OpenShift Container Platform 4.1

Service Mesh

Service Mesh installation, usage and release notes

Last Updated: 2019-09-20

OpenShift Container Platform 4.1 Service Mesh

Service Mesh installation, usage and release notes

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. SERVICE MESH ARCHITECTURE

1.1. UNDERSTANDING RED HAT OPENSHIFT 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.

1.1.1. 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

1.1.2. 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.

1.1.3. 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.

1.1.4. 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.

1.1.5. 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.

1.1.6. 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>"
```

1.1.7. 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.

1.1.8. 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.

1.2. KIALI OVERVIEW

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

1.2.1. 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.

1.2.2. 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 Telemetery. 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.

1.2.3. 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.

1.3. 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.

1.3.1. 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.

1.3.2. 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.

1.3.3. 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.

1.4. 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:

1.4.1. 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.

1.4.2. Multi-tenancy in Red Hat OpenShift Service Mesh versus cluster-wide installations

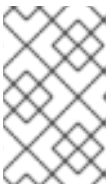
The main difference between a multi-tenant installation and a cluster-wide installation is the scope of privileges used by the control plane deployments, for example, Galley and Pilot. The components no longer use cluster-scoped Role Based Access Control (RBAC) resource **ClusterRoleBinding**, but rely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

1.4.3. 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.

1.4.4. 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>"
```

1.4.5. OpenSSL

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

1.4.6. 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.

1.4.7. 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.

1.4.8. 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 2. SERVICE MESH INSTALLATION

2.1. PREPARING TO INSTALL RED HAT OPENSHIFT SERVICE MESH

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

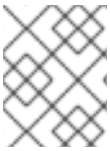
Prerequisites

- Possess an active OpenShift Container Platform subscription on your Red Hat account. If you do not have a subscription, contact your sales representative for more information.
- Review the OpenShift Container Platform 4.1 overview .
- Install OpenShift Container Platform 4.1.
  - Install OpenShift Container Platform 4.1 on AWS
  - Install OpenShift Container Platform 4.1 on user-provisioned AWS
  - Install OpenShift Container Platform 4.1 on bare metal
  - Install OpenShift Container Platform 4.1 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 4.1, see About the CLI.

2.1.1. Red Hat OpenShift Service Mesh supported configurations

The following are the only supported configurations for the Red Hat OpenShift Service Mesh 1.0:

- Red Hat OpenShift Container Platform version 4.1.



NOTE

OpenShift Online and OpenShift Dedicated are not supported for Red Hat OpenShift Service Mesh 1.0.

- The deployment must be contained to a single OpenShift Container Platform cluster that is not federated.
- This release of Red Hat OpenShift Service Mesh is only available on OpenShift Container Platform x86_64.
- Red Hat OpenShift Service Mesh is only suited for OpenShift Container Platform Software Defined Networking (SDN) configured as a flat network with no external providers.
- This release only supports configurations where all Service Mesh components are contained in the OpenShift cluster in which it operates. It does not support management of microservices that reside outside of the cluster, or in a multi-cluster scenario.
- This release only supports configurations that do not integrate external services such as virtual machines.

2.1.1.1. 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.

2.1.1.2. Supported Mixer adapters

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

2.1.2. 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.

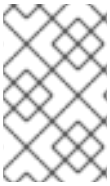
- **Red Hat OpenShift Service Mesh** - based on the open source Istio project, lets you connect, secure, control, and observe the microservices that make up your applications.

Next steps

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

2.2. INSTALLING RED HAT OPENSHIFT 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.

Prerequisites

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

2.2.1. 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, 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.

2.2.1.1. 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 **Catalog → 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.

2.2.1.2. 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 **Catalog → 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.

2.2.1.3. Installing the Kiali Operator

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

WARNING

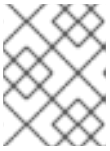
Do not install Community versions of the Operators. Community Operators are not supported.

Prerequisites

- Access to the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Catalog → 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.

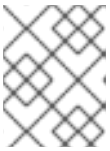
2.2.1.4. Installing the Red Hat OpenShift Service Mesh Operator

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 **Catalog → 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.

2.2.2. Deploying the Red Hat OpenShift Service Mesh control plane

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

2.2.2.1. Deploying the control plane with 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 Customize the Red Hat OpenShift Service Mesh installation instructions.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Create a new project named **istio-system**.
3. Navigate to **Catalogs → Installed Operators**.
4. Click the Red Hat OpenShift Service Mesh Operator.
5. Under **Provided APIs**, the Operator enables you to create two resource types:
   - A **ServiceMeshControlPlane** resource
   - A **ServiceMeshMemberRoll** resource
6. Click **Create New** under **Istio Service Mesh Control Plane**
7. Modify the minimal **ServiceMeshControlPlane** template.

NOTE

Review Customize the Red Hat OpenShift Service Mesh installation for additional information on customizing the control plane and control plane parameters.

8. Click **Create** to create the control plane.
9. The Operator starts up the pods, services, and Service Mesh control plane components.
10. Click the **Istio Service Mesh Control Plane**tab.
11. Click the name of the new control plane.
12. Click the **Resources** tab to see the Red Hat OpenShift Service Mesh control plane resources the Operator created and configured.

2.2.2.2. Deploying the control plane from the CLI

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

Prerequisites

- The Red Hat OpenShift Service Mesh Operator must be installed.
- Review the Customize the Red Hat OpenShift Service Mesh installation instructions.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as **oc**.

NOTE

Review Customize the Red Hat OpenShift Service Mesh installation for additional information on customizing the control plane and control plane parameters.

Procedure

1. Log in to the OpenShift Container Platform CLI.
2. Create a **ServiceMeshControlPlane** file named **istio-installation.yaml**.
3. Run this command to deploy the control plane:

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

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

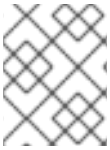
```
$ oc get pods -n istio-system -w
```

2.2.3. Configure the Red Hat OpenShift Service Mesh member roll

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

WARNING

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



NOTE

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

- 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.

2.2.3.1. Configure the member roll from the OpenShift Container Platform web console

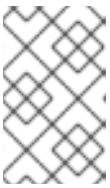
Follow this procedure to add the Bookinfo project to the Service Mesh member roll by using the web console.

Prerequisites

- An installed, verified Red Hat OpenShift Service Mesh Operator.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Click to **Home → Projects**.
3. Click **Create Project**.
4. Enter a **Project Name** (for example, **bookinfo**), a **Display Name**, and a **Description**, then click **Create**.
5. Click **Catalog → Installed Operators**.
6. Click the **Project** menu and choose **istio-system** from the list.
7. Click the **Istio Service Mesh Member Roll**link under **Provided APIs** for the **Red Hat OpenShift Service Mesh** Operator.
8. Click on **All Instances**, click **Create New**, and then click **Create Istio Service Mesh Member Roll**.



NOTE

It can take a short time for the Operator to finish creating the projects, therefore you may need to refresh the screen before the web console presents the **Create Istio Service Mesh Member Roll** button.

9. Edit the default Service Mesh Member Roll YAML and add **bookinfo** to the **members** list.
10. Click **Create** to save the updated Service Mesh Member Roll.

2.2.3.2. Configure the member roll from the CLI

This example joins the Bookinfo project to the Service Mesh from the CLI.

Prerequisites

- An installed, verified Service Mesh Operator.
- Name of the project with the **ServiceMeshMemberRoll** resource.
- 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. Create **ServiceMeshMemberRoll** resource in the same project as the **ServiceMeshControlPlane** resource.

3. Name the resource **default**.

4. Add the Bookinfo project to the member list in the **ServiceMeshMemberRoll**. In this example, the **bookinfo** project is joined to the Service Mesh deployed in the same project as the **ServiceMeshMembereRoll** resource.

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

Next steps

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

2.3. CUSTOMIZING THE RED HAT OPENSHIFT 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

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

2.3.1. Red Hat OpenShift Service Mesh custom resources

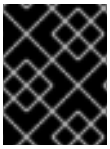
NOTE

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 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).

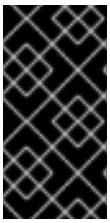
Full example istio-installation.yaml

```
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: 100.0

    kiali:
      enabled: true

    tracing:
```

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

2.3.2. 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.

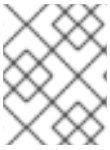
2.3.2.1. Istio global example

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**.

```
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
```



NOTE

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

Table 2.1. Global parameters

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

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

Table 2.2. Proxy parameters

Type	Parameter	Description	Values	Default value
Resources	**cpu**	The amount of CPU resources requested for Envoy proxy	CPU resources in cores or millicores based on your environment's configuration	**100m**
	memory	The amount of memory requested for Envoy proxy	Available memory in bytes based on your environment's configuration	**128Mi**

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

2.3.2.2. 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.

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

Table 2.3. Istio Gateway parameters

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

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

2.3.2.3. 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.

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

Table 2.4. Istio Mixer policy parameters

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

Table 2.5. Istio Mixer telemetry parameters

Type	Parameter	Description	Values	Default
Resources	**cpu**	The percentage of CPU resources requested for Mixer telemetry	CPU resources in millicores based on your environment's configuration	**1000m**
	memory	The amount of memory requested for Mixer telemetry	Available memory in bytes based on your environment's configuration	**1G**
Limits	**cpu**	The maximum percentage of CPU resources Mixer telemetry is permitted to use	CPU resources in millicores based on your environment's configuration	**4800m**
	memory	The maximum amount of memory Mixer telemetry is permitted to use	Available memory in bytes based on your environment's configuration	**4G**

2.3.2.4. 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.

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

Table 2.6. Istio Pilot parameters

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

2.3.3. 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 Kaili instances.

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

Default Kiali parameters

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

Table 2.7. Kiali parameters

Parameter	Description	Values	Default value
enabled	This enables or disables Kiali in Service Mesh. Kiali is installed by default. If you do not want to install Kiali, change the **enabled** value to **false**.	**true**/**false**	**true**
dashboard viewOnlyMode	Whether the Kiali console should be in a view-only mode, not allowing the user to make changes to the Service Mesh.	**true**/**false**	**false**
ingress enabled	This enables/disables ingress.	**true**/**false**	**true**

2.3.3.1. 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 Kaili
- 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
```

2.3.3.2. 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 Kaili
- 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
```

2.3.4. 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 Jaeger parameters

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    tracing:
      enabled: true
      ingress:
        enabled: true
```

Table 2.8. Jaeger parameters

Parameter	Description	Values	Default value
`tracing` `  enabled`	This enables or disables tracing in Service Mesh. Jaeger is installed by default. If you do not want to install Jaeger, change the **enabled** value to **false**.	**true**/**false**	**true**

Parameter	Description	Values	Default value
ingress enabled	This enables/disables ingress.	**true**/**false**	**true**

2.3.4.1. Configuring Elasticsearch

Jaeger can be configured for different storage backends:

- **Memory** - Simple in-memory storage, only recommended for development, demo, or testing purposes. This is the default option for the **AllInOne** deployment strategy. Do NOT use for production environments.
- **Elasticsearch** - For production use. This is the default option for the **Production** deployment strategy.

NOTE

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

Elasticsearch is a memory intensive application. The initial set of nodes created by the OpenShift Container Platform installation may not be large enough to support the Elasticsearch cluster. Additional nodes must be added to the cluster if you want to run with the recommended amount (or more) memory. Each Elasticsearch node can operate with a lower memory setting though this is not recommended for production deployments.

You should modify the default Elasticsearch configuration to match your use case. You can adjust both the CPU and memory limits for each component by modifying the resources block with valid memory and CPU values.

Default Jaeger parameters for Elasticsearch in production

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

Table 2.9. Elasticsearch parameters

Parameter	Values	Description
nodeCount	integer value	Number of Elasticsearch nodes
cpu	Specified in units of cores (e.g., 200m, 0.5, 1)	Number of central processing units
memory	Specified in units of bytes (e.g., 200Ki, 50Mi, 5Gi)	Memory limit

Table 2.10. Sample configurations

Parameter	Proof of Concept	Minimal Deployment
Node count	1	3
Requests CPU	500m	1
Requests memory	1Gi	16Gi
Limits CPU	500m	1
Limits memory	1Gi	16Gi

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.

2.3.5. 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.

```
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 2.11. 3scale parameters

Parameter	Description	Values	Default value
enabled	Whether to use the 3scale adapter	**true**/**false**	**false**
PARAM_THREESCALE_LISTEN_ADDR	Sets the listen address for the gRPC server	Valid port number	**3333**
PARAM_THREESCALE_LOG_LEVEL	Sets the minimum log output level.	**debug**, **info**, **warn**, **error**, or **none**	**info**
PARAM_THREESCALE_LOG_JSON	Controls whether the log is formatted as JSON	**true**/**false**	**true**
PARAM_THREESCALE_LOG_GRPC	Controls whether the log contains gRPC info	**true**/**false**	**true**
PARAM_THREESCALE_REPORT_METRICS	Controls whether 3scale system and backend metrics are collected and reported to Prometheus	**true**/**false**	**true**
PARAM_THREESCALE_METRICS_PORT	Sets the port that the 3scale /**metrics** endpoint can be scrapped from	Valid port number	**8080**
PARAM_THREESCALE_CACHE_TTL_SECONDS	Time period, in seconds, to wait before purging expired items from the cache	Time period in seconds	**300**
PARAM_THREESCALE_CACHE_REFRESH_SECONDS	Time period before expiry when cache elements are attempted to be refreshed	Time period in seconds	**180**

Parameter	Description	Values	Default value
PARAM_THREESCALE_CACHE_ENTRIES_MAX	Max number of items that can be stored in the cache at any time. Set to **0** to disable caching	Valid number	**1000**
PARAM_THREESCALE_CACHE_REFRESH_RETRIES	The number of times unreachable hosts are retried during a cache update loop	Valid number	**1**
PARAM_THREESCALE_ALLOW_INSECURE_CONN	Allow to skip certificate verification when calling **3scale** APIs. Enabling this is not recommended.	**true**/**false**	**false**
PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS	Sets the number of seconds to wait before terminating requests to 3scale System and Backend	Time period in seconds	**10**
PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS	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.

2.4. REMOVING RED HAT OPENSHIFT SERVICE MESH

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

2.4.1. 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.

2.4.1.1. 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

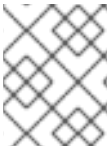
1. Log in to the OpenShift Container Platform web console.
2. Click the **Project** menu and choose the **istio-system** project from the list.
3. Navigate to **Catalogs → Installed Operators**.
4. Click on **Service Mesh Control Plane** under **Provided APIs**.
5. Click the **ServiceMeshControlPlane**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

2.4.1.2. 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>
```

2.4.2. 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.

2.4.2.1. 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. Navigate to **Catalogs → Operator Management**.
3. Click the **Project** menu, and then click **openshift-operators**.
4. Click the **servicemeshoperator** Options menu  , and then click **Remove Subscription**.
5. Select **Also completely remove the Operator from the selected namespace**, and then click **Remove**.

2.4.2.2. 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. Navigate to **Catalogs → Operator Management**.
3. Click the **Project** menu, and then click **openshift-operators**.
4. Click the **jaegeroperator** Options menu  , and then click **Remove Subscription**.
5. Select **Also completely remove the Operator from the selected namespace**, and then click **Remove**.

2.4.2.3. 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. Navigate to **Catalogs → Operator Management**.
3. Click the **Project** menu, and then click **openshift-operators**.
4. Click the **kiali-ossm** Options menu  , and then click **Remove Subscription**.
5. Select **Also completely remove the Operator from the selected namespace**, and then click **Remove**.

2.4.2.4. 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. Navigate to **Catalogs → Operator Management**.
3. Click the **Project** menu, and then click **openshift-operators**.
4. Click the **elasticsearch-operator** Options menu  , and then click **Remove Subscription**.
5. Select **Also completely remove the Operator from the selected namespace**, and then click **Remove**.

2.4.2.5. 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 3. DAY TWO

3.1. DEPLOYING APPLICATIONS ON RED HAT OPENSHIFT SERVICE MESH

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

Prerequisites

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

3.1.1. 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**.

3.1.1.1. Creating the ConfigMap

Follow this procedure to create the ConfigMap.

Prerequisites

- An installed, verified Service Mesh Operator.
- An account with cluster administrator access.
- 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
```

3.1.2. 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.

3.1.2.1. 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.

3.1.3. 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.

3.2. 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.

3.2.1. 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 3.1. 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

3.3. EXAMPLE APPLICATION

WARNING

The Bookinfo example application allows you to test your Red Hat OpenShift Service Mesh 1.0 installation on OpenShift Container Platform 4.1.

Red Hat does not provide support for the Bookinfo application.

3.3.1. 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:

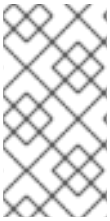
- Version v1 does not call the **ratings** Service.
- Version v2 calls the **ratings** Service and displays each rating as one to five black stars.
- Version v3 calls the **ratings** Service and displays each rating as one to five red stars.

3.3.2. 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.

Prerequisites:

- OpenShift Container Platform 4.1 or higher installed.
- Red Hat OpenShift Service Mesh 1.0 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.
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 **Catalog → Installed Operators**.
6. Click the **Project** menu and choose **openshift-operators** from the list.
7. Click the **Istio Service Mesh Member Roll**link under **Provided APIs** for the **Red Hat OpenShift Service Mesh** Operator.
8. Click **Create Service Mesh Member Roll**
9. Edit the default Service Mesh Member Roll YAML and add **bookinfo** to the **members** list.

   ```
   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}')
```

3.3.3. 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
```

3.3.4. 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 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** in your browser.
   - You can also verify that all pods are ready with this command:

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

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

3.3.5.1. 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
```

3.3.5.2. Remove the bookinfo project from the Service Mesh member roll

Procedure

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

CHAPTER 4. 3SCALE ADAPTER

4.1. USING THE 3SCALE ISTIO ADAPTER

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

4.1.1. Integrate the 3scale adapter with Red Hat OpenShift Service Mesh

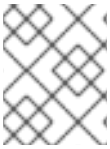
You can use these examples to configure requests to your services using the 3scale Istio Adapter.

Prerequisites:

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

NOTE

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



NOTE

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

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

   Handler configuration example

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

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

Rule configuration example

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

4.1.1.1. Generating 3scale custom resources

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

Table 4.1. Usage

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

4.1.1.1.1. Generate templates from URL examples

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

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

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

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

4.1.1.2. Generating manifests from a deployed adapter

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

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

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

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

   a. Label the workload (recommended)

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

4. Update the workload with the required annotations:

NOTE

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

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

4.1.1.3. 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.

4.1.2. 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**.

4.1.3. 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.

4.1.4. Authenticating requests

This Technology Preview release supports the following authentication methods:

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

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

4.1.4.1. Applying authentication patterns

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

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

NOTE

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

4.1.4.1.1. API key authentication method

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

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

API key authentication method example

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

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

4.1.4.1.2. Application ID and application key pair authentication method

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

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

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

Application ID and application key pair authentication method example

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

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

4.1.4.1.3. OpenID authentication method

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

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

OpenID authentication method example

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

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

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

OpenID Policy example

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

4.1.4.1.4. Hybrid authentication method

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

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

Hybrid authentication method example

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

4.1.5. 3scale Adapter metrics

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

CHAPTER 5. SERVICE MESH RELEASE NOTES

CHAPTER 6. RED HAT OPENSHIFT SERVICE MESH RELEASE NOTES

6.1. RED HAT OPENSHIFT 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.

6.2. 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:

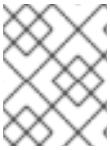
- Search or browse through the Red Hat Knowledgebase of technical support articles about Red Hat products
- Submit a support case to Red Hat Global Support Services (GSS)
- Access other product documentation

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.

6.3. RED HAT OPENSHIFT SERVICE MESH SUPPORTED CONFIGURATIONS

The following are the only supported configurations for the Red Hat OpenShift Service Mesh 1.0:

- Red Hat OpenShift Container Platform version 4.1.



NOTE

OpenShift Online and OpenShift Dedicated are not supported for Red Hat OpenShift Service Mesh 1.0.

- The deployment must be contained to a single OpenShift Container Platform cluster that is not federated.
- This release of Red Hat OpenShift Service Mesh is only available on OpenShift Container Platform x86_64.
- Red Hat OpenShift Service Mesh is only suited for OpenShift Container Platform Software Defined Networking (SDN) configured as a flat network with no external providers.
- This release only supports configurations where all Service Mesh components are contained in the OpenShift cluster in which it operates. It does not support management of microservices that reside outside of the cluster, or in a multi-cluster scenario.
- This release only supports configurations that do not integrate external services such as virtual machines.

6.3.1. 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.

6.3.2. Supported Mixer adapters

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

6.3.3. 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.

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.

6.3.4. New features Technology Preview 12

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

Other notable changes in this release include the following:

- Integrates the Kiali and Jaeger operators into the installation.
- Adds support for Kubernetes Container Network Interface (CNI) plug-in.

- Adds support for Operator Lifecycle Management (OLM).
- Updates the Istio OpenShift Router for multitenancy.
- Defaults to configuring the control planes for multitenancy. You can still configure a single tenant control plane in Red Hat OpenShift Service Mesh 0.12.TechPreview.

NOTE

To simplify installation and support, this release only supports a multi-tenant control plane for one or more tenants.

6.3.5. New features Technology Preview 11

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

6.3.6. New features Technology Preview 10

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

6.3.7. New features Technology Preview 9

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

6.3.8. New features Technology Preview 8

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

6.3.9. New features Technology Preview 7

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

6.3.10. New features Technology Preview 6

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

6.3.11. New features Technology Preview 5

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

6.3.12. New features Technology Preview 4

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

6.3.13. New features Technology Preview 3

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

6.3.14. New features Technology Preview 2

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

6.3.15. New features Technology Preview 1

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

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

6.4. 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 it 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.

NOTE

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

6.4.1. 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-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-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-348 To access a TCP service by using the ingress gateway on a port other than 80 or 443, use the service hostname provided by the AWS load balancer rather than the OpenShift router.
  The istio-ingressgateway route hostname (for example, **istio-ingressgateway-istio-system.apps.[cluster name].openshift.com**) works with port 80 or port 443 traffic. However, that route hostname does not support other port traffic.

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

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

  ```
  $ oc -n istio-system get service istio-ingressgateway -o
  jsonpath='{.status.loadBalancer.ingress[0].hostname}'
  ```
- MAISTRA-193 Unexpected console info messages are visible when health checking is enabled for citadel.
- MAISTRA-158 Applying multiple gateways referencing the same hostname will cause all gateways to stop functioning.
- MAISTRA-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.

6.4.2. Kiali known issues

- KIALI-3265 The Service Details page generates an unnecessary Notification in the Message Center that Kiali could not fetch Traces, even though it is displaying trace data on the page.
- 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-3251 In the Kiali Console, on the Istio Config page, if you cancel changes to a YAML file, the changes are not canceled and the YAML tab is marked with an asterisk (*) as having been edited.

- KIALI-3246 In the Kiali Console, on the Istio Config page, the Istio Config List cannot filter on more than a single Istio Name. The second Name selected overrides the first Name selected and the results returned are only filtered by the second Name. This issue only affects Istio Names, you can have multiple values of the other filter criteria.

- 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-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.

- 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.

6.5. FIXED ISSUES

The following issues been resolved in the current release:

6.5.1. Red Hat OpenShift Service Mesh fixed issues

- 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-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.
  The load balancer health check only checks the first port defined in the Istio ingress gateway ports list. This port is configured as 80/HTTP:31380/TCP. Without a service running on this port, the load balancer health check fails.

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

6.5.2. 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.