Service Mesh: hype or reality?

Red Hat Forum BeLux 2019

Filip Lenaerts
October 8, 2019
agenda.

1. Introduction
2. Service Mesh Concepts
3. Service Mesh with Istio
4. Use-cases
5. Conclusion
Intro
Intro
Practice Technology Focus
<table>
<thead>
<tr>
<th></th>
<th>New</th>
<th>Refresh</th>
<th>Fail fast</th>
<th>Cloud native</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airbnb</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uber</td>
<td>Cloud first</td>
<td>Do or die</td>
<td>Cognitive first</td>
<td>CDO</td>
</tr>
<tr>
<td>Millennials</td>
<td>Mobile first</td>
<td></td>
<td></td>
<td>Digital</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>INNOVATION BINGO</strong> (free square)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A/B testing</td>
<td>Layer of Innovation</td>
<td>Alibaba</td>
<td>IPO</td>
<td>Blue/green</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disruption</td>
<td>Sustainable innovation</td>
<td>Legacy</td>
<td>Incubator</td>
<td>strategy</td>
</tr>
</tbody>
</table>
Hype?

Sidecars?
Yet another new paradigm

Easy development?
Hard to get started with

Istio?
Yet another new Greek work to remember!

New Infrastructure?
My ops already get apesh*t crazy with those containers
Cloud native
Applications    Platform
Cloud Native

CNCF Cloud Native Definition v1.0

Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic, public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable developers to build systems that are resilient, manageable, and observable. Combined, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

The Cloud Native Computing Foundation seeks to drive adoption of this paradigm by fostering and sustaining an ecosystem of open source, vendor-neutral projects. We democratize state-of-the-art patterns to make these innovations accessible for everyone.

https://github.com/cncf/toc/blob/master/DEFINITION.md
Cloud native apps

Microservices - one of many implementation options

- Service-oriented architecture
- Each functionality is one service (anti-monolith)
  Communication between services: queueing service

Enables:
- Parallel development of services
- Fast release cycles
- Module based upgrades (theory)

Requires:
- DevOps approach
- Versioned APIs

“According to IDC, by 2022, 90% of all new apps will feature microservices architectures that improve the ability to design, debug, update, and leverage third-party code; 35% of all production apps will be cloud-native”
Cloud native platform

NIST 800-145

On demand self-service

Broad Network Access

Pooling

Rapid Elasticity

Measured Service
Infrastructure

Common ground
Service Mesh

Infrastructure layer for microservices communication

Alleviates microservice (code and its developers) from:
- Encryption (S2S - zero trust network)
- Authentication
- Authorisation
- Circuit breaker
- Load balancing
- Any non-functionals

Allows focus on the core uService functionality

Often implemented as side car pattern

Offers additional:
- Monitoring and Traceability (for Ops)
- Enforcement (for SecOps)
Business drivers

- non-functional features/bugfixes **without impacting** the core business functionality
- Faster time to market
- Enables **shift left** for Security
- **Dev+Ops** Happy:
  - Dev: doesn’t care (!) about non-functionals
  - Ops: implementing changes without impacting service
- Full **visibility** on traffic and versions
2 Service Mesh: concepts
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. In practice, the service mesh is typically implemented as an array of *lightweight network proxies* that are deployed alongside application code, without the application needing to be aware. (*Buoyant.io*)

A microservices architecture isolates software functionality into multiple independent services that are independently deployable, highly maintainable and testable, and organized around specific business capabilities. […] On a technical level, microservices enable *continuous delivery* and deployment of large, complex applications. On a higher business level, microservices help deliver speed, *scalability*, and *flexibility* to companies trying to achieve agility in rapidly evolving markets. (*New Relic*)

A service mesh is an emerging architecture for *dynamically* linking to one another the chunks of server-side applications -- most notably, the *microservices* (*ZDNet*)
PICTURES...

...OR IT DIDN'T HAPPEN.
Before Istio
After Istio
### Implementation types of Service Mesh

#### Library

**Definition**
Each µservice includes library code implementing service mesh features.

**Characteristics**
Languages Dependencies.  
Original, simple and straightforward.  
Trust boundary is small (call library inside the process).  
Dedicated Resources.

**Examples**
Hystrix & Ribbon

#### Sidecar

**Definition**
Container running adjacent to the container in the same pod.

**Characteristics**
One sidecar per pod which manages the traffic in/out of the application container.  
Language agnostic  
Dedicated resource (per pod)  
More flexibility (Security – routing – …) and easier to secure

**Examples**
Istio – AVI networks – Tigera – Aspen Mesh

#### Node Agent

**Definition**
Agent (running in a User space process) installed per node/machine managing all the containers on a particular node/machine.

**Characteristics**
Servicing an heterogeneous mix of workloads  
Language Agnostic  
Resources Sharing less complex for the configuration (one config per node).

**Examples**
Consul - Linkerd
Standardisation

A standard interface for Service Meshes

**Basic feature set** for most common features
- Traffic policy
- Traffic telemetry
- Traffic management

**Kubernetes native** specified as a collection of Kubernetes Custom Resource Definitions (CRD)

**Provider agnostic**

https://smi-spec.io/
SMI: partners

- Microsoft
- LINKERD
- HashiCorp
- solo.io
- Red Hat
- RANCHER
- docker
- meshery
- weaveworks
- ASPEN MESH
- Pivotal
- VMware
- Canonical
- Kubecost
Service Mesh with Istio
Istio @ GitHub
14,500 stars
6,400 commits
300 contributors

Integrations
Aspen Mesh
Avi Networks
Cisco
OpenShift
NGINX
Rancher
Tufin Orca
Tigera
Twistlock
VMware.

Features

Automatic load balancing for HTTP, gRPC, WebSocket, and TCP traffic.

Fine-grained control of traffic behavior with rich routing rules, retries, failovers, and fault injection.

A pluggable policy layer and configuration API supporting access controls, rate limits and quotas.

Automatic metrics, logs, and traces for all traffic within a cluster, including cluster ingress and egress.

Secure service-to-service communication in a cluster with strong identity-based authentication and authorization.
Main architecture

Separation Data / Control pane

**Data Plane:**
Intelligent sidecar envoy proxies

**Control Plane:**
Sets routes
Configures policy and telemetry mixer hub
Istio – Envoy proxy

A real working horse

- A high-performance proxy developed in C++

- Built-in features, for example:
  - Dynamic service discovery
  - Load balancing
  - TLS termination
  - HTTP/2 and gRPC proxies
  - Circuit breakers
  - Health checks
  - Staged rollouts with %-based traffic split
  - Fault injection
  - Rich metrics

- Customization / Extending
  - Lua scripting
  - C++ envoy

- Platform independent
Istio – Envoy proxy at the core of your traffic

- as a sidecar
  - extract a wealth of signals about traffic behavior as attributes.
  - use these attributes in Mixer to enforce policy decisions,
  - send them to monitoring systems to provide information about the behavior of the entire mesh.

- The sidecar proxy model also allows you to add Istio capabilities to an existing deployment with no need to rearchitect or rewrite code.

Credit: Jess Wheelock
Istio – Mixer

Mixer
- a platform-independent component.
- Mixer **enforces access control** and usage policies across the service mesh, and **collects telemetry** data from the Envoy proxy and other services. The proxy extracts request level attributes, and sends them to Mixer for evaluation.
Istio – Pilot

- **Pilot** provides **service discovery** for the Envoy sidecars, **traffic management** capabilities for intelligent routing (e.g., A/B tests, canary rollouts, etc.), and **resiliency** (timeouts, retries, circuit breakers, etc.).

- converts high level routing rules that control traffic behavior into Envoy-specific configurations, and propagates them to the sidecars at runtime.

- abstracts platform-specific service discovery mechanisms and synthesizes them into a standard format that any sidecar conforming with the Envoy data plane APIs can consume.
Istio – Citadel

Citadel
- Identity component
- Enables strong service-to-service and end-user authentication with built-in identity and credential management.
- Built-in PKI
Istio – Galley

Galley

- is Istio’s **configuration validation**, ingestion, processing and distribution component.
- insulating the rest of the Istio components from the details of obtaining **user configuration** from the underlying platform (e.g. Kubernetes).
Kiali

- Manage, observe and analyzes service mesh, services and related objects as deployments
- Now OpenShift integrated
Jaeger tracing

- Jaeger + Kiali integration
Netflix Vizceral
Service Mesh applications
4 Use-cases
Use-cases: overview

1. Traffic management
2. Canary deployments
3. Environment as service mesh
4. Traffic shadowing
5. Canary Analysis
6. Istio gateway: build real hybrid applications
Traffic management

**Rule-based traffic control** means you can route a specific portion of traffic to a specific instance of a service (for example, specify the percentage of traffic that should hit a canary deploy), or set routing rules based on the content of a request.

Flexible routing strategies, based on:
- Domain, subdomain
- URL, paths
- Headers
- User-agent
- Geolocalisation

---

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: productpage
spec:
  hosts:
    - productpage
  http:
    - match:
        - uri:
            prefix: /api/v1
    route:
      ...
```
Traffic management – Visibility with Kiali

Flexible routing strategies, based on:

- Domain, subdomain
- Url, paths
- Headers
- User-agent
- Geolocalisation

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: productpage
spec:
  hosts:
    - productpage
  http:
    - match:
        - uri:
            prefix: /api/v1
  route:
  ...```

Traffic management – Visibility with Kiali

Flexible routing strategies, based on:
- Domain, subdomain
- URL, paths
- Headers
- User-agent
- Geolocalisation

apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: productpage
spec:
  hosts:
  - productpage
  http:
    match:
    - uri:
        prefix: /api/v1
    route:
Canary Deployment
Traffic shadowing and sensitive deliveries

- Test for errors, exceptions, performance, and result parity.
- Mirror 100% of the traffic
- No impact on current traffic
- Compare and observe with production data (Twitter Diffy like)
- Traffic is mirrored as “fire-and-forget”
A prerequisite to implementing canary releases is the ability to effectively observe and monitor your infrastructure and application stack.

Gradual rollout of new functionality limits the potential system blast radius of any operational issues.

Deployment impact analysis by metrics and traces analytics (Harness like)
Service-to-Service as Code :-) 

Define cluster wide routing definitions 

Apply to any environment
Hype or Reality
Oh, my God! Okay, it's happening!
Reality?

Sidecars?
Yet another new paradigm?
Proxies are proven technology
Easy development?
Hard to get started with.
Easier, no non-functionals

Istio?
Yet another **new greek work to remember**!

YEP!
New Infrastructure?

My ops already get apesh*t crazy with those containers
Your Ops get F_I_N_A_L_L_L_Y insight in those containers
any questions?
thank you.

#TechforPeople.