Istio – An introduction for developers

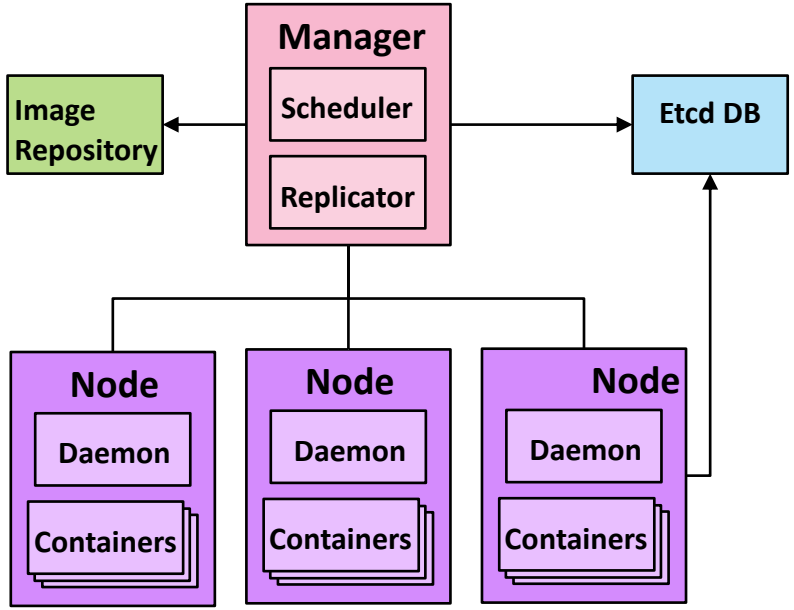
WW Developer Advocacy Team

IBM **Developer**

IBM

Benefits of Kubernetes

- Automated scheduling and scaling
- Zero downtime deployments
- High availability and fault tolerance
- A/B Testing



Microservice Challenges

Controlling Traffic

1. How do I do canary testing?
2. How do I A/B testing?

Resilient Services

1. How do I implement circuit breakers?
2. How do I test faults in the system?
3. How do I limit set rate limits for each service?

Telemetry

1. How can I trace a request through my system of multiple services
2. How can I perform monitoring in a distributed deployment?

Security

1. How can I apply policies across services?

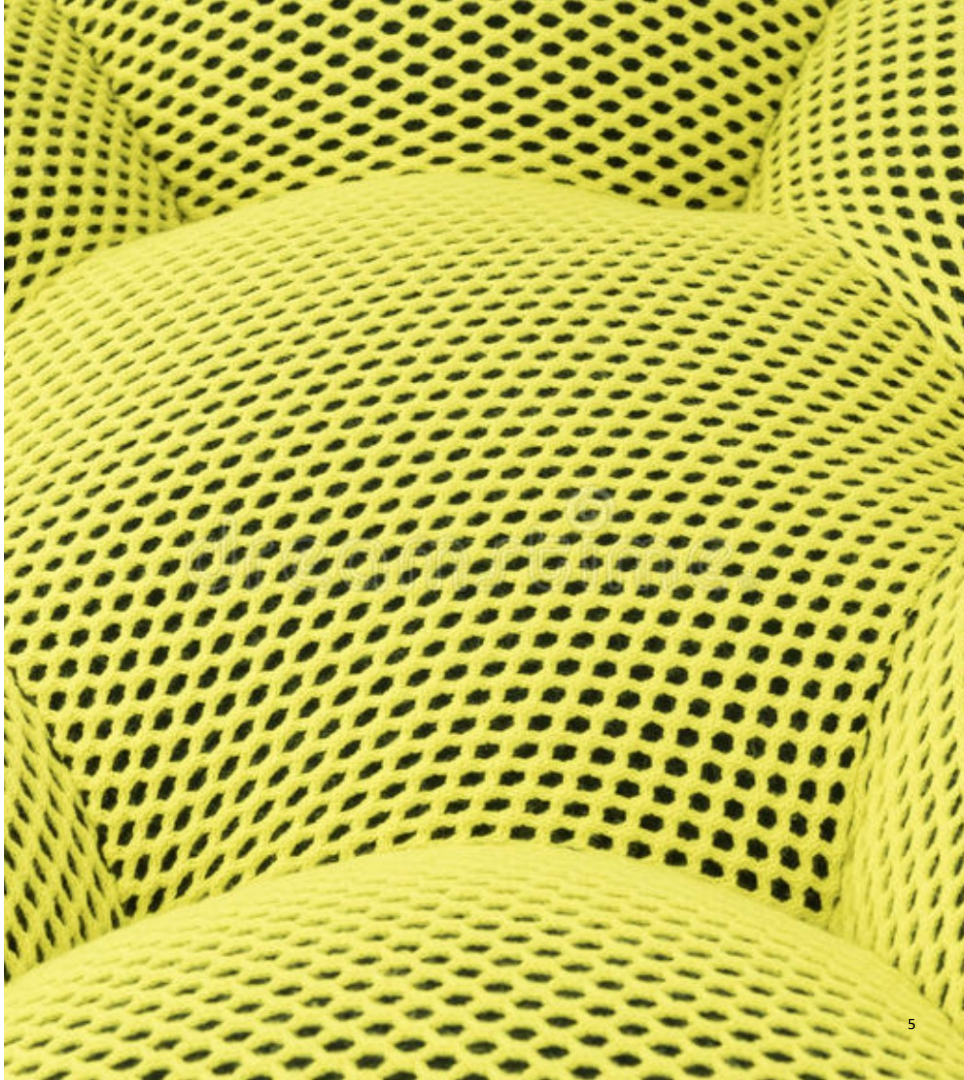
← All of these problems are common across services no matter the runtime.

← It would be cool if we could solve these problems in a standard way **without changing application code**

Service Mesh
and
Istio Architecture

What is a service mesh?

A service mesh provides a **transparent** and **language-independent** way to flexibly and easily manage the **communication** between microservices**.**

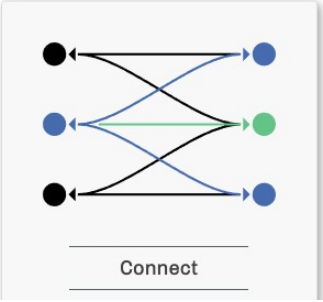


Istio is a **service mesh** that supports managing traffic flows between microservices, enforcing access policies, and aggregating telemetry data, all without requiring changes to the microservice code.

Current version: Istio 1.9

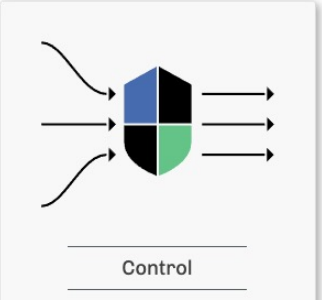
Istio

Connect, secure, control, and observe services.

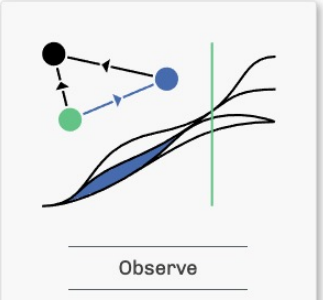


Traffic control, Discovery, Load Balancing, Resiliency

Encryption (TLS, mTLS), Authentication, Authorization of Service-to-Service communication

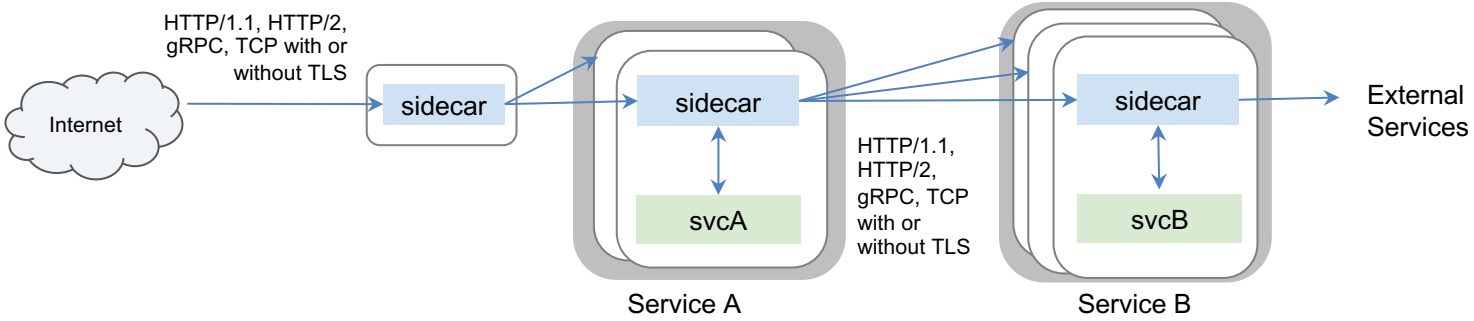


Policy Enforcement



Metrics, Logging, Tracing

Weaving the mesh



Inbound features:

- Service authentication
- Authorization
- Rate limits
- Load shedding
- Telemetry
- Request Tracing
- Fault Injection

Outbound features:

- Service authentication
- Load balancing
- Retry and circuit breaker
- Fine-grained routing
- Telemetry
- Request Tracing
- Fault Injection

Istio Architecture

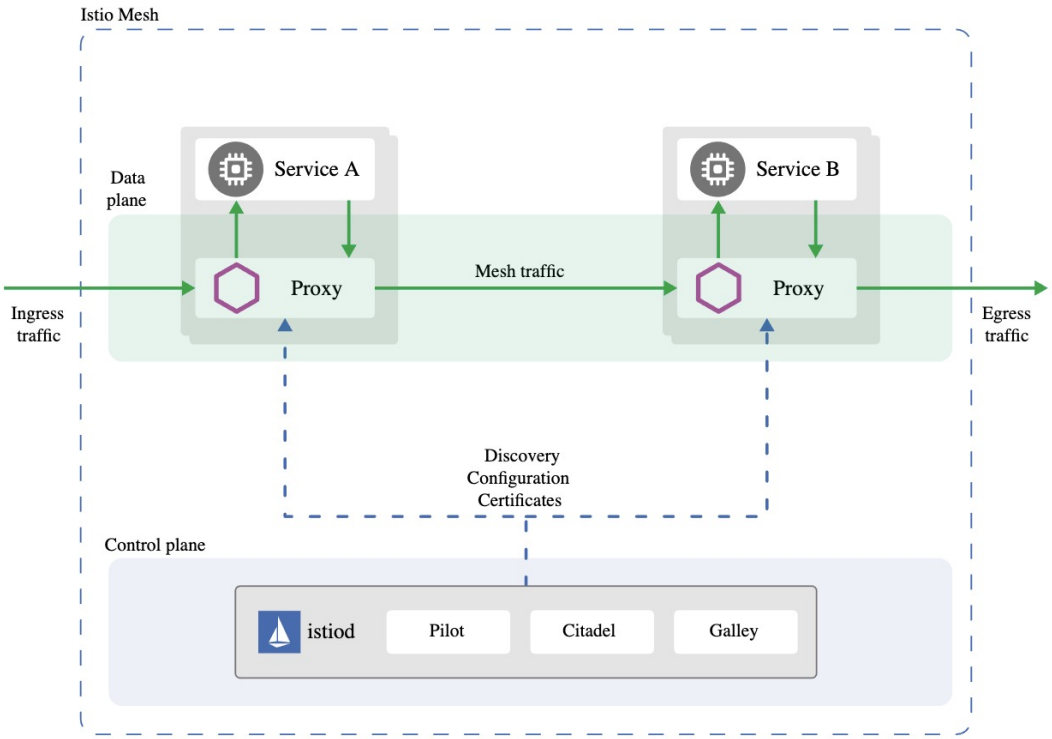
Since Istio 1.5, the control plane functionality is packaged into a single binary called **Istiod**.

Pilot
- Service discovery.
- Listens for your configuration around traffic routing, circuit-breakers and fault injection.
- Converts that to low-level routing rules, and distributes those to envoys at runtime.

Galley
- Configuration

Citadel
- Security: encryption, mTLS, authentication and authorization, access policies, certificate authority, network configuration, auditing tools.

WebAssembly (Mixer)
- Extensibility for Istio proxy (Envoy):
  - Efficiency, Function (to enforce policy, collect telemetry, payload mutations), Isolation, Configuration, Operator, Extension developer.
  - Metrics (Proxy, Service, Control Plane level metrics), Distributed Traces (with support for Zipkin, Jaeger, Lightstep and Datadog), Access Logs.



The Sidecar

Envoy

Envoy is an L7 proxy deployed as sidecar to services, adding:

- 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

https://www.envoyproxy.io

At its core, Envoy is an L3/L4 network proxy. A pluggable filter chain mechanism for gRPC, MongoDB, DynamoDB, Redis, Postgres. Envoy supports an additional HTTP L7 filter layer.

Installing the Sidecar

- Manual installation

```
istioctl kube-inject -f guestbook-deployment.yaml
kubectl get pod -l app=guestbook
NAME          READY        STATUS        RESTARTS    AGE
guestbook-64c6f57bc8-f5n4x      2/2      Running        0      24s
```

- Automatic installation. When you set istio-injection=enabled label on a namespace, a mutating admission controller webhook (Istio injection webhook) is enabled

```
kubectl label namespace default istio-injection=enabled --overwrite
```

Traffic Management

- In order to direct traffic within your mesh, Istio populates its **own service registry**, Istio connects to a service discovery system. In Kubernetes, Istio automatically detects the services and endpoints.
- Envoy proxies use a **round-robin load balancing** to distribute traffic.
- **VirtualServices and DestinationRules**, are the key building blocks of Istio's traffic routing, extending Envoy.
- A VirtualService uses **routing rules** to send traffic to appropriate destinations. A routing rule consists of a destination and 0 or more match conditions.
- **Gateways** control ingress and egress traffic. A Gateway is bound to the Istio VirtualService.
- A **service entry** can add an entry to the service registry to redirect and forward traffic for external destinations such as APIs consumed from the web or legacy infrastructure, add retry, timeout and fault injection policies, add services from a different cluster.
- You can add **failure recovery** (timeouts, retries, circuit breakers with a connection pool) **and fault injection** features to a Virtual Service.

https://istio.io/latest/docs/concepts/traffic-management/

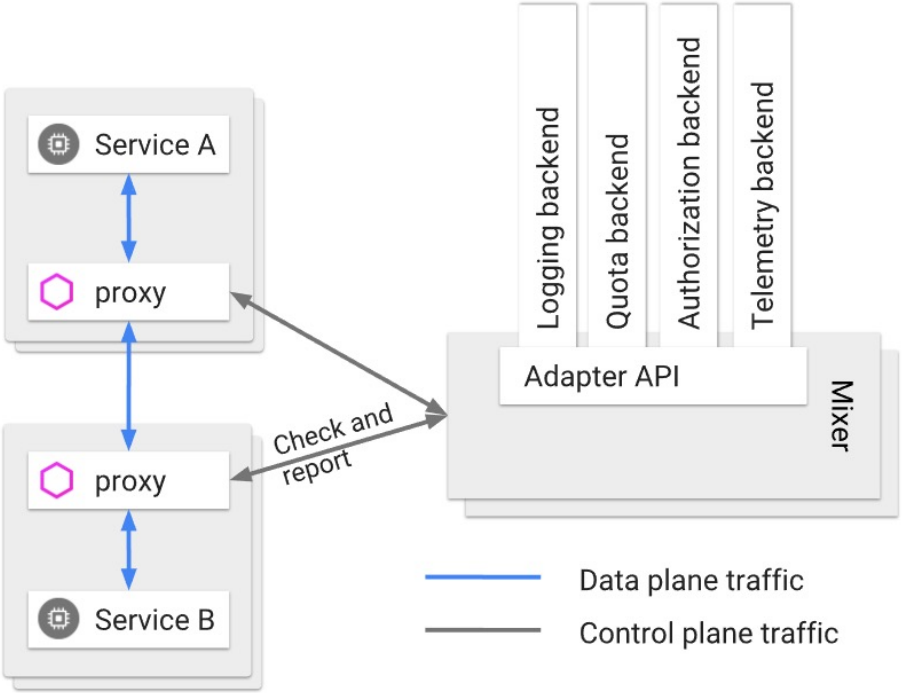
Observability

Prior to Istio 1.6, Mixer was the Istio component responsible for providing an adapter model to integrate with infrastructure backends, like policy controls and telemetry collection.

In Istio 1.6 a new method for integration with telemetry addons was introduced.

Integrations:

- Cert-manager
- Grafana for Istio dashboards,
- Jaeger,
- Kiali
- Prometheus,
- Zipkin



Telemetry Tools

Istio uses the Envoy proxy to generate the following types of telemetry:

- **Metrics**,
  - **Service level metrics**, based on the four golden signals of monitoring (latency, traffic, errors, and saturation).
  - **Proxy level metrics**, with full record of all inbound and outbound requests.
  - **Control plane metrics**, self-monitoring metrics
- **Distributed traces**, for call flows and service dependencies.
- Envoy **Access logs**, configurable set of formats, providing operators with full control of the how, what, when and where of logging.

You can configure metrics in the EnvoyFilter. Configuring custom statistics involves two sections of the EnvoyFilter: definitions and metrics.

Metrics

Service Mesh Visualization

Request Tracing

Dashboard for Metrics

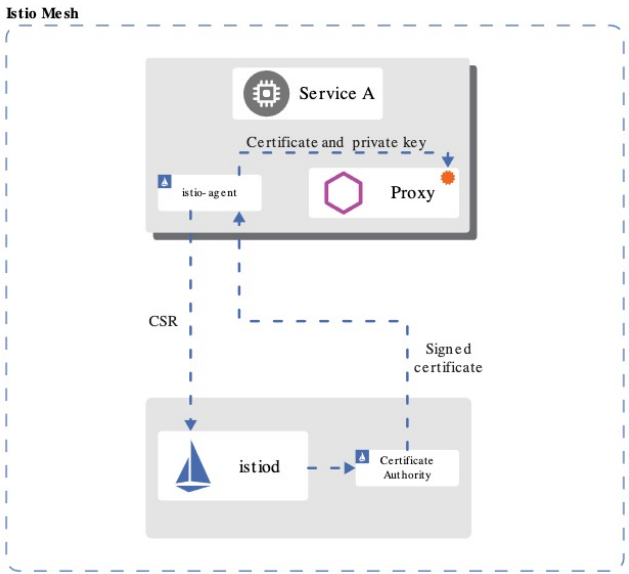
Security

Security components:

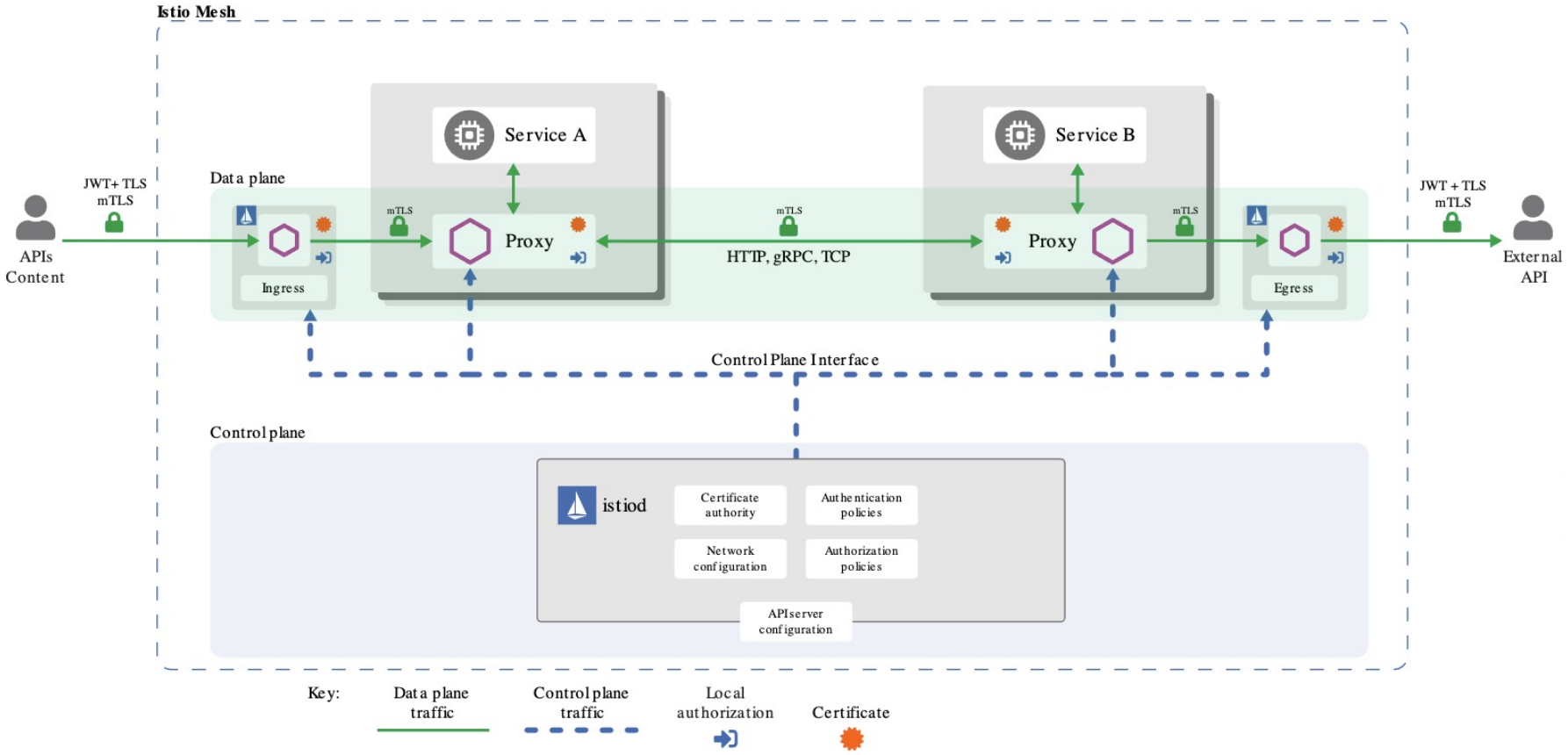
- Certificate Authority (CA) for key and certificate management,
- Configuration API Server distributes to proxies:
  - Authentication policies,
  - Authorization policies,
  - Secure naming information,
- Proxies work as Policy Enforcement Points (PEPs) to secure communication,
- Envoy proxy extensions to manage telemetry and auditing.

Istio security features:

- Istio Identity, using service identity like ServiceAccounts and Authorization Policies,
- Strong identity using X509 certificates, Istio agents run alongside Envoy proxy and work with istiod to automate key and certificate rotation,
- Authentication Policies,
  - Peer authentication: mTLS
  - Request authentication: ORA Hydra, Keycloak, Auth0, Firebase Auth, Google Auth
- Authorization Policies

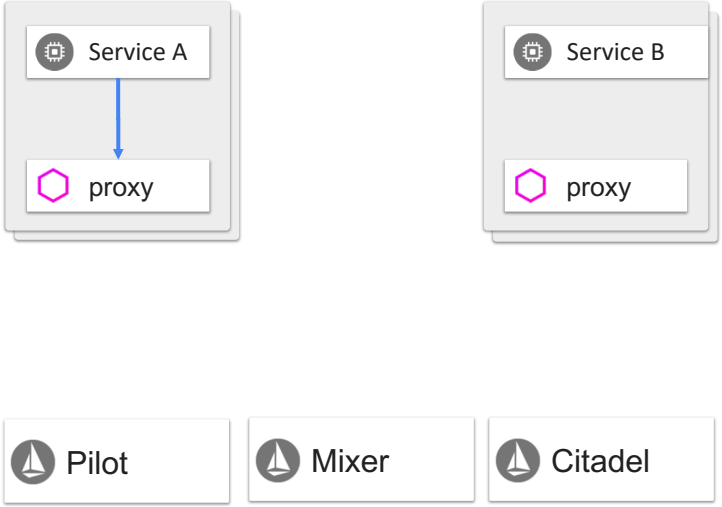


Security


Istio Mesh
Data plane
JWT+ TLS
mTLS
APIs
Content
Ingress
mTLS
Service A
Proxy
mTLS
HTTP, gRPC, TCP
Service B
Proxy
mTLS
Egress
JWT + TLS
mTLS
External
API
Control Plane Interface
Control plane
istiod
Certificate authority
Authentication policies
Network configuration
Authorization policies
API server configuration
Key:
Data plane traffic
Control plane traffic
Local authorization
Certificate

Lifecycle of a Request

Lifecycle of a Request

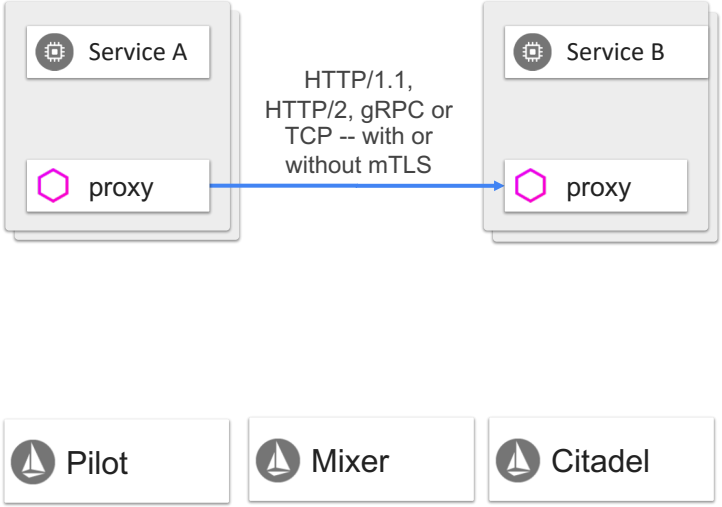


Service A places a call to http://service-b.

Client-side Envoy intercepts the call.

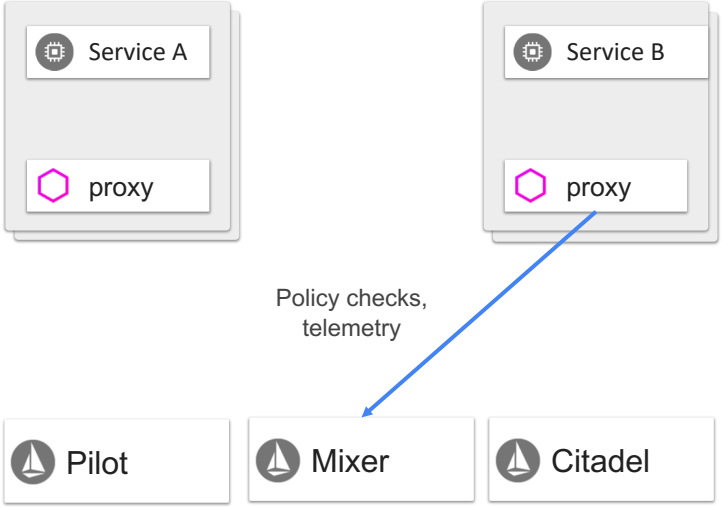
Envoy consults config (previously received from Pilot) to know how/where to route call to service B (taking into account service discovery, load balancing, and routing rules), forwards the call to the right destination.

Life of a request



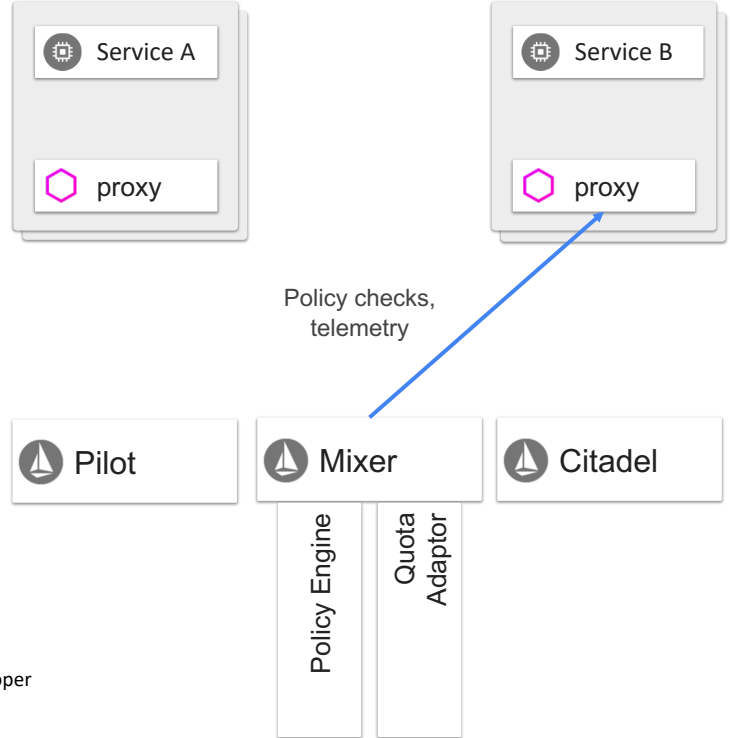
Envoy forwards request to appropriate instance of service B. There, the Envoy proxy deployed with the service intercepts the call.

Life of a request



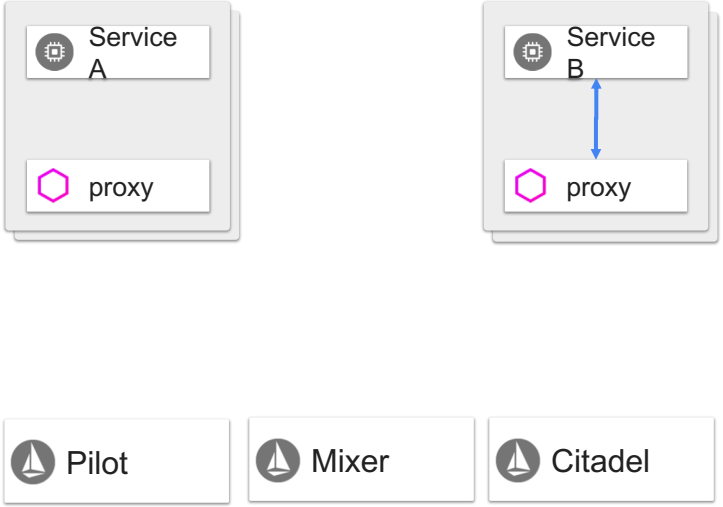
Server-side Envoy checks with Mixer to validate that call should be allowed (ACL check, quota check, etc).

Life of a request



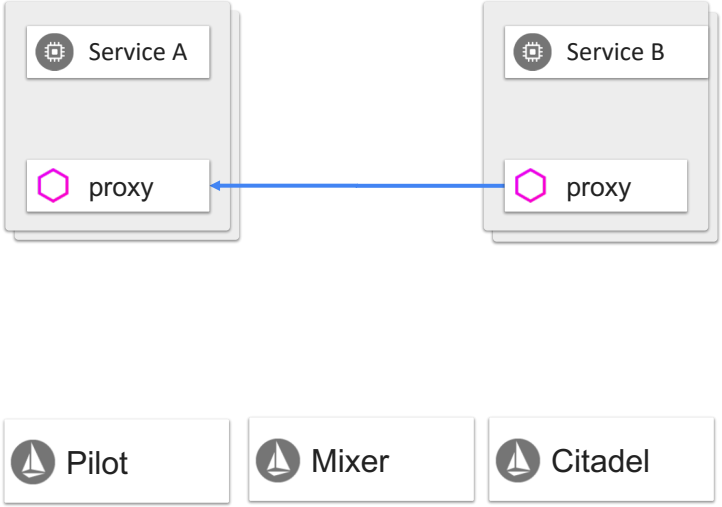
Mixer checks with appropriate adaptors (policy engine, quota adaptor) to verify that the call can proceed and returns true/false to Envoy

Life of a request



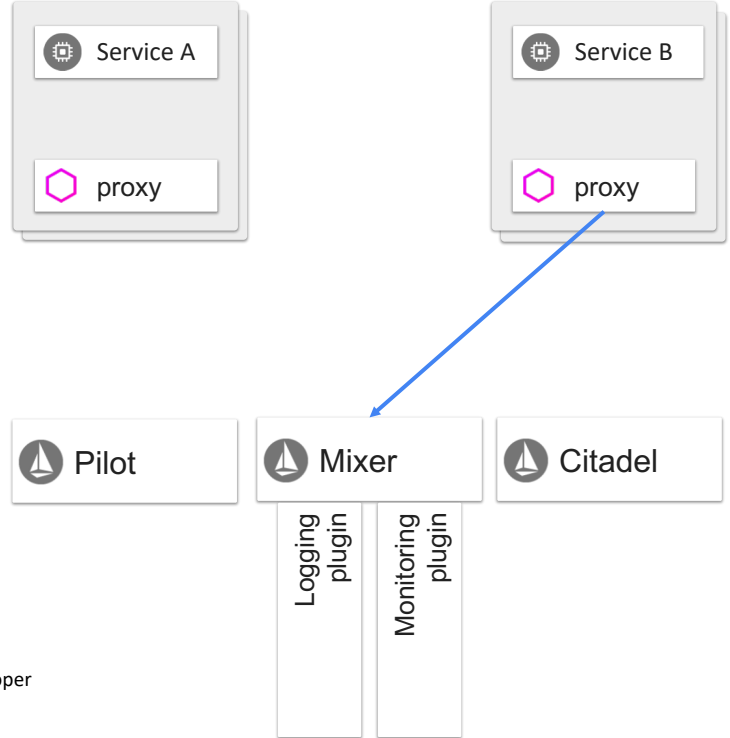
Server-side Envoy forwards request to service B, which process request and returns response

Life of a request



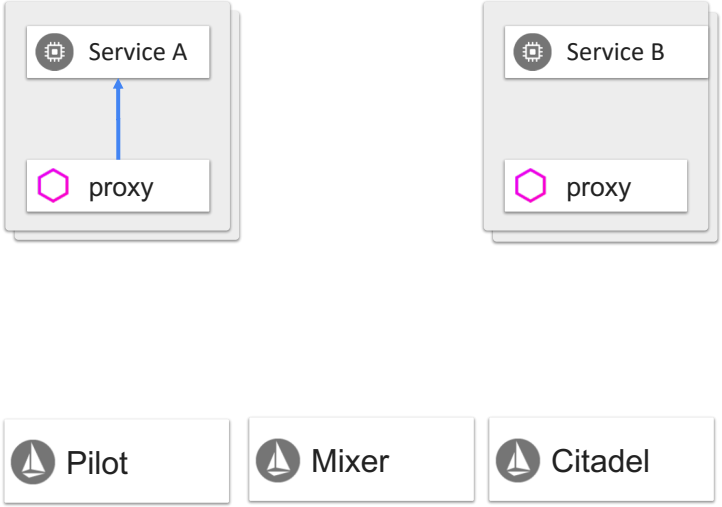
Envoy forwards response to the original caller, where response is intercepted by Envoy on the caller side.

Life of a request



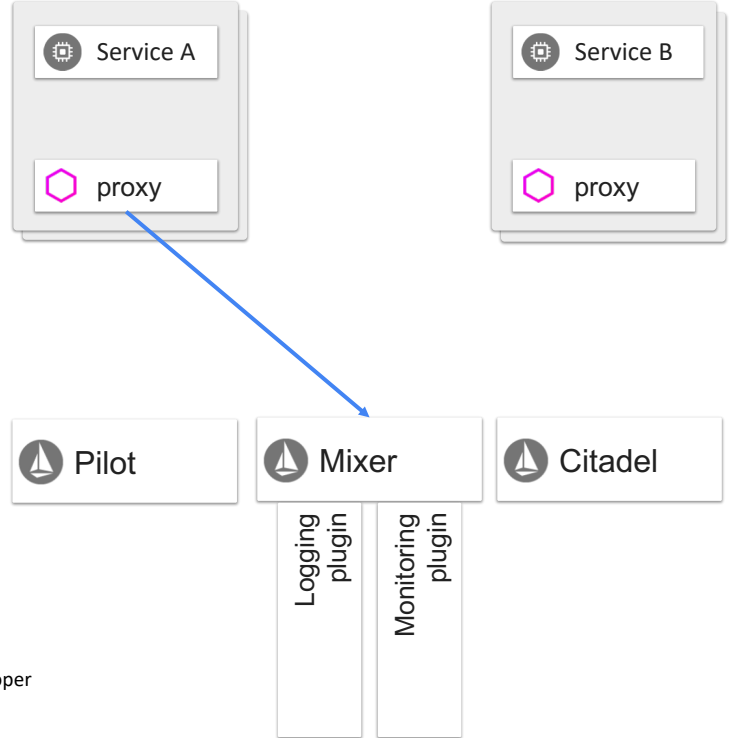
Envoy reports telemetry to Mixer, which in turn notifies appropriate plugins

Life of a request



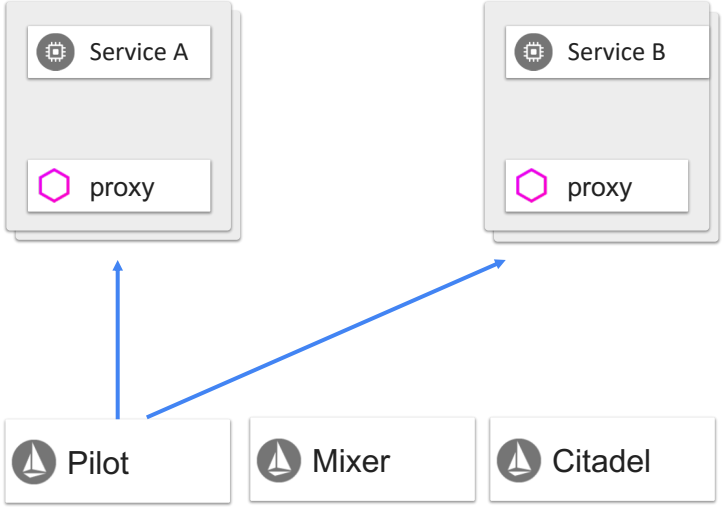
Client-side Envoy forwards response to original caller.

Life of a request



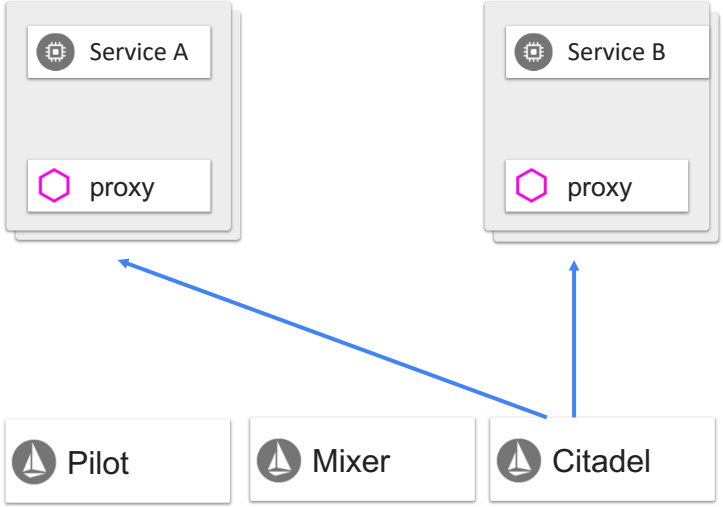
Client-side Envoy reports telemetry to Mixer (including client-perceived latency), which in turn notifies appropriate plugins

At Anytime



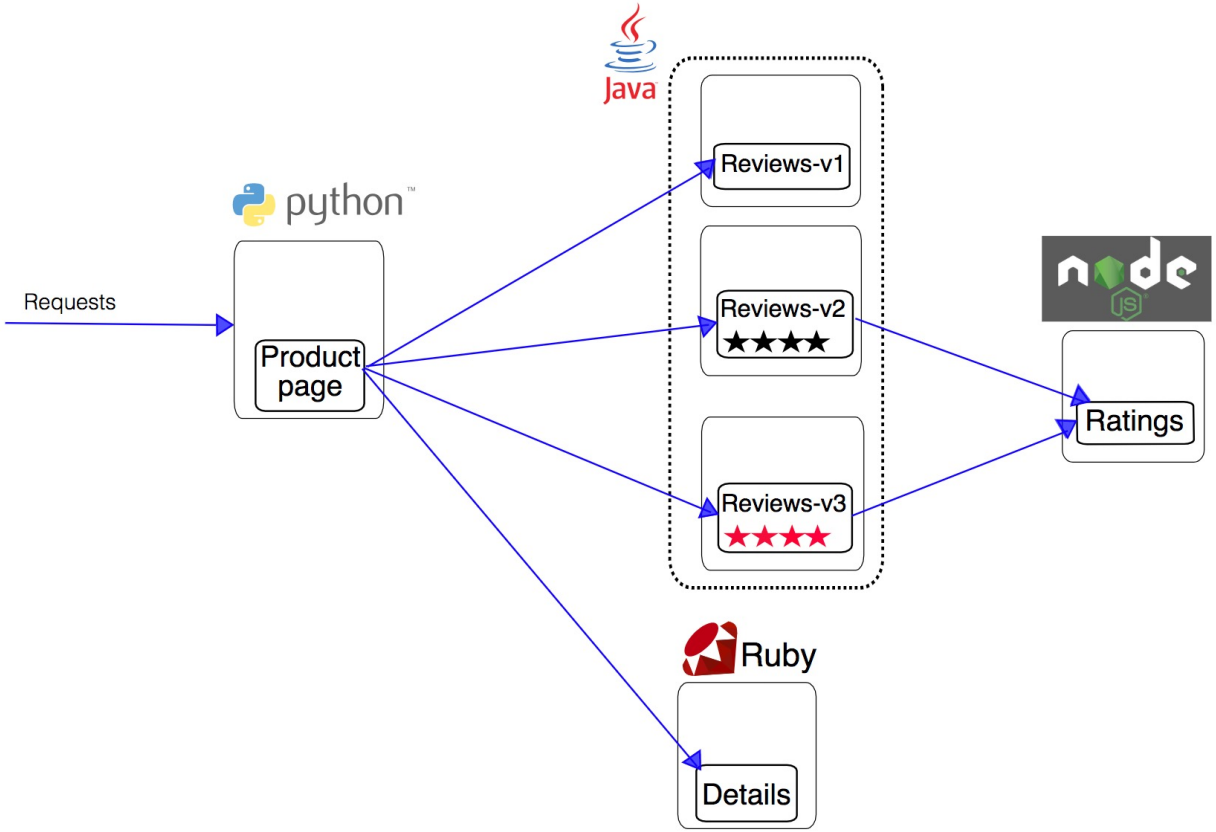
Pilot listens to your configuration, such as routing rules, circuit breaking and fault injection. Converts to low-level config (routing rules) and distributes to proxy side cars

At Anytime



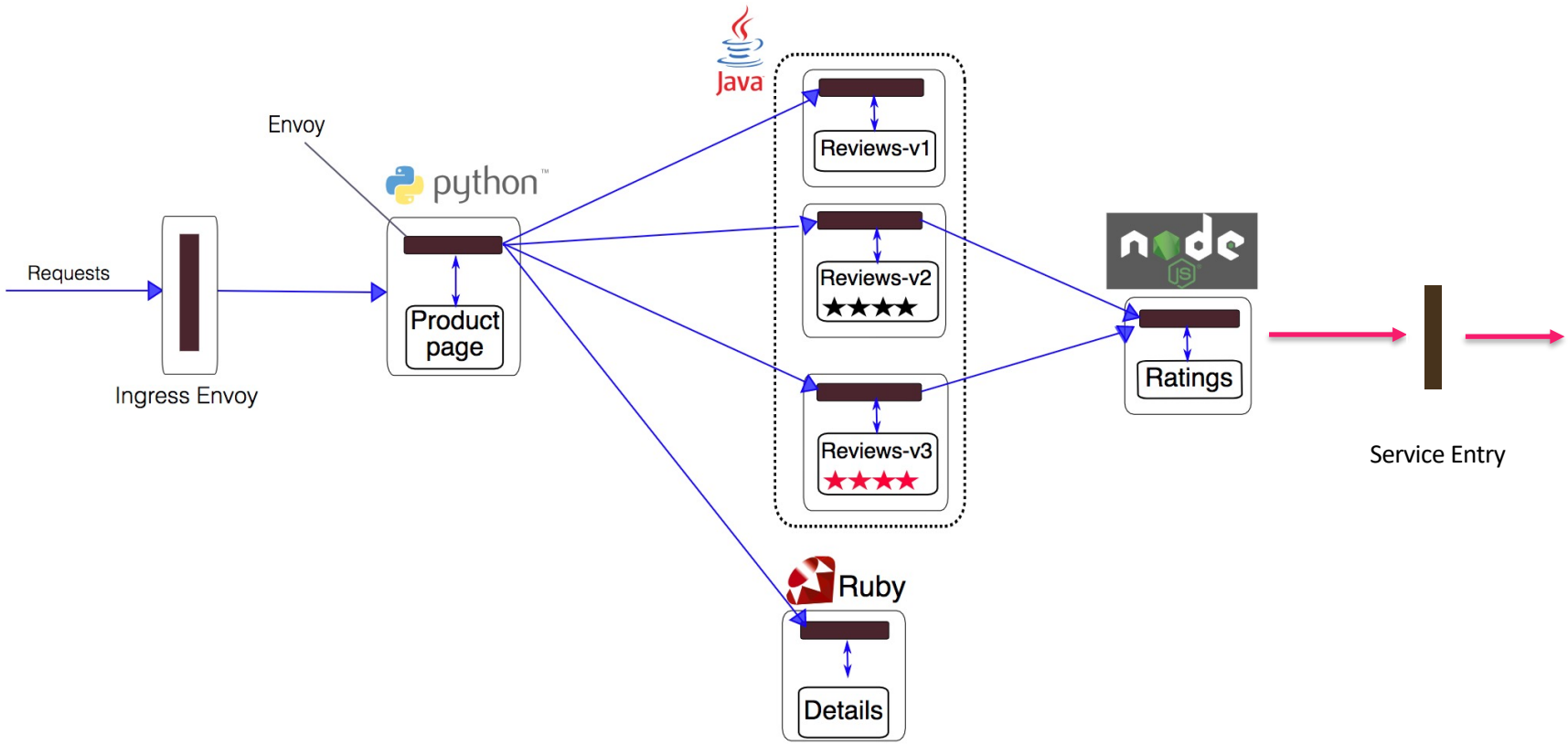
Citadel distributes keys and certificates as Kubernetes Secrets available to sidecar containers

Bookinfo Sample Microservices Application (without Istio)



Bookinfo Application without Istio

Bookinfo Sample Microservices Application (with Istio)



Bookinfo Application

Istio Examples

Request Routing

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v1
```

Canary Testing

Route user:jason to reviews:v2

Others still get reviews:v1

Traffic Shifting

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
      weight: 50
    - destination:
        host: reviews
        subset: v3
      weight: 50
```

50% -> v1

50% -> v3

Rate Limits

```yaml
apiVersion: "config.istio.io/v1alpha2"
kind: memquota
metadata:
  name: handler
  namespace: istio-system
spec:
  quotas:
  - name: requestcount.quota.istio-system
    maxAmount: 5000
    validDuration: 1s
    # The first matching override is applied.
    # A requestcount instance is checked against override dimensions.
    overrides:
    # The following override applies to 'ratings' when
    # the source is 'reviews'.
    - dimensions:
        destination: ratings
        source: reviews
      maxAmount: 1
      validDuration: 1s
    # The following override applies to 'ratings' regardless
    # of the source.
    - dimensions:
        destination: ratings
      maxAmount: 100
```

5000 requests per 1s

ratings: 100 requests per 1s

Circuit Breaking

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: httpbin
spec:
  host: httpbin
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 1
      http:
        http1MaxPendingRequests: 1
        maxRequestsPerConnection: 1
    outlierDetection:
      consecutiveErrors: 1
      interval: 1s
      baseEjectionTime: 3m
      maxEjectionPercent: 100
```

Max 1 concurrent connection & request

Delay Injection

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    fault:
      delay:
        percent: 100
        fixedDelay: 7s
    route:
    - destination:
        host: ratings
        subset: v1
  - route:
    - destination:
        host: ratings
        subset: v1
```

Inject 7 second delay

Fault Injection

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    fault:
      abort:
        percent: 100
        httpStatus: 500
    route:
    - destination:
        host: ratings
        subset: v1
  - route:
    - destination:
        host: ratings
        subset: v1
```

jason: Return with Error 500

IBM