Istio

# Overview

**Service Mesh** - A service mesh is a configurable infrastructure layer for a microservices application.

It makes communication between service instances flexible, reliable, and fast.

The mesh provides features such as - service discovery, load balancing, encryption, authentication and authorization, support for the circuit breaker pattern etc.

**How does Service Mesh Work?**

A service mesh is usually implemented by providing a proxy instance, called a sidecar, for each service instance.

Sidecars handle inter‑service communications, monitoring, security‑related concerns – anything that can be abstracted away from the individual services.

This way, developers can handle development, support, and maintenance for the application code in the services; operations can maintain the service mesh and run the app.

**Istio** –

Istio is a Service Mesh that layers transparently onto existing distributed applications.

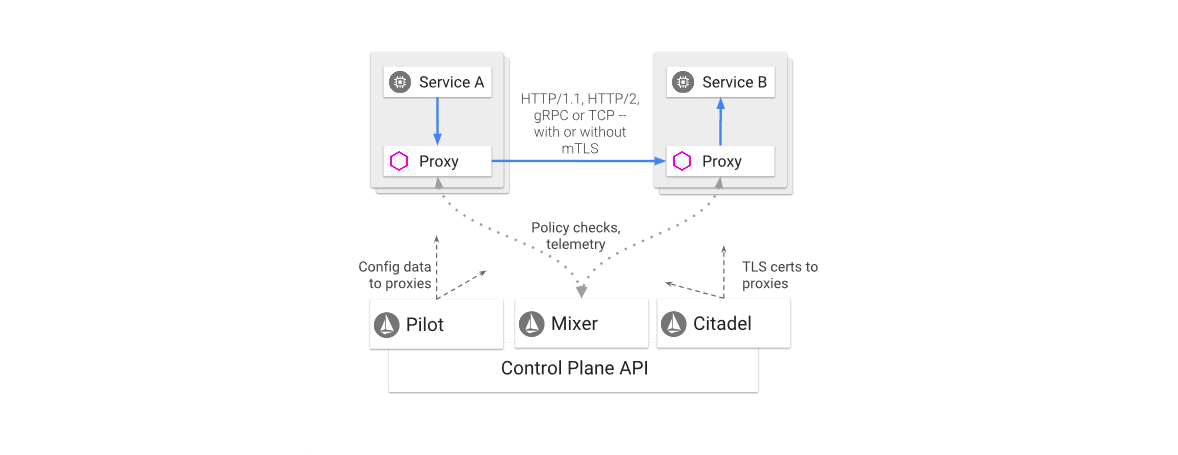
Istio makes it easy to create a network of deployed services with load balancing, service-to-service authentication, monitoring, and more, without any changes in service code.

You add Istio support to services by deploying a special sidecar proxy throughout your environment that intercepts all network communication between microservices.

Istio allows the user to then configure and manage Istio using its control plane functionality.

**What are the Istio Components?**

Istio has 5 major components that allow the user to configure and utilize its features. There are additional service that can be enabled and utilized.

The major components of Istio are -

**Envoy** **-** Envoy is a high-performance **proxy** to mediate all inbound and outbound traffic for all services in the service mesh.

**Mixer -** Mixer enforces access control and usage policies across the service mesh, and collects telemetry data.

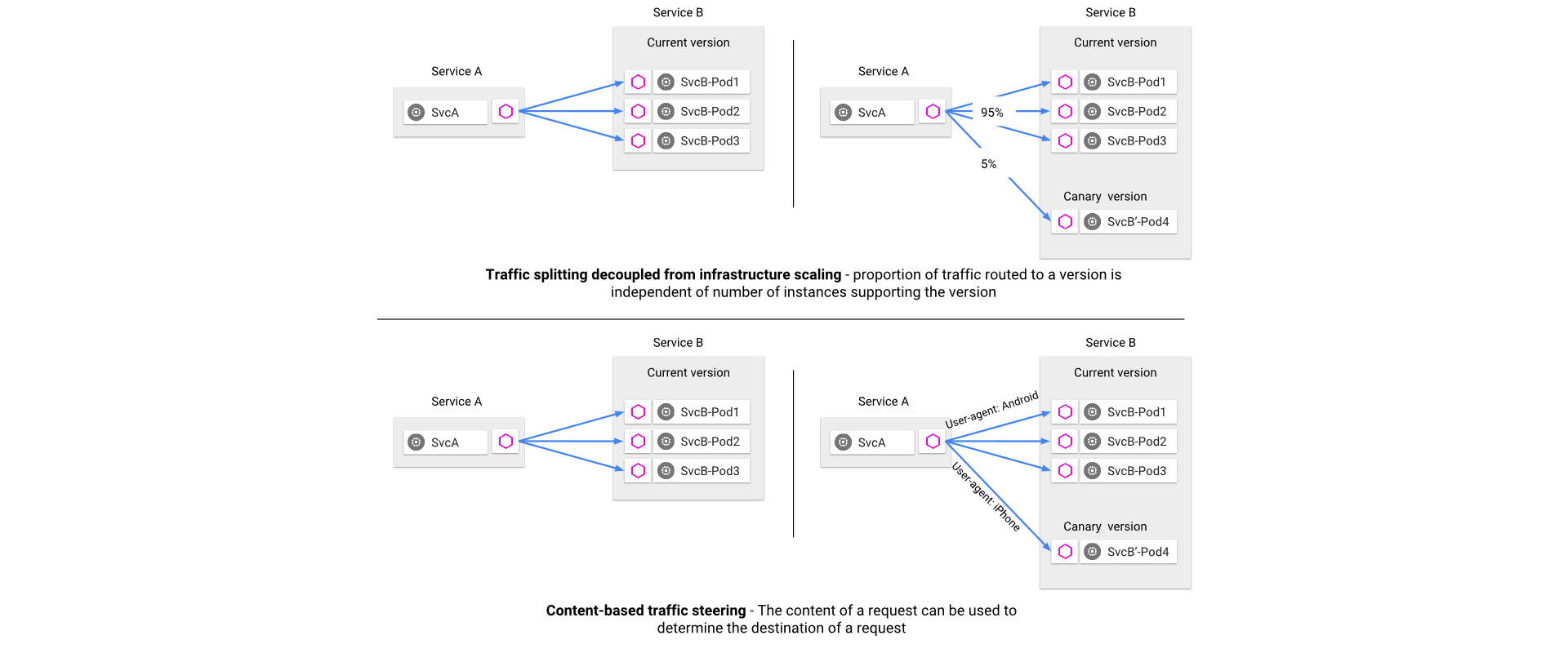
**Pilot -** Pilot provides service discovery for the Envoy sidecars, traffic management capabilities for intelligent routing and resiliency.

**Citadel -** Citadel provides strong service-to-service and end-user authentication with built-in identity and credential management.

Istio’s features include –

**Traffic Management**

Istio’s traffic management model essentially decouples traffic flow and infrastructure scaling.

It lets you specify via Pilot what rules they want traffic to follow rather than which specific pods/VMs should receive traffic.

Pilot & Envoy Proxies use these rules to intelligently route traffic to a specific version or based on user.

**Example -** You can route 95% of your traffic to your Current version while you test your New Version with the other 5% of traffic and then increment the weightage to your convenience.

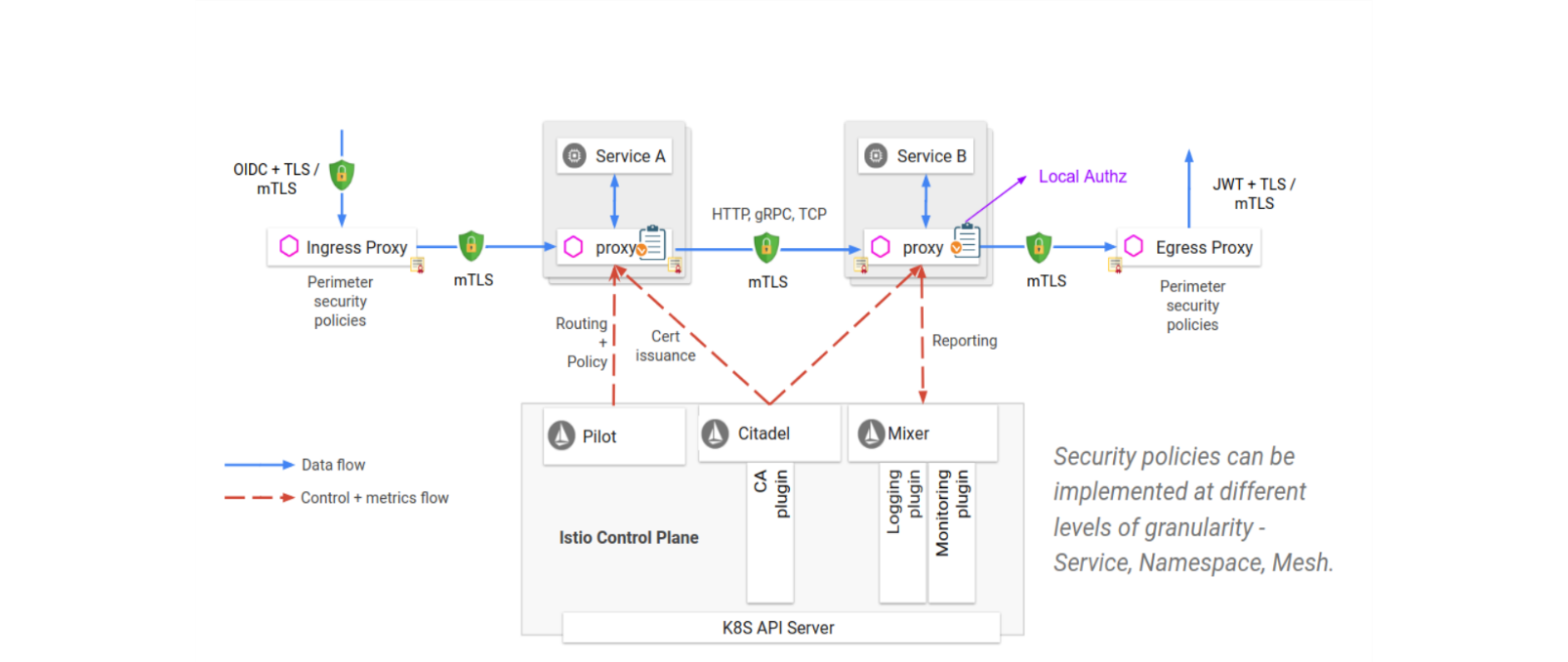
**Security**

Istio ‘s Security features include Authentication, Authorization, Access Control and securing Service-to-Service Communication by applying Policies on Service, Namespace or Mesh Level.

**The components of Istio that handle Security are – Citadel, Pilot, Envoy Sidecars & Mixer**

**Citadel** is primarily used for Key & Certificate Management by communicating with the **Envoy** **Sidecars** that Implement that Secure Communication between Clients & Servers.

**Pilot** is used to distribute the Authentication Policies using its Routing & also distributes the Secure Naming Information to the **Envoy** **Proxies**.

**Mixer** is used to manage the Authorization & Auditing of the Control & Metric Flow.

**Policies & Telemetry**

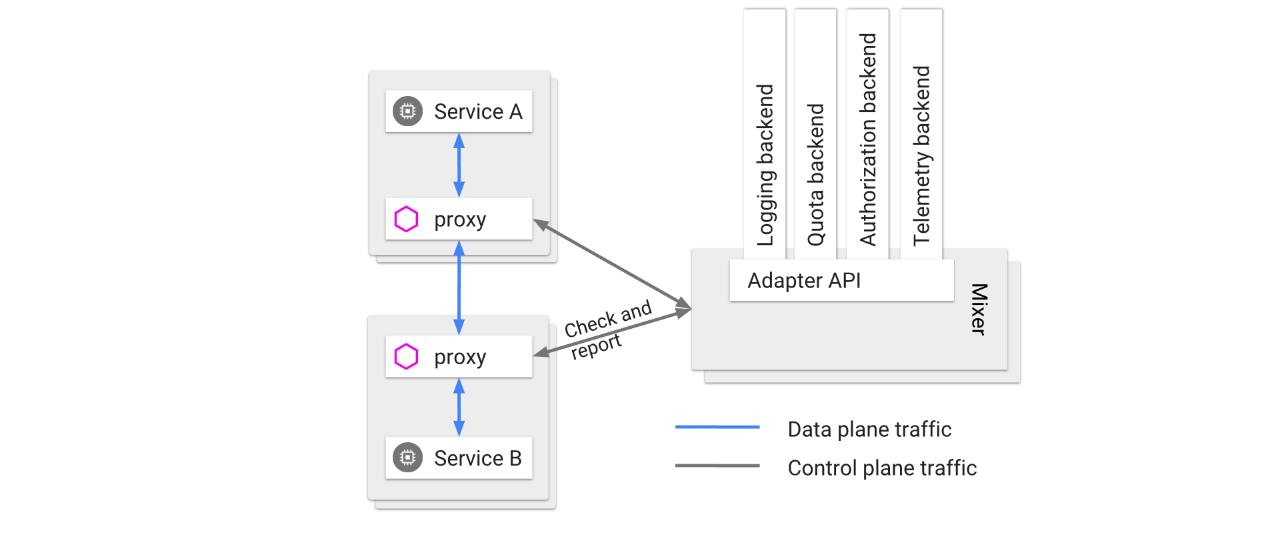
Istio’s Envoy Sidecars logically call Mixer, **BEFORE** each request to perform a Precondition check & **AFTER** each request to report Telemetry.

**NOTE:** Envoy Sidecar has a Local Cache which has a large percentage of Precondition Checks from this Cache. Envoy also buffers outgoing telemetry such that it only calls Mixer infrequently.

Mixer is a highly modular and extensible component. It comes with a general purpose plug-in model known as Adapters that allow Mixer to interface with many backend services that serve a purpose.

Example –

The Apigee Mixer adapter provides Apigee’s distributed authentication and quota policy checks as well as the ingestion of Istio telemetry for analysis and reporting.

****

**Telemetry –**

Istio has multiple Monitoring & Tracing Tools to Enable.

It allows you to watch multiple metrics with Dashboard like **Grafana, Prometheus, Jaeger, & Kiali**.

**Grafana** gives you Visualization of Metrics I the form of Mesh Dashboard, Performance Dashboard, Workloads Dashboard, Service Dashboard as well as Mixer & Pilot Dashboards.

**Prometheus** allows you to query for Metrics that are specific your need.

**Jaeger** provides distributed tracing that can be used to monitor and troubleshoot microservices-based systems.

**Kiali** is an Open-source web-based graphical user interface to view service graphs of the mesh and your Istio configuration objects. It generates graph data that allows you to visualize your application traffic with different graph types including – Versioned, Workload and Service Type. It also gives you the Health Check Information to see Degraded Service Traffic.

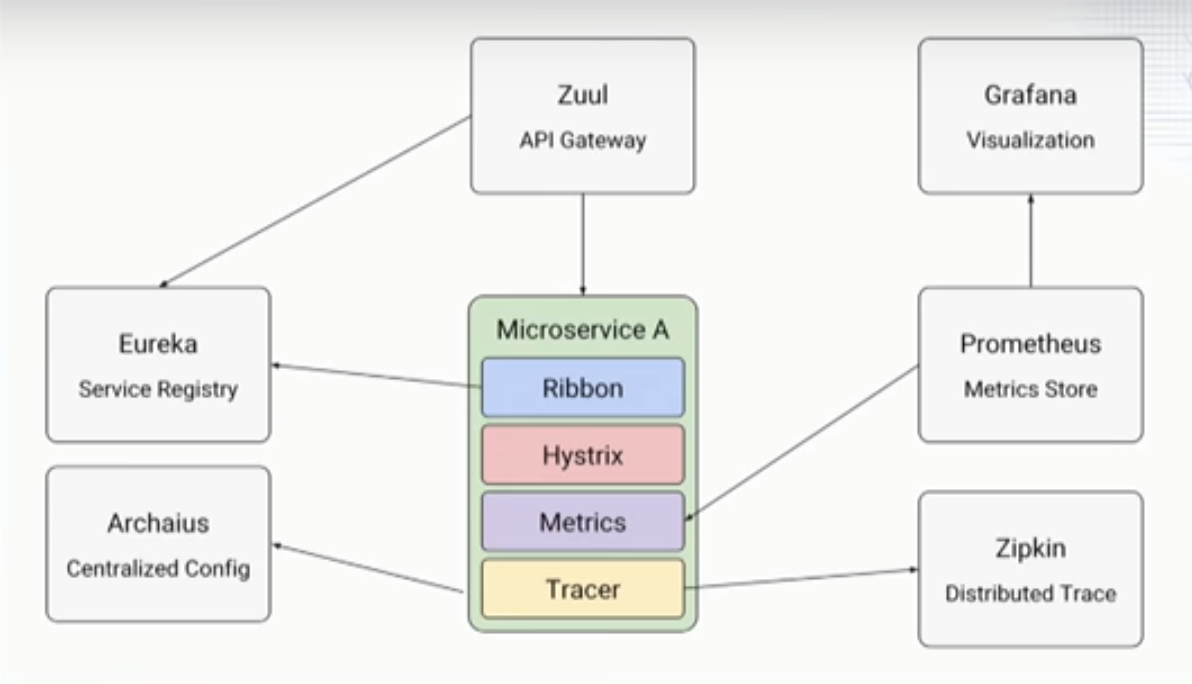
**Performance & Scalability**

Istio also comes with an open-source End-to-End Load Testing Tool known as **Fortio**.

This allows the user to test based on Queries per second (qps) and records the histograms of execution times and calculates percentiles. It can run for a set duration, fixed calls, or until interrupted.

**How does Istio Compare to the Netflix Stack?**

Consider a springboot microservice that communicates using the Netflix Stack. It looks as follows -

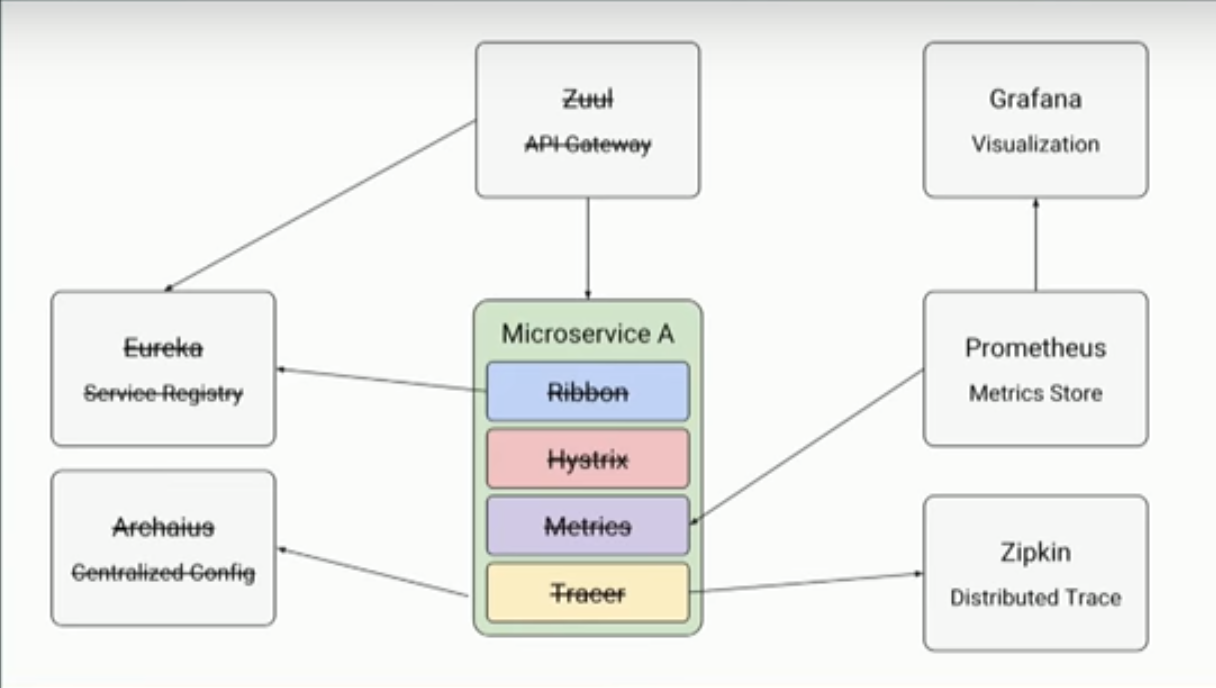


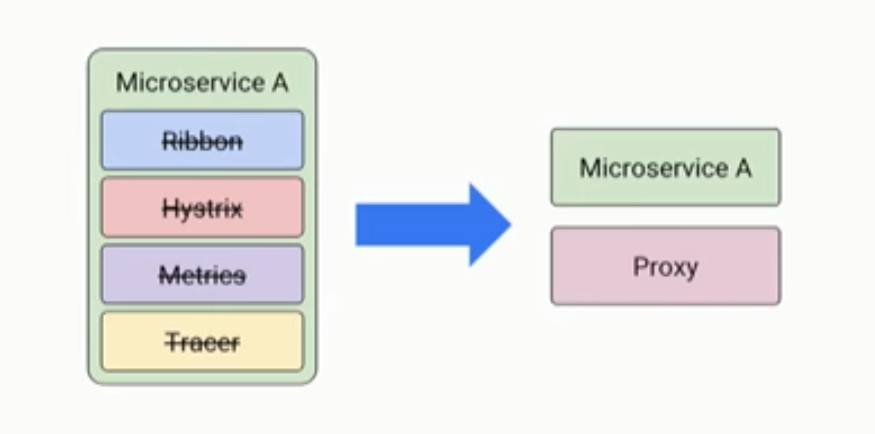
There is Eureka for Service Registry, Zuul for Checking for the Service from the Registry and serving traffic as a Gateway to the Microservice. Metrics using Prometheus and Grafana (Optional).

There is Zipkin for Tracing and Archaius for Central Configuration Management.

Although, this is a working model, Service Registry using Eureka can in scenarios act as a single point of failure. Maintenance of Eureka Service all the time is a tedious task.

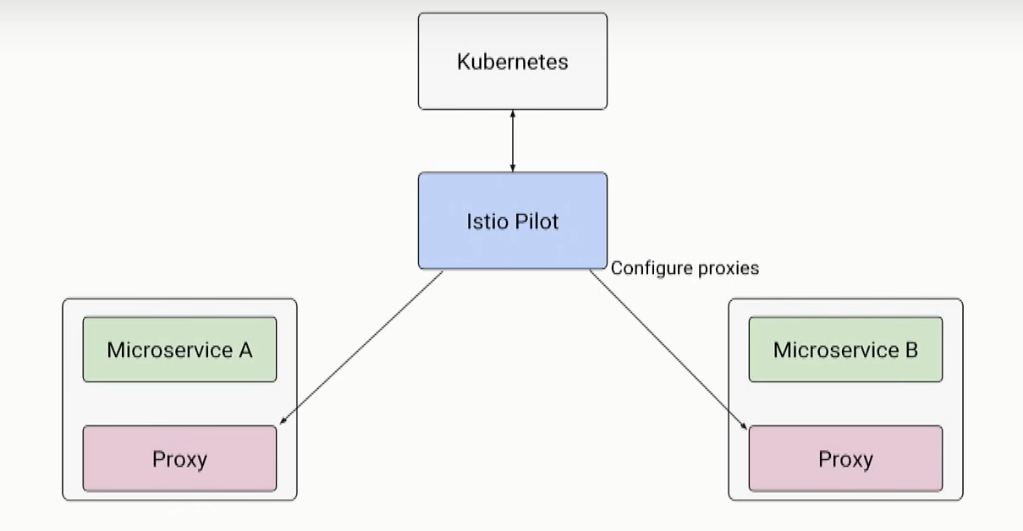
This is where Istio can be an effective replacement.



Using Sidecar Proxy Instances for Each Microservice, the Service Discovery of All Running Microservices can be done.

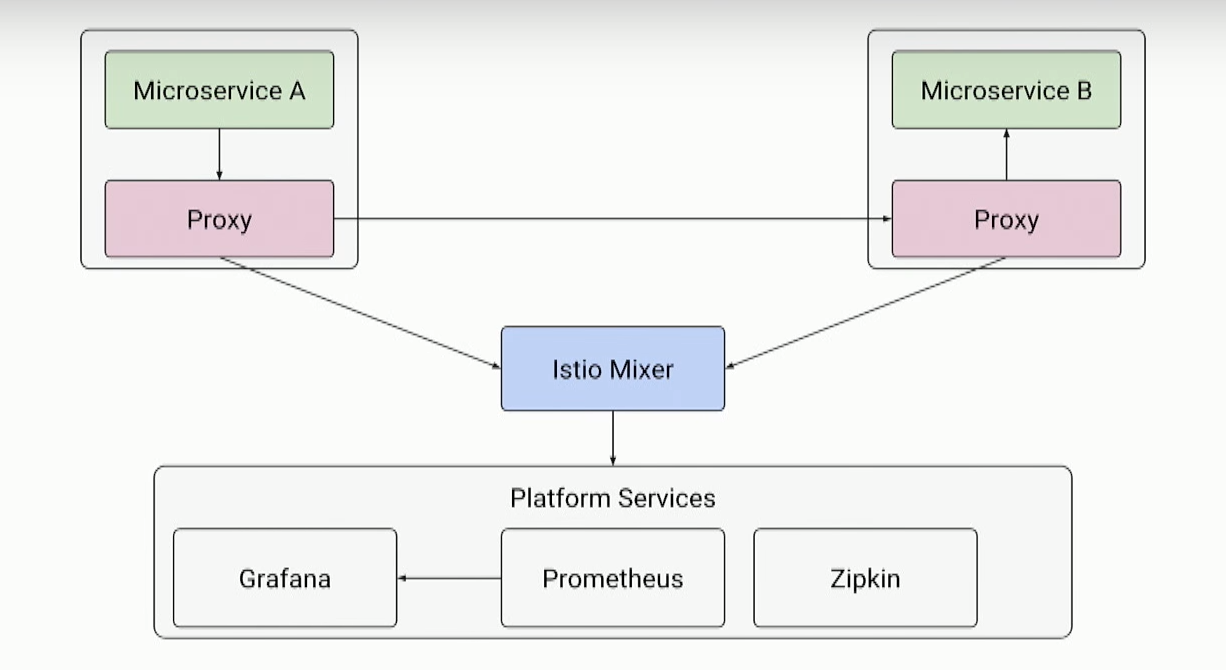
When you Deploy Microservices using Istio, your microservices come up with Sidecar Proxy Instances (Envoy) attached to them. These Proxy Instances Register these Microservices to Pilot.

This allows us to apply traffic management rules on Pilot to Route Traffic to the desired Microservices.



Mixer on the Other End is an Platform-Independent Component. This enforces access cntrol an usage policies aross the service mesh.

Mixer is also used to collect telemetry data from these Envoy Sidecars.

Mixer includes a flexible plugin model. This allows you to interface with variety of host environments and infrastructure backends.

Install & Configure Istio using Helm

# Setup Proxy to Download Latest Istio Version

export HTTP\_PROXY="http://proxy.ebiz.verizon.com:80/"

export HTTPS\_PROXY="http://proxy.ebiz.verizon.com:80/"

#Curl to Download the Latest Istio

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

#Set the istioctl PATH

export PATH="$PATH:/home/k8suser/istio-1.0.5/bin"

#Unset Proxy before Continuing

unset HTTP\_PROXY

unset HTTPS\_PROXY

# Make Changes to istio-1.0.5/install/kubernetes/helm/istio/values.yaml

# Change Ingressgateway from LoadBalancer -> NodePort

# Enable All Add-ons like Telemetry, ServiceGraph, Kiali, Jaeger

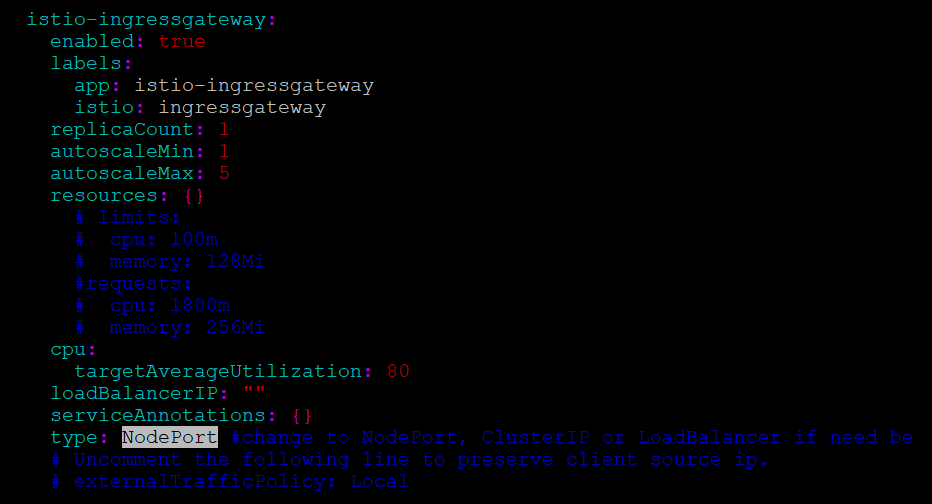
# Grafana, Prometheus & Disable Galley

cd istio-1.0.5

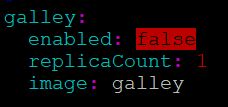
vi install/kubernetes/helm/istio/values.yaml

The Values the need to be changed in values.yaml

1. Istio-Ingressgateway needs to be changed to a NodePort.



1. Disable the Galley Service

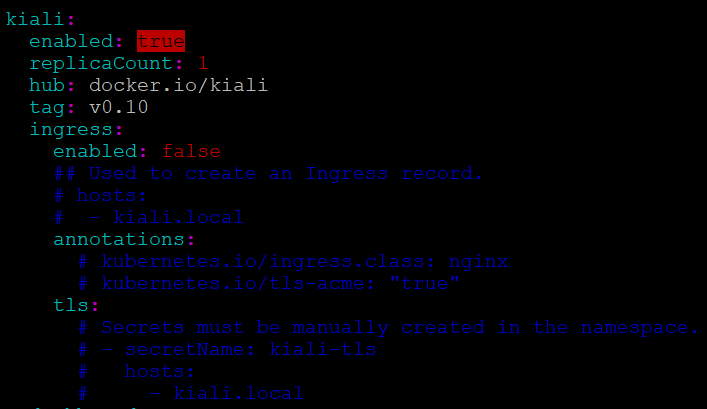


1. Enable the Telemetry – Grafana, Prometheus.

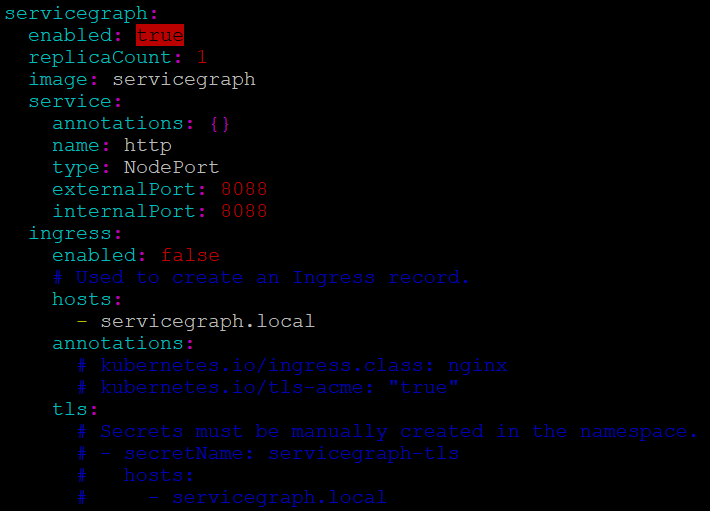


1. Enable Tracing – Jaeger, Kiali





1. Enable Servicegraph & Change Type to a NodePort



Once these Changes are done, save the changes and continue with the following Steps.

# Create a Helm Template

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

# Istio has Custom Resource Definitions (CRDs) to Create Gateway Routing etc.

kubectl apply -f install/kubernetes/helm/istio/templates/crds.yaml

# Create the Istio-System Namespace

kubectl create namespace istio-system

#Apply the Helm Template

kubectl apply -f $HOME/istio.yaml

#Check your Istio-System Pods & Services to Verify All Istio Enabled Services

kubectl get svc -n istio-system

watch -n 1 kubectl get pods -n istio-system

# Change Service Types - Kiali, ServiceGraph & Jaeger-query 🡪 NodePort

# This Allows you to see the Dashboard for these Services

#Change Type ClusterIP 🡪 NodePort

kubectl edit svc kiali -n istio-system

#Change Type ClusterIP 🡪 NodePort

kubectl edit svc servicegraph -n istio-system

#Change Type ClusterIP 🡪 NodePort

kubectl edit svc jaeger-query -n istio-system

#Check your Istio-System Pods & Services to Verify All Istio Enabled Services

kubectl get svc -n istio-system

# Grafana & Prometheus have a Default NodePort Set in the istio-ingressgateway

The default NodePort for Grafana and Prometheus on the Istio-Ingressgateway Service

C:\Users\jogasa8.USWIN\Documents\Sarat\Grafana-Prometheus-NodePort.PNG

### This Completes the Installation of Istio ####

Uninstall Istio Setup

############ Istio Uninstall Steps ################

#Terminate all the resources using the YAML used to install

kubectl delete -f $HOME/istio.yaml