Contents

[Service Mesh Features – 2](#_Toc163049498)

[Service Mesh is Responsible For – 2](#_Toc163049499)

[Above responsibility can also be handled by application librarie 3](#_Toc163049500)

[Pre-requisite 4](#_Toc163049501)

[Introduction to Istio Service Mesh 4](#_Toc163049502)

[**High Level Istio Request Flow** 5](#_Toc163049503)

[Verify if Istio Installation is Compatible with Kubernetes Cluster 6](#_Toc163049504)

[Istio Profile 6](#_Toc163049505)

[Requirement 7](#_Toc163049506)

[High Level Architecture 8](#_Toc163049507)

[Demo Example 9](#_Toc163049508)

[How Istio Manage Traffic 11](#_Toc163049509)

[Traffic Management Resources 11](#_Toc163049510)

[Virtual Service 11](#_Toc163049511)

[Destination Rule 11](#_Toc163049512)

[Gateways 11](#_Toc163049513)

[Istio Installation 12](#_Toc163049514)

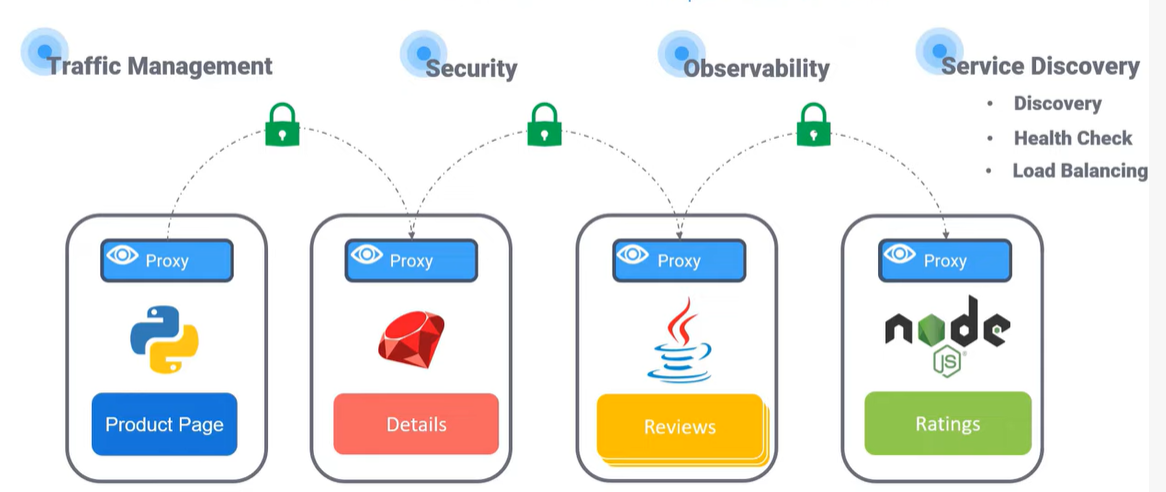
[ Install with Istioctl 12](#_Toc163049515)

[Install with Operator 13](#_Toc163049516)

# Service Mesh Features –

A Service Mesh is a dedicated and configurable infrastructure layer that handles the communication between services without having to change the code in a microservice.

# Service Mesh is Responsible For –



* Traffic Management
  + Discovery
  + Routing
  + Load balancing
  + Failure handling
* Security
  + Encryption
  + Authentication
  + Authorization
* Observability
  + Visualization
  + Tracing
  + Monitoring
* Extensibility

Above responsibility can also be handled by application librarie**s –**

* Stubby from Google
* Finagle from Twitter
* NetflixOSS from Netflix
  + Hystrix
  + Ribbon
  + Eureka
  + Zuul

**Drawback of above libraries -**

* These libraries are targeted for Java Runtimes and can be only used in java project. And it’s a constraint for adopting microservice architecture. Because MSA should support polyglot programming language. These libraries are not good choice for services developed in different programming languages.
* The application code is tightly coupled with above libraries with dependencies.
* The services size gets bigger.
* Developer distracted from original business logic.

### Some Playground Links-

<https://github.com/DevOpsPlayground/Hands-on-with-Istio>

<https://www.knowledgehut.com/blog/cloud-computing/test-drive-your-first-istio-deployment-using-play-with-kubernetes-platform-cloud-computing>

### Some Examples and Concepts-

<https://www.kubecost.com/kubernetes-devops-tools/istio-envoy/>

<https://layer5.io/resources/service-mesh/service-mesh-istio>

# Pre-requisite

Kubernetes Cluster

Basic Kubernetes Knowledge

Side Car Pattern

## Pod Requirements –

* Pod should always run behind one or more services.
* Pod should not run with a security context with user-id 1337
* Pods should run with NET\_ADMIN and NET\_RAW capabilities
* Pods/Deployments should labels with “app” and “version”.

## Introduction to Istio Service Mesh

Istio was originally built to run on Kubernetes, but was written from the perspective of being

deployment-platform agnostic. This means you can leverage an Istio-based service mesh across

deployment platforms like Kubernetes, OpenShift, Mesos, and Cloud Foundry and even

traditional deployment environments like VMs.

With a service proxy next to each application instance, applications no longer need to have

language-specific resilience libraries for circuit breaking, timeouts, retries, service discovery,

load balancing, et. al. Moreover, the service proxy also handles metric collection, distributed

tracing, and log collection.

Since the traffic in the service mesh is flowing through the Istio service proxy, Istio has control

points at each application to influence and direct its networking behavior. This allows a service

operator to control traffic flow and implement fine-grained releases with canary releases, dark

launches, graduated roll outs, and A/B style testing. We’ll explore these capabilities in later

chapters.

### **High Level Istio Request Flow**

A diagram of a product

Description automatically generated with medium confidence

1. Traffic comes into the cluster (ie, a client issues a POST "/checkout" request to our

shopping cart service)

1. Traffic goes to the shopping-cart service but goes to the Istio service proxy (Envoy).

Note, this traffic (and all inter-service communication in the mesh) is secured with

mutual TLS by default. The certificates for establishing mTLS are provided by (7).

1. Istio determines (by looking at the request headers) that this request was initiated by a

customer in the North America region, and for those customers, we want to route some of

those requests to v1.1 of the Tax service which has a fix for certain tax calculations; Istio

routes the traffic to the v1.1 Tax service

1. Istio Pilot is used to configure the istio proxies which handle routing, security, and

resilience

1. Request metrics are periodically sent back to the Istio Mixer which stores them to back

end adapters (to be discussed later)

1. Distributed tracing spans (like Jaeger or Zipkin) are sent back to an tracing store which

can be used to later track the path and latency of a request through the system

1. Istio Auth which manages certificates (expiry, rotation, etc) for each of the istio proxies

so mTLS can be enabled transparently

## Access any Kubernetes Service from External to Istio -Mesh

A diagram of a service

Description automatically generated

## Verify if Istio Installation is Compatible with Kubernetes Cluster

*istioctl x precheck*

**$ istioctl x precheck**

Checking the cluster to make sure it is ready for Istio installation...

#1. Kubernetes-api

-----------------------

Can initialize the Kubernetes client.

Can query the Kubernetes API Server.

#2. Kubernetes-version

-----------------------

Istio is compatible with Kubernetes: v1.16.6-beta.0.

#3. Istio-existence

-----------------------

Istio will be installed in the istio-system namespace.

#4. Kubernetes-setup

-----------------------

Can create necessary Kubernetes configurations: Namespace,

ClusterRole,ClusterRoleBinding,CustomResourceDefinition,Role,

ServiceAccount,Service,Deployments,ConfigMap.

#5. SideCar-Injector

-----------------------

This Kubernetes cluster supports automatic sidecar injection.

To enable automatic sidecar injection see

https://istio.io/docs/setup/kubernetes/additional-setup/ \

sidecar-injection/#deploying-an-app

-----------------------

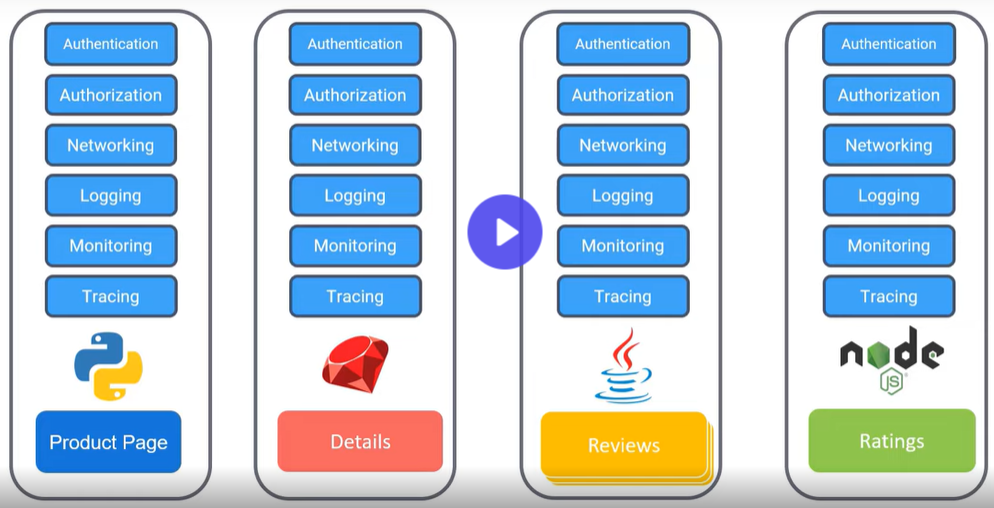
Install Pre-Check passed! The cluster is ready for Istio installation.

## Istio Profile

A grid with check marks

Description automatically generated

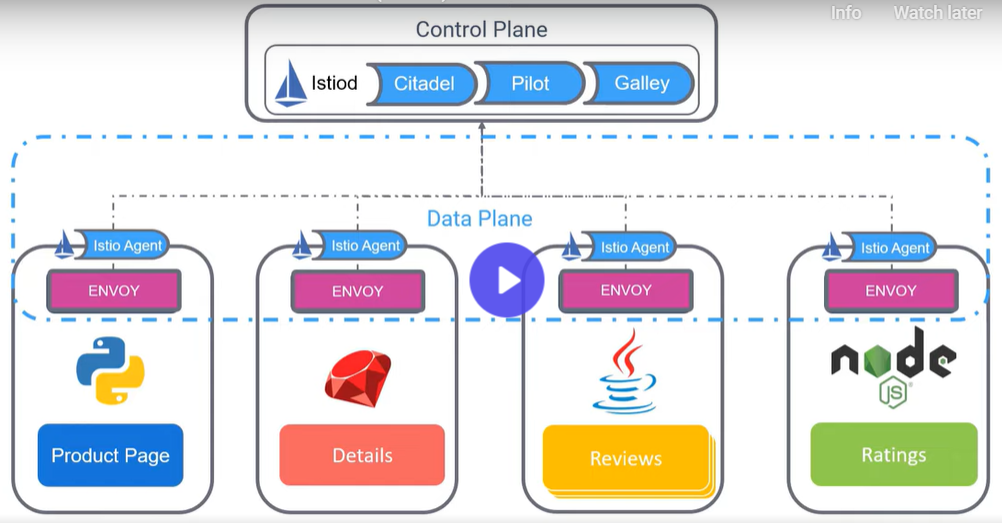
Requirement –



A diagram of a data plane

Description automatically generated

# High Level Architecture



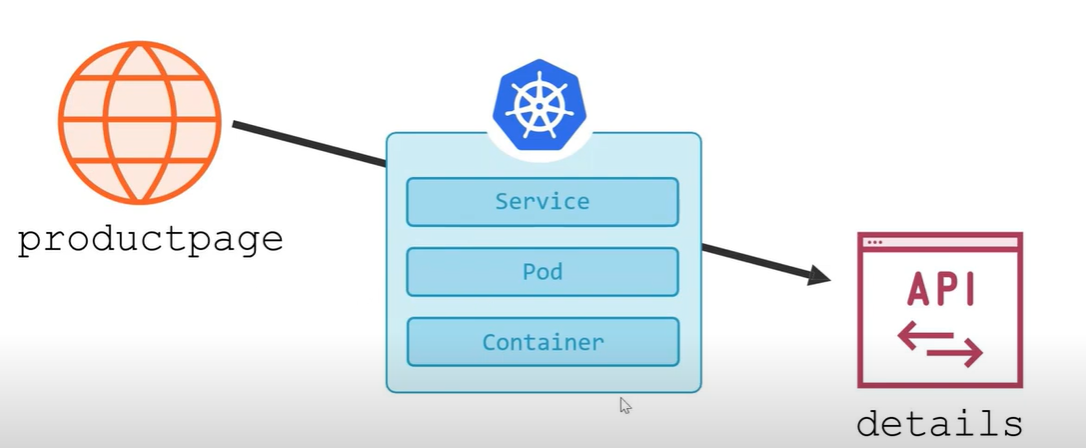
A screenshot of a computer

Description automatically generated

# Demo Example

A diagram of a program

Description automatically generated



A screenshot of a computer

Description automatically generated

# How Istio Manage Traffic

A screenshot of a computer

Description automatically generated

# Traffic Management Resources

Virtual Service – Virtual service, along with Destinations Rules are the key building block of the Istio’s traffic routing functionality.

A Virtual service, lets you configure how requests are routed to a service within Istio service mesh, building on the basic connectivity and discovery provided by Istio and your platform.

Each Virtual Service consists of a set of routing rules that are evaluated in order, letting Istio match each given request to the Virtual Service to a specific real destination within the mesh.

## 

Destination Rule – Destination Rule define policies that apply to traffic intended for a service after routing has occurred. These rules specify configuration for –

* Load Balancing
* Connection pool size from sidecar
* Outlier detection settings to detect
* Evict unhealthy hosts from the load balancing pool.

Gateways – You use gateway to manage inbound and outbound traffic for your mesh, letting you specify which traffic you want to enter or leave the mesh.

Gateway configurations are applied to standalone Envoy Proxies that are running at the edge of the mesh, rather than sidecar Envoy proxies running alongside your service workload.

* Service entries
* Sidecars

# Istio Installation

* Install with Istioctl
* Install with Operator
* Install with Helm

### Install with Istioctl

A screenshot of a computer

Description automatically generated

Verify installation –

A screenshot of a computer screen

Description automatically generated

### Install with Operator

Deprecated approach, because it required privileged access to the Kubernetes cluster, and not good for security purpose.

The Kubernetes Operator pattern is often used as a mechanism for installing software in Kubernetes.

Installation with Operator works as follows –

* Deploy the Operator with *istioctl operator init*
* Apply the IstioOperator resources to the cluster with *kubectl*
* The operator reconciles the state of the cluster to match the specification

A diagram of a product

Description automatically generated

# Upgrading Istio Version

Assuming existing installed istio version is 1.14.4, and we are targeting upgradation to 1.20.0

Existing Installation –

A screenshot of a computer program

Description automatically generated

**Step1** – download target istio release binary

*curl -L https://istio.io/downloadIstio | ISTIO\_VERSION=1.20.4 sh -*

A screenshot of a computer program

Description automatically generated

**Step 2 –** set path as**:**

*export PATH="$PATH:/root/istio-1.20.4/bin"*

**Step 3 –** now install the version of downloaded Istio version 1.20.4

*istioctl install --set profile=demo -y --manifests=/root/istio-1.20.4/manifests*

A screen shot of a computer

Description automatically generated

# Sidecar Injection

Sidecar can be injected by two ways – manual and automatic.

## 1. Manual –

**Step 1** – Generate a pod spec-

*kubectl run my-web-server --image nginx --dry-run=client -o yaml > nginx-pod.yaml*

A screen shot of a computer

Description automatically generated

**Step 2 –** Now manually inject sidecar with kube-inject

*istioctl kube-inject --filename ./nginx-pod.yaml > transformed\_with\_sidecar.yaml*

**Step 3** – Inspect file with istio-proxy with cat command –

A screen shot of a computer

Description automatically generated

**Step 4 –** Now apply the manifest file created in step 3

*kubectl apply -f transformed\_with\_sidecar.yaml*



**Step 5 –** now get the pods, and inspect

A black background with white text

Description automatically generated

## 2. Automatic -

# Service Registry

Istio maintains an internal service registry which can be observed through a debug endpoint /debug/registryz exposed by istiod:

Curl the registry endpoint –

*kubectl exec -n istio-system deploy/istiod -- curl -s localhost:15014/debug/registryz*

A screen shot of a computer screen

Description automatically generated

The output can be prettified, and filtered with a tool such as *jq* .

*kubectl exec -n istio-system deploy/istiod -- curl -s localhost:15014/debug/registryz | jq .[].hostname*

A screen shot of a computer

Description automatically generated

Get confirmed that helloworld service is listed with appropriate namespace.

# Security Authentication Policy