Análise do Uso de Configuração do Plano de Controle de um Service Mesh Para Tarefas de Gerência de Rede

Haroldo R. S. Silva[1]**,Pedro Jacobus**[1]**, Rafael Trevisan**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

Abstract. *This article addresses common problems encountered in network management and presents solutions based on the Istio Service Mesh. Throughout this article, the categories of Failures and Configuration from the FCAPS model will be discussed. Two solutions are described for handling unresponsive services. The first solution involves configuring a timeout to terminate the pending connection. The second solution uses load balancing to evict unresponsive pods in case backups exist. These solutions were implemented in a Kubernetes cluster using Minikube and Istio's demo profile. The implementation of these measures was able to maintain service availability despite injected failures.*

Resumo. *Este artigo aborda problemas comuns encontrados em gerência de redes e apresenta soluções baseadas no Service Mesh Istio. Durante este artigo serão abordadas as categorias Falhas e Configuração do modelo FCAPS. São descritas duas soluções para o tratamento de serviços não responsivos. A primeira solução consiste na configuração de um time-out para terminar a conexão pendente. A segunda solução se utiliza do balanceamento de carga para ejetar pods não responsivos caso existam backups. Essas soluções foram implementadas em um cluster Kubernetes utilizando minikube e o perfil demo do Istio. A implementação dessas medidas foi capaz de manter a disponibilidade dos serviços apesar de falhas injetadas.*

1. Introdução

Com o aumento da complexidade dos serviços prestados através da internet, a implantação e manutenção deles se torna cada vez mais custosa e complexa. O Kubernetes é uma ferramenta de orquestração de containers utilizada para automatizar e simplificar este processo. Desde o seu lançamento em 2014, o Kubernetes teve as suas funcionalidades estendidas através da integração de outras ferramentas como o Istio.

O Istio cria uma rede configurável e ciente de aplicação que pode ser utilizada para telemetria, segurança e maior controle sobre o tráfego em aplicações de larga escala. Para implementar a *Service Mesh*, o Istio utiliza os *Envoy Proxies* para realizar o roteamento e a comunicação entre serviços. Estes elementos podem ser configurados para impedir e tratar erros, com o uso de *time-outs* e do *circuit-break*. Este trabalho tem como objetivo utilizar estes recursos para o tratamento de erros. O restante deste trabalho está organizado da seguinte forma: A seção 2 discute a metodologia utilizada, abordando o *setup* utilizado, as configurações que foram realizadas e quais falhas foram injetadas para testar o comportamento da rede. A seção 3 demonstra e discute os resultados obtidos, e a seção 4 apresenta as conclusões desse trabalho.

2. Metodologia

2.1. Setup

Para realizar essa tarefa, foi necessária a configuração de um setup utilizando o Minikube, o Istio e o Kiali.

2.1.1. Minikube

Com a finalidade de executar clusters Kubernets locais na máquina virtual, foi necessária a instalação e configuração do minikube, juntamente com a instalação e configuração do Kubectl, a fim de permitir a interação com estes clusters via linha de comando. Importante ressaltar o uso do Docker para a criação dos clusters.

2.1.2. Istio

Foi instalado o perfil demo do Istio para que as configurações base do Istio interfiram de uma maneira esperada. Para isso, foi escolhido o samples/bookinfo [Istio a], configurando-se os pods *productPage*, *reviews* v1, v2 e v3, *ratings* e *details*. Além disso, a fim de se implementar o circuit break, uma das soluções que escolhemos para nosso trabalho, foi necessária a configuração do samples/httpbin contendo os pods httpbin e fortio.

2.1.3. Kiali

Para se ter acesso a certas métricas, health check dos serviços e análise de tráfego dos mesmos, foi utilizado o Kiali. Como consequência, foi necessária a instalação e configuração do Prometheus, indispensável para o funcionamento do primeiro.

2.2. Injeção de Falhas

Para testar a resiliência de serviços, é possível injetar falhas através de configurações a nível de plano de controle com serviços virtuais do Istio [Istio c]. Através dessa funcionalidade é possível forçar diversos comportamentos relacionados a possíveis erros em dependências ou na própria rede para verificar como os serviços disponíves, ou a rede como um todo, reagem a tais comportamentos.

Para demonstrar essa funcionalidade foi injetada uma falha no pod *ratings* que gera um *delay* de 30 segundos.

```
http:
- fault:
    delay:
      percent: 100
      fixedDelay: 30s
  route:
  - destination:
      host: ratings
      subset: v1
```

Dessa forma simulamos que o serviço não está respondendo em tempo hábil, para que possamos verificar como os demais serviços da rede (neste caso os serviços que nos interessam são aqueles que interagem com *ratings*) reagem a essa situação de falha. A figura 1 demonstra como uma aplicação responde quando houve uma implementação de *time-out* no próprio serviço.

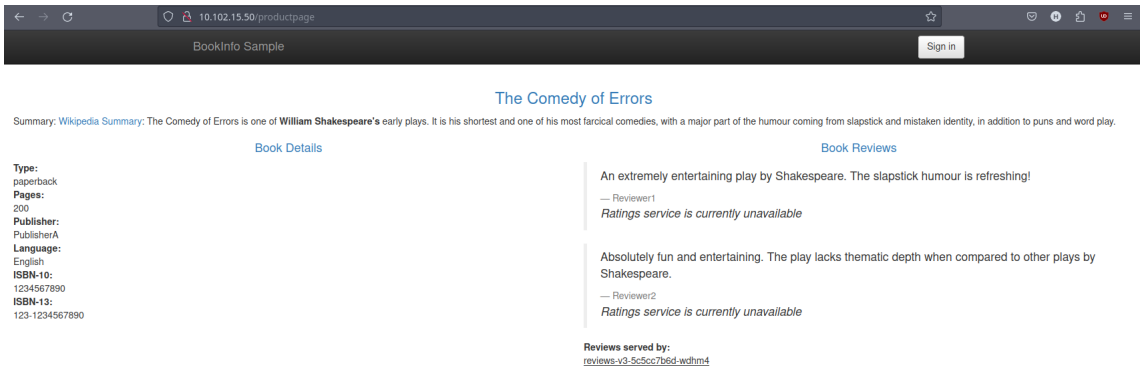

Figura 1. Reação da versão 3 de *reviews* à injeção da falha

Nesse caso o *time-out* de *reviews-v3* é menor que o *time-out* de *productpage*, dessa forma é possível ver as avaliações dos usuários, mesmo que o serviço de notas esteja indisponível.

Já a figura 2 demonstra como um serviço que não possui implementação interna de *time-out* influencia no funcionamento geral da aplicação,

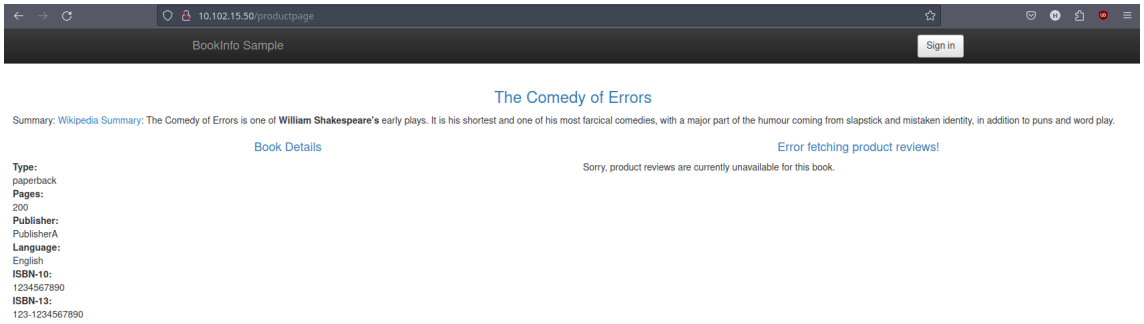

Figura 2. Reação da versão 2 de *reviews* à injeção da falha

Podemos observar que, apesar do bom funcionamento do serviço de avaliações, a aplicação perde a sua funcionalidade, causando uma propagação indireta do erro. Essa situação poderia ser ainda pior caso o serviço base para a página (*productpage*) também não implementasse um *time-out*, visto que a página ficaria indisponível na sua totalidade (ou demoraria o suficiente para que o usuário desistisse do acesso) apsesar de três dos quatro serviços estarem funcionando corretamente.

2.3. Configurações

Na seção 2.2 observamos que o comportamento do serviço *reviews-v2* é indesejado, pois propaga o erro do serviço *ratings*, e que o serviço *reviews-v3* demonstra o comportamento desejado nessa situação.

Através do Istio, podemos mitigar o problema simulado sem que seja necessário alterar a implementação do serviço. Para fazê-lo, basta adicionar uma configuração de *time-out* [Istio d] no roteamento do serviço chamado para que o *time-out* interno de *productpage* não seja disparado.

```
http:
  - route:
    - destination:
        host: ratings
        subset: v1
    timeout: 0.002s
```

Observe que, para propósitos demonstrativos, o valor de *time-out* usado é exageradamente baixo para que ele entre em efeito, visto que essa configuração não é compatível diretamente com a injeção de falha de *delay*.

A configuração feita impede a propagação de um erro entre serviços, mas existem maneiras de tratar a causa desse possível erro através de configurações de *circuit breaking* [Istio b]. Para demonstrar essa funcionalidade foi utilizado o serviço *httpbin* configurado com *circuit breaking*.

```
trafficPolicy:
  connectionPool:
    tcp:
      maxConnections: 1
    http:
      http1MaxPendingRequests: 1
      maxRequestsPerConnection: 1
  outlierDetection:
    consecutive5xxErrors: 1
    interval: 1s
    baseEjectionTime: 3m
    maxEjectionPercent: 100
```

As configurações em *connectionPool* indicam que o serviço pode receber apenas uma requisição antes que o cliente precise reestabelecer a conexão, que existe uma fila de tamanho 1 para requisições pendentes e que as seguintes requisições serão rejeitadas. Se, mesmo com essas condições, o serviço não conseguir lidar com o volume de requisições, a configuração *outlierDetection* impede que usuários descubram o serviço por determinado tempo, o que pode ser utilizado juntamente com réplicas do serviço para evitar longas esperas por respostas.

3. Resultados

Com as configurações é possível observar os comportamentos da rede.

A imagem 3 indica que apesar do problema no serviço *ratings*, usuário ainda é capaz de acessar os outros serviços, mesmo quando o *reviews-v2* é utilizado, devido às configurações utilizadas, não compromentendo a disponibilidade dos serviços que não apresentam erros.

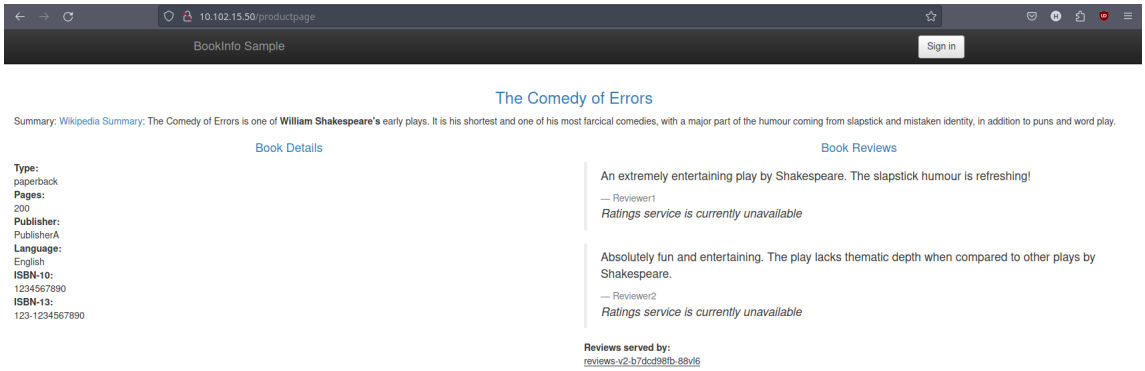

Figura 3. Como cliente é possível acessar *reviews* apesar do erro

```
Connection time (s) : count 21 avg 7.0891524e-05 +/- 5.273e-05 min 4.0646e-05 max 0.000286079 sum 0.001488722
Sockets used: 21 (for perfect keepalive, would be 3)
Uniform: false, Jitter: false, Catchup allowed: true
IP addresses distribution:
10.105.116.183:8000: 21
Code 200 : 11 (36.7 %)
Code 503 : 19 (63.3 %)
Response Header Sizes : count 30 avg 84.333333 +/- 110.8 min 0 max 230 sum 2530
Response Body/Total Sizes : count 30 avg 454.76667 +/- 280.9 min 241 max 824 sum 13643
All done 30 calls (plus 0 warmup) 1.469 ms avg, 1365.9 qps
```

Figura 4. Descrição de requisições realizadas pelo *fortio*

Para testes do *circuit break* foi utilizado o serviço *fortio* para gerar uma carga de requisições em várias conexões no serviço *httpbin*.

A imagem 4 mostra que a configuração feita no serviço *httpbin* causou a rejeição de 63.3% das requisições, demonstrando que o *circuit break* funcionou, mas que existe certa variação em como as regras definidas são aplicadas.

4. Conclusão

Este trabalho teve como objetivo implementar uma Mesh de serviço do istio e configurá-la em nível de rede para tratar erros. Um erro foi injetado no pod *ratings*, e duas configurações foram testadas para resolvê-lo, uma se utilizando de *time-out* e outra usando o *circuit break*. Para ambas, as configurações foram capazes de aumentar a robustez da rede, sem alterações nos serviços que a compõe.

Referências

Istio. Bookinfo application. `https://istio.io/latest/docs/examples/bookinfo/`. [Online; acesso em 07-dezembro-2023].

Istio. Circuit breaking. `https://istio.io/latest/docs/tasks/traffic-management/circuit-breaking/`. [Online; acesso em 07-dezembro-2023].

Istio. Fault injection. `https://istio.io/latest/docs/tasks/traffic-management/fault-injection/`. [Online; acesso em 07-dezembro-2023].

Istio. Request timeouts. `https://istio.io/latest/docs/tasks/traffic-management/request-timeouts/`. [Online; acesso em 07-dezembro-2023].