



# OpenShift Container Platform 4.3

## Service Mesh

Service Mesh installation, usage, and release notes



# OpenShift Container Platform 4.3 Service Mesh

---

Service Mesh installation, usage, and release notes

## Legal Notice

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

This document provides information on how to use Service Mesh in OpenShift Container Platform

# Table of Contents

<b>CHAPTER 1. SERVICE MESH RELEASE NOTES</b>	<b>5</b>
1.1. RED HAT OPENSIFT SERVICE MESH OVERVIEW	5
1.2. GETTING SUPPORT	5
1.3. RED HAT OPENSIFT SERVICE MESH SUPPORTED CONFIGURATIONS	5
1.3.1. Supported configurations for Kiali on Red Hat OpenShift Service Mesh	6
1.3.2. Supported Mixer adapters	6
1.3.3. New features Red Hat OpenShift Service Mesh 1.0.4	6
1.3.4. New features Red Hat OpenShift Service Mesh 1.0.3	6
1.3.5. New features Red Hat OpenShift Service Mesh 1.0.2	6
1.3.6. New features Red Hat OpenShift Service Mesh 1.0.1	7
1.3.7. New features Red Hat OpenShift Service Mesh 1.0	7
1.4. KNOWN ISSUES	7
1.4.1. Red Hat OpenShift Service Mesh known issues	7
1.4.2. Kiali known issues	8
1.5. FIXED ISSUES	8
1.5.1. Red Hat OpenShift Service Mesh fixed issues	8
1.5.2. Kiali fixed issues	9
<b>CHAPTER 2. SERVICE MESH ARCHITECTURE</b>	<b>10</b>
2.1. UNDERSTANDING RED HAT OPENSIFT SERVICE MESH	10
2.1.1. Understanding service mesh	10
2.1.2. Red Hat OpenShift Service Mesh Architecture	10
2.1.3. Red Hat OpenShift Service Mesh control plane	11
2.1.4. Multi-tenancy in Red Hat OpenShift Service Mesh versus cluster-wide installations	11
2.1.5. Automatic injection	12
2.1.6. Istio Role Based Access Control features	12
2.1.7. OpenSSL	13
2.1.8. The Istio Container Network Interface (CNI) plug-in	13
2.2. KIALI OVERVIEW	13
2.2.1. Kiali overview	13
2.2.2. Kiali architecture	13
2.2.3. Kiali features	14
2.3. UNDERSTANDING JAEGER	15
2.3.1. Jaeger overview	15
2.3.2. Jaeger architecture	15
2.3.3. Jaeger features	16
2.4. COMPARING SERVICE MESH AND ISTIO	16
2.4.1. Red Hat OpenShift Service Mesh control plane	16
2.4.2. Multi-tenancy in Red Hat OpenShift Service Mesh versus cluster-wide installations	16
2.4.3. Automatic injection	17
2.4.4. Istio Role Based Access Control features	17
2.4.5. OpenSSL	18
2.4.6. The Istio Container Network Interface (CNI) plug-in	18
2.4.7. Kiali and service mesh	18
2.4.8. Jaeger and service mesh	19
<b>CHAPTER 3. SERVICE MESH INSTALLATION</b>	<b>20</b>
3.1. PREPARING TO INSTALL RED HAT OPENSIFT SERVICE MESH	20
3.1.1. Red Hat OpenShift Service Mesh supported configurations	20
3.1.1.1. Supported configurations for Kiali on Red Hat OpenShift Service Mesh	20
3.1.1.2. Supported Mixer adapters	21

3.1.2. Red Hat OpenShift Service Mesh installation activities	21
3.2. INSTALLING RED HAT OPENSIFT SERVICE MESH	21
3.2.1. Installing the Operators from OperatorHub	22
3.2.1.1. Installing the Elasticsearch Operator	22
3.2.1.2. Installing the Jaeger Operator	23
3.2.1.3. Installing the Kiali Operator	24
3.2.1.4. Installing the Red Hat OpenShift Service Mesh Operator	25
3.2.1.5. Deploying the Red Hat OpenShift Service Mesh control plane	26
3.2.1.5.1. Deploying the control plane from the web console	26
3.2.1.5.2. Deploying the control plane from the CLI	27
3.2.1.6. Creating the Red Hat OpenShift Service Mesh member roll	28
3.2.1.6.1. Creating the member roll from the web console	28
3.2.1.6.2. Creating the member roll from the CLI	29
3.2.1.7. Adding or removing projects from the service mesh	30
3.2.1.7.1. Modifying the member roll from the web console	30
3.2.1.7.2. Modifying the member roll from the CLI	31
3.2.1.8. Deleting the Red Hat OpenShift Service Mesh member roll	31
3.2.2. Updating your application pods	32
3.3. CUSTOMIZING THE RED HAT OPENSIFT SERVICE MESH INSTALLATION	32
3.3.1. Red Hat OpenShift Service Mesh custom resources	32
3.3.2. ServiceMeshControlPlane parameters	34
3.3.2.1. Istio global example	34
3.3.2.2. Istio gateway configuration	36
3.3.2.3. Istio Mixer configuration	37
3.3.2.4. Istio Pilot configuration	39
3.3.3. Configuring Kiali	40
3.3.3.1. Configuring Kiali for Grafana	41
3.3.3.2. Configuring Kiali for Jaeger	41
3.3.4. Configuring Jaeger	41
3.3.4.1. Configuring Elasticsearch	42
3.3.5. 3scale configuration	45
3.4. UPDATING RED HAT OPENSIFT SERVICE MESH FROM VERSION 1.0.1 TO 1.0.2	47
3.5. REMOVING RED HAT OPENSIFT SERVICE MESH	48
3.5.1. Removing the Red Hat OpenShift Service Mesh control plane	48
3.5.1.1. Removing the control plane with the web console	48
3.5.1.2. Removing the control plane from the CLI	48
3.5.2. Removing the installed Operators	49
3.5.2.1. Removing the Red Hat OpenShift Service Mesh Operator	49
3.5.2.2. Removing the Jaeger Operator	50
3.5.2.3. Removing the Kiali Operator	50
3.5.2.4. Removing the Elasticsearch Operator	51
3.5.2.5. Clean up Operator resources	51
<b>CHAPTER 4. DAY TWO</b>	<b>52</b>
4.1. DEPLOYING APPLICATIONS ON RED HAT OPENSIFT SERVICE MESH	52
4.1.1. Creating control plane templates	52
4.1.1.1. Creating the ConfigMap	52
4.1.2. Red Hat OpenShift Service Mesh's sidecar injection	53
4.1.2.1. Enabling automatic sidecar injection	54
4.1.3. Updating Mixer policy enforcement	54
4.2. CONFIGURING YOUR SERVICE MESH FOR DISTRIBUTED TRACING	55
4.2.1. Configuring the Elasticsearch index cleaner job	55
4.3. EXAMPLE APPLICATION	56

4.3.1. Bookinfo application	56
4.3.2. Installing the Bookinfo application	57
4.3.3. Adding default destination rules	58
4.3.4. Verifying the Bookinfo installation	59
4.3.5. Removing the Bookinfo application	59
4.3.5.1. Delete the Bookinfo project	59
4.3.5.2. Remove the Bookinfo project from the Service Mesh member roll	60
4.4. KIALI TUTORIAL	60
4.4.1. Accessing the Kiali console	61
4.4.2. Exploring the Graph page	62
4.4.3. Exploring the Applications page	63
4.4.4. Exploring the Workloads page	64
4.4.5. Exploring the Services page	64
4.4.6. Exploring the Istio Config page	65
4.5. DISTRIBUTED TRACING TUTORIAL	66
4.5.1. Generating traces and analyzing trace data	66
<b>CHAPTER 5. 3SCALE ADAPTER</b>	<b>69</b>
5.1. USING THE 3SCALE ISTIO ADAPTER	69
5.1.1. Integrate the 3scale adapter with Red Hat OpenShift Service Mesh	69
5.1.1.1. Generating 3scale custom resources	70
5.1.1.1.1. Generate templates from URL examples	71
5.1.1.2. Generating manifests from a deployed adapter	71
5.1.1.3. Routing service traffic through the adapter	72
5.1.2. Configure the integration settings in 3scale	72
5.1.3. Caching behavior	73
5.1.4. Authenticating requests	73
5.1.4.1. Applying authentication patterns	73
5.1.4.1.1. API key authentication method	73
5.1.4.1.2. Application ID and application key pair authentication method	74
5.1.4.1.3. OpenID authentication method	74
5.1.4.1.4. Hybrid authentication method	75
5.1.5. 3scale Adapter metrics	76





# CHAPTER 1. SERVICE MESH RELEASE NOTES

## 1.1. RED HAT OPENSIFT SERVICE MESH OVERVIEW

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.

## 1.2. 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.

## 1.3. RED HAT OPENSIFT SERVICE MESH SUPPORTED CONFIGURATIONS

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

- Red Hat OpenShift Container Platform version 4.x.



### NOTE

OpenShift Online and OpenShift Dedicated are not supported for Red Hat OpenShift Service Mesh 1.0.4.

- 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.
- 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.
- This release only supports configurations that do not integrate external services such as virtual machines.

### 1.3.1. Supported configurations for Kiali on Red Hat OpenShift Service Mesh

- The Kiali observability console is only supported on the two most recent releases of the Chrome, Edge, Firefox, or Safari browsers.

### 1.3.2. Supported Mixer adapters

- This release only supports the following Mixer adapter:
  - 3scale Istio Adapter

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.

### 1.3.3. New features Red Hat OpenShift Service Mesh 1.0.4

This release of Red Hat OpenShift Service Mesh adds support for Kiali 1.0.9, and addresses Common Vulnerabilities and Exposures (CVEs).

### 1.3.4. New features Red Hat OpenShift Service Mesh 1.0.3

This release of Red Hat OpenShift Service Mesh adds support for Kiali 1.0.8, and addresses Common Vulnerabilities and Exposures ([CVEs](#)).

### 1.3.5. New features Red Hat OpenShift Service Mesh 1.0.2

This release of Red Hat OpenShift Service Mesh adds support for Istio 1.1.17, Jaeger 1.13.1, Kiali 1.0.7, and the 3scale Istio Adapter 1.0 and OpenShift Container Platform 4.2.

### 1.3.6. New features Red Hat OpenShift Service Mesh 1.0.1

This release of Red Hat OpenShift Service Mesh adds support for Istio 1.1.11, Jaeger 1.13.1, Kiali 1.0.6, and the 3scale Istio Adapter 1.0 and OpenShift Container Platform 4.1.

### 1.3.7. New features Red Hat OpenShift Service Mesh 1.0

This release of Red Hat OpenShift Service Mesh adds support for Istio 1.1.11, Jaeger 1.13.1, Kiali 1.0.5, and the 3scale Istio Adapter 1.0 and OpenShift Container Platform 4.1.

Other notable changes in this release include the following:

- The Kubernetes Container Network Interface (CNI) plug-in is always on.
- The control plane is configured for multitenancy by default. Single tenant, cluster-wide control plane configurations are deprecated.
- The Elasticsearch, Jaeger, Kiali, and Service Mesh Operators are installed from OperatorHub.
- You can create and specify control plane templates.
- Automatic route creation was removed from this release.

## 1.4. 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 is not supported by the upstream Istio project, nor fully supported by OpenShift.
- 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.
- Red Hat OpenShift Service Mesh does not support installation on a restricted network.



#### NOTE

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

### 1.4.1. Red Hat OpenShift Service Mesh known issues

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

- [Istio-14743](#) Due to limitations in the version of Istio that this release of Red Hat OpenShift Service Mesh is based on, there are several applications that are currently incompatible with Service Mesh. See the linked community issue for details.
- [MAISTRA-858](#) The following Envoy log messages describing [deprecated options and configurations associated with Istio 1.1.x](#) are expected:
  - [2019-06-03 07:03:28.943][19][warning][misc]  
[external/envoy/source/common/protobuf/utility.cc:129] Using deprecated option 'envoy.api.v2.listener.Filter.config'. This configuration will be removed from Envoy soon.

- [2019-08-12 22:12:59.001][13][warning][misc]  
[external/envoy/source/common/protobuf/utility.cc:174] Using deprecated option 'envoy.api.v2.Listener.use\_original\_dst' from file lds.proto. This configuration will be removed from Envoy soon.
- [MAISTRA-681](#) and [KIALI-2686](#) When the control plane has many namespaces, it can lead to performance issues.
- [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-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-806](#) Evicted Istio Operator Pod causes mesh and CNI not to deploy.  
If the **istio-operator** pod is evicted while deploying the control pane, delete the evicted **istio-operator** pod.

### 1.4.2. Kiali known issues

- [KIALI-3262](#) In the Kiali console, when you click on Distributed Tracing in the navigation or on a Traces tab, you are asked to accept the certificate, and then asked to provide your OpenShift login credentials. This happens due to an issue with how the framework displays the Trace pages in the Console. The Workaround is to open the URL for the Jaeger console in another browser window and log in. Then you can view the embedded tracing pages in the Kiali console.
- [KIALI-3239](#) If a Kiali Operator pod has failed with a status of "Evicted" it blocks the Kiali operator from deploying. The workaround is to delete the Evicted pod and redeploy the Kiali operator.
- [KIALI-3118](#) After changes to the ServiceMeshMemberRoll, for example adding or removing projects, the Kiali pod restarts and then displays errors on the Graph page while the Kiali pod is restarting.
- [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.

## 1.5. FIXED ISSUES

The following issues been resolved in the current release:

### 1.5.1. Red Hat OpenShift Service Mesh fixed issues

- [OSSM-99](#) Workloads generated from direct Pod without labels may crash Kiali.
- [OSSM-93](#) IstioConfigList can't filter by two or more names.
- [OSSM-92](#) Cancelling unsaved changes on the VS/DR YAML edit page does not cancel the changes.
- [OSSM-90](#) Traces not available on the service details page.
- [MAISTRA-1001](#) Closing HTTP/2 connections could lead to segmentation faults in **istio-proxy**.
- [MAISTRA-932](#) Added the **requires** metadata to add dependency relationship between Jaeger operator and Elasticsearch operator. Ensures that when the Jaeger operator is installed, it automatically deploys the Elasticsearch operator if it is not available.
- [MAISTRA-862](#) Galley dropped watches and stopped providing configuration to other components after many namespace deletions and re-creations.
- [MAISTRA-833](#) Pilot stopped delivering configuration after many namespace deletions and re-creations.
- [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.
- [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 does not allow service access. Kiali cannot properly communicate with the Jaeger endpoint and it considers Jaeger to be disabled. See also [TRACING-591](#).
- [MAISTRA-357](#) In OpenShift 4 Beta on AWS, it is not possible, by default, 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. Without a service running on port 80, the load balancer health check fails.
- [MAISTRA-348](#) OpenShift 4 Beta on AWS does not support ingress gateway traffic on ports other than 80 or 443. If you configure your ingress gateway to handle TCP traffic with a port number other than 80 or 443, you have to use the service hostname provided by the AWS load balancer rather than the OpenShift router as a workaround.

### 1.5.2. Kiali fixed issues

- [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-3070](#) This bug only affects custom dashboards, not the default dashboards. When you select labels in metrics settings and refresh the page, your selections are retained in the menu but your selections are not displayed on the charts.

## CHAPTER 2. SERVICE MESH ARCHITECTURE

### 2.1. UNDERSTANDING RED HAT OPENSIFT SERVICE MESH

Red Hat OpenShift Service Mesh provides a platform for behavioral insight and operational control over your networked microservices in a service mesh. With Red Hat OpenShift Service Mesh, you can connect, secure, and monitor microservices in your OpenShift Container Platform environment.

#### 2.1.1. 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

Red Hat OpenShift Service Mesh also provides more complex operational functions including:

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

#### 2.1.2. 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, and 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.

### 2.1.3. Red Hat OpenShift Service Mesh control plane

Red Hat OpenShift Service Mesh installs a multi-tenant control plane by default. You specify the projects that can access the Service Mesh, and isolate the Service Mesh from other control plane instances.

### 2.1.4. Multi-tenancy in Red Hat OpenShift Service Mesh versus 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) resource **ClusterRoleBinding**, but rely on project-scoped **RoleBinding**.

Every project 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 projects. Each member project has a **maistra.io/member-of** label added to it, where the **member-of** value is the project containing the control plane installation.

Red Hat OpenShift Service Mesh configures each member project to ensure network access between itself, the control plane, and other member projects. The exact configuration differs depending on how OpenShift software-defined networking (SDN) is configured. See About OpenShift SDN for additional details.

If the OpenShift Container Platform cluster is configured to use the SDN plug-in:

- **NetworkPolicy**: Red Hat OpenShift Service Mesh creates a **NetworkPolicy** resource in each member project allowing ingress to all pods from the other members and the control plane. If

you remove a member from Service Mesh, this **NetworkPolicy** resource is deleted from the project.



#### NOTE

This also restricts ingress to only member projects. If ingress from non-member projects is required, you need to create a **NetworkPolicy** to allow that traffic through.

- **Multitenant:** Red Hat OpenShift Service Mesh joins the **NetNamespace** for each member project to the **NetNamespace** of the control plane project (the equivalent of running `oc adm pod-network join-projects --to control-plane-project member-project`). If you remove a member from the Service Mesh, its **NetNamespace** is isolated from the control plane (the equivalent of running `oc adm pod-network isolate-projects member-project`).
- **Subnet:** No additional configuration is performed.

### 2.1.5. Automatic injection

The upstream Istio community installation automatically injects the sidecar into pods within the projects you have labeled.

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

### 2.1.6. Istio Role Based Access Control features

Istio Role Based Access Control (RBAC) provides a mechanism you can use to control access to a service. You can identify subjects by user name 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.

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.

#### 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 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>"

```

### 2.1.7. 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.

### 2.1.8. The Istio Container Network Interface (CNI) plug-in

Red Hat OpenShift Service Mesh includes CNI plug-in, which provides you with an alternate way to configure application pod networking. The CNI plug-in replaces the **init-container** network configuration eliminating the need to grant service accounts and projects access to Security Context Constraints (SCCs) with elevated privileges.

#### Next steps

- [Prepare to install Red Hat OpenShift Service Mesh](#) in your OpenShift Container Platform environment.

## 2.2. KIALI OVERVIEW

Kiali provides visibility into your service mesh by showing you the microservices in your service mesh, and how they are connected.

### 2.2.1. Kiali overview

Kiali provides observability into the Service Mesh running on OpenShift Container Platform. Kiali helps you define, validate, and observe your Istio service mesh. It helps you to understand the structure of your service mesh by inferring the topology, and also provides information about the health of your service mesh.

Kiali provides an interactive graph view of your namespace in real time that provides visibility into features like circuit breakers, request rates, latency, and even graphs of traffic flows. Kiali offers insights about components at different levels, from Applications to Services and Workloads, and can display the interactions with contextual information and charts on the selected graph node or edge. Kiali also provides the ability to validate your Istio configurations, such as gateways, destination rules, virtual services, mesh policies, and more. Kiali provides detailed metrics, and a basic Grafana integration is available for advanced queries. Distributed tracing is provided by integrating Jaeger into the Kiali console.

Kiali is installed by default as part of the Red Hat OpenShift Service Mesh.

### 2.2.2. Kiali architecture

Kiali is composed of two components: the Kiali application and the Kiali console.

- **Kiali application** (back end) – This component runs in the container application platform and communicates with the service mesh components, retrieves and processes data, and exposes this data to the console. The Kiali application does not need storage. When deploying the application to a cluster, configurations are set in ConfigMaps and secrets.
- **Kiali console** (front end) – The Kiali console is a web application. The Kiali application serves the Kiali console, which then queries the back end for data in order to present it to the user.

In addition, Kiali depends on external services and components provided by the container application platform and Istio.

- **Red Hat Service Mesh(Istio)** – Istio is a Kiali requirement. Istio is the component that provides and controls the service mesh. Although Kiali and Istio can be installed separately, Kiali depends on Istio and will not work if it is not present. Kiali needs to retrieve Istio data and configurations, which are exposed through Prometheus and the cluster API.
- **Prometheus** – A dedicated Prometheus instance is included as part of the Red Hat OpenShift Service Mesh installation. When Istio telemetry is enabled, metrics data is stored in Prometheus. Kiali uses this Prometheus data to determine the mesh topology, display metrics, calculate health, show possible problems, and so on. Kiali communicates directly with Prometheus and assumes the data schema used by Istio Telemetry. Prometheus is an Istio dependency and a hard dependency for Kiali, and many of Kiali's features will not work without Prometheus.
- **Cluster API** – Kiali uses the API of the OpenShift Container Platform (cluster API) in order to fetch and resolve service mesh configurations. Kiali queries the cluster API to retrieve, for example, definitions for namespaces, services, deployments, pods, and other entities. Kiali also makes queries to resolve relationships between the different cluster entities. The cluster API is also queried to retrieve Istio configurations like virtual services, destination rules, route rules, gateways, quotas, and so on.
- **Jaeger** – Jaeger is optional, but is installed by default as part of the Red Hat OpenShift Service Mesh installation. When you install Jaeger as part of the default Red Hat OpenShift Service Mesh installation, the Kiali console includes a tab to display Jaeger's tracing data. Note that tracing data will not be available if you disable Istio's distributed tracing feature. Also note that user must have access to the namespace where the control plane is installed in order to view Jaeger data.
- **Grafana** – Grafana is optional, but is installed by default as part of the Red Hat OpenShift Service Mesh installation. When available, the metrics pages of Kiali display links to direct the user to the same metric in Grafana. Note that user must have access to the namespace where the control plane is installed in order to view links to the Grafana dashboard and view Grafana data.

### 2.2.3. Kiali features

The Kiali console is integrated with Red Hat Service Mesh and provides the following capabilities:

- **Health** – Quickly identify issues with applications, services, or workloads.
- **Topology** – Visualize how your applications, services, or workloads communicate via the Kiali graph.
- **Metrics** – Predefined metrics dashboards let you chart service mesh and application performance for Go, Node.js, Quarkus, Spring Boot, Thorntail and Vert.x. You can also create your own custom dashboards.

- **Tracing** – Integration with Jaeger lets you follow the path of a request through various microservices that make up an application.
- **Validations** – Perform advanced validations on the most common Istio objects (Destination Rules, Service Entries, Virtual Services, and so on).
- **Configuration** – Optional ability to create, update and delete Istio routing configuration using wizards or directly in the YAML editor in the Kiali Console.

## 2.3. UNDERSTANDING JAEGER

Every time a user takes an action in an application, a request is executed by the architecture that may require dozens of different services to participate in order to produce a response. The path of this request is a distributed transaction. Jaeger lets you perform distributed tracing, which follows the path of a request through various microservices that make up an application.

**Distributed tracing** is a technique that is used to tie the information about different units of work together—usually executed in different processes or hosts—in order to understand a whole chain of events in a distributed transaction. Distributed tracing lets developers visualize call flows in large service oriented architectures. It can be invaluable in understanding serialization, parallelism, and sources of latency.

Jaeger records the execution of individual requests across the whole stack of microservices, and presents them as traces. A **trace** is a data/execution path through the system. An end-to-end trace is comprised of one or more spans.

A **span** represents a logical unit of work in Jaeger that has an operation name, the start time of the operation, and the duration. Spans may be nested and ordered to model causal relationships.

### 2.3.1. Jaeger overview

Jaeger lets service owners instrument their services to get insights into what their architecture is doing. Jaeger is an open source distributed tracing platform that you can use for monitoring, network profiling, and troubleshooting the interaction between components in modern, cloud-native, microservices-based applications. Jaeger is based on the vendor-neutral OpenTracing APIs and instrumentation.

Using Jaeger lets you perform the following functions:

- Monitor distributed transactions
- Optimize performance and latency
- Perform root cause analysis

Jaeger is installed by default as part of Red Hat OpenShift Service Mesh.

### 2.3.2. Jaeger architecture

Jaeger is made up of several components that work together to collect, store, and display tracing data.

- **Jaeger Client** (Tracer, Reporter, instrumented application, client libraries)– Jaeger clients are language specific implementations of the OpenTracing API. They can be used to instrument applications for distributed tracing either manually or with a variety of existing open source frameworks, such as Camel (Fuse), Spring Boot (RHOAR), MicroProfile (RHOAR/Thorntail), Wildfly (EAP), and many more, that are already integrated with OpenTracing.

- **Jaeger Agent** (Server Queue, Processor Workers) - The Jaeger agent is a network daemon that listens for spans sent over User Datagram Protocol (UDP), which it batches and sends to the collector. The agent is meant to be placed on the same host as the instrumented application. This is typically accomplished by having a sidecar in container environments like Kubernetes.
- **Jaeger Collector** (Queue, Workers) - Similar to the Agent, the Collector is able to receive spans and place them in an internal queue for processing. This allows the collector to return immediately to the client/agent instead of waiting for the span to make its way to the storage.
- **Storage** (Data Store) - Collectors require a persistent storage backend. Jaeger has a pluggable mechanism for span storage. Note that for this release, the only supported storage is Elasticsearch.
- **Query** (Query Service) - Query is a service that retrieves traces from storage.
- **Jaeger Console** - Jaeger provides a user interface that lets you visualize your distributed tracing data. On the Search page, you can find traces and explore details of the spans that make up an individual trace.

### 2.3.3. Jaeger features

Jaeger tracing is installed with Red Hat Service Mesh by default, and provides the following capabilities:

- Integration with Kiali - When properly configured, you can view Jaeger data from the Kiali console.
- High scalability - The Jaeger backend is designed to have no single points of failure and to scale with the business needs.
- Distributed Context Propagation - Lets you connect data from different components together to create a complete end-to-end trace.
- Backwards compatibility with Zipkin - Jaeger provides backwards compatibility with Zipkin by accepting spans in Zipkin formats (Thrift or JSON v1/v2) over HTTP.

## 2.4. COMPARING SERVICE MESH AND ISTIO

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 Container Platform.

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

### 2.4.1. Red Hat OpenShift Service Mesh control plane

Red Hat OpenShift Service Mesh installs a multi-tenant control plane by default. You specify the projects that can access the Service Mesh, and isolate the Service Mesh from other control plane instances.

### 2.4.2. Multi-tenancy in Red Hat OpenShift Service Mesh versus 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) resource **ClusterRoleBinding**, but rely on project-scoped **RoleBinding**.

Every project 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 projects. Each member project has a **maistra.io/member-of** label added to it, where the **member-of** value is the project containing the control plane installation.

Red Hat OpenShift Service Mesh configures each member project to ensure network access between itself, the control plane, and other member projects. The exact configuration differs depending on how OpenShift software-defined networking (SDN) is configured. See About OpenShift SDN for additional details.

If the OpenShift Container Platform cluster is configured to use the SDN plug-in:

- **NetworkPolicy:** Red Hat OpenShift Service Mesh creates a **NetworkPolicy** resource in each member project allowing ingress to all pods from the other members and the control plane. If you remove a member from Service Mesh, this **NetworkPolicy** resource is deleted from the project.



#### NOTE

This also restricts ingress to only member projects. If ingress from non-member projects is required, you need to create a **NetworkPolicy** to allow that traffic through.

- **Multitenant:** Red Hat OpenShift Service Mesh joins the **NetNamespace** for each member project to the **NetNamespace** of the control plane project (the equivalent of running **oc adm pod-network join-projects --to control-plane-project member-project**). If you remove a member from the Service Mesh, its **NetNamespace** is isolated from the control plane (the equivalent of running **oc adm pod-network isolate-projects member-project**).
- **Subnet:** No additional configuration is performed.

### 2.4.3. Automatic injection

The upstream Istio community installation automatically injects the sidecar into pods within the projects you have labeled.

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

### 2.4.4. Istio Role Based Access Control features

Istio Role Based Access Control (RBAC) provides a mechanism you can use to control access to a service. You can identify subjects by user name 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.

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.

## 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 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>"
```

### 2.4.5. 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.

### 2.4.6. The Istio Container Network Interface (CNI) plug-in

Red Hat OpenShift Service Mesh includes CNI plug-in, which provides you with an alternate way to configure application pod networking. The CNI plug-in replaces the **init-container** network configuration eliminating the need to grant service accounts and projects access to Security Context Constraints (SCCs) with elevated privileges.

### 2.4.7. Kiali and service mesh

Installing Kiali via the Service Mesh on OpenShift Container Platform differs from community Kiali installations in multiple ways. These modifications are sometimes necessary to resolve issues, provide additional features, or to handle differences when deploying on OpenShift Container Platform.

- Kiali has been enabled by default.
- Ingress has been enabled by default.
- Updates have been made to the Kiali ConfigMap.
- Updates have been made to the ClusterRole settings for Kiali.

- Users should not manually edit the ConfigMap or the Kiali custom resource files as those changes might be overwritten by the Service Mesh or Kiali operators. All configuration for Kiali running on Red Hat OpenShift Service Mesh is done in the **ServiceMeshControlPlane** custom resource file and there are limited configuration options. Updating the operator files should be restricted to those users with cluster-admin privileges.

### 2.4.8. Jaeger and service mesh

Installing Jaeger with the Service Mesh on OpenShift Container Platform differs from community Jaeger installations in multiple ways. These modifications are sometimes necessary to resolve issues, provide additional features, or to handle differences when deploying on OpenShift Container Platform.

- Jaeger has been enabled by default for Service Mesh.
- Ingress has been enabled by default for Service Mesh.
- The name for the Zipkin port name has changed to jaeger-collector-zipkin (from http)
- Jaeger uses Elasticsearch for storage by default.
- The community version of Istio provides a generic "tracing" route. Red Hat OpenShift Service Mesh uses a "jaeger" route that is installed by the Jaeger operator and is already protected by OAuth.
- Red Hat OpenShift Service Mesh uses a sidecar for the Envoy proxy, and Jaeger also uses a sidecar, for the Jaeger agent. These two sidecars are configured separately and should not be confused with each other. The proxy sidecar creates spans related to the pod's ingress and egress traffic. The agent sidecar receives the spans emitted by the application and sends them to the Jaeger Collector.

## CHAPTER 3. SERVICE MESH INSTALLATION

### 3.1. PREPARING TO INSTALL RED HAT OPENSIFT SERVICE MESH

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

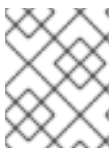
#### Prerequisites

- 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.
- Review the [OpenShift Container Platform 4.3 overview](#).
- Install OpenShift Container Platform 4.3.
  - [Install OpenShift Container Platform 4.3 on AWS](#)
  - [Install OpenShift Container Platform 4.3 on user-provisioned AWS](#)
  - [Install OpenShift Container Platform 4.3 on bare metal](#)
  - [Install OpenShift Container Platform 4.3 on vSphere](#)
- 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 4.3, see [About the CLI](#).

#### 3.1.1. Red Hat OpenShift Service Mesh supported configurations

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

- Red Hat OpenShift Container Platform version 4.x.



#### NOTE

OpenShift Online and OpenShift Dedicated are not supported for Red Hat OpenShift Service Mesh 1.0.4.

- 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.
- 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.
- This release only supports configurations that do not integrate external services such as virtual machines.

##### 3.1.1.1. Supported configurations for Kiali on Red Hat OpenShift Service Mesh



- The Kiali observability console is only supported on the two most recent releases of the Chrome, Edge, Firefox, or Safari browsers.

### 3.1.1.2. Supported Mixer adapters

- This release only supports the following Mixer adapter:
  - 3scale Istio Adapter

### 3.1.2. Red Hat OpenShift Service Mesh installation activities

To install the Red Hat OpenShift Service Mesh Operator, you must first install these Operators:



#### WARNING

Please see configuring Elasticsearch for details on configuring the default Jaeger parameters for Elasticsearch in a production environment.

- **Elasticsearch** - Based on the open source [Elasticsearch](#) project that enables you to configure and manage an Elasticsearch cluster for tracing and logging with Jaeger.
- **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, provides observability for your service mesh. By using Kiali you can view configurations, monitor traffic, and view and analyze traces in a single console.

After you install the Elasticsearch, Jaeger, and Kiali Operators, then you install the Red Hat OpenShift Service Mesh Operator. The Service Mesh Operator defines and monitors the **ServiceMeshControlPlane** resources that manage the deployment, updating, and deletion of the Service Mesh components.

- **Red Hat OpenShift Service Mesh** - based on the open source [Istio](#) project, lets you connect, secure, control, and observe the microservices that make up your applications.

#### Next steps

- [Install Red Hat OpenShift Service Mesh](#) in your OpenShift Container Platform environment.

## 3.2. INSTALLING RED HAT OPENSIFT SERVICE MESH

Installing the Service Mesh involves installing the Elasticsearch, Jaeger, Kiali and Service Mesh Operators, creating and managing a **ServiceMeshControlPlane** resource to deploy the control plane, and creating a **ServiceMeshMemberRoll** resource to specify the namespaces associated with the Service Mesh.

**NOTE**

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**

Multi-tenant control plane installations are the default configuration starting with Red Hat OpenShift Service Mesh 1.0.

**NOTE**

The Service Mesh documentation uses **istio-system** as the example project, but you may deploy the service mesh to any project.

**Prerequisites**

- Follow the [Preparing to install Red Hat OpenShift Service Mesh](#) process.
- An account with the **cluster-admin** role.

### 3.2.1. Installing the Operators from OperatorHub

The Service Mesh installation process uses the [OperatorHub](#) to install the **ServiceMeshControlPlane** custom resource definition within the **openshift-operators** project. The Red Hat OpenShift Service Mesh defines and monitors the **ServiceMeshControlPlane** related to the deployment, update, and deletion of the control plane.

Starting with Red Hat OpenShift Service Mesh 1.0.4, you must install the Elasticsearch Operator, the Jaeger Operator, and the Kiali Operator before the Red Hat OpenShift Service Mesh Operator can install the control plane.

#### 3.2.1.1. Installing the Elasticsearch Operator

You must install the Elasticsearch Operator for the Red Hat OpenShift Service Mesh Operator to install the control plane.

**WARNING**

Do not install Community versions of the Operators. Community Operators are not supported.

**Prerequisites**

- Access to the OpenShift Container Platform web console.

**Procedure**

1. Log in to the OpenShift Container Platform web console.

2. Navigate to **Operators → OperatorHub**.
3. Type **Elasticsearch** into the filter box to locate the Elasticsearch Operator.
4. Click the **Elasticsearch Operator** to display information about the Operator.
5. Click **Install**.
6. On the **Create Operator Subscription** page, select **All namespaces on the cluster (default)**  
This installs the Operator in the default **openshift-operators** project and makes the Operator available to all projects in the cluster.
7. Select the **preview** Update Channel.
8. Select the **Automatic** Approval Strategy.

**NOTE**

The Manual approval strategy requires a user with appropriate credentials to approve the Operator install and subscription process.

9. Click **Subscribe**.
10. The **Subscription Overview** page displays the Elasticsearch Operator's installation progress.

### 3.2.1.2. Installing the Jaeger Operator

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

**WARNING**

Do not install Community versions of the Operators. Community Operators are not supported.

#### Prerequisites

- Access to the OpenShift Container Platform web console.
- The Elasticsearch Operator must be installed.

#### Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Operators → OperatorHub**.
3. Type **Jaeger** into the filter box to locate the Jaeger Operator.
4. Click the **Jaeger Operator** provided by Red Hat to display information about the Operator.

5. Click **Install**.
6. On the **Create Operator Subscription** page, select **All namespaces on the cluster (default)**  
This installs the Operator in the default **openshift-operators** project and makes the Operator available to all projects in the cluster.
7. Select the **stable** Update Channel.
8. Select the **Automatic** Approval Strategy.

**NOTE**

The Manual approval strategy requires a user with appropriate credentials to approve the Operator install and subscription process.

9. Click **Subscribe**.
10. The **Subscription Overview** page displays the Jaeger Operator's installation progress.

### 3.2.1.3. Installing the Kiali Operator

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

**WARNING**

Do not install Community versions of the Operators. Community Operators are not supported.

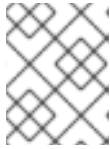
#### Prerequisites

- Access to the OpenShift Container Platform web console.

#### Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Operators → OperatorHub**.
3. Type **Kiali** into the filter box to find the Kiali Operator.
4. Click the **Kiali Operator** provided by Red Hat to display information about the Operator.
5. Click **Install**.
6. On the **Create Operator Subscription** page, select **All namespaces on the cluster (default)**  
This installs the Operator in the default **openshift-operators** project and makes the Operator available to all projects in the cluster.
7. Select the **stable** Update Channel.

8. Select the **Automatic** Approval Strategy.



#### NOTE

The Manual approval strategy requires a user with appropriate credentials to approve the Operator install and subscription process.

9. Click **Subscribe**.
10. The **Subscription Overview** page displays the Kiali Operator's installation progress.

### 3.2.1.4. Installing the Red Hat OpenShift Service Mesh Operator

#### Prerequisites

- Access to the OpenShift Container Platform web console.
- The Elasticsearch Operator must be installed.
- The Jaeger Operator must be installed.
- The Kiali Operator must be installed.

#### Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Operators → OperatorHub**.
3. Type **Red Hat OpenShift Service Mesh** into the filter box to find the Red Hat OpenShift Service Mesh Operator.
4. Click the Red Hat OpenShift Service Mesh Operator to display information about the Operator.
5. On the **Create Operator Subscription** page, select **All namespaces on the cluster (default)**. This installs the Operator in the default **openshift-operators** project and makes the Operator available to all projects in the cluster.
6. Click **Install**.
7. Select the **1.0** Update Channel.
8. Select the **Automatic** Approval Strategy.



#### NOTE

The Manual approval strategy requires a user with appropriate credentials to approve the Operator install and subscription process.

9. Click **Subscribe**.
10. The **Subscription Overview** page displays the Red Hat OpenShift Service Mesh Operator's installation progress.

### 3.2.1.5. Deploying the Red Hat OpenShift Service Mesh control plane

The **ServiceMeshControlPlane** resource defines the configuration to be used during installation. You can deploy the default configuration provided by Red Hat or customize the **ServiceMeshControlPlane** file to fit your business needs.

You can deploy the Service Mesh control plane by using the OpenShift Container Platform web console or from the command line using the **oc** client tool.

#### 3.2.1.5.1. Deploying the control plane from the web console

Follow this procedure to deploy the Red Hat OpenShift Service Mesh control plane by using the web console.

#### Prerequisites

- The Red Hat OpenShift Service Mesh Operator must be installed.
- Review the instructions for how to customize the Red Hat OpenShift Service Mesh installation.
- An account with the **cluster-admin** role.

#### Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.
2. Create a new project named **istio-system**.
  - a. Navigate to **Home → Projects**.
  - b. Click **Create Project**.
  - c. Enter **istio-system** in the **Name** field.
  - d. Click **Create**.
3. Navigate to **Catalogs → Installed Operators**.
4. If necessary, select **istio-system** from the Project menu. You may have to wait a few moments for the Operators to be copied to the new project.
5. Click the Red Hat OpenShift Service Mesh Operator. Under **Provided APIs**, the Operator provides links to create two resource types:
  - A **ServiceMeshControlPlane** resource
  - A **ServiceMeshMemberRoll** resource
6. Under **Istio Service Mesh Control Plane** click **Create New**.
7. On the **Create Service Mesh Control Plane** page, modify the YAML for the default **ServiceMeshControlPlane** template as needed.



## NOTE

For additional information about customizing the control plane, see customizing the Red Hat OpenShift Service Mesh installation. Note that for production use you *must* change the default Jaeger template.

8. Click **Create** to create the control plane. The Operator creates Pods, services, and Service Mesh control plane components based on your configuration parameters.
9. Click the **Istio Service Mesh Control Plane** tab.
10. Click the name of the new control plane.
11. Click the **Resources** tab to see the Red Hat OpenShift Service Mesh control plane resources the Operator created and configured.

### 3.2.1.5.2. Deploying the control plane from the CLI

Follow this procedure to deploy the Red Hat OpenShift Service Mesh control plane the command line.

#### Prerequisites

- The Red Hat OpenShift Service Mesh Operator must be installed.
- Review the instructions for how to customize the Red Hat OpenShift Service Mesh installation.
- An account with the **cluster-admin** role.
- Access to the OpenShift Container Platform Command-line Interface (CLI), commonly known as **oc**.

#### Procedure

1. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role.

```
$ oc login https://{HOSTNAME}:8443
```

2. Create a new project named **istio-system**.

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

3. Create a **ServiceMeshControlPlane** file named **istio-installation.yaml** using the full example found in "Customize the Red Hat OpenShift Service Mesh installation". You can customize the values as needed to match your use case. Note that for production use you *must* change the default Jaeger template.

4. Run the following command to deploy the control plane:

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

5. Execute the following command to see the status of the control plane installation.

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

The installation has finished successfully when the **READY** column is true.

NAME	READY
basic-install	True

- Run the following command to watch the progress of the Pods during the installation process:

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

You should see output similar to the following:

NAME	READY	STATUS	RESTARTS	AGE
grafana-7bf5764d9d-2b2f6	2/2	Running	0	28h
istio-citadel-576b9c5bbd-z84z4	1/1	Running	0	28h
istio-egressgateway-5476bc4656-r4zdv	1/1	Running	0	28h
istio-galley-7d57b47bb7-lqdxv	1/1	Running	0	28h
istio-ingressgateway-dbb8f7f46-ct6n5	1/1	Running	0	28h
istio-pilot-546bf69578-ccg5x	2/2	Running	0	28h
istio-policy-77fd498655-7pvjw	2/2	Running	0	28h
istio-sidecar-injector-df45bd899-ctxdx	1/1	Running	0	28h
istio-telemetry-66f697d6d5-cj28l	2/2	Running	0	28h
jaeger-896945cbc-7lqrr	2/2	Running	0	11h
kiali-78d9c5b87c-snjzh	0/1	Running	0	22h
prometheus-6dff867c97-gr2n5	2/2	Running	0	28h

For a multitenant installation, Red Hat OpenShift Service Mesh supports multiple independent control planes within the cluster. You can create reusable configurations with **ServiceMeshControlPlane** templates. For more information, see [Creating control plane templates](#).

### 3.2.1.6. Creating the Red Hat OpenShift Service Mesh member roll

The **ServiceMeshMemberRoll** lists the projects belonging to the control plane. Only projects listed in the **ServiceMeshMemberRoll** are affected by the control plane. A project does not belong to a service mesh until you add it to the member roll for a particular control plane deployment.

You must create a **ServiceMeshMemberRoll** resource named **default** in the same project as the **ServiceMeshControlPlane**.



#### NOTE

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

#### 3.2.1.6.1. Creating the member roll from the web console

Follow this procedure to add one or more projects to the Service Mesh member roll by using the web console.

#### Prerequisites

- An installed, verified Red Hat OpenShift Service Mesh Operator.
- Location of the installed **ServiceMeshControlPlane**.



- List of projects to add to the service mesh.

### Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Catalogs → Installed Operators**.
3. Click the **Project** menu and choose the project where your **ServiceMeshControlPlane** is deployed from the list, for example **istio-system**.
4. Click the Red Hat OpenShift Service Mesh Operator.
5. Click the **All Instances** tab.
6. Click **Create New**, and then select **Create Istio Service Mesh Member Roll**



### NOTE

It can take a short time for the Operator to finish copying the resources, therefore you may need to refresh the screen to see the **Create Istio Service Mesh Member Roll** button.

7. On the **Create Service Mesh Member Roll** page, modify the YAML to add your projects as members. You can add any number of projects, but a project can only belong to **one ServiceMeshMemberRoll** resource.
8. Click **Create** to save the Service Mesh Member Roll.

### 3.2.1.6.2. Creating the member roll from the CLI

Follow this procedure to add a project to the **ServiceMeshMemberRoll** from the command line.

### Prerequisites

- An installed, verified Red Hat OpenShift Service Mesh Operator.
- Location of the installed **ServiceMeshControlPlane**.
- List of projects to add to the service mesh.
- Access to the OpenShift Container Platform Command-line Interface (CLI) commonly known as **oc**.

### Procedure

1. Log in to the OpenShift Container Platform CLI.

```
$ oc login
```

2. Create a **ServiceMeshMemberRoll** resource in the same project as the **ServiceMeshControlPlane** resource, in our example that is **istio-system**. The resource must be named **default**.

```
$ oc create -n istio-system -f servicemeshmemberroll-default.yaml
```

■

**Example servicemeshmemberroll-default.yaml**

```

apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
  namespace: istio-system
spec:
  members:
    # a list of projects joined into the service mesh
    - your-project-name
    - another-project-name

```

3. Modify the default YAML to add your projects as **members**. You can add any number of projects, but a project can only belong to **one ServiceMeshMemberRoll** resource.

**3.2.1.7. Adding or removing projects from the service mesh**

Follow this procedure to modify an existing Service Mesh **ServiceMeshMemberRoll** resource using the web console.

- You can add any number of projects, but a project can only belong to **one ServiceMeshMemberRoll** resource.
- The **ServiceMeshMemberRoll** resource is deleted when its corresponding **ServiceMeshControlPlane** resource is deleted.

**3.2.1.7.1. Modifying the member roll from the web console****Prerequisites**

- An installed, verified Red Hat OpenShift Service Mesh Operator.
- An existing **ServiceMeshMemberRoll** resource.
- Name of the project with the **ServiceMeshMemberRoll** resource.
- Names of the projects you want to add or remove from the mesh.

**Procedure**

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Catalogs → Installed Operators**.
3. Click the **Project** menu and choose the project where your **ServiceMeshControlPlane** is deployed from the list, for example **istio-system**.
4. Click the Red Hat OpenShift Service Mesh Operator.
5. Click the **Istio Service Mesh Member Roll** tab.
6. Click the **default** link.

7. Click the YAML tab.
8. Modify the YAML to add or remove projects as members. You can add any number of projects, but a project can only belong to **one ServiceMeshMemberRoll** resource.
9. Click **Save**.
10. Click **Reload**.

### 3.2.1.7.2. Modifying the member roll from the CLI

Follow this procedure to modify an existing Service Mesh member roll using the command line.

#### Prerequisites

- An installed, verified Red Hat OpenShift Service Mesh Operator.
- An existing **ServiceMeshMemberRoll** resource.
- Name of the project with the **ServiceMeshMemberRoll** resource.
- Names of the projects you want to add or remove from the mesh.
- Access to the OpenShift Container Platform Command-line Interface (CLI) commonly known as **oc**.

#### Procedure

1. Log in to the OpenShift Container Platform CLI.
2. Edit the **ServiceMeshMemberRoll** resource.

```
$ oc edit smmr -n <controlplane-namespace>
```

3. Modify the YAML to add or remove projects as members. You can add any number of projects, but a project can only belong to **one ServiceMeshMemberRoll** resource.

#### Example servicemeshmemberroll-default.yaml

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
  namespace: istio-system
spec:
  members:
    # a list of projects joined into the service mesh
    - your-project-name
    - another-project-name
```

### 3.2.1.8. Deleting the Red Hat OpenShift Service Mesh member roll

The **ServiceMeshMemberRoll** resource is automatically deleted when you delete the **ServiceMeshControlPlane** resource it is associated with.

### 3.2.2. Updating your application pods

If you selected the Automatic Approval Strategy when you were installing your Operators, then the Operators update the control plane automatically, but not your applications. Existing applications continue to be part of the mesh and function accordingly. The application administrator must restart applications to upgrade the sidecar.

If your deployment uses Automatic sidecar injection, you can update the pod template in the deployment by adding or modifying an annotation. Run the following command to redeploy the pods:

```
$ oc patch deployment/<deployment> -p '{"spec":{"template":{"metadata":{"annotations":{"kubectrl.kubernetes.io/restartedAt": "'date -lseconds'"}}}}}'
```

If your deployment does not use automatic sidecar injection, you must manually update the sidecars by modifying the sidecar container image specified in the deployment or pod.

#### Next steps

- [Customize the Red Hat OpenShift Service Mesh installation](#) .
- [Prepare to deploy applications](#) on Red Hat OpenShift Service Mesh.

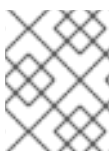
## 3.3. CUSTOMIZING THE RED HAT OPENSIFT SERVICE MESH INSTALLATION

You can customize your Red Hat OpenShift Service Mesh by modifying the default Service Mesh custom resource or by creating a new custom resource.

#### Prerequisites

- An account with the **cluster-admin** role.
- Completed the [Preparing to install Red Hat OpenShift Service Mesh](#) process.
- Have installed the operators.

### 3.3.1. Red Hat OpenShift Service Mesh custom resources



#### NOTE

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

A *custom resource* allows you to extend the API in an Red Hat OpenShift Service Mesh project or cluster. When you deploy Service Mesh it creates a default **ServiceMeshControlPlane** that you can modify to change the project parameters.

The Service Mesh operator extends the API by adding the **ServiceMeshControlPlane** resource type, which enables you to create **ServiceMeshControlPlane** objects within projects. By creating a **ServiceMeshControlPlane** object, you instruct the Operator to install a Service Mesh control plane into the project, configured with the parameters you set in the **ServiceMeshControlPlane** object.

This example **ServiceMeshControlPlane** definition contains all of the supported parameters and deploys Red Hat OpenShift Service Mesh 1.0.4 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: full-install
spec:

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

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

    mixer:
      policy:
        autoscaleEnabled: false

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

    pilot:
      autoscaleEnabled: false
      traceSampling: 100

    kiali:
      enabled: true

    grafana:
```

```

enabled: true

tracing:
  enabled: true
  jaeger:
    template: all-in-one

```

### 3.3.2. ServiceMeshControlPlane parameters

The following examples illustrate use of the **ServiceMeshControlPlane** parameters and the tables provide additional information about supported parameters.



#### IMPORTANT

The resources you configure for Red Hat OpenShift Service Mesh with these 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.

#### 3.3.2.1. Istio global example

Here is an example that illustrates the Istio global parameters for the **ServiceMeshControlPlane** and a description of the available parameters with appropriate values.



#### NOTE

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

#### Example global parameters

```

istio:
  global:
    tag: 1.0.0
    hub: registry.redhat.io/openshift-service-mesh/
    proxy:
      resources:
        requests:
          cpu: 100m
          memory: 128Mi
        limits:
          cpu: 500m
          memory: 128Mi
    mtls:
      enabled: false
    disablePolicyChecks: true
    policyCheckFailOpen: false
    imagePullSecrets:
      - MyPullSecret

```

Table 3.1. Global parameters

Parameter	Description	Values	Default value
<b>disablePolicyChecks</b>	This parameter enables/disables policy checks.	<b>true/false</b>	<b>true</b>
<b>policyCheckFailOpen</b>	This parameter indicates whether traffic is allowed to pass through to the Envoy sidecar when the Mixer policy service cannot be reached.	<b>true/false</b>	<b>false</b>
<b>tag</b>	The tag that the Operator uses to pull the Istio images.	A valid container image tag.	<b>1.0.0</b>
<b>hub</b>	The hub that the Operator uses to pull Istio images.	A valid image repository.	<b>maistra/</b> or <b>registry.redhat.io/openshift-service-mesh/</b>
<b>mtls</b>	This parameter controls whether to enable/disable Mutual Transport Layer Security (mTLS) between services by default.	<b>true/false</b>	<b>false</b>
<b>imagePullSecrets</b>	If access to the registry providing the Istio images is secure, list an <a href="#">imagePullSecret</a> here.	redhat-registry-pullsecret OR quay-pullsecret	None

These parameters are specific to the proxy subset of global parameters.

**Table 3.2. Proxy parameters**

Type	Parameter	Description	Values	Default value
Resources	<b>cpu</b>	The amount of CPU resources requested for Envoy proxy.	CPU resources, specified in cores or millicores (for example, 200m, 0.5, 1) based on your environment's configuration.	<b>100m</b>

Type	Parameter	Description	Values	Default value
	<b>memory</b>	The amount of memory requested for Envoy proxy	Available memory in bytes(for example, 200Ki, 50Mi, 5Gi) based on your environment's configuration.	<b>128Mi</b>
Limits	<b>cpu</b>	The maximum amount of CPU resources requested for Envoy proxy.	CPU resources, specified in cores or millicores (for example, 200m, 0.5, 1) based on your environment's configuration.	<b>2000m</b>
	<b>memory</b>	The maximum amount of memory Envoy proxy is permitted to use.	Available memory in bytes (for example, 200Ki, 50Mi, 5Gi) based on your environment's configuration.	<b>128Mi</b>

### 3.3.2.2. Istio gateway configuration

Here is an example that illustrates the Istio gateway parameters for the **ServiceMeshControlPlane** and a description of the available parameters with appropriate values.

#### Example gateway parameters

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

Table 3.3. Istio Gateway parameters

Type	Parameter	Description	Values	Default value
------	-----------	-------------	--------	---------------



Type	Parameter	Description	Values	Default value
istio-egressgateway	<b>autoscaleEnabled</b>	This parameter enables/disables autoscaling.	<b>true/false</b>	<b>true</b>
	<b>autoscaleMin</b>	The minimum number of pods to deploy for the egress gateway based on the <b>autoscaleEnabled</b> setting.	A valid number of allocatable pods based on your environment's configuration.	<b>1</b>
	<b>autoscaleMax</b>	The maximum number of pods to deploy for the egress gateway based on the <b>autoscaleEnabled</b> setting.	A valid number of allocatable pods based on your environment's configuration.	<b>5</b>
istio-ingressgateway	<b>autoscaleEnabled</b>	This parameter enables/disables autoscaling.	<b>true/false</b>	<b>true</b>
	<b>autoscaleMin</b>	The minimum number of pods to deploy for the ingress gateway based on the <b>autoscaleEnabled</b> setting.	A valid number of allocatable pods based on your environment's configuration.	<b>1</b>
	<b>autoscaleMax</b>	The maximum number of pods to deploy for the ingress gateway based on the <b>autoscaleEnabled</b> setting.	A valid number of allocatable pods based on your environment's configuration.	<b>5</b>

### 3.3.2.3. Istio Mixer configuration

Here is an example that illustrates the Mixer parameters for the **ServiceMeshControlPlane** and a description of the available parameters with appropriate values.

#### Example mixer parameters

```

mixer:
  enabled: true
  policy:

```

```

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

```

Table 3.4. Istio Mixer policy parameters

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

Table 3.5. Istio Mixer telemetry parameters

Type	Parameter	Description	Values	Default
Resources	<b>cpu</b>	The percentage of CPU resources requested for Mixer telemetry.	CPU resources in millicores based on your environment's configuration.	<b>100m</b>
	<b>memory</b>	The amount of memory requested for Mixer telemetry.	Available memory in bytes (for example, 200Ki, 50Mi, 5Gi) based on your environment's configuration.	<b>1G</b>

Type	Parameter	Description	Values	Default
Limits	<b>cpu</b>	The maximum percentage of CPU resources Mixer telemetry is permitted to use.	CPU resources in millicores based on your environment's configuration.	<b>500m</b>
	<b>memory</b>	The maximum amount of memory Mixer telemetry is permitted to use.	Available memory in bytes (for example, 200Ki, 50Mi, 5Gi) based on your environment's configuration.	<b>4G</b>

### 3.3.2.4. Istio Pilot configuration

Here is an example that illustrates the Istio Pilot parameters for the **ServiceMeshControlPlane** and a description of the available parameters with appropriate values.

#### Example pilot parameters

```
pilot:
  resources:
    requests:
      cpu: 100m
      memory: 128Mi
  autoscaleEnabled: false
  traceSampling: 100
```

Table 3.6. Istio Pilot parameters

Parameter	Description	Values	Default value
<b>cpu</b>	The percentage of CPU resources requested for Pilot.	CPU resources in millicores based on your environment's configuration.	<b>500m</b>
<b>memory</b>	The amount of memory requested for Pilot.	Available memory in bytes (for example, 200Ki, 50Mi, 5Gi) based on your environment's configuration.	<b>2048Mi</b>
<b>autoscaleEnabled</b>	This parameter enables/disables autoscaling. Disable this for small environments.	<b>true/false</b>	<b>true</b>

Parameter	Description	Values	Default value
<b>traceSampling</b>	This value controls how often random sampling occurs. <b>Note:</b> Increase for development or testing.	A valid percentage.	<b>100</b>

### 3.3.3. Configuring Kiali

When the Service Mesh Operator creates the **ServiceMeshControlPlane** it also processes the Kiali resource. The Kiali Operator then uses this object when creating Kiali instances.

The default Kiali parameters specified in the **ServiceMeshControlPlane** are as follows:

#### Example Kiali parameters

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  kiali:
    enabled: true
    dashboard:
      viewOnlyMode: false
    ingress:
      enabled: true
```

Table 3.7. Kiali parameters

Parameter	Description	Values	Default value
<b>enabled</b>	This parameter enables/disables Kiali. Kiali is enabled by default.	<b>true/false</b>	<b>true</b>
<b>dashboard viewOnlyMode</b>	This parameter enables/disables view-only mode for the Kiali console. When view-only mode is enabled, users cannot use the console to make changes to the Service Mesh.	<b>true/false</b>	<b>false</b>
<b>ingress enabled</b>	This parameter enables/disables ingress for Kiali.	<b>true/false</b>	<b>true</b>

### 3.3.3.1. Configuring Kiali for Grafana

When you install Kiali and Grafana as part of Red Hat OpenShift Service Mesh the Operator configures the following by default:

- Grafana is enabled as an external service for Kiali
- Grafana authorization for the Kiali console
- Grafana URL for the Kiali console

Kiali can automatically detect the Grafana URL. However if you have a custom Grafana installation that is not easily auto-detectable by Kiali, you must update the URL value in the **ServiceMeshControlPlane** resource.

#### Additional Grafana parameters

```
spec:
  kiali:
    enabled: true
    dashboard:
      viewOnlyMode: false
    grafanaURL: "https://grafana-istio-system.127.0.0.1.nip.io"
    ingress:
      enabled: true
```

### 3.3.3.2. Configuring Kiali for Jaeger

When you install Kiali and Jaeger as part of Red Hat OpenShift Service Mesh the Operator configures the following by default:

- Jaeger is enabled as an external service for Kiali
- Jaeger authorization for the Kiali console
- Jaeger URL for the Kiali console

Kiali can automatically detect the Jaeger URL. However if you have a custom Jaeger installation that is not easily auto-detectable by Kiali, you must update the URL value in the **ServiceMeshControlPlane** resource.

#### Additional Jaeger parameters

```
spec:
  kiali:
    enabled: true
    dashboard:
      viewOnlyMode: false
    jaegerURL: "http://jaeger-query-istio-system.127.0.0.1.nip.io"
    ingress:
      enabled: true
```

### 3.3.4. Configuring Jaeger

When the Service Mesh Operator creates the **ServiceMeshControlPlane** resource it also creates the Jaeger resource. The Jaeger Operator then uses this object when creating Jaeger instances.

The default Jaeger parameters specified in the **ServiceMeshControlPlane** are as follows:

### Default all-in-one Jaeger parameters

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    tracing:
      enabled: true
  jaeger:
    template: all-in-one
```

Table 3.8. Jaeger parameters

Parameter	Description	Values	Default value
tracing enabled	This parameter enables/disables tracing in Service Mesh. Jaeger is installed by default.	true/false	true
jaeger template	This parameter specifies which Jaeger deployment strategy to use.	<ul style="list-style-type: none"> <li><b>all-in-one</b> – For development, testing, demonstrations, and proof of concept.</li> <li><b>production-elasticsearch</b> – For production use.</li> </ul>	all-in-one



#### NOTE

The default template in the **ServiceMeshControlPlane** resource is the **all-in-one** deployment strategy which uses in-memory storage. For production, the only supported storage option is Elasticsearch, therefore you must configure the **ServiceMeshControlPlane** to request the **production-elasticsearch** template when you deploy Service Mesh within a production environment.

### 3.3.4.1. Configuring Elasticsearch

The default Jaeger deployment strategy uses the **all-in-one** template so that the installation can be completed using minimal resources. However, because the **all-in-one** template uses in-memory storage, it is only recommended for development, demo, or testing purposes and should NOT be used for production environments.

If you are deploying Service Mesh and Jaeger in a production environment you must change the template to the **production-elasticsearch** template, which uses Elasticsearch for Jaeger's storage needs.

Elasticsearch is a memory intensive application. The initial set of nodes specified in the default OpenShift Container Platform installation may not be large enough to support the Elasticsearch cluster. You should modify the default Elasticsearch configuration to match your use case and the resources you have requested for your OpenShift Container Platform installation. You can adjust both the CPU and memory limits for each component by modifying the resources block with valid CPU and memory values. Additional nodes must be added to the cluster if you want to run with the recommended amount (or more) of memory. Ensure that you do not exceed the resources requested for your OpenShift Container Platform installation.

### Default "production" Jaeger parameters with Elasticsearch

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    tracing:
      enabled: true
    ingress:
      enabled: true
  jaeger:
    template: production-elasticsearch
    elasticsearch:
      nodeCount: 3
      redundancyPolicy:
        resources:
          requests:
            cpu: "1"
            memory: "16Gi"
          limits:
            cpu: "1"
            memory: "16Gi"
```

Table 3.9. Elasticsearch parameters

Parameter	Description	Values	Default Value	Examples
tracing: enabled	This parameter enables/disables tracing in Service Mesh. Jaeger is installed by default.	true/false	true	
ingress: enabled	This parameter enables/disables ingress for Jaeger.	true/false	true	

Parameter	Description	Values	Default Value	Examples
<b>jaeger</b> template	This parameter specifies which Jaeger deployment strategy to use.	<b>all-in-one/production-elasticsearch</b>	<b>all-in-one</b>	
<b>elasticsearch:</b> nodeCount	Number of Elasticsearch nodes to create.	Integer value.	1	Proof of concept = 1, Minimum deployment = 3
<b>requests:</b> cpu	Number of central processing units for requests, based on your environment's configuration.	Specified in cores or millicores (for example, 200m, 0.5, 1).	1Gi	Proof of concept = 500m, Minimum deployment = 1
<b>requests:</b> memory	Available memory for requests, based on your environment's configuration.	Specified in bytes (for example, 200Ki, 50Mi, 5Gi).	500m	Proof of concept = 1Gi, Minimum deployment = 16Gi*
<b>limits:</b> cpu	Limit on number of central processing units, based on your environment's configuration.	Specified in cores or millicores (for example, 200m, 0.5, 1).		Proof of concept = 500m, Minimum deployment = 1
<b>limits:</b> memory	Available memory limit based on your environment's configuration.	Specified in bytes (for example, 200Ki, 50Mi, 5Gi).		Proof of concept = 1Gi, Minimum deployment = 16Gi*
	* Each Elasticsearch node can operate with a lower memory setting though this is <b>not</b> recommended for production deployments. For production use, you should have no less than 16Gi allocated to each Pod by default, but preferably allocate as much as you can, up to 64Gi per Pod.			

## Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.
2. Navigate to **Catalogs → Installed Operators**.
3. Click the Red Hat OpenShift Service Mesh Operator.
4. Click the **Istio Service Mesh Control Plane** tab.



5. Click the name of your control plane file, for example, **basic-install**.
6. Click the **YAML** tab.
7. Edit the Jaeger parameters, replacing the default **all-in-one** template with parameters for the **production-elasticsearch** template, modified for your use case. Ensure that the indentation is correct.
8. Click **Save**.
9. Click **Reload**. OpenShift Container Platform redeploys Jaeger and creates the Elasticsearch resources based on the specified parameters.

For more information about configuring Elasticsearch with OpenShift Container Platform, see [Configuring Elasticsearch](#).

### 3.3.5. 3scale configuration

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.

#### Example 3scale parameters

```
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 3.10. 3scale parameters

Parameter	Description	Values	Default value
<b>enabled</b>	Whether to use the 3scale adapter	<b>true/false</b>	<b>false</b>
<b>PARAM_THREESCALE_LISTEN_ADDR</b>	Sets the listen address for the gRPC server	Valid port number	<b>3333</b>
<b>PARAM_THREESCALE_LOG_LEVEL</b>	Sets the minimum log output level.	<b>debug, info, warn, error, or none</b>	<b>info</b>

Parameter	Description	Values	Default value
<b>PARAM_THREESCALE_LOG_JSON</b>	Controls whether the log is formatted as JSON	<b>true/false</b>	<b>true</b>
<b>PARAM_THREESCALE_LOG_GRPC</b>	Controls whether the log contains gRPC info	<b>true/false</b>	<b>true</b>
<b>PARAM_THREESCALE_REPORT_METRICS</b>	Controls whether 3scale system and backend metrics are collected and reported to Prometheus	<b>true/false</b>	<b>true</b>
<b>PARAM_THREESCALE_METRICS_PORT</b>	Sets the port that the 3scale <b>/metrics</b> endpoint can be scrapped from	Valid port number	<b>8080</b>
<b>PARAM_THREESCALE_CACHE_TTL_SECONDS</b>	Time period, in seconds, to wait before purging expired items from the cache	Time period in seconds	<b>300</b>
<b>PARAM_THREESCALE_CACHE_REFRESH_SECONDS</b>	Time period before expiry when cache elements are attempted to be refreshed	Time period in seconds	<b>180</b>
<b>PARAM_THREESCALE_CACHE_ENTRIES_MAX</b>	Max number of items that can be stored in the cache at any time. Set to <b>0</b> to disable caching	Valid number	<b>1000</b>
<b>PARAM_THREESCALE_CACHE_REFRESH_RETRIES</b>	The number of times unreachable hosts are retried during a cache update loop	Valid number	<b>1</b>
<b>PARAM_THREESCALE_ALLOW_INSECURE_CONN</b>	Allow to skip certificate verification when calling <b>3scale</b> APIs. Enabling this is not recommended.	<b>true/false</b>	<b>false</b>
<b>PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS</b>	Sets the number of seconds to wait before terminating requests to 3scale System and Backend	Time period in seconds	<b>10</b>

Parameter	Description	Values	Default value
<b>PARAM_THREESCALE_GRPC_CONNECTION_SECONDS</b>	Sets the maximum amount of seconds (+/- 10% jitter) a connection may exist before it is closed	Time period in seconds	60

### Next steps

- [Prepare to deploy applications](#) on Red Hat OpenShift Service Mesh.

## 3.4. UPDATING RED HAT OPENSIFT SERVICE MESH FROM VERSION 1.0.1 TO 1.0.2

Updating Red Hat OpenShift Service Mesh requires extra steps before you update OpenShift Container Platform to version 4.2. You must upgrade Red Hat OpenShift Service Mesh to 1.0.2 before upgrading OpenShift Container Platform 4.1.x to 4.2.

### Prerequisites

- Red Hat OpenShift Service Mesh version 1.0.1
- OpenShift Container Platform version 4.1

### Procedure

1. Configure existing SMCP resource requests by running the following **oc patch** command. Replace the `<smcp_namespace>` and `<smcp_name>` with your specific names:

```
$ oc patch -n <smcp_namespace> smcp <smcp_name> \
--type=merge -p \
'{"spec": {"istio": {"global": {"defaultResources": {"requests": {"cpu": "10m", "memory": "128Mi"}, "limits": {}}, "proxy": {"resources": {"requests": {"cpu": "10m", "memory": "128Mi"}, "limits": {}}, "defaultPodDisruptionBudget": {"enabled": false}}, "security": {"resources": {"requests": {"cpu": "10m", "memory": "128Mi"}}, "galley": {"resources": {"requests": {"cpu": "10m", "memory": "128Mi"}}, "pilot": {"resources": {"requests": {"cpu": "10m", "memory": "128Mi"}}, "mixer": {"telemetry": {"resources": {"requests": {"cpu": "10m", "memory": "128Mi"}}, "gateways": {"istio-egressgateway": {"resources": {"requests": {"cpu": "10m", "memory": "128Mi"}}, "istio-ingressgateway": {"resources": {"requests": {"cpu": "10m", "memory": "128Mi"}}, "prometheus": {"resources": {"requests": {"cpu": "10m", "memory": "128Mi"}}}}}'
```

- 1 For example, **basic-install**.

After running this command, wait until all SMCP Pods are replaced in the SMCP namespace.

2. After the Pods are running in the SMCP namespace, redeploy your Data Plane applications, such as **bookinfo**.

3. Log in as a **cluster-admin** user such as **kubeadmin**, and then run the following command to delete the CNI **istio-node** DaemonSet. Replace **openshift-operators** if your Red Hat OpenShift Service Mesh Operator was not installed in the default **openshift-operators** namespace:

```
$ oc delete -n openshift-operators daemonset istio-node
```

4. Upgrade Red Hat OpenShift Service Mesh Operator and SMCP to 1.0.2. After all Pods are running in the SMCP namespace, patch Data Plane applications by running the following command for each deployment:

```
$ oc patch -n <data_plane_namespace> deployment/<deployment_name> -p \
  '{"spec":{"template":{"metadata":{"annotations":{"kubectrl.kubernetes.io/restartedAt": ""`date`
  -lseconds`""}}}}}'
```

5. Upgrade OpenShift Container Platform using the OpenShift Container Platform web console.

## 3.5. REMOVING RED HAT OPENSIFT SERVICE MESH

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

### 3.5.1. Removing the Red Hat OpenShift Service Mesh control plane

You can remove the Service Mesh control plane by using the OpenShift Container Platform web console or the CLI.


#### 3.5.1.1. Removing the control plane with the web console

Follow this procedure to remove the Red Hat OpenShift Service Mesh control plane by using the web console.

##### Prerequisites

- The Red Hat OpenShift Service Mesh control plane must be deployed.

##### Procedure

1. Log in to the OpenShift Container Platform web console.
2. Click the **Project** menu and choose the **istio-system** project from the list.
3. Navigate to **Catalogs → Installed Operators**.
4. Click on **Service Mesh Control Plane** under **Provided APIs**.
5. Click the **ServiceMeshControlPlane** menu .
6. Click **Delete Service Mesh Control Plane**.
7. Click **Delete** on the confirmation dialog window to remove the **ServiceMeshControlPlane**.

#### 3.5.1.2. Removing the control plane from the CLI

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

### Prerequisites

- The Red Hat OpenShift Service Mesh control plane must be deployed.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as **oc**.



### PROCEDURE

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

### TIP

You can use the shortened **smcp** alias in place of **servicemeshcontrolplane**.

1. Log in to the OpenShift Container Platform CLI.
2. Run this command to retrieve the name of the installed **ServiceMeshControlPlane**:

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

3. 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>
```

## 3.5.2. Removing the installed 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, Kiali Operator, and the Elasticsearch Operator.

### 3.5.2.1. Removing the Red Hat OpenShift Service Mesh Operator

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

### Prerequisites

- Access to the OpenShift Container Platform web console.
- The Red Hat OpenShift Service Mesh Operator must be installed.

### Procedure

1. Log in to the OpenShift Container Platform web console.
2. From the **Operators → Installed Operators** page, scroll or type a keyword into the **Filter by name** to find the Red Hat OpenShift Service Mesh Operator. Then, click on it.
3. On the right-hand side of the **Operator Details** page, select **Uninstall Operator** from the **Actions** drop-down menu.

4. When prompted by the **Remove Operator Subscription** window, optionally select the **Also completely remove the Operator from the selected namespace** check box if you want all components related to the installation to be removed. This removes the CSV, which in turn removes the Pods, Deployments, CRDs, and CRs associated with the Operator.

### 3.5.2.2. Removing the Jaeger Operator

Follow this procedure to remove the Jaeger Operator.

#### Prerequisites

- Access to the OpenShift Container Platform web console.
- The Jaeger Operator must be installed.

#### Procedure

1. Log in to the OpenShift Container Platform web console.
2. From the **Operators → Installed Operators** page, scroll or type a keyword into the **Filter by name** to find the Jaeger Operator. Then, click on it.
3. On the right-hand side of the **Operator Details** page, select **Uninstall Operator** from the **Actions** drop-down menu.
4. When prompted by the **Remove Operator Subscription** window, optionally select the **Also completely remove the Operator from the selected namespace** check box if you want all components related to the installation to be removed. This removes the CSV, which in turn removes the Pods, Deployments, CRDs, and CRs associated with the Operator.

### 3.5.2.3. Removing the Kiali Operator

Follow this procedure to remove the Kiali Operator.

#### Prerequisites

- Access to the OpenShift Container Platform web console.
- The Kiali Operator must be installed.

#### Procedure

1. Log in to the OpenShift Container Platform web console.
2. From the **Operators → Installed Operators** page, scroll or type a keyword into the **Filter by name** to find the Kiali Operator. Then, click on it.
3. On the right-hand side of the **Operator Details** page, select **Uninstall Operator** from the **Actions** drop-down menu.
4. When prompted by the **Remove Operator Subscription** window, optionally select the **Also completely remove the Operator from the selected namespace** check box if you want all components related to the installation to be removed. This removes the CSV, which in turn removes the Pods, Deployments, CRDs, and CRs associated with the Operator.

### 3.5.2.4. Removing the Elasticsearch Operator

Follow this procedure to remove the Elasticsearch Operator.

#### Prerequisites

- Access to the OpenShift Container Platform web console.
- The Elasticsearch Operator must be installed.

#### Procedure

1. Log in to the OpenShift Container Platform web console.
2. From the **Operators → Installed Operators** page, scroll or type a keyword into the **Filter by name** to find the Elasticsearch Operator. Then, click on it.
3. On the right-hand side of the **Operator Details** page, select **Uninstall Operator** from the **Actions** drop-down menu.
4. When prompted by the **Remove Operator Subscription** window, optionally select the **Also completely remove the Operator from the selected namespace** check box if you want all components related to the installation to be removed. This removes the CSV, which in turn removes the Pods, Deployments, CRDs, and CRs associated with the Operator.

### 3.5.2.5. Clean up Operator resources

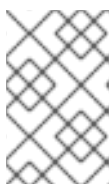
Follow this procedure to manually remove resources left behind after removing the Red Hat OpenShift Service Mesh Operator by using the OperatorHub interface.

#### Prerequisites

- An account with cluster administration access.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as **oc**.

#### Procedure

1. Log in to the OpenShift Container Platform CLI as a cluster administrator.
2. Run the following commands to clean up resources after uninstalling the Operators:



#### NOTE

Replace **<operator-project>** with the name of the project where the Red Hat OpenShift Service Mesh Operator was installed. This is typically **openshift-operators**.

```
$ oc delete validatingwebhookconfiguration/<operator-project>.servicemesh-
resources.maistra.io
$ oc delete -n <operator-project> daemonset/istio-node
$ oc delete clusterrole/istio-admin
$ oc get crds -o name | grep '.*\.istio\.io' | xargs -r -n 1 oc delete
$ oc get crds -o name | grep '.*\.maistra\.io' | xargs -r -n 1 oc delete
```

## CHAPTER 4. DAY TWO

### 4.1. DEPLOYING APPLICATIONS ON RED HAT OPENSIFT 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](#)

#### 4.1.1. Creating control plane templates

You can create reusable configurations with **ServiceMeshControlPlane** templates. Individual users can extend the templates you create with their own configurations. Templates can also inherit configuration information from other templates. For example, you can create an accounting control plane for the accounting team and a marketing control plane for the marketing team. If you create a development template and a production template, members of the marketing team and the accounting team can extend the development and production templates with team specific customization.

When you configure control plane templates, which follow the same syntax as the **ServiceMeshControlPlane**, users inherit settings in a hierarchical fashion. The Operator is delivered with a **default** template with default settings for Red Hat OpenShift Service Mesh. To add custom templates you must create a ConfigMap named **smcp-templates** in the **openshift-operators** project and mount the ConfigMap in the Operator container at **/usr/local/share/istio-operator/templates**.

##### 4.1.1.1. Creating the ConfigMap

Follow this procedure to create the ConfigMap.

#### Prerequisites

- An installed, verified Service Mesh Operator.
- An account with the **cluster-admin** role.
- Location of the Operator deployment.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as **oc**.

#### Procedure

1. Log in to the OpenShift Container Platform CLI as a cluster administrator.
2. From the CLI, run this command to create the ConfigMap named **smcp-templates** in the **openshift-operators** project and replace **<templates-directory>** with the location of the **ServiceMeshControlPlane** files on your local disk:

```
$ oc create configmap --from-file=<templates-directory> smcp-templates -n openshift-operators
```



3. Locate the Operator ClusterServiceVersion name.

```
$ oc get clusterserviceversion -n openshift-operators | grep 'Service Mesh'
maistra.v1.0.0          Red Hat OpenShift Service Mesh  1.0.0          Succeeded
```

4. Edit the Operator cluster service version to instruct the Operator to use the **smcp-templates** ConfigMap.

```
$ oc edit clusterserviceversion -n openshift-operators maistra.v1.0.0
```

5. Add a volume mount and volume to the Operator deployment.

```
deployments:
  - name: istio-operator
    spec:
      template:
        spec:
          containers:
            volumeMounts:
              - name: discovery-cache
                mountPath: /home/istio-operator/.kube/cache/discovery
              - name: smcp-templates
                mountPath: /usr/local/share/istio-operator/templates/
          volumes:
            - name: discovery-cache
              emptyDir:
                medium: Memory
            - name: smcp-templates
              configMap:
                name: smcp-templates
...

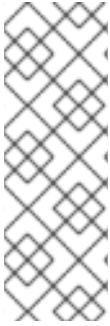
```

6. Save your changes and exit the editor.
7. You can now use the **template** parameter in the **ServiceMeshControlPlane** to specify a template.

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
metadata:
  name: minimal-install
spec:
  template: default
```

### 4.1.2. Red Hat OpenShift Service Mesh's sidecar injection

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 projects. 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 project. Red Hat OpenShift Service Mesh requires you to opt in to having the sidecar automatically injected to a deployment, so you are not required to label the project. This avoids injecting a sidecar if it is not wanted (for example, in build or deploy pods).

The webhook checks the configuration of pods deploying into all projects to see if they are opting in to injection with the appropriate annotation.

### 4.1.2.1. 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.

### 4.1.3. 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.

### Prerequisites

- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as **oc**.

### Procedure

1. Log in to the OpenShift Container Platform CLI.
2. Run this command to check the current Mixer policy enforcement status:

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

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

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

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

### Next steps

- [Deploy Bookinfo](#) on Red Hat OpenShift Service Mesh.

## 4.2. CONFIGURING YOUR SERVICE MESH FOR DISTRIBUTED TRACING

This section describes configuration that is performed in the CRD or in the CR file.

### Prerequisites

- Access to an OpenShift Container Platform cluster with cluster-admin user privileges.
- Elasticsearch operator has been installed on the cluster
- Jaeger operator has been installed on the cluster.

#### 4.2.1. Configuring the Elasticsearch index cleaner job

When the Service Mesh Operator creates the **ServiceMeshControlPlane** it also creates the custom resource (CR) for Jaeger. The Jaeger operator then uses this CR when creating Jaeger instances.

When using Elasticsearch storage, by default a job is created to clean old traces from it. To configure the options for this job, you edit the Jaeger custom resource (CR), to customize it for your use case. The relevant options are listed below.

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
```

```
spec:
  strategy: production
  storage:
    type: elasticsearch
    esIndexCleaner:
      enabled: false
      numberOfDays: 7
      schedule: "55 23 * * *"
```

Table 4.1. Elasticsearch index cleaner parameters

Parameter	Values	Description
enabled	true/ false	Enable or disable the index cleaner job.
numberOfDays	integer value	Number of days to wait before deleting an index.
schedule	"55 23 * * *"	Cron expression for the job to run

## 4.3. EXAMPLE APPLICATION



### WARNING

The Bookinfo example application allows you to test your Red Hat OpenShift Service Mesh 1.0.4 installation on OpenShift Container Platform.

Red Hat does not provide support for the Bookinfo application.

### 4.3.1. 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 these 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.

### 4.3.2. Installing the Bookinfo application

This tutorial walks you through creating a Bookinfo project, deploying the Bookinfo application, and running Bookinfo on OpenShift Container Platform with Service Mesh 1.0.4.

#### Prerequisites:

- OpenShift Container Platform 4.1 or higher installed.
- Red Hat OpenShift Service Mesh 1.0.4 installed.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as **oc**.



#### 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 for Red Hat OpenShift Service Mesh.

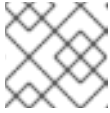
#### Procedure

1. Log in to the OpenShift Container Platform web console as a user with cluster-admin rights.
2. Click to **Home → Projects**.
3. Click **Create Project**.
4. Enter **bookinfo** as the **Project Name**, enter a **Display Name**, and enter a **Description**, then click **Create**.

- Alternatively, you can run this command from the CLI to create the **bookinfo** project.

```
$ oc new-project bookinfo
```

5. Click **Operators → Installed Operators**.
6. Click the **Project** menu and choose **bookinfo** from the list.
7. Click the **Red Hat OpenShift Service Mesh Operator**.
8. Click the **Istio Service Mesh Member Roll** link.
  - a. If you have already created a Istio Service Mesh Member Roll, click the name, then click the **YAML** tab to open the YAML editor.
  - b. If you have not created a Istio Service Mesh Member Roll, click **Create Service Mesh Member Roll**.

**NOTE**

You need cluster-admin rights to edit the Istio Service Mesh Member Roll.

9. Edit the default Service Mesh Member Roll YAML and add **bookinfo** to the **members** list.

**Bookinfo ServiceMeshMemberRoll example**

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
spec:
  members:
    - bookinfo
```

- Alternatively, you can run this command from the CLI to add the **bookinfo** project to the **ServiceMeshMemberRoll**. Replace **<control plane project>** with the name of your control plane project.

```
$ oc -n <control plane project> patch --type='json' smmr default -p '[{"op": "add", "path":
"/spec/members", "value":["bookinfo"]}]'
```

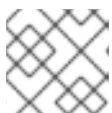
10. Click **Create** to save the updated Service Mesh Member Roll.
11. From the CLI, deploy the Bookinfo application in the `bookinfo` project by applying the **bookinfo.yaml** file:

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

12. Create the ingress gateway by applying the **bookinfo-gateway.yaml** file:

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

13. Set the value for the **GATEWAY\_URL** parameter:

**NOTE**

Replace **<control\_plane\_project>** with the name of your control plane project.

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

**4.3.3. 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 bookinfo -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 bookinfo -f https://raw.githubusercontent.com/istio/istio/release-1.1/samples/bookinfo/networking/destination-rule-all-mtls.yaml
```

#### 4.3.4. Verifying the Bookinfo installation

Before configuring your application, verify that it successfully deployed.

##### Prerequisites

- OpenShift Container Platform 4.1 or higher installed.
- Red Hat OpenShift Service Mesh 1.0.4 installed.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as **oc**.

##### Procedure

1. Log in to the OpenShift Container Platform CLI.
2. Run this command to confirm that Bookinfo is deployed:

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

- Alternatively, you can open [http://\\$GATEWAY\\_URL/productpage](http://$GATEWAY_URL/productpage) in your browser.
- You can also verify that all pods are ready with this command:

```
$ oc get pods -n bookinfo
```

#### 4.3.5. Removing the Bookinfo application

Follow these steps to remove the Bookinfo application.

##### Prerequisites

- OpenShift Container Platform 4.1 or higher installed.
- Red Hat OpenShift Service Mesh 1.0.4 installed.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as **oc**.

##### 4.3.5.1. Delete the Bookinfo project

##### Procedure

1. Log in to the OpenShift Container Platform web console.
2. Click to **Home → Projects**.

3. Click on the **bookinfo** menu , and then click **Delete Project**.

4. Type **bookinfo** in the confirmation dialog box, and then click **Delete**.


- Alternatively, you can run this command from the CLI to create the **bookinfo** project.

```
$ oc delete project bookinfo
```

#### 4.3.5.2. Remove the Bookinfo project from the Service Mesh member roll

##### Procedure

1. Log in to the OpenShift Container Platform web console.
2. Click **Operators → Installed Operators**.
3. Click the **Project** menu and choose **openshift-operators** from the list.
4. Click the **Istio Service Mesh Member Roll** link under **Provided APIS** for the **Red Hat OpenShift Service Mesh Operator**.

5. Click the **ServiceMeshMemberRoll** menu  and select **Edit Service Mesh Member Roll**.
6. Edit the default Service Mesh Member Roll YAML and remove **bookinfo** from the **members** list.

- Alternatively, you can run this command from the CLI to remove the **bookinfo** project from the **ServiceMeshMemberRoll**. Replace **<control plane project>** with the name of your control plane project.

```
$ oc -n <control plane project> patch --type='json' smmr default -p '[{"op": "remove", "path": "/spec/members", "value":["bookinfo"]}]'
```

7. Click **Save** to update Service Mesh Member Roll.

## 4.4. 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.





## NOTE

The Bookinfo example application allows you to test your Red Hat OpenShift Service Mesh 1.0.4 installation on OpenShift Container Platform.

Red Hat does not provide support for the Bookinfo application.

### 4.4.1. 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

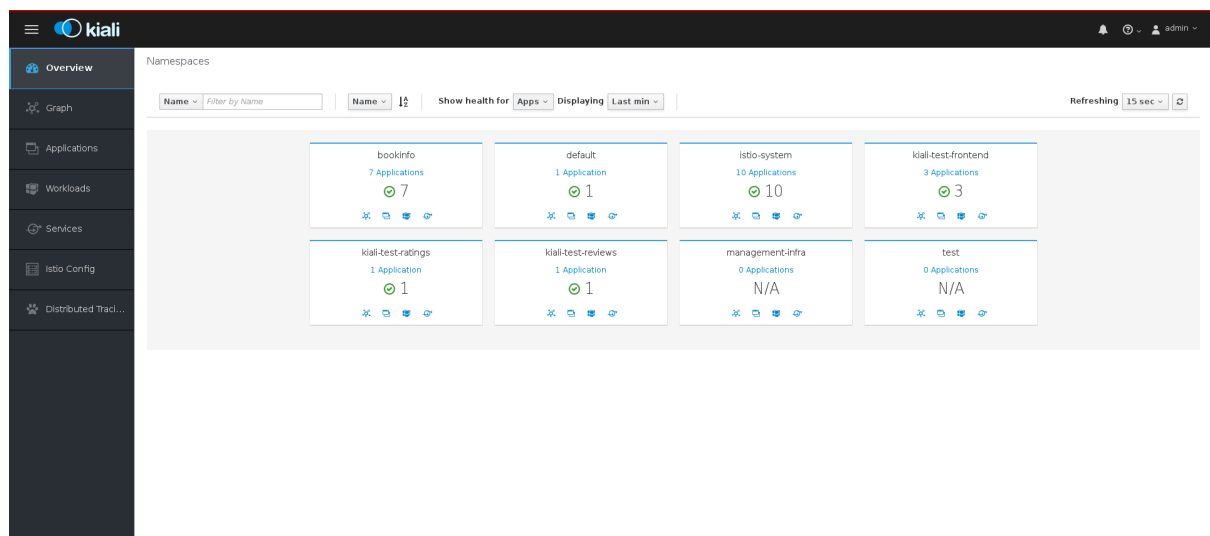
- OpenShift Container Platform 4.1 or higher installed.
- Red Hat OpenShift Service Mesh 1.0.4 installed.
- Kiali enabled during the installation.
- Bookinfo demonstration application installed.

The installation process creates a route to access the Kiali console.

#### Procedure from the console

1. In the OpenShift Container Platform console, navigate to **Networking → Routes** and search for the Kiali route.
2. Click the URL listed under **Location**.

The default login strategy is **openshift** which means that you should be automatically logged into the Kiali console using the same user name and password as you use to access the OpenShift Container Platform console. 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.



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

## Procedure from the CLI

1. Run this command from the CLI to obtain the route and Kiali URL:

```
$ oc get routes
```

### Sample CLI output showing routes

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

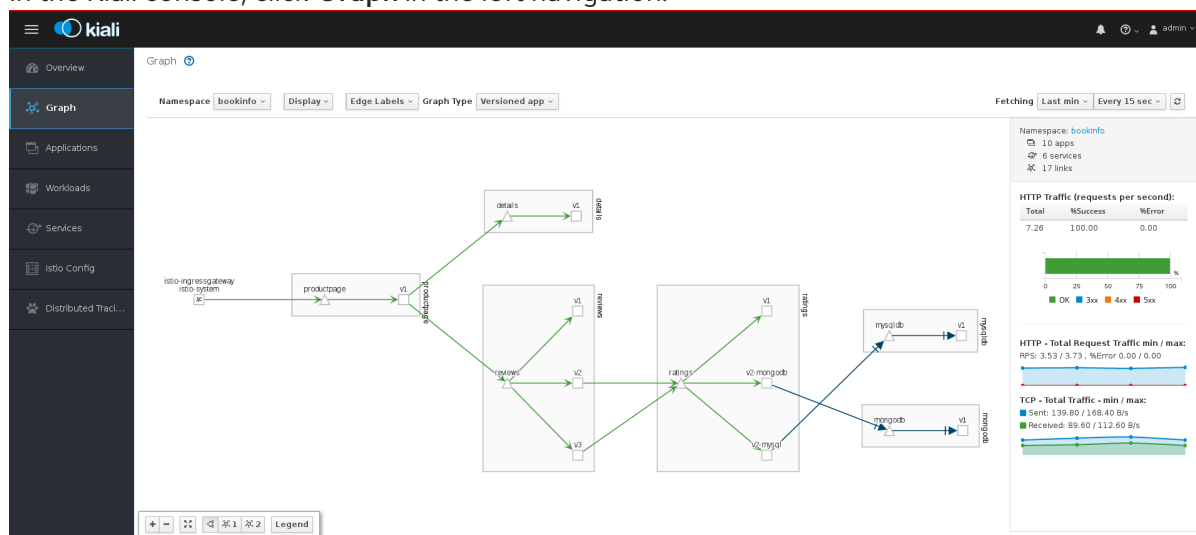
2. Launch a browser and navigate to [https://<KIALI\\_URL>](https://<KIALI_URL>) (in the CLI output example, this is **kiali-openshift-operators.127.0.0.1.nip.io**). You should see the Kiali console login screen.
3. Log in to the Kiali console using the user name and password that you use when logging in to the OpenShift Container Platform console.

## 4.4.2. Exploring the Graph page

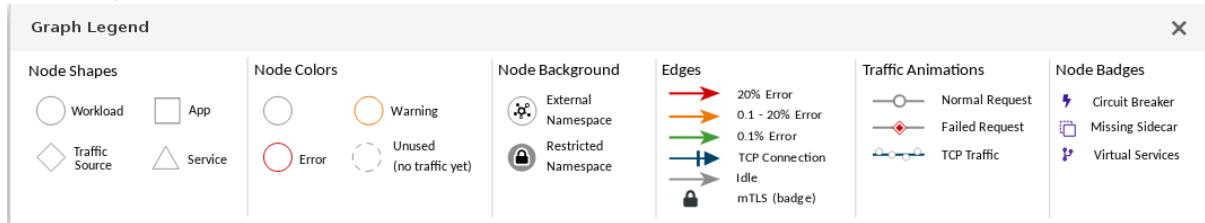
The Graph page shows a graph of microservices, which are connected by the requests going through them. On this page, you can see how Applications, Workloads, or Services interact with each other.

### Procedure

1. In the Kiali console, 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.



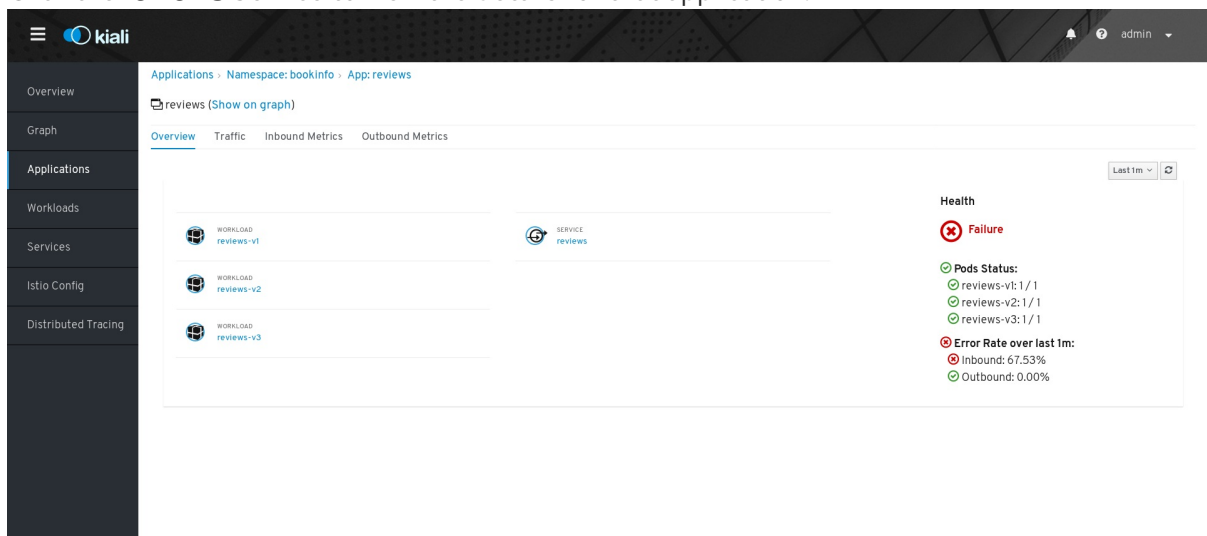
6. Close the Graph Legend.
7. Hover over the **productpage** Node. Note how the graph highlights only the incoming and outgoing traffic from the Node.
8. Click the **productpage** Node. Note how the details on the right side of the page change to display the **productpage** details.

### 4.4.3. Exploring the Applications page

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

#### Procedure

1. In the Kiali console, 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.

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

#### 4.4.4. Exploring the Workloads page

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

##### Procedure

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

Status	Name	Created at	Created by	Labels	Istio Init Containers	Istio Containers	Phase
Running	reviews-v1-6976f9d5bb-xt97t	5/15/2019, 10:35:45 AM	reviews-v1-6976f9d5bb (ReplicaSet)	app: reviews pod-template-hash: 2532958166 version: v1	quay.io/maistra/proxy-init:0.10.0-qe	quay.io/maistra/proxyv2:0.10.0-qe	Running

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

#### 4.4.5. Exploring the Services page

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

##### Procedure

- In the Kiali console, click **Services** in the left navigation.
- 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.
- 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.
- Click the **Reviews** Service to view its details. Note that there are three different versions of this Service.

Services > Namespace: bookinfo > Service: reviews > Service Info

Info Inbound Metrics Traces

reviews

Labels: app reviews  
Type ClusterIP  
IP 172.30.0.198  
Created at 1/7/2019, 9:14:35 AM  
Resource Version 11972921

Ports: TCP http (9080)

Endpoints: 10.129.1.173 : reviews-v1-855-88995-dc-fp0wf  
10.129.1.174 : reviews-v3-585-d8886f-f82pk  
10.129.1.176 : reviews-v2-6ff4f86db6-8vp5

Health:

Workloads (3) Source Workloads (1) Virtual Services (0) Destination Rules (0)

Name	Type	Labels	Created at	Resource version
reviews-v1	Deployment	app reviews version v1	1/7/2019, 9:14:36 AM	11973526
reviews-v2	Deployment	app reviews version v2	1/7/2019, 9:14:36 AM	11973510
reviews-v3	Deployment	app reviews version v3	1/7/2019, 9:14:36 AM	11973534

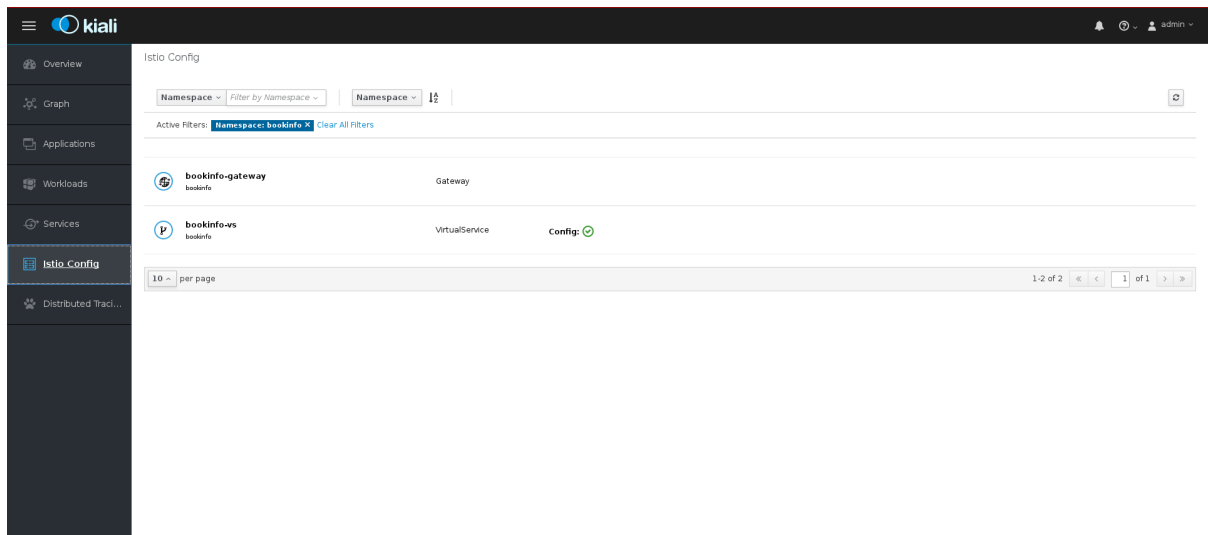
5. On the Services Details page you can view an overview of Workloads, virtual Services, and destination rules associated with the Service.
6. From the Services Details page you can also click tabs to view Traffic, Inbound Metrics, and Traces for the Service.
7. Click the Actions menu. From here you 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.

#### 4.4.6. Exploring the Istio Config page

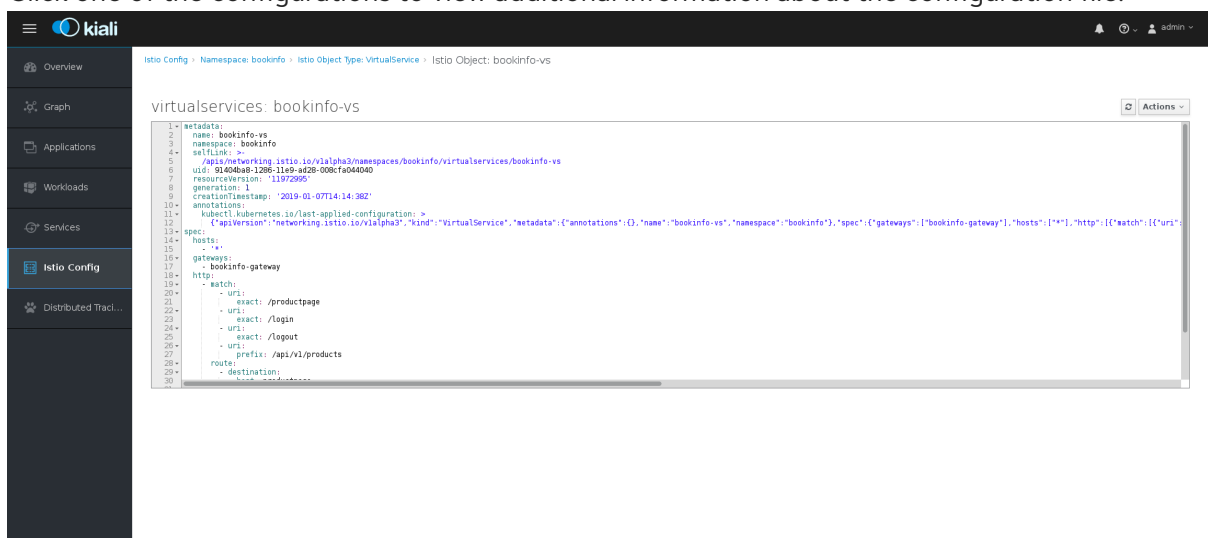
The Istio Config page lets you view all of the currently running configurations to your Service Mesh, such as Circuit Breakers, Destination Rules, Fault Injection, Gateways, Routes, Route Rules, and Virtual Services.

##### Procedure

1. In the Kiali console, 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.



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



## 4.5. 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.

This tutorial uses Service Mesh and the bookinfo tutorial to demonstrate how you can use Jaeger to perform distributed tracing.



### NOTE

The Bookinfo example application allows you to test your Red Hat OpenShift Service Mesh 1.0.4 installation on OpenShift Container Platform.

Red Hat does not provide support for the Bookinfo application.

### 4.5.1. 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:

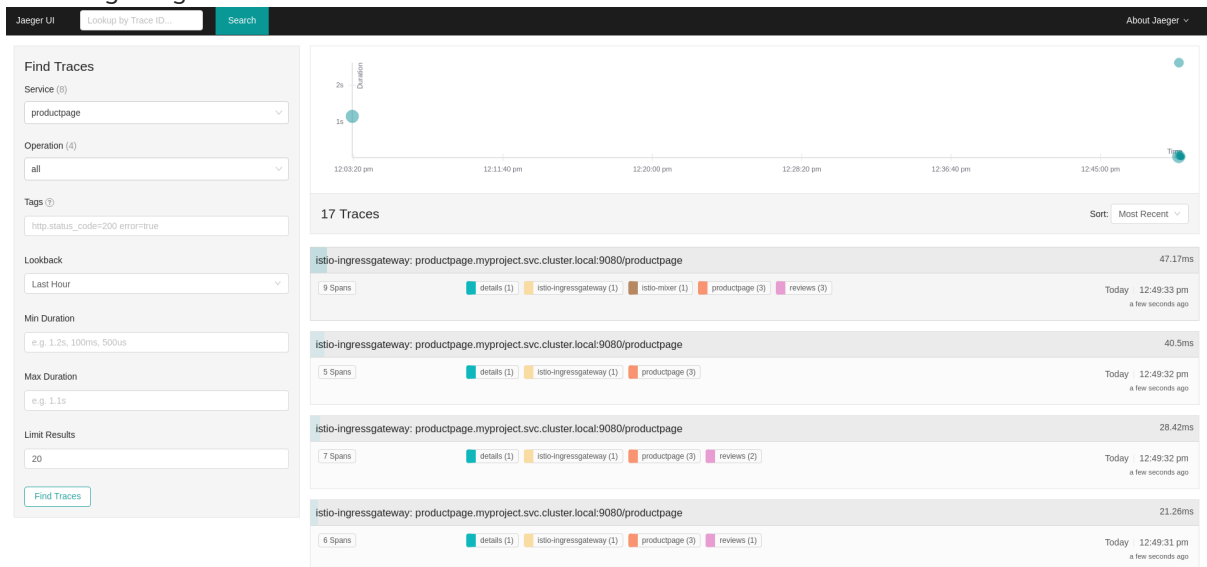
- OpenShift Container Platform 4.1 or higher installed.
- Red Hat OpenShift Service Mesh 1.0.4 installed.
- Jaeger enabled during the installation.
- Bookinfo example application installed.

## Procedure

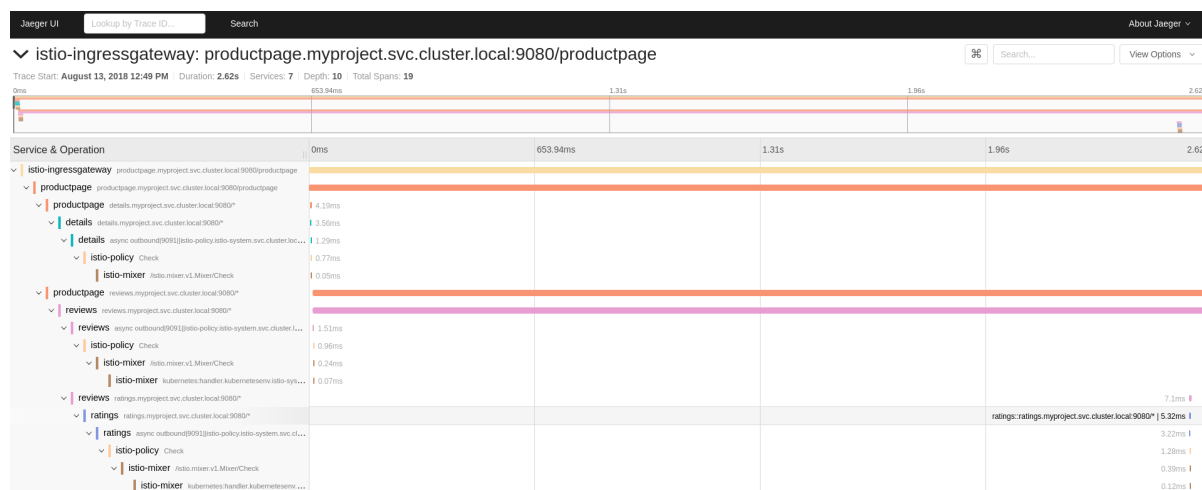
1. After you have deployed the Bookinfo application you will need to generate calls to the Bookinfo application so that you have some trace data to analyze. Access [http://<GATEWAY\\_URL>/productpage](http://<GATEWAY_URL>/productpage) and refresh the page a few times to generate some trace data.
2. The installation process creates a route to access the Jaeger console.
  - a. In the OpenShift Container Platform console, navigate to **Networking → Routes** and search for the Jaeger route, which is the URL listed under **Location**.
  - b. Use the CLI to query for details of the route:

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

3. Launch a browser and navigate to [https://<JAEGER\\_URL>](https://<JAEGER_URL>).
4. If necessary, log in using the same user name and password as you use to access the OpenShift Container Platform console.
5. 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:



6. 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.



## CHAPTER 5. 3SCALE ADAPTER

### 5.1. USING THE 3SCALE ISTIO ADAPTER

The 3scale Istio Adapter is an optional adapter that allows you to label a service running within the Red Hat OpenShift Service Mesh and integrate that service with the 3scale API Management solution. It is not required for Red Hat OpenShift Service Mesh.

#### 5.1.1. 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 section 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"
```

Optionally, you can provide a **backend\_url** field within the *params* section to override the URL provided by the 3scale configuration. This may be useful if the adapter runs on the same cluster as the 3scale on-premise instance, and you wish to leverage the internal cluster DNS.

1. 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
```

#### 5.1.1.1. Generating 3scale custom resources

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

Table 5.1. Usage

Option	Description	Required	Default value
<b>-h, --help</b>	Produces help output for available options	No	
<b>--name</b>	Unique name for this URL, token pair	Yes	
<b>-n, --namespace</b>	Namespace to generate templates	No	istio-system
<b>-t, --token</b>	3scale access token	Yes	
<b>-u, --url</b>	3scale Admin Portal URL	Yes	
<b>--backend-url</b>	3scale backend URL. If set, it overrides the value that is read from system configuration	No	
<b>-s, --service</b>	3scale API/Service ID	No	

Option	Description	Required	Default value
<b>--auth</b>	3scale authentication pattern to specify (1=Api Key, 2=App Id/App Key, 3=OIDC)	No	Hybrid
<b>-o, --output</b>	File to save produced manifests to	No	Standard output
<b>--version</b>	Outputs the CLI version and exits immediately	No	

#### 5.1.1.1. 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]"
```

#### 5.1.1.2. Generating manifests from a deployed adapter

- 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}
```

- This will produce sample output to the terminal. Edit these samples if required and create the objects using the **oc create** command.
- 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:
  - Label the workload (recommended)
  - Hard code the handler as **service\_id**
- 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}"
```

### 5.1.1.3. 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.

### 5.1.2. 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**.

### 5.1.3. 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.

### 5.1.4. Authenticating requests

This 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.

#### 5.1.4.1. 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 **User-Key**, this must be referenced in the configuration as **request.headers["user-key"]**.

##### 5.1.4.1.1. 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 **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["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"]**.

#### 5.1.4.1.2. 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["app-id"] | ""
      app_key: request.query_params["app_key"] | request.headers["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"]**.

#### 5.1.4.1.3. 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["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 **jwtUri** 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
        jwtUri: >-
          http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak/protocol/openid-
connect/certs
    principalBinding: USE_ORIGIN
  targets:
    - name: productpage

```

### 5.1.4.1.4. 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["user-key"] |
    properties:
      app_id: request.query_params["app_id"] | request.headers["app-id"] | ""
      app_key: request.query_params["app_key"] | request.headers["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"] | ""
```

### 5.1.5. 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.