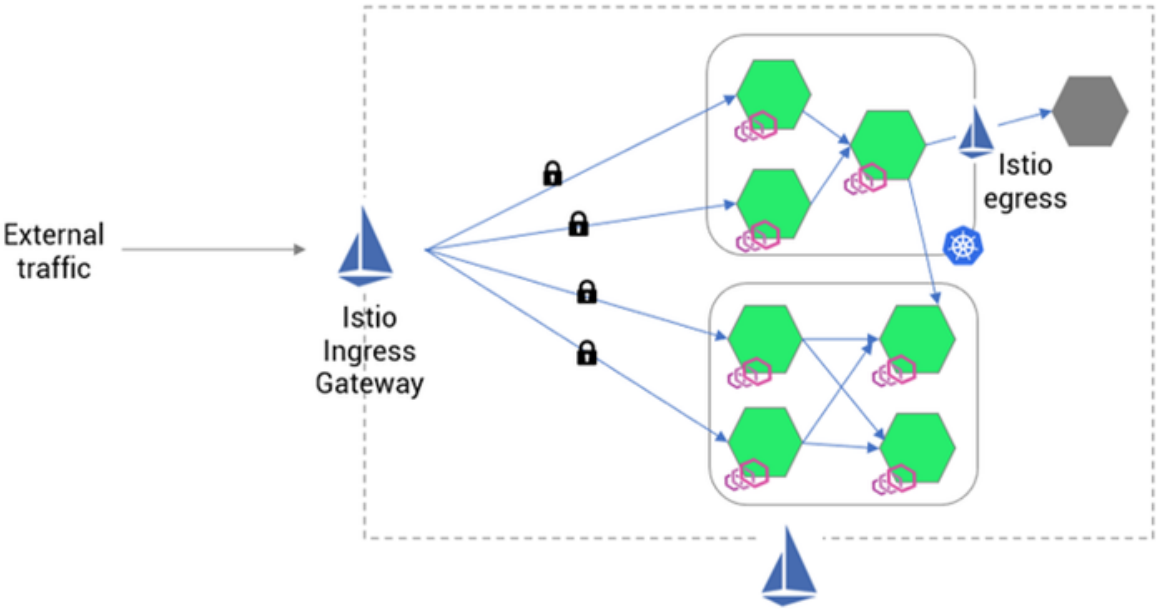
BEST PRACTICE DOCUMENT FOR ADOPTING ISTIO INGRESS GATEWAY

Table of Contents

Introduction to Istio Ingress Gateway

Istio service mesh provides an Istio Ingress gateway (or Istio Ingress) for managing L7 traffic from external clients. In other words, Istio is a controller to implement ingress in your Kubernetes cluster. Istio Ingress can operate at the L4 and L7 layers to manage and secure traffic at the edge in cloud-native applications. It supports various network protocols such as HTTP, gRPC, TCP, etc.



Introduction to Istio Ingress Gateway

Istio Ingress Gateway offers the following features:

1. **Load balancing**
2. **Advanced traffic management includes rate limiting, circuit breaking, failover, etc.**
3. **Flexibility to manipulate HTTP headers for requests and responses**
4. **Out-of-the-box mTLS and access control features**
5. **Extensible policy controls for network and security management**
6. **Offers extensive telemetry and observability options**
7. **Multiple load-balancing techniques and protocols are supported**

Istio Ingress Gateway provides resources to carry specific network and security functionalities. Istio ingress gateway is one of the three resources Istio provides for managing traffic in cloud-native applications; other resources are **Virtual Service** and **Destination Rule**.

Introduction to Istio Ingress Gateway

Istio Ingress Gateway

Istio Ingress Gateway resource receives traffic from external sources. If you want to implement certificates for TLS authentication, Istio gateway resources can be implemented. However, the Istio gateway cannot handle the routing logic to the traffic being handled.

Virtual Service

The Virtual Service resource is configured to route the gateway's traffic to back-end services.

Destination Rule

The Destination Rule defines the variations in deployments in Kubernetes and implements advanced deployment strategies such as canary or blue/green.

Installing Istio Ingress Gateway

The Istio Ingress Gateway can be installed while creating the service mesh. Istio Ingress is an Envoy proxy that will run at the edge and provide the DevOps and application team with fine-grained control over the traffic entering and leaving the mesh.

Note: Istio Ingress Gateway should always be decoupled from the Istio control plane, which will help easily upgrade and maintain. When new updates are available for the Istio gateway, you can just restart the pods, and all the changes will take place automatically.

It is recommended to install the Istio ingress gateway in a different namespace than the Isito control plane to ensure better security posture. In our example, we have installed the Istio gateway in **istio-ingress** namespace.

Installing Istio Ingress Gateway is the same as sidecar injection.

There are 3 ways to install Istio ingress gateways- IstioOperator, Helm, and Kubernetes YAML.

For installing using IstioOperator, you can use the following yaml:

Installing Istio Ingress Gateway

```yaml
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
metadata:
  name: ingress
spec:
  profile: empty # Do not install CRDs or the control plane
  components:
    ingressGateways:
    - name: istio-ingressgateway
      namespace: istio-ingress
      enabled: true
      label:
        # Set a unique label for the gateway. This is required to ensure Gateways
        # can select this workload
        istio: ingressgateway
  values:
    gateways:
      istio-ingressgateway:
        # Enable gateway injection
        injectionTemplate: gateway
```

To install the above yaml, you can use the following command:

```
kubectl create namespace istio-ingress
istioctl install -f ingress.yaml
```

Installing Istio Ingress Gateway

For installing using HELM, you can run the following commands:

```
$ kubectl create namespace istio-ingress
$ helm install istio-ingressgateway istio/gateway -n istio-ingress
```

If you want to use Kubernetes YAML, then here is the below example to create ingress service, Deployment, and Role and Rolebinding resources.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: istio-ingressgateway
  namespace: istio-ingress
spec:
  type: LoadBalancer
  selector:
    istio: ingressgateway
  ports:
  - port: 80
    name: http
  - port: 443
    name: https
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: istio-ingressgateway
  namespace: istio-ingress
spec:
  selector:
    matchLabels:
      istio: ingressgateway
```

Installing Istio Ingress Gateway

```
template:
    metadata:
      annotations:
        # Select the gateway injection template (rather than the default sidecar template)
        inject.istio.io/templates: gateway
      labels:
        # Set a unique label for the gateway. This is required to ensure Gateways can select this
workload
        istio: ingressgateway
        # Enable gateway injection. If connecting to a revisioned control plane, replace with
"istio.io/rev: revision-name"
        sidecar.istio.io/inject: "true"
    spec:
      # Allow binding to all ports (such as 80 and 443)
      securityContext:
        sysctls:
        - name: net.ipv4.ip_unprivileged_port_start
          value: "0"
      containers:
      - name: istio-proxy
        image: auto # The image will automatically update each time the pod starts.
        # Drop all privileges, allowing to run as non-root
        securityContext:
          capabilities:
            drop:
            - ALL
          runAsUser: 1337
          runAsGroup: 1337
```

Installing Istio Ingress Gateway

```
# Set up roles to allow reading credentials for TLS
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: istio-ingressgateway-sds
  namespace: istio-ingress
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: istio-ingressgateway-sds
  namespace: istio-ingress
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: istio-ingressgateway-sds
subjects:
- kind: ServiceAccount
  name: default
```

Creating Gateway resource

Once you install the Istio Ingress Gateway, you can create gateway resources using the Istio Gateway CRD. You need to specify the labels (*istio: ingressgateway*) in the selector to create a gateway resource for Gateway Deployment (ingressgateway).

An example of creating a gateway resource is below. Once you deploy the resource, you will get a public IP address that will listen to HTTP traffic at port 443 and from host- ext-host.example.com.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: ext-host-gwy
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 443
      name: https
      protocol: HTTPS
    hosts:
    - ext-host.example.com
    tls:
      mode: SIMPLE
      credentialName: ext-host-cert
```

Strategies to configure the gateway

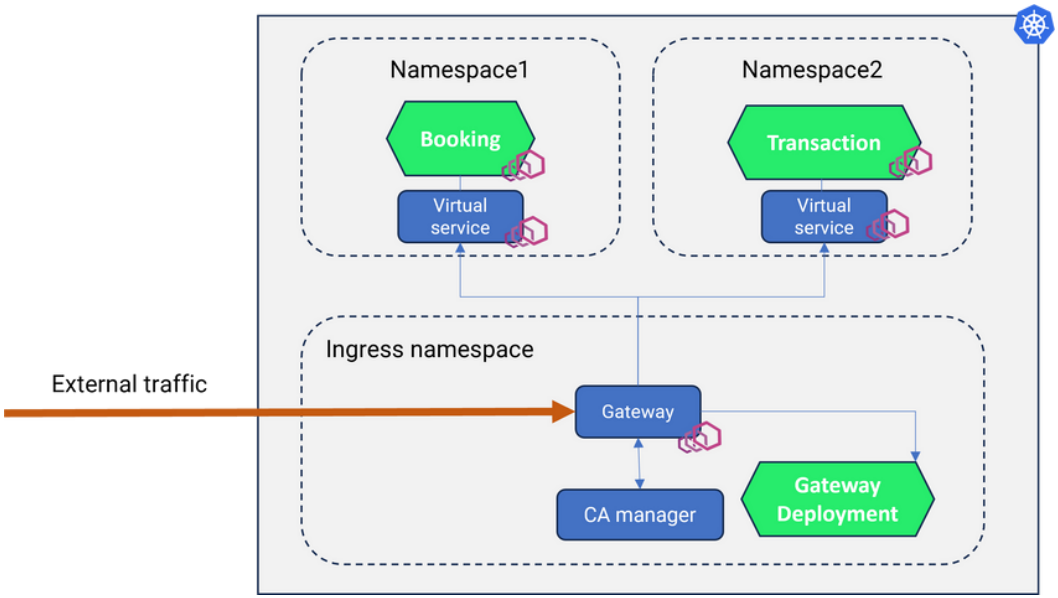
Once Istio Ingress gateway is installed, one can create as many Gateway objects or resources per their enterprise requirements. Below are the various strategies to configure the Istio gateway resource.

1. Shared gateway
2. Dedicated gateway
3. Multicluster gateway

Note: You can mix and match these ways based on your requirements.

Shared gateway and multiple virtual services

A shared gateway resource or object is created and deployed into a dedicated namespace (say ingress) by the DevOps team and is shared by multiple application teams (refer to the below image).



Strategies to configure the gateway

The gateway will be responsible for fetching all the external traffic and routing them to various applications in different namespaces. The app owners will define the routing rules using the virtual service Istio resource. The DevOps team uses the gateway resource to handle certificates and TLS at the front or the edge.

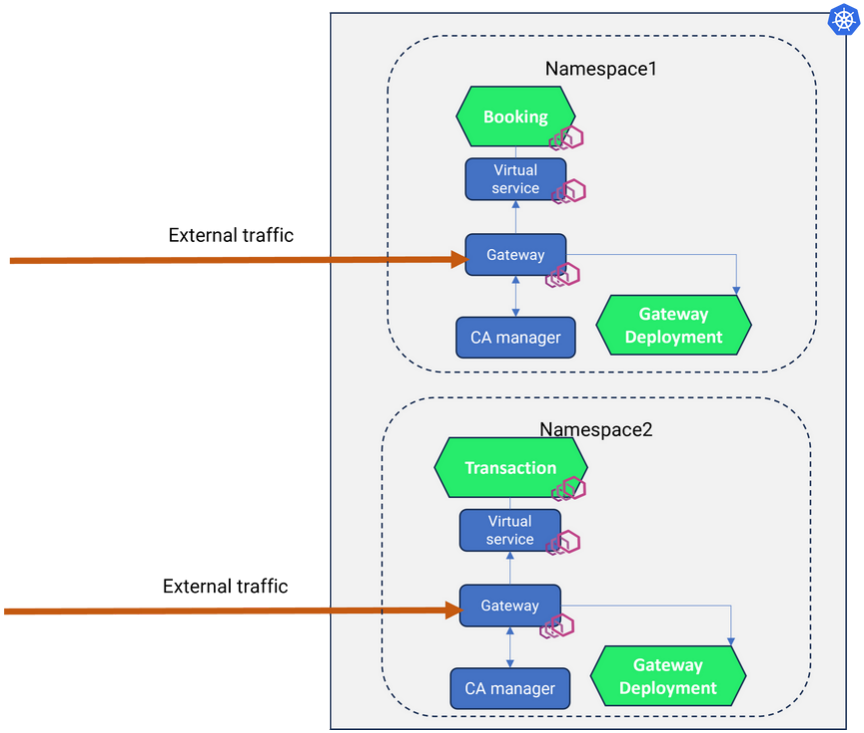
This strategy is considered when isolation is not a concern for cluster or infra admin. A shared gateway strategy is also proper when all the applications share the same TLS certificates, domain, and infra. This model is easy to implement and is less expensive.

The only disadvantage of the shared gateway strategy is that there is no isolation per se, and in case there is only one gateway replica, and if that fails, the traffic to the entire application will cease. It can severely hamper the customer experience. Hence, it is recommended to have at least 3 replicas (pods) of the gateway Deployment.

Strategies to configure the gateway

Dedicated gateway for each application

A Dedicated gateway is a strategy of installing and creating a separate gateway for each application. Suppose you have 10 applications deployed into respective namespaces; then, you must create 10 dedicated gateway resources (refer to the image below).



Each gateway will have its CA manager (like Istio) to authenticate the certificate in the namespaces. The routing rules must be configured in the respective virtual service resource. Each gateway resource will have its IP. This strategy is created to achieve isolation for critical applications and meet stringent security requirements in an enterprise.

Strategies to configure the gateway

Dedicated gateway for each application

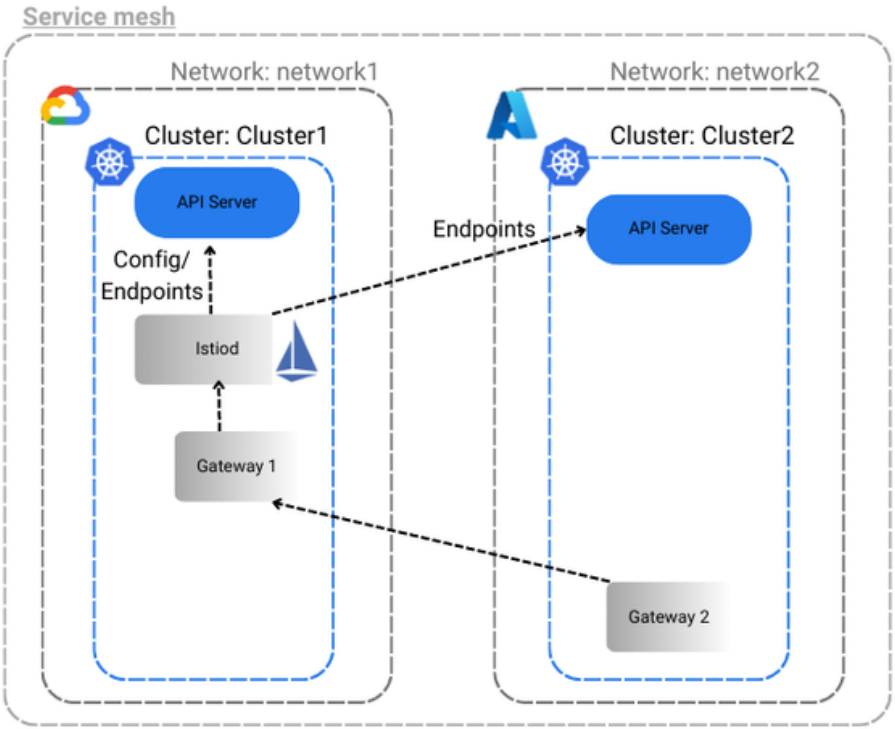
The only downside of this strategy is that it is complicated to set up and maintain. For instance, since there will be many public IPs, configuring DNS and applying network security policies can be messy for the DevOps and architect teams. Hence, enterprises must seek expert help to achieve a Dedicated gateway strategy.

Note: One can use an external load balancer to handle the traffic at the edge and simply the DNS configuration for a Dedicated gateway strategy.

Strategies to configure the gateway

Multicluster gateway for handling traffic into multicloud app

There can be scenarios when multiple microservices of an application are deployed into multiple clusters of various clouds (refer to the image below). In that case, the gateway configuration will involve setting up east-west gateway to send the traffic from one cluster (say GKE) to another (AKS). This strategy is also called a multicluster gateway for handling multicloud traffic.



Questionnaire to test the landscape

There ain't any golden rule for gateway configuration strategy to follow, but a standard practice of understanding the existing landscape will help the DevOps team to apply the best strategy.

Before creating gateway resources, DevOps and architects must ask the following questions.

1. Do we have a large-scale system?
2. Is there any mission-critical system that gets a high traffic volume?
3. Do we have a global developer team developing various applications?
4. Will it require developers to go through a steep learning curve to implement traffic rules into their applications?
5. Does our organization have to follow strict security and compliance measures?

Based on the answers, they can determine various strategies to configure a gateway in your enterprise. Suppose an enterprise is large and has a developer team spread across the globe. In that case, they can use a hybrid approach to configure a dedicated gateway for a few application teams and consider using a shared gateway for the rest.

Updation of Istio Ingress Gateway

Istio Ingress Gateway can be upgraded using in-place or canary methods or HELM. We will focus on the first two methods in this document.

Upgrading with Istio gateway (in-place upgrade):

As the Istio ingress gateway utilizes the pod injection mechanism, the newly created gateway will be injected with the latest configuration. When the pods are restarted using the command *kubectl rollout restart deployment*, the new and updated changes will be picked up in the gateway configuration.

The in-place upgrades are suitable for small teams or lower environments but not suitable for production because there is a possibility of downtime. For example, if the security propagation from the control plane to the gateway (data plane) takes time, your application may not respond to external requests.

Note: The recommended practice is canary deployment for upgrading gateway configuration because the new changes will be released gradually, and there will be no downtime in case of errors.

Upgrading with Istio gateway (canary upgrade)

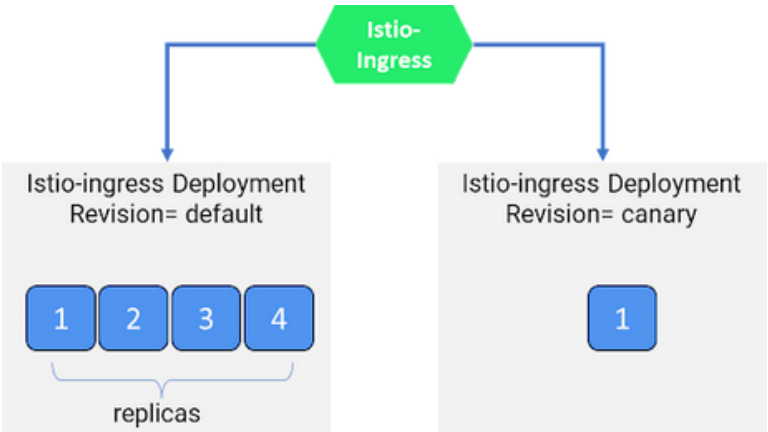
Releasing changes to production gradually in the canary deployment method decreases the chances of failure during deployment. While performing a canary release, you can introduce the new changes to the production in two ways-

1. Canary using Deployments
2. Canary using Services

Updation of Istio Ingress Gateway

Canary using Deployments

Create another Deployment version called canary, under the same service- Istio Ingress, and then adjust the replicas of the default ingress Deployment (older version) and canary Deployment. Since the Ingress service will distribute the traffic among all the replicas (pods) equally, you can create fewer replicas of canary in the initial phase and then gradually increase the replicas (while decreasing the pods of the default version. As you gain confidence with the new version (canary) of the gateway, you can increase the load and gradually phase out the old/default gateway. Refer to the image below.



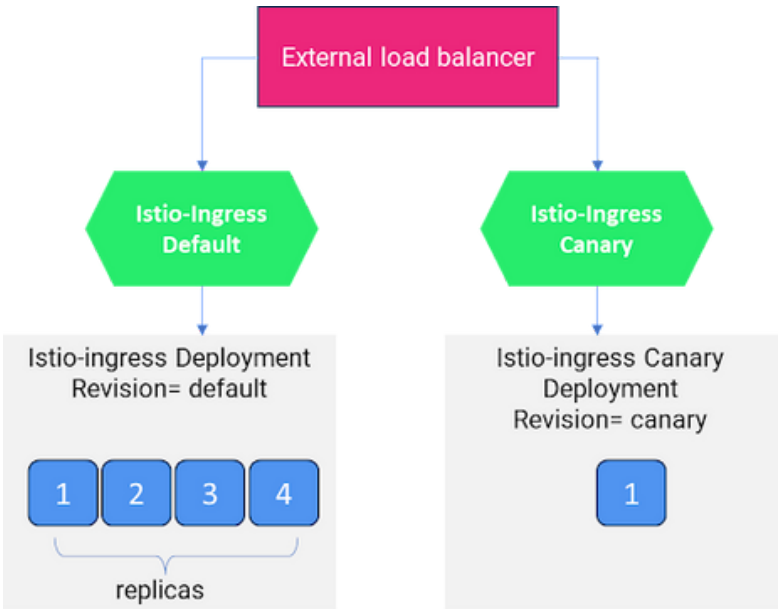
The only problem with this model is when you have limited traffic to your application and a few pods (say 4) in the default Ingress deployment. If you deploy one canary pod, the Istio Ingress service will send 20% of the traffic to the canary pod; there is little flexibility for weight-based traffic splitting.

To overcome the limitation, an external load balancer must be used to achieve fine-grained weight-based traffic distribution for canary release, and we need to create two separate services for Istio ingress.

Updation of Istio Ingress Gateway

Canary using Services

DevOps and Infra team can create a new service and Deployment of the updated Istio Ingress Gateway. Using canary upgrade approach, shift the traffic between the default version and the latest gateway version using a high-level construct outside Istio, such as an external load balancer or DNS. With wight-based traffic splitting, a small percentage (say 5-10%) of the traffic can be sent to the canary version, and performance and behavior can be studied. Based on confidence, the traffic can be increased to 100% with the upgraded Isito Ingress service.



Use cases with Gateway and Virtual Service/Destination Rule

It is essential to know how to use the Istio gateway in tandem with virtual services (VS) and destination rules (DR) and how to distribute the ownership roles and responsibility for creating these resources. Here are a few tips for implementing Istio resources:

1. DevOps or the Infra team should create Gateway resources. They should evaluate the landscape and select a strategy to implement gateway resources in their enterprise.
2. Virtual service and destination rules can be implemented by developers of individual applications irrespective of the gateway implementation strategy.
3. There can be a provision to share a common virtual service (VS) between various (Kubernetes) services for smaller dev teams and fewer microservices. E.g. sharing a VS resource can be good for a small application team, with less than 5 developers working in tandem and 5-10 microservices. Hence, sharing a common VS resource is not recommended for larger teams working on a fleet of microservices.
4. But if the team gradually scales, separate VS resources should be created for individual applications or services for a mid-large scale working team with more than 5 developers or more than 10 microservices. Creating separate VS resources increases the manageability and accountability of traffic rules.

Best practices for naming convention

Best practices for naming various Istio resources are subject to contextualization. A few ways Istio users practice naming conventions are

a. **Gateway name**- [name of gateway]-[namespace where the gateway is installed]. E.g. **gateway1-devns**
b. **Virtual Service**- [Host name or service name]-[“route”]. E.g. **transactions-route**

Destination Rule- [Host name or service name]-[“destination”]. E.g. **transactions-destination**

Caution of Ingress Gateway and Way Forward

1. As IstioOperator and HELM bundle both the service and Deployment, hence the upgradation of the Ingress gateway will not follow canary deployment methods. Hence it is recommended to use Kubernetes YAML for upgrading the Istio ingress gateway in the production system through a canary deployment strategy.
2. Although Istio allows the creation of multiple gateways referring to the same Deployment. However, this should not be practiced or used with caution, otherwise, conflict may arise due to different specifications, such as ports, paths, etc; this may lead to unintended behavior of ingress.
    a. To avoid confusion and complexity, it is better to use Gateway API.