ISTIO

A service mesh for microservices at scale

Yong Feng – ICP STSM
yongfeng@ca.ibm.com

IBM Cloud

AGENDA

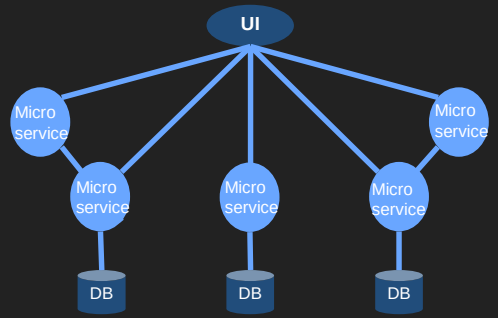
1. Microservices Architecture
   - The Problem Space & Challenges

2. ISTIO
   - What Is It?
   - Architecture Overview

3. Demo

MICROSERVICES ARCHITECTURE

An **engineering approach** focused on decomposing an application into **single function** modules with **well defined interfaces** which are **independently** deployed and operated by a **small team** who owns the **entire lifecycle** of the service.



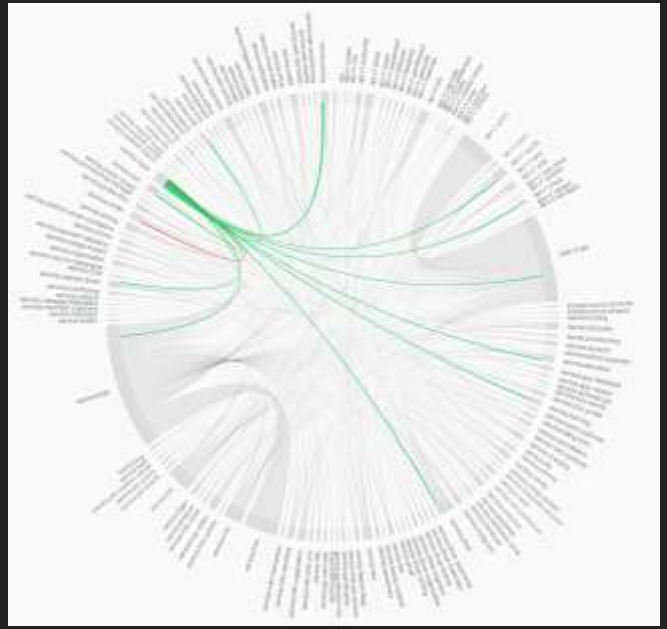
THE TRADE OFF

Improved delivery velocity in exchange for increased operational complexity.

Containers and Kubernetes are great enablers to these design goals: clean packaging, rapid deployment, consistency, reliability & scalability

This is the reality of microservices implementation at scale.

Kubernetes and containers in of themselves does not address these complexity challenges



Hailo microservices

Cited from https://medium.com/@mattheath/a-long-journey-into-a-microservice-world-a714992d2841

MICROSERVICES ARE HARD

- Applications aren’t running in green-field environments
- Challenges in the network in between the Services
- Network layer is hard to manage
- Tooling is nascent

Things to consider

- Security
- Canary deployments
- A/B testing
- Circuit breaking
- Rate limiting
- Fault injection
- Tracing
- Monitoring
- Many more….

It’s doable, but…

It will require a lot of coding

Service Mesh

A dedicated infrastructure layer for managing service-to-service communication to make it manageable, visible and controlled

A Control Plane & Network Overlay in between the Services

Istio

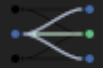
A service mesh designed to connect, manage and secure micro services.

Using Open Source & Open Standards

(Joint project between IBM, Google, Lyft & others)

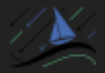
With Zero Application Code Changes

MAIN FEATURES



Traffic management

A/B tests, Canary Releases, Red/Black deployments, Circuit Breaker, Fault Injection



Observability

Dependancies and traffic, Distributed Tracing, Performance metrics

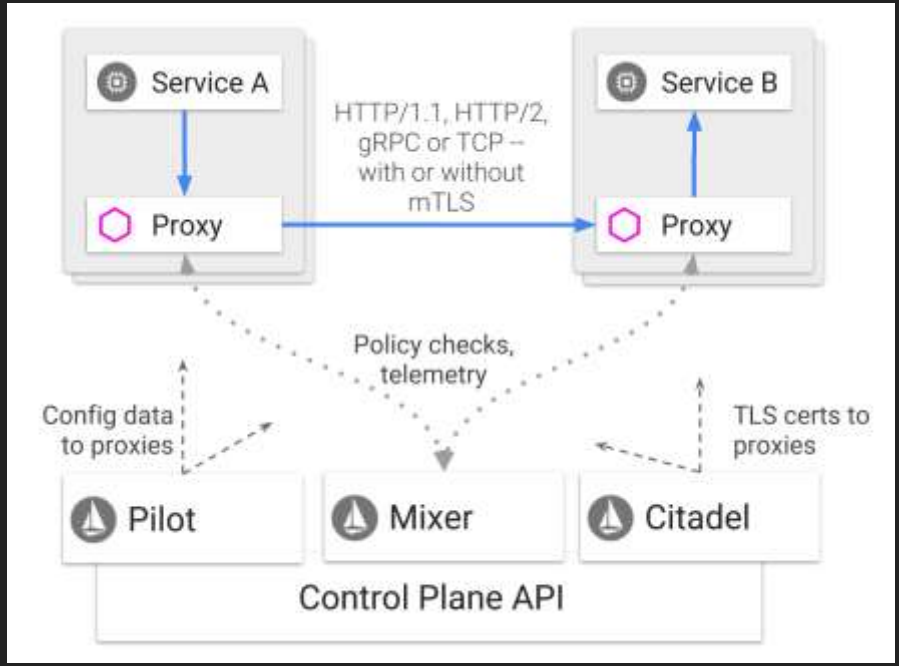
Security

Enterprise access policy, Security policy, Certification management

ISTIO CONCEPTS

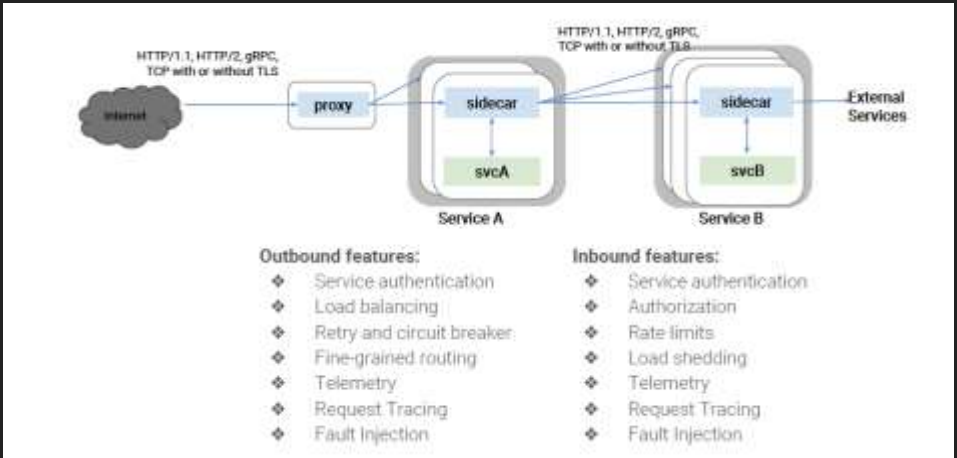
Data Plane: Composed of a set of intelligent proxies (Envoy) deployed as sidecars. These proxies mediate and control all network communication between microservices along with Mixer, a general-purpose policy and telemetry hub.

Control Plane: Manages and configures the proxies to route traffic. Additionally, the control plane configures Mixers to enforce policies and collect telemetry.



Proxy: Based on Envoy, mediates inbound and outbound traffic for all Istio-managed services.
Pilot: Configures Istio deployments and propagate configuration to the other components of the system.
Mixer: Responsible for policy decisions and aggregating telemetry data from the other components in the system using a flexible plugin architecture.
Citadel: Secures the service-to-service communication and provides a key management system to manage keys and certificates.

ENVOY
SIDE CAR TECHNOLOGY OF CHOICE



Cited from https://www.infoq.com/presentations/istio-service-mesh

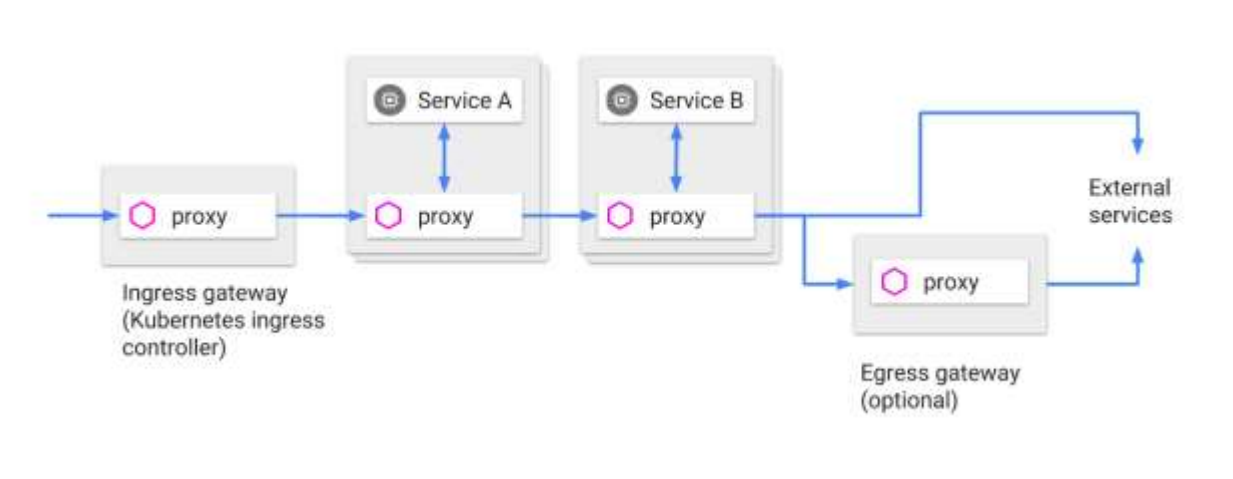
envoy

ENVOY IS AN OPEN SOURCE EDGE AND SERVICE PROXY, DESIGNED FOR CLOUD-NATIVE APPLICATIONS

https://github.com/envoyproxy/envoy

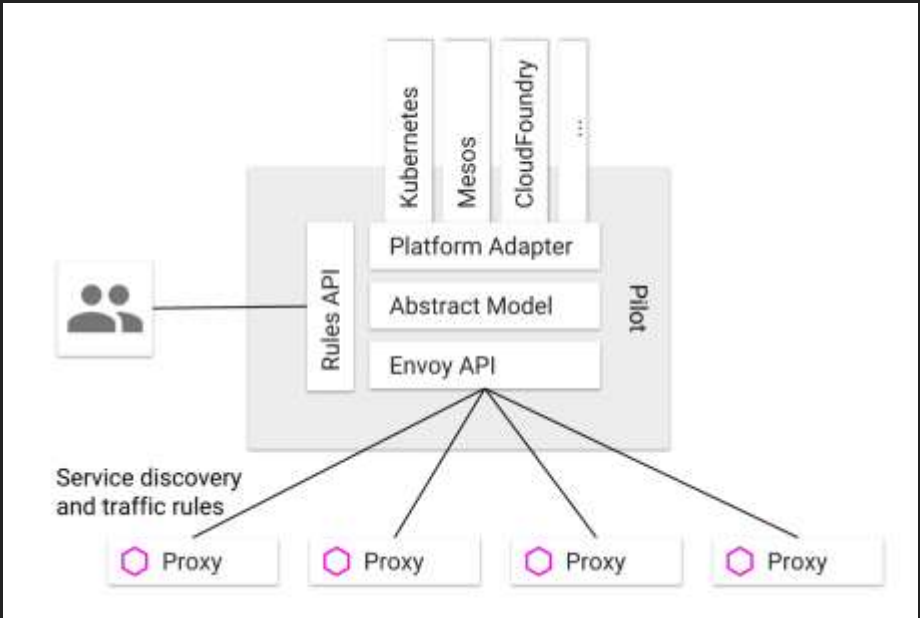
- L7 proxy and communication bus
- Written in C++11
- Pluggable architecture (L3 and L7)
- HTTP/1.1, HTTP/2 and gRPC
- Health checking
- L7 routing
- Advanced loadbalancer
- Dynamic configuration
- Metrics and tracing
- Battle tested at Lyft

ROUTING CONTROL



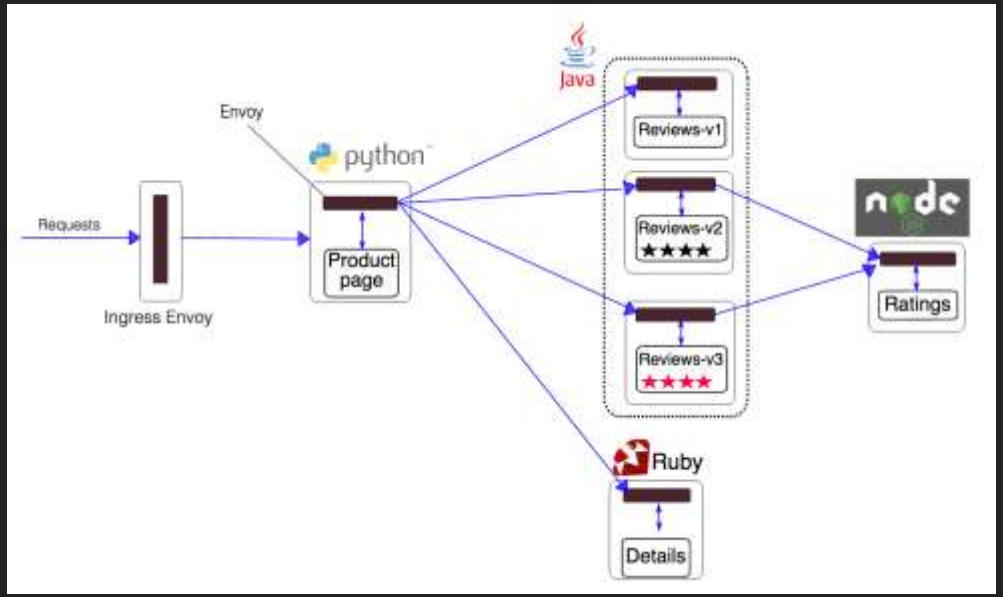
- **Ingress**
  - **Gateway**: configures a load balancer (one or multiple FQDN) for HTTP/TCP traffic at the edge of the mesh
- **Inside mesh**
  - **VirtualService**: defines the rules that control how requests for a service (one or multiple FQDN) are routed.
  - **DestinationRule**: configures the set of policies (a set of instances) to be applied to a request after VirtualService routing has occurred
- **Egress**
  - **ServiceEntry**: commonly used to enable requests to services outside of an Istio service mesh

TRAFFIC MANAGEMENT - PILOT



- Maintain platform-agnostic model of services in the mesh
- Platform-specific adapter implement platform specific logic
  - Service discovery
  - Ingress resource
  - Rule definition
- Push configuration to Envoy and apply without restarts

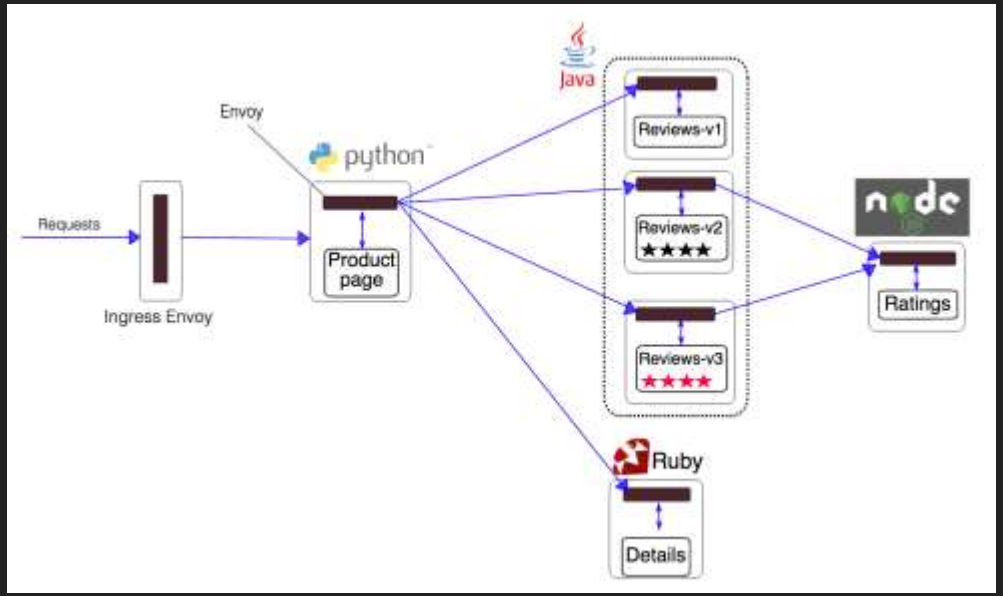
TRAFFIC MANAGEMENT
REQUEST ROUTE (DEMO)



- Route to specific version of service
- Route based on request attribute
- …

Three versions of the service of reviews

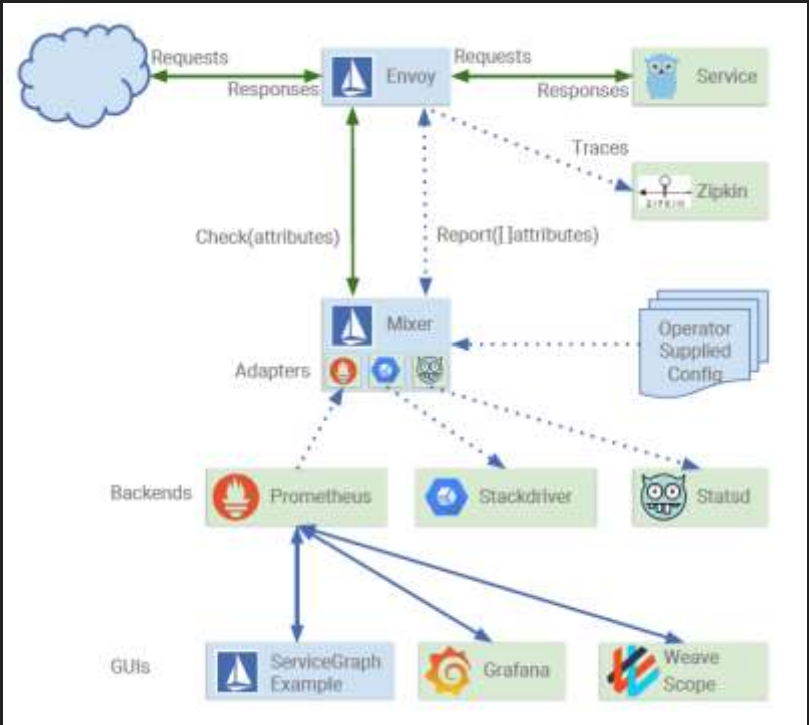
TRAFFIC MANAGEMENT
TRAFFIC SHIFT (DEMO)



Three versions of the service of reviews

- Route to existing version of service
- Route part of the request to new version of service
- Route all the request to the new version of service

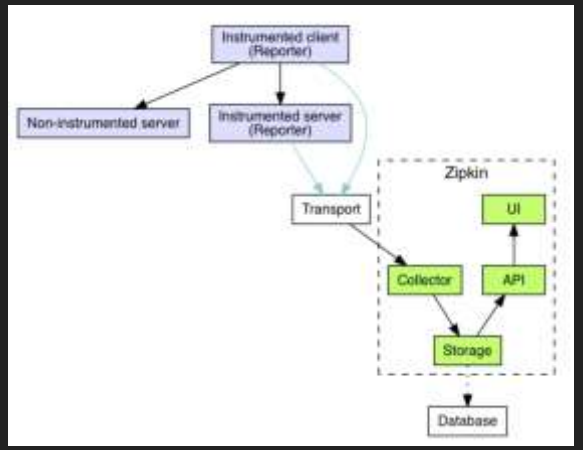
OBSERVABILITY - MIXER

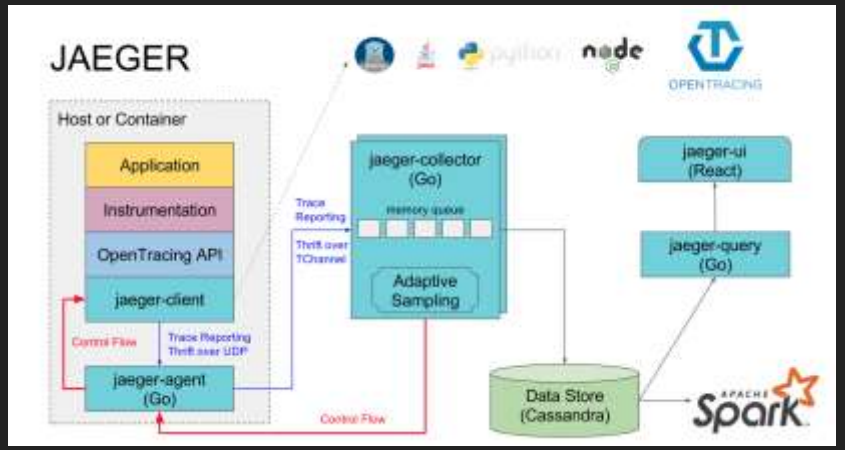


- Collect metrics and logs emitted by Envoys without instrumenting apps
- Provide a uniform abstraction between application and infra backend
- Adapters in the Mixer normalize and forward to backends (monitoring, billing …)
- Trace flow of requests across services
- Mixer is stateless with caching and buffering

Cited from https://www.infoq.com/presentations/istio-service-mesh

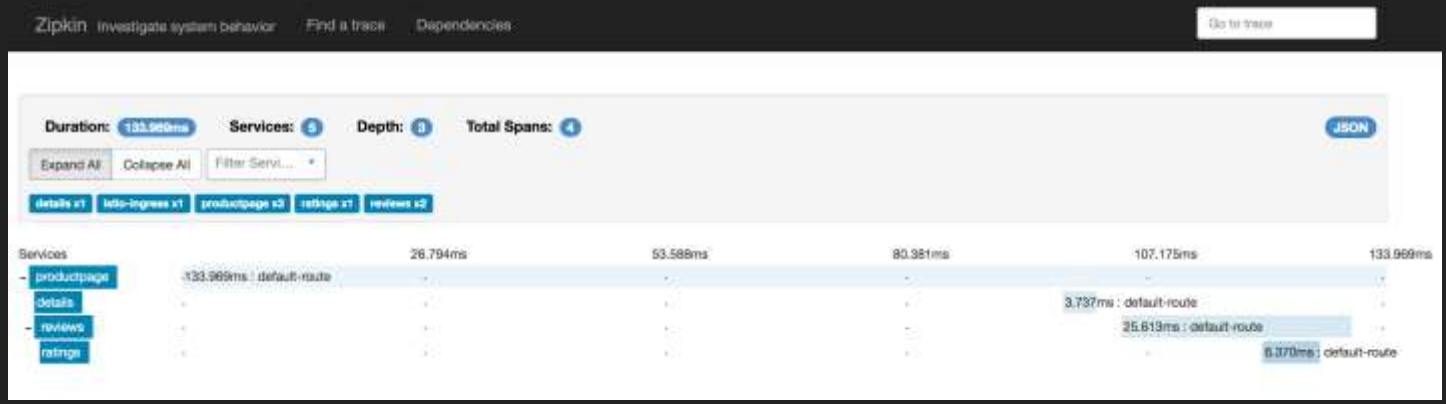
OBSERVABILITY
TRACING WITH ZIPKIN AND JAEGER





- Instrumentation
- Transport or agent
- Collector
- Storage
- API and UI

OBSERVABILITY
TRACING WITH ZIPKIN AND JAEGER (DEMO)

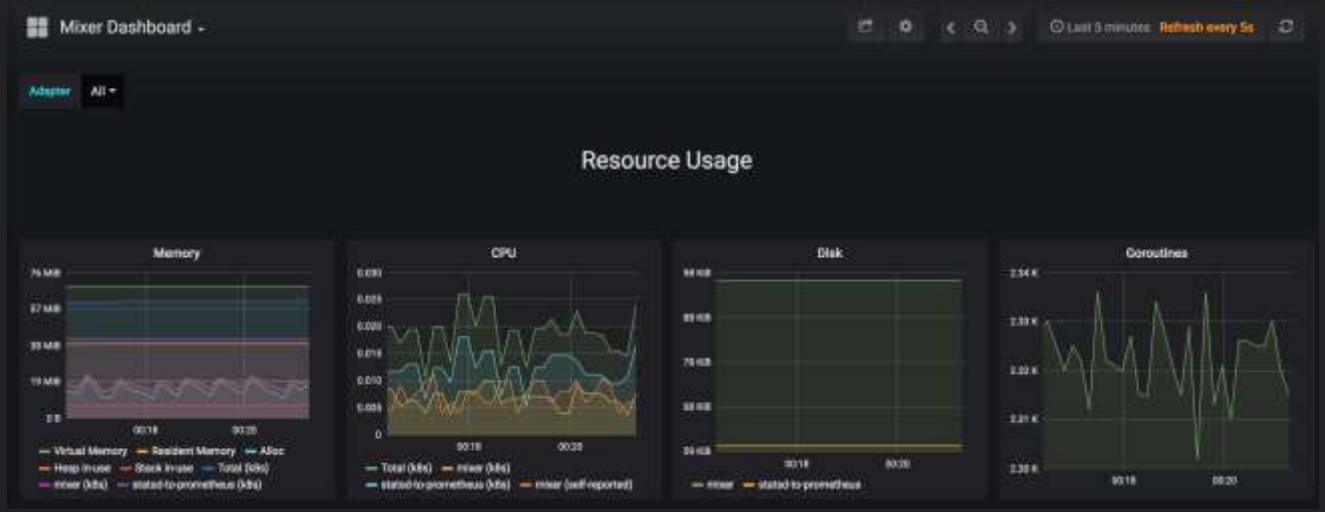


Application is required to collect and propagate the following headers from the incoming request to any outgoing requests

- x-request-id
- x-b3-traceid
- x-b3-spanid
- x-b3-parentspanid
- x-b3-sampled
- x-b3-flags
- x-ot-span-context

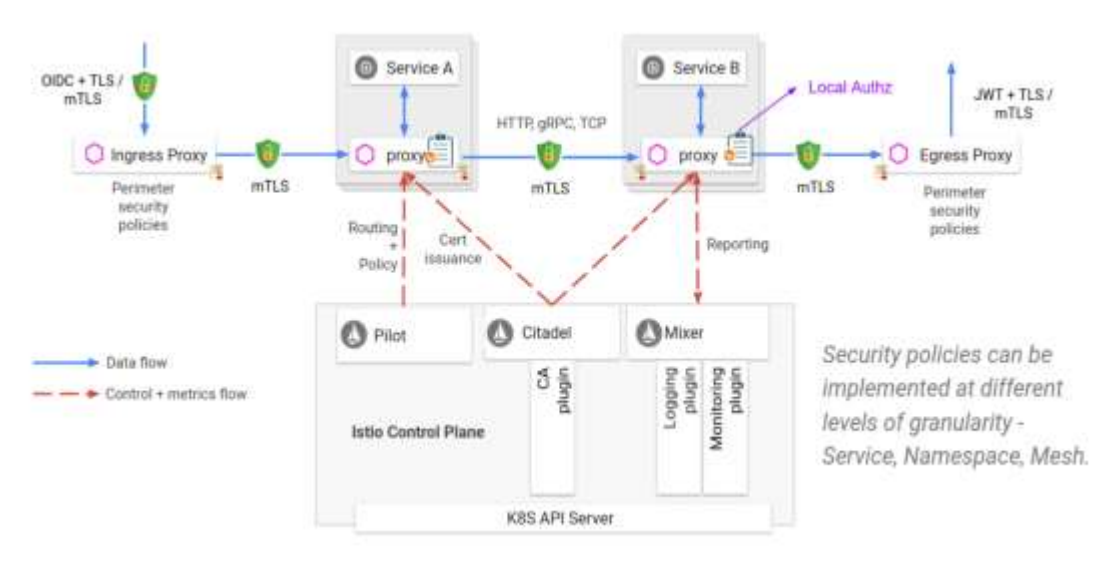
Zipkin has been replaced by Jaeger in Istio 0.8 and newer version

OBSERVABILITY
METRICS WITH MIXER + PROMETHEUS (DEMO)



- Generation of instances (in this example, metric values) from Istio attributes
- Creation of handlers (configured Mixer adapters) capable of processing generated instances
- Dispatch of instances to handlers according to a set of rules

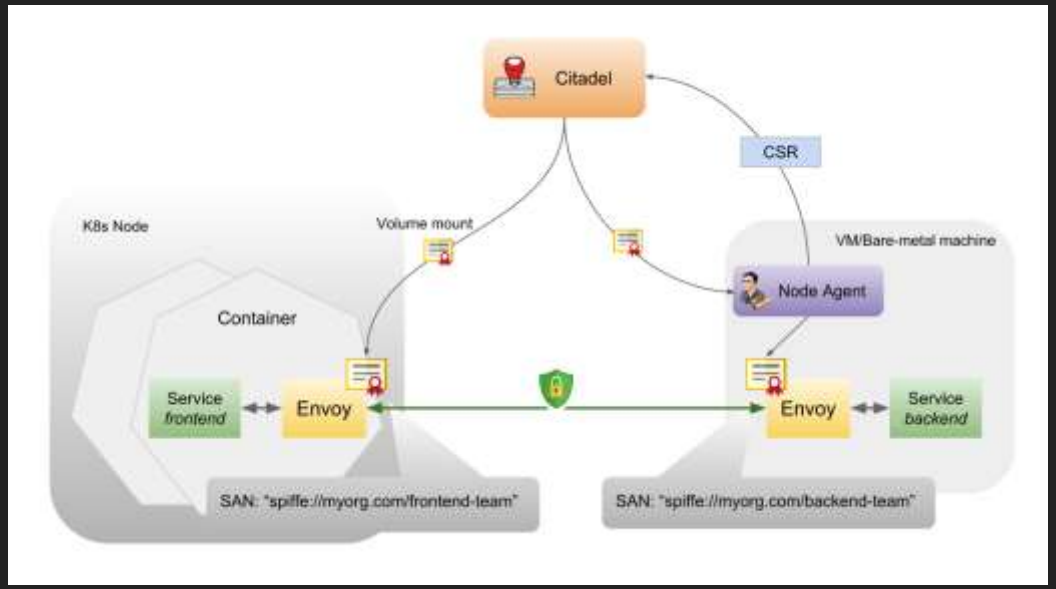
SECURITY



Need a management service to

- Secure microservices and their communication without instrumenting apps
  - ID management
  - Key/cert management
  - Access control

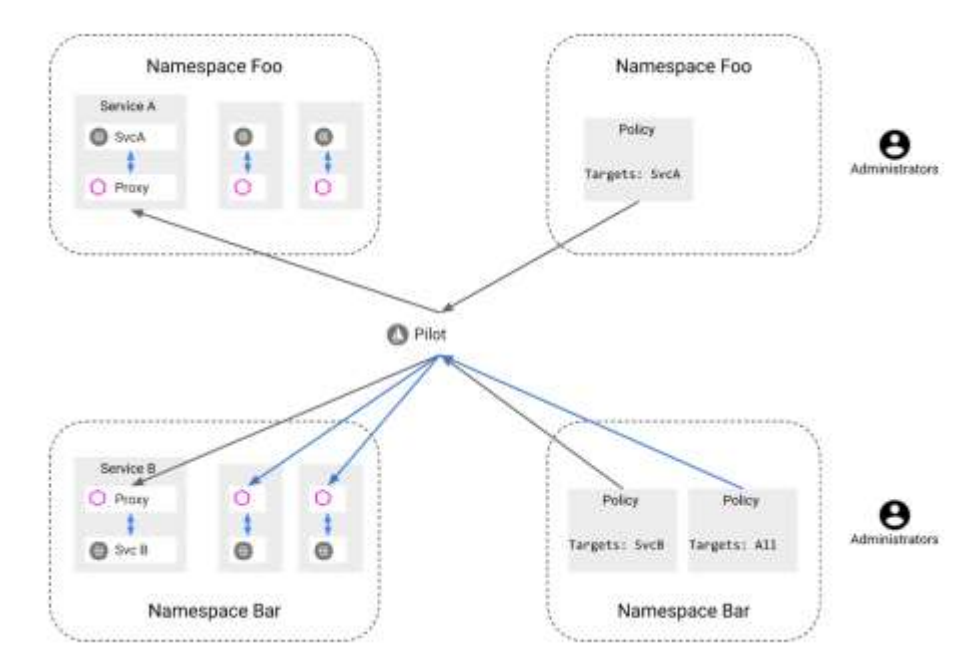
SECURITY
CERT MANAGEMENT - CITADEL



- Key management system
  - Automate key and certificate generation for service account
  - Distribute key/certs as kuberentes secret
  - Rotate keys/certs periodically
  - Revoke key/certs when required
  - Self-signed (root) CA vs user defined (root) CA
- Identity management
  - Identify service by service account in SPIFFE format

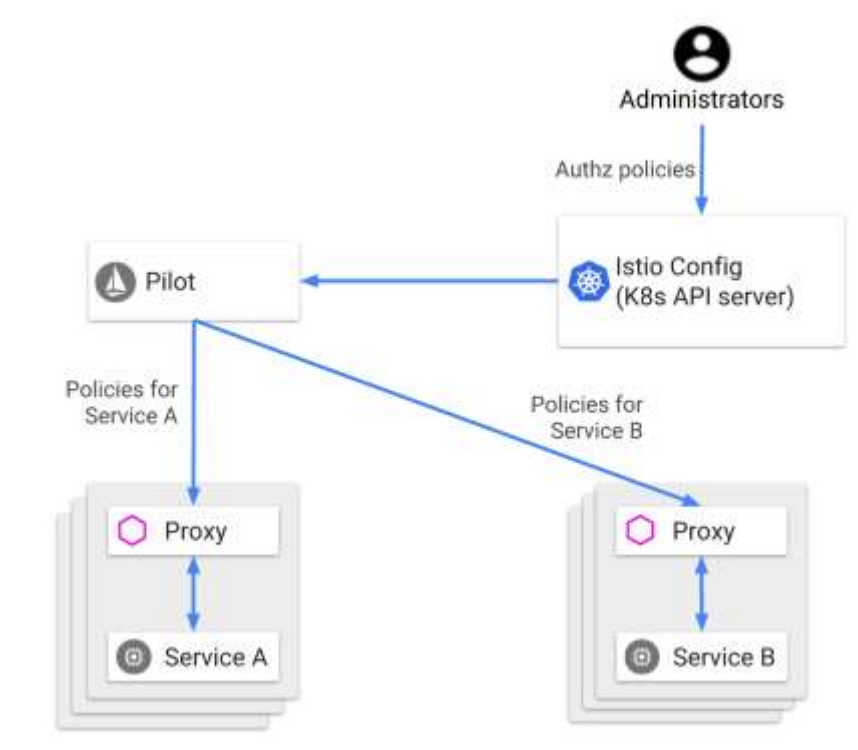
"spiffe://*<domain>*/ns/*<namespace>*/sa/*<serviceaccount>*"

SECURITY
ID MANAGEMENT - PILOT



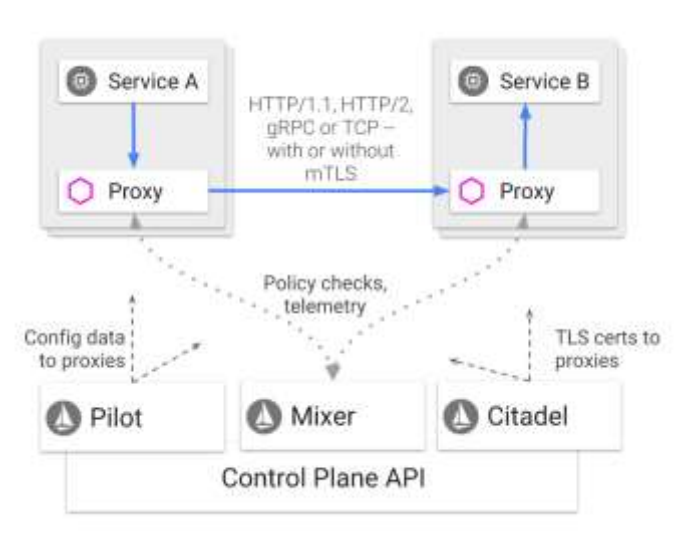
- Generate the config with proper information of key/certs and secure naming information, and then pass to envoy

SECURITY
ACCESS CONTROL - PILOT



- Access control between services
  - Rule based by defining “Rule” with “match”, “handler” and “instance”
  - RBAC based by defining “ServiceRole” with “action”, and defining “ServiceRoleBinding” with “subject” and “roleRef”

KUBERNETES INTEGRATION



- Inject sidecar by leveraging MutatingAdmissionWebhook
- Manage policy and configuration by leveraging Custom Resource Definitions
- Identify account by leveraging Service Account and Secret
- Identify service by leveraging Service Discovery

PERFORMANCE AND SCALABILITY

- 1 vCPU per peak thousand requests per second for the sidecar(s) with access logging (which is on by default) and 0.5 without, fluentd on the node is a big contributor to that cost as it captures and uploads logs.
- Assuming typical cache hit ratio (>80%) for mixer checks: 0.5 vCPU per peak thousand requests per second for the mixer pods.
- Latency cost/overhead is approximately 10 millisecond for service-to-service (2 proxies involved, mixer telemetry and checks) as of 0.7.1, we expect to bring this down to a low single digit ms.
- mTLS costs are negligible on AES-NI (aes in /proc/cpuinfo) capable hardware in terms of both CPU and latency.

USEFUL LINKS

- Web istio.io
- Twitter: @Istiomesh
- Github: https://github.com/istio/istio
- Community Doc: https://istio.io/docs
- Traffic management using Istio: https://ibm.co/2F7xSnf
- Resiliency and fault-tolerance using Istio: https://bit.ly/2qStF2B
- Reliable application roll out and operations using Istio: https://bit.ly/2K9IRQX