UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE CENTRO DE INFORMÁTICA

FELIPE GOMES DE SANTANA SILVA

Service Meshes e Zero Trust: Avaliando o Desempenho em Ambientes Distribuídos

Recife
2025

FELIPE GOMES DE SANTANA SILVA

Service Meshes e Zero Trust: Avaliando o Desempenho em Ambientes Distribuídos

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Orientador (a): Prof. Dr. Carlos André Guimarães Ferraz

Recife

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Silva, Felipe Gomes de Santana.
Service Meshes e Zero Trust: Avaliando o Desempenho em Ambientes Distribuídos / Felipe Gomes de Santana Silva. - Recife, 2025.
9 p. : il., tab.

Orientador(a): Carlos Andre Guimaraes Ferraz
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2025.
Inclui referências.

1. Microsserviços. 2. Segurança. 3. Nuvem. I. Ferraz, Carlos Andre Guimaraes. (Orientação). II. Título.

000 CDD (22.ed.)

FELIPE GOMES DE SANTANA SILVA

Service Meshes e Zero Trust: Avaliando o Desempenho em Ambientes Distribuídos

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Aprovado em: 05/08/2025

BANCA EXAMINADORA

_____

Prof. Dr. Carlos André Guimarães Ferraz (Orientador)

Universidade Federal de Pernambuco

_____

Prof. Dr. David Junio Mota Cavalcanti (Examinador Interno)

Universidade Federal de Pernambuco

_____

Prof. Dr. Carlos André Guimarães Ferraz (Examinador Interno)

Universidade Federal de Pernambuco

Service Meshes e Zero Trust: Avaliando o Desempenho em Ambientes Distribuídos

Felipe G. S. Silva[1],**Carlos A. G. Ferraz**[1]

[1]Centro de Informática – Universidade Federal do Pernambuco (UFPE)
Caixa Postal 7851 50732-970 – Recife – PE – Brazil

{fgss,cagf}@cin.ufpe.br

Abstract. *Distributed systems based on microservices require robust security strategies to deal with communication between its components. The Zero Trust model, which enforces strict access control and continuous authentication, has proven effective—especially when implemented using service meshes. To assess its performance impact, three security levels were tested in simulated environments. Results showed an average latency increase with policies enabled, though communication stability remained unaffected throughout the tests.*

Resumo. *Ambientes distribuídos baseados em microsserviços demandam estratégias de segurança robustas para lidar com comunicações entre seus componentes. O modelo Zero Trust, ao restringir acessos e exigir autenticação contínua, tem se mostrado eficaz nesse cenário, especialmente quando aplicado por meio de service meshes. Para entender o impacto dessa abordagem na performance, foram simulados três cenários com diferentes níveis de segurança. Os resultados apontaram aumento médio na latência com as políticas ativas, mas sem comprometer a estabilidade na comunicação.*

1. Introdução

O uso da arquitetura de microsserviços tornou-se um padrão consolidado no desenvolvimento de sistemas distribuídos, especialmente em aplicações modernas implantadas em ambientes de nuvem. Essa abordagem, ao quebrar sistemas monolíticos em serviços menores e independentes, trás escalabilidade, resiliência e flexibilidade. Contudo, esse modelo também traz novos desafios, principalmente relacionados à interação entre os diversos componentes da aplicação, como autenticação e autorização e observabilidade na comunicação. Para enfrentar tais desafios, soluções como o service mesh vêm sendo amplamente usadas por empresas que desejam melhorar a visibilidade, segurança e controle do tráfego interno entre os microsserviços [MAIA; CORREIA 2022].

Apesar dessas evoluções, persiste um desafio: os ambientes distribuídos em nuvem operam milhares de workloads que se comunicam constantemente dentro de clusters, o que amplia consideravelmente a possibilidade de ataque. Tradicionalmente, os sistemas administravam sua segurança em perímetros definidos por firewalls, confiando implicitamente em qualquer entidade que estivesse dentro da rede. Diante desse cenário, surge o modelo de segurança Zero Trust como uma alternativa mais adequada para ambientes distribuídos em nuvem. [RODIGARI et al. 2023].

Nesse contexto, surge o modelo de segurança Zero Trust como alternativa mais adequada para arquiteturas distribuídas. Assim como explicam Kang et al. (2023),

esse modelo propõe a verificação contínua de identidades, transações e dispositivos, mesmo quando já se encontram dentro da rede corporativa. Essa abordagem baseia-se em princípios como ”verify explicitly”, use ”least-privilege access” e ”assume breach” [ROSE et al. 2020], que se alinham diretamente à necessidade de mitigar ameaças internas e externas em tempo real. Estudos recentes, como os de Mehraj e Banday (2023), reforçam que essa estratégia é eficaz em cenários nos quais dados, dispositivos e usuários estão dispersos em múltiplas localidades.

Contudo, embora o Zero Trust e o service mesh tenham ganhado espaço na literatura e na prática, ainda existe uma lacuna: poucos trabalhos avaliam os impactos práticos da aplicação rigorosa dessas políticas em ambientes corporativos. Segundo Costa (2024), há indícios de que o aumento da segurança possa afetar negativamente a performance das aplicações.

Diante desse panorama, o objetivo deste trabalho é avaliar, por meio de experimentos controlados, o impacto da aplicação de políticas de segurança baseadas em Zero Trust em ambientes com service mesh.

2. Metodologia

O Istio foi escolhido neste trabalho como implementação de service mesh, tanto por sua popularidade no ecossistema Kubernetes quanto por oferecer, de forma nativa, recursos como mTLS, roteamento avançado, observabilidade e controle de políticas [Istio, 2024]. Através dessas capacidades, torna-se possível aplicar os princípios do Zero Trust de forma granular e gerenciável.

Foram definidos três cenários distintos: comunicação entre microsserviços sem Istio, com Istio instalado mas sem políticas de segurança, e com Istio e as políticas de Zero Trust devidamente aplicadas. Os experimentos executados neste trabalho buscam medir a latência total e a estabilidade dos serviços nesses cenários, possibilitando uma análise objetiva dos trade-offs entre segurança e performance.

2.1. Arquitetura da Solução

O sistema foi estruturado com base em uma arquitetura de microsserviços, implantada em um cluster Kubernetes, contando com a malha de serviço Istio ativada. A solução é composta por dois componentes principais desenvolvidos em Kotlin, utilizando o framework Vert.x. Esses componentes foram transformados em imagens Docker e implantados em um único namespace dentro do cluster.

Toda a comunicação entre os microsserviços acontece exclusivamente dentro do namespace do cluster (Figura 1), sem exposição externa dos endpoints, favorecendo a aplicação de mecanismos de segurança e alinhando-se a cenários corporativos reais. Além dos dois serviços principais, foi implantado um terceiro serviço com a função de gerar carga de requisições, que simula acessos ao serviço number-gen-service.

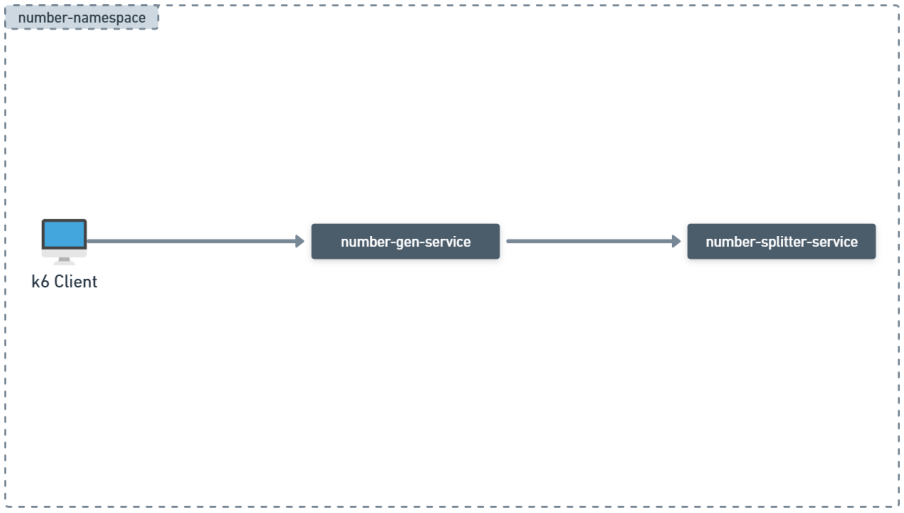

Figura 1. Arquitetura da solução

2.2. Aplicações implementadas

2.2.1. number-gen-service

Este serviço possui um endpoint HTTP do tipo GET que, ao ser chamado, gera um número aleatório de três dígitos. Esse número é então inserido em um campo ”number” dentro de um JSON e enviado por meio de uma requisição POST ao serviço number-splitter-service.

2.2.2. number-splitter-service

Este segundo serviço recebe requisições POST contendo um JSON com o campo ”number”, representando uma string numérica de três dígitos. A função do serviço é separar os dígitos e devolvê-los como um Json array.

2.2.3. k6-client

Este serviço dispara chamadas HTTP GET para o serviço number-gen-service. Sua função é medir a latência dessas chamadas.

Na Figura 2, é possível visualizar todo o fluxo da comunicação dentro do namespace.

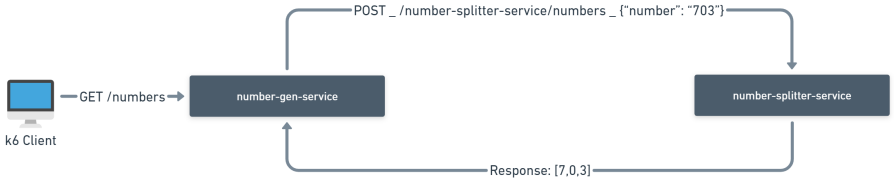

Figura 2. Fluxo de comunicação entre as aplicações

3. Configurações de Segurança (Zero Trust)

Para garantir que a comunicação entre os serviços siga os princípios da arquitetura Zero Trust, algumas configurações de segurança foram aplicadas no ambiente Kubernetes.

3.1. Peer Authentication com mTLS

Foi criada uma política PeerAuthentication com o modo STRICT (Figura 3), exigindo que toda comunicação entre serviços ocorra utilizando mTLS (Mutual TLS). Isso garante que apenas serviços dentro da malha e autenticados possam se comunicar.

```yaml
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: number
spec:
  mtls:
    mode: STRICT
```

Figura 3. Manifesto PeerAuthentication

3.2. Authorization Policy: deny-all

Além da autenticação mútua, foi definida uma politica que serve para definir regras de autorização na comunicação dos serviços do namespace (Figura 4). Como nenhuma regra de permissão é especificada, nenhuma requisição é autorizada. Na prática, toda comunicação de entrada para os serviços do namespace ”number” será bloqueada por padrão.

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: number
spec: {}
```

Figura 4. Manifesto deny-all

3.3. Authorization Policy: only-post-from-number-gen

Para especificar uma regra de permissão no namespace, foi criada uma última política (Figura 5) que permite exclusivamente que o serviço number-gen-service (identificado por sua ServiceAccount) envie requisições POST para um endpoint específico do number-splitter-service.

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: only-post-from-number-gen
  namespace: number
spec:
  selector:
    matchLabels:
      app: number-splitter
  rules:
  - from:
    - source:
        principals: ["cluster.local/ns/number/sa/sa-number-gen"]
    to:
    - operation:
        methods: ["POST"]
        paths: ["/number-splitter-service/numbers"]
```

Figura 5. Manifesto deny-all

Com essas configurações, conseguimos aplicar os conceitos da arquitetura Zero Trust (ROSE et al., 2020):

- Verify explicitly: A identidade dos serviços é verificada por certificados emitidos pelo Istio.
- Use least-privilege access: Somente serviços autorizados têm permissão de comunicação.
- Assume breach: Todo o tráfego interno é criptografado via mTLS e requisições externas ou não autorizadas são bloqueadas.

3.4. Configurações do Ambiente de Execução

Os testes foram realizados em ambiente local utilizando o Rancher Desktop, que disponibiliza um cluster Kubernetes adequado para desenvolvimento e simulações. Configurações do sistema:

- Processador: AMD Ryzen 7 9800X3D
- Memória RAM: 32 GB
- Sistema Operacional: Rancher Desktop WSL Distribution (Kernel 5.15.167.4-microsoft-standard-WSL2)

Dentro do Rancher Desktop, os recursos alocados para o ambiente Kubernetes foram:

- Memória RAM: 16 GB
- CPU : 16 núcleos

3.5. Teste de Desempenho

Para medir o impacto das configurações de segurança no desempenho da aplicação, utilizou-se o K6 que é uma ferramenta de testes de carga HTTP, onde configuramos um script e a ferramenta se encarrega de executar os testes de acordo com o que foi passado. Os testes foram realizados por um pod específico chamado k6-client, contendo o K6 instalado. O script configurado (Figura 6) simulava 150 chamadas por segundo por um período de 60 segundos para o serviço number-gen-service.

```
import http from 'k6/http';

    export const options = {
      scenarios: {
        zero_trust_test: {
          executor: 'constant-arrival-rate',
          rate: 150,
          timeUnit: '1s',
          duration: '60s',
          preAllocatedVUs: 100,
          maxVUs: 200,
        },
      },
      thresholds: {
        http_req_duration: ['p(95)<1000', 'avg<500'],
        http_req_failed: ['rate<0.01'],
      },
    };

    export default function () {
      http.get('http://number-gen-service:8085/number-gen-service/numbers');
    }
```

Figura 6. Script para execução de testes

Essa configuração foi definida devido a alocação de recursos que foi feita ao cluster Kubernetes, com isso a sobrecarga de recursos é evitada e conseguimos obter resultados consistentes de latência. Cada uma dessas chamadas desencadeia uma requisição POST para o number-splitter-service, cobrindo todo o fluxo entre os serviços.

3.6. Cenários Avaliados

Para investigar o impacto do Zero Trust no desempenho, os testes foram realizados em três cenários distintos:

1. Istio com Zero Trust ativado: Incluindo mTLS e regras de autorização aplicadas conforme descrito.
2. Istio sem Zero Trust: O Istio está instalado (containers side car injetados), mas sem qualquer política de autenticação ou acesso restrito.
3. Ambiente sem Istio: Os microsserviços se comunicam diretamente através da rede Kubernetes, sem intervenção da malha de serviço.

Esses três cenários permitiram comparar o impacto real das políticas Zero Trust na latência.

4. Resultados e Discussão

Os resultados obtidos do latência estão resumidos na Tabela 1.

Tabela 1. Resultados em diferentes cenários de teste

Cenário	Média	Mediana	Desvio Padrão
Sem Istio	1.3679 ms	1.3647 ms	0.2185 ms
Istio sem Zero Trust	2.4076 ms	2.3904 ms	0.1629 ms
Istio com Zero Trust	2.7894 ms	2.7793 ms	0.2026 ms

A partir da análise, nota-se que a introdução do Istio — mesmo sem regras de segurança — causa um aumento perceptível no tempo de resposta, o que é esperado, pois o padrão sidecar (utilizado por service meshes como Istio) traz uma sobrecarga natural na

camada de comunicação [MICROSOFT, 2025]. A média salta de aproximadamente 1,36 ms para 2,40 ms, representando um crescimento de cerca de 76%. Ao aplicar a política de Zero Trust com mTLS e autorização baseada em identidade, a latência média se eleva ainda mais, alcançando 2,78 ms. Comparando este valor com o cenário sem Istio, temos um aumento de cerca de 104% na média e 103% na mediana.

Além disso, vale destacar que os valores de desvio padrão se mantiveram estáveis entre os cenários, o que indica que, embora o tempo de resposta tenha aumentado com a adoção de segurança, a variação entre os tempos foi relativamente constante.

A Figura 7. mostra uma comparação entre três situações distintas em relação ao tempo de resposta de requisições. A ideia é visualizar como diferentes configurações de rede e segurança afetam a latência.

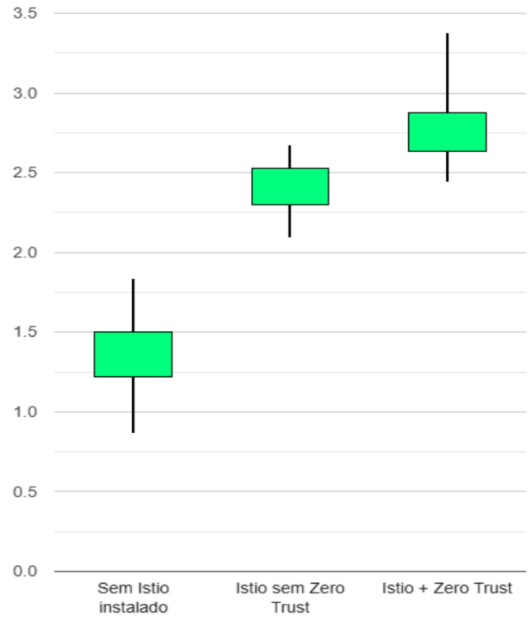

Figura 7. Gráfico de tempo de resposta para cada cenário (em milissegundos)

Esses resultados estão alinhados com observações feitas em estudos similares. No trabalho de Costa (2024), ao comparar cenários semelhantes em um ambiente local com Istio, observou-se um acréscimo na mediana de 0,06 ms (sem service mesh instalado) para 0,44 ms (com Istio e Zero Trust), representando um aumento percentual de aproximadamente 633Apesar das diferenças nos valores absolutos — possivelmente influenciadas por fatores como infraestrutura e tecnologias empregadas — a conclusão permanece consistente: a introdução de mecanismos de segurança com base em Zero Trust resulta em penalidades de desempenho mensuráveis.

Enquanto este estudo utilizou serviços codificados em Kotlin com Vert.x, o trabalho de Costa empregou uma arquitetura mista com serviços em Golang, Python e JavaScript. Além disso, a carga de testes aqui aplicada simulou requisições contínuas por 60 segundos, enquanto no estudo de referência optou-se por disparos agrupados em intervalos de 10 ms, com um total de 10.000 requisições. Essas distinções ajudam a justi-

ficar as diferenças nos resultados absolutos, mas não invalidam a tendência observada em ambos os estudos.

Por fim, vale ressaltar que em todos os testes executados foi garantido que os microsserviços se mantiveram estáveis e sem sobrecarga, processando o volume de requisições de maneira consistente. A ferramenta K6 demonstrou que o número de requisições por segundo foi mantido durante toda a duração dos testes, com taxa de falha nula, como demonstrado na Figura 8.

Figura 8. Resultado de um dos testes executados

5. Conclusão

Os testes realizados mostraram que a aplicação do modelo Zero Trust, junto ao uso de service meshes, traz um aumento na latência das comunicações entre os microsserviços. Apesar desse impacto, os serviços mantiveram estabilidade e responderam de forma consistente ao longo dos experimentos. Isso demonstra que é possível adotar práticas de segurança mais rígidas sem comprometer a integridade operacional do sistema. Ainda assim, cabe às empresas analisarem cuidadosamente o equilíbrio entre segurança e desempenho, considerando as particularidades de cada ambiente antes de aplicar essas estratégias de forma ampla.

6. Referências

Costa, L. A. G. Zero Trust and service meshes on microservice cloud-based applications: A comparative study. Master's dissertation, Centro de Informática, Universidade Federal de Pernambuco. 2024.

Istio. What is Istio? Disponível em: https://istio.io/latest/docs/overview/what-is-istio/. Acesso em: 20 de julho, 2025.

Kang, H., Liu, G., Wang, Q., Meng, L., and Liu, J. Theory and application of Zero Trust security: A brief survey. Entropy, 25, 1595. https://doi.org/10.3390/e25121595. 2023.

Maia, T. and Correia, F. F. Service mesh patterns. 27th European Conference on Pattern Languages of Programs (EuroPLoP '22), Irsee, Germany. ACM, New York, NY, USA. https://doi.org/10.1145/3551902.3551962. 2022.

Mehraj, S. and Banday, M. T. Establishing a Zero Trust strategy in cloud computing environment. 2020 International Conference on Computer Communication and Informatics (ICCCI-2020), Coimbatore, India. IEEE. 2020.

Microsoft. Sidecar pattern. Disponível em: https://learn.microsoft.com/en-us/azure/architecture/patterns/sidecar. Acesso em: 21 de julho, 2025.

Rodigari, S., O'Shea, D., McCarthy, P., McCarry, M., and McSweeney, S. Performance analysis of Zero-Trust multi-cloud. 2021 IEEE 14th International Conference on Cloud Computing (CLOUD). IEEE, 2021. p. 730-732.

Rose, S., Borchert, O., Mitchell, S., and Connelly, S.. Zero Trust Architecture. NIST Special Publication 800-207. https://doi.org/10.6028/NIST.SP.800-207. 2020