Zarządzanie Systemami Rozproszonymi
Laboratoria z Service Mesg i Istio

mgr inż. Jakub Woźniak

1 Wprowadzenie

1.1 Co to jest Service Mesh?

Service Mesh to dedykowana warstwa infrastruktury, która zarządza komunikacją między mikroserwisami w aplikacjach rozproszonych. Umożliwia dodanie zaawansowanych funkcji bez modyfikacji kodu aplikacji.
Podstawowe funkcje Service Mesh:

- **Routing ruchu:** Określanie, jak ruch sieciowy jest kierowany między mikroserwisami.
- **Obserwowalność:** Zbieranie metryk, logów i tras przepływu ruchu w aplikacjach.
- **Bezpieczeństwo:** Szyfrowanie ruchu między usługami (TLS), autoryzacja i uwierzytelnianie.
- **Zarządzanie błędami:** Mechanizmy takie jak *Circuit Breaking* i *Fault Injection*, które zwiększają odporność systemu na awarie.

1.2 Dlaczego Service Mesh jest potrzebny?

W tradycyjnych monolitycznych aplikacjach komunikacja wewnętrzna była zarządzana w obrębie jednego procesu. W przypadku aplikacji mikroserwisowych, które działają jako niezależne komponenty, komunikacja między nimi wymaga zaawansowanego zarządzania.
Korzyści z Service Mesh:

- **Spójność:** Ujednolicone zarządzanie ruchem między usługami.
- **Niezależność:** Funkcje takie jak routing czy bezpieczeństwo są niezależne od kodu aplikacji.
- **Skalowalność:** Automatyczne równoważenie obciążenia oraz optymalizacja ruchu.
- **Odporność:** Mechanizmy zapewniające stabilność systemu w przypadku awarii części usług.

1.3 Wprowadzenie do Istio

Istio to jedno z najpopularniejszych narzędzi implementujących Service Mesh. Jest w pełni zgodne z Kubernetes i zapewnia zaawansowane funkcje zarządzania komunikacją między usługami.
Główne komponenty Istio:

- **Pilot:** Zarządza konfiguracją ruchu w klastrze.
- **Envoy:** Proxy działające w każdej usłudze, obsługujące ruch między mikroserwisami.
- **Mixer:** Zbieranie metryk i egzekwowanie polityk.
- **Citadel:** Obsługa uwierzytelniania i autoryzacji (m.in. TLS).
- **Ingress/Egress Gateway:** Zarządzanie ruchem przychodzącym i wychodzącym z klastra.

1.4 Architektura Istio

Architektura Istio opiera się na modelu proxy-sidecar, gdzie każda usługa w klastrze ma dedykowany proxy (Envoy) do obsługi ruchu.
Kluczowe elementy architektury:

- **Data Plane:** Odpowiada za ruch sieciowy między usługami. Składa się z proxy Envoy wstrzykniętych jako *sidecar* w każdym Podzie.
- **Control Plane:** Zarządza konfiguracją Data Plane oraz politykami ruchu i bezpieczeństwa.

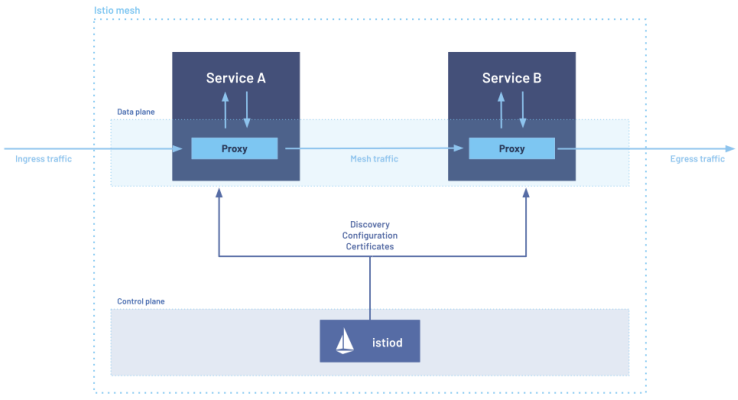

Figure 1: Architektura Istio. Źródło: dokumentacja Istio.

Istio ułatwia zarządzanie złożonymi systemami mikroserwisowymi, zapewniając niezbędne funkcje do skalowalnych, bezpiecznych i wydajnych wdrożeń aplikacji w środowiskach produkcyjnych.

2 Przygotowanie środowiska

W tej sekcji studenci przygotują środowisko do pracy z Istio, instalując wymagane narzędzia oraz konfigurując klaster Minikube.

2.1 Wymagane narzędzia

Przed rozpoczęciem ćwiczeń należy upewnić się, że na komputerze są zainstalowane następujące narzędzia:

- **Minikube:** Klaster Kubernetes działający lokalnie. Instalację Minikube można przeprowadzić zgodnie z oficjalną dokumentacją: `https://minikube.sigs.k8s.io/docs/start/`.
- **kubectl:** Narzędzie CLI do zarządzania klastrem Kubernetes. Instrukcje instalacji znajdują się na stronie: `https://kubernetes.io/docs/tasks/tools/install-kubectl/`.
- **Istio CLI:** Narzędzie do instalacji i zarządzania Istio. Można je pobrać poleceniem:

```
curl -L https://istio.io/downloadIstio | sh -
cd istio-<version>
export PATH=$PWD/bin:$PATH
```

2.2 Instalacja Istio

Kroki:

1. **Uruchom klaster Minikube:**

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

   Upewnij się, że klaster jest gotowy:

   ```
   kubectl get nodes
   ```

2. **Zainstaluj Istio z profilem demo:**

   ```
   istioctl install --set profile=demo -y
   ```

   Profil demo zawiera wszystkie podstawowe komponenty Istio, takie jak Ingress Gateway, Prometheus, Grafana i Kiali.

3. **Zweryfikuj instalację Istio:**

   ```
   kubectl get pods -n istio-system
   ```

   Upewnij się, że wszystkie Pody w namespace `istio-system` mają status `Running`.

4. **Włącz wstrzykiwanie sidecarów:** Oznacz namespace `default`, aby Istio automatycznie wstrzykiwało proxy Envoy do nowych Podów:

```
kubectl label namespace default
    istio-injection=enabled
```

2.3 Testowanie instalacji

Aby upewnić się, że Istio działa poprawnie:

1. **Przejdź do interfejsu Kiali:**

```
kubectl -n istio-system port-forward svc/kiali
    20001:20001
```

   Otwórz przeglądarkę i przejdź pod adres `http://localhost:20001`.

2. **Przejdź do interfejsu Grafana:**

```
kubectl -n istio-system port-forward svc/grafana
    3000:3000
```

   Otwórz przeglądarkę i przejdź pod adres `http://localhost:3000`.

Po wykonaniu powyższych kroków środowisko jest gotowe do pracy z Istio i ćwiczeń na aplikacji Bookinfo.

3 Wdrożenie aplikacji Bookinfo

Aplikacja Bookinfo jest przykładem aplikacji mikroserwisowej, która pozwala na demonstrację funkcji Istio, takich jak routing ruchu, monitorowanie i fault injection. Składa się z czterech głównych usług: `productpage`, `details`, `reviews` oraz `ratings`.

3.1 Architektura aplikacji Bookinfo

Komponenty aplikacji:

- **productpage:** Frontend aplikacji wyświetlający szczegóły produktu.
- **details:** Usługa dostarczająca szczegóły produktu.
- **reviews:** Usługa obsługująca recenzje produktu. Istnieją trzy wersje:
  - **v1:** Wyświetla tylko tekst recenzji.
  - **v2:** Wyświetla tekst recenzji z oceną liczbową.
  - **v3:** Wyświetla tekst recenzji z oceną liczbową oraz gwiazdkami.
- **ratings:** Usługa obsługująca oceny liczbowe.

Architektura aplikacji Bookinfo jest idealnym przykładem rozproszonego systemu mikroserwisowego, umożliwiającym testowanie różnych scenariuszy w środowisku Istio.

3.2 Wdrożenie aplikacji Bookinfo

Kroki:

1. **Zastosuj manifesty aplikacji Bookinfo:**

   ```
   kubectl apply -f
       samples/bookinfo/platform/kube/bookinfo.yaml
   ```

   Manifest `bookinfo.yaml` zawiera definicje Deployment oraz Service dla wszystkich komponentów aplikacji.

2. **Sprawdź status wdrożenia:**

   ```
   kubectl get pods
   kubectl get services
   ```

   Upewnij się, że wszystkie Pody i usługi są uruchomione.

3. **Eksponuj aplikację za pomocą Istio Gateway:** Zastosuj manifest gateway dla Bookinfo:

   ```
   kubectl apply -f
       samples/bookinfo/networking/bookinfo-gateway.yaml
   ```

   Gateway Istio umożliwia dostęp do aplikacji spoza klastra Kubernetes.

4. **Znajdź adres Ingress Gateway:** Sprawdź adres i port `EXTERNAL-IP` dla gateway:

   ```
   kubectl get svc istio-ingressgateway -n
       istio-system
   ```

5. **Uzyskaj dostęp do aplikacji:** Przejdź w przeglądarce pod adres:

   ```
   http://<EXTERNAL-IP>/productpage
   ```

3.3 Obserwacja ruchu w Kiali

Kroki:

1. Uruchom interfejs Kiali:

   ```
   kubectl -n istio-system port-forward svc/kiali
       20001:20001
   ```

   Przejdź pod adres `http://localhost:20001` w przeglądarce.

2. Sprawdź przepływ ruchu między usługami:

   - W Kiali wybierz namespace `default`.
   - Obserwuj graficzną reprezentację przepływu ruchu między komponentami aplikacji Bookinfo.

3.4 Podsumowanie sekcji

Po wdrożeniu aplikacji Bookinfo studenci powinni być w stanie:

- Zrozumieć architekturę aplikacji mikroserwisowej.
- Eksponować aplikacje przy użyciu Istio Gateway.
- Obserwować przepływ ruchu między usługami w narzędziu Kiali.

4 Routing ruchu i kontrola przepływu

Routing ruchu w Istio pozwala na precyzyjne kontrolowanie, jak ruch sieciowy jest kierowany między usługami. Dzięki regułom routingu można np. rozdzielać ruch na różne wersje aplikacji, wprowadzać ważenie ruchu czy zmieniać konfiguracje zależnie od użytkownika.

4.1 Domyślne routing ruchu

Teoria: Domyślne reguły routingu w Istio kierują cały ruch do pierwszej dostępnej wersji usługi. Można jednak skonfigurować reguły bardziej precyzyjnie, np. tak, aby cały ruch do usługi `reviews` był kierowany tylko do wersji `v1`.
Kroki:

1. **Dodaj reguły routingu:**

   ```
   kubectl apply -f
       samples/bookinfo/networking/destination-rule-all.yaml
   kubectl apply -f
       samples/bookinfo/networking/virtual-service-all-v1.yaml
   ```

   Pierwsze polecenie definiuje reguły `DestinationRule`, które opisują wszystkie dostępne wersje usług. Drugie polecenie ustawia `VirtualService`, który kieruje cały ruch do wersji `v1`.

2. **Sprawdź konfigurację routingu:**

   ```
   kubectl get virtualservice reviews -o yaml
   ```

3. **Zweryfikuj działanie:** Przejdź pod adres `http://<EXTERNAL-IP>/productpage` i upewnij się, że na stronie wyświetlane są recenzje bez gwiazdek (cecha wersji `v1`).

4.2 Ważenie ruchu między wersjami

Teoria: Ważenie ruchu pozwala na równoczesne kierowanie zapytań do różnych wersji usługi. Na przykład można ustawić, aby 50% ruchu było kierowane do wersji `v1`, a pozostałe 50% do wersji `v2`.
Kroki:

1. **Stwórz regułę ważenia ruchu:** Utwórz plik YAML o nazwie `virtual-service-reviews-v1-v2.yaml`:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
      weight: 50
    - destination:
        host: reviews
        subset: v2
      weight: 50
```

2. **Zastosuj regułę:**

```
kubectl apply -f
    virtual-service-reviews-v1-v2.yaml
```

3. **Testuj ważenie ruchu:** Odświeżaj stronę `http://<EXTERNAL-IP>/productpage` i obserwuj, jak niektóre zapytania wyświetlają recenzje bez gwiazdek (`v1`), a inne z gwiazdkami (`v2`).

4.3 Routing na podstawie użytkownika

Teoria: Istio pozwala na kierowanie ruchu na podstawie określonych atrybutów, takich jak nazwa użytkownika, nagłówki HTTP czy parametry zapytania. W tym przykładzie cały ruch użytkownika `test-user` będzie kierowany do wersji v3.

Kroki:

1. **Dodaj regułę routingu opartego na użytkowniku:** Utwórz plik YAML o nazwie `virtual-service-reviews-user.yaml`:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
```

```
http:
- match:
  - headers:
      end-user:
        exact: test-user
  route:
  - destination:
      host: reviews
      subset: v3
- route:
  - destination:
      host: reviews
      subset: v1
```

2. **Zastosuj regułę:**

```
kubectl apply -f
    virtual-service-reviews-user.yaml
```

3. **Testuj:** W przeglądarce zaloguj się jako użytkownik `test-user` (jeśli aplikacja to obsługuje) lub ustaw odpowiedni nagłówek HTTP i obserwuj, jak użytkownik `test-user` trafia na wersję `v3`, a inni użytkownicy na wersję `v1`.

4.4 Podsumowanie sekcji

Po ukończeniu tej sekcji studenci powinni być w stanie:

- Skonfigurować routing ruchu w aplikacji Bookinfo.
- Zastosować ważenie ruchu między różnymi wersjami usług.
- Kierować ruch na podstawie atrybutów, takich jak nazwa użytkownika.

5 Fault Injection i Circuit Breaking

5.1 Fault Injection (Wstrzykiwanie błędów)

Teoria: Fault Injection umożliwia symulowanie różnych warunków awaryjnych w komunikacji między usługami, takich jak opóźnienia w odpowiedziach lub błędy HTTP. Mechanizm ten pozwala testować odporność systemu na awarie oraz obserwować jego zachowanie w nieoczekiwanych sytuacjach.
Kroki:

1. **Wstrzyknięcie opóźnienia:** Utwórz plik YAML o nazwie `virtual-service-reviews-delay.yaml`:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - fault:
      delay:
        fixedDelay: 7s
        percentage:
          value: 100
    route:
    - destination:
        host: reviews
        subset: v1
```

Zastosuj regułę:

```
kubectl apply -f
    virtual-service-reviews-delay.yaml
```

2. **Symulacja błędów HTTP:** Utwórz plik YAML o nazwie `virtual-service-reviews-abort.yaml`:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - fault:
      abort:
        httpStatus: 500
        percentage:
          value: 50
    route:
    - destination:
        host: reviews
        subset: v1
```

Zastosuj regułę:

```
kubectl apply -f
    virtual-service-reviews-abort.yaml
```

3. **Obserwacja efektów:** Odświeżaj stronę `http://<EXTERNAL-IP>/productpage` i obserwuj, jak aplikacja reaguje na opóźnienia lub błędy HTTP w usłudze `reviews`.

5.2 Circuit Breaking (Zabezpieczenie przeciążeniowe)

Teoria: Circuit Breaking to mechanizm ograniczania przeciążenia w komunikacji między usługami, który zapobiega dalszemu obciążaniu usługi w przypadku wystąpienia awarii lub nadmiernego ruchu.
Kroki:

1. **Konfiguracja Circuit Breaking:** Utwórz plik YAML o nazwie `destination-rule-reviews-cb.yaml`:

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: reviews
spec:
  host: reviews
  trafficPolicy:
    connectionPool:
      http:
        http1MaxPendingRequests: 2
        maxRequestsPerConnection: 1
```

   Zastosuj regułę:

```
kubectl apply -f destination-rule-reviews-cb.yaml
```

2. **Testowanie Circuit Breaking:** Uruchom narzędzie `ab` (Apache Benchmark):

```
ab -n 100 -c 5 http://<EXTERNAL-IP>/productpage
```

   Obserwuj, jak Circuit Breaker ogranicza liczbę równoczesnych połączeń do usługi `reviews`, zwracając błędy dla nadmiarowego ruchu.

3. **Obserwacja w Kiali i Prometheus:** W Kiali obserwuj przepływ ruchu oraz błędy związane z Circuit Breaking. W Prometheus możesz użyć zapytań takich jak:

```
sum(rate(istio_requests_total{response_code=~"5.*"}[1m]))
```

5.3 Zadanie samodzielne

Cel: Wprowadzenie obu mechanizmów do jednej usługi i analiza ich wzajemnego wpływu.

- Wstrzyknij 5-sekundowe opóźnienie dla usługi `ratings`.
- Dodaj regułę Circuit Breaking, która ograniczy liczbę równoczesnych połączeń do `ratings` do 3.
- Uruchom obciążenie za pomocą `ab` i obserwuj, jak mechanizmy Fault Injection i Circuit Breaking wpływają na działanie aplikacji.

6 Zadania końcowe

W tej sekcji znajdziesz zadania, które pozwolą Ci utrwalić zdobytą wiedzę i jeszcze lepiej zrozumieć możliwości Istio. Wykonując je, przećwiczysz zaawansowane mechanizmy routingu, Fault Injection i Circuit Breaking, a także ich połączenie.

6.1 Zadanie 1: Wdrożenie zaawansowanego routingu

- Skonfiguruj routing tak, aby 50% ruchu było kierowane do `reviews:v2`, 30% do `reviews:v3`, a 20% do `reviews:v1`.
- Przetestuj konfigurację, odświeżając stronę aplikacji `Bookinfo`.
- Zweryfikuj przepływ ruchu w narzędziu Kiali.

6.2 Zadanie 2: Wstrzykiwanie błędów w wybranej usłudze

- Wprowadź błędy HTTP 503 dla 20% zapytań kierowanych do usługi `ratings`.
- Obserwuj reakcję aplikacji Bookinfo na błędy w usłudze podrzędnej.
- Monitoruj liczbę błędów HTTP za pomocą Prometheus.

6.3 Zadanie 3: Testowanie Circuit Breaking w warunkach wysokiego obciążenia

- Skonfiguruj Circuit Breaking dla usługi `details`, ograniczając liczbę równoczesnych połączeń do 5.
- Wygeneruj obciążenie za pomocą narzędzia `ab` (Apache Benchmark).
- Sprawdź efekty w Kiali i Prometheus.

6.4 Zadanie 4: Integracja Fault Injection i Circuit Breaking

- Połącz mechanizmy Fault Injection i Circuit Breaking dla usługi `reviews`.
- Wprowadź opóźnienie 3 sekundy dla 10% ruchu i ogranicz liczbę równoczesnych połączeń do 2.

- Przeanalizuj wpływ obu mechanizmów w narzędziach Kiali i Prometheus.

7 Podsumowanie

Podczas tego laboratorium zapoznałeś się z podstawowymi koncepcjami Istio oraz sposobami zarządzania ruchem w aplikacjach mikroserwisowych. Wdrożenie aplikacji Bookinfo pozwoliło na praktyczne przećwiczenie mechanizmów Istio, takich jak:

- Routing ruchu (ważenie ruchu, routing na podstawie użytkownika).
- Fault Injection i Circuit Breaking jako narzędzia do testowania odporności systemu.
- Obserwacja ruchu i analiza metryk za pomocą Kiali i Prometheus.

Istio dostarcza istotnych narzędzi w zarządzaniu systemami rozproszonymi, szczególnie w środowiskach produkcyjnych. Warto pamiętać, że zastosowanie takich rozwiązań jak Service Mesh jest najbardziej efektywne w dużych systemach złożonych z wielu usług, gdzie ręczne zarządzanie komunikacją staje się niewykonalne.
Jeżeli napotkałeś trudności w realizacji zadań, zachęcamy do analizy kroków opisanych w skrypcie lub sięgnięcia po dokumentację Istio, która zawiera szczegółowe informacje na temat każdego z omawianych mechanizmów.