#### Set up a service mesh using Istio in a Kubernetes environment. This involves installing Istio, enabling the service mesh with secure TLS for two nginx deployments (nginx-v1 and nginx-v2), and configuring various Istio components like PeerAuthentication, DestinationRule, and VirtualService. Additionally, got access and utilize Istio's observability features, such as the Kiali dashboard, to monitor and manage the service mesh. Also integrated Prometheus and Grafana dashboards with Kiali to visualize graphs of the labelled namespace and its pods.

#### This documentation includes all steps, commands, YAML configurations, and theoretical explanations from the initial setup to accessing the Kiali dashboard and integrating it with Grafana and Prometheus.

#### Istio Overview

Istio is an open-source service mesh that provides a way to control how microservices share data with one another. It offers features such as load balancing, service-to-service authentication, monitoring, and more.

#### Sidecar Proxy

In an Istio-enabled Kubernetes cluster, each service has an Envoy proxy running alongside it in a pod. This is referred to as the sidecar proxy. The sidecar handles all inbound and outbound traffic to the service.

#### Mutual TLS (mTLS)

mTLS ensures that traffic between services is encrypted and that both the client and server mutually authenticate each other.

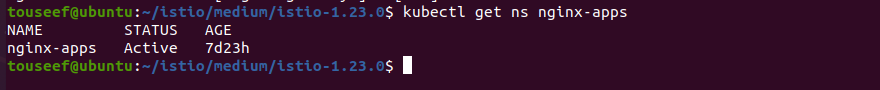
**1. Prerequisites**

* Kubernetes Cluster: A running Kubernetes cluster.
* kubectl: Installed and configured to interact with the Kubernetes cluster.
* Istio: Downloaded and installed.

**2. Create Nginx Deployments**

To separate the nginx deployments, create a dedicated namespace:

**$ kubectl create namespace nginx-apps**



Create and deploy Nginx Applications:

**Nginx-v1 Deployment**

Create a deployment YAML file for the first version of the nginx application:

# nginx-v1.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-v1

namespace: nginx-apps

labels:

app.kubernetes.io/name: nginx-v1

spec:

replicas: 1

selector:

matchLabels:

app.kubernetes.io/name: nginx-v1

template:

metadata:

labels:

app.kubernetes.io/name: nginx-v1

spec:

containers:

- image: nginx

name: nginx-v1

command:

- /bin/sh

- -c

- "echo 'welcome to my nginx version ONE!' > /usr/share/nginx/html/index.html && nginx -g 'daemon off;'"

ports:

- containerPort: 80

dnsConfig:

options:

- name: ndots

value: "2"

---

apiVersion: v1

kind: Service

metadata:

name: nginx-v1

namespace: nginx-apps

labels:

app.kubernetes.io/name: nginx-v1

spec:

selector:

app.kubernetes.io/name: nginx-v1

ports:

- name: http

port: 8080

protocol: TCP

targetPort: 80

type: LoadBalancer

**Nginx-v2 Deployment**

Similarly, create a deployment for the second version:

# nginx-v2.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-v2

namespace: nginx-apps

labels:

app.kubernetes.io/name: nginx-v2

spec:

replicas: 1

selector:

matchLabels:

app.kubernetes.io/name: nginx-v2

template:

metadata:

labels:

app.kubernetes.io/name: nginx-v2

spec:

containers:

- image: nginx

name: nginx-v2

command:

- /bin/sh

- -c

- "echo 'welcome to my nginx version TWO!' > /usr/share/nginx/html/index.html && nginx -g 'daemon off;'"

ports:

- containerPort: 80

dnsConfig:

options:

- name: ndots

value: "2"

---

apiVersion: v1

kind: Service

metadata:

name: nginx-v2

namespace: nginx-apps

labels:

app.kubernetes.io/name: nginx-v2

spec:

selector:

app.kubernetes.io/name: nginx-v2

ports:

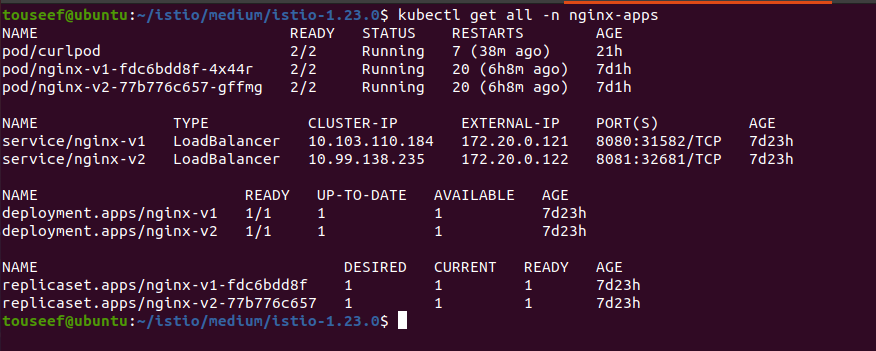
- name: http

port: 8081

protocol: TCP

targetPort: 80

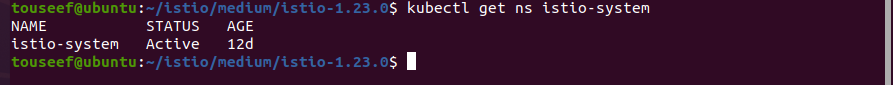
type: LoadBalancer



**3. Install Istio**

Create the namespace, istio-system, for the Istio components:

**$ kubectl create namespace istio-system**



Istio installation using manifest:

Download and install Istio on your system:

**$ curl -L https://istio.io/downloadIstio | ISTIO\_VERSION=1.23.0 sh -**

**$ cd istio-1.23.0**

**$ export PATH=$PWD/bin:$PATH**

Install Istio's base components:

**$ istioctl install --set profile=demo -y**

Verify that Istio is installed:

**$ kubectl get pods -n istio-system**

Istio SideCar Mode installation using Helm:

Install the Istio base chart which contains cluster-wide Custom Resource Definitions (CRDs) which must be installed prior to the deployment of the Istio control plane:

**$ helm install istio-base istio/base -n istio-system --set defaultRevision=default**

Install the Istio discovery chart which deploys the istiod service:

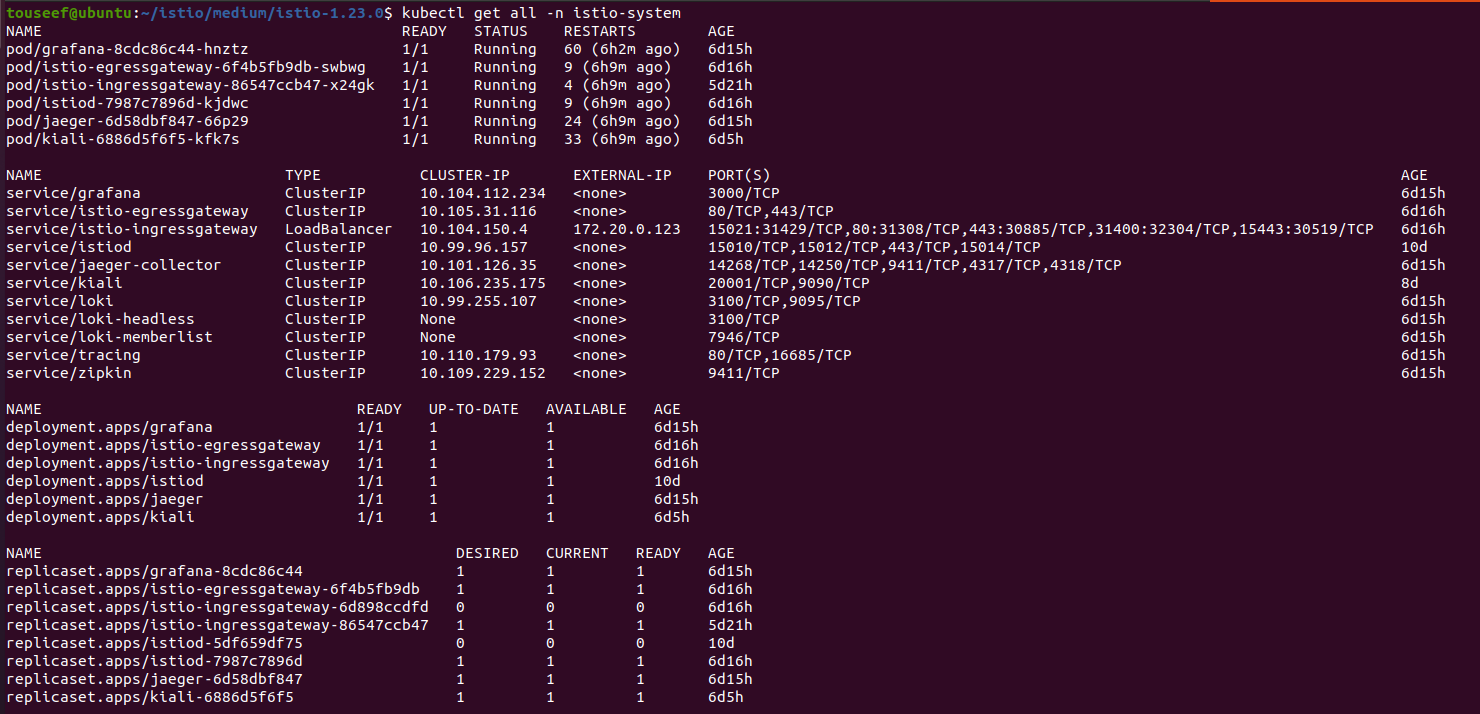
**$ helm install istiod istio/istiod -n istio-system --wait**

Validate the CRD installation with the helm ls command:

**$ helm ls -n istio-system**

Install an ingress gateway:

**helm install istio-ingress istio/gateway -n istio-ingress –wait**



**4. Labelling the Namespace with Istio**

Remove the Existing Label

**$ kubectl label namespace nginx-apps istio-injection-**

Verify the Label Removal

Check to ensure the label has been removed:

**$ kubectl get namespace nginx-apps --show-labels**

Reapply the Label

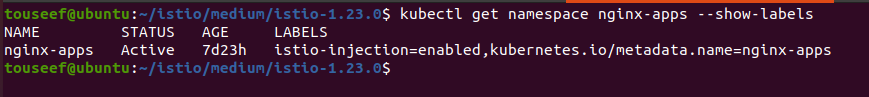
Now, reapply the label:

**$ kubectl label namespace nginx-apps istio-injection=enabled**

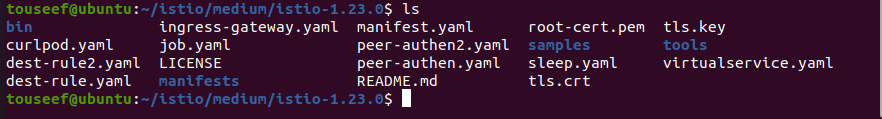
Verify the Label Application

Check again to ensure the label has been applied:

**$ kubectl get namespace nginx-apps --show-labels**

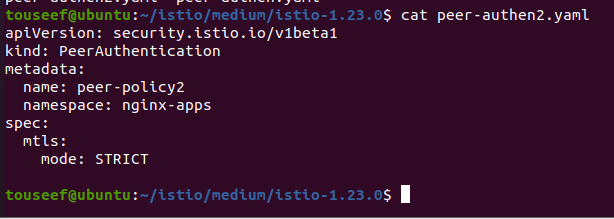


**5. Apply Istio Policies for mTLS**



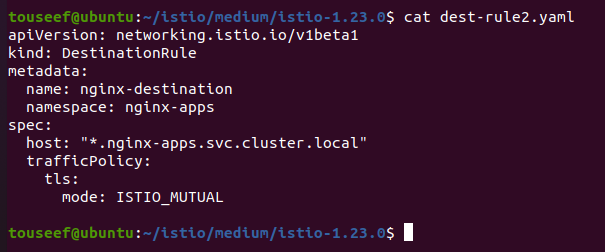
Peer Authentication Policy

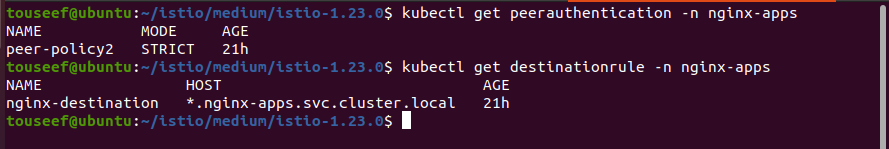
Create a PeerAuthentication resource to enforce mTLS:



Destination Rule

Create a DestinationRule to define the traffic policy:

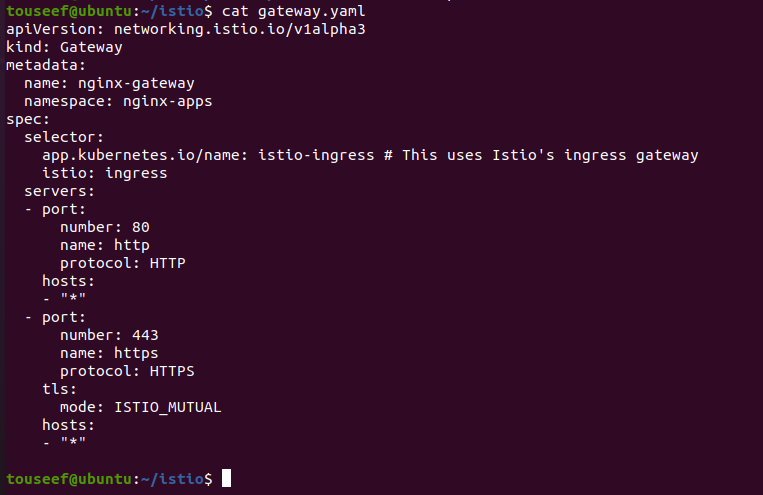




**6. Create a Virtual Service and Gateway**

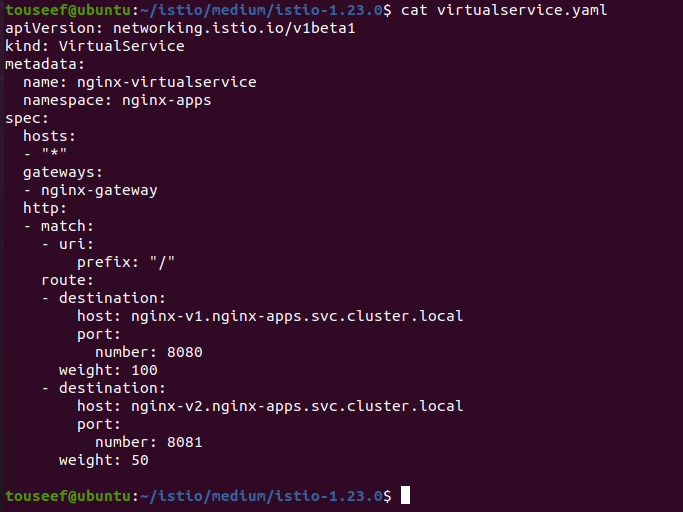
Gateway Configuration

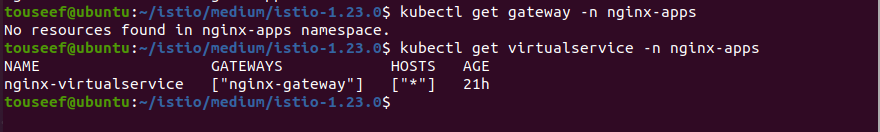
Create an Istio Gateway resource to handle external traffic:



Virtual Service

Define a VirtualService to route traffic to the nginx services:





**7. Verification and Testing**

Test the configuration by deploying a curl pod in the nginx-apps namespace and issuing curl requests to your services. Creates test pod:

**$ kubectl run curlpod --image=curlimages/curl -n nginx-apps --command -- sleep 3600**

Test from a Pod inside the Mesh

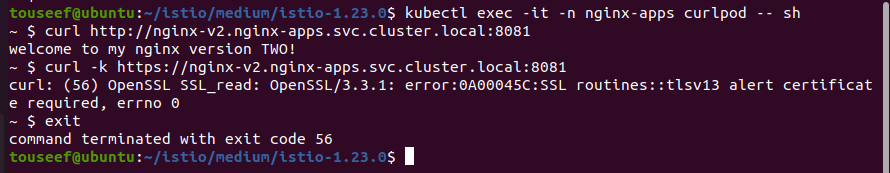
**$ kubectl exec -it -n nginx-apps curlpod -- sh**

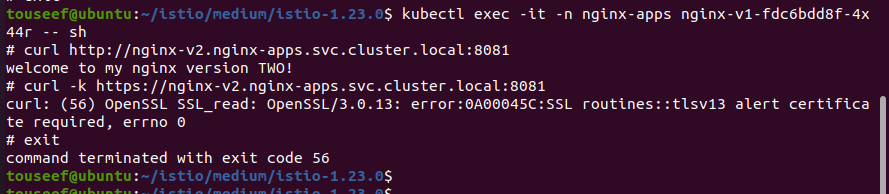
**$ curl http://nginx-v1.nginx-apps.svc.cluster.local:8080**

**$ curl http://nginx-v2.nginx-apps.svc.cluster.local:8081**

**$ curl -k https://nginx-v1.nginx-apps.svc.cluster.local:8080**

**$ curl -k https://nginx-v2.nginx-apps.svc.cluster.local:8081**





Test Using TLS Certificates

**$ kubectl exec -it -n nginx-apps curlpod -- curl --key /etc/certs/key.pem --cert /etc/certs/cert-chain.pem --cacert /etc/certs/root-cert.pem https://nginx-v1.nginx-apps.svc.cluster.local:8080**

**$ kubectl exec -it -n nginx-apps curlpod -- curl --key /etc/certs/key.pem --cert /etc/certs/cert-chain.pem --cacert /etc/certs/root-cert.pem https://nginx-v2.nginx-apps.svc.cluster.local:8081**

**8. Access Kiali Dashboard**

Install and Access Grafana and Prometheus

Install Grafana and Prometheus: If you haven’t already installed Grafana and Prometheus, you can do so using Helm:

**$ helm install prometheus prometheus-community/prometheus --namespace monitoring**

**$ helm install grafana grafana/grafana --namespace monitoring**

Port Forwarding to Access Dashboards: Use kubectl port-forward to access the Grafana and Prometheus dashboards locally:

**$ kubectl port-forward --namespace monitoring svc/grafana 3000:80**

**$ kubectl port-forward --namespace monitoring svc/prometheus-server 9090:80**

Access Grafana at http://localhost:3000

Access Prometheus at http://localhost:9090

Integrate Kiali with Prometheus and Grafana

Configure Prometheus as a Data Source in Grafana:

* Open Grafana in your browser.
* Go to Configuration > Data Sources.
* Click Add data source and select Prometheus.
* Set the URL to **http://prometheus-server.monitoring.svc.cluster.local**
* Click Save & Test.

Configure Kiali to Use Prometheus and Grafana:

* Edit the Kiali ConfigMap:

**$ kubectl edit configmap kiali -n istio-system**

* Add the following configuration under external\_services:

**external\_services:**

**prometheus:**

**url: "http://prometheus-server.monitoring.svc.cluster.local"**

**grafana:**

**url: "http://grafana.monitoring.svc.cluster.local"**

* Save and exit the editor. Kiali will reload the configuration automatically.

Port Forwarding to Access Kiali: Use kubectl port-forward to access the Kiali dashboard locally:

**kubectl port-forward -n istio-system svc/kiali 20001:20001**

* Access Kiali at http://localhost:20001

Visualize Graphs of Prometheus and Grafana

Verify Metrics in Prometheus:

* Open Prometheus in your browser.
* Use PromQL queries to verify that metrics from the nginx-apps namespace are being collected. For example:

**up{namespace="nginx-apps"}**

Create Grafana Dashboard:

* Open Grafana in your browser.
* Go to Dashboards > New Dashboard.
* Click Add new panel.
* In the Query section, enter a PromQL query to fetch metrics from the nginx-apps namespace. For example:

**$ rate(http\_requests\_total{namespace="nginx-apps"}[5m])**

* Customize the visualization options as needed.
* Click Apply to save the panel.
* Click Save Dashboard to save the entire dashboard.

Visualize Metrics in Kiali:

* Open Kiali in your browser.
* Navigate to the Graph section.
* Select the nginx-apps namespace to visualize the service mesh graph and metrics.

**References:**

<https://istio.io/latest/docs/setup/install/helm/>

<https://istio.io/latest/docs/examples/bookinfo/>

<https://medium.com/google-cloud/istio-service-mesh-101-part-1-3-f07a8fedeea8>

<https://istio.io/latest/docs/ops/integrations/kiali/#installation>

<https://kiali.io/docs/installation/installation-guide/install-with-helm/>