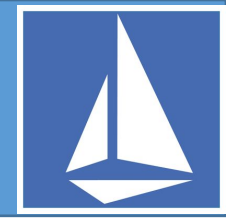


ISTIO

SERVICE MESH

Karim AMMOUS and Fabrice JAMMES

Agenda

- Introduction
- Service mesh
- Istio
  - Architecture
  - Installation
- Core features
  - Connect
  - Secure
  - Control
  - Observe
- Demo

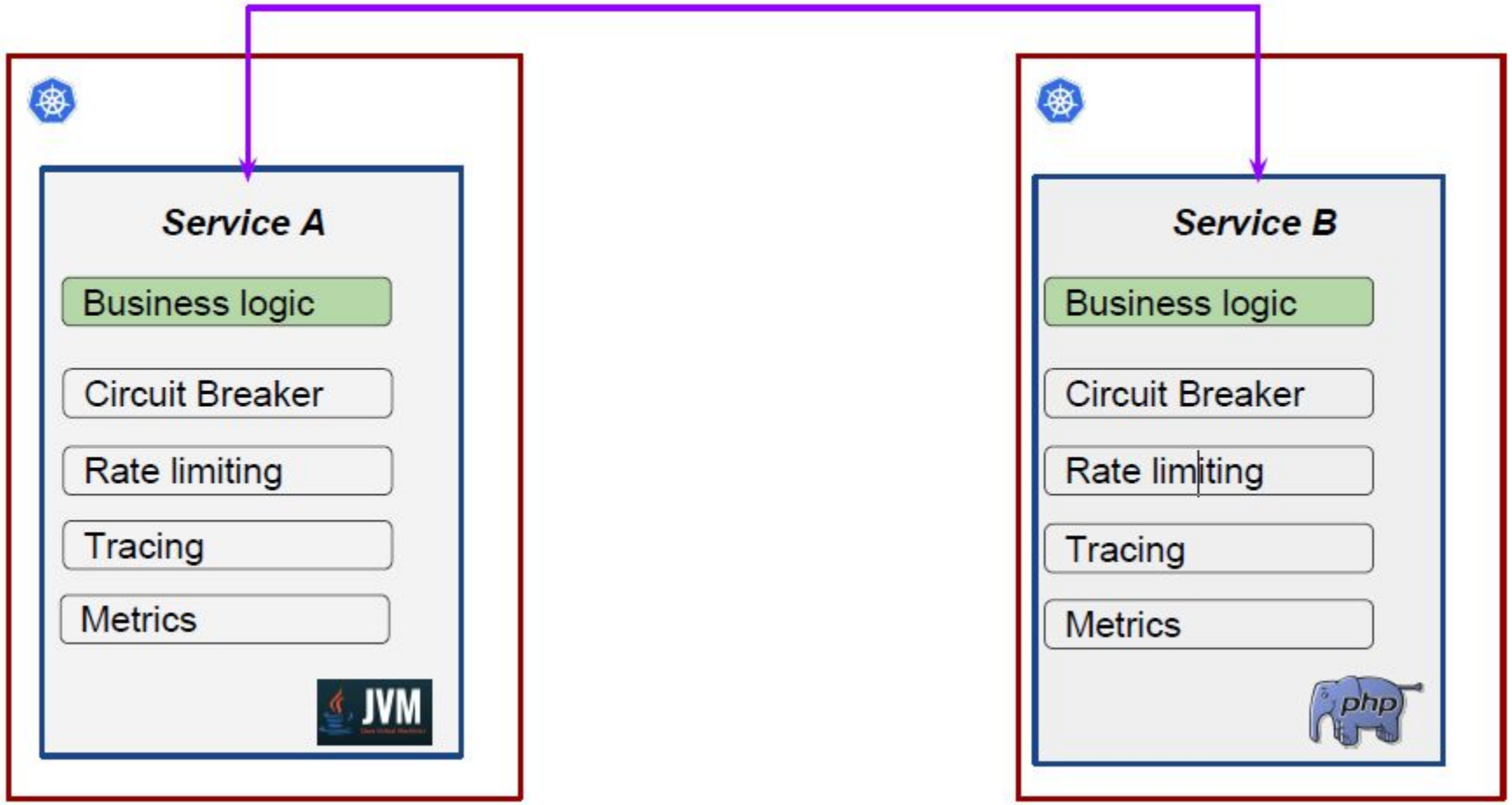
Introduction: Microservices Challenges

The Eight fallacies of Distributed Systems

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

- Network Reliability
  - Network latency / bandwidth
  - Transport cost
  - Topology and administration
- Fault Tolerance
  - Avoid cascading failure
  - Retries
  - Circuit breaking
- Monitoring
  - Services interactions
  - Trace requests and identify potential hotspots

Introduction: Code Oriented Solution



Introduction: 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 team in the same organization may have different implementation.
- Each team should maintain his implementation.

Service mesh: Definitions

- **A service mesh** is a configurable infrastructure layer for microservices application that makes communication flexible, reliable, and fast.

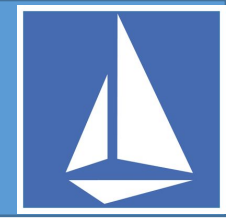
Buoyant (Linkerd vendor)

- **A service mesh** provides a transparent and language-independent way to flexibly and easily automate application network functions.

Service mesh: Why?

- Need to add the following **capabilities** to our services **without requiring changes** to the underlying **services**:
  - automated baseline traffic resilience,
  - service metrics collection,
  - distributed tracing,
  - traffic encryption,
  - protocol upgrades,
  - advanced routing functionality

Service mesh: Implementations



- Linkerd by Buoyant

- Conduit (joined Linkerd2) by Buoyant

- Consul Connect by HashiCorp

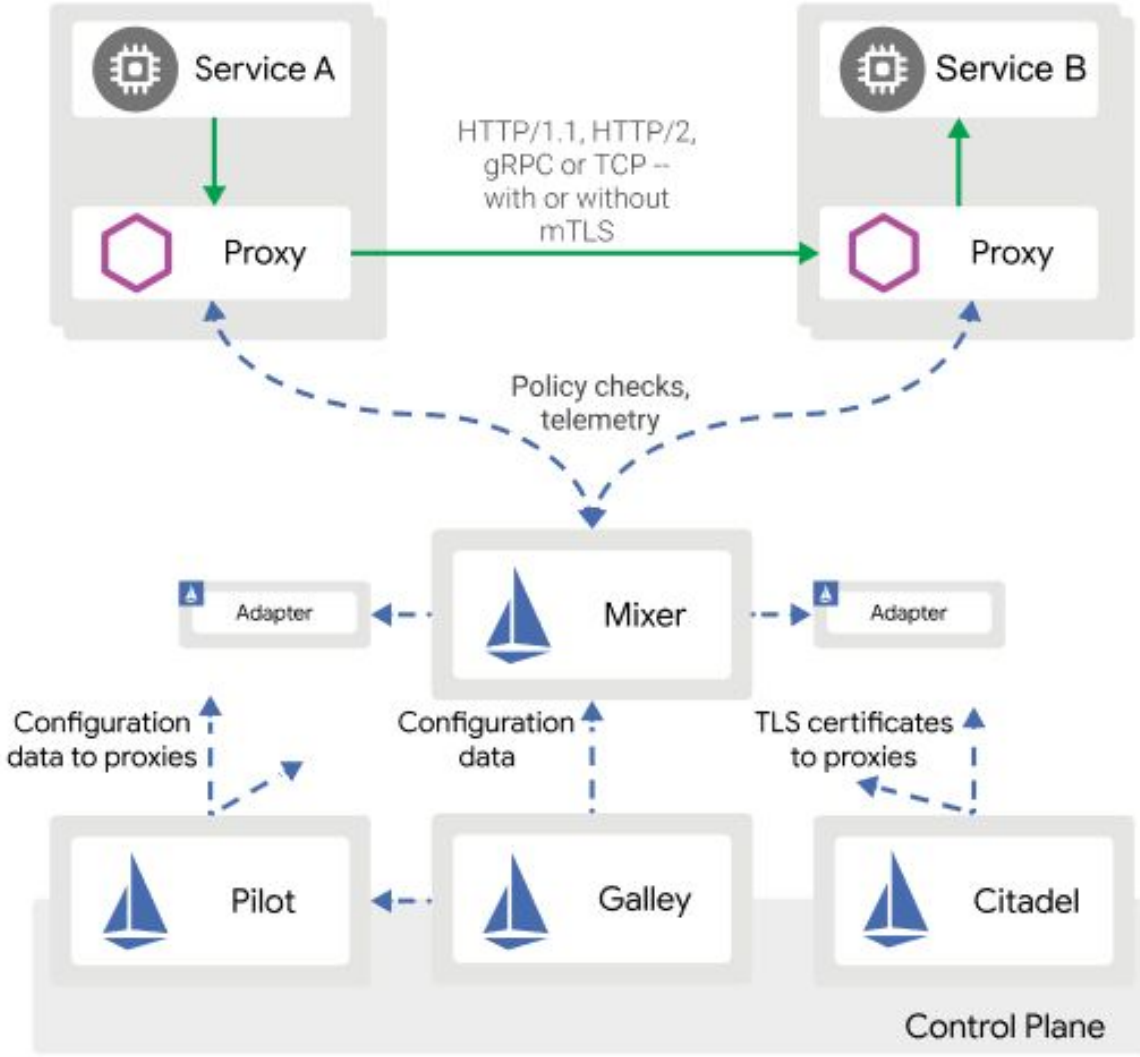
- Istio

ISTIO

Istio: What is it ?

- Started by teams from **Google** and **IBM** in partnership with the **Envoy** team from **Lyft**
- Open source under **Apache License 2.0** license.
- Platform-independent: **Kubernetes** (v1.9 or greater) and Nomad (with **Consul**)
- First major version released in **July 2018**

Istio: Architecture



Envoy: Sidecar Network proxy to intercept communication and apply policies.

Pilot: Control plane to configure and push service communication policies.

Mixer: Provides telemetry collection as well as sophisticated policy checks.

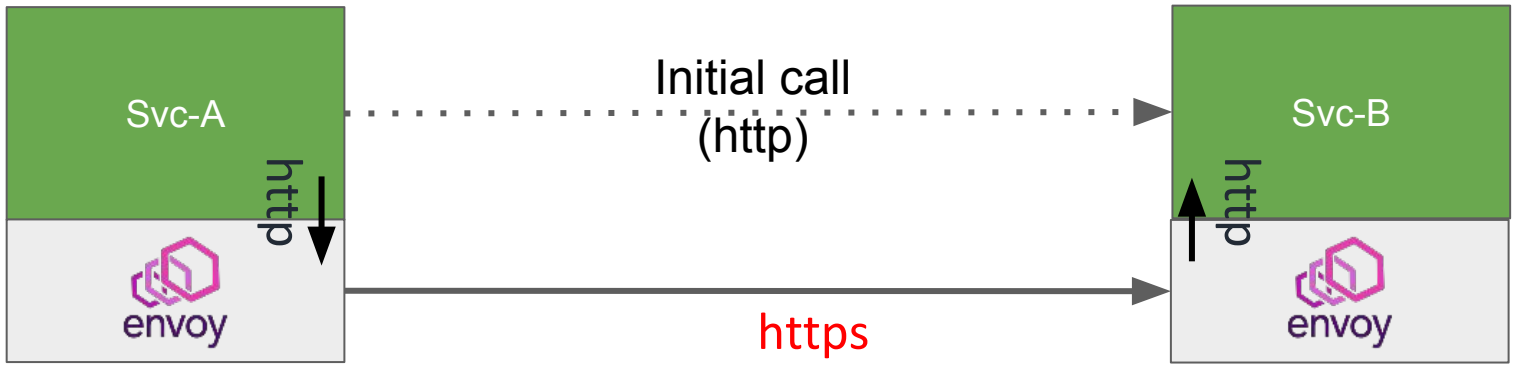
Citadel: Service-to-service auth[n,z] using mutual TLS, with built-in identity and credential management.

Galley: Configuration validation, distribution

Istio: Installation

See https://istio.io/docs/setup/getting-started/

Istio: How It Works?

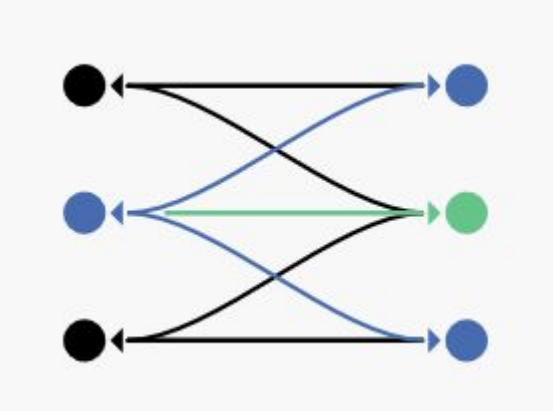


Istio: Sidecar Injection

- The port names MUST be of the form protocol-suffix in order to take advantage of Istio's routing features
  - Protocols example: http, http2, grpc, mongo, or redis.
  - Unrecognized prefix or unnamed ports => port is treated as plain TCP traffic.
  - UDP is not supported yet (due to Envoy)
- **Manual injection**: The sidecar Envoy proxy is injected manually (using Istioctl CLI) into yaml files before deployment
- **Automatic injection**: If enabled (requires Kubernetes 1.9 or later), the sidecar proxy is injected seamlessly.

CORE FEATURES

Istio capabilities

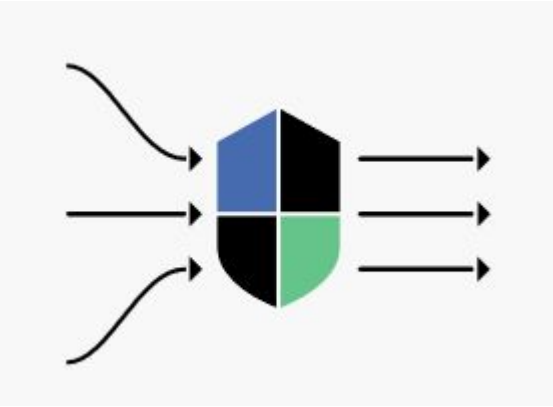


Connect

Istio can intelligently control the flow of traffic between services, conduct a range of tests and upgrade gradually with blue/green deployments.

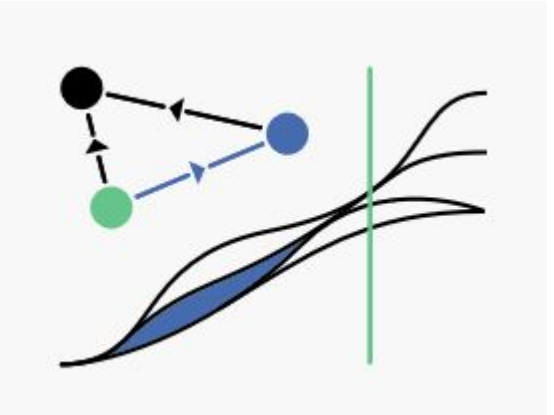
Secure

Automatically secure your services through managed authentication, authorization, and encryption of communication between services.



Control

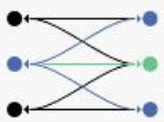
Apply policies and ensure that they are enforced and that resources are fairly distributed among consumers.



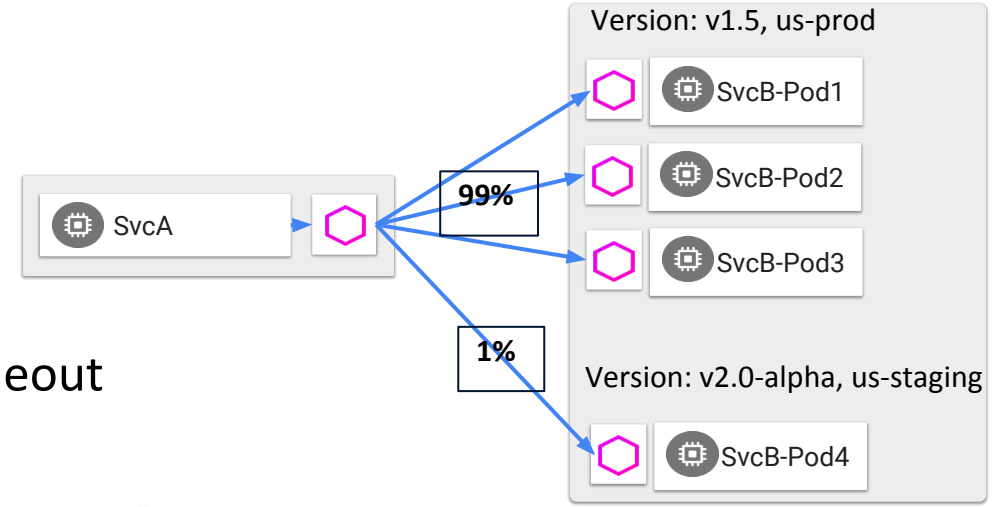
Observe

See what's happening with rich automatic tracing, monitoring, logging of all your services.

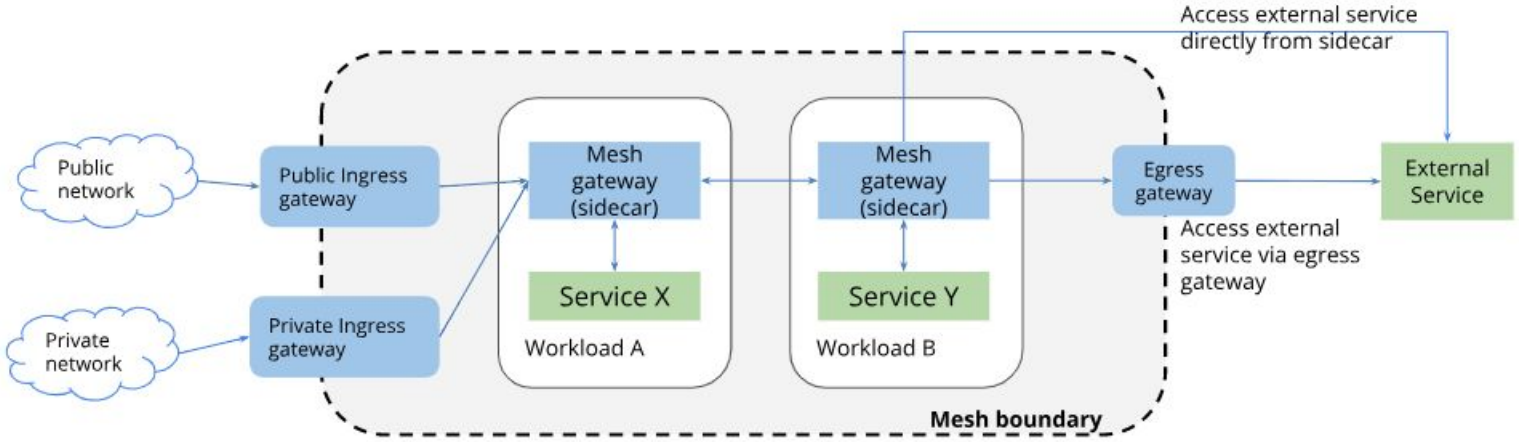
Connect: Traffic Management



- Service discovery and load balancing (client side)
- Request Routing, Traffic Shifting
- Resiliency: Retry, Fault Injection, Circuit breaking, Timeout



Canary deployment

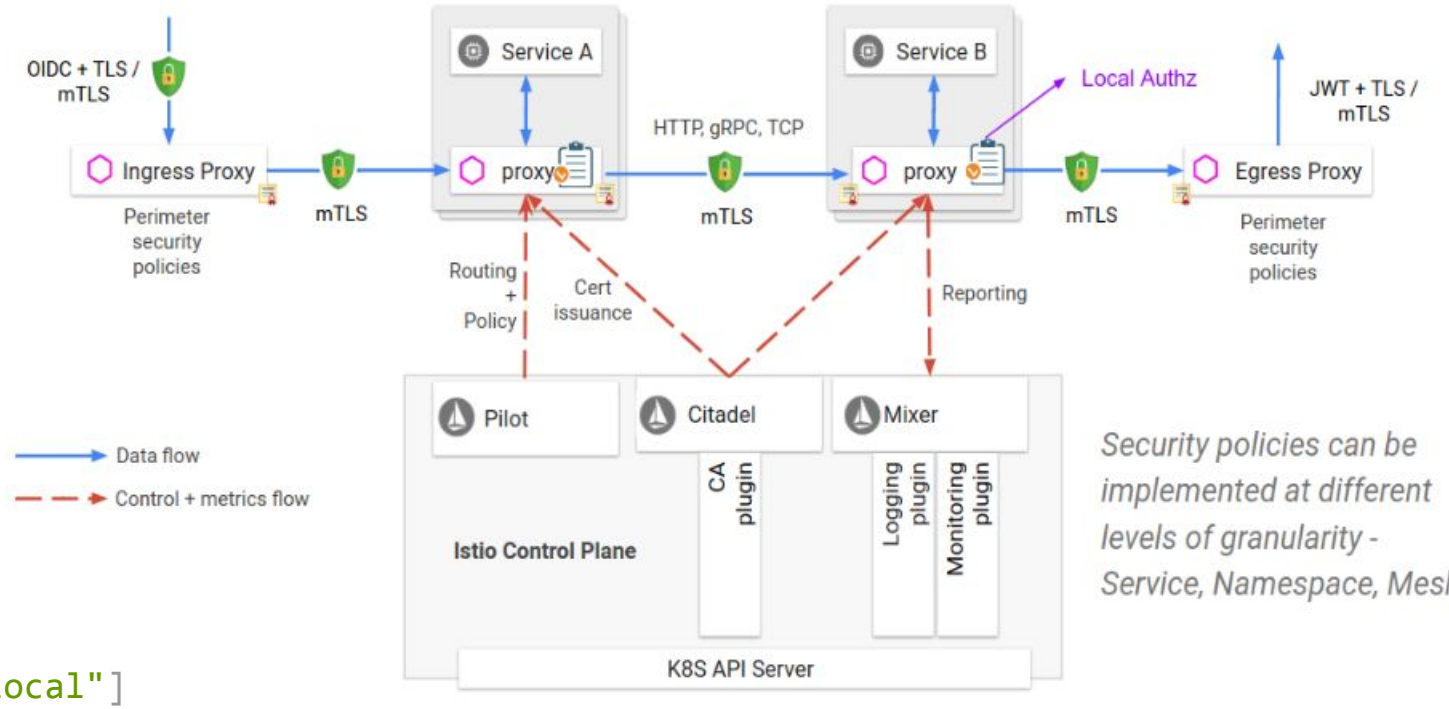


Ingress and Egress Gateways

Secure: TLS

- Authentication: Mutual TLS
- Authorization: Could be ON or OFF per service or per namespace
- Three access control levels:
  - namespace
  - service
  - method
- CRD: ServiceRole and ServiceRoleBinding

```
apiVersion: "rbac.istio.io/v1alpha1"
kind: ServiceRole
metadata:
  name: products-viewer
  namespace: default
spec:
  rules:
  - services: ["products.default.svc.cluster.local"]
    methods: ["GET", "HEAD"]
```



Istio Security Architecture

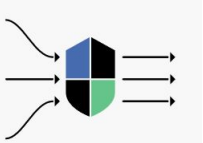
Security policies can be implemented at different levels of granularity - Service, Namespace, Mesh.

Secure: Authorization & Network Policy

- Istio has no requirement on the underlying CNI
- Network Policy could be used in concert with Istio

	Istio Authorization	Network Policy
Layer	“RPC” — L7	“Network” — L3-4
Based on	Envoy proxy	Iptables
Particularity	Flexible	Universal, Fast

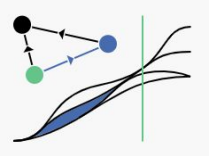
Control



Control where and how requests flow, and which requests are allowed.

- Fine grained traffic control
  - L7, not L4!
  - Route by headers, destination or source ID, etc
- Policy on requests
  - Authn/z, rate limiting, arbitrary policy based on L7 request metadata

Observe: Metrics, tracing and logging



Monitoring is a MUST:

- RED Method: Rate, Errors and Duration of requests.
- USE Method: Utilization, Saturation and Errors of resources
- The Four Golden Signals: Latency, Traffic, Errors and Saturation

We can understand what's actually happening in our deployment thanks to:

- Metrics (Prometheus and Grafana)
- Logs (Kibana)
- Tracing (Jaeger)
- Service Graph

Performance benchmark

See [Linkerd Benchmarks]

DEMO

See https://github.com/k8s-school/istio-example

And https://istio.io/docs/examples/bookinfo/

Demo: Bookinfo Application