



OpenShift Container Platform 3.11

Service Mesh Install

OpenShift Container Platform 3.11 Service Mesh Installation Guide

OpenShift Container Platform 3.11 Service Mesh Install

OpenShift Container Platform 3.11 Service Mesh Installation Guide

Legal Notice

Copyright © 2018 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, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

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

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

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

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

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

The OpenStack ® 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

Getting started with Service Mesh installation on OpenShift

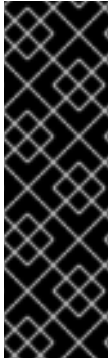
Table of Contents

CHAPTER 1. INSTALLING RED HAT OPENSIFT SERVICE MESH	3
1.1. PRODUCT OVERVIEW	3
1.1.1. Red Hat OpenShift Service Mesh overview	3
1.1.2. Red Hat OpenShift Service Mesh product architecture	3
1.1.3. Supported configurations	4
1.1.4. Red Hat OpenShift Service Mesh installation overview	4
1.2. PREREQUISITES	6
1.2.1. Red Hat OpenShift Service Mesh installation prerequisites	6
1.2.1.1. Preparing the OpenShift Container Platform installation	6
1.2.1.2. Updating the master configuration	6
1.2.1.3. Updating the node configuration	7
1.3. INSTALLING SERVICE MESH	7
1.3.1. Installing the Red Hat OpenShift Service Mesh	7
1.3.1.1. Creating a custom resource file	8
1.3.1.2. Custom resource parameters	9
1.3.1.3. Installing the operator	12
1.3.1.4. Verifying operator installation	12
1.3.1.5. Deploying the control plane	13
1.4. POST INSTALLATION TASKS	13
1.4.1. Verifying the installation	13
1.5. TUTORIALS	14
1.5.1. Bookinfo tutorial	14
1.5.1.1. Installing the Bookinfo application	15
1.5.1.2. Verifying the Bookinfo installation	16
1.5.1.3. Add default destination rules	16
1.5.1.4. Removing the Bookinfo application	16
1.5.2. Distributed tracing tutorial	17
1.5.2.1. Generating traces and analyzing trace data	17
1.5.2.2. Removing the tracing tutorial	18
1.5.3. Prometheus tutorial	18
1.5.3.1. Querying metrics	19
1.5.3.2. Removing the Prometheus tutorial	20
1.5.4. Kiali tutorial	20
1.5.4.1. Accessing the Kiali console	20
1.5.4.2. Overview page	21
1.5.4.3. Graph page	22
1.5.4.4. Services page	22
1.5.4.5. Istio config page	23
1.5.4.6. Distributed tracing page	24
1.5.4.7. Removing the Kiali tutorial	24
1.5.5. Grafana tutorial	24
1.5.5.1. Accessing the Grafana dashboard	25
1.5.5.2. Removing the Grafana tutorial	27
1.5.6. Red Hat OpenShift Application Runtime Missions	27
1.6. REMOVING RED HAT OPENSIFT SERVICE MESH	29
1.6.1. Removing Red Hat OpenShift Service Mesh	29
1.7. UPGRADING RED HAT OPENSIFT SERVICE MESH	30
1.7.1. Upgrading Red Hat OpenShift Service Mesh	30

CHAPTER 1. INSTALLING RED HAT OPENSIFT SERVICE MESH

1.1. PRODUCT OVERVIEW

1.1.1. Red Hat OpenShift Service Mesh overview



IMPORTANT

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

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

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

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

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

1.1.2. Red Hat OpenShift Service Mesh product architecture

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

- The **data plane** is composed of 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.
- The **control plane** is responsible for managing and configuring proxies to route traffic, and configuring Mixers to enforce policies and collect telemetry.

The components that make up the data plane and the control plane are:

- **Envoy proxy** is the *data plane* component that 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.
- **Mixer** is the *control plane* component responsible responsible for enforcing access control and usage policies (such as authorization, rate limits, quotas, authentication, request tracing) and collecting telemetry data from the Envoy proxy and other services.
- **Pilot** - is the *control plane* component responsible for configuring 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** - is the *control plane* component responsible for certificate issuance and rotation. 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. Using Citadel, operators can enforce policies based on service identity rather than on network controls.

1.1.3. Supported configurations

The following are the only supported configurations for the Red Hat OpenShift Service Mesh 0.3.TechPreview:

- Red Hat OpenShift Container Platform version 3.11.



NOTE

OpenShift Online and OpenShift Dedicated are not supported for Red Hat OpenShift Service Mesh 0.3.TechPreview.

- The deployment must be contained to a single OpenShift Container Platform cluster (no federation).
- This release of Red Hat OpenShift Service Mesh is only available on OpenShift Container Platform x86_64.
- Red Hat OpenShift Service Mesh is only suited OpenShift Container Platform Software Defined Networking (SDN) configured as a flat network with no external providers.
- This release supports only configurations where all service mesh components are contained in the OpenShift cluster in which it operates. It does not support management of microservices that reside outside of the cluster, or in a multi-cluster scenario.
- The Kiali observability console is only supported on the two most recent releases of the Chrome, Edge, Firefox, or Safari browsers.

For more information about support for this technology preview, see this [Red Hat Knowledge Base article](#)

1.1.4. Red Hat OpenShift Service Mesh installation overview

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

- istio-operator project (1 pod)

- `istio-system` project (17 pods)

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

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

- **Istio** - based on the open source [Istio](#) project, lets you connect, secure, control, and observe the microservices that make up your applications.
- **Jaeger** - based on the opensource [Jaeger](#) project, lets you perform tracing to monitor and troubleshoot transactions in complex distributed systems.
- **Kiali** - based on the opensource [Kiali](#) project, Kiali provides observability for your service mesh. Using Kiali lets you view configurations, monitor traffic, and view and analyze traces in a single console.
- **Launcher** - based on the opensource [fabric8](#) community, this integrated development platform helps you build cloud native applications and microservices. Red Hat OpenShift Service Mesh includes several boosters that let you explore features of the Service Mesh.

During the installation the operator creates an Ansible job that runs an Ansible playbook that performs the following installation and configuration tasks automatically:

- Creates the `istio-system` namespace
- Creates the `openshift-ansible-istio-installer-job` which installs the following components:
 - Istio components:
 - `istio-citadel`
 - `istio-egressgateway`
 - `istio-galley`
 - `istio-ingressgateway`
 - `istio-pilot`
 - `istio-policy`
 - `istio-sidecar-injector`
 - `istio-statsd-prom-bridge`
 - `istio-telemetry`
 - Elasticsearch
 - Grafana
 - Jaeger components:
 - `jaeger-agent`

- jaeger-collector
- jaeger-query
- Kiali components (if configured in the custom resource definition):
 - Kiali
- Prometheus
- Performs the following launcher configuration tasks (if configured in the custom resource definition):
 - Creates a **devex** project and installs the Fabric8 launcher into that project.
 - Adds the cluster admin role to the OpenShift Container Platform user specified in the launcher parameters in the custom resource file.

1.2. PREREQUISITES

1.2.1. Red Hat OpenShift Service Mesh installation prerequisites

Before you can install Red Hat OpenShift Service Mesh, you must meet the following prerequisites:

- Possess an active OpenShift Container Platform subscription on your Red Hat account. If you do not have a subscription, contact your sales representative for more information.
- Install OpenShift Container Platform version 3.11, or higher. For more information about the system and environment requirements, see the [OpenShift Container Platform documentation](#).
- 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. For example, if you have OpenShift Container Platform 3.11 you must have the matching **oc** client version 3.11. For installation instructions, see the OpenShift Container Platform [Command Line Reference](#) document.

1.2.1.1. Preparing the OpenShift Container Platform installation

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

1.2.1.2. Updating the master configuration



NOTE

The community version of Istio will inject the sidecar by default if you have labeled the namespace. You are not required to label the namespace with Red Hat OpenShift Service Mesh. However, Red Hat OpenShift Service Mesh requires you to opt-in to having the sidecar automatically injected to a deployment. This is to avoid injecting a sidecar where it is not wanted (for example build or deploy pods).

To enable the automatic injection of the Service Mesh sidecar you must first modify the master configuration on each master to include support for webhooks and signing of Certificate Signing Requests (CSRs).

Make the following changes on each master within your OpenShift Container Platform installation:

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

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

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

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

1.2.1.3. Updating the node configuration

To run the Elasticsearch application, you must make a change to the kernel configuration on each node. This change is handled through the **sysctl** service.

Make the following changes on each node within your OpenShift Container Platform installation:

1. Create a file named **/etc/sysctl.d/99-elasticsearch.conf** with the following contents:
vm.max_map_count = 262144
2. Execute the following command:

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

1.3. INSTALLING SERVICE MESH

1.3.1. Installing the Red Hat OpenShift Service Mesh

Installing the Service Mesh involves creating a custom resource definition file, then installing the operator to create and manage the custom resource.

1.3.1.1. Creating a custom resource file

To deploy the Service Mesh control plane, you must deploy a custom resource. A *custom resource* is an object that extends the Kubernetes API, or allows you to introduce your own API into a project or a cluster. You define a custom resource as a yaml file that defines the object. Then you use the yaml file to create the object. The following example contains all of the supported parameters and deploys Red Hat OpenShift Service Mesh 0.3.TechPreview images based on Red Hat Enterprise Linux (RHEL).

TIP

Deploying an ***istio-installation.yaml*** file that includes all of the parameters ensures that you have installed all of the Istio components that are required to complete the tutorials included in this document.

Full example istio-installation.yaml

```
apiVersion: "istio.openshift.com/v1alpha1"
kind: "Installation"
metadata:
  name: "istio-installation"
spec:
  deployment_type: openshift
  istio:
    authentication: true
    community: false
    prefix: openshift-istio-tech-preview/
    version: 0.3.0
  jaeger:
    prefix: distributed-tracing-tech-preview/
    version: 1.7.0
    elasticsearch_memory: 1Gi
  kiali:
    username: username
    password: password
    prefix: openshift-istio-tech-preview/
    version: 0.8.1
  launcher:
    openshift:
      user: user
      password: password
    github:
      username: username
      token: token
    catalog:
      filter: booster.mission.metadata.istio
      branch: v62
      repo: https://github.com/fabric8-launcher/launcher-booster-
catalog.git
```

The following example illustrates the minimum required to install the control plane. This minimal example custom resource deploys the CentOS-based community Istio images.

Minimum example istio-installation.yaml

```
apiVersion: "istio.openshift.com/v1alpha1"
kind: "Installation"
```

```
metadata:
  name: "istio-installation"
```

1.3.1.2. Custom resource parameters

The following tables list the supported custom resource parameters for Red Hat OpenShift Service Mesh.

Table 1.1. General parameters

Parameter	Values	Description	Default
deployment_type	origin, openshift	Specifies whether to use Origin (community) or OpenShift Container Platform (product) default values for undefined parameter values.	origin

Table 1.2. Istio parameters

Parameter	Values	Description	Default
authentication	true/false	Whether to enable mutual authentication.	false
community	true/false	Whether to modify image names to match community images.	false
prefix	Any valid image repo	Which prefix to apply to Istio image names that are used in docker pull commands.	If deployment_type=origin the default value is maistra/ . If deployment_type=openshift the default value is openshift-istio-tech-preview/ .
version	Any valid Docker tag	Docker tag to use with Istio images.	0.3.0

Table 1.3. Jaeger parameters

Parameter	Values	Description	Default
-----------	--------	-------------	---------

Parameter	Values	Description	Default
prefix	Any valid image repo.	Which prefix to apply to Jaeger image names used in docker pull.	<p>If deployment_type=origin the default value is jaegertracing/.</p> <p>If deployment_type=openshift the default value is distributed-tracing-tech-preview/.</p>
version	Any valid Docker tag.	Which Docker tag to use with Jaeger images.	<p>The default value is 1.7 if deployment_type=origin.</p> <p>The default value is 1.7.0 if deployment_type=openshift.</p>
elasticsearch_memory	Memory size in megabytes or gigabytes.	The amount of memory to allocate to the Elasticsearch installation, for example, 1000MB or 1 GB .	1Gi

Table 1.4. Kiali parameters

Parameter	Values	Description	Default
username	valid user	The user name to use to access the Kiali console. Note that this is not related to any account on OpenShift Container Platform.	N/A
password	valid password	The password to use to access the Kiali console. Note that this is not related to any account on OpenShift Container Platform.	N/A

Parameter	Values	Description	Default
prefix	valid image repository	Which prefix to apply to the Kiali image names used in docker pull commands.	<p>If deployment_type=origin the default value is kiali/.</p> <p>If deployment_type=openshift the default value is openshift-istio-tech-preview/.</p>
version	valid Kiali tag	Which Docker tag to use with Kiali images.	<p>The default value is v0.8.1 if deployment_type=origin.</p> <p>The default value is 0.8.1 if deployment_type=openshift.</p>

Table 1.5. Launcher parameters

Component	Parameter	Description	Default
openshift	user	The OpenShift Container Platform user that you want to run the Fabric8 launcher.	developer
	password	The OpenShift Container Platform user password to run the Fabric8 launcher.	developer
github	username	Should be modified to reflect the GitHub account you want to use to run the Fabric8 launcher.	N/A
	token	GitHub personal access token you want to use to run the Fabric8 launcher.	N/A
catalog	filter	Filter to apply to the Red Hat booster catalog.	booster.mission.metadata.istio

Component	Parameter	Description	Default
	branch	Version of the Red Hat booster catalog that should be used with Fabric8.	v62
	repo	GitHub repository to use for Red Hat booster catalog.	https://github.com/fabric8-launcher/launcher-booster-catalog.git

1.3.1.3. Installing the operator

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

You can find the [operator templates on GitHub](#).

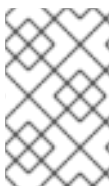


NOTE

You **must** name the custom resource **istio-installation**, that is, the metadata value for **name** must be **istio-installation** and you **must** install it into the **istio-operator** namespace that is created by the operator.

The following commands install the Service Mesh operator into an existing OpenShift Container Platform installation; you can run them from any host with access to the cluster. Ensure that you are logged in as a cluster admin before executing these commands.

```
$ oc new-project istio-operator
$ oc new-app -f istio_product_operator_template.yaml --
  param=OPENSHIFT_ISTIO_MASTER_PUBLIC_URL=<master public url>
```



NOTE

The OpenShift Master Public URL must be configured to match the public URL of your OpenShift Container Platform Console, this parameter is required by the Fabric8 Launcher.

1.3.1.4. Verifying operator installation

The previous commands create a new deployment within the **istio-operator** project and run the operator responsible for managing the state of the Red Hat OpenShift Service Mesh control plane through the custom resource.

1. To verify that the operator is installed correctly, access the logs from the operator pod by running the following command:


```
$ oc logs -n istio-operator $(oc -n istio-operator get pods -l
name=istio-operator --output=jsonpath={.items..metadata.name})
```

While your exact environment may be different from the example, you should see output that looks similar to the following example:

```
time="2018-08-31T17:42:39Z" level=info msg="Go Version: go1.9.4"
time="2018-08-31T17:42:39Z" level=info msg="Go OS/Arch: linux/amd64"
time="2018-08-31T17:42:39Z" level=info msg="operator-sdk Version:
0.0.5+git"
time="2018-08-31T17:42:39Z" level=info msg="Metrics service istio-
operator created"
time="2018-08-31T17:42:39Z" level=info msg="Watching resource
istio.openshift.com/v1alpha1, kind Installation, namespace istio-
operator, resyncPeriod 0"
time="2018-08-31T17:42:39Z" level=info msg="Installing istio for
Installation istio-installation"
```

1.3.1.5. Deploying the control plane

You use the custom resource definition file that you created to deploy the Service Mesh control plane. To deploy the control plane, run the following command:

```
$ oc create -f cr.yaml -n istio-operator
```

The operator creates the **istio-system** namespace and runs the installer job; this job installs and configures the control plane using Ansible playbooks. You can follow the progress of the installation by either watching the pods or the log output from the **openshift-ansible-istio-installer-job** pod.

To watch the progress of the pods, run the following command:

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

1.4. POST INSTALLATION TASKS

1.4.1. Verifying the installation

After the **openshift-ansible-istio-installer-job** has completed, run the following command:

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

Verify that you have a state similar to the following:

NAME	READY	STATUS	
elasticsearch-0	1/1	Running	0
2m			
grafana-6d5c5477-k7wrh	1/1	Running	0
2m			
istio-citadel-6f9c778bb6-q9tg9	1/1	Running	0
3m			

istio-egressgateway-957857444-2g84h 3m	1/1	Running	0
istio-galley-c47f5dfffc-dm27s 3m	1/1	Running	0
istio-ingressgateway-7db86747b7-s2dv9 3m	1/1	Running	0
istio-pilot-5646d7786b-rh54p 3m	2/2	Running	0
istio-policy-7d694596c6-pfdzt 3m	2/2	Running	0
istio-sidecar-injector-57466d9bb-4cjrs 3m	1/1	Running	0
istio-statsd-prom-bridge-7f44bb5ddb-6vx7n 3m	1/1	Running	0
istio-telemetry-7cf7b4b77c-p8m2k 3m	2/2	Running	0
jaeger-agent-5mswn 2m	1/1	Running	0
jaeger-collector-9c9f8bc66-j7kjb 2m	1/1	Running	0
jaeger-query-fdc6dcd74-99pnx 2m	1/1	Running	0
kiali-779bcc566f-qqt65 2m	1/1	Running	0
openshift-ansible-istio-installer-job-f8n9g 7m	0/1	Completed	0
prometheus-84bd4b9796-2vcpc 3m	1/1	Running	0

If you also chose to install the Fabric8 launcher, monitor the containers within the **devex** project until the following state is reached:

NAME	READY	STATUS	RESTARTS	AGE
configmapcontroller-1-8rr6w	1/1	Running	0	1m
launcher-backend-2-2wg86	1/1	Running	0	1m
launcher-frontend-2-jxjsd	1/1	Running	0	1m

1.5. TUTORIALS

There are several tutorials to help you learn more about the Service Mesh.

1.5.1. Bookinfo tutorial

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

The Bookinfo application consists of four separate microservices:

- **productpage** - The **productpage** microservice calls the **details** and **reviews** microservices to populate the page.
- **details** - The **details** microservice contains book information.

- **reviews** - The **reviews** microservice contains book reviews. It also calls the **ratings** microservice.
- **ratings** - The **ratings** microservice contains book ranking information that accompanies a book review.

There are three versions of the reviews microservice:

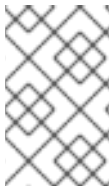
- Version v1 does not call the **ratings** service.
- Version v2 calls the **ratings** service and displays each rating as one to five black stars.
- Version v3 calls the **ratings** service and displays each rating as one to five red stars.

1.5.1.1. Installing the Bookinfo application

The following steps describe deploying and running the Bookinfo tutorial on OpenShift Container Platform with Service Mesh 0.3.TechPreview.

Prerequisites:

- OpenShift Container Platform 3.11 or higher installed.
- Red Hat OpenShift Service Mesh 0.3.TechPreview installed.



NOTE

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

1. Create a project for the Bookinfo application.

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

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

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

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

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

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

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

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

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

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

1.5.1.2. Verifying the Bookinfo installation

To confirm that the application is successfully deployed, run this command:

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

Alternatively, you can open [http://\\$GATEWAY_URL/productpage](http://$GATEWAY_URL/productpage) in your browser.

1.5.1.3. Add default destination rules

1. If you did not enable mutual TLS:

```
$ curl -o destination-rule-all.yaml
https://raw.githubusercontent.com/istio/istio/release-
1.0/samples/bookinfo/networking/destination-rule-all.yaml
$ oc apply -f destination-rule-all.yaml
```

2. If you enabled mutual TLS:

```
$ curl -o destination-rule-all-mtls.yaml
https://raw.githubusercontent.com/istio/istio/release-
1.0/samples/bookinfo/networking/destination-rule-all-mtls.yaml
$ oc apply -f destination-rule-all-mtls.yaml
```

3. To list all available destination rules:

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

1.5.1.4. Removing the Bookinfo application

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

TIP

Several of the other tutorials in this document also use the Bookinfo application. Do not run the cleanup script if you plan to continue with the other tutorials.

1. Download the cleanup script:

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

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

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

3. Confirm shutdown by running these commands:

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

1.5.2. 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 perform a trace using the Jaeger component of Red Hat OpenShift Service Mesh.

Prerequisites:

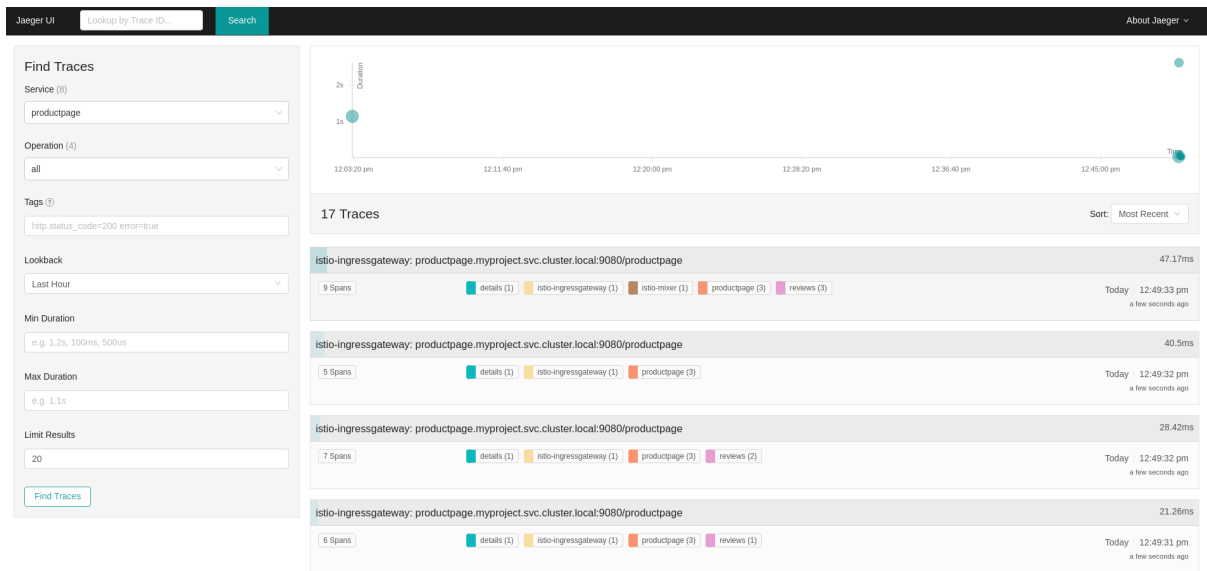
- OpenShift Container Platform 3.11 or higher installed.
- Red Hat OpenShift Service Mesh 0.3.TechPreview installed.
- **Bookinfo** demonstration application installed.

1.5.2.1. Generating traces and analyzing trace data

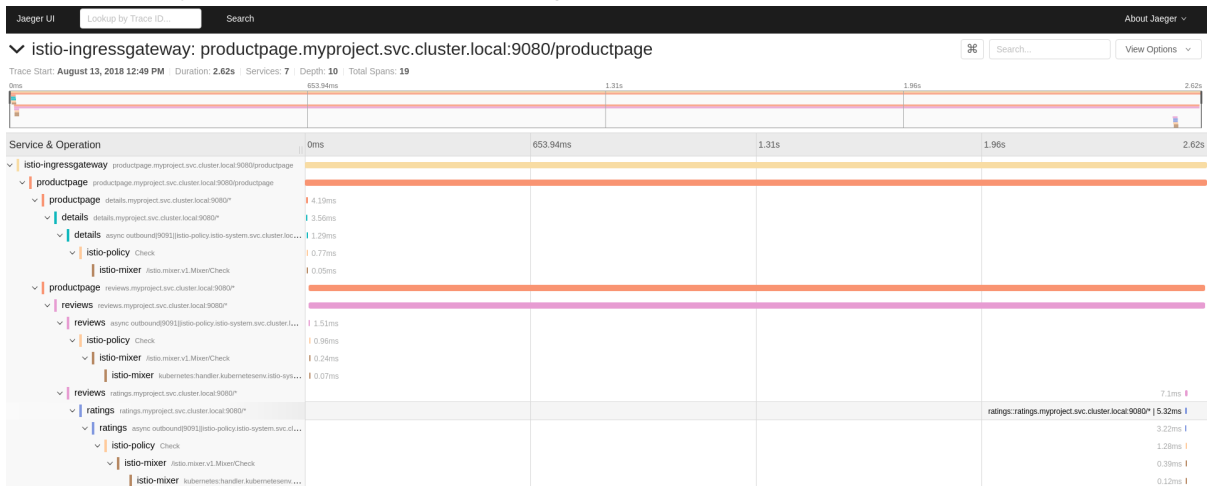
1. After you have deployed the Bookinfo application, generate some activity by accessing [http://\\$GATEWAY_URL/productpage](http://$GATEWAY_URL/productpage) and refreshing the page a few times.
2. A route to access the Jaeger dashboard already exists. Query for details of the route:

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

3. Launch a browser and navigate to [https://\\$JAEGER_URL](https://$JAEGER_URL).
4. In the left pane of the Jaeger dashboard, from the Service menu, select "productpage" and click the **Find Traces** button at the bottom of the pane. A list of traces is displayed, as shown in the following image:



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



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

1.5.2.2. Removing the tracing tutorial

Follow the procedure for removing the Bookinfo tutorial to remove the Tracing tutorial.

1.5.3. Prometheus tutorial

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

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

Prerequisites:

- OpenShift Container Platform 3.11 or higher installed.
- Red Hat OpenShift Service Mesh 0.3.TechPreview installed.
- **Bookinfo** demonstration application installed.

1.5.3.1. Querying metrics

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

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

You will see something like the following:

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

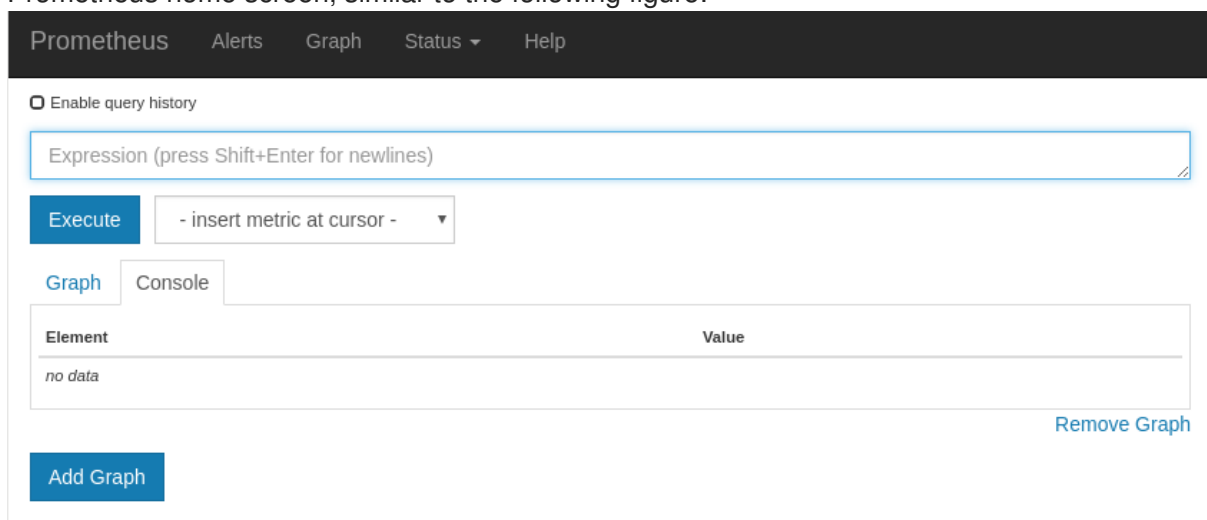
2. Generate network traffic by accessing the Bookinfo application:

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

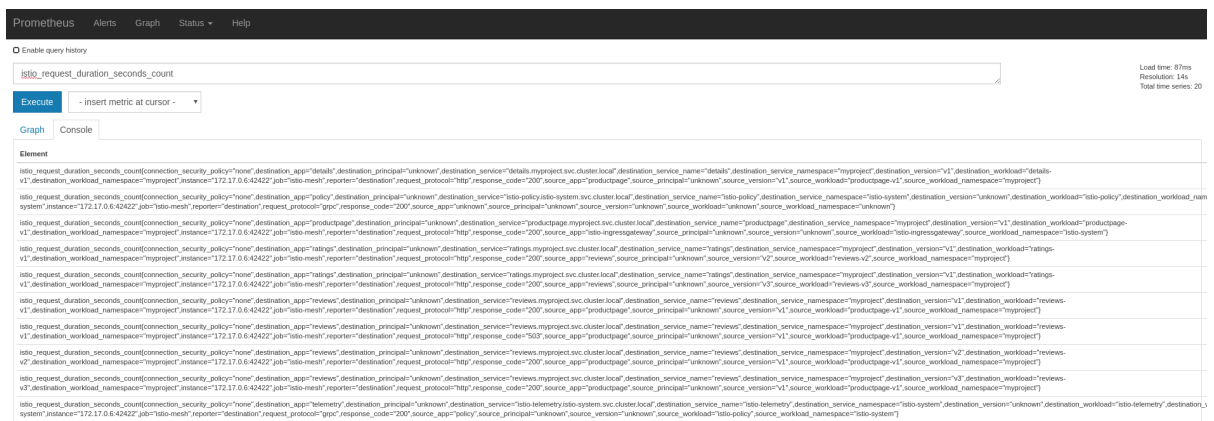
3. A route to access the Prometheus user interface already exists. Query for details of the route:

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

4. Launch a browser and navigate to [http://\\${PROMETHEUS_URL}](http://${PROMETHEUS_URL}). You will see the Prometheus home screen, similar to the following figure:



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



- You can narrow down queries by using selectors. For example **`istio_request_duration_seconds_count{destination_workload="reviews-v2"}`** shows only counters with the matching **`destination_workload`** label. For more information about using queries, see the [Prometheus documentation](#).
- To list all available Prometheus metrics, run the following command:

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

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

1.5.3.2. Removing the Prometheus tutorial

Follow the procedure for removing the Bookinfo tutorial to remove the Prometheus tutorial.

1.5.4. 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.

Prerequisites:

- OpenShift Container Platform 3.11 or higher installed.
- Red Hat OpenShift Service Mesh 0.3.TechPreview installed.
- Kiali parameters specified in the custom resource file.
- Bookinfo** demonstration application installed.

1.5.4.1. Accessing the Kiali console

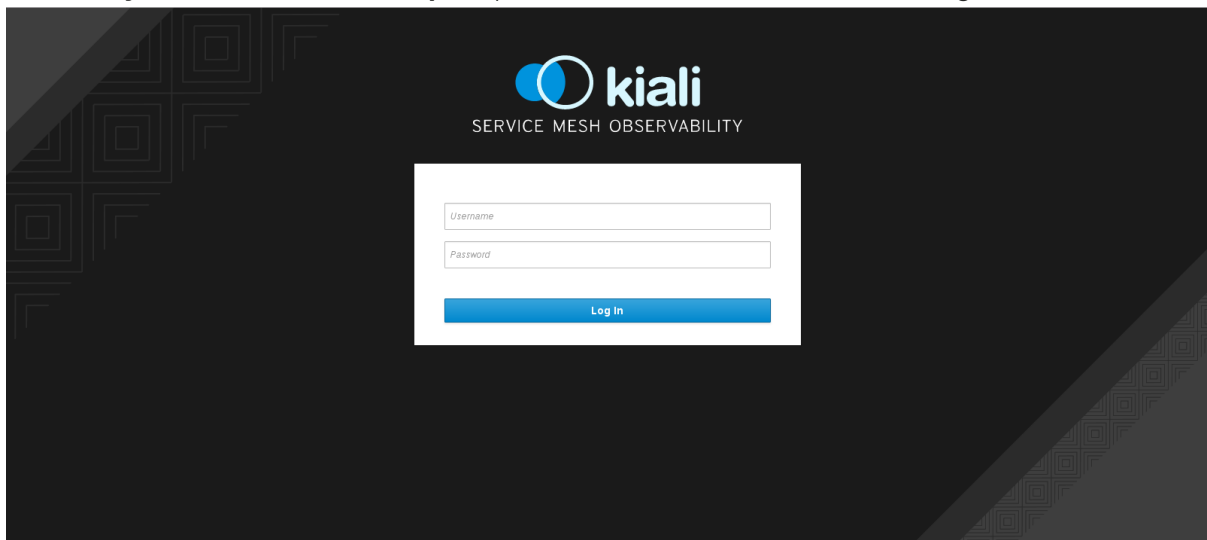
1. A route to access the Kiali console already exists. Run the following command to obtain the route and Kiali URL:

```
$ oc get routes
```

While your exact environment may be different, you should see a result that's something like this:

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

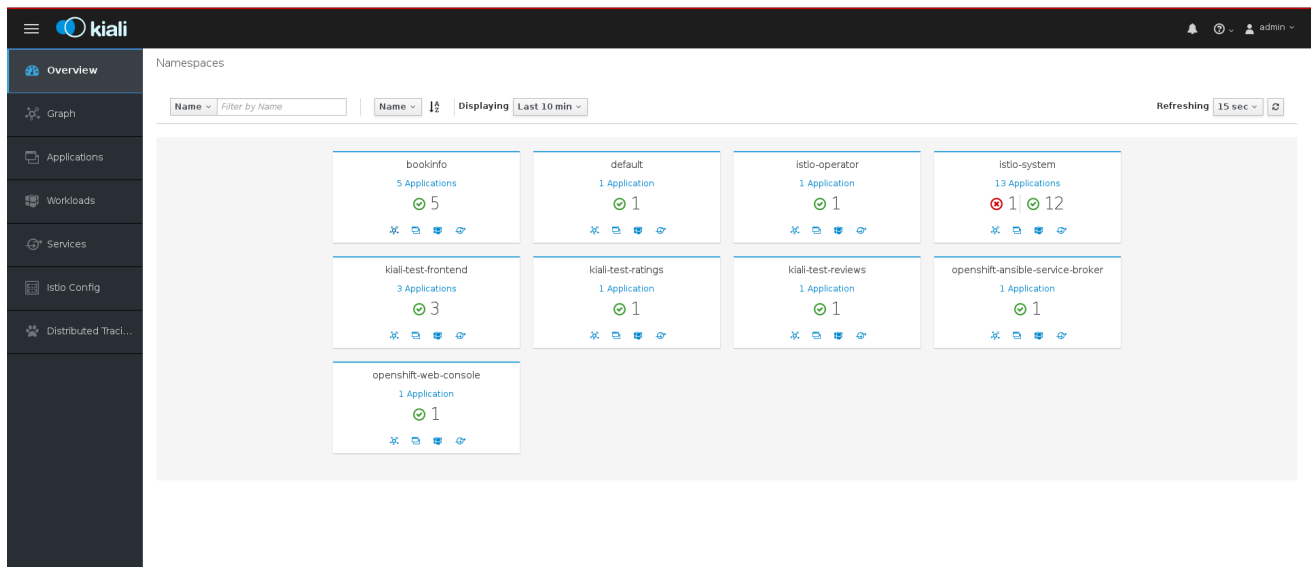
2. Launch a browser and navigate to [https://\\${KIALI_URL}](https://${KIALI_URL}) (in the output above, this is **kiali-istio-system.127.0.0.1.nip.io**). You should see the Kiali console login screen.



Log in to the Kiali console using the user name and password that you specified in the custom resource file during installation.

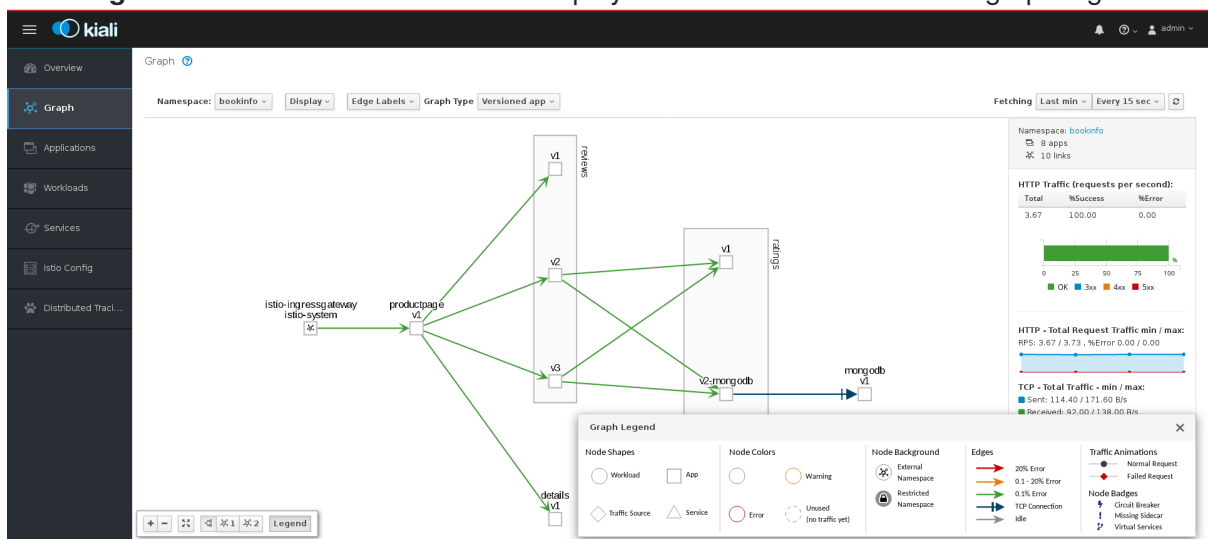
1.5.4.2. Overview page

After you log in you see the Overview page, which provides you with a quick overview of the health the various namespaces in your system.



1.5.4.3. Graph page

1. Click **Graph** in the left navigation. The Graph page shows a graph with all the microservices, which are connected by the requests going through them. On this page, you can see how the services interact with each other.
2. From the **Namespace** menu, select **bookinfo**. Now the graph displays only the services in the Bookinfo application.
3. Click **Legend** in the lower left corner. Kiali displays a window that contains the graph legend.

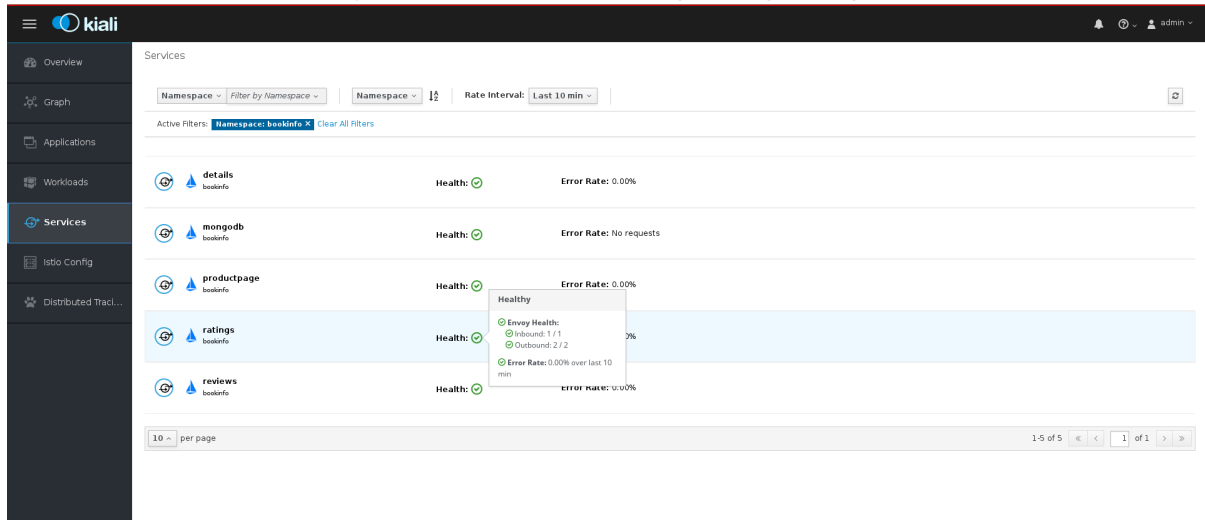


4. Close the Graph Legend.
5. Hover over the **productpage** node. Note how the graph highlights only the incoming and outgoing traffic from the node.
6. Click the **productpage** node. Note how the details on the right side of the page change to show the **productpage** details.

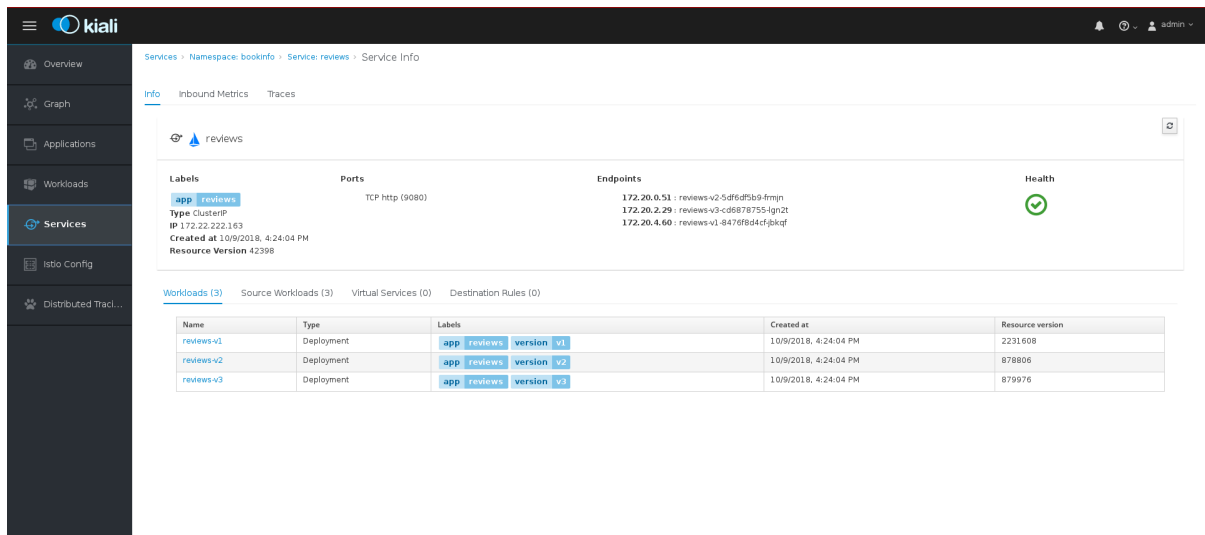
1.5.4.4. Services page

1. Click the **Services** link in the left navigation. On the Services page, you can view a listing of all the services that are running in the cluster and additional information about them, such as health status and request error rate.

2. 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.



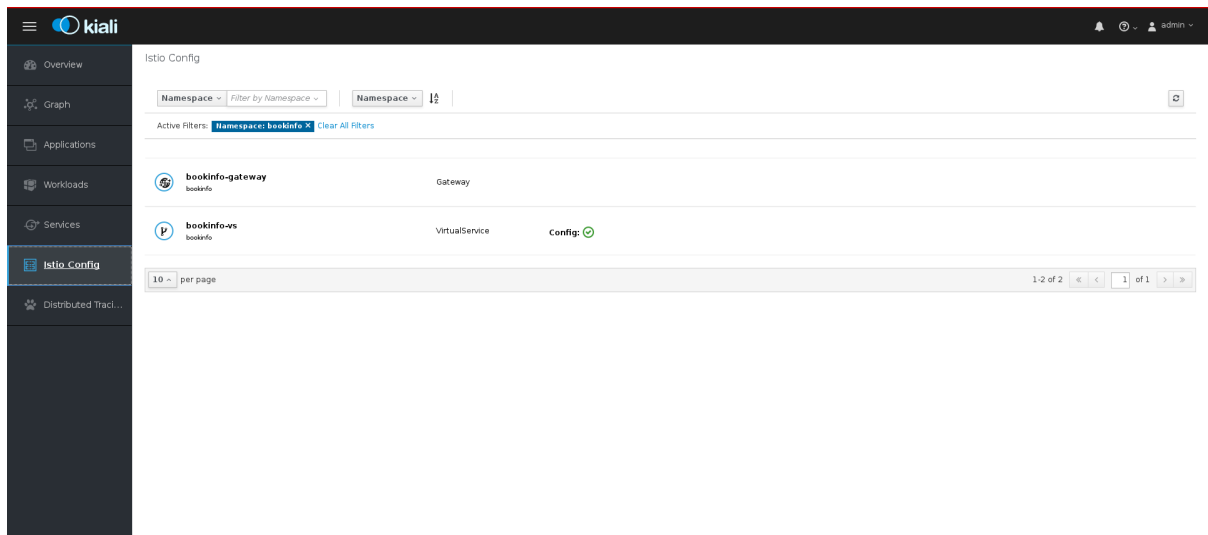
3. Click the **Reviews** service to view its details. Note that there are three different versions of this service.



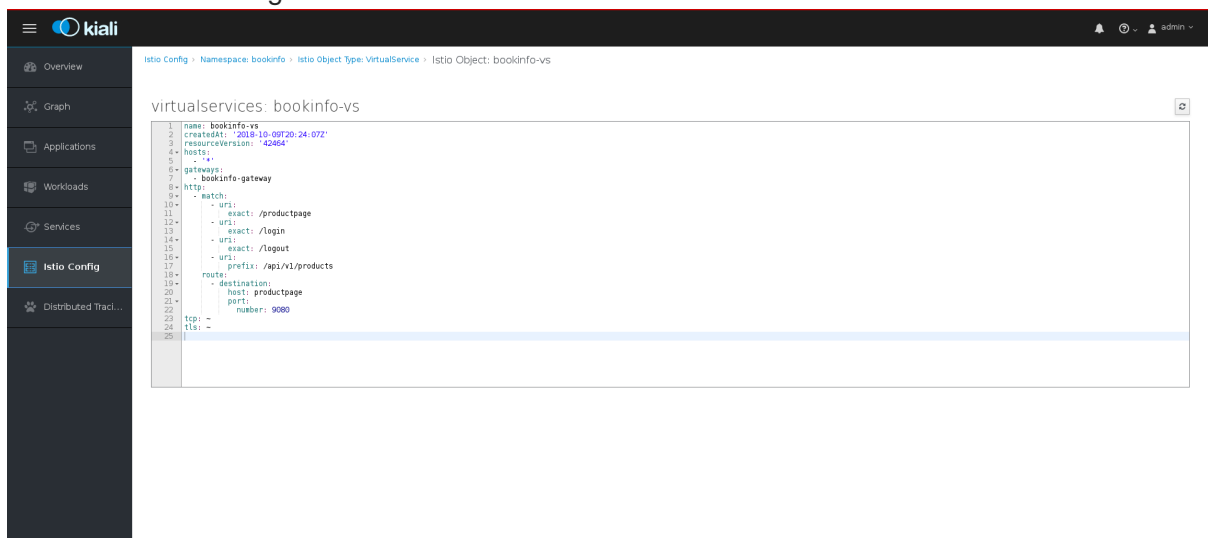
4. Click the name of one of the services to view additional details about that service.

1.5.4.5. Istio config page

1. Click the **Istio Config** link in the left navigation. On this page, you can see all of the currently running configurations, such as Circuit Breakers, Destination Rules, Fault Injection, Gateways, Routes, Route Rules, and Virtual Services.



2. Click one of the configurations to view additional information.



1.5.4.6. Distributed tracing page

Click the **Distributed Tracing** link in the left navigation. On this page you can see tracing data as provided by Jaeger.

1.5.4.7. Removing the Kiali tutorial

The procedure for removing the Kiali tutorial is the same as removing the Bookinfo tutorial.

1.5.5. Grafana tutorial

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

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

Prerequisites:

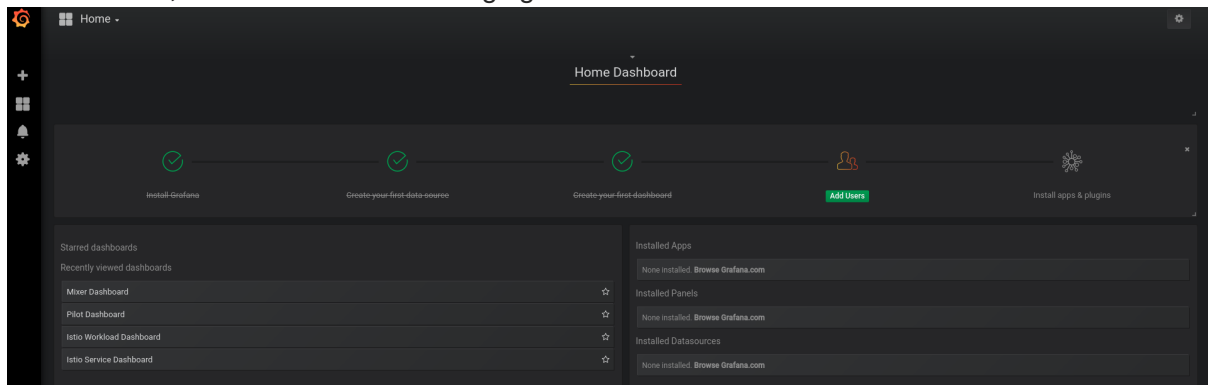
- OpenShift Container Platform 3.11 or higher installed.
- Red Hat OpenShift Service Mesh 0.3.TechPreview installed.
- **Bookinfo** demonstration application installed.

1.5.5.1. Accessing the Grafana dashboard

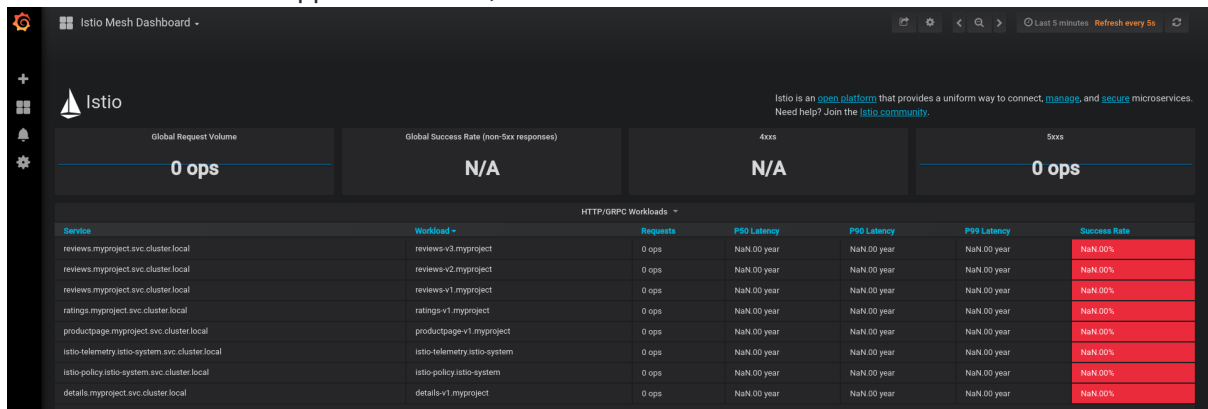
1. A route to access the Grafana dashboard already exists. Query for details of the route:

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

2. Launch a browser and navigate to [http://\\${GRAFANA_URL}](http://${GRAFANA_URL}). You see Grafana's home screen, as shown in the following figure:



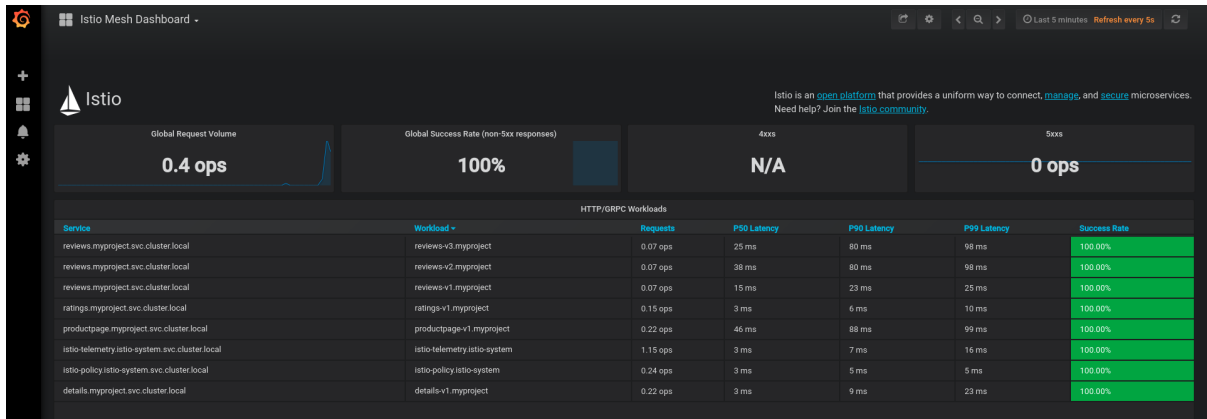
3. From the menu in the upper-left corner, select **Istio Mesh Dashboard** to see Istio mesh metrics.



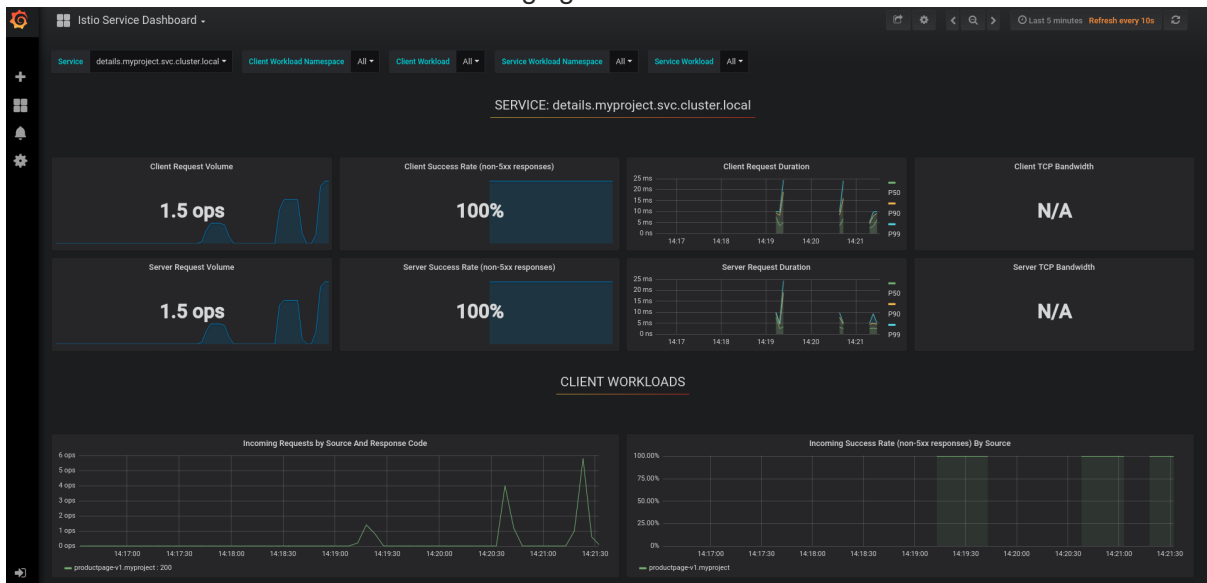
4. Generate some traffic by accessing the Bookinfo application:

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

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

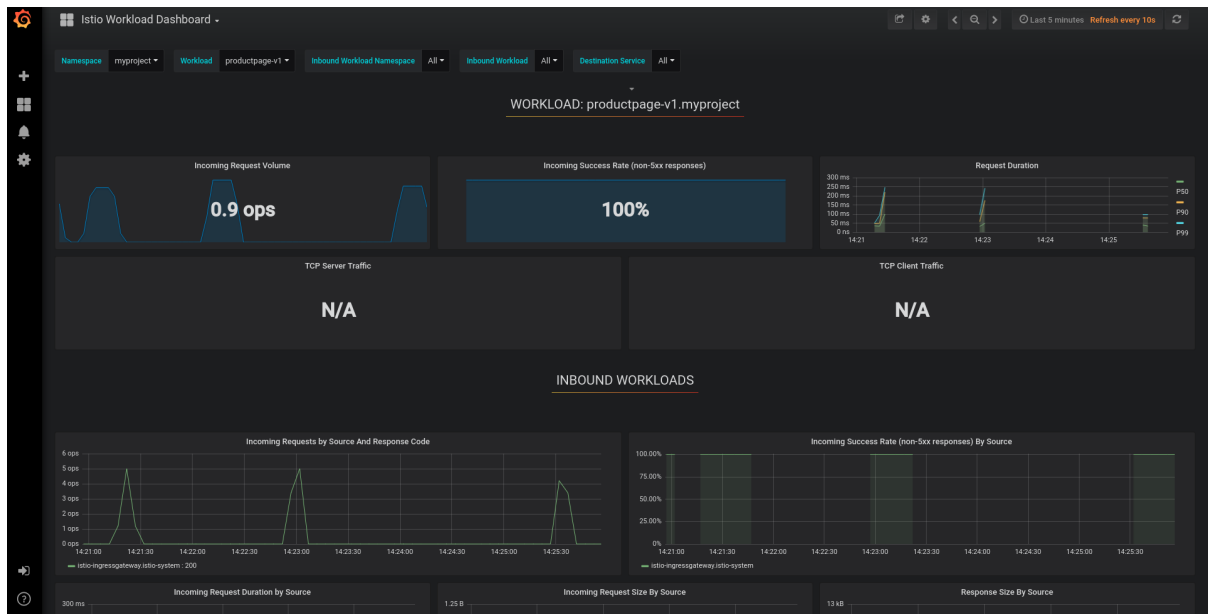


- To see detailed metrics for a service, click on a service name in the **Service** column. The service dashboard resembles the following figure:



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

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



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

1.5.5.2. Removing the Grafana tutorial

Follow the procedure for removing the Bookinfo tutorial to remove the Grafana tutorial.

1.5.6. Red Hat OpenShift Application Runtime Missions

In addition to the **bookinfo** based tutorials, there are four Service Mesh-specific tutorials (missions) and sample applications (boosters) that you can use with the Fabric8 integrated development platform launcher to explore some of the Istio features. Each of these boosters and missions is available for four different application runtimes. For more information about Fabric8, see the [Fabric8 documentation](#).

Prerequisites:

- OpenShift Container Catalog 3.11 or higher installed.
- Red Hat OpenShift Service Mesh 0.3.TechPreview installed.
- Launcher parameters specified in the custom resource file.

Table 1.6. RHOAR Tutorials

Runtime	Mission	Description
Springboot	Istio Circuit Breaker mission	This scenario showcases how Istio can be used to implement the Circuit Breaker architectural pattern.

Runtime	Mission	Description
Springboot	Istio Distributed Tracing mission	This scenario showcases the interaction of Distributed Tracing capabilities of Jaeger and properly instrumented microservices running in the Service Mesh.
Springboot	Security mission	This scenario showcases Istio security concepts whereby access to services is controlled by the platform rather than independently by constituent applications.
Springboot	Routing mission	This scenario showcases using Istio's dynamic traffic routing capabilities with a set of example applications designed to simulate a real-world rollout scenarios scenario.
Thorntail (A.K.A. Wildfly Swarm)	Istio Circuit Breaker mission	This scenario showcases how Istio can be used to implement the Circuit Breaker architectural pattern.
Thorntail	Istio Distributed Tracing mission	This scenario showcases the interaction of Distributed Tracing capabilities of Jaeger and properly instrumented microservices running in the Service Mesh.
Thorntail	Security mission	This scenario showcases Istio security concepts whereby access to services is controlled by the platform rather than independently by constituent applications.
Thorntail	Routing mission	This scenario showcases using Istio's dynamic traffic routing capabilities with a set of example applications designed to simulate a real-world rollout scenarios scenario.
Vert.x	Istio Circuit Breaker mission	This scenario showcases how Istio can be used to implement a Circuit Breaker via a (minimally) instrumented Eclipse Vert.x microservices.

Runtime	Mission	Description
Vert.x	Istio Distributed Tracing mission	This scenario showcases the interaction of Distributed Tracing capabilities of Jaeger and a (minimally) instrumented set of Eclipse Vert.x applications.
Vert.x	Security mission	This scenario showcases Istio Transport Layer Security(TLS) and Access Control Lists (ACL) via a set of Eclipse Vert.x applications.
Vert.x	Routing mission	This scenario showcases using Istio's dynamic traffic routing capabilities with a with a minimal set of example applications.
Node.js	Istio Circuit Breaker mission	This scenario showcases how Istio can be used to implement the Circuit Breaker architectural pattern in Node.js applications.
Node.js	Istio Distributed Tracing mission	This scenario showcases the interaction of Distributed Tracing capabilities of Jaeger and properly instrumented Node.js applications running in the Service Mesh.
Node.js	Security mission	This scenario showcases Istio Transport Layer Security(TLS) and Access Control Lists (ACL) with Node.js applications.
Node.js	Routing mission	This scenario showcases using Istio's dynamic traffic routing capabilities with a set of example applications designed to simulate a real-world rollout scenarios scenario.

1.6. REMOVING RED HAT OPENSIFT SERVICE MESH

1.6.1. Removing Red Hat OpenShift Service Mesh

The following command removes the Service Mesh from an existing installation. You can run these commands from any host with access to the cluster. Ensure you are logged in as a cluster admin before running the following commands.

1. Remove the custom resource by running the following command:

```
$ oc delete -n istio-operator installation istio-installation
```

2. Remove the operator by running the following command:

```
$ oc process -f istio_product_operator_template.yaml | oc delete -f -
```

1.7. UPGRADING RED HAT OPENSIFT SERVICE MESH

1.7.1. Upgrading Red Hat OpenShift Service Mesh

While Red Hat OpenShift Service Mesh is a Technology Preview, there is no upgrade. If you have an existing Service Mesh installation (for example if you have installed the developer preview) then that installation **must** be removed prior to installing a new version of Red Hat OpenShift Service Mesh.