- Additional Resources

- [The Istio documentation]
https://istio.io/latest/docs/

- [The Service Mesh Handbook (PDF)]
https://www.tetrate.io/service-mesh-handbook/

- [Istio Cheat Sheet]
https://tetr8.io/istio-cheatsheet

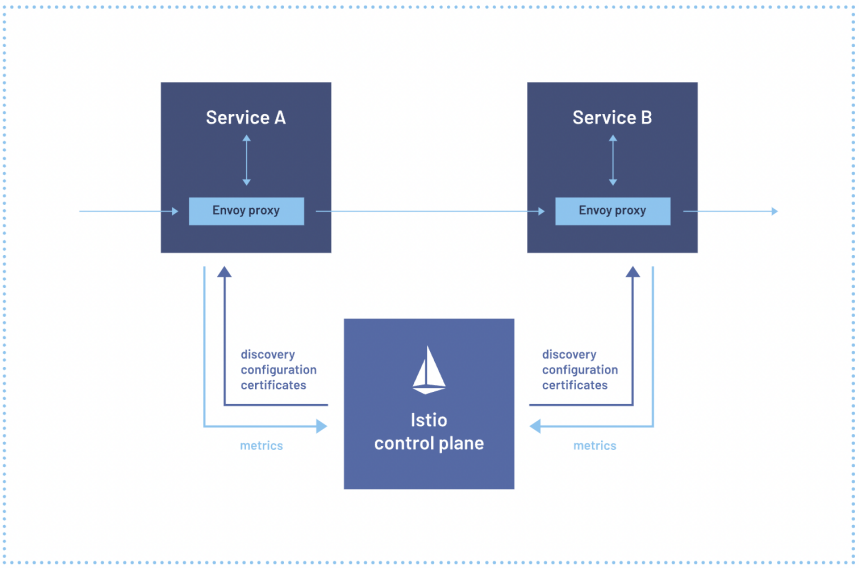
- [Collection of Istio articles and resources]
https://github.com/askmeegs/learn-istio

- [Istio weekly YouTube playlist]
https://www.youtube.com/playlist?list=PLm51GPKRAmTnMzTf9N95w_yXo7izg80Jc

- [Tetrate Tech Talks YouTube playlist]
https://www.youtube.com/playlist?list=PLm51GPKRAmTlOkjWDJBQYtjcc9WPk4E4F

- [Envoy fundamentals course]
https://academy.tetrate.io/courses/envoy-fundamentals

- Overview



Sidecar Injection

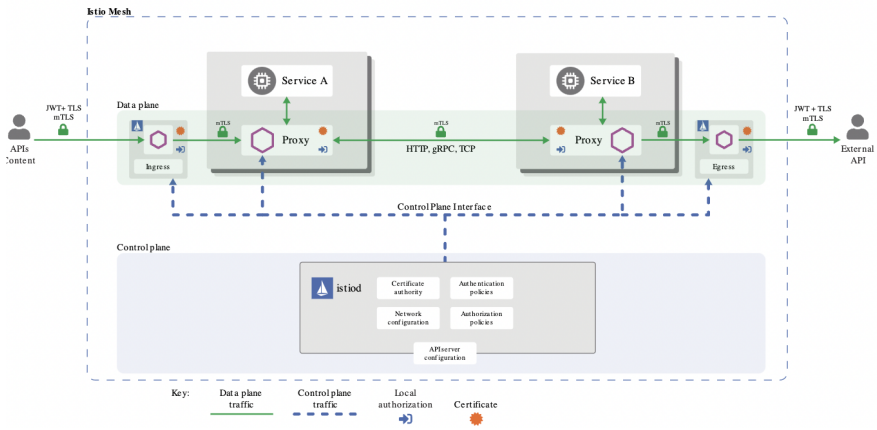
Manual sidecar injection

Istio has a command-line interface (CLI) named **`istioctl`** with the subcommand **`kube-inject`**. The subcommand processes the original deployment manifest to produce a modified manifest with the sidecar container specification added to the pod (or pod template) specification. The modified output can then be applied to a Kubernetes cluster with the **`kubectl apply -f`** command.
With manual injection, the process of altering the manifests is explicit.

Automatic sidecar injection

With automatic injection, the bundling of the sidecar is made transparent to the user.
The webhook is triggered according to a simple convention, where the application of the label **`istio-injection=enabled`** to a Kubernetes namespace governs whether the webhook should modify any deployment or pod resource applied to that namespace to include the sidecar.

In addition to the Envoy sidecar, the sidecar injection process injects a Kubernetes init container. This **`init-container`** is a process that applies these **`iptables`** rules before the Pod containers are started.



- DEMO K8s

https://istio.io/latest/docs/setup/getting-started/#download - **install Istio binary**

https://kubernetes.io/docs/tasks/tools/ - **Install kubectl**

Configuration profiles

Istio service mesh has numerous configuration settings that operators can update before installing Istio. To group the most common configuration settings into a higher-level abstraction, Istio uses the concept of configuration profiles.

The configuration profiles contain different configuration settings for the control plane as well as the data plane of Istio. The installation configuration profiles are expressed through the Istio Operator API and the IstioOperator resource.

Six configuration profiles are currently available, as shown in the list below. To get an up-to-date list of Istio configuration profiles, run the **`istioctl profile list`** command.

- **Default profile**
  The default profile is meant for production deployments and deployments of primary clusters in multi-cluster scenarios. It deploys the control plane and ingress gateway.
- **Demo profile**
  The demo profile is intended for demonstration deployments. It deploys the control plane and ingress and egress gateways and has a high level of tracing and access logging enabled.
- **Minimal profile**
  The minimal profile is equivalent to the default profile but without the ingress gateway. Itth deploys the control plane.
- **External profile**
  The external profile is used for configuring remote clusters in a multi-cluster scenario. It does not deploy any components.
- **Empty profile**
  The empty profile is used as a base for custom configuration. It does not deploy any components.
- **Preview profile**
  The preview profile contains experimental features. It deploys the control plane and ingress gateway.
-

To install Istio using the Istio CLI, we can use the **`--set`** flag and specify the profile like this:
istioctl install --set profile=demo

The Istio CLI offers convenience commands that allow us to get a full dump of Istio configuration profiles and the differences between the two profiles.

istioctl profile dump demo

***Additional info**
- The Istio Operator API and the IstioOperator resource allow us to install and configure Istio on a Kubernetes cluster.
- We can install install using helm

Installation

Install istio binary

```
curl -L https://istio.io/downloadIstio |ISTIO_VERSION=1.14.3 sh -
cd istio-1.14.3
export PATH=$PWD/bin:$PATH

#Deploy demo profile
istioctl install --set profile=demo

#Check status
kubectl get po -n istio-system
```

Enable sidecar injection

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


#Check status
kubectl get namespace -L istio-injection

#Test
kubectl create deploy my-nginx —image=nginx

kubectl get po
kubectl describe po `k8s  get pod |grep nginx| awk '{print $1}'`
kubectl delete deployment my-nginx


#Deploy the BookInfo sample application that is bundled with the Istio distribution
kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml

#Finally, deploy the bundled sleep sample service:
kubectl apply -f samples/sleep/sleep.yaml
kubectl get pod

PRODUCTPAGE_POD=$(kubectl get pod -l app=productpage
-ojsonpath='{.items[0].metadata.name}')

SLEEP_POD=$(kubectl get pod -l app=sleep -ojsonpath='{.items[0].metadata.name}')

kubectl exec $SLEEP_POD -it -- curl productpage:9080/productpage | head

kubectl get pod $PRODUCTPAGE_POD -ojsonpath='{.spec.containers[*].name}'

#Use the below istioctl dashboard command to open the Envoy dashboard web page:

istioctl dashboard envoy deploy/productpage-v1.default
```

Observability

Deploy metrics server

- **Using minikube**
  minikube addons enable metrics-server
  minikube dashboard

Prometheus for Istio

kubectl apply -f samples/addons/prometheus.yaml
Next
With Prometheus now running and collecting metrics, send another request to the **`productpage`** service:

SLEEP_POD=$(kubectl get pod -l app=sleep -ojsonpath='{.items[0].metadata.name}')

kubectl exec $SLEEP_POD -it -- curl productpage:9080/productpage | head

Run the following command, which Istio provides as a convenience to expose the Prometheus server's dashboard locally:

istioctl dashboard prometheus

Enter the metric name **`istio_requests_total`**

istio_requests_total{response_code="200"}

istio_requests_total{source_app="sleep",destination_app="productpage"}
rate(istio_requests_total{destination_app="productpage"}[5m])

while true; do kubectl exec $SLEEP_POD -it -- curl productpage:9080/productpage; sleep 1; done

Grafana Dashboards for Istio

kubectl apply -f samples/addons/grafana.yaml

Launch the Grafana UI with the following command:

istioctl dashboard grafana

Inside that folder, you will find six dashboards:

- **Mesh**: provides a high-level overview of the health of services in the mesh.
- **Service**: for monitoring a specific service. The metrics shown here are aggregated from multiple workloads.
- **Workload**: this allows you to inspect the behavior of a single workload.
- **Control Plane**: designed to monitor the health of `istiod`, the Control plane itself. It helps determine whether the control plane is healthy and able to synchronize the Envoy sidecars to the state of the mesh.
- **Performance**: this allows you to monitor the resource consumption of `istiod` and the sidecars.
- **Wasm Extension**: For monitoring the health of custom Web Assembly extensions deployed to the mesh.

Capture the name of the `sleep` pod:

SLEEP_POD=$(kubectl get pod -l app=sleep -ojsonpath='{.items[0].metadata.name}')

Run the following simple script to make repeated calls to the `productpage` endpoint:

while true; do kubectl exec $SLEEP_POD -it -- curl productpage:9080/productpage; sleep 0.3; done

Begin by navigating to the **Istio Mesh Dashboard**. This dashboard is a perfect place to get our bearings and see what services are running, inspect their health, and view global stats such as the global request volume.

Visit the **Istio Service Dashboard**, select the service named `productpage.default.svc.cluster.local`, and expand the *General* panel. There you will find the typical *golden signals*, including request volume, success rate (or errors), and request duration. The other two panels, *Client Workloads* and *Service Workloads,* break down incoming requests to this service by source and by destination, respectively.

Distributed Tracing

Distributed tracing is an important component of observability that complements metrics dashboards.
The idea is to provide the capability to "see" the end-to-end request-response flow through a series of microservices and to draw important information from it.
From a view of a distributed trace, developers can discover potential latency issues in their applications.
Terms
The end-to-end request-response flow is known as a *trace*. Each component of a trace, such as a single call from one service to another, is called a *span*. Traces have unique IDs, and so do spans. All spans that are part of the same trace bear the same trace ID.
The IDs are propagated across the calls between services in HTTP headers whose names begin with `x-b3` and are known as *B3 trace headers*

Deploy Jaeger

kubectl apply -f samples/addons/jaeger.yaml

The resulting deployment can be seen in the **`istio-system`** namespace.
As in previous labs, store the name of the **`sleep`** pod in an environment variable:

SLEEP_POD=$(kubectl get pod -l app=sleep -ojsonpath='{.items[0].metadata.name}')

while true; do kubectl exec $SLEEP_POD -it -- curl productpage:9080/productpage; sleep 1; done

istioctl dashboard jaeger

From the search form on the left-hand side of the UI, select the service **`productpage.default`**, and click on the button **Find Traces** at the bottom of the form.

Kiali Console

Kiali is an open-source graphical console specifically designed for Istio and includes numerous features.
Through alerts and warnings, it can help validate that the service mesh configuration is correct and that it does not have any problems.
With Kiali, one can view Istio custom resources, services, workloads, or applications.

Begin by deploying Kiali to your Kubernetes cluster:

kubectl apply -f samples/addons/kiali.yaml

As in previous labs, store the name of the **`sleep`** pod in an environment variable:
SLEEP_POD=$(kubectl get pod -l app=sleep -ojsonpath='{.items[0].metadata.name}')

Next, run the following command to send requests to the **`productpage`** service at a 1-2 second interval:

while true; do kubectl exec $SLEEP_POD -it -- curl productpage:9080/productpage; sleep 1; done

Finally, launch the Kiali dashboard:
istioctl dashboard kiali

- **DEMO OCP**