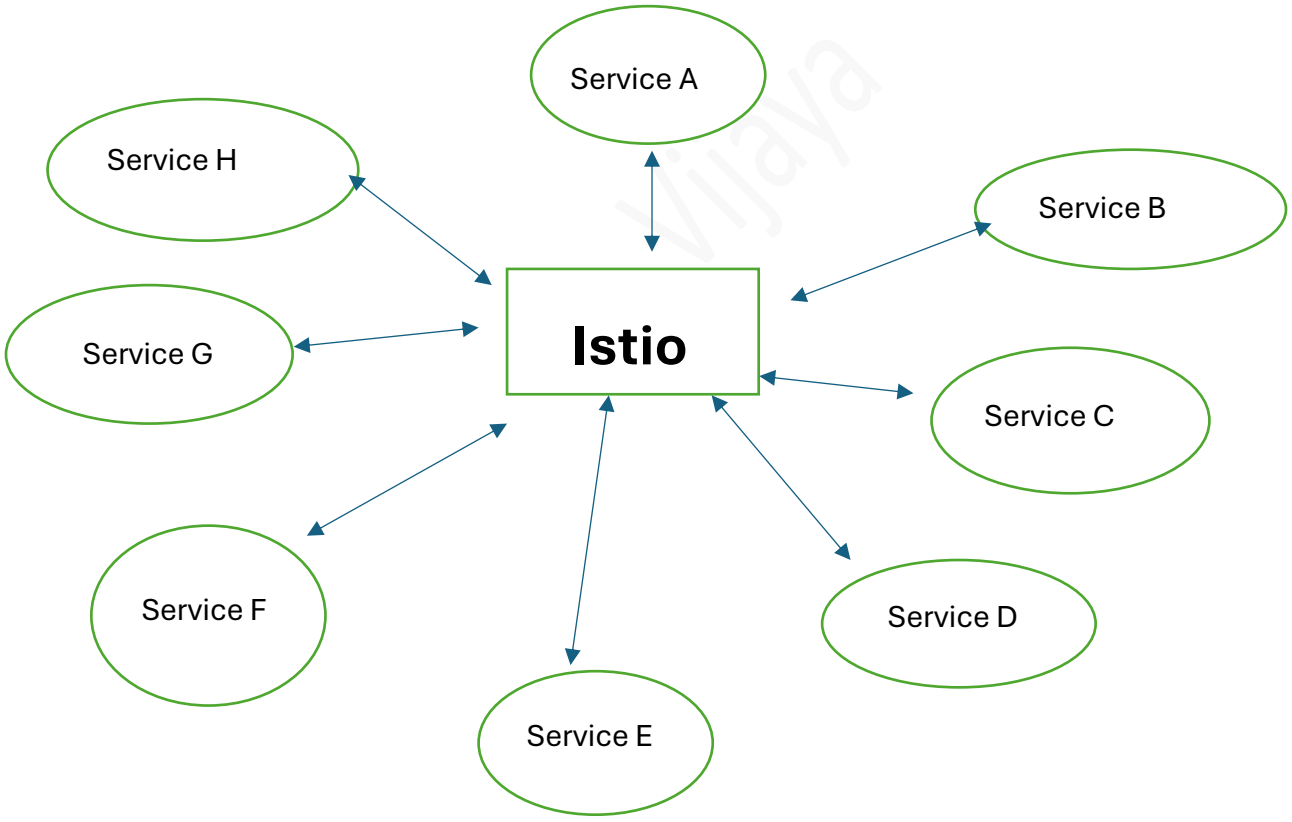
# Understanding Istio and Service Mesh: A Hands-on Implementation Guide

## What is a Service Mesh?

A Service Mesh is a dedicated infrastructure layer that facilitates service-to-service communication (East-West traffic) by providing structured networking, security, and observability. When Kubernetes services need to communicate, they require various capabilities like networking, security policies, authentication, authorization, observability, and rate limiting. A Service Mesh provides all these capabilities out of the box, making service communication significantly easier.

## Istio Service Mesh Overview



Istio makes service communication transparent and efficient without requiring code changes. It's particularly valuable in large-scale applications with complex networking requirements and strict security compliance needs.

### Key Components:

**Istio control plane** is the brain behind Istio that controls all the traffic, security and routing.

**Envoy Proxy** is added to every pod as a side car container. It controls the traffic in and out of pods. Proxy takes the configuration from control plane and control the networking layer. When a request comes to API server to create a pod, it uses admission controllers to inform Istio to inject a side car container.

## Istio Service Mesh Hands-on Project with Sidecar Injection, Traffic management and Sample website Hosting

To integrate a sample website into this Istio hands-on project, we will deploy a sample website provided in Istio documentation. https://istio.io/latest/docs/setup/getting-started/

Follow the documentation to install Istio and to add a namespace label to instruct Istio to automatically inject Envoy sidecar proxies when application is deployed.

*kubectl label namespace default istio-injection=enabled*

As we have configured Istio to inject sidecar containers into any application that deployed in default namespace, let us deploy the BookInfo sample application in default namespace.

The Bookinfo application is broken into four separate microservices:

- Productpage: Calls details and reviews services
- Details: Contains book information
- Reviews: Contains book reviews and calls ratings service
- Ratings: Provides book ranking information

The reviews service has three versions:

- v1: Basic version without ratings
- v2: Shows ratings with black stars
- v3: Shows ratings with red stars

*kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.23/samples/bookinfo/platform/kube/bookinfo.yaml*

```
root@controlplane ~/istio-1.23.2 ➜  kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.23/samples/bookinfo/platform/k
ube/bookinfo.yaml
service/details created
serviceaccount/bookinfo-details created
deployment.apps/details-v1 created
service/ratings created
serviceaccount/bookinfo-ratings created
deployment.apps/ratings-v1 created
service/reviews created
serviceaccount/bookinfo-reviews created
deployment.apps/reviews-v1 created
deployment.apps/reviews-v2 created
deployment.apps/reviews-v3 created
service/productpage created
serviceaccount/bookinfo-productpage created
deployment.apps/productpage-v1 created
```

```
root@controlplane ~/istio-1.23.2 ➜  kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
details-v1-7fbb49ff8d-zng79       2/2     Running   0          38s
productpage-v1-76dc96cc5c-8hmtz   2/2     Running   0          37s
ratings-v1-f857b954d-wfdrz        2/2     Running   0          38s
reviews-v1-79bf45d856-x7gvt       2/2     Running   0          38s
reviews-v2-5d58bc596-cp7bq        2/2     Running   0          38s
reviews-v3-7696657c5b-mmn92       2/2     Running   0          37s
```

When we check pods, each pod has 2/2 means running 2 containers inside each pod. Describing the pod shows the container details.

```
Containers:
  reviews:
    Container ID:   containerd://90bae4f24fce3bac58462c20f45f9e229a6993daeacdcc42baea4ee719aba1ff
    Image:          docker.io/istio/examples-bookinfo-reviews-v3:1.20.1
    Image ID:       docker.io/istio/examples-bookinfo-reviews-v3@sha256:90e66120ba5b471f99c102d87d44799b1df7913ab8bafab6048739d1746d4e6d
    Port:           9080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Thu, 17 Oct 2024 17:45:40 +0000
    Ready:          True
    Restart Count:  0
    Environment:
      LOG_DIR:  /tmp/logs
    Mounts:
      /opt/ibm/wlp/output from wlp-output (rw)
      /tmp from tmp (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-9pkm5 (ro)
  istio-proxy:
    Container ID:  containerd://50a1db859f89e0f778f5368885ee5ca9852cce9650239b540ec7c8d4ab194be7
    Image:         docker.io/istio/proxyv2:1.23.2
    Image ID:      docker.io/istio/proxyv2@sha256:2876cfc2fdf47e4b9665390ccc9ccf2bf913b71379325b8438135c9f35578e1a
    Port:          15090/TCP
```

Install the Gateway API CRDs.

Create a Kubernetes Gateway for the Bookinfo application:

```
root@controlplane ~/istio-1.23.2 ✖ kubectl get crd gateways.gateway.networking.k8s.io &> /dev/null || \
> { kubectl kustomize "github.com/kubernetes-sigs/gateway-api/config/crd?ref=v1.1.0" | kubectl apply -f -; }
customresourcedefinition.apiextensions.k8s.io/gatewayclasses.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/gateways.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/grpcroutes.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/httproutes.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/referencegrants.gateway.networking.k8s.io created

root@controlplane ~/istio-1.23.2 ➜  kubectl apply -f samples/bookinfo/gateway-api/bookinfo-gateway.yaml
gateway.gateway.networking.k8s.io/bookinfo-gateway created
httproute.gateway.networking.k8s.io/bookinfo created
```

By default, Istio creates a LoadBalancer service for a gateway. As we will access this gateway by a tunnel, we don't need a load balancer.

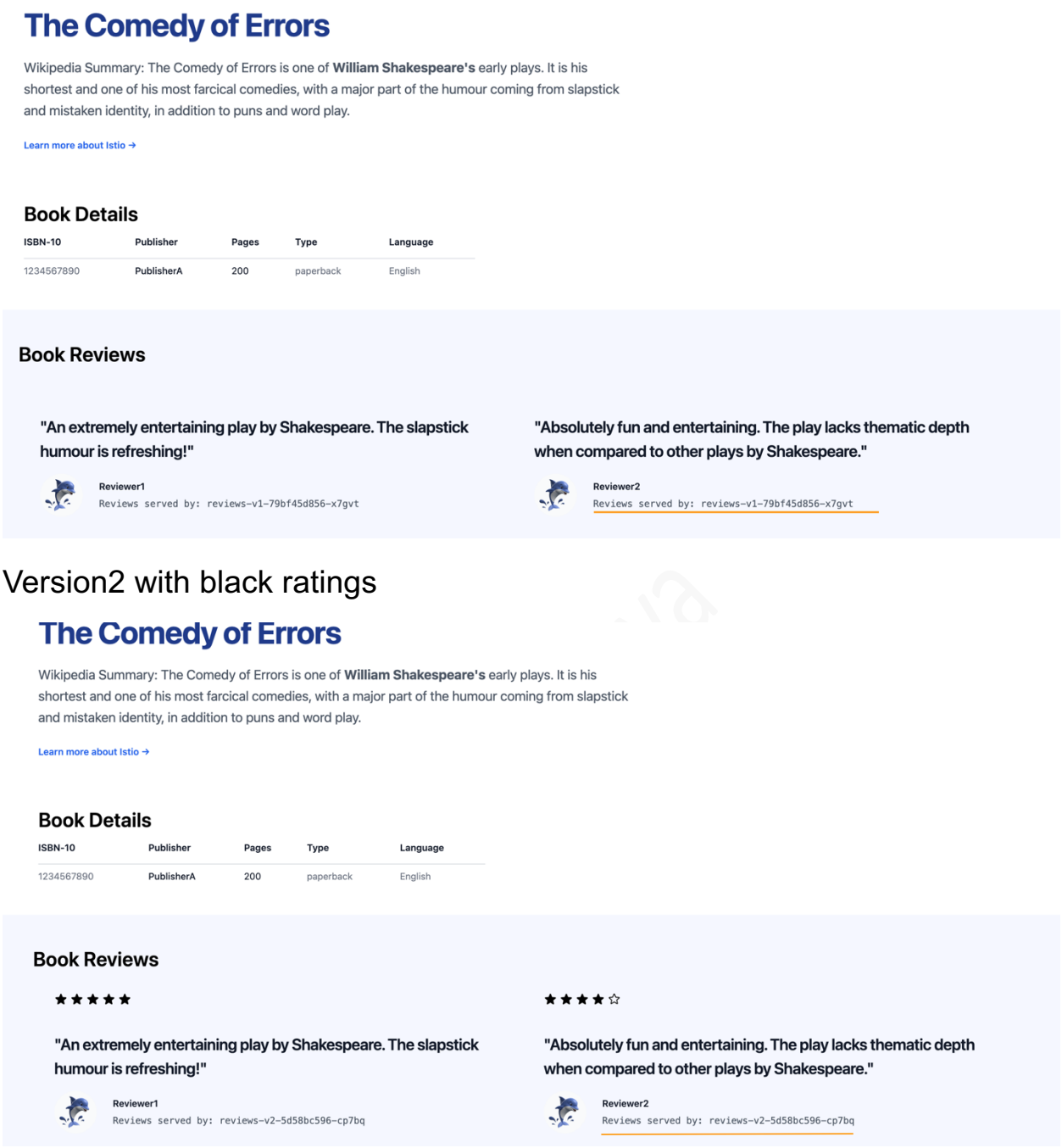
```
root@controlplane ~/istio-1.23.2 ➜  k get po
NAME                                    READY   STATUS    RESTARTS   AGE
bookinfo-gateway-istio-f9895cfff-2gp98   1/1     Running   0          9m32s
details-v1-7fbb49ff8d-zng79              2/2     Running   0          31m
productpage-v1-76dc96cc5c-8hmtz          2/2     Running   0          31m
ratings-v1-f857b954d-wfdrz               2/2     Running   0          31m
reviews-v1-79bf45d856-x7gvt              2/2     Running   0          31m
reviews-v2-5d58bc596-cp7bq               2/2     Running   0          31m
reviews-v3-7696657c5b-mmn92              2/2     Running   0          31m

root@controlplane ~/istio-1.23.2 ➜  k get svc
NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
bookinfo-gateway-istio   LoadBalancer   10.99.84.245     <pending>     15021:30060/TCP,80:30099/TCP   9m34s
details                 ClusterIP      10.109.167.0     <none>        9080/TCP                     31m
kubernetes              ClusterIP      10.96.0.1        <none>        443/TCP                      39m
productpage             ClusterIP      10.99.243.161    <none>        9080/TCP                     31m
ratings                 ClusterIP      10.110.120.132   <none>        9080/TCP                     31m
reviews                 ClusterIP      10.100.53.139    <none>        9080/TCP                     31m
```

Update the gateway to NodePort service and access the application.

```
root@controlplane ~/istio-1.23.2 ✖ kubectl annotate gateway bookinfo-gateway networking.istio.io/service-type=NodePort --namespace=default
--overwrite=true
gateway.gateway.networking.k8s.io/bookinfo-gateway annotate

root@controlplane ~/istio-1.23.2 ➜  k get svc
NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
bookinfo-gateway-istio   NodePort    10.99.84.245     <none>        15021:32594/TCP,80:30665/TCP   11m
details                 ClusterIP   10.109.167.0     <none>        9080/TCP                     33m
kubernetes              ClusterIP   10.96.0.1        <none>        443/TCP                      41m
productpage             ClusterIP   10.99.243.161    <none>        9080/TCP                     33m
ratings                 ClusterIP   10.110.120.132   <none>        9080/TCP                     33m
reviews                 ClusterIP   10.100.53.139    <none>        9080/TCP                     33m
```

Keep refreshing the page to see 3 versions of reviews microservice.

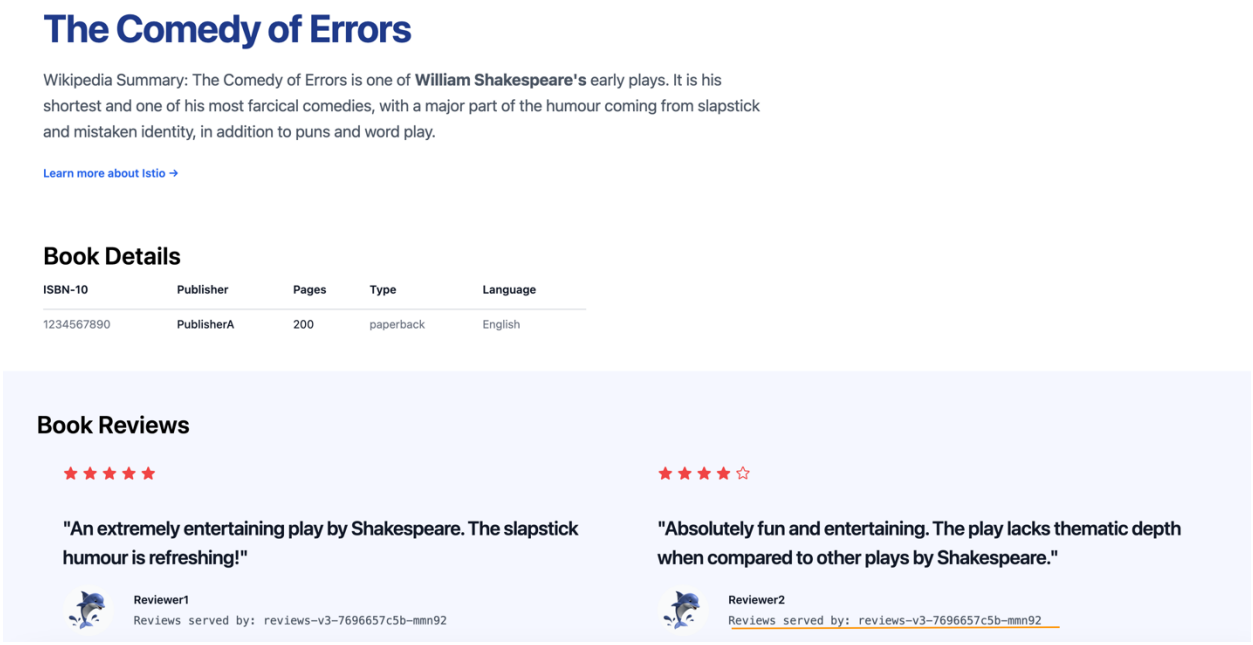
Version1 with no ratings




Version2 with black ratings




Version3 with red ratings




We have installed Istio, enabled side car injection and deployed sample application.

## Mutual TLS with Istio

Now we have the services up and running, anyone with clusterip can access the service or API as follows. By default, Istio runs mTLS in permissive mode.

```
root@controlplane ~/istio-1.23.2 → curl 10.99.243.161:9080/api/v1/products
[{"id": 0, "title": "The Comedy of Errors", "descriptionHtml": "<a href=\"https://en.wikipedia.org/wiki/The_Comedy_of_Errors
mmary</a>: The Comedy of Errors is one of <b>William Shakespeare's</b> early plays. It is his shortest and one of his most f
s, with a major part of the humour coming from slapstick and mistaken identity, in addition to puns and word play."}]
root@controlplane ~/istio-1.23.2 →
```

We want to restrict the access to service to service only. Make the mtls-mode strict. Istio side car container will reject the connection with no certificate.

```yaml
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: mtls-mode
  namespace: default
spec:
  mtls:
    mode: STRICT
~
~
~
~
```

```
root@controlplane ~/istio-1.23.2 ➜  vi mlts.yml

root@controlplane ~/istio-1.23.2 ➜  k apply -f mlts.yml
peerauthentication.security.istio.io/mtls-mode created

root@controlplane ~/istio-1.23.2 ➜  curl 10.99.243.161:9080/api/v1/products
curl: (56) Recv failure: Connection reset by peer

root@controlplane ~/istio-1.23.2 ✖
```

## Canary deployment with Istio

Virtual service and destination rules help Istio in traffic management. Let us add virtual service and rules to use only reviews v1 service. Add a virtual service for reviews to route to destination v1 all the time.

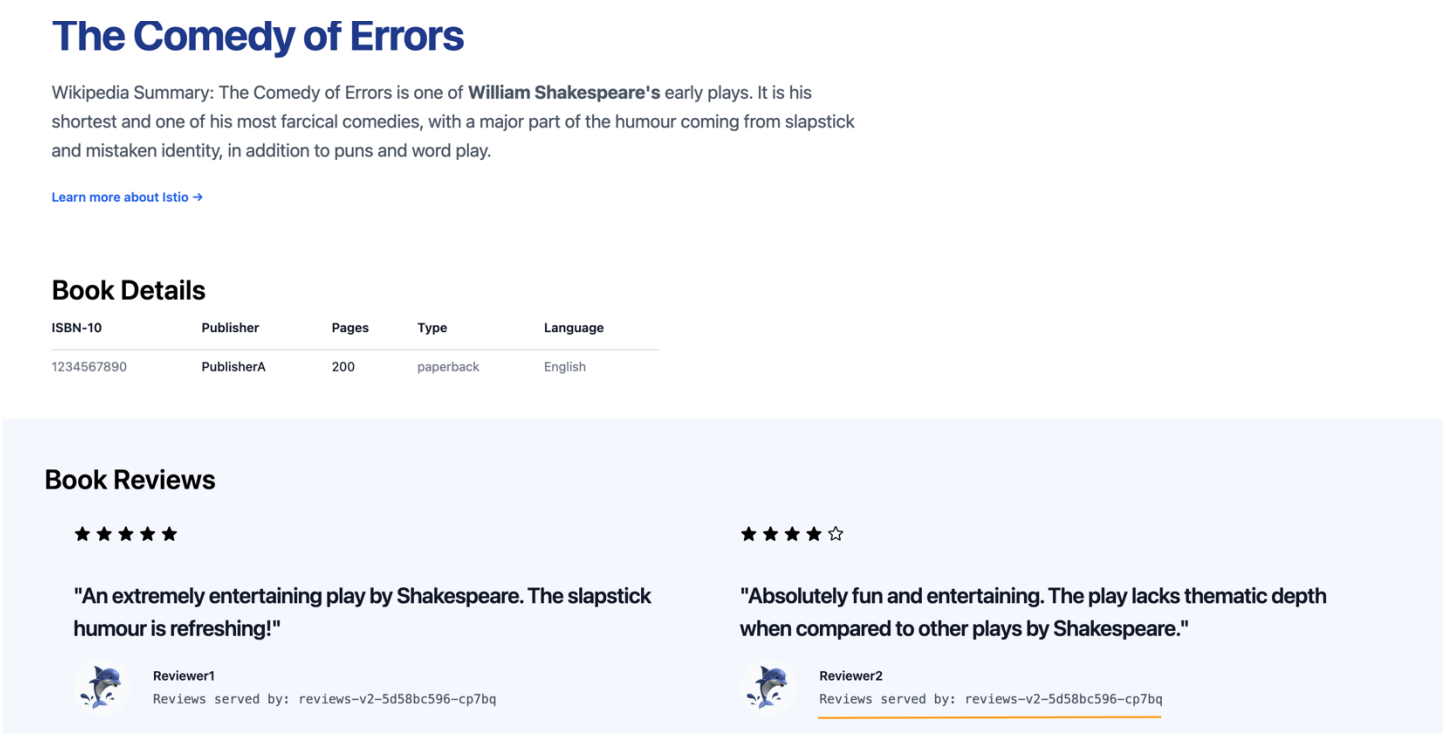
```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - route:
    - destination:
```

```
root@controlplane ~/istio-1.23.2 ➜  vi goto-v1.yml

root@controlplane ~/istio-1.23.2 ➜  k apply -f goto-v1.yml
virtualservice.networking.istio.io/productpage created
virtualservice.networking.istio.io/reviews created
virtualservice.networking.istio.io/ratings created
virtualservice.networking.istio.io/details created

root@controlplane ~/istio-1.23.2 ➜  k get vs
NAME          GATEWAYS   HOSTS              AGE
details                  ["details"]        12s
productpage              ["productpage"]    12s
ratings                  ["ratings"]        12s
reviews                  ["reviews"]        12s

root@controlplane ~/istio-1.23.2 ➜
```

Apply destination rules

```
root@controlplane ~/istio-1.23.2 ➜  kubectl apply -f samples/bookinfo/networking/destination-rule-all.yaml
destinationrule.networking.istio.io/productpage created
destinationrule.networking.istio.io/reviews created
destinationrule.networking.istio.io/ratings created
destinationrule.networking.istio.io/details created

root@controlplane ~/istio-1.23.2 ➜  kubectl get destinationrules -o yaml
apiVersion: v1
items:
- apiVersion: networking.istio.io/v1
  kind: DestinationRule
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
        {"apiVersion":"networking.istio.io/v1alpha3","kind":"DestinationRule","metadata":{"annotations":{},"name":"details","namespace":"de
fault"},"spec":{"host":"details","subsets":[{"labels":{"version":"v1"},"name":"v1"},{"labels":{"version":"v2"},"name":"v2"}]}}
    creationTimestamp: "2024-10-17T18:46:50Z"
    generation: 1
    name: details
    namespace: default
    resourceVersion: "8970"
    uid: a9dec692-2c82-46ab-9054-fc033bd00cc8
  spec:
```

Now accessing the application always access v1 of reviews service.




Now let us deploy virtual service to route 50% of traffic to reviews-v1 and 50% to review-v3

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
      weight: 50
    - destination:
        host: reviews
        subset: v3
      weight: 50
~
~
~
```

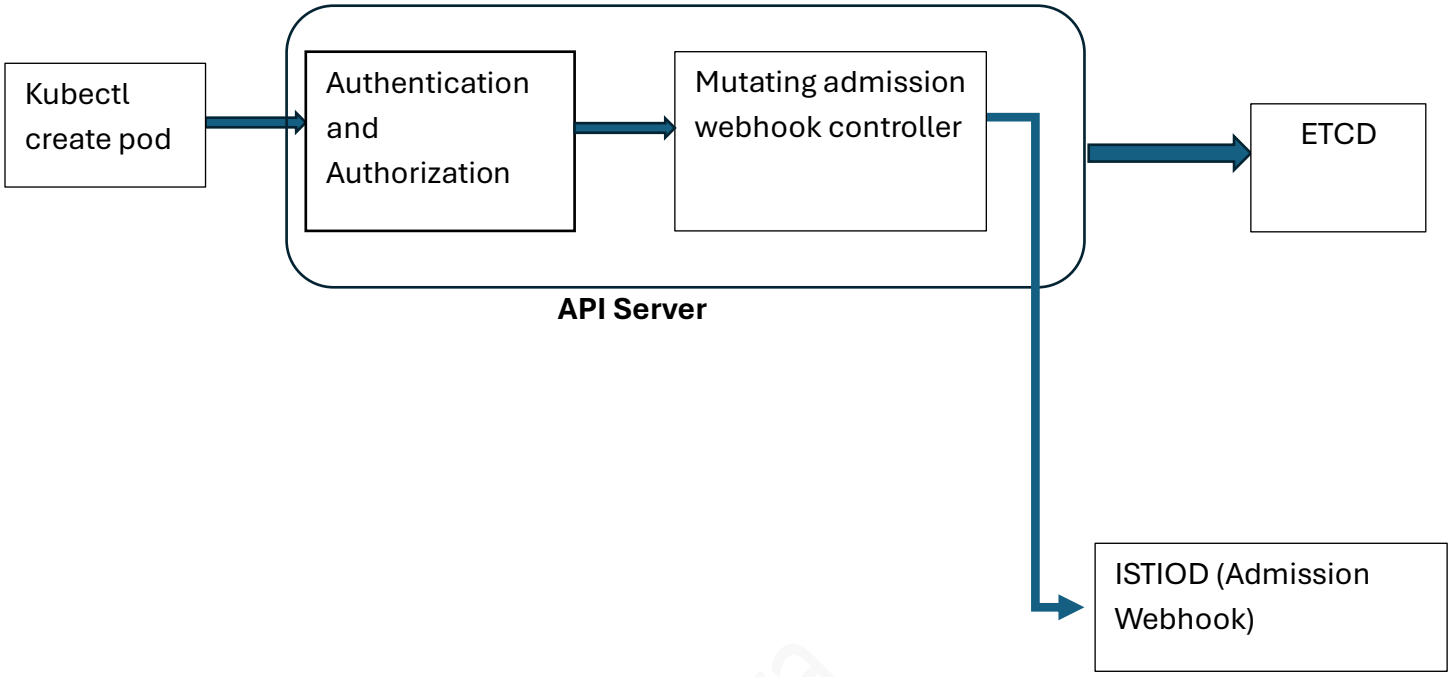
Accessing the application now shares the traffic between v1 and v3. Once the application is ready to switch, we can update the weight from 50 to 100 to the new version.

We can try different traffic management with Istio.https://istio.io/latest/docs/tasks/traffic-management/

### Admission Controller in Istio

To inject side car container, API server should notify Istio while creating any pod in the application. This process is called Dynamic admission control.

- When pod creation request comes to API server, it performs the Authentication and Authorization
- Mutating admission webhook controller notifies Istio admission webhook which is Istiod
- Istiod performs the mutation logic to inject side car containers
- Then object persisted in Etcd




Let us understand the webhook configuration that is doing this logic

*Kubectl get mutatingwebhookconfiguration istio-sidecar-injector*

Rules section specifies whenever a pod is created then forward the request to istiod service

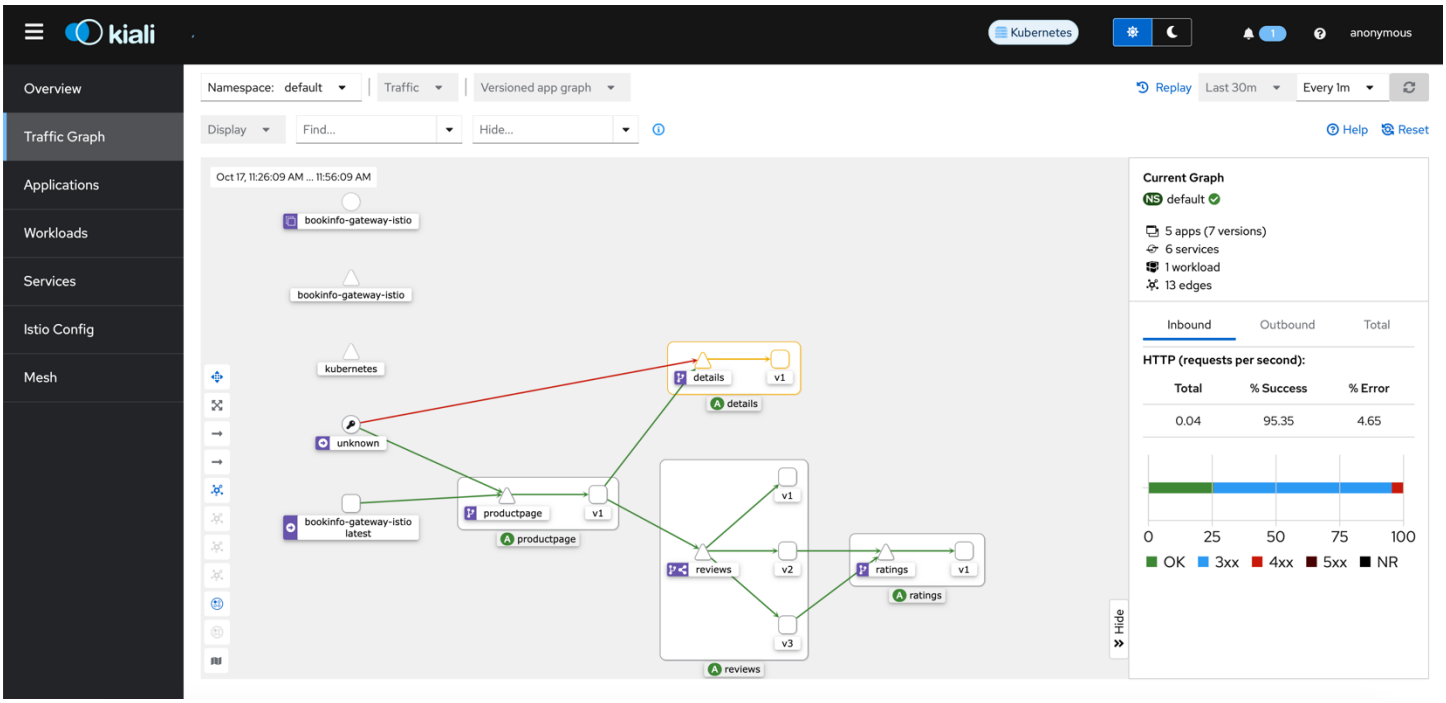
```
  name: object.sidecar-injector.istio.io
  namespaceSelector:
    matchLabels:
      istio.io/deactivated: never-match
  objectSelector:
    matchLabels:
      istio.io/deactivated: never-match
  reinvocationPolicy: Never
  rules:
  - apiGroups:
    - ""
    apiVersions:
    - v1
    operations:
    - CREATE
    resources:
    - pods
    scope: '*'
  sideEffects: None
  timeoutSeconds: 10
```

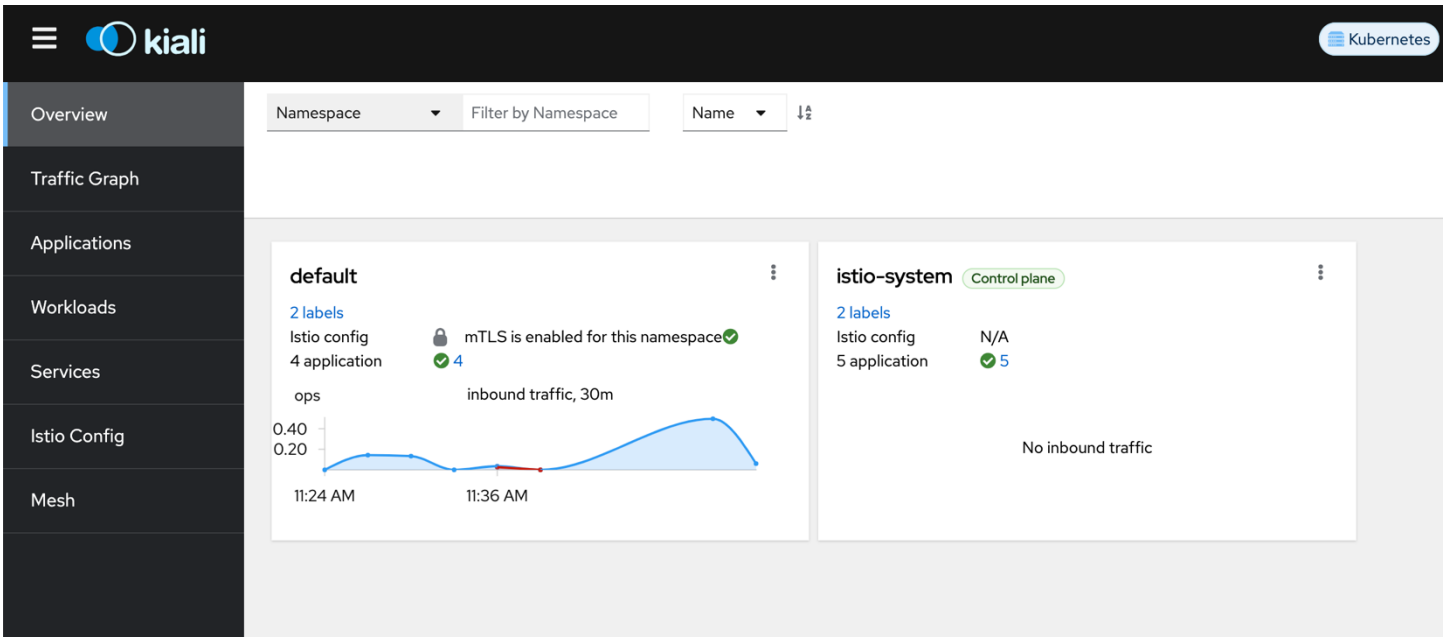
```
    service:
      name: istiod
      namespace: istio-system
      path: /inject
      port: 443
  failurePolicy: Fail
  matchPolicy: Equivalent
  name: rev.namespace.sidecar-injector.istio.io
  namespaceSelector:
    matchLabels:
      istio.io/deactivated: never-match
  objectSelector:
    matchLabels:
      istio.io/deactivated: never-match
  reinvocationPolicy: Never
  rules:
  - apiGroups:
    - ""
    apiVersions:
    - v1
    operations:
    - CREATE
```

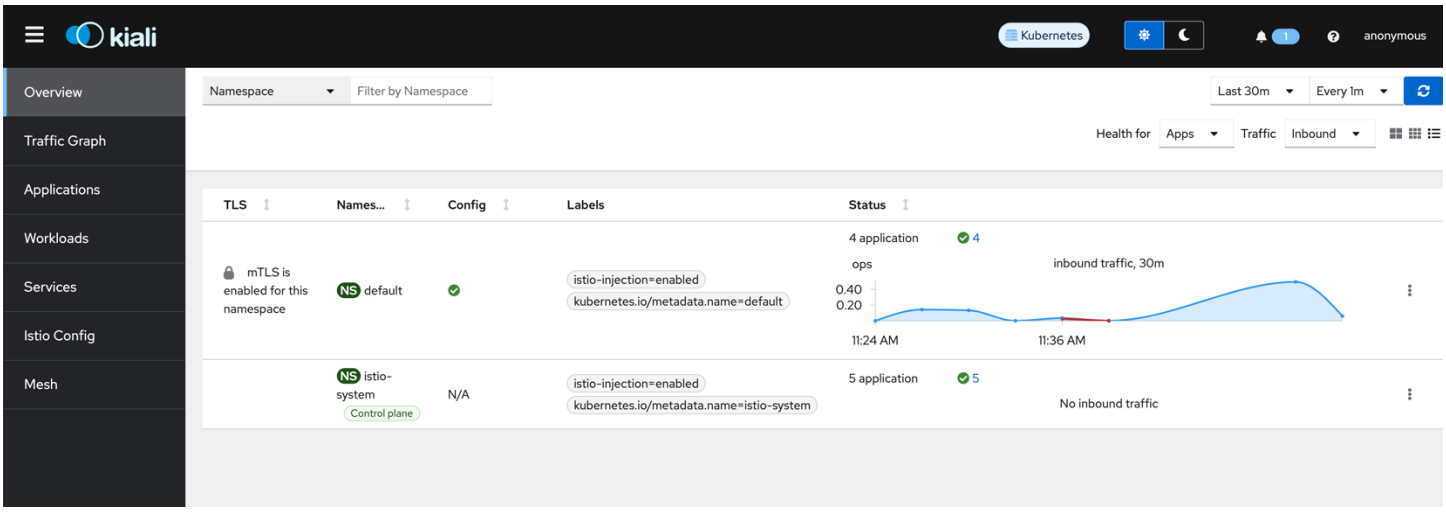
Istio also comes an in-built observability. Install Kiali.

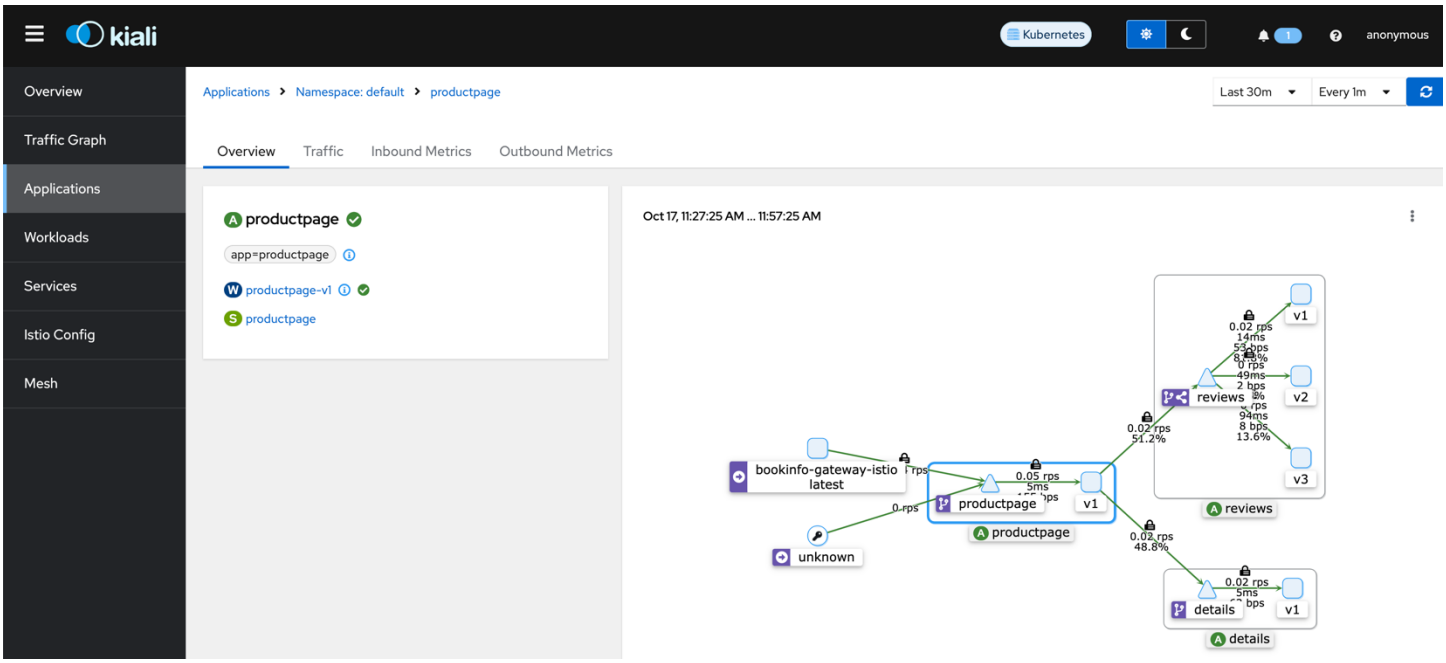
*kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.23/samples/addons/kiali.yaml*

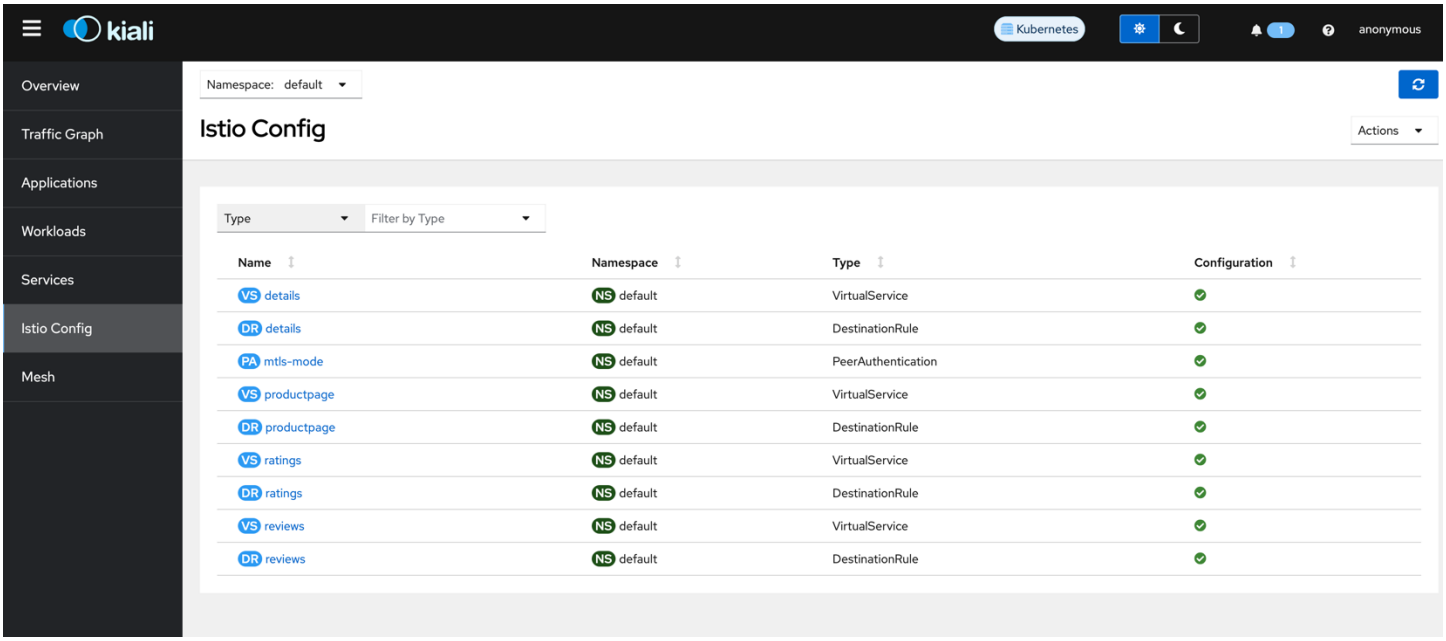
*istioctl dashboard kiali*

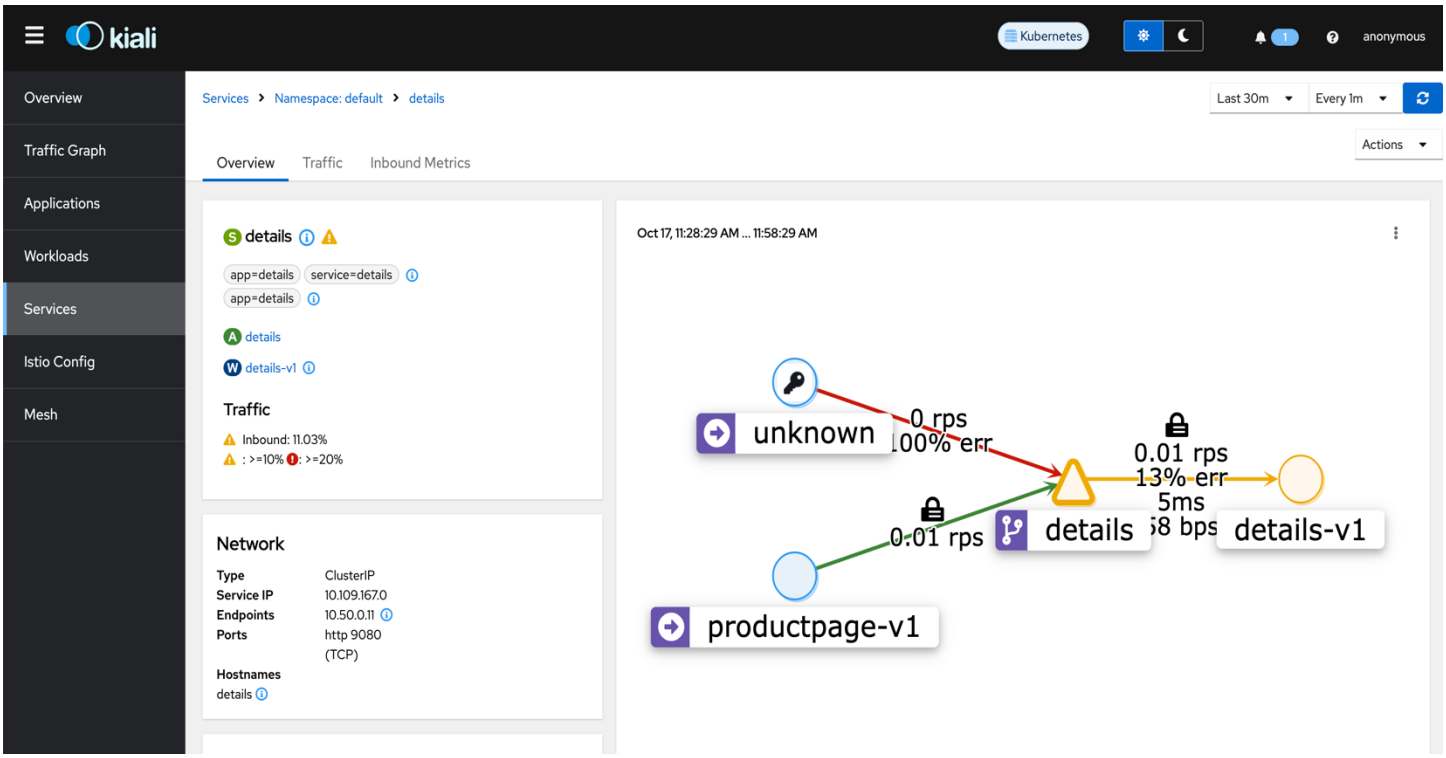

kiali
Kubernetes
anonymous
Overview
Traffic Graph
Applications
Workloads
Services
Istio Config
Mesh
Namespace: default
Traffic
Versioned app graph
Replay
Last 30m
Every 1m
Display
Find...
Hide...
Help
Reset
Oct 17, 11:26:09 AM ... 11:56:09 AM
bookinfo-gateway-istio
bookinfo-gateway-istio
kubernetes
unknown
bookinfo-gateway-istio
latest
productpage
v1
productpage
details
v1
details
reviews
v1
v2
v3
reviews
ratings
v1
ratings
Current Graph
default
5 apps (7 versions)
6 services
1 workload
13 edges
Inbound
Outbound
Total
HTTP (requests per second):
Total
% Success
% Error
0.04
95.35
4.65
0
25
50
75
100
OK
3xx
4xx
5xx
NR
Hide


kiali
Kubernetes
Overview
Traffic Graph
Applications
Workloads
Services
Istio Config
Mesh
Namespace
Filter by Namespace
Name
default
2 labels
Istio config
mTLS is enabled for this namespace
4 application
4
ops
inbound traffic, 30m
0.40
0.20
11:24 AM
11:36 AM
istio-system
Control plane
2 labels
Istio config
N/A
5 application
5
No inbound traffic


kiali
Kubernetes
anonymous
Overview
Traffic Graph
Applications
Workloads
Services
Istio Config
Mesh
Namespace
Filter by Namespace
Last 30m
Every 1m
Health for
Apps
Traffic
Inbound
TLS
Names...
Config
Labels
Status
mTLS is enabled for this namespace
default
istio-injection=enabled
kubernetes.io/metadata.name=default
4 application
4
ops
inbound traffic, 30m
0.40
0.20
11:24 AM
11:36 AM
istio-system
Control plane
N/A
istio-injection=enabled
kubernetes.io/metadata.name=istio-system
5 application
5
No inbound traffic


kiali
Kubernetes
anonymous
Overview
Traffic Graph
Applications
Workloads
Services
Istio Config
Mesh
Applications > Namespace: default > productpage
Last 30m
Every 1m
Overview
Traffic
Inbound Metrics
Outbound Metrics
productpage
app=productpage
productpage-v1
productpage
Oct 17, 11:27:25 AM ... 11:57:25 AM
bookinfo-gateway-istio
latest
unknown
productpage
v1
reviews
v1
v2
v3
details
v1
0.05 rps
0.02 rps
51.2%
0.02 rps
48.8%
0.02 rps
8 bps
13.6%
0.02 rps
14ms
49ms
94ms


kiali
Kubernetes
anonymous
Overview
Traffic Graph
Applications
Workloads
Services
Istio Config
Mesh
Namespace: default
Istio Config
Actions
Type
Filter by Type
Name
Namespace
Type
Configuration
details
default
VirtualService
details
default
DestinationRule
mtls-mode
default
PeerAuthentication
productpage
default
VirtualService
productpage
default
DestinationRule
ratings
default
VirtualService
ratings
default
DestinationRule
reviews
default
VirtualService
reviews
default
DestinationRule


kiali
Kubernetes
anonymous
Overview
Traffic Graph
Applications
Workloads
Services
Istio Config
Mesh
Services › Namespace: default › details
Last 30m
Every 1m
Actions
Overview
Traffic
Inbound Metrics
details
app=details
service=details
app=details
details
details-v1
Traffic
Inbound: 11.03%
: >=10% : >=20%
Network
Type ClusterIP
Service IP 10.109.167.0
Endpoints 10.50.0.11
Ports http 9080 (TCP)
Hostnames
details
Oct 17, 11:28:29 AM ... 11:58:29 AM
unknown
0 rps
100% err
0.01 rps
13% err
5ms
0.01 rps
details
details-v1
productpage-v1