



Introduction to Istio



+



Leon Jalfon

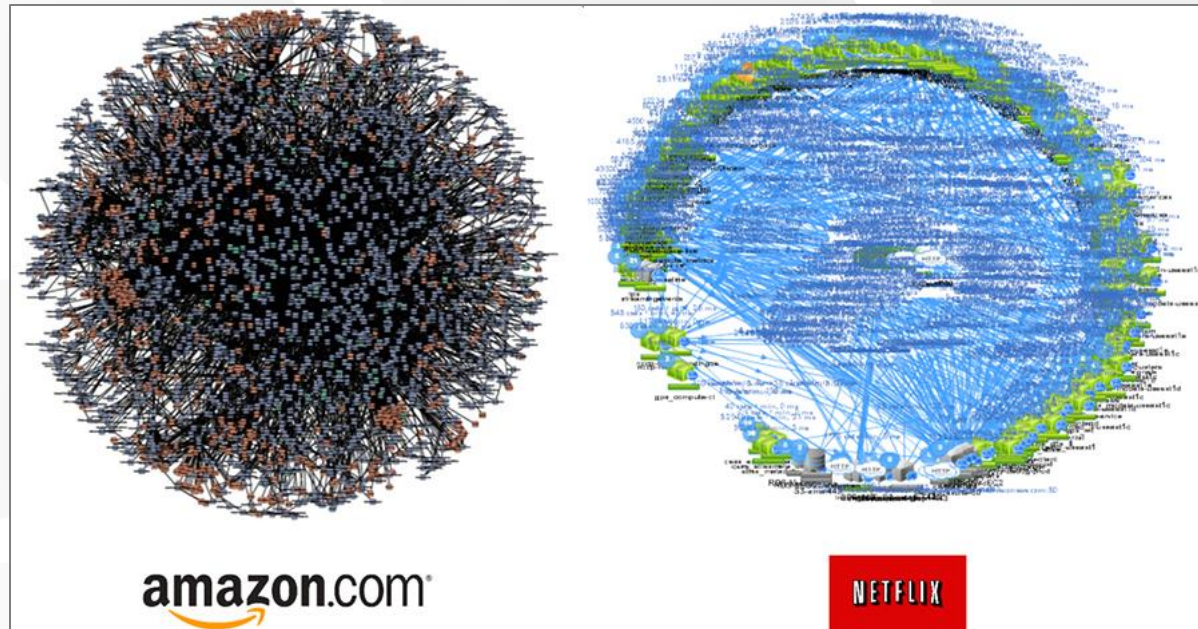
DevOps & Cloud Architect


Agenda

- ✦ Microservices Architecture
- ✦ Service Mesh
- ✦ Istio Architecture
- ✦ Main Features
- ✦ Demo

Microservices

- ✦ As application grow the communication between services becomes more complex.





Micro-services applications are not that simple

With simplicity, comes complexity

- ✦ How to deploy or update services with zero-downtime?
- ✦ How to A/B test the application?
- ✦ How to handle network failures?
- ✦ How to manage security between services?
- ✦ How to handle timeouts? Retries?
- ✦ How to rate limit? Add quotas?
- ✦ Telemetry, Logging, Monitoring?
- ✦ What about run multiple versions at the same time (canary)?
- ✦ Different Tech Stacks

Service Mesh

- The term service mesh is used to describe the network of microservices that make up such applications and the interactions between them.
- Service Meshes are taking care of all communication and policies needs between services and allows extensibility by middleware's
- Its requirements can include service discovery, load balancing, failure recovery, metrics, and monitoring.



Istio

- Initiative from Google, IBM and Lyft
- Built for Kubernetes (but also supports Nomad and Consul)
- Provides a uniform way to connect, manage, and secure microservices
- It supports managing traffic flows between services, enforcing access policies, and aggregating telemetry data, all without requiring changes to the microservice code
- Istioctl – like Kubectl, only for Istio (we use Kubectl most of the time)

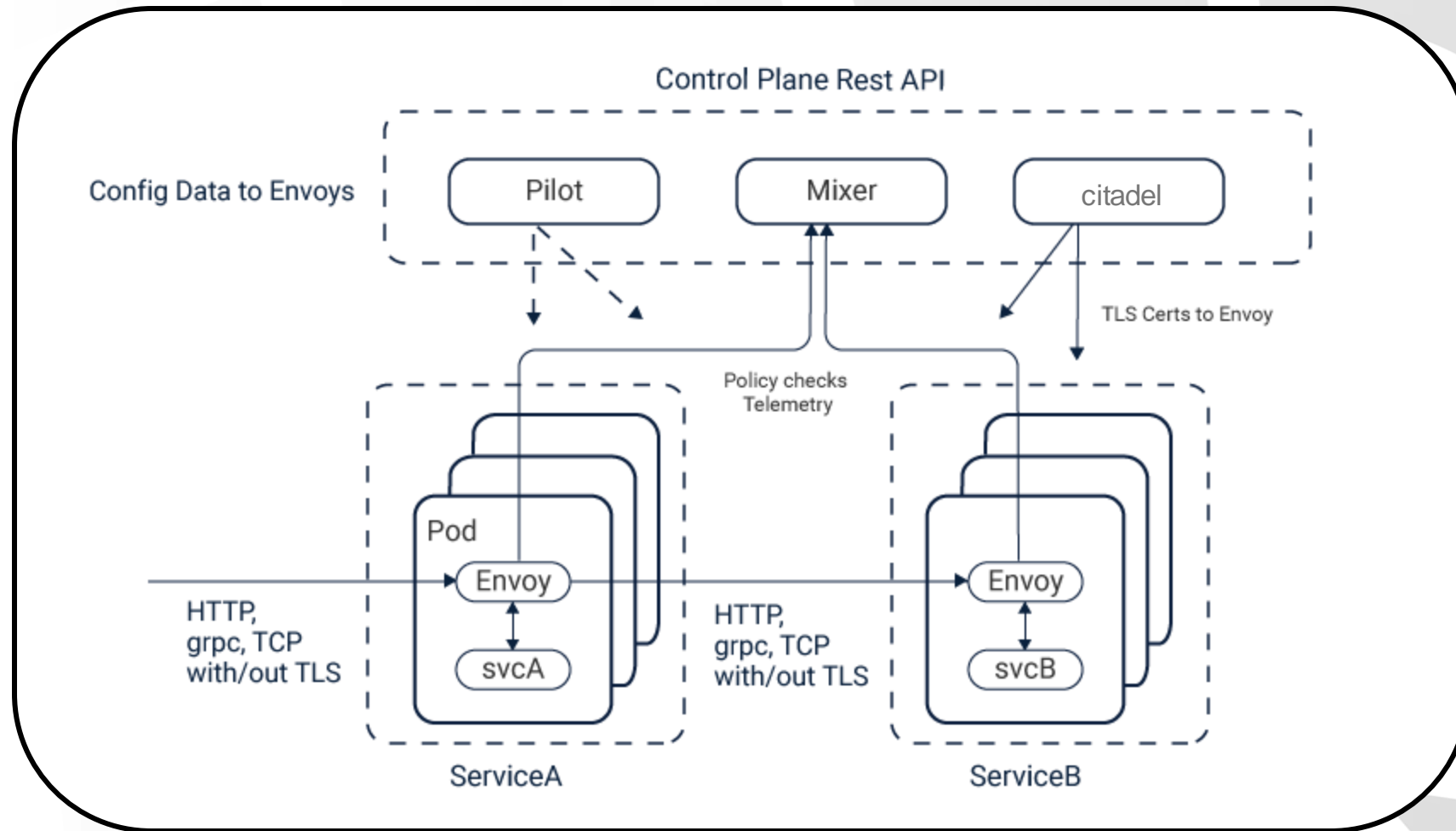
Why Istio?

- 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.

Architecture (data plane & control plane)

- The **data plane** is 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.
- The **control plane** manages and configures the proxies to route traffic. Additionally, the control plane configures Mixers to enforce policies and collect telemetry.

Architecture



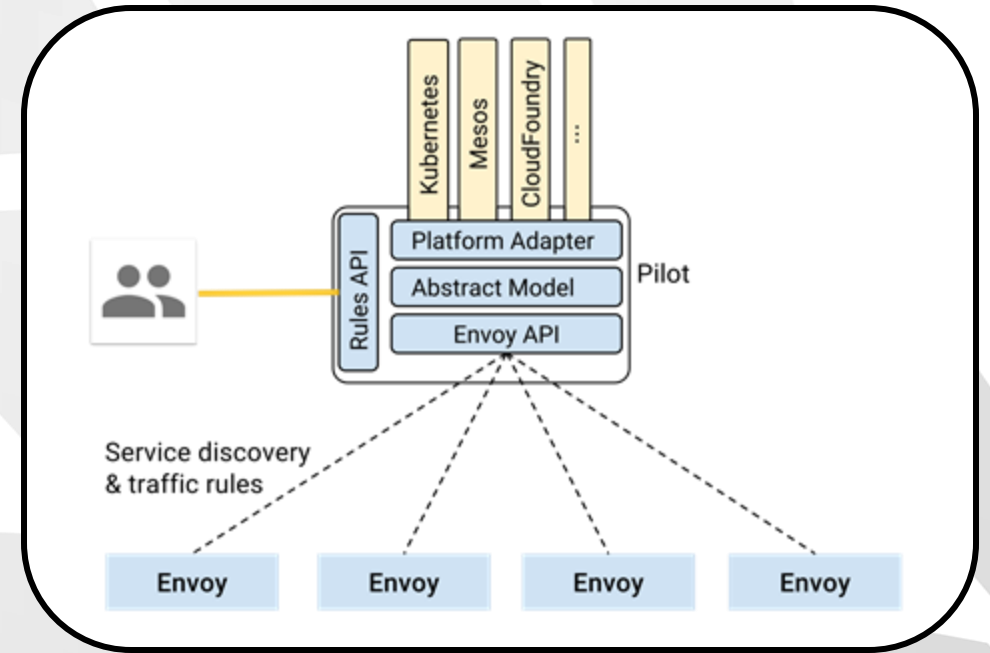
Architecture (envoy)



- Originally built at Lyft
- CNCF Graduate
- A C++ based L4/L7 proxy
- Battle-tested with great performance
- Acts as the smart Data-Plane managed by Istio
- Many built-in mechanism used by Istio
- API Driven updates (without hot-reload)
- Injected as a side-car

Architecture (pilot)

- Provides traffic management capabilities for intelligent routing
- Provides service discovery for Envoy sidecars
 - A/B tests
 - canary deployments
- Provides resiliency
 - Timeouts
 - Retries
 - circuit breakers



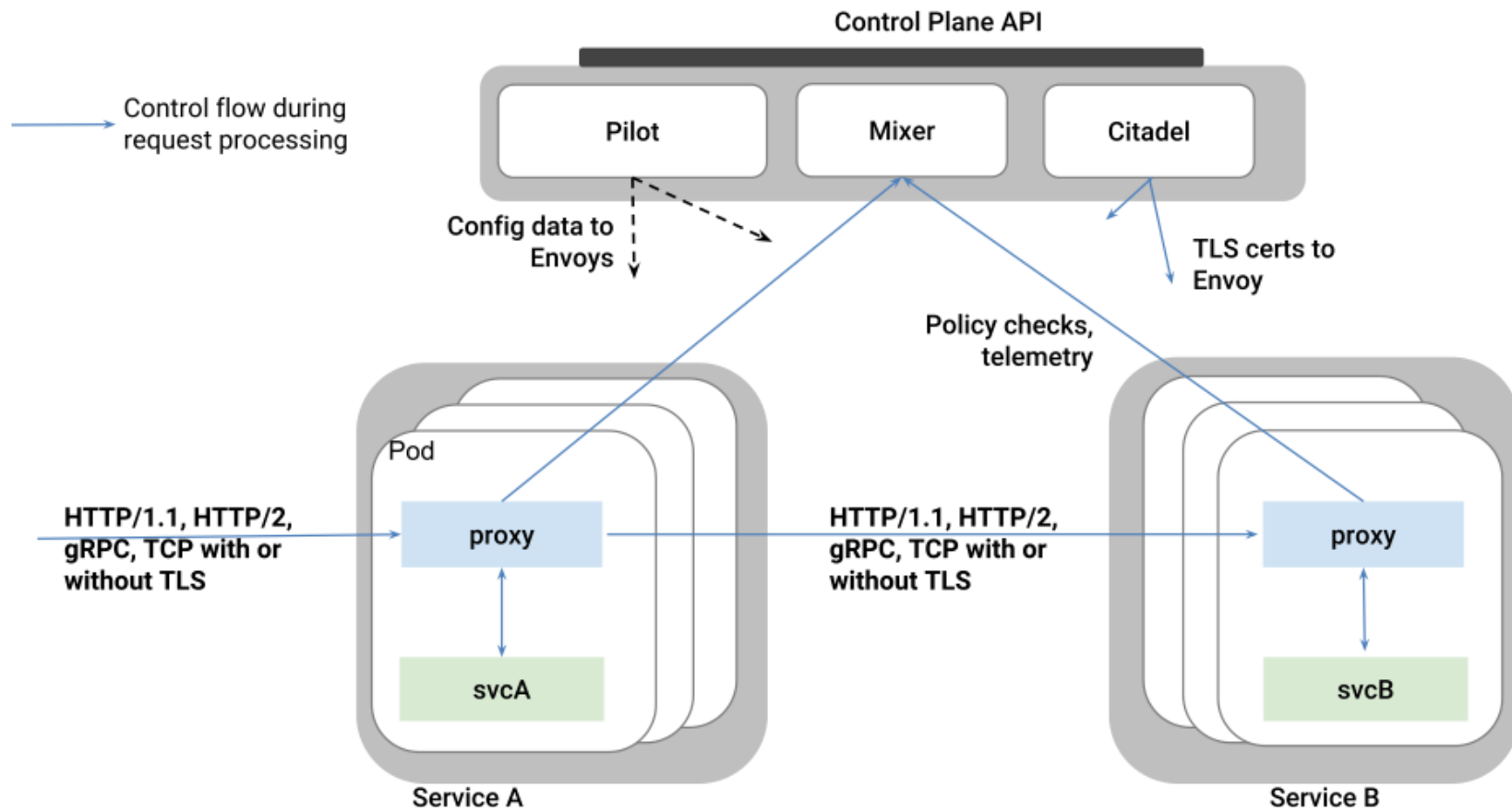
Architecture (mixer & citadel)

- **Mixer**

- Manages Access Control and Policies
- Extract request attributes
- Collects Telemetry and metrics

- **Citadel**

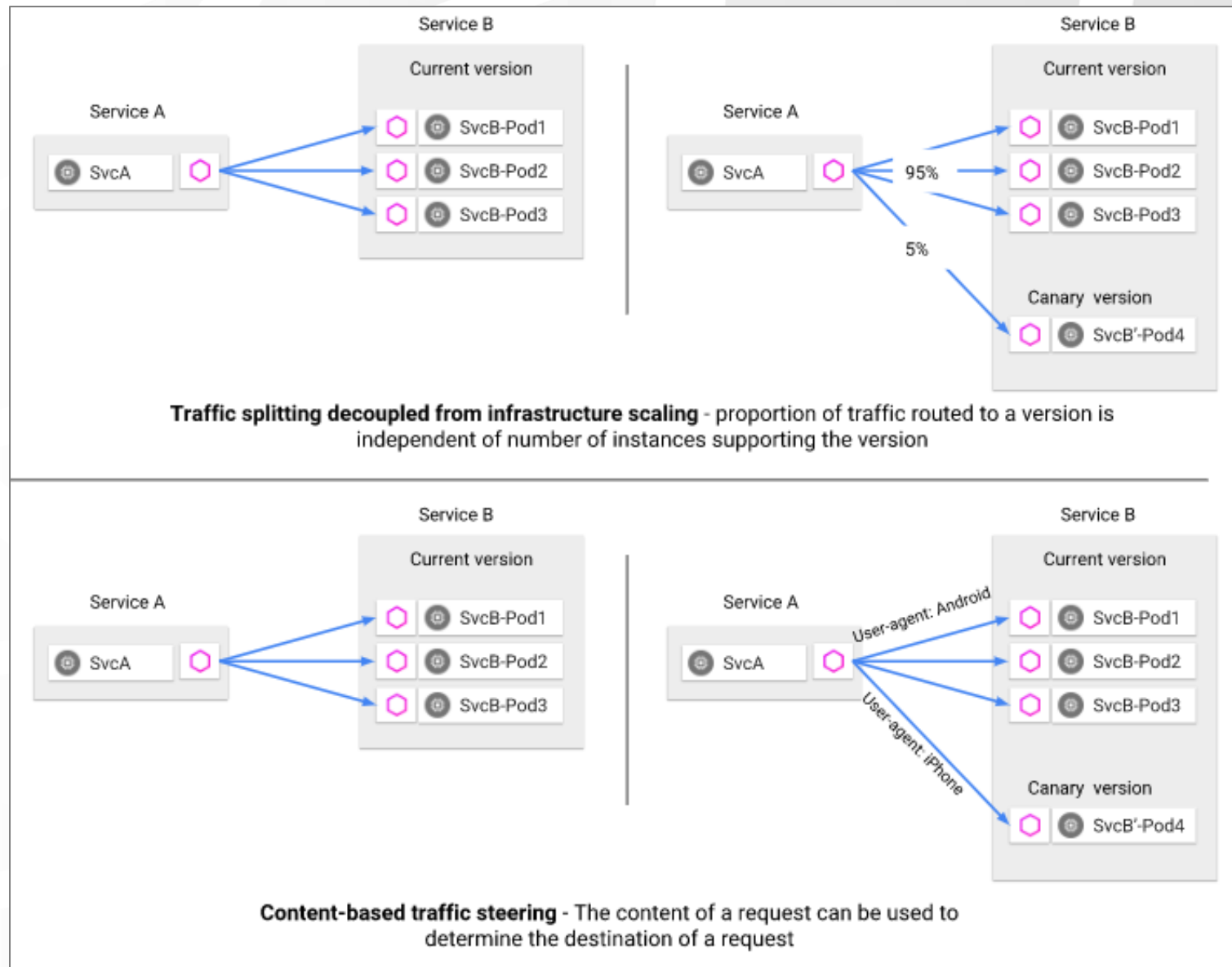
- Service-to-service authentication and Mutual TLS
- Supports RBAC (Role-Based Access Control) - like Kubernetes
- Automatically manages credentials and certificates



Intelligent Routing Capabilities

Request Routing

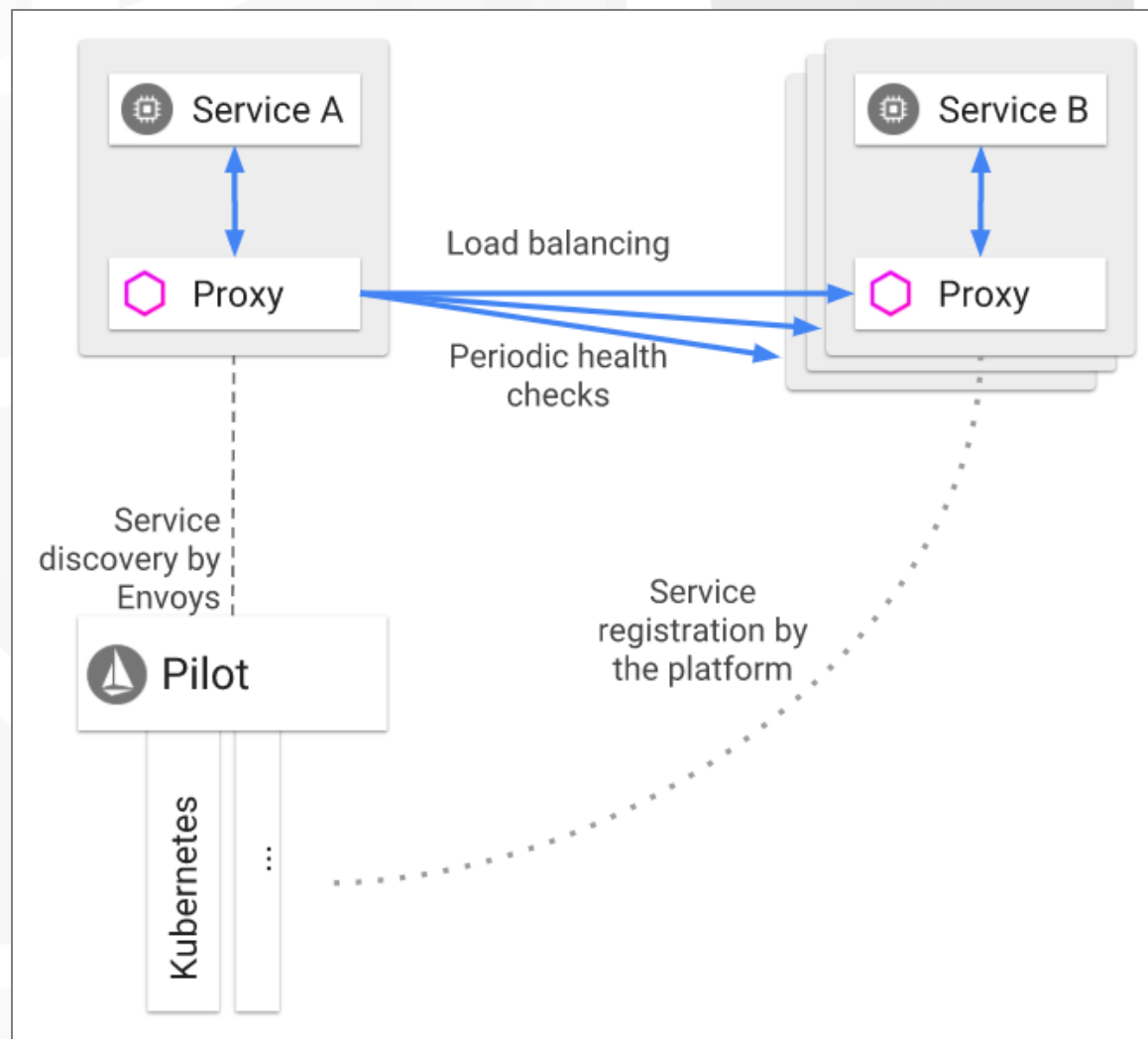
- Manage multiple environments (dev, test, prod) and multiple versions (vX, vY) at the same time while configuring sophisticated rules based-on Uri, Headers and more.
- Implement Weight-based version routing
- Allows A/B testing and Canary Deployments
- Handle Ingress and Egress routing rules and gateways
- Warm-up services with request mirroring



Intelligent Routing Capabilities

Load Balancing

- Handle service-registration and service-discovery
- Advanced Algorithms
 - Weighted round robin
 - Weighted least request
 - Ring-Hash
 - Maglev
 - Random
 - Orig-Destination
- Zone-awareness, priorities and more



Failure Handling

Timeouts and Deadlines

- Following request journey in the Service Mesh
- Supports per-request configuration

Retries

- Supports variable jitter between retries

Rate-limiting and Quotas

- Connection limits, requests throttling

Circuit-Breaker

- Help getting failed services back to shape after subsequent failures (fully configurable)

Fault Injection

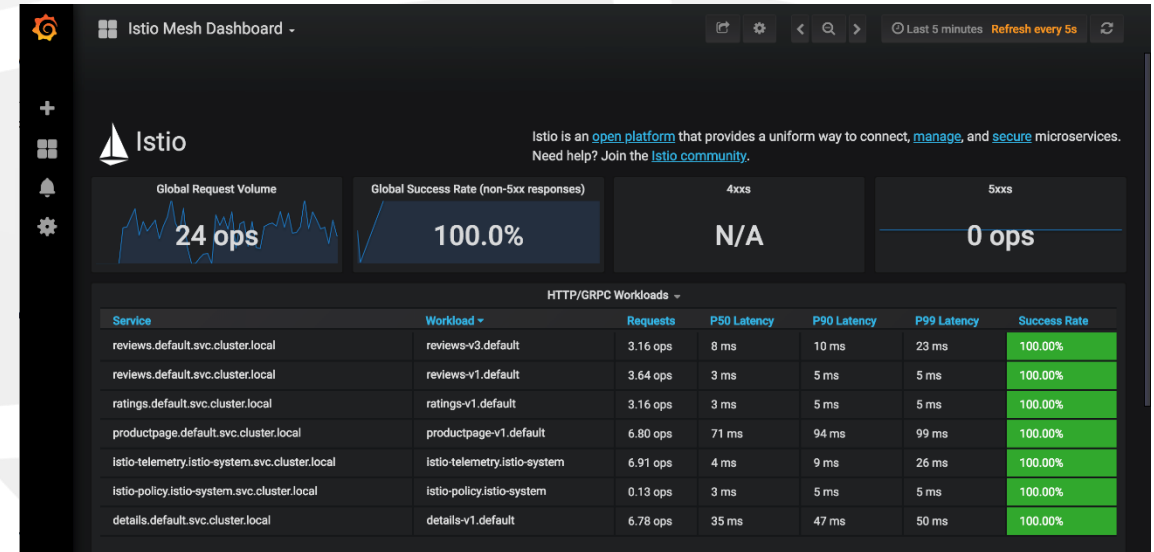
- Allows to test the failure handling mechanism
- Supports Chaos testing (i.e Netflix's Chaos Monkey)
- Introduce latency to specific services or users
- Inject statistical errors to requests

Security

- Authentication
 - Transport authentication (service-to-service authentication)
 - Origin authentication (end-user authentication)
- Mutual TLS authentication
- Permissive mode
 - Allows a service to accept both plain text traffic and mutual TLS traffic at the same time
- Secure naming
- Authentication policies

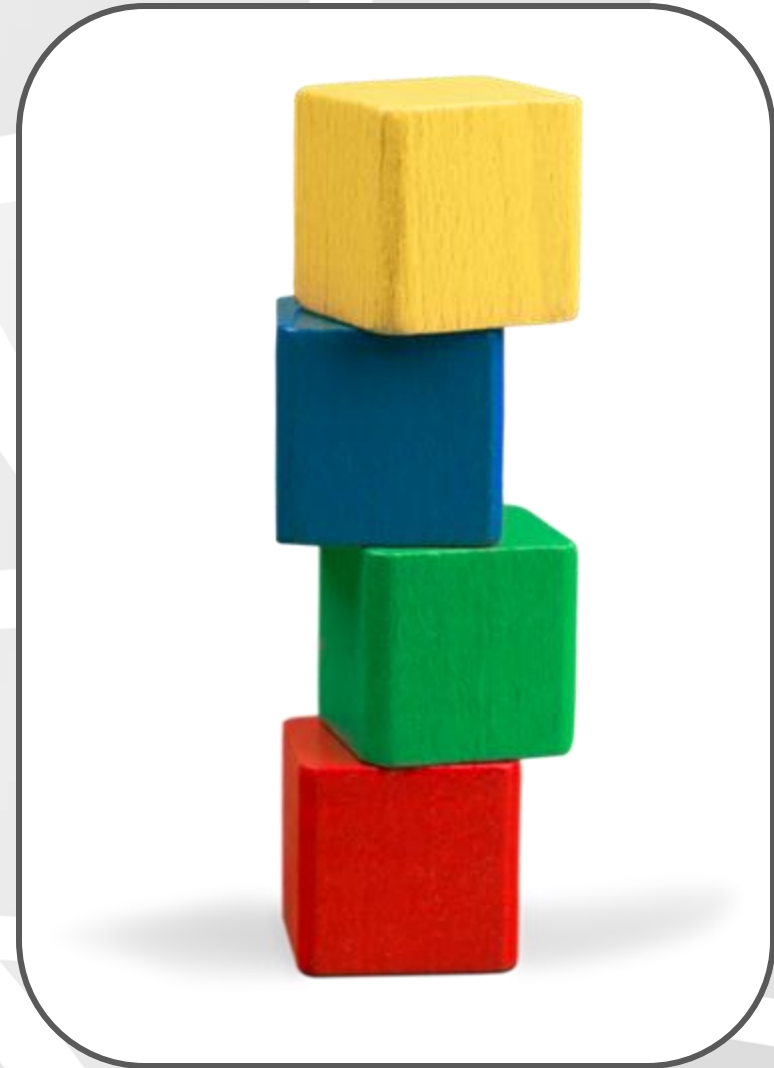
Observability

- Monitoring
 - Using Prometheus and Grafana
- Visibility
 - Using Kiali
- Tracing
 - Using Jaeger / Zipkin
- Attributes
 - Is a small bit of data that describes a single property of a specific service request or the environment for the request



Istio Building Blocks

- Gateway
- VirtualService
- DestinationRule
- etc



Gateway

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gateway
spec:
  selector:
    istio: ingressgateway    # use istio default controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

Gateway describes a load balancer operating at the edge of the mesh receiving incoming or outgoing HTTP/TCP connections

VirtualService

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - "*"
  gateways:
    - bookinfo-gateway
  http:
    - route:
        - destination:
            host: reviews
            subset: v1
```

A VirtualService defines the rules that control how requests for a service are routed within an Istio service mesh

DestinationRule

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: reviews
spec:
  host: reviews
  subsets:
    - name: v1
      labels:
        version: v1
    - name: v2
      labels:
        version: v2
    - name: v3
      labels:
        version: v3
```

DestinationRule defines policies that apply to traffic intended for a service after routing has occurred.

Questions



Introduction to Istio

Demo



<https://github.com/leonjalfon1/sdp-istio>



Thank you for your attention!

Introduction to Istio

Leon Jalfon

leonj@sela.co.il