

What is Service Mesh and Istio

A **service mesh** is a dedicated infrastructure layer for making service-to-service communication safe, fast, and reliable.

Istio is a service mesh which allows you to connect, manage and secure your microservices in an easy and none intrusive way.

Some of the features that offer Istio are:

- Intelligent routing and load balancing
- Resilience against network failures
- Policy enforcement between services
- Observability of your architecture. Tracing and Metrics
- Securing service to service communication

Istio Architecture

Istio is composed of two major components:

- **Data plane** which is composed of **Envoy** proxies deployed as sidecar container along with your service for managing network along with policy and telemetry features.
- **Control plane** which is in charge of managing and configuring all **Envoy** proxies.

All communication within your **service mesh** happens through **Envoy** proxy, so any network logic to apply is moved from your service into your infrastructure.

Key Concepts of Istio

DestinationRule

A **DestinationRule** configures the set of rules to be applied when forwarding traffic to a service. Some of the purposes of a **DestinationRule** are describing circuit breakers, load balancer, and TLS settings or define **subsets** (named versions) of the destination host so they can be reused in other Istio elements.

For example to define two services based on the version label of a service with hostname **recommendation** you could do:

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

VirtualService

A **VirtualService** describes the mapping between one or more user-addressable destinations to the actual destination inside the mesh.

For example, to define two virtual services where the traffic is split between 50% to each one.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
        subset: version-v1
      weight: 90
    - destination:
        host: recommendation
        subset: version-v2
      weight: 10
```

ServiceEntry

A **ServiceEntry** is used to configure traffic to external services of the mesh such as APIs or legacy systems. You can use it in conjunction with a **VirtualService** and/or **DestinationRule**.

For example to configure *httpbin* external service:

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: httpbin-egress-rule
  namespace: istioegress
spec:
  hosts:
  - httpbin.org
  ports:
  - name: http-80
    number: 80
    protocol: http
```

Gateway

A **Gateway** is used to describe a load balancer operating at the edge of the mesh for incoming/outgoing HTTP/TCP connections. You can bind a **Gateway** to a **VirtualService**.

To configures a load balancer to allow external https traffic for host foo.com into the mesh:

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: foo-gateway
spec:
  servers:
  - port:
      number: 443
      name: https
      protocol: HTTPS
    hosts:
    - foo.com
    tls:
      mode: SIMPLE
      serverCertificate: /tmp/tls.crt
      privateKey: /tmp/tls.key
```

Getting started with Istio

Istio can be installed with *automatic sidecar injection* or without it. We recommend as starting point **without** *automatic sidecar injection* so you understand each of the steps.

Installing Istio

First you need to download Istio and register in PATH:

```
curl -L https://github.com/istio/istio/releases/download/1.0.2/istio-1.0.2-linux-amd64.tar.gz

tar xvf istio-1.0.2-linux-amd64.tar.gz

cd istio-1.0.2
export ISTIO_HOME=`pwd`
export PATH=$ISTIO_HOME/bin:$PATH
```

You can install Istio into Kubernetes cluster by either using helm install or helm template.

```
$ helm template install/kubernetes/helm/istio --name istio --namespace=istio-system > istio.yaml

kubectl create namespace istio-system
kubectl create -f $HOME/istio.yaml
```

Wait until all pods are up and running.

Intelligent Routing

Routing some percentage of traffic between two versions of recommendation service:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
        subset: version-v1
      weight: 75
    - destination:
        host: recommendation
        subset: version-v2
      weight: 25
```

Routing to a specific version in case of prefixed URI and cookie with a value matching a regular expression:

```
spec:
  hosts:
  - ratings
  http:
  - match:
    - headers:
        cookie:
          regex: "^(.*?;)?(user=jason)(;.*)?"
      uri:
        prefix: "/ratings/v2/"
    route:
    - destination:
        host: ratings
        subset: version-v2
```

Possible **match** options:

Field	Type	Description
uri	StringMatch	URI value to match. exact, prefix, regex
scheme	StringMatch	URI Scheme to match. exact, prefix, regex
method	StringMatch	Http Method to match. exact, prefix, regex
authority	StringMatch	Http Authority value to match. exact, prefix, regex
headers	map<string, StringMatch>	Headers key/value. exact, prefix, regex
port	int	Set port being addressed. If only one port exposed, not required
sourceLabels	map<string, string>	Caller labels to match
gateways	string[]	Names of the gateways where rule is applied to.

Sending traffic depending on caller labels:

```
- match:
  - sourceLabels:
      app: preference
      version: v2
    route:
  - destination:
      host: recommendation
      subset: version-v2
- route:
  - destination:
      host: recommendation
      subset: version-v1
```

When caller contains labels `app=preference` and `version=v2` traffic is routed to **subset** `version-v2` if not routed to `version-v1`

Mirroring traffic between two versions:


```
spec:
  hosts:
  - recommendation
  http:
  - route:
      - destination:
          host: recommendation
          subset: version-v1
    mirror:
      host: recommendation
      subset: version-v2
```


For routing purposes `VirtualService` also supports **redirects**, **rewrites**, **corsPolicies** or **appending** custom headers.

Apart from HTTP rules, `VirtualService` also supports matchers at *tcp* level.

```
spec:
  hosts:
  - postgresql
  tcp:
  - match:
      - port: 5432
        sourceSubnet: "172.17.0.0/16"
    route:
  - destination:
      host: postgresql
      port:
        number: 5555
```

Authors :

- 

@alexsotob
Java Champion and SW Engineer at Red Hat
- 

@christianposta
Chief Architect at Red Hat