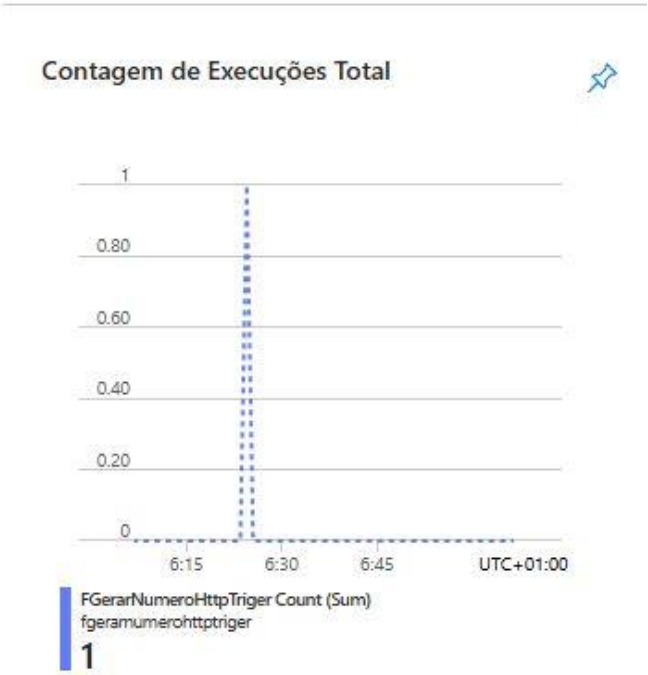
- *Taxa de sucesso :* A taxa de sucesso das execuções é alta, com 5 execuções bem-sucedidas de 6, resultatndo emu ma taxa de sucesso de aproximadamente 83.3%
- *Falhas:* Houve uma falha de execução, indicacndo a necessidade de investigar e corrigir possíveis problemas que possam estar causando essa falha.

V.2 O could starts

O “Cold starts” refere-se ao tempo adicional que leva para iniciar uma função na primeira vez em que é chamado ou após um período de inatividade. Este fenômeno geralmente é medido observavndo a latência inicial em comparação com as execuções subsequentes.

V.2.1 O Resultado da segunda execução da função

1º) Contagem de Execução Total:

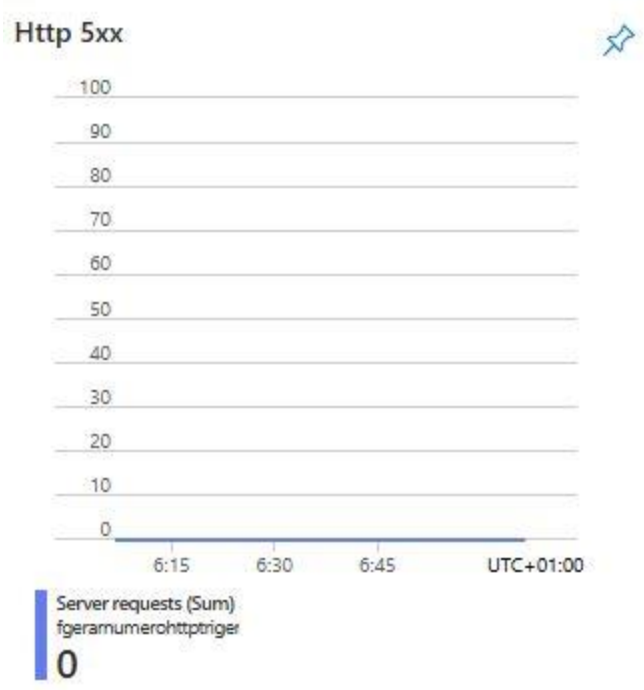


Interpretação: A função foi executada uma única vez no período de medição específico, conform indica o gráfico.

2º) Contagem de Execução Realizada com èxito

Interpretação: A execução durante o period específicado foi bem sucedida. Portanto a única execução registrada foi concluída com sucesso.

3º) Falha na Contagem de Execução



Interpretação: Não houve falhaa na execução da função durante o período específicado. Todas execuções (no caso, uma única execução) foram bem sucedidas,

4º) *Resumo e interpretação geral:*

- Taxa de sucesso: A única execução durante o período especificado foi bem-sucedida, resultatndo emu ma taxa de sucesso de 100%
- Falhas: Não houve falhas na execução, o que é positive.

V.3 Análise comparative entre a latência e o Cold Starts

1. *Número de Execuções:*
   - Primeira Execução: 6 execuções no tal
   - Segunda Execução: 1 execução no total
2. *Taxa de Sucesso:*
   - Primeira execução: 5/6 execuções bem sucedidas (83.3%)
   - Segunda execução: 1/1 execução bem sucedida (100%)
3. *Falhas:*
   - Primeira execução: 1 falha.
   - Segunda execução: Zero falha

VI. CONCLUSÕES

A implementação de uma função sem servidor para gerar números primos demonstrou ser eficiente e escalável. A escolha do AWS Lambada facilitou o processo de *deployment* e geração, proporcionando uma solução flexível e economicamente viável. A computação sem servidor mpstrou-se uma excelente opção para aplicações que necessitam de execução sob demanda e escalabilidade autom,ática.

VII. REFERÊNCIAS

1. *Amazon Web Services*, “AWS Lambada”, disponível em: AWS Lmabda
2. *Google Cloud* “Google Cloud Functions” disponível em Google Cloud Function
3. *IBM Cloud* “IBM Cloud Functions” disponível em IBM Cloud Functions