Istio

Traffic Management

traffic routing rules let you easily control the flow of traffic and API calls between services

service-level properties like circuit breakers, timeouts, and retries, and makes it easy to set up important tasks like A/B testing, canary rollouts, and staged rollouts

relies on the Envoy proxies that are deployed along with your services. All traffic that your mesh services send and receive (data plane traffic) is proxied through Envoy

Istio on a Kubernetes cluster, then Istio automatically detects the services and endpoints in that cluster.

Envoy proxies distribute traffic across each service’s load balancing pool using a round-robin model

* [Virtual services](https://istio.io/latest/docs/concepts/traffic-management/#virtual-services)

configure how requests are routed to a service within an Istio service mesh, building on the basic connectivity and discovery provided by Istio and your platform

strongly decoupling where clients send their requests from the destination workloads

send traffic to different versions of a service, specified as service subsets. Clients send requests to the virtual service host as if it was a single entity, and Envoy then routes the traffic to the different versions depending on the virtual service rules: for example, “20% of calls go to the new version” or “calls from these users go to version 2

Routing rule precedence

* [Destination rules](https://istio.io/latest/docs/concepts/traffic-management/#destination-rules)

virtual services as how you route your traffic **to** a given destination, and then you use destination rules to configure what happens to traffic **for** that destination. Destination rules are applied after virtual service routing rules are evaluated, so they apply to the traffic’s “real” destination.

Routing rules are evaluated in sequential order from top to bottom, with the first rule in the virtual service definition being given highest priority

routing rules are a powerful tool for routing particular subsets of traffic to particular destinations. You can set match conditions on traffic ports, header fields, URIs, and more. For example, this virtual service lets users send traffic to two separate services, ratings and reviews, as if they were part of a bigger virtual service at http://bookinfo.com/. The virtual service rules match traffic based on request URIs and direct requests to the appropriate service.

* [Gateways](https://istio.io/latest/docs/concepts/traffic-management/#gateways)

Gateway to manage inbound and outbound traffic for your mesh, letting you specify which traffic you want to enter or leave the mesh. Gateway configurations are applied to standalone Envoy proxies that are running at the edge of the mesh, rather than sidecar Envoy proxies running alongside your service workloads.

* [Service entries](https://istio.io/latest/docs/concepts/traffic-management/#service-entries)

add an entry to the service registry that Istio maintains internally. After you add the service entry, the Envoy proxies can send traffic to the service

Configuring service entries allows you to manage traffic for services running outside of the mesh, including the following tasks:

* Redirect and forward traffic for external destinations, such as APIs consumed from the web, or traffic to services in legacy infrastructure.
* Define [retry](https://istio.io/latest/docs/concepts/traffic-management/#retries), [timeout](https://istio.io/latest/docs/concepts/traffic-management/#timeouts), and [fault injection](https://istio.io/latest/docs/concepts/traffic-management/#fault-injection) policies for external destinations.
* Run a mesh service in a Virtual Machine (VM) by [adding VMs to your mesh](https://istio.io/latest/docs/examples/virtual-machines/).
* [Sidecars](https://istio.io/latest/docs/concepts/traffic-management/#sidecars)

 configures every Envoy proxy to accept traffic on all the ports of its associated workload, and to reach every workload in the mesh when forwarding traffic. You can use a [sidecar](https://istio.io/latest/docs/reference/config/networking/sidecar/#Sidecar) configuration to do the following:

* Fine-tune the set of ports and protocols that an Envoy proxy accepts.
* Limit the set of services that the Envoy proxy can reach.

## Network resilience and testing

### Timeouts

A timeout is the amount of time that an Envoy proxy should wait for replies from a given service, ensuring that services don’t hang around waiting for replies indefinitely and that calls succeed or fail within a predictable timeframe. The Envoy timeout for HTTP requests is disabled in Istio by default.

### Retries

A retry setting specifies the maximum number of times an Envoy proxy attempts to connect to a service if the initial call fails.

### Circuit breakers

circuit breaker, you set limits for calls to individual hosts within a service, such as the number of concurrent connections or how many times calls to this host have failed. Once that limit has been reached the circuit breaker “trips” and stops further connections to that host

### Fault injection

can inject two types of faults, both configured using a [virtual service](https://istio.io/latest/docs/concepts/traffic-management/#virtual-services):

* Delays: Delays are timing failures. They mimic increased network latency or an overloaded upstream service.
* Aborts: Aborts are crash failures. They mimic failures in upstream services. Aborts usually manifest in the form of HTTP error codes or TCP connection failures.

For example, this virtual service introduces a 5 second delay for

**Security**

traffic encryption 🡺 man-in-the-middle attacks

mutual TLS and fine-grained access policies🡺 flexible service access control

auditing tools🡺 who did what at what time

* Security by default: no changes needed to application code and infrastructure
* Defense in depth: integrate with existing security systems to provide multiple layers of defense
* Zero-trust network: build security solutions on distrusted networks

A Certificate Authority (CA) for key and certificate management

The configuration API server distributes to the proxies:

authentication policies

authorization policies

secure naming information

Sidecar and perimeter proxies work as Policy Enforcement Points (PEPs) to secure communication between clients and servers.

A set of Envoy proxy extensions to manage telemetry and auditing

## Authentication

**Peer authentication**

* Istio offers [mutual TLS](https://en.wikipedia.org/wiki/Mutual_authentication) as a full stack solution for transport authentication, which can be enabled without requiring service code changes. This solution:
  + Provides each service with a strong identity representing its role to enable interoperability across clusters and clouds.
  + Secures service-to-service communication.
  + **Provides a key management system to automate key and** certificate generation, distribution, and rotation.

**Request Authentication**

end-user authentication to verify the credential attached to the request.

### **Mutual TLS authentication**

Istio tunnels service-to-service communication through the client- and server-side PEPs, which are implemented as [Envoy proxies](https://envoyproxy.github.io/envoy/).

**observability**