Istio Traffic Management

Contents

[ISTIO 2](#_Toc40783479)

[Istio’s traffic management API. 3](#_Toc40783480)

[Virtual Services: 4](#_Toc40783481)

[Destination rules 5](#_Toc40783482)

[Gateways 5](#_Toc40783483)

[Service entries 6](#_Toc40783484)

[Sidecars 6](#_Toc40783485)

[Install the Istio 6](#_Toc40783486)

[Clone the project: 8](#_Toc40783487)

[Change to the cloned directory: 8](#_Toc40783488)

[kubectl get pods 8](#_Toc40783489)

[To get the IP 8](#_Toc40783490)

[Add Destination rule: 8](#_Toc40783491)

# ISTIO

Istio’s traffic routing rules let you easily control the flow of traffic and API calls between services.

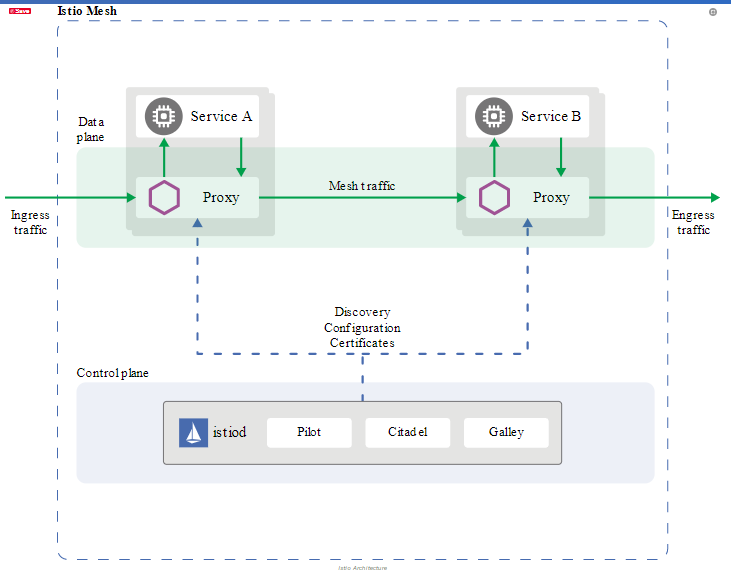
 Circuit breakers, timeouts, and retries

 A/B testing, canary rollouts, and staged rollouts with percentage-based traffic splits.

Working:

Relies on the **Envoy** proxies that are deployed along with the services.

All traffic that your mesh services send and receive (data plane traffic) is proxied through Envoy.



The **data plane** is composed of a set of intelligent proxies ([Envoy](https://www.envoyproxy.io/)) deployed as sidecars.

The **control plane** manages and configures the proxies to route traffic.

Pilot: Service discovery

Citadel: Authentication

Gallery: Configuration Management

By default, the Envoy proxies distribute traffic across each service’s load balancing pool using a round-robin model.

 Special rules to traffic

# Istio’s traffic management API.

* [Virtual services](https://istio.io/docs/concepts/traffic-management/#virtual-services)
* [Destination rules](https://istio.io/docs/concepts/traffic-management/#destination-rules)
* [Gateways](https://istio.io/docs/concepts/traffic-management/#gateways)
* [Service entries](https://istio.io/docs/concepts/traffic-management/#service-entries)
* [Sidecars](https://istio.io/docs/concepts/traffic-management/#sidecars)

# Virtual Services:

Configure how requests are routed to a service within an Istio service mesh.

Consists of a set of routing rules.

Management flexible and powerful –

Strong decoupling - clients send their requests from the destination workloads

Without virtual services, Envoy distributes traffic using round-robin load balancing.

Eg: configure traffic routes based on percentages across different service versions

Address multiple application services through a single virtual service

* Call to specific URI config to go different service
* Configure traffic rules in combination with [gateways](https://istio.io/docs/concepts/traffic-management/#gateways) to control ingress and egress traffic.

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: v3

# Destination rules

Destination rules to configure what happens to traffic **for** that destination.

Defines policies that apply to traffic intended for a service **after routing** has occurred.

* configuration for load balancing
* connection pool size
* outlier detection (for circuit breaking)
* subset of endpoints of a service

# Gateways

Manage inbound and outbound traffic for your mesh.

Specify which traffic you want to enter or leave the mesh.

Gateway configurations are applied to standalone Envoy proxies.

Configure layer 4-6 load balancing properties (ports to expose, TLS settings)

Can bind virtual service to gateway.

apiVersion: networking.istio.io/v1alpha3

kind: Gateway

metadata:

name: ext-host-gwy

spec:

selector:

app: my-gateway-controller

servers:

- port:

number: 443

name: https

protocol: HTTPS

hosts:

- ext-host.example.com

tls:

mode: SIMPLE

serverCertificate: /tmp/tls.crt

privateKey: /tmp/tls.key

apiVersion: networking.istio.io/v1alpha3

kind: VirtualService

metadata:

name: virtual-svc

spec:

hosts:

- ext-host.example.com

gateways:

- ext-host-gwy

# Service entries

To add an entry to the service registry that Istio maintains internally.

Manage traffic for services running outside of the mesh.

* Redirect and forward traffic for external destinations, such as APIs consumed.
* Define [retry](https://istio.io/docs/concepts/traffic-management/#retries), [timeout](https://istio.io/docs/concepts/traffic-management/#timeouts), and [fault injection](https://istio.io/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/docs/examples/virtual-machines/).
* Logically add services from a different cluster to the mesh to configure a [multicluster Istio mesh](https://istio.io/docs/setup/install/multicluster/gateways/" \l "configure-the-example-services) on Kubernetes.

# Sidecars

By default, Istio configures every Envoy proxy

Sidecars config:

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

Demo

# Install the Istio

export ISTIO\_VERSION=1.2.2

Download

curl -L https://git.io/getLatestIstio | sh -

Add istioctl to your PATH:

export PATH=$PWD/istio-$ISTIO\_VERSION/bin/:$PATH

Create a namespace for Istio's system components:

kubectl create namespace istio-system

Create the CRDs:

helm template istio-$ISTIO\_VERSION/install/kubernetes/helm/istio-init --name istio-init --namespace istio-system | kubectl apply -f -

Ensure that there are 23 CRDs:

kubectl get crds | grep 'istio.io\|certmanager.k8s.io' | wc -l

Deploy Istio:

will install the Istio's pilot and ingressgateway which are the minimum components we need to manage traffic and expose endpoints.

helm template istio-$ISTIO\_VERSION/install/kubernetes/helm/istio --name istio --namespace istio-system \

> --values istio-$ISTIO\_VERSION/install/kubernetes/helm/istio/values-istio-minimal.yaml \

> --set gateways.enabled=true | kubectl apply -f -

kubectl -n istio-system get po

NAME READY STATUS RESTARTS AGE

istio-ingressgateway-8b858ff84-wvwgm 0/1 Running 0 41s

istio-init-crd-10-p96gz 0/1 Completed 0 113s

istio-init-crd-11-shh9p 0/1 Completed 0 113s

istio-init-crd-12-8xf7x 0/1 Completed 0 113s

istio-pilot-56bc797997-hng6x 0/1 Pending 0 41s

## Clone the project:

git clone https://github.com/boxboat/istio-blog-resources.git

## Change to the cloned directory:

cd istio-blog-resources

istioctl kube-inject -f ./traffic-weighting/deployment.yaml | kubectl apply -f -

kubectl apply -f ./traffic-weighting/services.yaml

## kubectl get pods

kubectl apply -f ./traffic-weighting/gateway.yaml

kubectl apply -f ./traffic-weighting/virtualservice.yaml

## To get the IP

kubectl get svc istio-ingressgateway -n istio-system

Access application through IP/colors

## Add Destination rule:

kubectl apply -f ./traffic-weighting/destinationrule.yaml

Access application through IP/colors

Reference:

[1] <https://istio.io/docs/concepts/traffic-management/>

[2] <https://istio.io/docs/ops/deployment/architecture/>

[3] <https://www.katacoda.com/boxboat/courses/kubernetes-basic/istio-traffic-management/>

[4] <https://github.com/t2run/istio-blog-resources>