

Prerequisites

1. Understanding of Kubernetes Working

Service Mesh

In simple terms, Service Mesh is for management of communication between microservices in a cluster.

Or you can say,

A service mesh is a dedicated infrastructure layer that you can add to your applications. It allows you to transparently add capabilities like observability, traffic management, and security, without adding them to your own code

Why ?

With problems in monolithic applications, we are moving to microservices, but they have their own problems to be handled.

Challenges with microservices:

1. Business Logic (BL) for every microservice like payment logic for payment service
2. Communication Configuration (COMM) eg endpoint of each service
3. Security Logic (SEC) including firewall rules, proxies
4. Retry Logic (RL)
5. Metrics & Tracing logic (MT)

This prevents developers to work on actual stuff and create these logic with additional required by each service. Adding metric and network logics complicates the services.

How does Service Mesh work?

1. Non Business logic is implemented using sidecar proxies
2. These act as third party applications configured by clustered operators with apis
3. This allow developers to focus on actual business logic
4. The Control plane of service mesh injects these sidecar proxies in every microservice pod.

This layer containing these proxies and control plane forms the service mesh.

Traffic Split:

When you want to divert a percent of traffic like in canary or in blue-green deployment to test new updated deployment you use this.

Istio

While Service Mesh is a pattern of infrastructure required to manage the network of the microservices, Istio is the implementation of Service Mesh Paradigm started by Google.

Important Points:

1. The Proxies or side cars used are called envoy proxies (independent open source project) used by Istio.
2. Control plane is called Istiod which manages every envoy proxy for its:-
 - a. Configuration
 - b. Discovery
 - c. Certificates
3. Earlier before version 1.5, there were components that increased number of pods that ran with Istio implementations:
 - a. Pilot : config data to proxies
 - b. Citadel : TLS certificates
 - c. Galley
 - d. Mixer : policy checks, telemetry of data
4. Now they are part of Istiod.
5. Control Plane + Data Plane = Istio

Working of Istio

1. Do not require to rewrite or adjust YAML files of K8s
 2. Istio has its own configuration separate from application
 3. Istio is configured with K8s YAML files
 4. No need to learn Istio specific language
-
- Istio uses Custom Resource Definitions (CRD) that extend Kubernetes API
 - CRDs are custom Kubernetes components/objects that integrate with third party technologies like Istio or Prometheus and are used like any other Kubernetes objects.
 - Istio configurations with CRD allows us:
 - Traffic routing

- Traffic Split
 - Service communication rules
 - Retry rules
- CRDs used
 - Virtual Service
 - How to route traffic to a destination
 - Destination Rule
 - On top of Virtual Service
 - Configure traffic rules, load balancers etc. what traffic is on this route
- We create CRDs, Configuration at control plane level.
- Istiod converts them to a form for envoys specific configurations and propagates to particular proxies. Configuration for proxies
- Proxies do not talk with control plane and do direct communication between them.

Uses of Istio

1. Configuration
2. Service Discovery: Dynamic detection
 - a. Internal Registry for Service and their endpoints
 - i. New microservice gets registered automatically
 - ii. No static configuration of endpoints
3. Certification Management
 - a. Generates certificates for all microservices.
 - b. Secure TLS communication between microservices.
4. Metrics and Tracing Data from envoy proxies
 - a. Gather Telemetry Data
 - b. Provide to monitoring servers for metrics of services
 - c. Tracing becomes easy

Istio Ingress Gateway

1. Entrypoint to cluster and an alternative to native nginx ingress controller.
2. Runs as separate pod acting as Load Balancer
3. Uses the Virtual Service to guide the traffic inside the cluster for microservice.
4. Configured using Gateway CRD.

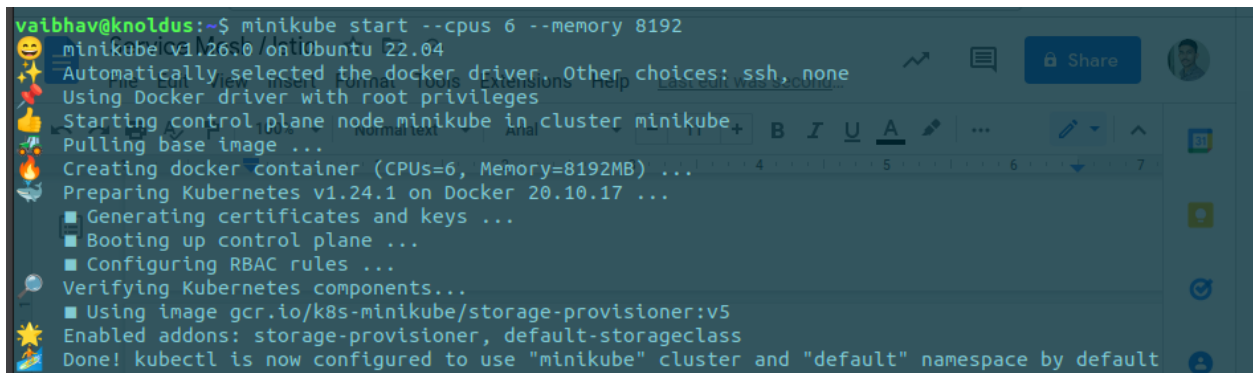
Traffic Flow in Istio

1. Request comes on web server for microservices
2. It comes on Istio Gateway which uses Virtual Service rules to direct the flow.

3. The request will reach the microservice pod.
4. The Envoy proxy will analyze the request and forward to the main container of the pods.
5. Now to communicate with other services , the main container will contact the envoy proxy.
6. The proxy will contact the services as per Virtual Service and other rules
7. Proxies gather the info and forward to control plane

Setting Up Istio with Minikube:

1. Install Docker which will used as driver
 - a. For reference on How: <https://docs.docker.com/engine/install/ubuntu/>
2. Install Kubectl
 - a. Reference: <https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/>
3. Install Minikube
 - a. There two ways I used 2nd as 1st gave some problems
 - i. Using this link : <https://minikube.sigs.k8s.io/docs/start/>
 - ii. USing these steps instead:
 1. wget <https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64>
 2. chmod +x minikube-linux-amd64
 3. sudo mv minikube-linux-amd64 /usr/local/bin/minikube



```
vaibhav@knoldus:~$ minikube start --cpus 6 --memory 8192
minikube v1.26.0 on Ubuntu 22.04
Automatically selected the docker driver. Other choices: ssh, none
Using Docker driver with root privileges
Starting control plane node minikube in cluster minikube
Pulling base image ...
Creating docker container (CPUs=6, Memory=8192MB) ...
Preparing Kubernetes v1.24.1 on Docker 20.10.17 ...
  ■ Generating certificates and keys ...
  ■ Booting up control plane ...
  ■ Configuring RBAC rules ...
Verifying Kubernetes components...
  ■ Using image gcr.io/k8s-minikube/storage-provisioner:v5
Enabled addons: storage-provisioner, default-storageclass
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

4. Install Istio
 - a. For references : <https://istio.io/latest/docs/setup/getting-started/#download>
 - b. Run minikube cluster, I ran with defined cpus and memory
 - i. minikube start --cpus 6 --memory 8192

Note: To run properly, Istio requires 4 vCPUs and 8GB of RAM to work in minikube.

- c. Install Istio in the cluster created
 - i. istioctl install

```
vaibhav@knoldus:~$ curl -L https://istio.io/downloadIstio | sh -
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 1011e 100t  V101 Inse0  Form0t  To288 Extens0ns 0:00:01 0:00:01 0:00:01 288
100 4926 100 4926 0 0 3683 0 0:00:01 0:00:01 0:00:01 0

Download istio-1.14.2 from https://github.com/istio/istio/releases/download/1.14.2/istio-1.14.2-linux-amd64.tar.gz ...
a. Reference: https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/

3. Install Minikube
a. There two ways I used 2nd as 1st gave some problems
i. Using this link : https://minikube.sigs.k8s.io/docs/start/
ii. Using these steps instead:
1. wget
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
To configure the istioctl client tool for your workstation,
add the /home/vaibhav/istio-1.14.2/bin directory to your environment path variable with:
export PATH="$PATH:/home/vaibhav/istio-1.14.2/bin"

Begin the Istio pre-installation check by running:
istioctl x precheck

Need more information? Visit https://istio.io/latest/docs/setup/install/
vaibhav@knoldus:~$ cd istio-1.14.2
vaibhav@knoldus:~/istio-1.14.2$ export PATH=$PWD/bin:$PATH
vaibhav@knoldus:~/istio-1.14.2$ cd
vaibhav@knoldus:~$ istioctl install
This will install the Istio 1.14.2 default profile with ["Istio core" "Istiod" "Ingress gateways"] components into the cluster. Proceed? (y/N) y
3. sudo mv minikube-linux-amd64 /usr/local/bin/minikube
4. Install Istio
a. For references : https://istio.io/latest/docs/setup/getting-started/#download
b. Run minikube cluster, I ran with defined cpus and memory
i. minikube start --cpus 6 --memory 8192
i. istioctl install
Making this installation the default for injection and validation.
Thank you for installing Istio 1.14. Please take a few minutes to tell us about your install/upgrade experience! https://forms.gle/yEtCbt45FZ3VoDT5A
```

Installs Istio gateway , istiod in istio-system namespace

NOTE: You need to provide PATH every time to use this istio configuration for binary file of Istio in form

export PATH=CurrentDirectoryPath/bin:\$PATH

Or

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

Apply Istio Sidecars Implementation

Here we are using this kubernetes yaml to create a cluster and test

- <https://raw.githubusercontent.com/istio/istio/release-1.14/samples/bookinfo.yaml>

```

vaibhav@knoldus:~/Documents/Devops1$ kubectl apply -f 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

```

1. When we first apply the manifest there are only 1 container without any side container

```

vaibhav@knoldus:~/Documents/Devops1$ kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
details-v1-b48c969c5-m6997         1/1     Running   0           45s
productpage-v1-74fdfbd7c7-qggwl    1/1     Running   0           40s
ratings-v1-b74b895c5-vhts5         1/1     Running   0           44s
reviews-v1-68b4dcdb9-n98jz         1/1     Running   0           44s
reviews-v2-565bcd7987-j9q9f        1/1     Running   0           43s
reviews-v3-d88774f9c-j22mr         1/1     Running   0           43s

```

2. We need to enable this side car proxies for that
 - a. We use
 - i. Check current namespaces

```

vaibhav@knoldus:~/Documents/Devops1$ kubectl get ns default --show-labels
NAME      STATUS   AGE   LABELS
default   Active   12h   kubernetes.io/metadata.name=default

```

- ii. Add namespace required
 1. kubectl label namespace default istio-injection=enabled
 2. Crosscheck it

```

Thunderbird Mail
vaibhav@knoldus:~/Documents/Devops1$ kubectl label namespace default istio-injection=enabled
namespace/default labeled
vaibhav@knoldus:~/Documents/Devops1$ kubectl get ns default --show-labels
NAME      STATUS   AGE   LABELS
default   Active   12h   istio-injection=enabled,kubernetes.io/metadata.name=default

```

- iii. Now delete the pods deployments and again deploy

```

vaibhav@knoldus: ~/Documents/Devops1$ kubectl delete -f bookinfo.yaml
service "details" deleted
serviceaccount "bookinfo-details" deleted
deployment.apps "details-v1" deleted
service "ratings" deleted
serviceaccount "bookinfo-ratings" deleted
deployment.apps "ratings-v1" deleted
service "reviews" deleted
serviceaccount "bookinfo-reviews" deleted
deployment.apps "reviews-v1" deleted
deployment.apps "reviews-v2" deleted
deployment.apps "reviews-v3" deleted
service "productpage" deleted
serviceaccount "bookinfo-productpage" deleted
deployment.apps "productpage-v1" deleted
vaibhav@knoldus: ~/Documents/Devops1$ kubectl apply -f 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

```

iv. Check the pods there will be two containers in each pod

```

vaibhav@knoldus: ~/Documents/Devops1$ kubectl get po

```

NAME	READY	STATUS	RESTARTS	AGE
details-v1-b48c969c5-8m2d4	2/2	Running	0	2m24s
productpage-v1-74fdfbd7c7-7p6vl	2/2	Running	0	2m10s
ratings-v1-b74b895c5-zrkdb	2/2	Running	0	2m22s
reviews-v1-68b4dcdb9-kdjfx	2/2	Running	0	2m18s
reviews-v2-565bcd7987-8rxwj	2/2	Running	0	2m17s
reviews-v3-d88774f9c-6rkpd	2/2	Running	0	2m16s

Automatically creates istio init container in the pods

Adding Ingress Gateway to cluster to access the application

1. `kubectl apply -f`
<https://raw.githubusercontent.com/istio/istio/release-1.14/samples/bookinfo/networking/bookinfo-gateway.yaml>
2. Expose the ports from it
 - a. `minikube tunnel` (for loadbalancer)
 - b. `export INGRESS_HOST=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}')`
 - c. `export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].port}')`
 - d. `export SECURE_INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].port}')`
 - e. `export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT`
 - f. `echo "http://$GATEWAY_URL/productpage"`

Adding Monitoring to Kubectl

2. Run following to add kiali, prometheus, grafana, jaeger, zipkin; all to your cluster
 - a. `kubectl apply -f`
<https://raw.githubusercontent.com/istio/istio/release-1.14/samples/addons>
3. Now see that they are create in istio-system namespace

```
vaibhav@knoldus:~$ kubectl get po -n istio-system
```

NAME	READY	STATUS	RESTARTS	AGE
grafana-78588947bf-vwxl6	0/1	Running	0	2m51s
istio-ingressgateway-5f86977657-8l27w	1/1	Running	0	5h40m
istiod-7587989b4f-599lx	1/1	Running	0	5h41m
jaeger-b5874fcc6-bvn7s	1/1	Running	0	2m50s
kiali-575cc8cbf-mk5s5	1/1	Running	0	2m41s
prometheus-6544454f65-qtftz	0/2	ContainerCreating	0	2m37s

```
vaibhav@knoldus:~$ kubectl get svc -n istio-system
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
grafana	ClusterIP	10.97.115.141	<none>	3000/TCP	12m
istio-ingressgateway	LoadBalancer	10.105.142.123	<pending>	15021:31838/TCP,80:31251/TCP,443:30901/TCP	5h50m
istiod	ClusterIP	10.103.61.173	<none>	15010/TCP,15012/TCP,443/TCP,15014/TCP	5h50m
jaeger-collector	ClusterIP	10.102.190.37	<none>	14268/TCP,14250/TCP,9411/TCP	12m
kiali	ClusterIP	10.103.37.150	<none>	20001/TCP,9090/TCP	12m
prometheus	ClusterIP	10.104.81.151	<none>	9090/TCP	12m
tracing	ClusterIP	10.96.148.85	<none>	80/TCP,16685/TCP	12m
zipkin	ClusterIP	10.101.19.8	<none>	9411/TCP	12m

Accessing Kiali

```
vaibhav@knoldus:~$ kubectl port-forward svc/kiali -n istio-system 20001
Forwarding from 127.0.0.1:20001 -> 20001
Forwarding from [::1]:20001 -> 20001
```


Note: for the istio system to work the label “app: something” is mandatory kiali picks it from it

```
vaibhav@knoldus:~$ istioctl dashboard kiali
Unable to listen on port 20001: Listeners failed to create with the following errors: [unable to create listener: Error listen tcp4 127.0.0.1:20001: bind: address already in use unable to create listener: Error listen tcp6 [::1]:20001: bind: address already in use]
http://localhost:40307/kiali
```

Same way other monitoring services can be accessed but the prometheus and grafana would not provide any useful output as they are not configured with the cluster for data exporter.

References:

<https://pre-v1-41.kiali.io/documentation/v1.35/features/>

Command to generate traffic on bookinfo

```
for i in $(seq 1 100); do curl -s -o /dev/null "http://$GATEWAY_URL/productpage"; done
```