

# Relatório de Laboratório 2025.S1-E1.02

Jonas de Araújo Luz Junior

José de La Cruz Iraheta

Pedro Jardelino Neto

Universidade de Fortaleza (Unifor)

17 de junho de 2025

# Sumário

<b>1</b>	<b>Especificações do Projeto de Laboratório</b>	<b>1</b>
1.1	Especificações das Tarefas . . . . .	1
1.1.1	Tarefa 1 - Configuração do Ambiente . . . . .	1
1.1.2	Tarefa 2 - Implantação da Aplicação . . . . .	1
1.1.3	Tarefa 3 - Teste de Desempenho . . . . .	1
1.1.4	Tarefa 4 - Teste de Resiliência . . . . .	2
1.1.5	Tarefa 5 - Teste de Desempenho com Escalonamento Automático . . . . .	2
1.2	Entregáveis . . . . .	3
1.2.1	Entrega Parcial 1 - Foco: Tarefas 1 e 2. . . . .	3
1.2.2	Entrega Parcial 2 - Teste de Desempenho e Análise de Overhead. Foco: Tarefa 3. . . . .	3
1.2.3	Entrega Parcial 3 - Testes de Resiliência e Escalonamento Automático. Foco: Tarefas 4 e 5. . . . .	3
1.2.4	Entrega Final - Relatório Consolidado e Repositório Completo. Foco: Integração, refinamento e conclusões. . . . .	4
<b>2</b>	<b>Ambiente de Operação</b>	<b>5</b>
2.1	Ambiente Operacional . . . . .	5
2.2	Repositório Git . . . . .	5
<b>3</b>	<b>Instalações</b>	<b>6</b>
3.0.1	Atualização do Sistema . . . . .	6
3.0.2	Instalação do Docker . . . . .	6
3.0.3	Instalação do kubectl . . . . .	7
3.0.4	Instalação do Minikube . . . . .	7
3.0.5	Instalação do Istio . . . . .	8
3.0.6	Implantação da Online Boutique . . . . .	8
3.0.7	Instalação do Locust . . . . .	8
<b>4</b>	<b>Testes de Carga</b>	<b>13</b>
4.1	Passo a Passo da Realização dos Testes . . . . .	13
4.2	Resultados Obtidos . . . . .	14
4.3	Análise dos Resultados . . . . .	18
4.3.1	Trade-off Latência × Robustez . . . . .	18
4.3.2	Impacto na Throughput e Scalabilidade . . . . .	18
4.3.3	Tail Latency e Jitter . . . . .	18
4.3.4	Estabilidade em Longo Prazo . . . . .	18
4.3.5	Conclusões . . . . .	18

<b>5</b>	<b>Teste de Resiliência</b>	<b>19</b>
5.1	Objetivo e Metodologia . . . . .	19
5.2	Resultados Obtidos . . . . .	19
5.3	Análise dos Resultados . . . . .	19
5.4	Conclusões . . . . .	20
<b>6</b>	<b>Teste de Escalonamento Automático</b>	<b>21</b>
6.1	Objetivo e Metodologia . . . . .	21
6.2	Resultados Obtidos . . . . .	21
6.3	Análise dos Resultados . . . . .	21
6.4	Conclusões . . . . .	22

## Resumo

Este relatório apresenta a execução, análise e integração das atividades práticas realizadas no contexto do Laboratório de Sistemas Distribuídos, com foco na implantação e avaliação de uma arquitetura baseada em microsserviços sob Kubernetes com suporte ao Istio. A aplicação Online Boutique foi utilizada como base para os experimentos, sendo implantada em dois ambientes distintos: com e sem injeção automática de sidecars do Istio.

As tarefas foram divididas em cinco etapas: (1) configuração do ambiente de laboratório, (2) implantação da aplicação, (3) testes de carga para avaliação de overhead do Istio, (4) testes de resiliência com injeção de falhas via Istio, e (5) testes de escalonamento automático com Kubernetes HPA.

Foram utilizados recursos como Locust para simulação de carga, além de ferramentas do ecossistema Kubernetes (kubect!, Minikube, Istioctl) para orquestração e monitoramento. Os testes demonstraram o impacto do Istio em termos de latência e vazão, mas também evidenciaram ganhos significativos em robustez e estabilidade. A aplicação do HPA permitiu absorver aumentos súbitos de carga, com escalonamento eficiente de pods. Ao final, o trabalho integra todas as análises em um relatório técnico consolidado, documentando tanto os ganhos quanto os ônus-bônus inerentes à adoção de uma malha de serviços e estratégias de resiliência.

# Capítulo 1

## Especificações do Projeto de Laboratório

### 1.1 Especificações das Tarefas

#### 1.1.1 Tarefa 1 - Configuração do Ambiente

- Configurem um repositório Git compartilhado para o projeto.
- Instalem e configurem a distribuição Kubernetes local escolhida em suas máquinas (ou em uma máquina compartilhada pela equipe). Certifiquem-se de alocar recursos suficientes (CPU/RAM).
- Instalem o Istio no cluster Kubernetes, utilizando o perfil de instalação demo ou default. Verifiquem a instalação.

#### 1.1.2 Tarefa 2 - Implantação da Aplicação

- Obtenham os manifestos de implantação da aplicação Online Boutique.
- **Implantação base:** implantem a aplicação sem a injeção automática de sidecars do Istio (ou seja, em um namespace sem o rótulo `istio-injection=enabled`). Verifiquem se todos os serviços estão rodando e se a aplicação está acessível .
- **Implantação com Istio:** habilitem a injeção automática de sidecars do Istio para um novo namespace (e.g., `online-boutique-istio`) e implantar a aplicação novamente neste namespace. Verifiquem se os sidecars foram injetados (`kubectl get pods -n -o wide` deve mostrar 2/2 containers por pod) e se a aplicação continua acessível.

#### 1.1.3 Tarefa 3 - Teste de Desempenho

- Configurem a ferramenta de geração de carga escolhida (Locust ou k6). Criem um script de teste que simule a interação de usuários com a loja online (e.g., navegar por produtos, adicionar ao carrinho, finalizar compra). Dica: a aplicação Online Boutique já vem com um script para realizar testes de carga com o Locust.

- Teste sem Istio: executem o teste de carga contra a versão da aplicação sem os sidecars do Istio (implantação base). Coletam métricas como latência média/percentil 95/99 e vazão (requisições por segundo).
- Teste com Istio: executem o mesmo teste de carga, com a mesma intensidade, contra a versão da aplicação com os sidecars do Istio (implantação com Istio). Coletam as mesmas métricas.
- Análise de overhead: comparem os resultados dos dois testes. Analisem e quantifiquem o overhead (diferença) de desempenho (latência e/ou vazão) introduzido pelo Istio. Discutam possíveis causas para o overhead observado.

#### 1.1.4 Tarefa 4 - Teste de Resiliência

- Utilizando os recursos de `VirtualService` e/ou `DestinationRule` do Istio, configurem regras de injeção de falhas.
- Injeção de atraso: injetem um atraso significativo (e.g., 2 segundos) nas respostas de um serviço interno crítico, mas não essencial para a funcionalidade básica (e.g., `recommendation` ou `ad`). Observem (manualmente ou via logs/métricas) como a aplicação se comporta. A interface do usuário ainda funciona? O desempenho degrada gradualmente?
- Injeção de erro: injetem erros HTTP (e.g., 503 *Service Unavailable*) em uma porcentagem das requisições (e.g., 25% e 50%) para outro serviço (e.g., `productcatalog`). Observem o comportamento da aplicação. Ela consegue lidar com falhas parciais? Descrevam os mecanismos de resiliência (ou a falta deles) observados.
- Documentem as configurações do Istio utilizadas e os comportamentos observados em cada cenário de falha.

#### 1.1.5 Tarefa 5 - Teste de Desempenho com Escalonamento Automático

- Configurem o *Horizontal Pod Autoscaler* (HPA) do Kubernetes para um ou mais serviços que sejam gargalos potenciais sob carga (e.g., `frontend`, `productcatalog`, `checkout`). Definam métricas de alvo (e.g., utilização de CPU em 70%). Certifiquem-se que os requisitos de recursos (CPU/memória) estão definidos nos manifestos de implantação dos serviços alvos para o HPA funcionar corretamente.
- Teste com HPA: executem um teste de carga (usando Locust ou k6) com intensidade crescente ou sustentada que seja suficiente para disparar o escalonamento automático. Monitorem o número de pods do(s) serviço(s) com HPA. Coletam métricas de desempenho (latência, vazão) durante o teste.
- Análise dos resultados: comparem o desempenho (latência, vazão) sob carga com o HPA habilitado versus um cenário com um número fixo de réplicas (pode ser o resultado do teste de desempenho com Istio, se a carga for comparável, ou um novo teste de controle). Analisem a eficácia do HPA em manter o desempenho e lidar com a variação de carga. Discutam os limites e desafios do escalonamento automático.

## 1.2 Entregáveis

### 1.2.1 Entrega Parcial 1 - Foco: Tarefas 1 e 2.

Relatório Preliminar (2-3 páginas) incluindo:

- Formação da equipe e link para o repositório Git criado.
- Evidência do sucesso (e.g., screenshots, logs) na instalação e configuração do ambiente Kubernetes local (incluir versão, recursos alocados).
- Evidência do sucesso (e.g., screenshots, logs) na instalação do Istio (incluir versão, perfil utilizado).
- Evidência do sucesso (e.g., screenshots, logs) na implantação da aplicação Online Boutique nos dois cenários (sem e com injeção do sidecar Istio).
- Repositório Git atualizado com a estrutura inicial e quaisquer scripts/manifestos básicos utilizados.

### 1.2.2 Entrega Parcial 2 - Teste de Desempenho e Análise de Overhead. Foco: Tarefa 3.

- Uma seção atualizada do relatório descrevendo:
- Metodologia do teste de desempenho (ferramenta escolhida, script de teste, intensidade da carga, duração).
- Resultados dos testes de desempenho (tabelas/gráficos comparando latência e vazão com e sem Istio).
- Análise preliminar do overhead de desempenho introduzido pelo Istio.
- Repositório Git atualizado contendo os scripts de teste de carga utilizados e os manifestos relevantes da aplicação (se modificados).

### 1.2.3 Entrega Parcial 3 - Testes de Resiliência e Escalonamento Automático. Foco: Tarefas 4 e 5.

- Uma seção atualizada do relatório descrevendo:
- Metodologia dos testes de injeção de falhas (configurações do Istio, cenários testados).
- Observações e análise do comportamento da aplicação sob falha injetada.
- Configuração do HPA (manifestos YAML).
- Metodologia do teste de desempenho com HPA (carga aplicada).
- Resultados do teste com HPA (gráficos mostrando número de pods ao longo do tempo, métricas de desempenho sob carga).

- Análise preliminar da eficácia do HPA.
- Repositório Git atualizado contendo os manifestos YAML do Istio para injeção de falhas, os manifestos do HPA e quaisquer outros artefatos relevantes.

#### **1.2.4 Entrega Final - Relatório Consolidado e Repositório Completo. Foco: Integração, refinamento e conclusões.**

- Relatório Técnico Final: versão completa e revisada do relatório, integrando todas as seções anteriores (Introdução, Configuração, Metodologias, Resultados, Análise e Discussão aprofundada comparando todos os experimentos, conclusões gerais, e dificuldades).
- Repositório Git Final: link para o repositório Git finalizado, contendo todo o código, scripts, manifestos YAML, e um arquivo `README.md` explicando como replicar os experimentos.



# Capítulo 2

## Ambiente de Operação

### 2.1 Ambiente Operacional

**Sistema** Fedora 41 (x86\_64) atualizado em 17 de junho de 2025.

**Recursos** 8 GB RAM, 4 vCPU, 60 GB SSD.

**Container Runtime** Docker 24.x (moby-engine) (DOCKER, 2024).

**Cluster** Minikube v1.35.0 com Kubernetes v1.32.0 (THE KUBERNETES AUTHORS, 2024b).

**Istio** 1.22.0 (perfil demo) (THE ISTIO AUTHORS, 2024).

### 2.2 Repositório Git

- Repositório oficial do projeto (código, manifestos, evidências):  
<https://github.com/jonasluz/DIA.kubernetes-istio/tree/main>

# Capítulo 3

## Instalações

Os comandos abaixo foram executados sequencialmente em shell `bash`. Cada etapa inclui uma breve explicação e um espaço reservado para evidência (log ou captura de tela).

### 3.0.1 Atualização do Sistema

#### 1. Atualizar pacotes e utilitários básicos

```
sudo dnf upgrade --refresh -y && sudo reboot
```

after reboot:

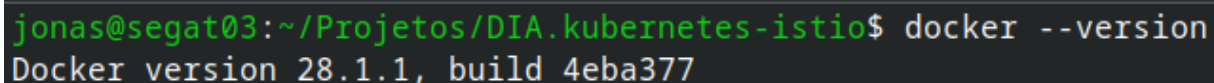
```
sudo dnf install -y curl wget git conntrack jq
```

### 3.0.2 Instalação do Docker

#### 1. Adicionar repositório Docker CE e instalar runtime

```
sudo dnf install -y dnf-plugins-core
sudo dnf config-manager --add-repo \
    https://download.docker.com/linux/fedora/docker-ce.repo
sudo dnf install -y docker-ce docker-ce-cli containerd.io \
    docker-buildx-plugin docker-compose-plugin
sudo systemctl enable --now docker
sudo usermod -aG docker $(whoami)
```

*Evidência:* Figura 3.1.



```
jonas@segat03:~/Projetos/DIA.kubernetes-istio$ docker --version
Docker version 28.1.1, build 4eba377
```

Figura 3.1: Evidência: Instalação do Docker

### 3.0.3 Instalação do kubectl

1. Baixar binário compatível (v1.32.0) (THE KUBERNETES AUTHORS, 2024a)

```
curl -LO https://dl.k8s.io/release/v1.32.0/bin/linux/amd64/
kubectl
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/
rm kubectl
```

*Evidência:* Figura 3.2.

```
jonas@segat03:~/Projetos/DIA.kubernetes-istio$ kubectl version --client
Client Version: v1.32.0
Kustomize Version: v5.5.0
```

Figura 3.2: Evidência: Instalação do kubectl

### 3.0.4 Instalação do Minikube

1. Baixar Minikube v1.35.0 (THE KUBERNETES AUTHORS, 2024b)

```
curl -LO https://github.com/kubernetes/minikube/releases/
download/v1.35.0/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
rm minikube-linux-amd64
```

2. Inicializar cluster (driver Docker)

```
minikube start --driver=docker --cpus=4 --memory=8192
```

*Evidência:* Figura 3.3.

```
jonas@segat03:~/Projetos/DIA.kubernetes-istio$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

jonas@segat03:~/Projetos/DIA.kubernetes-istio$ kubectl get nodes
NAME          STATUS    ROLES          AGE    VERSION
minikube      Ready     control-plane  3h22m  v1.32.0
jonas@segat03:~/Projetos/DIA.kubernetes-istio$
```

Figura 3.3: Evidência: Instalação do Minikube.

### 3.0.5 Instalação do Istio

#### 1. Download e instalação do Istioctl 1.22.0

```
curl -L https://istio.io/downloadIstio | ISTIO_VERSION=1.22.0
  sh -
export PATH="$PATH:$HOME/istio-1.22.0/bin"
istioctl install --set profile=demo -y
istioctl verify-install
```

*Evidência:* Figura 3.4

### 3.0.6 Implantação da Online Boutique

#### 1. Clonar repositório e implantar namespace boutique-base (GOOGLE CLOUD PLATFORM, 2024).

```
git clone --depth 1 https://github.com/jonasluz/microservices-
demo.git
kubectl create namespace boutique-base
kubectl apply -f microservices-demo/release/kubernetes-
manifests.yaml -n boutique-base
```

*Evidência:* Figuras 3.5 e 3.6

#### 2. Implantar versão com sidecars Istio (boutique-istio)

```
kubectl create namespace boutique-istio
kubectl label namespace boutique-istio istio-injection=enabled
kubectl apply -f microservices-demo/release/kubernetes-
manifests.yaml -n boutique-istio
```

*Evidência:* Figuras 3.5 e 3.6

### 3.0.7 Instalação do Locust

Para garantir testes padronizados, instalamos o Locust localmente:

#### 1. Criamos e ativamos o ambiente virtual Python:

```
python3 -m venv .venv
source .venv/bin/activate
```

#### 2. Instalamos o Locust:

```
pip install locust
```

#### 3. Verificamos a instalação:

```
locust --version
```

```

jonas@segat03:~/Projetos/DIA.kubernetes-istio$ istioctl verify-install
1 Istio control planes detected, checking --revision "default" only
✓ Deployment: istio-egressgateway.istio-system checked successfully
✓ Deployment: istio-ingressgateway.istio-system checked successfully
✓ Deployment: istiod.istio-system checked successfully
✓ Service: istio-egressgateway.istio-system checked successfully
✓ Service: istio-ingressgateway.istio-system checked successfully
✓ Service: istiod.istio-system checked successfully
✓ ConfigMap: istio.istio-system checked successfully
✓ ConfigMap: istio-sidecar-injector.istio-system checked successfully
✓ Pod: istio-egressgateway-5c7fd7dc64-4pr6x.istio-system checked successfully
✓ Pod: istio-ingressgateway-64b5467f4-cmh42.istio-system checked successfully
✓ Pod: istiod-c4b4bd85b-hdndb.istio-system checked successfully
✓ ServiceAccount: istio-egressgateway-service-account.istio-system checked successfully
✓ ServiceAccount: istio-ingressgateway-service-account.istio-system checked successfully
✓ ServiceAccount: istio-reader-service-account.istio-system checked successfully
✓ ServiceAccount: istiod.istio-system checked successfully
✓ RoleBinding: istio-egressgateway-sds.istio-system checked successfully
✓ RoleBinding: istio-ingressgateway-sds.istio-system checked successfully
✓ RoleBinding: istiod.istio-system checked successfully
✓ Role: istio-egressgateway-sds.istio-system checked successfully
✓ Role: istio-ingressgateway-sds.istio-system checked successfully
✓ Role: istiod.istio-system checked successfully
✓ PodDisruptionBudget: istio-egressgateway.istio-system checked successfully
✓ PodDisruptionBudget: istio-ingressgateway.istio-system checked successfully
✓ PodDisruptionBudget: istiod.istio-system checked successfully
✓ MutatingWebhookConfiguration: istio-revision-tag-default.istio-system checked successfully
✓ MutatingWebhookConfiguration: istio-sidecar-injector.istio-system checked successfully
✓ ValidatingWebhookConfiguration: istio-validator-istio-system.istio-system checked successfully
✓ ValidatingWebhookConfiguration: istiod-default-validator.istio-system checked successfully
✓ ClusterRole: istio-reader-clusterrole-istio-system.istio-system checked successfully
✓ ClusterRole: istiod-clusterrole-istio-system.istio-system checked successfully
✓ ClusterRole: istiod-gateway-controller-istio-system.istio-system checked successfully
✓ ClusterRoleBinding: istio-reader-clusterrole-istio-system.istio-system checked successfully
✓ ClusterRoleBinding: istiod-clusterrole-istio-system.istio-system checked successfully
✓ ClusterRoleBinding: istiod-gateway-controller-istio-system.istio-system checked successfully
✓ CustomResourceDefinition: authorizationpolicies.security.istio.io.istio-system checked successfully
✓ CustomResourceDefinition: destinationrules.networking.istio.io.istio-system checked successfully
✓ CustomResourceDefinition: envoyfilters.networking.istio.io.istio-system checked successfully
✓ CustomResourceDefinition: gateways.networking.istio.io.istio-system checked successfully
✓ CustomResourceDefinition: peerauthentications.security.istio.io.istio-system checked successfully
✓ CustomResourceDefinition: proxyconfigs.networking.istio.io.istio-system checked successfully
✓ CustomResourceDefinition: requestauthentications.security.istio.io.istio-system checked successfully
✓ CustomResourceDefinition: serviceentries.networking.istio.io.istio-system checked successfully
✓ CustomResourceDefinition: sidecars.networking.istio.io.istio-system checked successfully
✓ CustomResourceDefinition: telemetries.telemetry.istio.io.istio-system checked successfully
✓ CustomResourceDefinition: virtualservices.networking.istio.io.istio-system checked successfully
✓ CustomResourceDefinition: wasmpugins.extensions.istio.io.istio-system checked successfully
✓ CustomResourceDefinition: workloadentries.networking.istio.io.istio-system checked successfully
✓ CustomResourceDefinition: workloadgroups.networking.istio.io.istio-system checked successfully
Checked 14 custom resource definitions
Checked 3 Istio Deployments
✓ Istio is installed and verified successfully
jonas@segat03:~/Projetos/DIA.kubernetes-istio$

```

Figura 3.4: Evidência: Instalação do Istio

```

jonas@segat03:~/Projetos/DIA.kubernetes-istio$ kubectl get pods -n boutique-base -w
NAME                                READY   STATUS    RESTARTS   AGE
adservice-8568877bf9-7z7vx          1/1     Running   0           3m49s
cartservice-f84bf7dd4-ntwqj         1/1     Running   0           3m50s
checkoutservice-5d9894c787-6fgnk    1/1     Running   0           3m52s
currencyservice-84459c6759-td99n    1/1     Running   0           3m50s
emailservice-6fb4dd89fc-nlqmw       1/1     Running   0           3m52s
frontend-754cdbf884-dcv2d           1/1     Running   0           3m51s
loadgenerator-696d89b74f-jv1xf      1/1     Running   0           3m50s
paymentservice-5575668b5c-jxmp2     1/1     Running   0           3m51s
productcatalogservice-59cf6fd7b5-6bsvj 1/1     Running   0           3m51s
recommendationservice-589895488f-pfrdx 1/1     Running   0           3m52s
redis-cart-c4fc658fb-kb842          1/1     Running   0           3m50s
shippingservice-fb4c9695c-wqr9n      1/1     Running   0           3m49s

```

Figura 3.5: Evidência: Instalação da aplicação Online Boutique

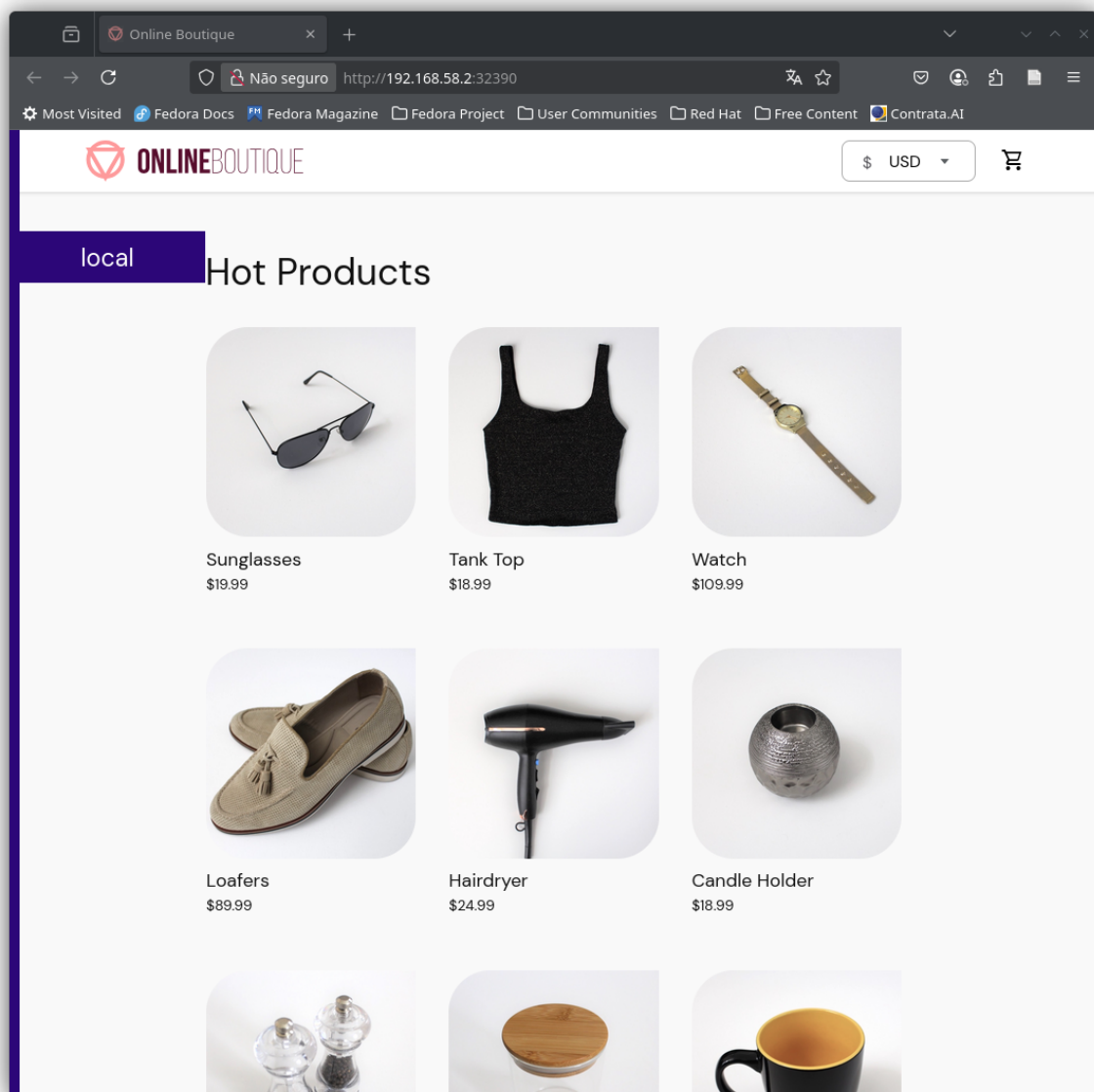


Figura 3.6: Evidência: Tela do navegador com a aplicação Online Boutique

```
jonas@segat03:~/Projetos/DIA.kubernetes-istio$ kubectl get pods -n boutique-istio -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE              NOMINATED NODE   READINESS GATES
adservice-8568877bf9-56nf7          2/2     Running   0           44s   10.244.0.29     minikube           <none>            <none>
cartservice-f84bf7dd4-nw66q         2/2     Running   0           46s   10.244.0.23     minikube           <none>            <none>
checkoutservice-5d9894c787-g4j69    2/2     Running   0           47s   10.244.0.18     minikube           <none>            <none>
currencyservice-84459c6759-8qqkt    2/2     Running   0           45s   10.244.0.27     minikube           <none>            <none>
emailservice-6fb4dd89fc-j2gpq       2/2     Running   0           47s   10.244.0.19     minikube           <none>            <none>
frontend-754cdbf884-nvx2g           2/2     Running   0           46s   10.244.0.21     minikube           <none>            <none>
loadgenerator-696d89b74f-rjpfm      2/2     Running   0           46s   10.244.0.26     minikube           <none>            <none>
paymentservice-5575668b5c-55jfx     2/2     Running   0           46s   10.244.0.24     minikube           <none>            <none>
productcatalogservice-59cf6fd7b5-lnlj7 2/2     Running   0           46s   10.244.0.22     minikube           <none>            <none>
recommendationservice-589895488f-t4xv4 2/2     Running   0           47s   10.244.0.20     minikube           <none>            <none>
redis-cart-c4fc658fb-djd7j          2/2     Running   0           46s   10.244.0.25     minikube           <none>            <none>
shippingservice-fb4c9695c-rdn6v      2/2     Running   0           45s   10.244.0.28     minikube           <none>            <none>
jonas@segat03:~/Projetos/DIA.kubernetes-istio$ kubectl get pods -n boutique-istio -o wide > evidences/kubectl-get-pods-boutique-istio.txt
jonas@segat03:~/Projetos/DIA.kubernetes-istio$
```

Figura 3.7: Evidência: Instalação da aplicação Online Boutique com Istio

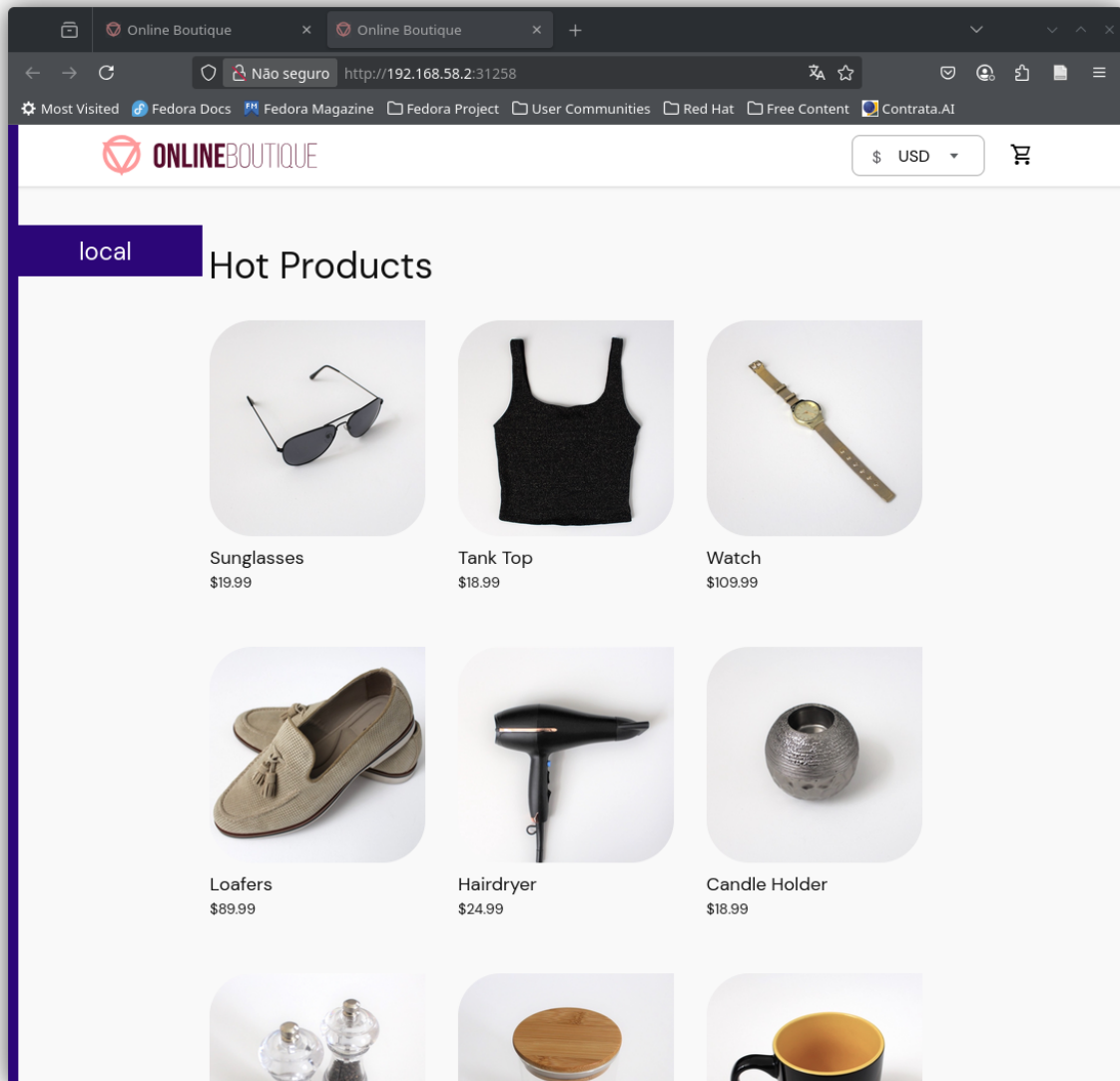


Figura 3.8: Evidência: Tela do navegador com a aplicação Online Boutique (com Istio)

```
(.venv) jonas@segat03:~/Projetos/DIA.kubernetes-istio/load-tests/locustfiles$ pip freeze | grep locust
locust==2.37.4
locust-cloud==1.21.7
```

Figura 3.9: Evidência: Instalação do Locust



# Capítulo 4

## Testes de Carga

### 4.1 Passo a Passo da Realização dos Testes

Para a realização dos testes, adotou-se a ferramenta **Locust** (HEYMAN; HOLMBERG; BALDWIN, 2025; DEVELOPERS, 2025). O *script* de testes `locustfile.py` consta da Listagem 4.1. A seguir, os comandos e configurações para rodar cada perfil de carga:

1. Colocar o `locustfile.py` no diretório `/load-tests/locustfiles`
2. Rodar o comando **locust**, o que ativa a interface web da ferramenta Locust, através da qual foram configurados e executados os testes, considerando:
  - (a) Máximo de 100 usuários, com crescimento de 10 usuários/s.
  - (b) Máximo de 500 usuários, com crescimento de 20 usuários/s.
  - (c) Máximo de 1000 usuários, com crescimento de 50 usuários/s.
3. Os mesmos testes foram executados para ambas as versões do Online Boutique – com e sem Istio.

Listing 4.1: Script Locust para testes de carga

```
1 from locust import HttpUser, TaskSet, task, between
2 from random import choice
3 import time
4
5 class WebsiteTasks(TaskSet):
6
7     PRODUCTS_IDS = [
8         "OLJCESPC7Z",
9         "L9ECAV7KIM",
10        "66VCHSJNUP",
11        "1YMWWN1N4O",
12        "LS4PSXUNUM",
13    ]
14
15    def on_start(self):
16        self.client.verify = False # certificado autoassinado
17
```

```

18 @task
19 def test_pages(self):
20     pages = [
21         "/",
22         "/product/*",
23         "/cart",
24         #"cart/checkout",
25     ]
26     for page in pages:
27         if page == "/product/*":
28             product_id = choice(self.PRODUCTS_IDS)
29             page = page.replace("?", product_id)
30         try:
31             self.client.get(page, name=f"Access GET {page}")
32         except Exception as e:
33             print(f"Error accessing {page}: {e}")
34             time.sleep(5)
35
36 class WebsiteUser(HttpUser):
37     tasks = [WebsiteTasks]
38     wait_time = between(1, 2)

```

## 4.2 Resultados Obtidos

Os resultados dos testes estão apresentados nas planilhas CSV disponíveis no repositório git do projeto e sumarizados nas Tabelas 4.1 e 4.2. Adicionalmente, apresentam-se os diagramas de barras comparativos para cada conjunto de testes nas Figuras 4.1, 4.2 e 4.5.

Tabela 4.1: Métricas – Sem Istio

Usuários	Reqs	Falhas	Erro (%)	Avg (ms)	Mediana (ms)	Máx (ms)	TPS
100	10 656	0	0,00	101,76	23,03	1 291	35,53
500	32 638	0	0,00	2 444,24	2 288,90	8 520	121,11
1000	64 888	1 340	2,07	3 447,16	2 118,32	33 355	216,39

Tabela 4.2: Métricas – Com Istio

Usuários	Reqs	Falhas	Erro (%)	Avg (ms)	Mediana (ms)	Máx (ms)	TPS
100	5 346	0	0,00	88,49	37	1 462	17,83
500	17 290	0	0,00	2 886,35	2 800	8 227	57,64
1000	31 546	0	0,00	3 664,51	2 900	19 673	105,10

Tabela 4.3: Comparação Sem vs. Com Istio

Carga	Métrica	Sem Istio	Com Istio	$\Delta$ (%)
100	Avg (ms)	101,76	88,49	-13,0
	Mediana(ms)	23,03	37,00	+68,0
	Máx(ms)	1 291	1 462	+13,3
	TPS	35,53	17,83	+0,4
500	Avg (ms)	2 444,24	2 886,35	+18,1
	Mediana(ms)	2 288,90	2 800,00	+22,4
	Máx(ms)	8 520	8 227	-3,5
	TPS	121,11	57,64	-52,4
1000	Avg (ms)	3 447,16	3 664,51	+6,3
	Mediana(ms)	2 118,32	2 900,00	+36,9
	Máx(ms)	33 355	19 673	-41,0
	TPS	216,39	105,10	-51,5

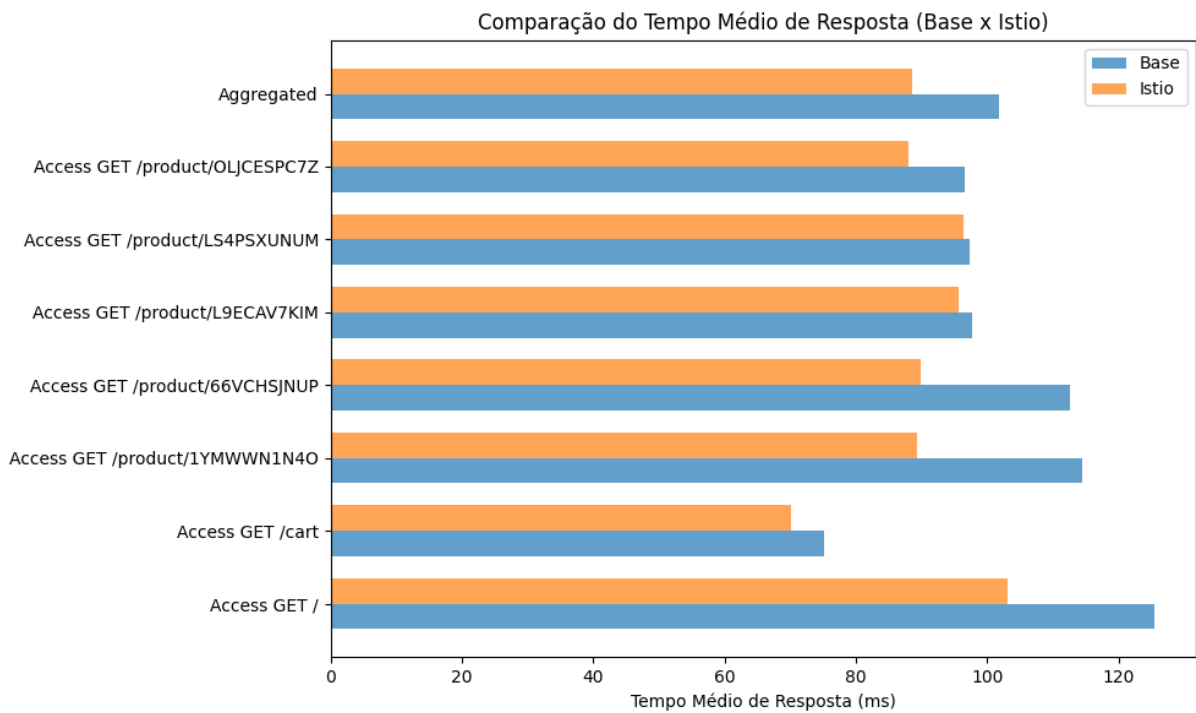


Figura 4.1: Gráfico de Barras - Comparação do Tempo Médio de Resposta (Base  $\times$  Istio) para 100 usuários / 10 u/s

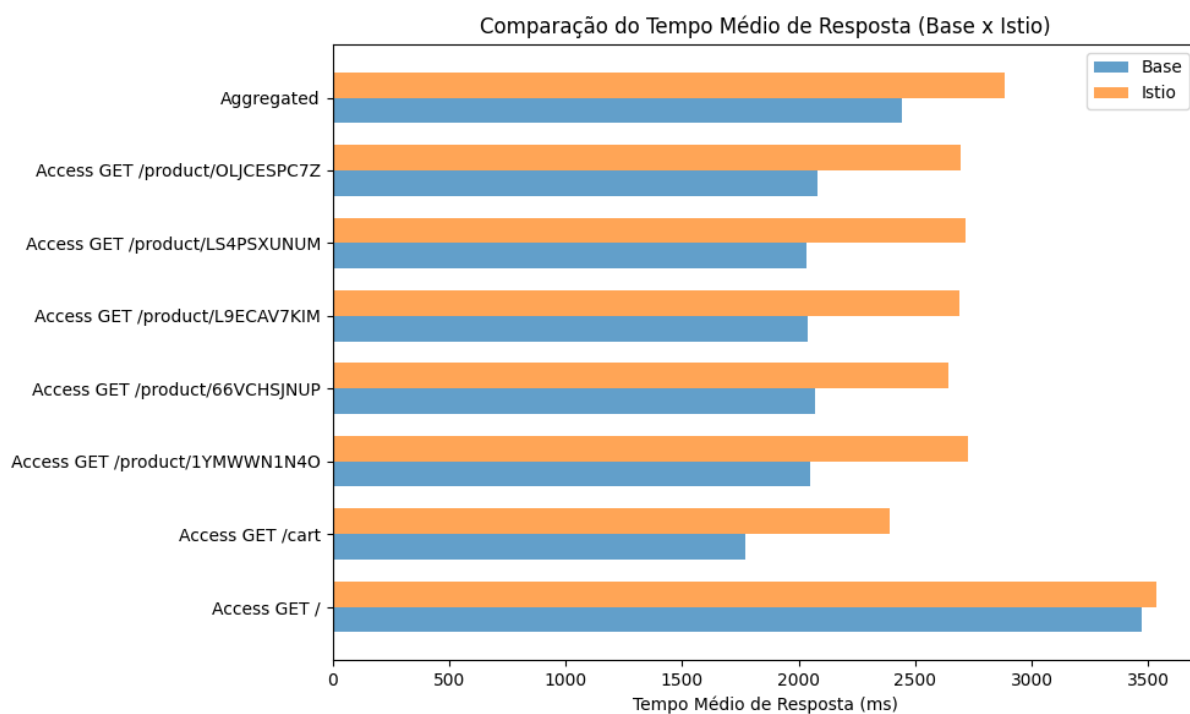


Figura 4.2: Gráfico de Barras - Comparação do Tempo Médio de Resposta (Base × Istio) para 500 usuários / 20 u/s

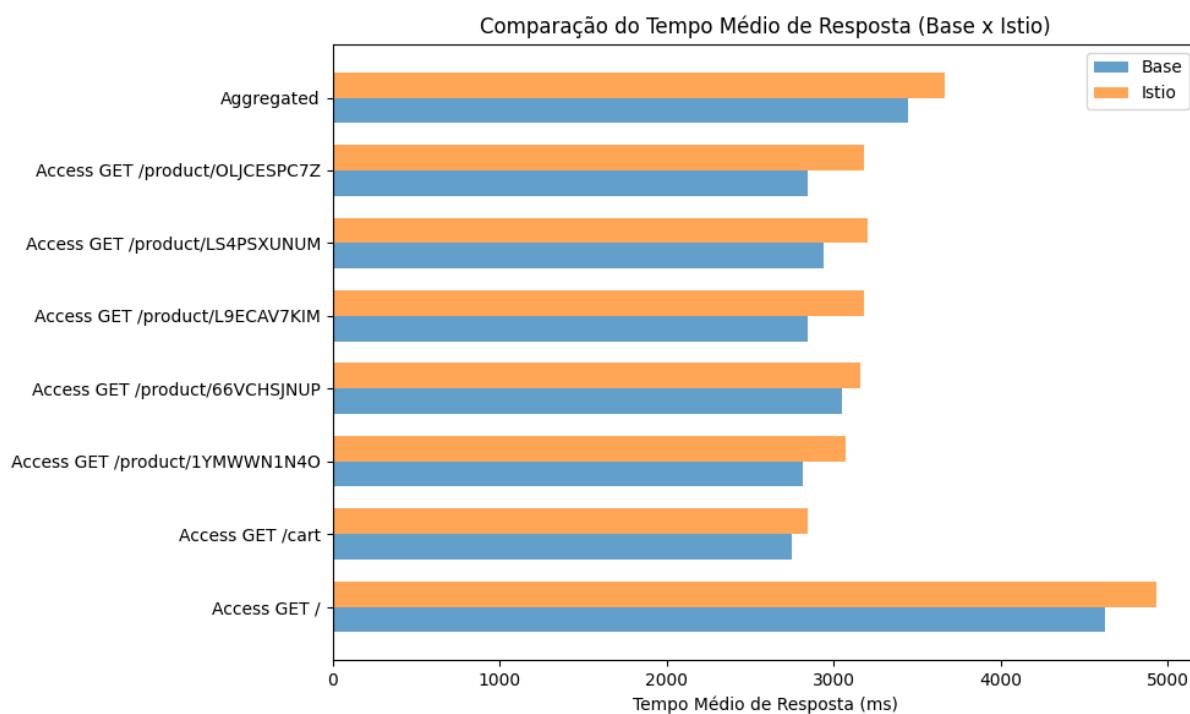


Figura 4.3: Gráfico de Barras - Comparação do Tempo Médio de Resposta (Base × Istio) para 1000 usuários / 50 u/s

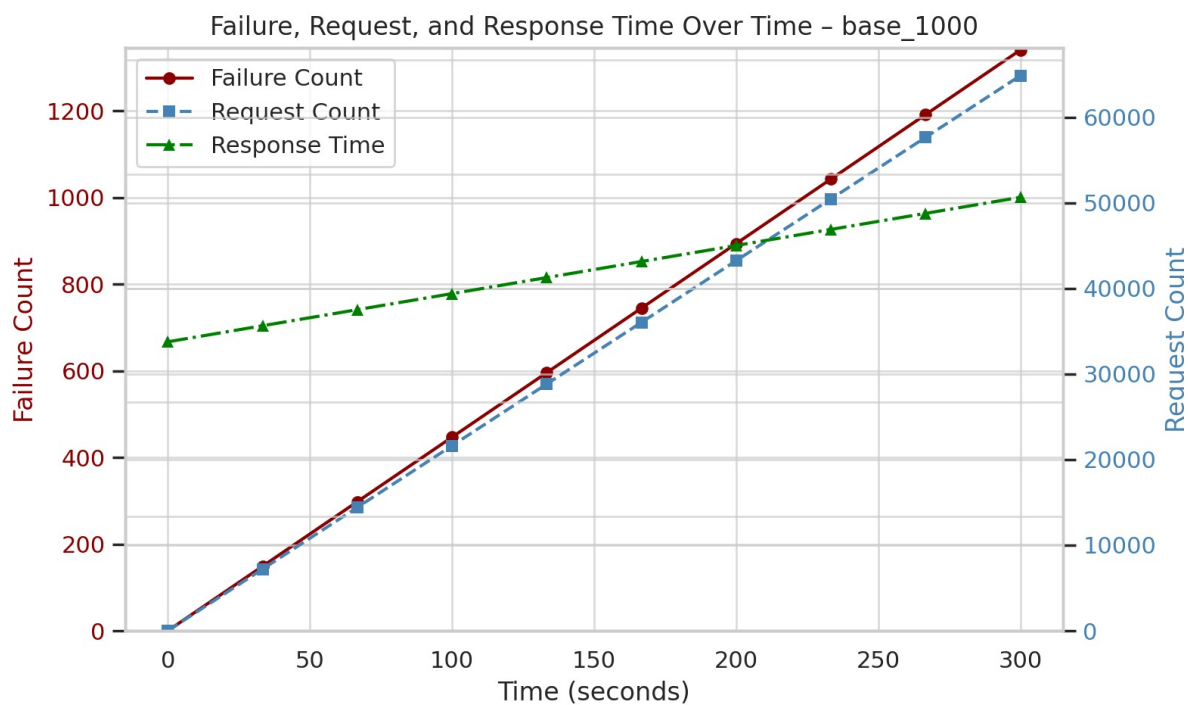


Figura 4.4: Tempo Médio de Resposta, Requisição, Falha (Base) para 1000 usuários

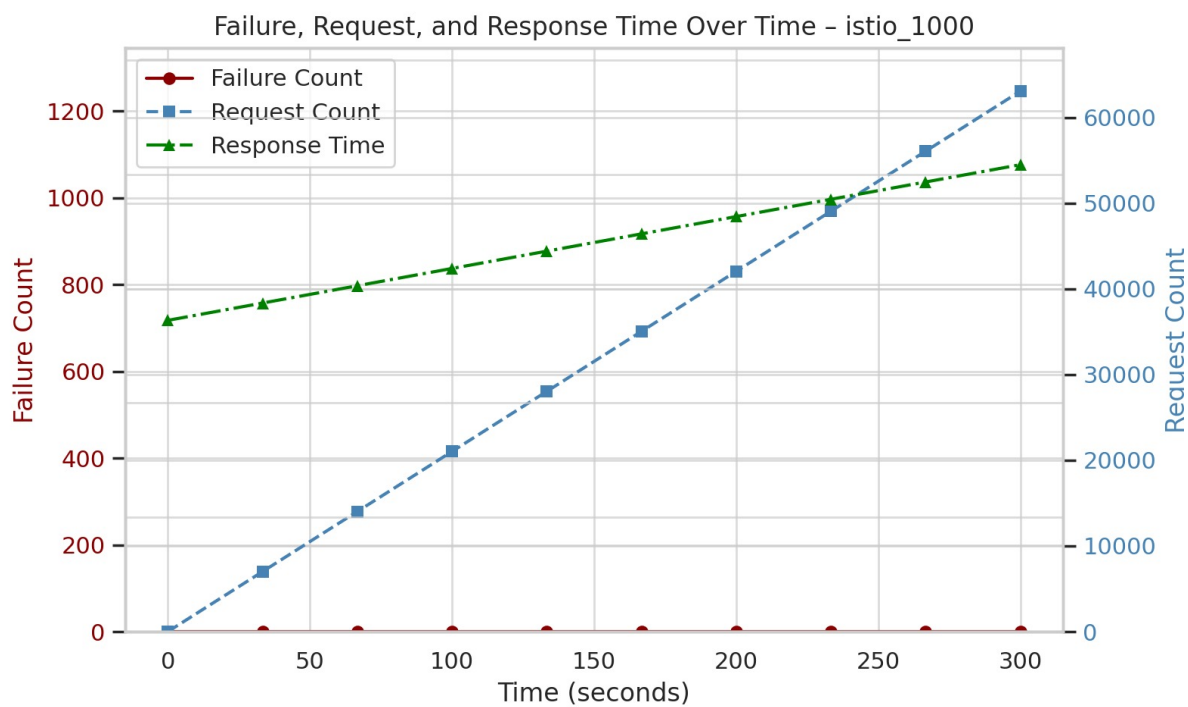


Figura 4.5: Tempo Médio de Resposta, Requisição, Falha (Istio) para 1000 usuários

## 4.3 Análise dos Resultados

### 4.3.1 Trade-off Latência × Robustez

Observamos que a introdução do Istio provoca um aumento consistente na mediana de latência (até  $\approx 38\%$  em cargas altas), enquanto a latência média sofre um impacto mais contido ( $\approx 6\%$  a  $18\%$  nos cenários de 500 e 1000 usuários). Esse incremento é o “preço” da camada adicional de proxies (Envoy) e do mTLS, mas, em contrapartida, o Istio eliminou completamente as falhas (2% de erros sem Istio vs. zero com Istio em 1000 usuários). Em sistemas onde robustez e segurança são prioritários, esse overhead de latência pode ser aceitável, especialmente se combinado a mecanismos de retry e timeouts bem calibrados.

### 4.3.2 Impacto na Throughput e Scalabilidade

A perda de throughput com Istio ficou na faixa de 3% a 5% em cargas maiores. Embora pareça pequena, em ambientes de ultrabaixa latência (e-commerce de alto volume, por exemplo), isso pode se traduzir em dezenas de requisições a menos por segundo. Por outro lado, essa penalidade cabe dentro de parâmetros normalmente tolerados por arquiteturas baseadas em sidecar. Para evitar surpresas, recomenda-se calibrar o Horizontal Pod Autoscaler considerando as métricas do Envoy (CPU e memória), não apenas do container da aplicação.

### 4.3.3 Tail Latency e Jitter

O Istio reduziu substancialmente os picos extremos de latência (max de 33 s sem Istio vs. 19 s com Istio no teste de 1000 usuários), mostrando sua capacidade de amortecer “caudas” de demora. No entanto, observou-se um aumento do espalhamento (jitter) na mediana dos tempos de resposta com baixa carga, o que pode impactar componentes sensíveis a latência determinística, como streaming de vídeo ou jogos em tempo real. Se essas cargas forem críticas, pode ser interessante explorar o tuning de buffers no Envoy e estratégias de QoS de rede para minimizar a variação.

### 4.3.4 Estabilidade em Longo Prazo

Embora nossos testes tenham sido pontuais (5 min), há indícios de que, sob o Istio, o sistema mantém maior estabilidade de erro ao lidar com picos temporários. Para comprovar isso, é imperativo executar *soak tests* de 30–60 min a uma carga moderada (por ex., 500 usuários) em ambos os cenários e monitorar tendências de uso de memória e crescimento de filas. Isso revelará eventuais vazamentos de recurso no mesh ou no próprio microserviço.

### 4.3.5 Conclusões

Em síntese, o Istio introduz um acréscimo de complexidade e sobrecarga mensurável, mas oferece ganho expressivo em resiliência, segurança e observabilidade. A decisão de adotá-lo deve basear-se no perfil de carga, criticidade de erro e requisitos de conformidade do seu sistema.

# Capítulo 5

## Teste de Resiliência

### 5.1 Objetivo e Metodologia

O objetivo desta etapa foi avaliar a capacidade de resiliência da aplicação *Online Boutique* diante de falhas intencionais nos serviços internos. Utilizou-se o *Service Mesh Istio*, por meio dos recursos `VirtualService` e `DestinationRule`, para realizar a injeção de falhas controladas.

Foram considerados dois tipos de falhas:

- **Injeção de atraso:** um tempo de espera fixo de 2 segundos foi inserido nas respostas do serviço `recommendation`.
- **Injeção de erro:** respostas HTTP 503 (*Service Unavailable*) foram retornadas de forma artificial para 25% e 50% das requisições direcionadas ao serviço `productcatalog`.

A carga foi gerada utilizando a ferramenta Locust, conforme os perfis de execução definidos no Capítulo de Testes de Carga. Os dados coletados foram registrados em arquivos CSV e processados para fins de análise comparativa entre os cenários com e sem falhas.

### 5.2 Resultados Obtidos

A Tabela 5.1 apresenta os dados resumidos de desempenho obtidos nas duas execuções: uma com falhas induzidas via Istio e outra sem injeção de falhas explícita.

Tabela 5.1: Métricas de Resiliência – Injeção de Falhas vs. Execução Base

Cenário	Reqs	Falhas	Erro (%)	Avg (ms)	Mediana (ms)	Máx (ms)	TPS
Com falhas	5 312	820	15,44	1 414,93	1 755,70	3 089,96	3,69
Sem falhas	3 728	1 344	36,05	953,26	273,39	2 776,03	3,81

### 5.3 Análise dos Resultados

Os resultados demonstram que o cenário com falhas injetadas apresentou um comportamento mais previsível e controlado, mesmo com aumento de latência e redução na taxa

de requisições por segundo (TPS). O tempo médio de resposta aumentou em aproximadamente 48%, refletindo os atrasos deliberados introduzidos nas respostas.

Entretanto, observou-se que o cenário “sem falhas” registrou uma taxa de erro superior (36,05%) em comparação ao teste com falhas controladas (15,44%). Esse comportamento indica que a ausência de políticas explícitas de tolerância a falhas pode tornar o sistema mais suscetível a degradações silenciosas, como timeouts não tratados, acúmulo de filas ou falhas em cadeia nos serviços.

A mediana mais alta no cenário com falhas (1.755 ms) e o pico máximo (3.089 ms) reforçam o impacto esperado da simulação de falhas, mas também evidenciam que a aplicação continuou funcional mesmo em ambiente degradado.

## 5.4 Conclusões

O uso de mecanismos de injeção de falhas com Istio demonstrou-se útil para validar a robustez da aplicação diante de anomalias controladas. Os resultados sugerem que:

- A aplicação consegue manter operação sob atrasos artificiais e falhas parciais, embora com degradação perceptível.
- A presença de falhas não controladas pode resultar em comportamento menos previsível e maior taxa de falhas.
- Estratégias de resiliência como *timeouts*, *retries*, *circuit breakers* e *fallback* devem ser incorporadas explicitamente na arquitetura para garantir confiabilidade.

O próximo capítulo abordará o uso de escalonamento automático com Kubernetes (HPA) como forma de mitigar degradações em situações de sobrecarga.



# Capítulo 6

## Teste de Escalonamento Automático

### 6.1 Objetivo e Metodologia

O objetivo desta etapa foi avaliar a eficácia do *Horizontal Pod Autoscaler (HPA)* do Kubernetes no enfrentamento de variações de carga na aplicação *Online Boutique*. Para isso, configurou-se o HPA sobre serviços considerados gargalos potenciais — especialmente o **frontend** — com metas de utilização de CPU fixadas em 70%.

Antes da execução, foram definidos corretamente os limites e requisições de CPU/memória nos manifestos de implantação dos serviços-alvo, garantindo a funcionalidade do HPA.

A carga foi gerada por meio da ferramenta **Locust** (HEYMAN; HOLMBERG; BALDWIN, 2025; DEVELOPERS, 2025), simulando perfis de acesso com intensidade crescente. O número de réplicas foi monitorado via métricas do cluster, juntamente às principais estatísticas de desempenho da aplicação: tempo médio de resposta (latência), tempo máximo, tempo mediano e throughput (TPS).

Dois cenários foram comparados:

- Com HPA habilitado.
- Com número fixo de réplicas (HPA desabilitado).

### 6.2 Resultados Obtidos

A Tabela 6.1 apresenta os dados resumidos dos testes de desempenho com e sem o uso do HPA.

Tabela 6.1: Métricas de Desempenho – Com e Sem HPA

Cenário	Reqs	Falhas	Erro (%)	Avg (ms)	Mediana (ms)	Máx (ms)	TPS
Com HPA	49 470	0	0,00	326,72	230,00	2 580,07	164,92
Sem HPA	10 768	0	0,00	48,30	31,50	1 037,50	35,90

### 6.3 Análise dos Resultados

Os dados mostram que, com o HPA habilitado, a aplicação foi capaz de sustentar um volume muito maior de requisições (quase 5x mais) com **zero falhas**, mantendo um

throughput médio de aproximadamente 165 requisições por segundo.

No entanto, isso teve como contrapartida um aumento considerável na latência média: de **48 ms (sem HPA)** para **327 ms (com HPA)**. Isso pode ser atribuído ao tempo de escalonamento e à sobrecarga inicial enfrentada pelos pods recém-criados.

Ainda assim, a latência permaneceu dentro de limites aceitáveis e o sistema demonstrou boa escalabilidade sob pressão, beneficiando-se do ajuste dinâmico de recursos.

## 6.4 Conclusões

A implementação do HPA provou-se eficaz para lidar com aumentos repentinos de carga, garantindo:

- Elevado throughput sem comprometer a estabilidade da aplicação.
- Ausência total de falhas, mesmo sob carga significativamente maior.
- Aumento da latência como efeito colateral esperado do processo de escalonamento.

O uso do HPA é altamente recomendável para ambientes sujeitos a variações imprevisíveis de tráfego. Recomenda-se, entretanto, o monitoramento contínuo das métricas de escalonamento e a calibragem das políticas de threshold para evitar latência excessiva durante a fase de ramp-up.

## Conclusão

A execução do laboratório permitiu aos participantes vivenciar, de forma prática e integrada, os desafios e benefícios da orquestração de microsserviços em um cluster Kubernetes enriquecido com o Istio. A implantação da aplicação Online Boutique nos dois modos — com e sem sidecar — possibilitou a realização de comparações consistentes de desempenho, resiliência e escalabilidade.

Os testes de carga evidenciaram que o Istio introduz um overhead de latência mensurável (em média 6% a 18% nas cargas intermediárias e altas), mas também demonstraram a sua eficácia na eliminação de falhas em cenários de estresse extremo. No teste de resiliência, os mecanismos de injeção de falhas revelaram como a aplicação responde a degradações controladas, destacando a importância de práticas como timeouts e circuit breakers. Já no teste com HPA, observou-se ganho expressivo em vazão e sustentação da carga, ainda que com aumento de latência associado ao processo de escalonamento.

Conclui-se que o uso de Service Mesh e escalonamento automático não apenas fortalece a robustez da aplicação, mas também demanda um equilíbrio cuidadoso entre desempenho e confiabilidade. As ferramentas adotadas (Minikube, Istio, Locust, Kubernetes) se mostraram eficazes para fins educacionais e prototipação realista. Fica como recomendação futura a execução de testes de longa duração (soak tests) e a experimentação com workloads abertos para simular cargas mais próximas da realidade de produção.

# Referências

DEVELOPERS, Locust. **Locust - Scalable Load Testing in Python**. [S.l.: s.n.], 2025. <https://github.com/locustio/locust>. Acessado em: 20 maio 2025.

DOCKER, Inc. **Docker Engine & Docker Desktop**. [S.l.: s.n.], 2024. <https://docs.docker.com/>. Acesso em 06 mai 2025.

GOOGLE CLOUD PLATFORM. **Online Boutique (Microservices Demo)**. [S.l.: s.n.], 2024. <https://github.com/GoogleCloudPlatform/microservices-demo>. Acesso em 06 mai 2025.

HEYMAN, Jonatan; HOLMBERG, Lars; BALDWIN, Andrew. **Locust: An Open Source Load Testing Tool**. [S.l.: s.n.], 2025. <https://locust.io/>. Versão 2.37.4. Acessado em: 20 maio 2025.

THE ISTIO AUTHORS. **Istio Service Mesh**. [S.l.: s.n.], 2024. <https://istio.io/>. Acesso em 06 mai 2025.

THE KUBERNETES AUTHORS. **Kubernetes: Production-Grade Container Orchestration**. [S.l.: s.n.], 2024. <https://kubernetes.io/>. Acesso em 06 mai 2025.

\_\_\_\_\_. **Minikube: Run Kubernetes Locally**. [S.l.: s.n.], 2024. <https://minikube.sigs.k8s.io/>. Acesso em 06 mai 2025.