OpenShift Container Platform 4.1

Service Mesh

Service Mesh installation, usage and release notes

Last Updated: 2019-08-28

OpenShift Container Platform 4.1 Service Mesh

Service Mesh installation, usage and release notes

Legal Notice

Abstract

This document provides information on how to use Service Mesh in OCP 4.1

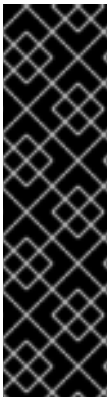
Table of Contents

CHAPTER 1. SERVICE MESH INSTALLATION

1.1. UNDERSTANDING RED HAT SERVICE MESH

Red Hat OpenShift Service Mesh is a platform that provides behavioral insight and operational control over the service mesh. It gives you a uniform way to connect, secure, and monitor microservices in your OpenShift Container Platform environment.

1.1.1. Red Hat OpenShift Service Mesh is a Technology Preview Release



IMPORTANT

This release of Red Hat OpenShift Service Mesh is a Technology Preview release only. Technology Preview releases are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete, and Red Hat does NOT recommend using them for production. Using Red Hat OpenShift Service Mesh on a cluster renders the whole OpenShift cluster as a technology preview, that is, in an unsupported state. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information see Red Hat Technology Preview Features Support Scope.

Related information

- Additional information about support for this technology preview is available in this Red Hat Knowledge Base article.

1.1.2. Understanding service mesh

A *service mesh* is the network of microservices that make up applications in a distributed microservice architecture and the interactions between those microservices. When a service mesh grows in size and complexity, it can become harder to understand and manage.

Based on the open source Istio project, Red Hat OpenShift Service Mesh adds a transparent layer on existing distributed applications without requiring any changes to the service code. You add Red Hat OpenShift Service Mesh support to services by deploying a special sidecar proxy to relevant services in the mesh that intercepts all network communication between microservices. You configure and manage the service mesh using the control plane features.

Red Hat OpenShift Service Mesh gives you an easy way to create a network of deployed services that provide:

- Discovery
- Load balancing
- Service-to-service authentication
- Failure recovery
- Metrics
- Monitoring

Service Mesh also provides more complex operational functions including:

- A/B testing
- Canary releases
- Rate limiting
- Access control
- End-to-end authentication

1.1.3. Red Hat OpenShift Service Mesh Architecture

Red Hat OpenShift Service Mesh is logically split into a data plane and a control plane:

The **data plane** is a set of intelligent proxies deployed as sidecars. These proxies intercept and control all inbound and outbound network communication between microservices in the service mesh. Sidecar proxies also communicate with Mixer, the general-purpose policy and telemetry hub.

- **Envoy proxy** intercepts all inbound and outbound traffic for all services in the service mesh. Envoy is deployed as a sidecar to the relevant service in the same pod.

The **control plane** manages and configures proxies to route traffic, and configures Mixers to enforce policies and collect telemetry.

- **Mixer** enforces access control and usage policies (such as authorization, rate limits, quotas, authentication, request tracing) and collects telemetry data from the Envoy proxy and other services.
- **Pilot** configures the proxies at runtime. Pilot provides service discovery for the Envoy sidecars, traffic management capabilities for intelligent routing (for example, A/B tests or canary deployments), and resiliency (timeouts, retries, and circuit breakers).
- **Citadel** issues and rotates certificates. Citadel provides strong service-to-service and end-user authentication with built-in identity and credential management. You can use Citadel to upgrade unencrypted traffic in the service mesh. Operators can enforce policies based on service identity rather than on network controls using Citadel.
- **Galley** ingests the service mesh configuration, then validates, processes, and distributes the configuration. Galley protects the other service mesh components from obtaining user configuration details from OpenShift Container Platform.

Red Hat OpenShift Service Mesh also uses the **istio-operator** to manage the installation of the control plane. An *Operator* is a piece of software that enables you to implement and automate common activities in your OpenShift cluster. It acts as a controller, allowing you to set or change the desired state of objects in your cluster.

1.1.4. Comparing Red Hat OpenShift Service Mesh and upstream Istio community installations

An installation of Red Hat OpenShift Service Mesh differs from upstream Istio community installations in multiple ways. The modifications to Red Hat OpenShift Service Mesh are sometimes necessary to resolve issues, provide additional features, or to handle differences when deploying on OpenShift.

The current release of Red Hat OpenShift Service Mesh differs from the current upstream Istio community release in the following ways:

1.1.4.1. Multi-tenant installations

NOTE

Single-tenant control plane installations are known to cause issues with OpenShift Container Platform restarts and upgrades. Multi-tenant control plane installations are the default configuration starting with Red Hat OpenShift Service Mesh 0.12.TechPreview.

Red Hat OpenShift Service Mesh allows you to configure multi-tenant control plane installations, specify the namespaces that can access its Service Mesh, and isolate the Service Mesh from other control plane instances.

1.1.4.2. Automatic injection

The upstream Istio community installation automatically injects the sidecar to namespaces you have labeled.

Red Hat OpenShift Service Mesh does not automatically inject the sidecar to any namespaces, but requires you to specify the **sidecar.istio.io/inject** annotation as illustrated in the Automatic sidecar injection section.

1.1.4.3. Role Based Access Control features

Role Based Access Control (RBAC) provides a mechanism you can use to control access to a service. You can identify subjects by username or by specifying a set of properties and apply access controls accordingly.

The upstream Istio community installation includes options to perform exact header matches, match wildcards in headers, or check for a header containing a specific prefix or suffix.

Upstream Istio community matching request headers example

```
apiVersion: "rbac.istio.io/v1alpha1"
kind: ServiceRoleBinding
metadata:
  name: httpbin-client-binding
  namespace: httpbin
spec:
  subjects:
  - user: "cluster.local/ns/istio-system/sa/istio-ingressgateway-service-account"
    properties:
      request.headers[<header>]: "value"
```

Red Hat OpenShift Service Mesh extends the ability to match request headers by using a regular expression. Specify a property key of **request.regex.headers** with a regular expression.

Red Hat OpenShift Service Mesh matching request headers by using regular expressions

```
apiVersion: "rbac.istio.io/v1alpha1"
kind: ServiceRoleBinding
metadata:
```

```
  name: httpbin-client-binding
  namespace: httpbin
spec:
  subjects:
  - user: "cluster.local/ns/istio-system/sa/istio-ingressgateway-service-account"
    properties:
      request.regex.headers[<header>]: "<regular expression>"
```

1.1.4.4. Automatic route creation

WARNING

Automatic route creation is currently incompatible with multi-tenant Service Mesh installations. Ensure that it is disabled in your **ServiceMeshControlPlane** if you plan to attempt a multi-tenant installation.

Red Hat OpenShift Service Mesh automatically manages OpenShift routes for Istio gateways. When an Istio gateway is created, updated, or deleted in the Service Mesh, a matching OpenShift route is created, updated, or deleted.

If the following gateway is created:

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: gateway1
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - www.bookinfo.com
    - bookinfo.example.com
```

The following OpenShift routes are automatically created:

```
$ oc -n istio-system get routes
NAME             HOST/PORT                                  PATH      SERVICES             PORT
TERMINATION   WILDCARD
gateway1-lvlfn   bookinfo.example.com                                  istio-ingressgateway   <all>
None
gateway1-scqhv   www.bookinfo.com                                     istio-ingressgateway   <all>
None
```

If this gateway is deleted, Red Hat OpenShift Service Mesh will delete the routes.

NOTE

Manually created routes are not managed by the Service Mesh.

1.1.4.4.1. Catch-all domains

Red Hat OpenShift Service Mesh does not support catch-all or wildcard domains. If Service Mesh finds a catch-all domain in the gateway definition, Red Hat OpenShift Service Mesh will create the route but relies on OpenShift to create a default hostname . The route that Service Mesh creates will **not** be a catch-all route and will have a hostname with a **<route-name>[-<namespace>].<suffix>** structure.

1.1.4.4.2. Subdomains

Subdomains are supported, but they are not enabled by default in OpenShift. Red Hat OpenShift Service Mesh will create the route with the subdomain, but it will only work after you enable subdomains in OpenShift. See the OpenShift documentation on Wildcard Routes for more information.

1.1.4.4.3. TLS

OpenShift routes are configured to support TLS.

NOTE

All OpenShift routes created by Red Hat OpenShift Service Mesh are in the **istio-system** namespace.

1.1.4.5. OpenSSL

Red Hat OpenShift Service Mesh replaces BoringSSL with OpenSSL. OpenSSL is a software library that contains an open source implementation of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols. The Red Hat OpenShift Service Mesh Proxy binary dynamically links the OpenSSL libraries (libssl and libcrypto) from the underlying Red Hat Enterprise Linux operating system.

1.1.4.6. Container Network Interface (CNI)

Red Hat OpenShift Service Mesh includes CNI which provides you with an alternate way to configure application pod networking. When you enable CNI, it replaces the **init-container** network configuration eliminating the need to grant service accounts and namespaces additional privileges by modifying their Security Context Constraints (SCCs).

Next steps

- Prepare to install Red Hat OpenShift Service Mesh in your OpenShift Container Platform environment.

1.2. PREPARING TO INSTALL RED HAT OPENSHIFT SERVICE MESH

Before you can install Red Hat OpenShift Service Mesh, review the installation activities, ensure that you meet the prerequisites:

Prerequisites

- Possess an active OpenShift Container Platform subscription on your Red Hat account. If you do not have a subscription, contact your sales representative for more information.

- Install OpenShift Container Platform 4.1.
  - For more information about OpenShift Container Platform 4.1, see the OpenShift Container Platform installation overview.
- Install the version of the OpenShift Container Platform command line utility (the **oc** client tool) that matches your OpenShift Container Platform version and add it to your path.
  - If you are using OpenShift Container Platform 4.1, see About the CLI.

1.2.1. Red Hat OpenShift Service Mesh installation activities

The Red Hat OpenShift Service Mesh installation process creates two different projects (namespaces):

- istio-operator project (1 pod)
- istio-system project (17 pods)
- kiali-operator project (1 pod)
- observability project (1 pod)

You first create a Kubernetes *Operator*. This Operator defines and monitors a *custom resource* that manages the deployment, updating, and deletion of the Service Mesh components.

Depending on how you define the custom resource file, you can install one or more of the following components when you install the Service Mesh:

- **Istio** - based on the open source Istio project, lets you connect, secure, control, and observe the microservices that make up your applications.
- **Jaeger** - based on the open source Jaeger project, lets you perform tracing to monitor and troubleshoot transactions in complex distributed systems.
- **Kiali** - based on the open source Kiali project, Kiali provides observability for your service mesh. Using Kiali lets you view configurations, monitor traffic, and view and analyze traces in a single console.

1.2.2. Red Hat OpenShift Service Mesh supported configurations

The following are the only supported configurations for the Red Hat OpenShift Service Mesh 0.12.TechPreview:

- Red Hat OpenShift Container Platform version 4.1.

NOTE

OpenShift Online and OpenShift Dedicated are not supported for Red Hat OpenShift Service Mesh 0.12.TechPreview.

- The deployment must be contained to a single OpenShift Container Platform cluster that is not federated.
- This release of Red Hat OpenShift Service Mesh is only available on OpenShift Container Platform x86_64.

- Red Hat OpenShift Service Mesh is only suited for OpenShift Container Platform Software Defined Networking (SDN) configured as a flat network with no external providers.
- This release only supports configurations where all Service Mesh components are contained in the OpenShift cluster in which it operates. It does not support management of microservices that reside outside of the cluster, or in a multi-cluster scenario.
- The Kiali observability console is only supported on the two most recent releases of the Chrome, Edge, Firefox, or Safari browsers.

1.2.3. Updating the node configuration

Before you can install the Service Mesh into an OpenShift Container Platform installation, you must modify the master configuration and each of the schedulable nodes. These changes enable the features that are required in the Service Mesh and also ensure that Elasticsearch features function correctly.

NOTE

Updating the node configuration is not necessary if you are running OpenShift Container Platform 4.1.

PROCEDURE

To run the Elastisearch application, you must repeat the steps in this procedure for each node in your OpenShift Container Platform installation.

1. Create a file named **/etc/sysctl.d/99-elasticsearch.conf** with the following contents:

```
vm.max_map_count = 262144
```

2. Execute the following command:

```
$ sysctl vm.max_map_count=262144
```

Next steps

- Install Red Hat OpenShift Service Mesh in your OpenShift Container Platform environment.

1.3. INSTALLING RED HAT OPENSHIFT SERVICE MESH

Installing the Service Mesh involves installing the Operator, and then creating and managing a custom resource definition file to deploy the control plane.

NOTE

Starting with Red Hat OpenShift Service Mesh 0.9.TechPreview, Mixer's policy enforcement is disabled by default. You must enable it to run policy tasks. See Update Mixer policy enforcement for instructions on enabling Mixer policy enforcement.

NOTE

Single-tenant control plane installations are known to cause issues with OpenShift Container Platform restarts and upgrades. Multi-tenant control plane installations are the default configuration starting with Red Hat OpenShift Service Mesh 0.12.TechPreview.

Prerequisites

- Follow the Preparing to install Red Hat OpenShift Service Mesh process.
- An account with cluster administration access.

1.3.1. Installing the Operators

The Service Mesh installation process introduces an Operator to manage the installation of the control plane within the **istio-operator** namespace. This Operator defines and monitors a custom resource related to the deployment, update, and deletion of the control plane.

You must install the Jaeger Operator and the Kiali Operator before the Service Mesh Operator can install the control plane.

1.3.1.1. Installing the Jaeger Operator

You must install the Jaeger Operator for the Service Mesh Operator to install the control plane.

Prerequisites

- An account with cluster administrator access.

Procedure

1. Log into your OpenShift Container Platform installation as a cluster administrator.
2. Run the following commands to install the Jaeger Operator:

```
$ oc new-project observability # create the project for the jaeger operator
$ oc create -n observability -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/v1.13.1/deploy/crds/jaegertracing_v1_jaeger_crd.yaml
$ oc create -n observability -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/v1.13.1/deploy/service_account.yaml
$ oc create -n observability -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/v1.13.1/deploy/role.yaml
$ oc create -n observability -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/v1.13.1/deploy/role_binding.yaml
$ oc create -n observability -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/v1.13.1/deploy/operator.yaml
```

1.3.1.2. Installing the Kiali Operator

You must install the Kiali Operator for the Service Mesh Operator to install the control plane.

Prerequisites

- Review the Kiali documentation.

- An account with cluster administrator access.

Procedure

1. Log into your OpenShift Container Platform installation as a cluster administrator.
2. Run the following command to install the Kiali Operator:

```
$ bash <(curl -L https://git.io/getLatestKialiOperator) --operator-image-version v1.0.0 --
operator-watch-namespace '**' --accessible-namespaces '**' --operator-install-kiali false
```

1.3.1.3. Installing the Service Mesh Operator

Prerequisites

- Review the Operator templates on GitHub.
- An account with cluster administrator access.
- The Jaeger Operator must be installed.
- The Kiali Operator must be installed.

Procedure

1. Log into your OpenShift Container Platform installation as a cluster administrator.
2. If the **istio-operator** and **istio-system** namespaces do not exist, run these commands to create the namespaces:

```
$ oc new-project istio-operator
$ oc new-project istio-system
```

3. Run this command to install the Service Mesh Operator. You can run them from any host with access to the cluster.

```
$ oc apply -n istio-operator -f https://raw.githubusercontent.com/Maistra/istio-
operator/maistra-0.12/deploy/servicemesh-operator.yaml
```

1.3.1.4. Verifying the Operator installation

Before proceeding with the install process, verify that the Operator is installed correctly.

Prerequisites

- An account with cluster administrator access.

Procedure

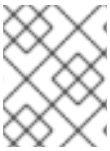
1. Log into your OpenShift Container Platform installation as a cluster administrator.
2. Run this command to verify that the Operator is installed correctly.

```
$ oc get pods -n istio-operator
```

3. When the Operator reaches a running state, it is installed correctly.

```
NAME                              READY   STATUS    RESTARTS   AGE
istio-operator-5cd6bcf645-fvb57   1/1     Running   0          1h
```

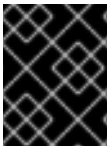
1.3.2. Red Hat OpenShift Service Mesh custom resource



NOTE

The **istio-system** namespace is used an example throughout the Service Mesh documentation, but you can use other namespaces as necessary.

To deploy the Service Mesh control plane, you must create a custom resource. A *custom resource* allows you to introduce your own API into a Kubernetes project or cluster. You create a custom resource yaml file that defines the project parameters and creates the object. This example custom resource yaml file contains all of the supported parameters and deploys Red Hat OpenShift Service Mesh 0.12.TechPreview images based on Red Hat Enterprise Linux (RHEL).



IMPORTANT

The 3scale Istio Adapter is deployed and configured in the custom resource file. It also requires a working 3scale account (SaaS or On-Premises).

Full example istio-installation.yaml

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
metadata:
  name: basic-install

  threeScale:
    enabled: false

  istio:
    global:
      proxy:
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 500m
            memory: 128Mi
      multitenant: true

    gateways:
      istio-egressgateway:
        autoscaleEnabled: false
      istio-ingressgateway:
        autoscaleEnabled: false
        ior_enabled: false
```

```
    mixer:
      policy:
        autoscaleEnabled: false

      telemetry:
        autoscaleEnabled: false
        resources:
          requests:
            cpu: 100m
            memory: 1G
          limits:
            cpu: 500m
            memory: 4G

    pilot:
      autoscaleEnabled: false
      traceSampling: 100.0

    kiali:
     dashboard:
        user: admin
        passphrase: admin
    tracing:
      enabled: true
```

1.3.3. Custom resource parameters

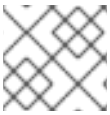
The following examples illustrate use of the supported custom resource parameters for Red Hat OpenShift Service Mesh and the tables provide additional information about supported parameters.

IMPORTANT

The resources you configure for Red Hat OpenShift Service Mesh with these custom resource parameters, including CPUs, memory, and the number of pods, are based on the configuration of your OpenShift cluster. Configure these parameters based on the available resources in your current cluster configuration.

1.3.3.1. Istio global example

Here is an example that illustrates the Istio global parameters for the Red Hat OpenShift Service Mesh custom resource and a description of the available parameters with appropriate values.



NOTE

In order for the 3scale Istio Adapter to work, **disablePolicyChecks** must be **false**.

```
istio:
  global:
    hub: `maistra/` or `registry.redhat.io/openshift-istio-tech-preview/`
    tag: 0.12.0
    proxy:
      resources:
        requests:
```

```
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 500m
        memory: 128Mi
  mtls:
    enabled: false
  disablePolicyChecks: true
  policyCheckFailOpen: false
  imagePullSecrets:
    - MyPullSecret
```

NOTE

See the OpenShift documentation on Scalability and performance for additional details on CPU and memory resources for the containers in your pod.

Table 1.1. Global parameters

Parameter	Description	Values	Default value
disablePolicyChecks	This boolean indicates whether to enable policy checks	**true**/**false**	**true**
policyCheckFailOpen	This boolean indicates whether traffic is allowed to pass through to the Envoy sidecar when the Mixer policy service cannot be reached	**true**/**false**	**false**
tag	The tag that the Operator uses to pull the Istio images	A valid container image tag	**0.12.0**
hub	The hub that the Operator uses to pull Istio images	A valid image repo	**maistra/** or **registry.redhat.io/openshift-istio-tech-preview/**
mtls	This controls whether to enable Mutual Transport Layer Security (mTLS) between services by default	**true**/**false**	**false**
imagePullSecret	If access to the registry providing the Istio images is secure, list an imagePullSecret here	redhat-registry-pullsecret OR quay-pullsecret	None

Table 1.2. Proxy parameters

Type	Parameter	Description	Values	Default value
Resources	**cpu**	The percentage of CPU resources requested for Envoy proxy	CPU resources in millicores based on your environment's configuration	**100m**
	memory	The amount of memory requested for Envoy proxy	Available memory in bytes based on your environment's configuration	**128Mi**
Limits	**cpu**	The maximum percentage of CPU resources requested for Envoy proxy	CPU resources in millicores based on your environment's configuration	**2000m**
	memory	The maximum amount of memory Envoy proxy is permitted to use	Available memory in bytes based on your environment's configuration	**128Mi**

1.3.3.2. Container Network Interface (CNI) example

This example illustrates the CNI parameter and its values for Red Hat OpenShift Service Mesh.

WARNING

If Container Network Interface (CNI) is enabled, manual sidecar injection will work, but pods will not be able to communicate with the control plane unless they are a part of the **ServiceMeshMemberRoll** resource.

CNI example

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
metadata:
 name: basic-install
spec:
```

```
istio:
  istio_cni:
    enabled: true
```

Table 1.3. CNI parameter

Type	Parameter	Description	Values	Default value
istio_cni	**enabled**	This parameter enables the Container Network Interface (CNI).	**true**/**false**	**false**

1.3.3.3. Istio gateway example

Here is an example that illustrates the Istio gateway parameters for the Red Hat OpenShift Service Mesh custom resource and a description of the available parameters with appropriate values.

WARNING

Automatic route creation does not currently work with multi-tenancy. Set **ior_enabled** to **false** for multi-tenant installations.

```
gateways:
    istio-egressgateway:
      autoscaleEnabled: false
      autoscaleMin: 1
      autoscaleMax: 5
    istio-ingressgateway:
      autoscaleEnabled: false
      autoscaleMin: 1
      autoscaleMax: 5
      ior_enabled: false
```

Table 1.4. Istio Gateway parameters

Type	Parameter	Description	Values	Default value
istio-egressgateway	**autoscaleEnabled**	This parameter enables autoscaling.	**true**/**false**	**true**

Type	Parameter	Description	Values	Default value
	autoscaleMin	The minimum number of pods to deploy for the egress gateway based on the autoscaleEnabled setting	A valid number of allocatable pods based on your environment's configuration	**1**
	autoscaleMax	The maximum number of pods to deploy for the egress gateway based on the autoscaleEnabled setting	A valid number of allocatable pods based on your environment's configuration	**5**
istio-ingressgateway	**autoscaleEnabled**	This parameter enables autoscaling.	**true**/**false**	**true**
	autoscaleMin	The minimum number of pods to deploy for the ingress gateway based on the autoscaleEnabled setting	A valid number of allocatable pods based on your environment's configuration	**1**
	autoscaleMax	The maximum number of pods to deploy for the ingress gateway based on the autoscaleEnabled setting	A valid number of allocatable pods based on your environment's configuration	**5**
	ior_enabled	This parameter controls whether Istio routes are automatically configured in OpenShift	**true**/**false**	**true**

1.3.3.4. Istio Mixer example

Here is an example that illustrates the Mixer parameters for the Red Hat OpenShift Service Mesh custom resource and a description of the available parameters with appropriate values.

```
mixer:
  enabled: true
    policy:
      autoscaleEnabled: false

    telemetry:
      autoscaleEnabled: false
      resources:
        requests:
          cpu: 100m
          memory: 1G
        limits:
          cpu: 500m
          memory: 4G
```

Table 1.5. Istio Mixer policy parameters

Parameter	Description	Values	Default value
enabled	This enables Mixer	**true**/**false**	**true**
autoscaleEnabled	This controls whether to enable autoscaling. Disable this for small environments.	**true**/**false**	**true**
autoscaleMin	The minimum number of pods to deploy based on the autoscaleEnabled setting	A valid number of allocatable pods based on your environment's configuration	**1**
autoscaleMax	The maximum number of pods to deploy based on the autoscaleEnabled setting	A valid number of allocatable pods based on your environment's configuration	**5**

Table 1.6. Istio Mixer telemetry parameters

Type	Parameter	Description	Values	Default
Resources	**cpu**	The percentage of CPU resources requested for Mixer telemetry	CPU resources in millicores based on your environment's configuration	**1000m**
	memory	The amount of memory requested for Mixer telemetry	Available memory in bytes based on your environment's configuration	**1G**

Type	Parameter	Description	Values	Default
Limits	**cpu**	The maximum percentage of CPU resources Mixer telemetry is permitted to use	CPU resources in millicores based on your environment's configuration	**4800m**
	memory	The maximum amount of memory Mixer telemetry is permitted to use	Available memory in bytes based on your environment's configuration	**4G**

1.3.3.5. Istio Pilot example

Here is an example that illustrates the Istio Pilot parameters for the Red Hat OpenShift Service Mesh custom resource and a description of the available parameters with appropriate values.

```
pilot:
  resources:
    requests:
      cpu: 100m
  autoscaleEnabled: false
  traceSampling: 100.0
```

Table 1.7. Istio Pilot parameters

Parameter	Description	Values	Default value
cpu	The percentage of CPU resources requested for Pilot	CPU resources in millicores based on your environment's configuration	**500m**
memory	The amount of memory requested for Pilot	Available memory in bytes based on your environment's configuration	**2048Mi**
traceSampling	This value controls how often random sampling occurs. Note: increase for development or testing.	A valid percentage	**1.0**

1.3.3.6. Tracing and Jaeger example

This example illustrates tracing parameters for the Red Hat OpenShift Service Mesh custom resource and a description of the available parameters with appropriate values.

```
tracing:
  enabled: false
  jaeger:
    tag: 1.13.1
    template: all-in-one
    agentStrategy: DaemonSet
```

Table 1.8. Tracing and Jaeger parameters

Parameter	Description	Value	Default value
enabled	This enables tracing in the environment	**true**/**false**	**true**
hub	The hub that the Operator uses to pull Jaeger images	A valid image repo	**jaegertracing/** or **registry.redhat.io/openshift-istio-tech-preview/**
tag	The tag that the Operator uses to pull the Jaeger images	A valid container image tag.	**1.13.1**
template	The deployment template to use for Jaeger	The name of a template type	**all-in-one**/**production-elasticsearch**
agentStrategy	Deploy the Jaeger Agent to each compute node	**DaemonSet** if required	None

1.3.3.7. Kiali example

Here is an example that illustrates the Kiali parameters for the Red Hat OpenShift Service Mesh custom resource and a description of the available parameters with appropriate values.

NOTE

Kiali supports OAuth authentication and hard-coded user credentials. By default, Kiali uses OpenShift OAuth, but you can add a user and passphrase.

```
kiali:
  enabled: true
  hub: kiali/
  tag: v01.0.0
  dashboard:
    user: admin
    passphrase: admin
```

Table 1.9. Kiali parameters

Parameter	Description	Values	Default value
enabled	This enables or disables Kiali in Service Mesh. Kiali is installed by default. If you do not want to install Kiali, change the **enabled** value to **false**.	**true**/**false**	**true**
hub	The hub that the operator uses to pull Kiali images	A valid image repo	**kiali/** or **registry.redhat.io/openshift-istio-tech-preview/**
tag	The tag that the operator uses to pull the Istio images	A valid container image tag	**1.0.0**
user	The username to access the Kiali console. Note: This is not related to any OpenShift account.	A valid Kiali dashboard username	None
passphrase	The password used to access the Kiali console. Note: This is not related to any OpenShift account.	A valid Kiali dashboard passphrase	None

1.3.3.8. 3scale example

Here is an example that illustrates the 3scale Istio Adapter parameters for the Red Hat OpenShift Service Mesh custom resource and a description of the available parameters with appropriate values.

```
threeScale:
  enabled: false
  PARAM_THREESCALE_LISTEN_ADDR: 3333
  PARAM_THREESCALE_LOG_LEVEL: info
  PARAM_THREESCALE_LOG_JSON: true
  PARAM_THREESCALE_LOG_GRPC: false
  PARAM_THREESCALE_REPORT_METRICS: true
  PARAM_THREESCALE_METRICS_PORT: 8080
  PARAM_THREESCALE_CACHE_TTL_SECONDS: 300
  PARAM_THREESCALE_CACHE_REFRESH_SECONDS: 180
  PARAM_THREESCALE_CACHE_ENTRIES_MAX: 1000
  PARAM_THREESCALE_CACHE_REFRESH_RETRIES: 1
  PARAM_THREESCALE_ALLOW_INSECURE_CONN: false
  PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS: 10
  PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS: 60
```

Table 1.10. 3scale parameters

Parameter	Description	Values	Default value
enabled	Whether to use the 3scale adapter	**true**/**false**	**false**
PARAM_THREESCALE_LISTEN_ADDR	Sets the listen address for the gRPC server	Valid port number	**3333**
PARAM_THREESCALE_LOG_LEVEL	Sets the minimum log output level.	**debug**, **info**, **warn**, **error**, or **none**	**info**
PARAM_THREESCALE_LOG_JSON	Controls whether the log is formatted as JSON	**true**/**false**	**true**
PARAM_THREESCALE_REPORT_METRICS	Controls whether 3scale system and backend metrics are collected and reported to Prometheus	**true**/**false**	**true**
PARAM_THREESCALE_METRICS_PORT	Sets the port that the 3scale /**metrics** endpoint can be scrapped from	Valid port number	**8080**
PARAM_THREESCALE_CACHE_TTL_SECONDS	Time period, in seconds, to wait before purging expired items from the cache	Time period in seconds	**300**
PARAM_THREESCALE_CACHE_REFRESH_SECONDS	Time period before expiry when cache elements are attempted to be refreshed	Time period in seconds	**180**
PARAM_THREESCALE_CACHE_ENTRIES_MAX	Max number of items that can be stored in the cache at any time. Set to **0** to disable caching	Valid number	**1000**
PARAM_THREESCALE_CACHE_REFRESH_RETRIES	The number of times unreachable hosts are retried during a cache update loop	Valid number	**1**
PARAM_THREESCALE_ALLOW_INSECURE_CONN	Allow to skip certificate verification when calling **3scale** APIs. Enabling this is not recommended.	**true**/**false**	**false**

Parameter	Description	Values	Default value
PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS	Sets the number of seconds to wait before terminating requests to 3scale System and Backend	Time period in seconds	**10**

1.3.4. Updating Mixer policy enforcement

In previous versions of Red Hat OpenShift Service Mesh, Mixer's policy enforcement was enabled by default. Mixer policy enforcement is now disabled by default. You must enable it before running policy tasks.

Procedure

1. Run this command to check the current Mixer policy enforcement status:

```
$ oc get cm -n istio-system istio -o jsonpath='{.data.mesh}' | grep disablePolicyChecks
```

2. If **disablePolicyChecks: true**, edit the Service Mesh ConfigMap:

```
$ oc edit cm -n istio-system istio
```

3. Locate **disablePolicyChecks: true** within the ConfigMap and change the value to **false**.
4. Save the configuration and exit the editor.
5. Re-check the Mixer policy enforcement status to ensure it is set to **false**.

1.3.5. Deploying the control plane

With the introduction of OpenShift Container Platform 4.1, the network capabilities of the host are now based on nftables rather than iptables. This change impacts the initialization of the Red Hat OpenShift Service Mesh application components. Service Mesh needs to know what host operating system OpenShift is running on to correctly initialize Service Mesh networking components.

NOTE

You do not need to follow this procedure if you are using OpenShift Container Platform 4.1.

If the OpenShift installation is deployed on a Red Hat Enterprise Linux (RHEL) 7 host, then the custom resource must explicitly request the RHEL 7 **proxy-init** container image by including the following:

Enabling the proxy-init container for RHEL 7 hosts

```
apiVersion: maistra.io/v1
 kind: ServiceMeshControlPlane
 spec:
```

```
istio:
  global:
    proxy_init:
      image: proxy-init
```

Use the custom resource definition file you created to deploy the Service Mesh control plane.

Procedure

1. Create a custom resource definition file named **istio-installation.yaml**.
2. Run this command to deploy the control plane:

```
$ oc create -n istio-system -f istio-installation.yaml
```

3. Run this command to watch the progress of the pods during the installation process:

```
$ oc get pods -n istio-system -w
```

1.3.5.1. Verifying the control plane installation

Verify that the control plane installation completed successfully.

NOTE

The name of the resource is **istio-installation**.

Procedure

1. Run this command to determine if the Operator finished deploying the control plane:

```
$ oc get servicemeshcontrolplane/basic-install -n istio-system --template='{{range
.status.conditions}}{{printf "%s=%s, reason=%s, message=%s\n\n" .type .status .reason
.message}}{{end}}'
```

   When the control plane installation is finished, the output is similar to the following:

```
Installed=True, reason=InstallSuccessful, message=%!s()

Reconciled=True, reason=InstallSuccessful, message=%!s()
```

2. After the control plane is deployed, run this command to check the status of the pods:

```
$ oc get pods -n istio-system
```

3. Verify that the pods are in a state similar to this:

NOTE

The results returned when you run this verification step vary depending on your configuration including the number of nodes in the cluster, and whether you are using 3scale, Jaeger, Kiali, or Prometheus.

```
NAME                                     READY   STATUS    RESTARTS   AGE
3scale-istio-adapter-67b96f97b5-cwvgt    1/1     Running   0          99s
grafana-75f4cbbc6-xw99s                  1/1     Running   0          54m
istio-citadel-8489b8bb96-ttqfd           1/1     Running   0          54m
stio-egressgateway-5ccd4d5ddd-wtp2h      1/1     Running   0          52m
istio-galley-58ff8db57c-jrpkz            1/1     Running   0          54m
istio-ingressgateway-698674848f-bk57s    1/1     Running   0          52m
istio-node-2d764                         1/1     Running   0          54m
istio-node-4h926                         1/1     Running   0          54m
istio-node-6qxcj                         1/1     Running   0          54m
istio-node-8fxqz                         1/1     Running   0          54m
istio-node-gzg5v                         1/1     Running   0          54m
istio-node-vxx5p                         1/1     Running   0          54m
istio-pilot-764966cf69-9nlhp             2/2     Running   0          19m
istio-policy-7c856f7d5f-4fjk4            2/2     Running   2          53m
istio-sidecar-injector-757b8ccdbf-znggc  1/1     Running   0          49m
istio-telemetry-65d8b47c98-jrp9h         2/2     Running   2          53m
jaeger-f775b79f8-cmbb2                   2/2     Running   0          54m
kiali-7646d796cd-kfx29                   1/1     Running   0          20m
prometheus-56cb9c859b-dlqmn              2/2     Running   0          54m
```

Next steps

- Prepare to deploy applications on Red Hat OpenShift Service Mesh.

1.4. INSTALLING RED HAT OPENSHIFT SERVICE MESH WITH MULTI-TENANT CONTROL PLANES

The Red Hat OpenShift Service Mesh operator supports multi-tenant control plane installations.

NOTE

- You cannot use multi-tenant control plane installations in conjunction with a cluster-wide control plane installation. Red Hat OpenShift Service Mesh installations must either be multi-tenant or single, cluster-wide installations.
- Single-tenant control plane installations are known to cause issues with OpenShift Container Platform restarts and upgrades. Multi-tenant control plane installations are the default configuration starting with Red Hat OpenShift Service Mesh 0.12.TechPreview.

Prerequisites

- Follow the instructions in Installing Red Hat OpenShift Service Mesh .

1.4.1. Known issues with multi-tenant Red Hat OpenShift Service Mesh installations

Here is a summary of the known issues with multi-tenant Service Mesh installations.

WARNING

Automatic route creation is currently incompatible with multi-tenant Service Mesh installations. Ensure that it is disabled, by setting **ior_enabled** to **false** in your **ServiceMeshControlPlane** if you plan to attempt a multi-tenant installation.

- **MeshPolicy** is still a cluster-scoped resource and applies to all control planes installed in OpenShift. This can prevent the installation of multiple control planes or cause unknown behavior if one control plane is deleted.
- The Jaeger agent runs as a **DaemonSet**, therefore **tracing** may only be enabled for a single **ServiceMeshControlPlane** instance.
- If you delete the project that contains the control plane before you delete the **ServiceMeshControlPlane** resource, some parts of the installation may not be removed:
  - Service accounts added to the **SecurityContextConstraints** may not be removed.
  - **OAuthClient** resources associated with Kiali may not be removed, or its list of redirectURIs may not be accurate.

1.4.2. Differences between multi-tenant and cluster-wide installations

The main difference between a multi-tenant installation and a cluster-wide installation is the scope of privileges used by the control plane deployments, for example, Galley and Pilot. The components no longer use cluster-scoped Role Based Access Control (RBAC) **ClusterRoleBinding**, but rely on namespace-scoped RBAC **RoleBinding**.

Every namespace in the **members** list will have a **RoleBinding** for each service account associated with a control plane deployment and each control plane deployment will only watch those member namespaces. Each member namespace has a **maistra.io/member-of** label added to it, where the **member-of** value is the namespace containing the control plane installation.

1.4.3. Installing Red Hat OpenShift Service Mesh with multi-tenancy

This procedure describes the how to enable multi-tenancy.

Prerequisites

- An installed, verified Service Mesh Operator.
- A custom resource file that defines the parameters of your Red Hat OpenShift Service Mesh control plane.

Procedure

- Set the **multitenant** option to **true** in the istio section of the **ServiceMeshControlPlane** resource. For example:

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
```

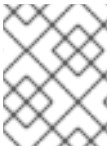
```
spec:
  istio:
    global:
    # enable multitenant
     multitenant: true
     # additional control plane configuration details
```

1.4.4. Configuring namespaces in multi-tenant installations

A multi-tenant control plane installation only affects namespaces configured as part of the Service Mesh. You must specify the namespaces associated with the Service Mesh in a **ServiceMeshMemberRoll** resource located in the same namespace as the **ServiceMeshControlPlane** resource and name it **default**.

WARNING

If Container Network Interface (CNI) is enabled, manual sidecar injection will work, but pods will not be able to communicate with the control plane unless they are a part of the **ServiceMeshMemberRoll** resource.



NOTE

The member namespaces are only updated if the Service Mesh control plane installation succeeds.

- You can add any number of namespaces, but a namespace can only belong to **one ServiceMeshMemberRoll**.
- **ServiceMeshMemberRoll** resources are reconciled in response to the following events:
  - The **ServiceMeshMemberRoll** is created, updated, or deleted
  - The **ServiceMeshControlPlane** resource in the namespace containing the **ServiceMeshMemberRoll** is created or updated
  - A namespace listed in the **ServiceMeshMemberRoll** is created or deleted

The **ServiceMeshMemberRoll** is deleted when its corresponding **ServiceMeshControlPlane** resource is deleted.

Here is an example that joins the bookinfo namespace into the Service Mesh:

Prerequisites

- An installed, verified Service Mesh Operator.
- Location of the namespace where the custom resource installed the control plane.

Procedure

1. Create a custom resource file named **ServiceMeshMemberRoll** in the same namespace as the **ServiceMeshControlPlane** custom resource.
2. Name the resource **default**.
3. Add the namespaces to the member list in the **ServiceMeshMemberRoll**. The **bookinfo** namespace is joined to the Service Mesh in this example.

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
spec:
  members:
  # a list of namespaces joined into the service mesh
  - bookinfo
```

Next steps

- Prepare to deploy applications on Red Hat OpenShift Service Mesh.

1.5. DEPLOYING APPLICATIONS ON RED HAT OPENSHIFT SERVICE MESH

When you deploy an application into the Service Mesh, there are several differences between the behavior of applications in the upstream community version of Istio and the behavior of applications within a Red Hat OpenShift Service Mesh installation.

Prerequisites

- Review Comparing Red Hat OpenShift Service Mesh and upstream Istio community installations
- Review Installing Red Hat OpenShift Service Mesh

1.5.1. Red Hat OpenShift Service Mesh security constraints

NOTE

The relaxing of security constraints is only necessary during the Red Hat OpenShift Service Mesh Technology Preview.

When you deploy an application into a Service Mesh running in an OpenShift environment, it is necessary to relax the security constraints placed on the application by its service account to ensure the application can function correctly. Each service account must be granted permissions with the **anyuid** and **privileged** Security Context Constraints (SCC) to enable the sidecars to run correctly.

The **privileged** SCC is required to ensure changes to the pod's networking configuration is updated successfully with the **istio-init** initialization container and the **anyuid** SCC is required to enable the sidecar container to run with its required user id of **1337**.

To configure the correct permissions it is necessary to identify the service accounts being used by your application's pods. For most applications, this will be the **default** service account, however your Deployment/DeploymentConfig may override this within the pod specification by providing the

serviceAccountName.

For each identified service account you must update the cluster configuration to ensure they are granted access to the **anyuid** and **privileged** SCCs by executing the following commands from an account with cluster admin privileges.

1.5.1.1. Configuring Red Hat OpenShift Service Mesh security constraints

Configure service accounts that require additional permissions to run in the Service Mesh with **anyuid** or **privileged** Security Context Constraints (SCCs).

NOTE

You do not need to follow this procedure if you are using OpenShift Container Platform 4.1.

Prerequisites

- Identify the service accounts that require SCC changes.
- Identify the namespaces associated with the service accounts that require SCC changes.

Procedure

1. Identify the service account(s) that require relaxed privileges.

NOTE

Replace **<service account>** and **<namespace>** with values specific to your application in the commands in this procedure.

2. Run this command for each service account that requires the **anyuid** SCC for its associated sidecar container.

```
$ oc adm policy add-scc-to-user anyuid -z <service account> -n <namespace>
```

3. Run this command for each service account that requires the **privileged** SCC to allow successful updates to its pod's networking configuration:

```
$ oc adm policy add-scc-to-user privileged -z <service account> -n <namespace>
```

1.5.2. Red Hat OpenShift Service Mesh's master configuration

Red Hat OpenShift Service Mesh relies on a proxy sidecar within the application's pod to provide Service Mesh capabilities to the application. You can enable automatic sidecar injection or manage it manually. Red Hat recommends automatic injection using the annotation with no need to label namespaces. This ensures that your application contains the appropriate configuration for the Service Mesh upon deployment. This method requires fewer privileges and does not conflict with other OpenShift capabilities such as builder pods.

NOTE

The upstream version of Istio injects the sidecar by default if you have labeled the namespace. You are not required to label the namespace with Red Hat OpenShift Service Mesh. However, Red Hat OpenShift Service Mesh requires you to opt in to having the sidecar automatically injected to a deployment. This avoids injecting a sidecar where it is not wanted (for example, build or deploy pods). The webhook checks the configuration of pods deploying into all namespaces to see if they are opting in to injection with the appropriate annotation.

1.5.2.1. Updating the master configuration

To enable the automatic injection of the Service Mesh sidecar, you must first modify the master configuration on each master node in your OpenShift Container Platform installation to include support for webhooks and signing of Certificate Signing Requests (CSRs).

NOTE

Updating the master configuration is not necessary if you are running OpenShift Container Platform 4.1.

Procedure

1. Change to the directory containing the master configuration file (for example, **/etc/origin/master/master-config.yaml**).
2. Create a file named **master-config.patch** with the following contents:

```
admissionConfig:
  pluginConfig:
    MutatingAdmissionWebhook:
      configuration:
        apiVersion: apiserver.config.k8s.io/v1alpha1
        kubeConfigFile: /dev/null
        kind: WebhookAdmission
    ValidatingAdmissionWebhook:
      configuration:
        apiVersion: apiserver.config.k8s.io/v1alpha1
        kubeConfigFile: /dev/null
        kind: WebhookAdmission
```

3. In the same directory, issue the following commands to apply the patch to the **master-config.yaml** file:

```
$ cp -p master-config.yaml master-config.yaml.prepatch
$ oc ex config patch master-config.yaml.prepatch -p "$(cat master-config.patch)" > master-config.yaml
$ /usr/local/bin/master-restart api && /usr/local/bin/master-restart controllers
```

1.5.2.1.1. Enabling automatic sidecar injection

When deploying an application into the Red Hat OpenShift Service Mesh you must opt in to injection by specifying the **sidecar.istio.io/inject** annotation with a value of **true**. Opting in ensures that the sidecar injection does not interfere with other OpenShift features such as builder pods used by numerous

frameworks within the OpenShift ecosystem.

Prerequisites

- Identify the deployments for which you want to enable automatic sidecar injection.
- Locate the application's yaml configuration file.

Procedure

1. Open the application's configuration yaml file in an editor.
2. Add **sidecar.istio.io/inject** to the configuration yaml with a value of **true** as illustrated here:

   Sleep test application example

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: sleep
spec:
  replicas: 1
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true"
      labels:
        app: sleep
    spec:
      containers:
      - name: sleep
        image: tutum/curl
        command: ["/bin/sleep","infinity"]
        imagePullPolicy: IfNotPresent
```

3. Save the configuration file.

1.5.2.1.2. Using manual sidecar injection

Manual injection of the sidecar is supported using the upstream Istio community **istioctl** command.

NOTE

When you use manual sidecar injection, ensure you have access to a running cluster so the correct configuration can be obtained from the istio-sidecar-injector configmap within the istio-system namespace.

Prerequisites

- Download the appropriate installation for your OS.

Procedure

1. Unpack the **istioctl** installation into a directory

2. Add the **istioctl** binary to the the bin directory in your PATH.

3. Run this command to inject the sidecar into your application and pipe the configuration to the **oc** command to create deployments:

```
$ istioctl kube-inject -f app.yaml | oc create -f -
```

Next steps

- Deploy applications on Red Hat OpenShift Service Mesh.

1.6. EXAMPLE APPLICATION

1.6.1. Bookinfo application

The upstream Istio project has an example tutorial called bookinfo, which is composed of four separate microservices used to demonstrate various Istio features. The Bookinfo application displays information about a book, similar to a single catalog entry of an online book store. Displayed on the page is a description of the book, book details (ISBN, number of pages and other information), and book reviews.

The Bookinfo application consists of four separate microservices:

- The **productpage** microservice calls the **details** and **reviews** microservices to populate the page.
- The **details** microservice contains book information.
- The **reviews** microservice contains book reviews. It also calls the **ratings** microservice.
- The **ratings** microservice contains book ranking information that accompanies a book review.

There are three versions of the reviews microservice:

- Version v1 does not call the **ratings** Service.
- Version v2 calls the **ratings** Service and displays each rating as one to five black stars.
- Version v3 calls the **ratings** Service and displays each rating as one to five red stars.

1.6.2. Installing the Bookinfo application

The following steps describe deploying and running the Bookinfo tutorial on OpenShift Container Platform with Service Mesh 0.12.TechPreview.

Prerequisites:

- OpenShift Container Platform 3.11 or higher installed.
- Red Hat OpenShift Service Mesh 0.12.TechPreview installed.

NOTE

Red Hat OpenShift Service Mesh implements auto-injection differently than the upstream Istio project, therefore this procedure uses a version of the **bookinfo.yaml** file annotated to enable automatic injection of the Istio sidecar.

Procedure

1. Create a project for the Bookinfo application.

```
$ oc new-project myproject
```

2. Update the Security Context Constraints (SCC) by adding the service account used by Bookinfo to the **anyuid** and **privileged** SCCs in the *"myproject"* Namespace:

```
$ oc adm policy add-scc-to-user anyuid -z default -n myproject
$ oc adm policy add-scc-to-user privileged -z default -n myproject
```

3. Deploy the Bookinfo application in the *"myproject"* Namespace by applying the **bookinfo.yaml** file:

```
$ oc apply -n myproject -f
https://raw.githubusercontent.com/Maistra/bookinfo/master/bookinfo.yaml
```

4. Create the ingress gateway for Bookinfo by applying the **bookinfo-gateway.yaml** file:

```
 $ oc apply -n myproject -f
https://raw.githubusercontent.com/Maistra/bookinfo/master/bookinfo-gateway.yaml
```

5. Set the value for the **GATEWAY_URL** parameter:

```
$ export GATEWAY_URL=$(oc get route -n istio-system istio-ingressgateway -o
jsonpath='{.spec.host}')
```

1.6.3. Verifying the Bookinfo installation

Before configuring your application, verify that it successfully deployed.

Procedure

- To confirm that the application is deployed:
  - Run this command:

```
$ curl -o /dev/null -s -w "%{http_code}\n" http://<GATEWAY_URL>/productpage
```

  - Alternatively, you can open **http://<GATEWAY_URL>/productpage** in your browser.

1.6.4. Adding default destination rules

Before you can use the Bookinfo application, you have to add default destination rules. There are two preconfigured yaml files, depending on whether or not you enabled mutual transport layer security (TLS) authentication.

Procedure

1. To add destination rules, run one of the following commands:
   - If you did not enable mutual TLS:

     ```
     $ oc apply -n myproject -f https://raw.githubusercontent.com/istio/istio/release-1.1/samples/bookinfo/networking/destination-rule-all.yaml
     ```

   - If you enabled mutual TLS:

     ```
     $ oc apply -n myproject -f https://raw.githubusercontent.com/istio/istio/release-1.1/samples/bookinfo/networking/destination-rule-all-mtls.yaml
     ```

2. To list all available destination rules:

   ```
   $ oc get destinationrules -o yaml
   ```

Now that you have a sample application, you can use the other tutorials in this guide to examine your service mesh.

1.6.5. Removing the Bookinfo application

When you finish with the Bookinfo application, you can remove it by running the cleanup script.

TIP

Several of the other Service Mesh tutorials also use the Bookinfo application. Do not run the cleanup script if you plan to explore other Service Mesh tutorials.

Procedure

1. Download the cleanup script:

   ```
   $ curl -o cleanup.sh https://raw.githubusercontent.com/Maistra/bookinfo/master/cleanup.sh && chmod +x ./cleanup.sh
   ```

2. Delete the Bookinfo virtualservice, gateway, and terminate the Pods by running the cleanup script:

   ```
   $ ./cleanup.sh
   namespace ? [default] myproject
   ```

3. Confirm shutdown by running these commands:

   ```
   $ oc get virtualservices -n myproject
   No resources found.
   $ oc get gateway -n myproject
   No resources found.
   $ oc get pods -n myproject
   No resources found.
   ```

1.7. KIALI TUTORIAL

Kiali works with Istio to visualize your service mesh topology to provide visibility into features like circuit breakers, request rates, and more. Kiali offers insights about the mesh components at different levels, from abstract Applications to Services and Workloads. Kiali provides an interactive graph view of your Namespace in real time. It can display the interactions at several levels (applications, versions, workloads) with contextual information and charts on the selected graph node or edge.

This tutorial uses Service Mesh and the **bookinfo** tutorial to demonstrate how you can use the Kiali console to view the topography and health of your service mesh.

1.7.1. Accessing the Kiali console

The Kiali console provides visualization and observability for your Service mesh. The Kiali console has different views that provide insights into Service mesh components at different levels, from Applications to Services to Workloads. It also provides validation for Istio configurations.

Prerequisites

- OpenShift Container Platform 3.11 or higher installed.
- Red Hat OpenShift Service Mesh 0.12.TechPreview installed.
- Kiali parameters specified in the custom resource file.
- **Bookinfo** demonstration application installed.

Procedure

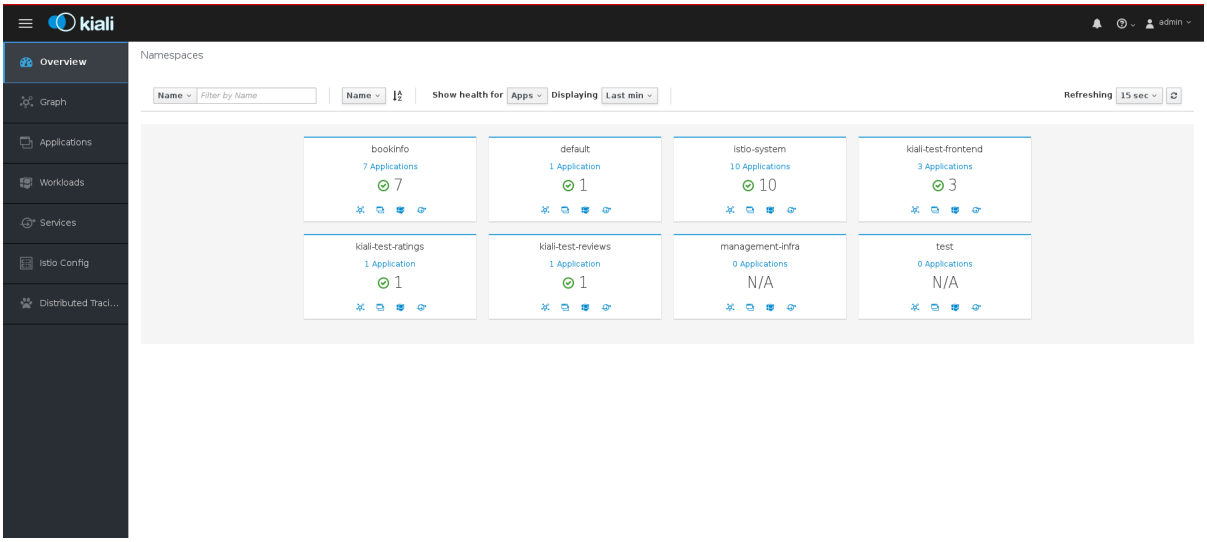
1. A route to access the Kiali console already exists. Run the following command to obtain the route and Kiali URL:

```
$ oc get routes
```

```
NAME                HOST/PORT                                        PATH      SERVICES
PORT            TERMINATION   WILDCARD
grafana             grafana-istio-system.127.0.0.1.nip.io                      grafana             http
None
istio-ingress       istio-ingress-istio-system.127.0.0.1.nip.io                istio-ingress       http
None
istio-ingressgateway   istio-ingressgateway-istio-system.127.0.0.1.nip.io          istio-
ingressgateway   http                    None
jaeger-query        jaeger-query-istio-system.127.0.0.1.nip.io                 jaeger-query
jaeger-query     edge          None
kiali               kiali-istio-system.127.0.0.1.nip.io                        kiali               <all>
None
prometheus          prometheus-istio-system.127.0.0.1.nip.io                   prometheus
http-prometheus              None
tracing             tracing-istio-system.127.0.0.1.nip.io                      tracing             tracing
edge          None
```

2. Launch a browser and navigate to https://<KIALI_URL> (in the output example, this is **kiali-istio-system.127.0.0.1.nip.io**). You should see the Kiali console login screen.

3. Log in to the Kiali console using the user name and password that you specified in the custom resource file during installation.
   When you first log in you see the Overview page, which provides a quick overview of the health of the various Namespaces that are part of your Service Mesh.



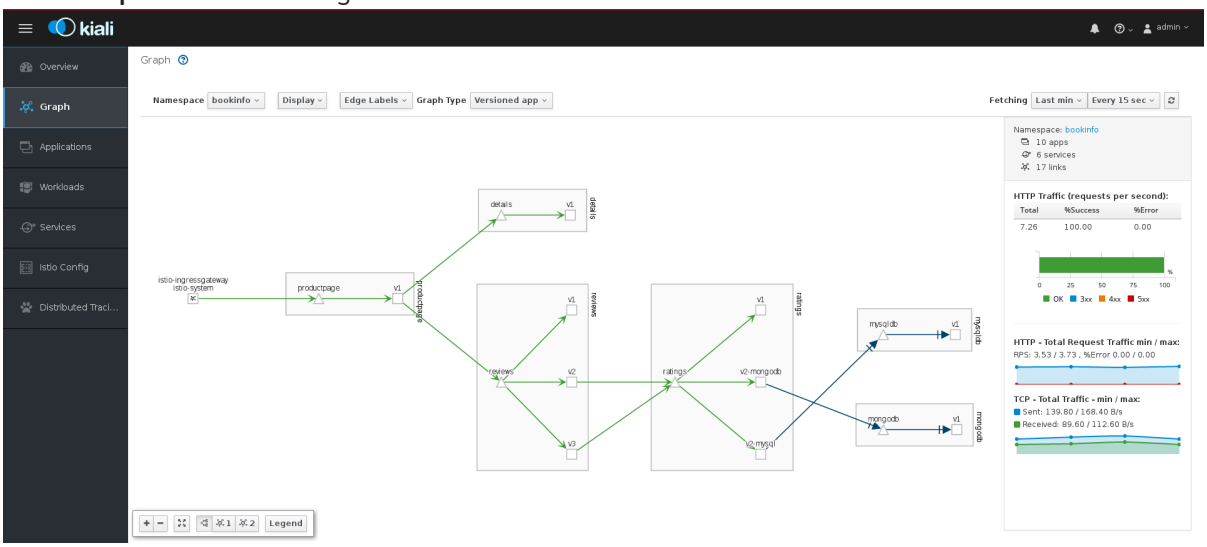
4. Use the left navigation or click one of the Namespace icons to view your Applications, Workloads, or Services.

1.7.2. Exploring the Graph page

The Graph page shows a graph of microservices, which are connected by the requests going through them. On this page, you can see how Applications, Workloads, or Services interact with each other.

Procedure

1. Click **Graph** in the left navigation.

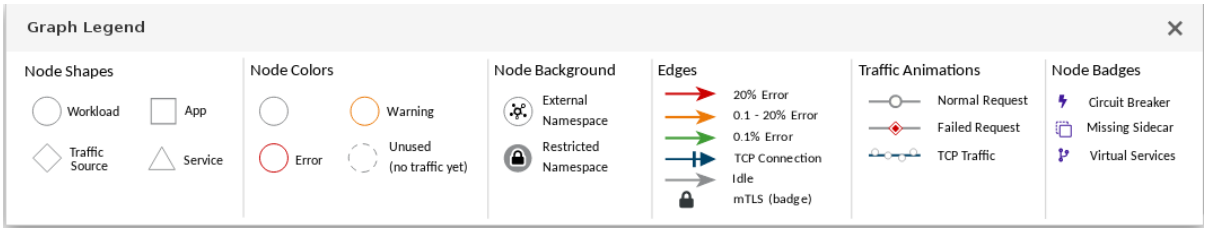


2. If necessary, select **bookinfo** from the **Namespace** menu. The graph displays the applications in the Bookinfo application.

3. Click the question mark (?) under the **Namespace** menu to take the Graph Help Tour.

4. Click **Done** to close the Help Tour.

5. Click **Legend** in the lower left corner. Kiali displays the graph legend.



6. Close the Graph Legend.

7. Hover over the **productpage** Node. Note how the graph highlights only the incoming and outgoing traffic from the Node.

8. Click the **productpage** Node. Note how the details on the right side of the page change to display the **productpage** details.

1.7.3. Exploring the Applications page

The Applications page lets you search for and view applications, their health, and other details.

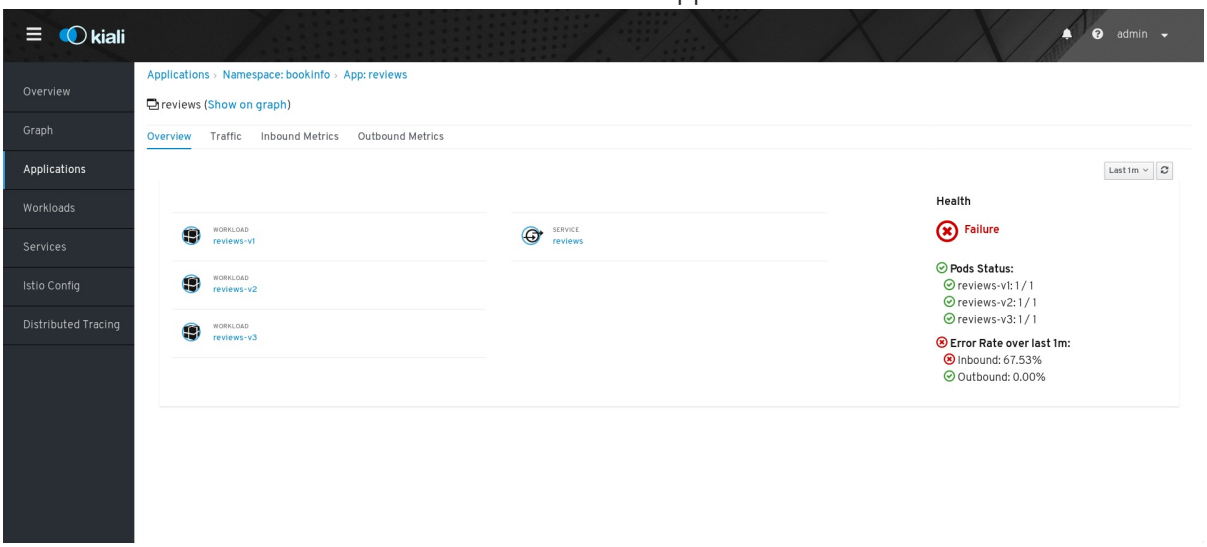
Procedure

1. Click **Applications** in the left navigation.

2. If necessary, select **bookinfo** from the **Namespace** menu. The page displays the applications in the selected Namespace and their health.

3. Hover over the Health icon to view additional health details.

4. Click the **reviews** Service to view the details for that application.



5. On the Applications Details page you can view more detailed health information, and drill down for further details about the three versions of the **reviews** Service.

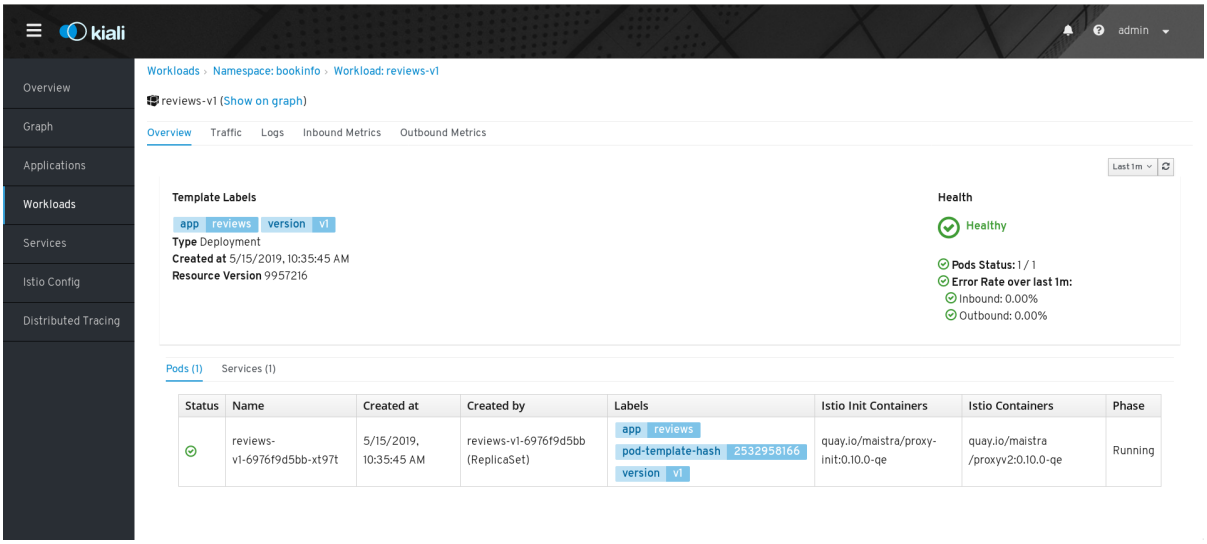
6. From the Application Details page you can also click tabs to view Traffic and Inbound and Outbound Metrics for the application.

1.7.4. Exploring the Workloads page

The Workloads page lets you search for and view Workloads, their health, and other details.

Procedure

1. Click **Workloads** in the left navigation.
2. If necessary, select **bookinfo** from the **Namespace** menu. The page displays the Workloads in the selected Namespace, their health, and labels.
3. Click the **reviews-v1** Workload to view the details for that Workload.
4. On the Workload Details page you can view an overview of Pods and Services associated with the Workload.



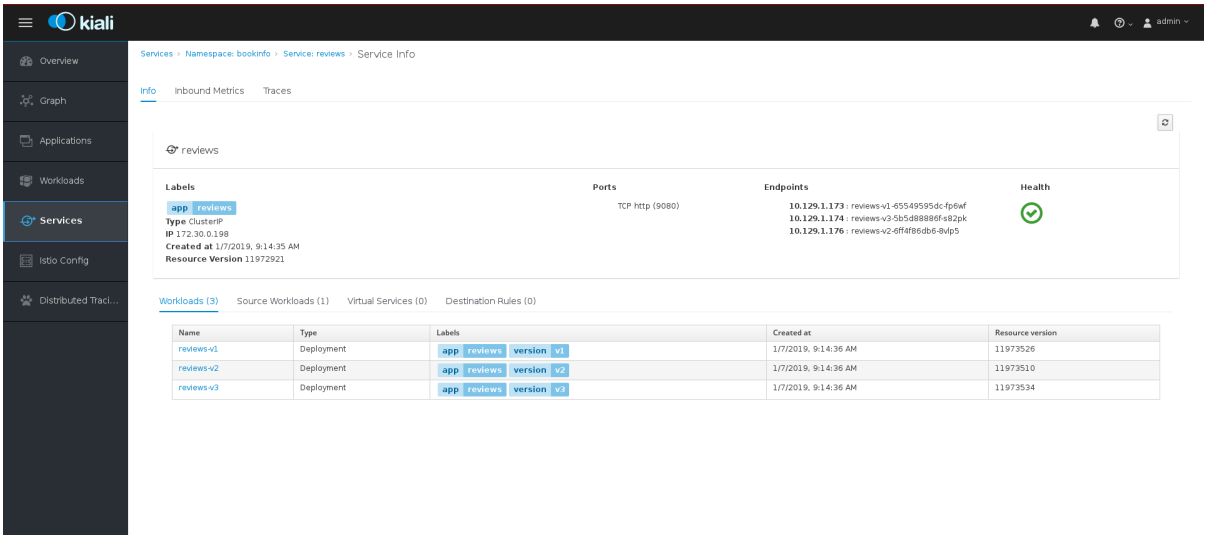
5. From the Workload Details page you can also click tabs to view Traffic, Logs, and Inbound and Outbound Metrics for the Workload.

1.7.5. Exploring the Services page

The Services page lets you search for and view Services, their health, and other details.

Procedure

1. Click **Services** in the left navigation.
2. If necessary, select **bookinfo** from the **Namespace** menu. The page displays a listing of all the Services that are running in the selected Namespace and additional information about them, such as health status.
3. Hover over the health icon for any of the Services to view health information about the Service. A Service is considered healthy when it is online and responding to requests without errors.
4. Click the **Reviews** Service to view its details. Note that there are three different versions of this Service.



5. On the Services Details page you can view an overview of Workloads, virtual Services, and destination rules associated with the Service.

6. From the Services Details page you can also click tabs to view Traffic, Inbound Metrics, and Traces for the Service.

7. Click the Actions menu. From here you can perform the following actions:

   - Create Weighted Routing
   - Create Matching Routing
   - Suspend Traffic
   - Delete ALL Traffic Routing

8. Click the name of one of the Services to view additional details about that specific version of the Service.

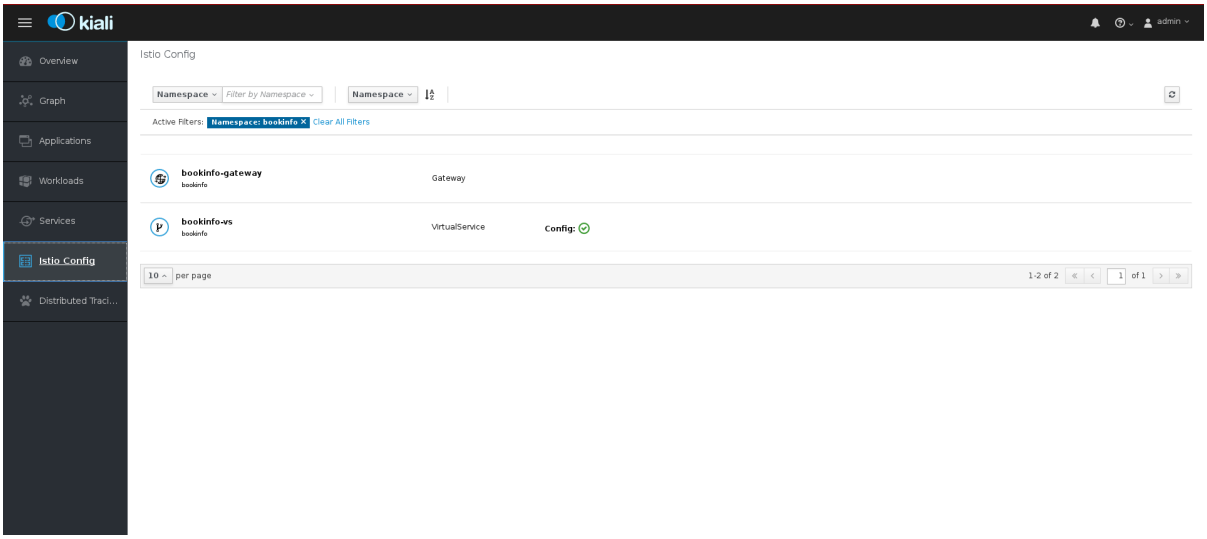
1.7.6. Exploring the Istio Config page

The Istio Config page lets you view all of the currently running configurations to your Service Mesh, such as Circuit Breakers, Destination Rules, Fault Injection, Gateways, Routes, Route Rules, and Virtual Services.

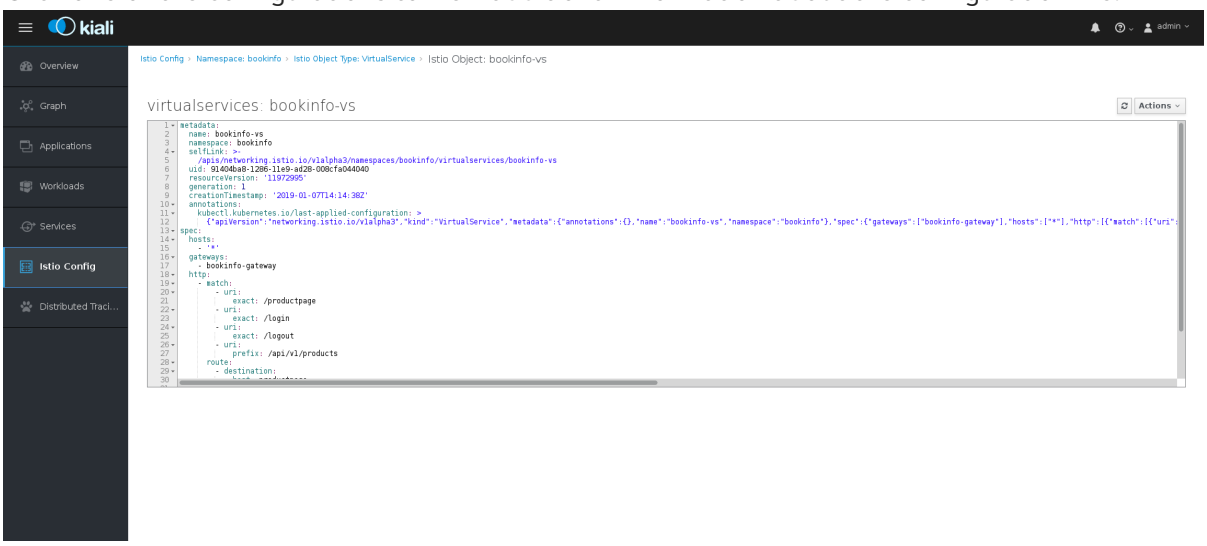
Procedure

1. Click **Istio Config** in the left navigation.

2. If necessary, select **bookinfo** from the **Namespace** menu. The page displays a listing of configurations running in the selected Namespace and validation status.



3. Click one of the configurations to view additional information about the configuration file.



1.8. DISTRIBUTED TRACING TUTORIAL

Jaeger is an open source distributed tracing system. You use Jaeger for monitoring and troubleshooting microservices-based distributed systems. Using Jaeger you can perform a trace, which follows the path of a request through various microservices that make up an application. Jaeger is installed by default as part of the Service Mesh.

1.8.1. Generating traces and analyzing trace data

This tutorial uses Service Mesh and the **bookinfo** tutorial to demonstrate how you can perform a trace using the Jaeger component of Red Hat OpenShift Service Mesh.

Prerequisites:

- OpenShift Container Platform 3.11 or higher installed.
- Red Hat OpenShift Service Mesh 0.12.TechPreview installed.
- **Bookinfo** demonstration application installed.

Procedure

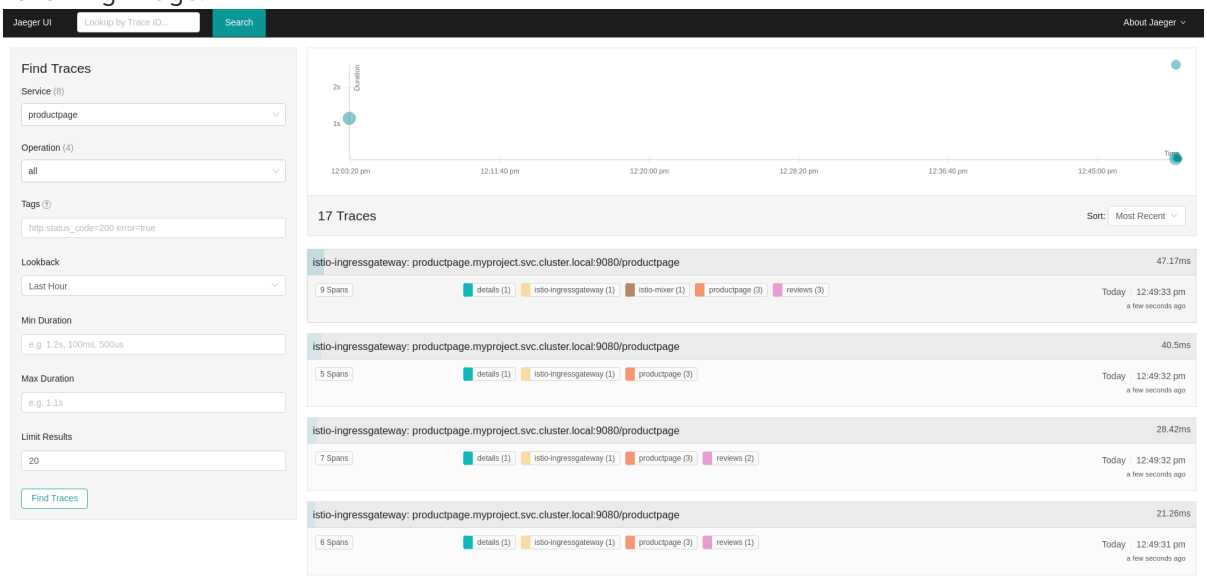
1. After you have deployed the Bookinfo application you will need to generate calls to the Bookinfo application so that you have some trace data to analyze. Access http://<GATEWAY_URL>/productpage and refresh the page a few times to generate some trace data.

2. A route to access the Jaeger dashboard already exists. Query for details of the route:

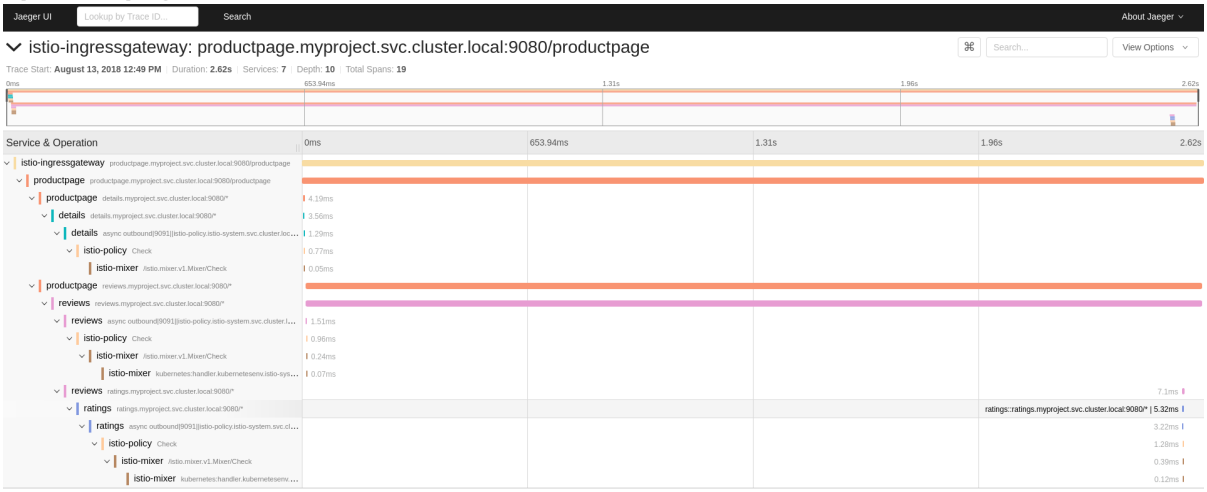
```
$ export JAEGER_URL=$(oc get route -n istio-system jaeger-query -o
jsonpath='{.spec.host}')
```

3. Launch a browser and navigate to **https://<JAEGER_URL>**.

4. In the left pane of the Jaeger dashboard, from the Service menu, select "productpage" and click the **Find Traces** button at the bottom of the pane. A list of traces is displayed, as shown in the following image:



5. Click one of the traces in the list to open a detailed view of that trace. If you click on the top (most recent) trace, you see the details that correspond to the latest refresh of the `/**productpage**.



The trace in the previous figure consists of a few nested spans, each corresponding to a Bookinfo Service call, all performed in response to a `/**productpage** request. Overall processing time was 2.62s, with the **details** Service taking 3.56ms, the **reviews** Service taking 2.6s, and the **ratings** Service taking 5.32ms. Each of the calls to remote Services is represented by a client-side and server-side span. For example, the **details** client-side span is labeled **productpage details.myproject.svc.cluster.local:9080**. The span nested underneath it, labeled **details**

details.myproject.svc.cluster.local:9080, corresponds to the server-side processing of the request. The trace also shows calls to **istio-policy**, which reflect authorization checks made by Istio.

1.9. GRAFANA TUTORIAL

Grafana is an open source a tool for creating monitoring and metric analytics and dashboards. You use Grafana to query, visualize, and alert on your metrics no matter where they are stored: Graphite, Elasticsearch, OpenTSDB, Prometheus, or InfluxDB. Istio includes monitoring via Prometheus and Grafana.

This tutorial uses Service Mesh and the **bookinfo** tutorial to demonstrate how to set up and use the Istio Dashboard to monitor mesh traffic. As part of this task, you install the Grafana Istio add-on to view service mesh traffic data.

1.9.1. Accessing the Grafana dashboard

The Grafana dashboard's web-based interface lets you visualize your metrics data.

Prerequisites:

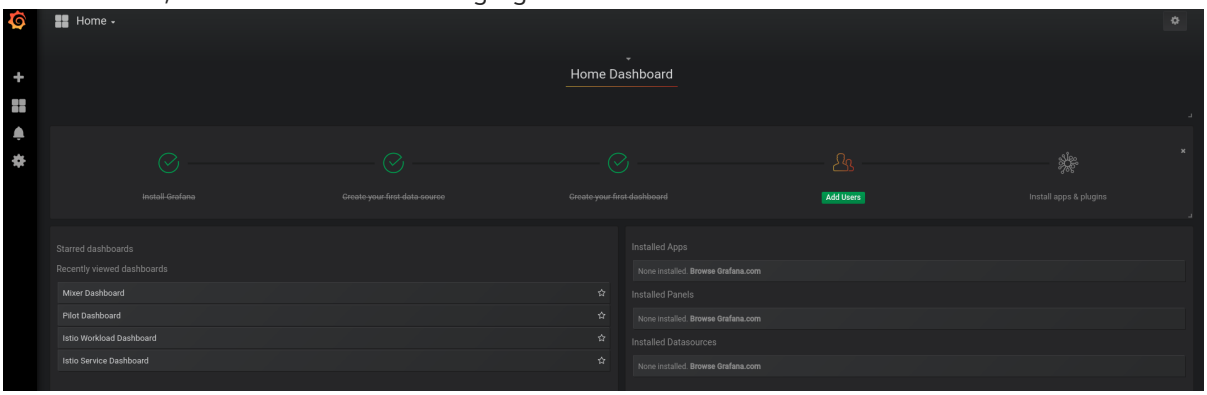
- OpenShift Container Platform 3.11 or higher installed.
- Red Hat OpenShift Service Mesh 0.12.TechPreview installed.
- **Bookinfo** demonstration application installed.

Procedure

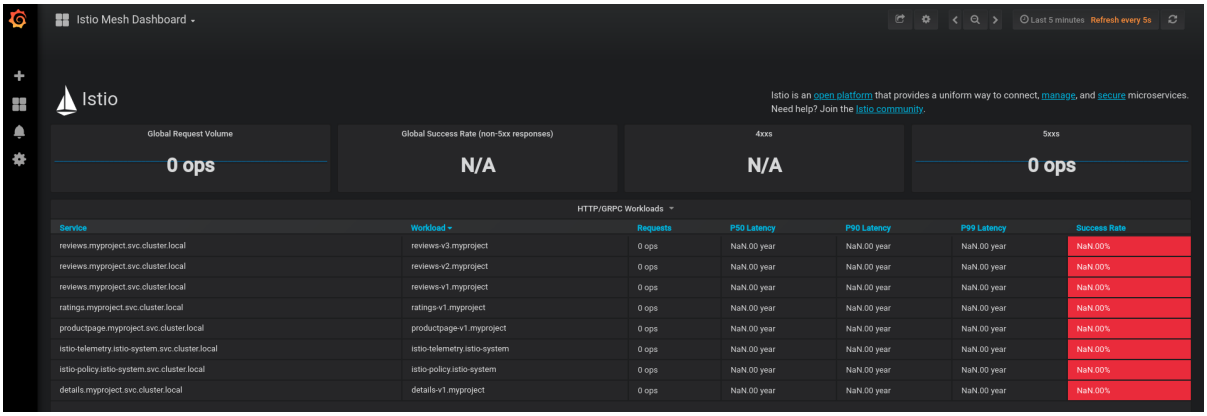
1. A route to access the Grafana dashboard already exists. Query for details of the route:

```
$ export GRAFANA_URL=$(oc get route -n istio-system grafana -o jsonpath='{.spec.host}')
```

2. Launch a browser and navigate to navigate to **http://<GRAFANA_URL>**. You see Grafana's home screen, as shown in the following figure:



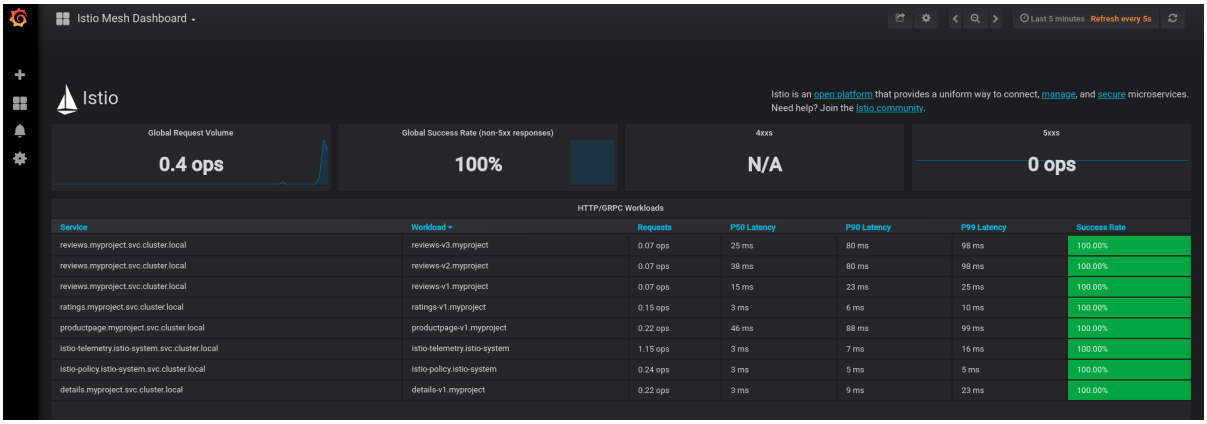
3. From the menu in the upper-left corner, select **Istio Mesh Dashboard** to see Istio mesh metrics.



4. Generate some traffic by accessing the Bookinfo application:

```
$ curl -o /dev/null http://<GATEWAY_URL>/productpage
```

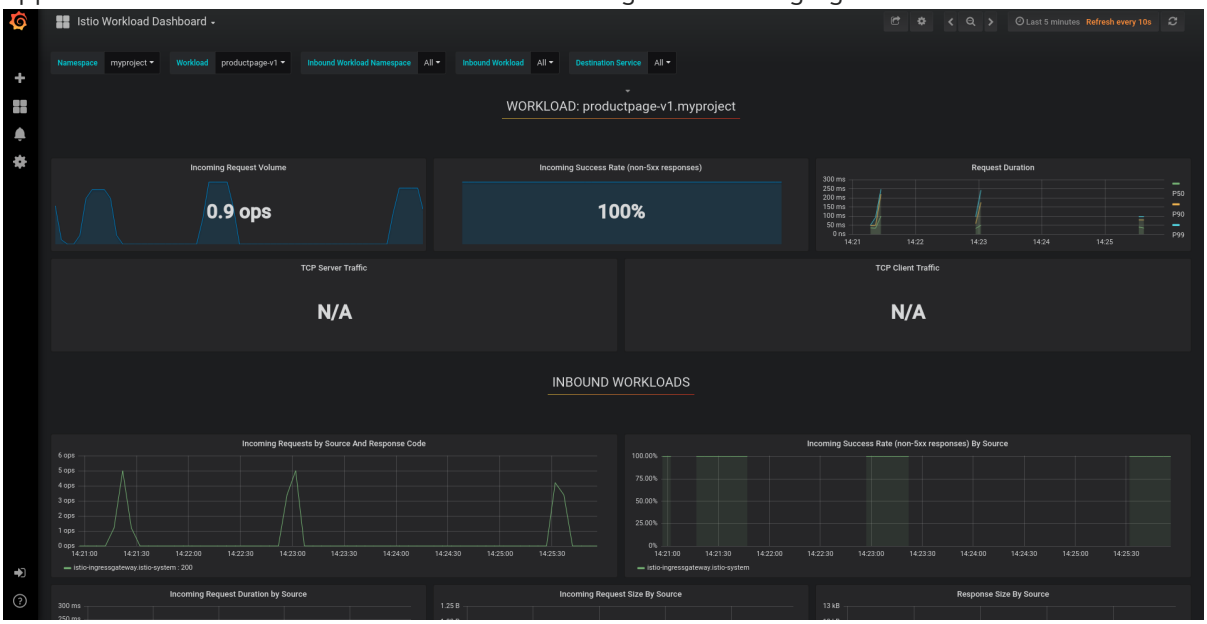
The dashboard reflects the traffic through the mesh, similar to the following figure:



5. To see detailed metrics for a Service, click on a Service name in the **Service** column. The Service dashboard resembles the following figure:

Note that **TCP Bandwidth** metrics are empty, because Bookinfo only uses http-based Services. The dashboard also displays metrics for Workloads that call the **Client Workloads** Service and for Workloads that process requests from the **Service Workloads**. You can switch to a different Service or filter metrics by client and Service Workloads by using the menus at the top of the dashboard.

6. To switch to the Workloads dashboard, click **Istio Workload Dashboard** on the menu in the upper-left corner. You will see a screen resembling the following figure:



This dashboard shows Workload metrics and metrics for client (inbound) and Service (outbound) Workloads. You can switch to a different Workload; to filter metrics by inbound or outbound Workloads, use the menus at the top of the dashboard.

1.10. PROMETHEUS TUTORIAL

Prometheus is an open source system and service monitoring toolkit. Prometheus collects metrics from configured targets at specified intervals, evaluates rule expressions, displays the results, and can trigger alerts if some condition is observed to be true. Grafana or other API consumers can be used to visualize the collected data.

1.10.1. Querying Prometheus metrics

This tutorial uses Service Mesh and the **bookinfo** tutorial to demonstrate how you can query for metrics using Prometheus.

Prerequisites:

- OpenShift Container Platform 3.11 or higher installed.
- Red Hat OpenShift Service Mesh 0.12.TechPreview installed.
- **Bookinfo** demonstration application installed.

After you have verified the Bookinfo application has deployed, you will need to generate calls to the application before you query for metrics.

Procedure

1. Verify that the **prometheus** Service is running in your cluster. In Kubernetes environments, execute the following command:

```
$ oc get svc prometheus -n istio-system
```

You will see something like the following:

```
NAME         CLUSTER-IP    EXTERNAL-IP  PORT(S)    AGE
prometheus   10.59.241.54  <none>       9090/TCP   2m
```

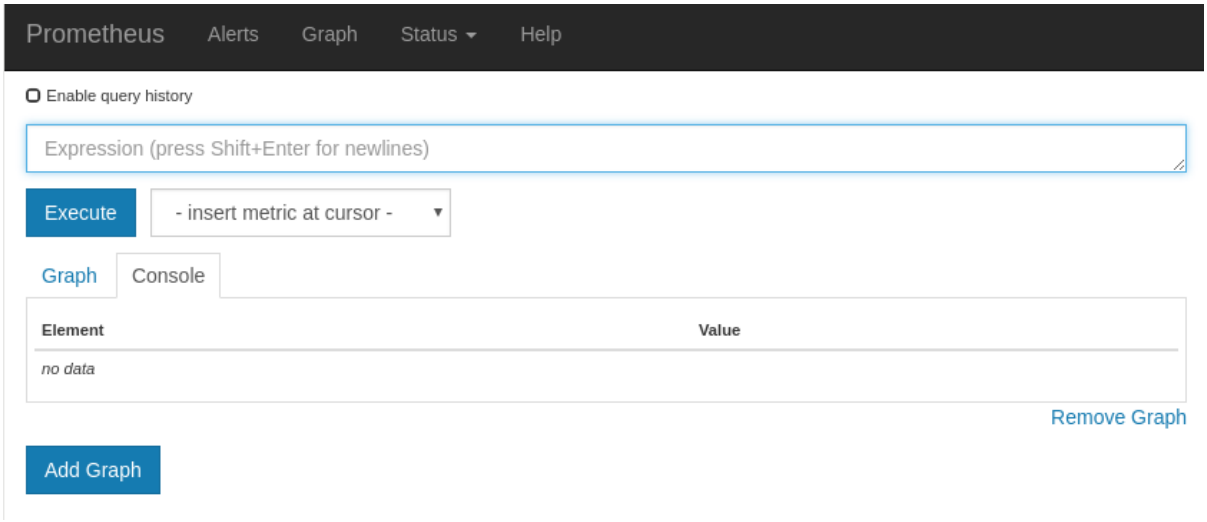
2. Generate network traffic by accessing the Bookinfo application:

```
$ curl -o /dev/null http://$GATEWAY_URL/productpage
```

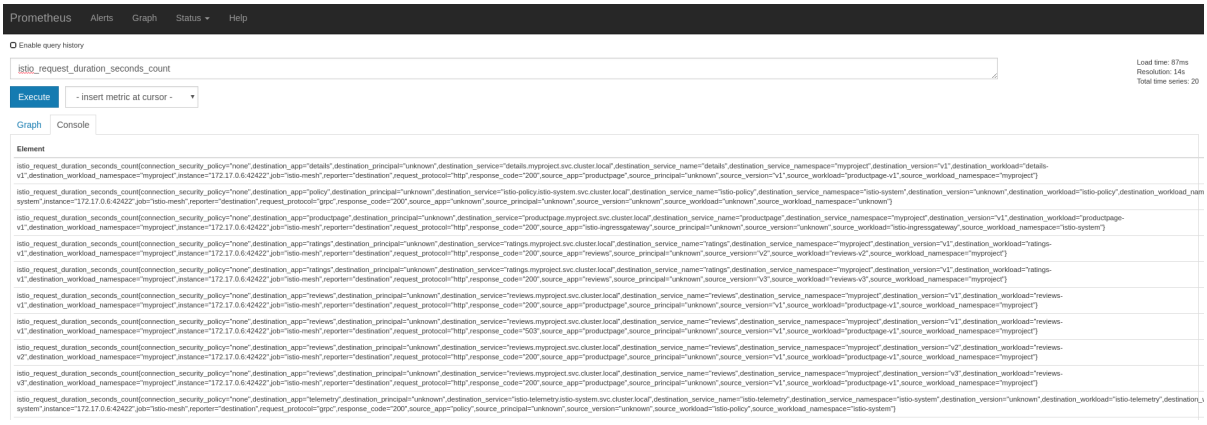
3. A route to access the Prometheus user interface already exists. Query for details of the route:

```
$ export PROMETHEUS_URL=$(oc get route -n istio-system prometheus -o
jsonpath='{.spec.host}')
```

4. Launch a browser and navigate to **http://<PROMETHEUS_URL>**. You will see the Prometheus home screen, similar to the following figure:



5. In the **Expression** field, enter **istio_request_duration_seconds_count**, and click the **Execute** button. You will see a screen similar to the following figure:



6. You can narrow down queries by using selectors. For example **istio_request_duration_seconds_count{destination_workload="reviews-v2"}** shows only counters with the matching **destination_workload** label. For more information about using queries, see the Prometheus documentation.

7. To list all available Prometheus metrics, run the following command:

```
$ oc get prometheus -n istio-system -o jsonpath='{.items[*].spec.metrics[*].name}'
requests_total request_duration_seconds request_bytes response_bytes
tcp_sent_bytes_total tcp_received_bytes_total
```

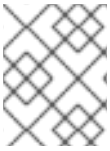
Note that returned metric names must be prepended with **istio_** when used in queries, for example, **requests_total** is **istio_requests_total**.

1.11. REMOVING RED HAT OPENSHIFT SERVICE MESH

This process allows you to remove Red Hat OpenShift Service Mesh from an existing OpenShift Container Platform instance.

1.11.1. Removing the control plane

Follow this procedure to remove the Red Hat OpenShift Service Mesh control plane.



PROCEDURE

When you remove the custom resource, Service Mesh tells the Operator to begin uninstalling everything it installed.

TIP

You can use the shortened **smcp** command in place of **servicemeshcontrolplanes**.

1. Run this command to retrieve the name of the installed custom resource:

```
$ oc get servicemeshcontrolplanes -n istio-system
```

2. Replace **<name_of_custom_resource>** with the output from the previous command, and run this command to remove the custom resource:

```
$ oc delete servicemeshcontrolplanes -n istio-system <name_of_custom_resource>
```

1.11.2. Removing the Operators

You must remove the Operators to successfully remove Red Hat OpenShift Service Mesh. Once you remove the Red Hat OpenShift Service Mesh Operator, you must remove the Jaeger Operator and the Kiali Operator.

1.11.2.1. Removing the Red Hat OpenShift Service Mesh Operator

Follow this procedure to remove the Red Hat OpenShift Service Mesh Operator.

Prerequisites

- An account with cluster administrator access.
- The Red Hat OpenShift Service Mesh Operator must be installed.

Procedure

- Run the following command to remove the Red Hat OpenShift Service Mesh Operator:

```
$ oc delete -n istio-operator -f https://raw.githubusercontent.com/Maistra/istio-operator/maistra-0.12/deploy/servicemesh-operator.yaml
```

1.11.2.2. Removing the Jaeger Operator

This procedure removes the Jaeger Operator.

Prerequisites

- An account with cluster administrator access.
- The Jaeger Operator must be installed.

Procedure

- Run the following commands to remove the Jaeger Operator:

```
$ oc delete -n observability -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/v1.13.1/deploy/operator.yaml
$ oc delete -n observability -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/v1.13.1/deploy/role_binding.yaml
$ oc delete -n observability -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/v1.13.1/deploy/role.yaml
$ oc delete -n observability -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/v1.13.1/deploy/service_account.yaml
$ oc delete -n observability -f https://raw.githubusercontent.com/jaegertracing/jaeger-operator/v1.13.1/deploy/crds/jaegertracing_v1_jaeger_crd.yaml
```

1.11.2.3. Removing the Kiali Operator

Prerequisites

- An account with cluster administrator access.
- The Kiali Operator must be installed.

Procedure

- Run the following command to remove the Kiali Operator:

```
$ bash <(curl -L https://git.io/getLatestKialiOperator) --uninstall-mode true --operator-watch-namespace '**'
```

1.11.3. Removing the projects

Follow this procedure to remove the Red Hat OpenShift Service Mesh projects.

Procedure

1. Run this command to remove the **istio-system** project:

```
$ oc delete project istio-system
```

2. Run this command to remove the **istio-operator** project:

```
$ oc delete project istio-operator
```

CHAPTER 2. 3SCALE ADAPTER

2.1. USING THE 3SCALE ISTIO ADAPTER

The 3scale Istio Adapter allows you to label a service running within the Red Hat OpenShift Service Mesh and integrate that service with the 3scale API Management solution.

2.1.1. Integrate the 3Scale adapter with Red Hat OpenShift Service Mesh

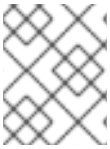
You can use these examples to configure requests to your services using the 3scale Istio Adapter.

Prerequisites:

- Red Hat OpenShift Service Mesh 0.12.0+
- A working 3scale account (SaaS or 3scale 2.5 On-Premises)
- Red Hat OpenShift Service Mesh prerequisites
- Ensure Mixer policy enforcement is enabled. Update Mixer policy enforcement provides instructions to check the current Mixer policy enforcement status and enable policy enforcement.

NOTE

To configure the 3scale Istio Adapter, refer to Red Hat OpenShift Service Mesh custom resources for instructions on adding adapter parameters to the custom resource file.



NOTE

Pay particular attention to the **kind: handler** resource. You must update this with your 3scale credentials and the service ID of the API you want to manage.

1. Modify the handler configuration with your 3scale configuration.

   Handler configuration example

```
apiVersion: "config.istio.io/v1alpha2"
kind: handler
metadata:
 name: threescale
spec:
 adapter: threescale
 params:
   service_id: "<SERVICE_ID>"
   system_url: "https://<organization>-admin.3scale.net/"
   access_token: "<ACCESS_TOKEN>"
 connection:
   address: "threescale-istio-adapter:3333"
```

2. Modify the rule configuration with your 3scale configuration to dispatch the rule to the threescale handler.

Rule configuration example

```
apiVersion: "config.istio.io/v1alpha2"
kind: rule
metadata:
  name: threescale
spec:
  match: destination.labels["service-mesh.3scale.net"] == "true"
  actions:
    - handler: threescale.handler
      instances:
        - threescale-authorization.instance
```

2.1.1.1. Generating 3scale custom resources

The adapter includes a tool that allows you to generate the **handler**, **instance**, and **rule** custom resources.

Table 2.1. Usage

Option	Description	Required	Default value
-h, --help	Produces help output for available options	No	
--name	Unique name for this URL, token pair	Yes	
-n, --namespace	Namespace to generate templates	No	istio-system
-t, --token	3scale access token	Yes	
-u, --url	3scale Admin Portal URL	Yes	
-s, --service	3scale API/Service ID	No	
--auth	3scale authentication pattern to specify (1=Api Key, 2=App Id/App Key, 3=OIDC)	No	Hybrid
-o, --output	File to save produced manifests to	No	Standard output
--version	Outputs the CLI version and exits immediately	No	

2.1.1.1.1. Generate templates from URL examples

- This example generates templates allowing the token, URL pair to be shared by multiple services as a single handler:

```
$ 3scale-gen-config --name=admin-credentials --url="https://<organization>-
admin.3scale.net:443" --token="[redacted]"
```

- This example generates the templates with the service ID embedded in the handler:

```
$ 3scale-gen-config --url="https://<organization>-admin.3scale.net" --name="my-unique-id" --
service="123456789" --token="[redacted]"
```

2.1.1.2. Generating manifests from a deployed adapter

1. Run this command to generate manifests from a deployed adapter in the **istio-system** namespace:

```
$ export NS="istio-system" URL="https://replaceme-admin.3scale.net:443" NAME="name"
TOKEN="token"
oc exec -n ${NS} $(oc get po -n ${NS} -o jsonpath='{.items[?
(@.metadata.labels.app=="3scale-istio-adapter")].metadata.name}') \
-it -- ./3scale-config-gen \
--url ${URL} --name ${NAME} --token ${TOKEN} -n ${NS}
```

2. This will produce sample output to the terminal. Edit these samples if required and create the objects using the **oc create** command.

3. When the request reaches the adapter, the adapter needs to know how the service maps to an API on 3scale. You can provide this information in two ways:

   a. Label the workload (recommended)

   b. Hard code the handler as **service_id**

4. Update the workload with the required annotations:

NOTE

You only need to update the service ID provided in this example if it is not already embedded in the handler. **The setting in the handler takes precedence**

```
$ export CREDENTIALS_NAME="replace-me"
export SERVICE_ID="replace-me"
export DEPLOYMENT="replace-me"
patch="$(oc get deployment "${DEPLOYMENT}"
patch="$(oc get deployment "${DEPLOYMENT}" --template='{"spec":{"template":{"metadata":
{"labels":{ {{ range $k,$v := .spec.template.metadata.labels }}"{{ $k }}":"{{ $v }}",{{ end
}}"service-mesh.3scale.net/service-id":"""${SERVICE_ID}""","service-
mesh.3scale.net/credentials":"""${CREDENTIALS_NAME}"""}}}}}' )"
oc patch deployment "${DEPLOYMENT}" --patch ""${patch}""
```

2.1.1.3. Routing service traffic through the adapter

Follow these steps to drive traffic for your service through the 3scale adapter.

Prerequisites

- Credentials and service ID from your 3scale administrator.

Procedure

1. Match the rule **destination.labels["service-mesh.3scale.net/credentials"] == "threescale"** that you previously created in the configuration, in the **kind: rule** resource.
2. Add the above label to **PodTemplateSpec** on the Deployment of the target workload to integrate a service. the value, **threescale**, refers to the name of the generated handler. This handler stores the access token required to call 3scale.
3. Add the **destination.labels["service-mesh.3scale.net/service-id"] == "replace-me"** label to the workload to pass the service ID to the adapter via the instance at request time.

2.1.2. Configure the integration settings in 3scale

Follow this procedure to configure the 3scale integration settings.

NOTE

For 3scale SaaS customers, Red Hat OpenShift Service Mesh is enabled as part of the Early Access program.

Procedure

1. Navigate to **[your_API_name] → Integration → Configuration**.
2. At the top of the **Integration** page click on **edit integration settings** in the top right corner.
3. Under the **Service Mesh** heading, click the **Istio** option.
4. Scroll to the bottom of the page and click **Update Service**.

2.1.3. Caching behavior

Responses from 3scale System APIs are cached by default within the adapter. Entries will be purged from the cache when they become older than the **cacheTTLSeconds** value. Also by default, automatic refreshing of cached entries will be attempted seconds before they expire, based on the **cacheRefreshSeconds** value. You can disable automatic refreshing by setting this value higher than the **cacheTTLSeconds** value.

Caching can be disabled entirely by setting **cacheEntriesMax** to a non-positive value.

By using the refreshing process, cached values whose hosts become unreachable will be retried before eventually being purged when past their expiry.

2.1.4. Authenticating requests

This Technology Preview release supports the following authentication methods:

- **Standard API Keys**: single randomized strings or hashes acting as an identifier and a secret token.

- **Application identifier and key pairs** immutable identifier and mutable secret key strings.
- **OpenID authentication method**: client ID string parsed from the JSON Web Token.

2.1.4.1. Applying authentication patterns

Modify the **instance** custom resource, as illustrated in the following authentication method examples, to configure authentication behavior. You can accept the authentication credentials from:

- Request headers
- Request parameters
- Both request headers and query parameters

NOTE

When specifying values from headers they must be lower case. For example, if you want to send a header as **X-User-Key**, this must be referenced in the configuration as **request.headers["x-user-key"]**.

2.1.4.1.1. API key authentication method

Service Mesh looks for the API key in query parameters and request headers as specified in the **user** option in the **subject** custom resource parameter. It checks the values in the order given in the custom resource file. You can restrict the search for the API key to either query parameters or request headers by omitting the unwanted option.

In this example Service Mesh looks for the API key in the **user_key** query parameter. If the API key is not in the query parameter, Service Mesh then checks the **x-user-key** header.

API key authentication method example

```
apiVersion: "config.istio.io/v1alpha2"
kind: instance
metadata:
  name: threescale-authorization
  namespace: istio-system
spec:
  template: authorization
  params:
    subject:
      user: request.query_params["user_key"] | request.headers["x-user-key"] | ""
    action:
      path: request.url_path
      method: request.method | "get"
```

If you want the adapter to examine a different query parameter or request header, change the name as appropriate. For example, to check for the API key in a query parameter named “key”, change **request.query_params["user_key"]** to **request.query_params["key"]**.

2.1.4.1.2. Application ID and application key pair authentication method

Service Mesh looks for the application ID and application key in query parameters and request headers, as specified in the **properties** option in the **subject** custom resource parameter. The application key is

optional. It checks the values in the order given in the custom resource file. You can restrict the search for the credentials to either query parameters or request headers by not including the unwanted option.

In this example, Service Mesh looks for the application ID and application key in the query parameters first, moving on to the request headers if needed.

Application ID and application key pair authentication method example

```
apiVersion: "config.istio.io/v1alpha2"
kind: instance
metadata:
  name: threescale-authorization
  namespace: istio-system
spec:
  template: authorization
  params:
    subject:
        app_id: request.query_params["app_id"] | request.headers["x-app-id"] | ""
        app_key: request.query_params["app_key"] | request.headers["x-app-key"] | ""
    action:
      path: request.url_path
      method: request.method | "get"
```

If you want the adapter to examine a different query parameter or request header, change the name as appropriate. For example, to check for the application ID in a query parameter named **identification**, change **request.query_params["app_id"]** to **request.query_params["identification"]**.

2.1.4.1.3. OpenID authentication method

To use the *OpenID Connect (OIDC) authentication method* , use the **properties** value on the **subject** field to set **client_id**, and optionally **app_key**.

You can manipulate this object using the methods described previously. In the example configuration shown below, the client identifier (application ID) is parsed from the JSON Web Token (JWT) under the label *azp*. You can modify this as needed.

OpenID authentication method example

```
apiVersion: "config.istio.io/v1alpha2"
kind: instance
metadata:
  name: threescale-authorization
spec:
  template: threescale-authorization
  params:
    Subject:
properties:
        app_key: request.query_params["app_key"] | request.headers["x-app-key"] | ""
        client_id: request.auth.claims["azp"] | ""
    action:
      path: request.url_path
      method: request.method | "get"
        service: destination.labels["service-mesh.3scale.net/service-id"] | ""
```

For this integration to work correctly, OIDC must still be done in 3scale for the client to be created in the identity provider (IdP). You should create end-user authentication for the service you want to protect in the same namespace as that service. The JWT is passed in the **Authorization** header of the request.

In the sample **Policy** defined below, replace **issuer** and **jwksUri** as appropriate.

OpenID Policy example

```
apiVersion: authentication.istio.io/v1alpha1
kind: Policy
metadata:
  name: jwt-example
  namespace: bookinfo
spec:
  origins:
    - jwt:
        issuer: >-
          http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak
        jwksUri: >-
http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak/protocol/openid-
connect/certs
  principalBinding: USE_ORIGIN
  targets:
    - name: productpage
```

2.1.4.1.4. Hybrid authentication method

You can choose to not enforce a particular authentication method and accept any valid credentials for either method. If both an API key and an application ID/application key pair are provided, Service Mesh uses the API key.

In this example, Service Mesh checks for an API key in the query parameters, then the request headers. If there is no API key, it then checks for an application ID and key in the query parameters, then the request headers.

Hybrid authentication method example

```
apiVersion: "config.istio.io/v1alpha2"
kind: instance
metadata:
  name: threescale-authorization
spec:
  template: authorization
  params:
    subject:
      user: request.query_params["user_key"] | request.headers["x-user-key"] |
      properties:
        app_id: request.query_params["app_id"] | request.headers["x-app-id"] | ""
        app_key: request.query_params["app_key"] | request.headers["x-app-key"] | ""
        client_id: request.auth.claims["azp"] | ""
    action:
      path: request.url_path
      method: request.method | "get"
        service: destination.labels["service-mesh.3scale.net/service-id"] | ""
```

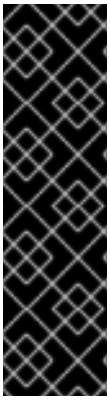
2.1.5. 3scale Adapter metrics

The adapter, by default reports various Prometheus metrics that are exposed on port **8080** at the /**metrics** endpoint. These metrics provide insight into how the interactions between the adapter and 3scale are performing. The service is labeled to be automatically discovered and scraped by Prometheus.

CHAPTER 3. SERVICE MESH RELEASE NOTES

CHAPTER 4. RED HAT OPENSHIFT SERVICE MESH RELEASE NOTES

4.1. RED HAT OPENSHIFT SERVICE MESH OVERVIEW



IMPORTANT

This release of Red Hat OpenShift Service Mesh is a Technology Preview release only. Technology Preview releases are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete, and Red Hat does NOT recommend using them for production. Using Red Hat OpenShift Service Mesh on a cluster renders the whole OpenShift cluster as a technology preview, that is, in an unsupported state. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information see Red Hat Technology Preview Features Support Scope.

Red Hat OpenShift Service Mesh is a platform that provides behavioral insight and operational control over the service mesh, providing a uniform way to connect, secure, and monitor microservice applications.

The term *service mesh* describes the network of microservices that make up applications in a distributed microservice architecture and the interactions between those microservices. As a service mesh grows in size and complexity, it can become harder to understand and manage.

Based on the open source Istio project, Red Hat OpenShift Service Mesh adds a transparent layer on existing distributed applications without requiring any changes to the service code. You add Red Hat OpenShift Service Mesh support to services by deploying a special sidecar proxy throughout your environment that intercepts all network communication between microservices. You configure and manage the service mesh using the control plane features.

Red Hat OpenShift Service Mesh provides an easy way to create a network of deployed services that provides discovery, load balancing, service-to-service authentication, failure recovery, metrics, and monitoring. A service mesh also provides more complex operational functionality, including A/B testing, canary releases, rate limiting, access control, and end-to-end authentication.

4.2. GETTING SUPPORT

If you experience difficulty with a procedure described in this documentation, visit the Red Hat Customer Portal at http://access.redhat.com. Through the customer portal, you can:

- Search or browse through the Red Hat Knowledgebase of technical support articles about Red Hat products
- Submit a support case to Red Hat Global Support Services (GSS)
- Access other product documentation

If you have a suggestion for improving this guide or have found an error, please submit a Bugzilla report at http://bugzilla.redhat.com against **Product** for the **Documentation** component. Please provide specific details, such as the section number, guide name, and Service Mesh version so we can easily locate the content.

4.2.1. New features Technology Preview 12

This release of Red Hat OpenShift Service Mesh adds support for Istio 1.1.8, Jaeger 1.13.1, Kiali 1.0.0, and the 3scale Istio Adapter 0.7.1.

Other notable changes in this release include the following:

- Integrates the Kiali and Jaeger operators into the installation.
- Adds support for Kubernetes Container Network Interface (CNI).
- Adds support for Operator Lifecycle Management (OLM).
- Updates the Istio OpenShift Router for multitenancy.
- Defaults to configuring the control planes for multitenancy. You can still configure a single tenant control plane in Red Hat OpenShift Service Mesh 0.12.TechPreview.

NOTE

To simplify installation and support, future releases will only support the a multi-tenant control plane for one or more tenants.

4.2.2. New features Technology Preview 11

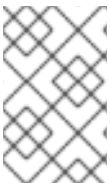
The release of Red Hat OpenShift Service Mesh adds support for multi-tenant installations, Red Hat Enterprise Linux (RHEL) Universal Base Images (UBI8), OpenSSL 1.1.1, Kiali 0.20.x, the 3scale Istio Adapter 0.6.0, and Istio 1.1.5.

4.2.3. New features Technology Preview 10

The release of Red Hat OpenShift Service Mesh adds support for Kiali 0.16.x, the 3scale Istio Adapter 0.5.0, and Istio 1.1.

4.2.4. New features Technology Preview 9

The release of Red Hat OpenShift Service Mesh adds support for Kiali 0.15.x, Jaeger 1.11, the 3scale Istio Adapter 0.4.1, and Istio 1.1.0-rc.2.



NOTE

Starting with Red Hat OpenShift Service Mesh 0.9.TechPreview, Mixer's policy enforcement is disabled by default, but you must enable it to run policy tasks. See Update Mixer policy enforcement for instructions on enabling Mixer policy enforcement.

4.2.5. New features Technology Preview 8

The release of Red Hat OpenShift Service Mesh adds support for Kiali 0.14.x and the 3scale Istio Adapter 0.3.

4.2.6. New features Technology Preview 7

The release of Red Hat OpenShift Service Mesh adds the 3scale Istio Adapter and support for Kiali 0.13.x, Jaeger 1.9.0, and Istio 1.1.

4.2.7. New features Technology Preview 6

The release of Red Hat OpenShift Service Mesh adds support for Kiali 0.11.x and and Istio 1.0.5.

4.2.8. New features Technology Preview 5

The release of Red Hat OpenShift Service Mesh adds support for Kiali 0.10.x, Jaeger 1.8.1, and Istio 1.0.4.

4.2.9. New features Technology Preview 4

The release of Red Hat OpenShift Service Mesh adds support for Kiali 0.9.x and Istio 1.0.3.

4.2.10. New features Technology Preview 3

The release of Red Hat OpenShift Service Mesh adds support for OpenShift 3.11, support for Kiali 0.8.x, and an updated base Ansible installer (3.11.16-3).

4.2.11. New features Technology Preview 2

The release adds the Kiali observability console to Red Hat OpenShift Service Mesh. Kiali provides a number of graphs that you can use to view the topography and health of the microservices that make up your service mesh. You can view predefined dashboards that provide detailed request and response metrics (volume, duration, size, TCP traffic) per inbound and outbound traffic. You can also browse your service mesh by application, workloads, and services to view the health of each element.

4.2.12. New features Technology Preview 1

Red Hat OpenShift Service Mesh provides a number of key capabilities uniformly across a network of services:

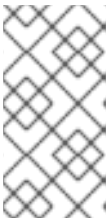
- **Traffic Management** - Control the flow of traffic and API calls between services, make calls more reliable, and make the network more robust in the face of adverse conditions.
- **Service Identity and Security**- Provide services in the mesh with a verifiable identity and provide the ability to protect service traffic as it flows over networks of varying degrees of trustworthiness.
- **Policy Enforcement** - Apply organizational policy to the interaction between services, ensure access policies are enforced and resources are fairly distributed among consumers. Policy changes are made by configuring the mesh, not by changing application code.
- **Telemetry** - Gain understanding of the dependencies between services and the nature and flow of traffic between them, providing the ability to quickly identify issues.

4.3. KNOWN ISSUES

These limitations exist in Red Hat OpenShift Service Mesh at this time:

- Red Hat OpenShift Service Mesh does not support IPv6, as it it not supported by the upstream Istio project, nor fully supported by OpenShift.
- The istio-init container requires a privileged security context or at least to run as root and to have the NET_ADMIN capability. The istio-init container needs to be privileged because it needs to properly configure the iptables rules in the pod to intercept network connections. The team

is currently investigating ideas for ways to reduce the privileges required by Istio.



NOTE

The Istio CNI plugin allows a pod to join the service mesh without requiring additional privileges to be granted for each pod. To enable the plugin, edit the control plane custom resource file to set the **enabled** field in the **istio_cni** section to **true**.

- Graph layout - The layout for the Kiali graph can render differently, depending on your application architecture and the data to display (number of graph nodes and their interactions). Because it is difficult if not impossible to create a single layout that renders nicely for every situation, Kiali offers a choice of several different layouts. To choose a different layout, you can choose a different **Layout Schema** from the **Graph Settings** menu.

NOTE

While Kafka publisher is included in the release as part of Jaeger, it is not supported.

4.3.1. Red Hat OpenShift Service Mesh Issues

These are the known issues in Red Hat OpenShift Service Mesh at this time:

- MAISTRA-684 The default Jaeger version in the **istio-operator** is 1.12.0, which does not match Jaeger version 1.13.1 that shipped in Red Hat OpenShift Service Mesh 0.12.TechPreview.
  If you install Red Hat OpenShift Service Mesh 0.12.TechPreview without changing the Jaeger version to 1.13.1, this is what you see when you check the pods in **istio-system**:

```
$ oc get pods -n istio-system

NAME                                READY   STATUS             RESTARTS   AGE
elasticsearch-0                     1/1     Running            0          16m
grafana-694d54c786-m6dfc            2/2     Running            0          18m
istio-citadel-7658c96954-r8p46      1/1     Running            0          18m
istio-egressgateway-d759556b8-hwc7n 1/1     Running            0          18m
istio-galley-7cf565999f-b729t       1/1     Running            0          18m
istio-ingressgateway-bc97545d5-5mpw4 1/1    Running            0          18m
istio-pilot-dd7c67fb5-r8nbt         2/2     Running            0          29m
istio-policy-b5df8c557-5xbbl        2/2     Running            0          18m
istio-sidecar-injector-5bccb75987-xl6t6 1/1 Running            0          18m
istio-telemetry-7f86b4f6d9-kgl5p    2/2     Running            0          30m
jaeger-collector-85c597698c-54b2c   0/1     ImagePullBackOff   0          18m
jaeger-query-59b877c9d9-92vmj       1/3     ImagePullBackOff   0          18m
kiali-54ff784b57-8clh2              1/1     Running            0          18m
prometheus-7b89468cf6-pbnqh         2/2     Running            0          18m
```

Run this command to see **istio-system** events:

```
$ oc get events -n istio-system
```

You will see this error:

```
...
```

```
8m50s       Warning   Failed                  pod/jaeger-query-59b877c9d9-92vmj
Failed to pull image "registry.redhat.io/distributed-tracing-tech-preview/jaeger-agent:1.12.0":
rpc error: code = Unknown desc = Error reading manifest 1.12.0 in
registry.redhat.io/distributed-tracing-tech-preview/jaeger-agent: error parsing HTTP 404
response body: invalid character 'F' looking for beginning of value: "File not found.\""
...
```

The workaround is to make the following change to your custom resource to ensure you install Jaeger 1.13.1:

```
tracing:
    enabled: true
    jaeger:
      tag: 1.13.1
```

- MAISTRA-622 In Maistra 0.12.0/TP12, permissive mode does not work. The user has the option to use Plain text mode or Mutual TLS mode, but not permissive.

- MAISTRA-572 Jaeger cannot be used with Kiali. In this release Jaeger is configured to use the OAuth proxy, but is also only configured to work through a browser and doesn't allow service access. Kiali cannot properly communicate with the Jaeger endpoint and it considers Jaeger to be disabled. See also TRACING-591.

- MAISTRA-465 The Maistra operator fails to create a service for operator metrics.

- MAISTRA-453 If you create a new project and deploy pods immediately, sidecar injection does not occur. The operator fails to add the **maistra.io/member-of** before the pods are created, therefore the pods must be deleted and recreated for sidecar injection to occur.

- MAISTRA-357 In OpenShift 4 Beta on AWS, it is not possible, out of the box, to access a TCP or HTTPS service through the ingress gateway on a port other than port 80. The AWS load balancer has a health check that verifies if port 80 on the service endpoint is active.
  The load balancer health check only checks the first port defined in the Istio ingress gateway ports list. This port is configured as 80/HTTP:31380/TCP. Without a service running on this port, the load balancer health check fails.

  To check HTTPS or TCP traffic by using an ingress gateway, you must have an existing HTTP service, for example, the Bookinfo sample application product page running on the ingress gateway port 80. Alternatively, using the AWS EC2 console, you can change the port that the load balancer uses to perform the health check, and replace 80 with the port your service actually uses.

- MAISTRA-348 To access a TCP service by using the ingress gateway on a port other than 80 or 443, use the service hostname provided by the AWS load balancer rather than the OpenShift router.
  The istio-ingressgateway route hostname (for example, **istio-ingressgateway-istio-system.apps.[cluster name].openshift.com**) works with port 80 or port 443 traffic. However, that route hostname does not support other port traffic.

  To access service(s) running on the ingress gateway TCP port(s), you can retrieve the istio-ingressgateway external hostname (for example, **[uuid].[aws region].elb.amazonaws.com**) and then check traffic by using that external hostname value.

  To retrieve the external IP hostname value, issue this command:

```
$ oc -n istio-system get service istio-ingressgateway -o
jsonpath='{.status.loadBalancer.ingress[0].hostname}'
```

- MAISTRA-193 Unexpected console info messages are visible when health checking is enabled for citadel.

- MAISTRA-158 Applying multiple gateways referencing the same hostname will cause all gateways to stop functioning.

- MAISTRA-108 Part of the process to install Operator Lifecycle Manager (OLM) in an OpenShift Container Platform cluster is to run the registry authorization playbook, but an error prevents the process from completing successfully.
  The task "openshift_control_plane : Check for file paths outside of /etc/origin/master in master's config" fails because if finds "/dev/null" string within the master-config.yaml file. This string is added when we patch the configuration when preparing the cluster to install Istio.

  Here are the two webhooks:

  ```
  MutatingAdmissionWebhook:
  configuration:
  apiVersion: apiserver.config.k8s.io/v1alpha1
  kind: WebhookAdmission
  kubeConfigFile: /dev/null
  ```

  ```
  ValidatingAdmissionWebhook:
  configuration:
  apiVersion: apiserver.config.k8s.io/v1alpha1s
  kind: WebhookAdmission
  kubeConfigFile: /dev/null
  ```

  Create a soft link from **/var/lib/origin/null** to **/dev/null** and replace the two kubeConfigFile variables to **/var/lib/origin/null**. This makes the task think /var/lib/origin/null is an asset in the origin "perimeter" and continue running.

- MAISTRA-62 After a successful Istio installation, when upgrading an OCP cluster from 3.10 to 3.11, the storage upgrade task fails causing the entire upgrade process to fail.

4.3.2. Kiali Issues

- KIALI-3118 After changes to the ServiceMeshMemberRoll, for example adding or removing namespaces, the Kiali pod restarts and then displays errors on the Graph page while the Kiali pod is restarting.

- KIALI-3096 Runtime metrics fail in Service Mesh. There is an oauth filter between the Service Mesh and Prometheus, requiring a bearer token to be passed to Prometheus before access will be granted. Kiali has been updated to use this token when communicating to the Prometheus server, but the application metrics are currently failing with 403 errors.

- KIALI-2206 When you are accessing the Kiali console for the first time, and there is no cached browser data for Kiali, the "View in Grafana" link on the Metrics tab of the Kiali Service Details page redirects to the wrong location. The only way you would encounter this issue is if you are accessing Kiali for the first time.

- KIALI-507 Kiali does not support Internet Explorer 11. This is because the underlying frameworks do not support Internet Explorer. To access the Kiali console, use one of the two most recent versions of the Chrome, Edge, Firefox or Safari browser.

4.4. FIXED ISSUES

The following issues been resolved in the current release:

- MAISTRA-4 - The uninstall does not remove all the files, and as a result, when you re-install the istio-operator installation fails because **customresourcedefinitions.apiextensions.k8s.io "installations.istio.openshift.com"** already exists.
- MAISTRA-5 - **openshift-ansible-istio-installer-job** pod tries to start but with errors.
- MAISTRA-13 - Installer should determine release version from installed components. At present, the installer queries the yaml file, however if the yaml has been modified, the installer is not able to remove an older version.
- MAISTRA-21 - The default in the installer of 512Mi was too low for tracing. Increased default Elasticsearch memory from 512 MB to 1 GB.
- MAISTRA-61 After all applicable resources are deployed to OpenShift, Istio sidecars may lose information about their routes and can no longer communicate with services until the next update is received.
- MAISTRA-79 - Running the **istiooc cluster up** command results in the istio-operator namespace deploying a pod responsible for continually ensuring the Elasticsearch sysctl requirements are met. This loop runs constantly causing a significant load on the system running the cluster.
- MAISTRA-110 In large clusters, citadel can take a significant amount of time to create secrets. This may cause the installation to fail.
- MAISTRA-157 The conditional rate limiting feature does not restrict access as expected.
- MAISTRA-196 If you edit the installation to modify a parameter, for example to enable authentication, the new installation will fail due to the existence of the new 1.1 CRD configmaps.
- MAISTRA-420 [Multi-tenant implementation] The Jaeger agent pods are unscheduled as a result of port collision in multi-tenancy deployment on OpenShift 4.1.rc.5.
- MAISTRA-245 The sidecar injector pod fails to start if you are running the upstream community version.
- MAISTRA-462 [Multi-tenant implementation] After adding a namespace member to a second control plane, Kiali does not display the namespace member in the namespace list because the namespace for the second control plane is missing the **maistra.io/member-of** and **istio.openshift.io/member-of** labels. This is a result of the installation of the second control plane failing due to MAISTRA-464(the operator can't create the **istio-ingressgateway** service in the second control plane, because the NodePort already being used by the first control plane). The workaround is to manually change the node ports of the **istio-ingressgateway** service in the first control plane's namespace (using **oc edit svc**). The operator will then be able to finish deploying the second control plane.
- MAISTRA-466 [Multi-tenant implementation] When you install more than one control plane, the operator overwrites the Kiali oauthclient yaml on any existing Kiali installations so that only the most recent installation functions. The Kiali oauthclient (**oc get oauthclients kiali**) contains a

list of the authorized URLs that OpenShift OAuth will redirect to. If your URL is not part of this list, then the oauth login will fail and return the following message:

> `{"error":"invalid_request","error_description":"The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed."}

- MAISTRA-469 If you delete a project before deleting the ServiceMeshControlPlane, the cleanup process does not execute for resources in the deleted project. Service accounts added to the SecurityContextConstraints are not removed and the Kiali OAuthClient resource is not updated properly.

- MAISTRA-470 If you include the control plane namespace in the member roll, the reconciliation process produces an error that spams the logs and prevents the status of the configuredMembers list from updating even for members that were successfully configured.

- KIALI-1284 In Istio, a Workload can be any pod or group of pods, regardless where they originate from. They may come from Kubernetes Deployments, Replica Sets or even as a single "orphan" pod. In Kiali the current assumption is that a Workload comes from a Deployment. This should represent the vast majority of the cases.

- KIALI-1570 When a graph is loading in the Kiali console, a message that the graph is empty is displayed instead of a message that the graph is loading.

- KIALI-1572 If you see this ERROR message when you view the Kiali logs, you can ignore it:

> Failed to determine console version from file [/opt/kiali/console/version.txt]. error=open /opt/kiali/console/version.txt: no such file or directory Kiali: Console version: unknown

- KIALI-1609 When dealing with very small values (for example, less than 0.0.1 rps) you might encounter some inconsistencies in the graph. We are working on making changes to have this function better when dealing with very small values.

- KIALI-2261 In the Kiali graph, unused links (that is, edges with no traffic) are being labeled as having 100% of the request traffic, even though there is currently no request traffic. See also KIALI-2296.

- KIALI-2403 The Istio version is no longer listed on the Kiali About page after moving to Istio 1.1.0-snapshot.6, because the latest Istio Pilot now listens on a different port. Istio Pilot listens on port 8080, and you can visit Pilot to determine the Istio version (http://istio-pilot:8080/version).

- KIALI-2430 When you click on a TCP edge, and then click on an HTTP edge the graph crashes in Kiali.

- KIALI-2671 If you **do not** specify a dashboard user/password in the control plane custom resource the operator will use Oauth for the installation. The OpenShift strategy for using Oauth does not work in Red Hat OpenShift Service Mesh 0.12.TechPreview Technology Preview 10. As a workaround, ensure that you provide a user and password in the custom resource.

- KIALI-2942 When using OpenShift OAuth, when you click the logout link, you are still logged into the Kiali console. When you go to the login URL it will check your cookies and then automatically log you back in. The workaround is to delete the Kiali cookie.