Managing microservices with Istio Service Mesh

Rafik Harabi, INNOVSQUARE

SRECon EMEA 2019
Quick survey before we start

- Who are already using Kubernetes?

- Who are developing microservices?

- Who are using client library approach to implement microservices common concerns (Circuit Breaker, Retry...)?

- Who are using Istio or any other service mesh technology?
Moving to microservices network challenges

Network Reliability

Fault tolerance and resiliency

Monitoring and Observability
Challenges deep-dive

Network Reliability
Service have to handle the network facts:
- Network latency / bandwidth
- Transport cost
- Topology and administration

Fault Tolerance
Service have to be able to handle outright failure and timeouts:
- Avoid cascading failure
- Retries
- Circuit breaking

Monitoring
We have to:
- monitor the delivered microservices and their interactions
- Trace requests and identify potential hotspots
The evolution of microservices frameworks: from Netflix OSS to Istio
2011
NetFlix OSS
first microservices patterns and libraries open-sourced

2013
Spring Cloud
Enterprise microservice framework for Java

2014
Kubernetes
Workload orchestration

2015
Docker
Containerization

2018
Istio
Service mesh
Microservices challenges

- N to N communications.
- Distributed software interconnection and troubleshooting is hard.
- Containers should stay thin and platform agnostic.
- Upgrade of polyglot microservices is hard at scale.
Microservices building blocks

- Configuration Service
- Service Registry / Discovery
- Circuit Breaker / Retry
- Rate Limiting
- Event Driven Messaging (Async)
- Audit
- Load Balancing / Intelligent Routing
- API Gateway
- Authentication & Authorization
- Monitoring
- Distributed tracing
- Log Aggregation
Microservices building blocks

- Business Value
  - API Gateway
  - Configuration Service
  - Circuit Breaker / Retry
  - Service Registry / Discovery
  - Rate Limiting
  - Load Balancing / Intelligent Routing
  - Event Driven Messaging (Async)
  - Authentication & Authorization
  - Monitoring
  - Distributed tracing
  - Log Aggregation

- Audit
Code oriented frameworks

Service A
- Business logic
- Circuit Breaker
- Rate limiting
- Tracing
- Metrics

Service B
- Business logic
- Circuit Breaker
- Rate limiting
- Tracing
- Metrics
Code oriented pattern

Business Values
- Business Service
- Load Balancing / Intelligent Routing
- Authentication & Authorization
- Circuit Breaker/Retry
- Rate Limiting

Communication
- Event Driven Messaging (Async)

Foundation
- Configuration Service
- Service Registry / Discovery
- API Gateway

Monitoring and Observability
- Monitoring
- Distributed tracing
- Log Aggregation
- Audit
Code oriented solutions limits

- Language oriented.
- Error prone (implementation).
- Hard to upgrade each microservice when system grow.
- Add technical challenges and duties to development teams.
- Different teams in the same organization may have different implementations.
- Each team should maintain his implementation.

Microservices challenges need to be solved uniformly.
Desired state

- Keep microservice concerns separate from the business logic.
- The network should be transparent to applications.
- Developers should focus on delivering business capabilities and not implementing microservices common concerns.
- Microservices interconnection should be language agnostic.
- Easy to upgrade solution.
A service mesh is a dedicated infrastructure layer for handling service-to-service communication. It’s responsible for the reliable delivery of requests through the complex topology of services that comprise a modern, cloud native application.

buoyant.io
Service Mesh

The design

Each service will have its own proxy service and all these proxy services together form the “Service Mesh”. All the requests to and from each service will go through the mesh proxies.

Proxies are also known as sidecars.
Sidecar pattern

Service to service communication

Injected

Network concerns become transparent

Proxy
- Circuit Breaker
- Rate limiting
- Tracing
- Metrics

Service A
- Business logic

Proxy
- Circuit Breaker
- Rate limiting
- Tracing
- Metrics

Service B
- Business logic
History of Istio

- Envoy proxy (Istio data plane) created by Lyft and open-sourced in 2016.
- IBM and Google launch the project in May 2017.
- First major version released in July 2018.
- Current version: 1.3
Istio goal

Develop an open technology that provides a uniform way to connect, secure, manage and monitor a network of microservices regardless of the platform source or vendor.
Solution

Istio Promises

- Focus on business logic and spent less time with common concerns.
- No change in the service code.
- Central configuration management.
- Easy to upgrade
- Security
<table>
<thead>
<tr>
<th>Istio does:</th>
<th>Istio does not:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Service discovery</td>
<td>- Event Driven Asynchronous communication</td>
</tr>
<tr>
<td>- Load Balancing &amp; Intelligent Routing</td>
<td>- Service Orchestration</td>
</tr>
<tr>
<td>- Resiliency: Circuit Breaker &amp; Retry</td>
<td></td>
</tr>
<tr>
<td>- Rate Limiting</td>
<td></td>
</tr>
<tr>
<td>- Authentication and Authorization</td>
<td></td>
</tr>
<tr>
<td>- Service to Service mTLS</td>
<td></td>
</tr>
<tr>
<td>- Policy enforcement</td>
<td></td>
</tr>
<tr>
<td>- Observability</td>
<td></td>
</tr>
<tr>
<td>- Monitoring metrics</td>
<td></td>
</tr>
<tr>
<td>- Distributed tracing</td>
<td></td>
</tr>
</tbody>
</table>
Service Discovery

Kubernetes provide service discovery, why do I need an extra one 😐?

Istio supports:

- HTTP L7 filter
- HTTP L7 routing (based on http headers and cookies)
- First class HTTP/2
- gRPC support
- Fine-grained traffic splitting
Architecture
## Istio building blocks 1/2

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pilot</td>
<td>Responsible for service discovery and for configuring the Envoy sidecar proxies</td>
</tr>
<tr>
<td>Citadel</td>
<td>Automated key and certificate management</td>
</tr>
</tbody>
</table>
| Mixer                | Istio-Policy: policy enforcement  
Istio-Telemetry: gather telemetry data                                       |
| Galley               | Configuration ingestion for istio components                                 |
| Ingress Gateway      | manage inbound connection to the service mesh                                |
| Egress Gateway       | manage outbound connection from the service mesh                             |
| Sidecar injector     | Inside sidecar for enabled pods/namespaces                                   |
Istio building blocks 1/2

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prometheus</td>
<td>Metrics collection</td>
</tr>
<tr>
<td>Grafana</td>
<td>Monitoring dashboard</td>
</tr>
<tr>
<td>Jaeger</td>
<td>Distributed tracing</td>
</tr>
<tr>
<td>Kiali</td>
<td>Observability dashboard</td>
</tr>
</tbody>
</table>
https://www.istioworkshop.io/