OpenShift Container Platform 4.5

Service Mesh

Service Mesh installation, usage, and release notes

Last Updated: 2021-01-06

OpenShift Container Platform 4.5 Service Mesh

Service Mesh installation, usage, and release notes

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. SERVICE MESH 1.X

1.1. SERVICE MESH RELEASE NOTES

1.1.1. 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.

1.1.2. Getting support

If you experience difficulty with a procedure described in this documentation, or with OpenShift Container Platform in general, visit the Red Hat Customer Portal . From the Customer Portal, you can:

- Search or browse through the Red Hat Knowledgebase of articles and solutions relating to Red Hat products.
- Submit a support case to Red Hat Support.
- Access other product documentation.

To identify issues with your cluster, you can use Insights in Red Hat OpenShift Cluster Manager. Insights provides details about issues and, if available, information on how to solve a problem.

If you have a suggestion for improving this documentation or have found an error, please submit a Bugzilla report against the **OpenShift Container Platform** product for the **Documentation** component. Please provide specific details, such as the section name and OpenShift Container Platform version.

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support.

The **must-gather** tool enables you to collect diagnostic information about your OpenShift Container Platform cluster, including virtual machines and other data related to Red Hat OpenShift Service Mesh.

For prompt support, supply diagnostic information for both OpenShift Container Platform and Red Hat OpenShift Service Mesh.

1.1.2.1. About the must-gather tool

The **oc adm must-gather** CLI command collects the information from your cluster that is most likely needed for debugging issues, such as:

- Resource definitions
- Audit logs
- Service logs

You can specify one or more images when you run the command by including the **--image** argument. When you specify an image, the tool collects data related to that feature or product.

When you run **oc adm must-gather**, a new pod is created on the cluster. The data is collected on that pod and saved in a new directory that starts with **must-gather.local**. This directory is created in the current working directory.

1.1.2.2. Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- The OpenShift Container Platform CLI (**oc**) installed.

1.1.2.3. About collecting service mesh data

You can use the **oc adm must-gather** CLI command to collect information about your cluster, including features and objects associated with Red Hat OpenShift Service Mesh.

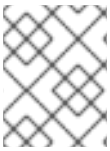
To collect Red Hat OpenShift Service Mesh data with **must-gather**, you must specify the Red Hat OpenShift Service Mesh image:

```
$ oc adm must-gather --image=registry.redhat.io/openshift-service-mesh/istio-must-gather-rhel7
```

1.1.3. Red Hat OpenShift Service Mesh supported configurations

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

- Red Hat OpenShift Container Platform version 4.x.



NOTE

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

- The deployment must be contained to a single OpenShift Container Platform cluster that is not federated.
- This release of Red Hat OpenShift Service Mesh is only available on OpenShift Container Platform x86_64.
- This release only supports configurations where all Service Mesh components are contained in the OpenShift cluster in which it operates. It does not support management of microservices that reside outside of the cluster, or in a multi-cluster scenario.
- This release only supports configurations that do not integrate external services such as virtual machines.

1.1.3.1. Supported configurations for Kiali on Red Hat OpenShift Service Mesh

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

1.1.3.2. Supported Mixer adapters

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

1.1.4. New Features

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

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

1.1.4.1. Component versions included in Red Hat OpenShift Service Mesh version 1.1.11

Component	Version
Istio	1.4.8
Jaeger	1.17.4
Kiali	1.12.7
3scale Istio Adapter	1.0.0

1.1.4.2. New features Red Hat OpenShift Service Mesh 1.1.11

This release of Red Hat OpenShift Service Mesh addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.1.4.3. New features Red Hat OpenShift Service Mesh 1.1.10

This release of Red Hat OpenShift Service Mesh addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.1.4.4. New features Red Hat OpenShift Service Mesh 1.1.9

This release of Red Hat OpenShift Service Mesh addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.1.4.5. New features Red Hat OpenShift Service Mesh 1.1.8

This release of Red Hat OpenShift Service Mesh addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.1.4.6. New features Red Hat OpenShift Service Mesh 1.1.7

This release of Red Hat OpenShift Service Mesh addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.1.4.7. New features Red Hat OpenShift Service Mesh 1.1.6

This release of Red Hat OpenShift Service Mesh addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.1.4.8. New features Red Hat OpenShift Service Mesh 1.1.5

This release of Red Hat OpenShift Service Mesh addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

This release also added support for configuring cipher suites.

1.1.4.9. New features Red Hat OpenShift Service Mesh 1.1.4

This release of Red Hat OpenShift Service Mesh addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

NOTE

There are manual steps that must be completed to address CVE-2020-8663.

1.1.4.9.1. Manual updates required by CVE-2020-8663

The fix for CVE-2020-8663**: envoy: Resource exhaustion when accepting too many connections** added a configurable limit on downstream connections. The configuration option for this limit must be configured to mitigate this vulnerability.

IMPORTANT

These manual steps are required to mitigate this CVE whether you are using the 1.1 version or the 1.0 version of Red Hat OpenShift Service Mesh.

This new configuration option is called **overload.global_downstream_max_connections**, and it is configurable as a proxy **runtime** setting. Perform the following steps to configure limits at the Ingress Gateway.

Procedure

1. Create a file named **bootstrap-override.json** with the following text to force the proxy to override the bootstrap template and load runtime configuration from disk:

```
{
  "runtime": {
    "symlink_root": "/var/lib/istio/envoy/runtime"
  }
}
```

2. Create a secret from the **bootstrap-override.json** file, replacing <SMCPnamespace> with the namespace where you created the service mesh control plane (SMCP):

```
$  oc create secret generic -n <SMCPnamespace> gateway-bootstrap --from-file=bootstrap-override.json
```

3. Update the SMCP configuration to activate the override.

Updated SMCP configuration example #1

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    gateways:
      istio-ingressgateway:
        env:
          ISTIO_BOOTSTRAP_OVERRIDE: /var/lib/istio/envoy/custom-bootstrap/bootstrap-override.json
        secretVolumes:
        - mountPath: /var/lib/istio/envoy/custom-bootstrap
          name: custom-bootstrap
          secretName: gateway-bootstrap
```

4. To set the new configuration option, create a secret that has the desired value for the **overload.global_downstream_max_connections** setting. The following example uses a value of **10000**:

```
$  oc create secret generic -n <SMCPnamespace> gateway-settings --from-literal=overload.global_downstream_max_connections=10000
```

5. Update the SMCP again to mount the secret in the location where Envoy is looking for runtime configuration:

Updated SMCP configuration example #2

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  template: default
#Change the version to "v1.0" if you are on the 1.0 stream.
  version: v1.1
  istio:
    gateways:
      istio-ingressgateway:
```

```
    env:
      ISTIO_BOOTSTRAP_OVERRIDE: /var/lib/istio/envoy/custom-bootstrap/bootstrap-override.json
    secretVolumes:
    - mountPath: /var/lib/istio/envoy/custom-bootstrap
      name: custom-bootstrap
      secretName: gateway-bootstrap
    # below is the new secret mount
    - mountPath: /var/lib/istio/envoy/runtime
      name: gateway-settings
      secretName: gateway-settings
```

1.1.4.9.2. Upgrading from Elasticsearch 5 to Elasticsearch 6

When updating from Elasticsearch 5 to Elasticsearch 6, you must delete your Jaeger instance, then recreate the Jaeger instance because of an issue with certificates. Re-creating the Jaeger instance triggers creating a new set of certificates. If you are using persistent storage the same volumes can be mounted for the new Jaeger instance as long as the Jaeger name and namespace for the new Jaeger instance are the same as the deleted Jaeger instance.

Procedure if Jaeger is installed as part of Red Hat Service Mesh

1. Determine the name of your Jaeger custom resource file:

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

   You should see something like the following:

   ```
   NAME     AGE
   jaeger   3d21h
   ```

2. Copy the generated custom resource file into a temporary directory:

   ```
   $ oc get jaeger jaeger -oyaml -n istio-system > /tmp/jaeger-cr.yaml
   ```

3. Delete the Jaeger instance:

   ```
   $ oc delete jaeger jaeger -n istio-system
   ```

4. Recreate the Jaeger instance from your copy of the custom resource file:

   ```
   $ oc create -f /tmp/jaeger-cr.yaml -n istio-system
   ```

5. Delete the copy of the generated custom resource file:

   ```
   $ rm /tmp/jaeger-cr.yaml
   ```

Procedure if Jaeger not installed as part of Red Hat Service Mesh

Before you begin, create a copy of your Jaeger custom resource file.

1. Delete the Jaeger instance by deleting the custom resource file:

   ```
   $ oc delete -f <jaeger-cr-file>
   ```

For example:

```
$ oc delete -f jaeger-prod-elasticsearch.yaml
```

2. Recreate your Jaeger instance from the backup copy of your custom resource file:

```
$ oc create -f <jaeger-cr-file>
```

3. Validate that your Pods have restarted:

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

1.1.4.10. New features Red Hat OpenShift Service Mesh 1.1.3

This release of Red Hat OpenShift Service Mesh addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.1.4.11. New features Red Hat OpenShift Service Mesh 1.1.2

This release of Red Hat OpenShift Service Mesh addresses a security vulnerability.

1.1.4.12. New features Red Hat OpenShift Service Mesh 1.1.1

This release of Red Hat OpenShift Service Mesh adds support for a disconnected installation.

1.1.4.13. New features Red Hat OpenShift Service Mesh 1.1.0

This release of Red Hat OpenShift Service Mesh adds support for Istio 1.4.6 and Jaeger 1.17.1.

1.1.4.13.1. Manual updates from 1.0 to 1.1

If you are updating from Red Hat OpenShift Service Mesh 1.0 to 1.1, you must update the **ServiceMeshControlPlane** resource to update the control plane components to the new version.

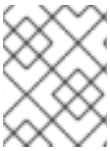
1. In the web console, click the Red Hat OpenShift Service Mesh Operator.
2. Click the **Project** menu and choose the project where your **ServiceMeshControlPlane** is deployed from the list, for example **istio-system**.
3. Click the name of your control plane, for example **basic-install**.
4. Click YAML and add a version field to the **spec:** of your **ServiceMeshControlPlane** resource. For example, to update to Red Hat OpenShift Service Mesh 1.1.0, add **version: v1.1**.

```
spec:
  version: v1.1
  ...
```

The version field specifies the version of ServiceMesh to install and defaults to the latest available version.

1.1.4.14. New features Red Hat OpenShift Service Mesh 1.0.11

This release of Red Hat OpenShift Service Mesh addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.



NOTE

There are manual steps that must be completed to address CVE-2020-8663. See instructions above.

1.1.4.15. New features Red Hat OpenShift Service Mesh 1.0.10

This release of Red Hat OpenShift Service Mesh addresses Common Vulnerabilities and Exposures (CVEs).

1.1.4.16. New features Red Hat OpenShift Service Mesh 1.0.9

This release of Red Hat OpenShift Service Mesh addresses Common Vulnerabilities and Exposures (CVEs).

1.1.4.17. New features Red Hat OpenShift Service Mesh 1.0.8

This release of Red Hat OpenShift Service Mesh addresses compatibility issues with OpenShift Container Platform 4.4. You must upgrade Red Hat OpenShift Service Mesh to 1.0.8 before you upgrade from OpenShift Container Platform 4.3 to OpenShift Container Platform 4.4.

1.1.4.18. New features Red Hat OpenShift Service Mesh 1.0.7

This release of Red Hat OpenShift Service Mesh addresses Common Vulnerabilities and Exposures (CVEs).

1.1.4.19. New features Red Hat OpenShift Service Mesh 1.0.6

This release contains internal improvements.

1.1.4.20. New features Red Hat OpenShift Service Mesh 1.0.5

This release contains internal improvements.

1.1.4.21. New features Red Hat OpenShift Service Mesh 1.0.4

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

1.1.4.22. New features Red Hat OpenShift Service Mesh 1.0.3

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

1.1.4.23. New features Red Hat OpenShift Service Mesh 1.0.2

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

1.1.4.24. New features Red Hat OpenShift Service Mesh 1.0.1

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

1.1.4.25. New features Red Hat OpenShift Service Mesh 1.0

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

Other notable changes in this release include the following:

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

1.1.5. Deprecated features

Some features available in previous releases have been deprecated or removed.

Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

1.1.5.1. Deprecated features Red Hat OpenShift Service Mesh 1.1.5

The following custom resources are deprecated in this release and will be removed in a future release.

- **Policy** - The **Policy** resource is deprecated and will be replaced by the **PeerAuthentication** resource in a future release.
- **MeshPolicy** - The **MeshPolicy** resource is deprecated and will be replaced by the **PeerAuthentication** resource in a future release.
- **v1alpha1** RBAC API -The v1alpha1 RBAC policy is deprecated by the v1beta1 **AuthorizationPolicy**. RBAC (Role Based Access Control) defines **ServiceRole** and **ServiceRoleBinding** objects.
  - **ServiceRole**
  - **ServiceRoleBinding**
- **RbacConfig** - **RbacConfig** implements the Custom Resource Definition for controlling Istio RBAC behavior.
  - **ClusterRbacConfig**(versions prior to Red Hat OpenShift Service Mesh 1.0)
  - **ServiceMeshRbacConfig** (Red Hat OpenShift Service Mesh version 1.0 and later)
- In Kiali, the **login** and **LDAP** strategies are deprecated. A future version will introduce authentication using OpenID providers.

The following components are also deprecated in this release and will be replaced by the **Istiod** component in a future release.

- **Mixer** - access control and usage policies
- **Pilot** - service discovery and proxy configuration
- **Citadel** - certificate generation
- **Galley** - configuration validation and distribution

1.1.6. Known issues

These limitations exist in Red Hat OpenShift Service Mesh:

- Red Hat OpenShift Service Mesh does not support IPv6 , as it is not supported by the upstream Istio project, nor fully supported by OpenShift.
- Graph layout - The layout for the Kiali graph can render differently, depending on your application architecture and the data to display (number of graph nodes and their interactions). Because it is difficult if not impossible to create a single layout that renders nicely for every situation, Kiali offers a choice of several different layouts. To choose a different layout, you can choose a different **Layout Schema** from the **Graph Settings** menu.
- The first time you access related services such as Jaeger and Grafana, from the Kiali console, you must accept the certificate and re-authenticate using your OpenShift Container Platform login credentials. This happens due to an issue with how the framework displays embedded pages in the console.

1.1.6.1. Service Mesh known issues

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

- Maistra-1502 As a result of CVEs fixes in version 1.0.10, the Istio dashboards are not available from the **Home Dashboard** menu in Grafana. The Istio dashboards still exist. To access them, click the **Dashboard** menu in the navigation panel and select the **Manage** tab.
- Bug 1821432 Toggle controls in OpenShift Container Platform Control Resource details page do not update the CR correctly. UI Toggle controls in the Service Mesh Control Plane (SMCP) Overview page in the OpenShift Container Platform web console sometimes update the wrong field in the resource. To update a SMCP, edit the YAML content directly or update the resource from the command line instead of clicking the toggle controls.
- Jaeger/Kiali Operator upgrade blocked with operator pending When upgrading the Jaeger or Kiali Operators with Service Mesh 1.0.x installed, the operator status shows as Pending. There is a solution in progress and a workaround. See the linked Knowledge Base article for more information.
- 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-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.

- MAISTRA-681 When the control plane has many namespaces, it can lead to performance issues.

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

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

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

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

1.1.6.2. Kiali known issues

These are the known issues in Kiali:

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

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

1.1.6.3. Jaeger known issues

These limitations exist in Jaeger:

- While Kafka publisher is included as part of Jaeger, it is not supported.

- Apache Spark is not supported.

- Only self-provisioned Elasticsearch instances are supported. External Elasticsearch instances are not supported in this release.

These are the known issues in Jaeger:

- TRACING-1166 It is not currently possible to use the Jaeger streaming strategy within a disconnected environment. When a Kafka cluster is being provisioned, it results in a error: **Failed to pull image registry.redhat.io/amq7/amq-streams-kafka-24-rhel7@sha256:f9ceca004f1b7dccb3b82d9a8027961f9fe4104e0ed69752c0bdd8078b4a1076**.

- TRACING-809 Jaeger Ingester is incompatible with Kafka 2.3. When there are two or more instances of the Jaeger Ingester and enough traffic it will continuously generate rebalancing messages in the logs. This is due to a regression in Kafka 2.3 that was fixed in Kafka 2.3.1. For more information, see Jaegertracing-1819.

1.1.7. Fixed issues

The following issues been resolved in the current release:

1.1.7.1. Service Mesh fixed issues

- MAISTRA-1352 Cert-manager Custom Resource Definitions (CRD) from the control plane installation have been removed for this release and future releases. If you have already installed Red Hat OpenShift Service Mesh, the CRDs must be removed manually if cert-manager is not being used.
  To remove the CRDs, run the following commands:

```
$ oc delete crd clusterissuers.certmanager.k8s.io
```

```
$ oc delete crd issuers.certmanager.k8s.io
```

```
$ oc delete crd certificates.certmanager.k8s.io
```

```
$ oc delete crd orders.certmanager.k8s.io
```

```
$ oc delete crd challenges.certmanager.k8s.io
```

- MAISTRA-1649 Headless services conflict when in different namespaces. When deploying headless services within different namespaces the endpoint configuration is merged and results in invalid Envoy configurations being pushed to the sidecars.

- MAISTRA-1541 Panic in kubernetesenv when the controller is not set on owner reference. If a pod has an ownerReference which does not specify the controller, this will cause a panic within the **kubernetesenv cache.go** code.

- TRACING-1300 Failed connection between Agent and Collector when using Istio sidecar. An update of the Jaeger Operator enabled TLS communication by default between a Jaeger sidecar agent and the Jaeger Collector.

- TRACING-1208 Authentication "500 Internal Error" when accessing Jaeger UI. When trying to authenticate to the UI using OAuth, I get a 500 error because oauth-proxy sidecar doesn't trust the custom CA bundle defined at installation time with the additionalTrustBundle.

- OSSM-99 Workloads generated from direct Pod without labels may crash Kiali.

- OSSM-93 IstioConfigList can't filter by two or more names.

- OSSM-92 Cancelling unsaved changes on the VS/DR YAML edit page does not cancel the changes.

- OSSM-90 Traces not available on the service details page.

- MAISTRA-1001 Closing HTTP/2 connections could lead to segmentation faults in **istio-proxy**.

- MAISTRA-932 Added the **requires** metadata to add dependency relationship between Jaeger operator and Elasticsearch operator. Ensures that when the Jaeger operator is installed, it automatically deploys the Elasticsearch operator if it is not available.

- MAISTRA-862 Galley dropped watches and stopped providing configuration to other components after many namespace deletions and re-creations.

- MAISTRA-833 Pilot stopped delivering configuration after many namespace deletions and re-creations.

- MAISTRA-684 The default Jaeger version in the **istio-operator** is 1.12.0, which does not match Jaeger version 1.13.1 that shipped in Red Hat OpenShift Service Mesh 0.12.TechPreview.

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

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

- MAISTRA-357 In OpenShift 4 Beta on AWS, it is not possible, by default, to access a TCP or HTTPS service through the ingress gateway on a port other than port 80. The AWS load balancer has a health check that verifies if port 80 on the service endpoint is active. Without a service running on port 80, the load balancer health check fails.

- MAISTRA-348 OpenShift 4 Beta on AWS does not support ingress gateway traffic on ports other than 80 or 443. If you configure your ingress gateway to handle TCP traffic with a port number other than 80 or 443, you have to use the service hostname provided by the AWS load balancer rather than the OpenShift router as a workaround.

1.1.7.2. Kiali fixed issues

- 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-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 is granted. Kiali has been updated to use this token when communicating to the Prometheus server, but the application metrics are currently failing with 403 errors.

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

- KIALI-2686 When the control plane has many namespaces, it can lead to performance issues.

1.2. 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.

1.2.1. 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

1.2.2. 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.

1.2.3. Understanding Kiali

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

1.2.3.1. 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.

1.2.3.2. Kiali architecture

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

- **Kiali application** (back end) – This component runs in the container application platform and communicates with the service mesh components, retrieves and processes data, and exposes this data to the console. The Kiali application does not need storage. When deploying the application to a cluster, configurations are set in ConfigMaps and secrets.

- **Kiali console** (front end) – The Kiali console is a web application. The Kiali application serves the Kiali console, which then queries the back end for data in order to present it to the user.

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

- **Red Hat Service Mesh** (Istio) - Istio is a Kiali requirement. Istio is the component that provides and controls the service mesh. Although Kiali and Istio can be installed separately, Kiali depends on Istio and will not work if it is not present. Kiali needs to retrieve Istio data and configurations, which are exposed through Prometheus and the cluster API.

- **Prometheus** - A dedicated Prometheus instance is included as part of the Red Hat OpenShift Service Mesh installation. When Istio telemetry is enabled, metrics data is stored in Prometheus. Kiali uses this Prometheus data to determine the mesh topology, display metrics, calculate health, show possible problems, and so on. Kiali communicates directly with Prometheus and assumes the data schema used by Istio Telemetry. Prometheus is an Istio dependency and a hard dependency for Kiali, and many of Kiali's features will not work without Prometheus.

- **Cluster API** - Kiali uses the API of the OpenShift Container Platform (cluster API) in order to fetch and resolve service mesh configurations. Kiali queries the cluster API to retrieve, for example, definitions for namespaces, services, deployments, pods, and other entities. Kiali also makes queries to resolve relationships between the different cluster entities. The cluster API is also queried to retrieve Istio configurations like virtual services, destination rules, route rules, gateways, quotas, and so on.

- **Jaeger** - Jaeger is optional, but is installed by default as part of the Red Hat OpenShift Service Mesh installation. When you install Jaeger as part of the default Red Hat OpenShift Service Mesh installation, the Kiali console includes a tab to display Jaeger's tracing data. Note that tracing data will not be available if you disable Istio's distributed tracing feature. Also note that user must have access to the namespace where the control plane is installed in order to view Jaeger data.

- **Grafana** - Grafana is optional, but is installed by default as part of the Red Hat OpenShift Service Mesh installation. When available, the metrics pages of Kiali display links to direct the user to the same metric in Grafana. Note that user must have access to the namespace where the control plane is installed in order to view links to the Grafana dashboard and view Grafana data.

1.2.3.3. 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.

1.2.4. 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.

1.2.4.1. Jaeger overview

As a service owner, you can use Jaeger to instrument your services to gather insights into your service architecture. 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.

Using Jaeger lets you perform the following functions:

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

Jaeger is based on the vendor-neutral OpenTracing APIs and instrumentation.

1.2.4.2. 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.
- **Ingester** (Ingester Service) - Jaeger can use Apache Kafka as a buffer between the collector and the actual backing storage (Elasticsearch). Ingester is a service that reads data from Kafka and writes to another storage backend (Elasticsearch).
- **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.

1.2.4.3. Jaeger features

Jaeger tracing 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 has APIs that enable it to be used as a drop-in replacement for Zipkin, but Red Hat is not supporting Zipkin compatibility in this release.

1.2.5. Next steps

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

1.3. SERVICE MESH AND ISTIO DIFFERENCES

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:

1.3.1. Red Hat OpenShift Service Mesh multitenant installation

Whereas upstream Istio takes a single tenant approach, Red Hat OpenShift Service Mesh supports multiple independent control planes within the cluster. Red Hat OpenShift Service Mesh uses a multitenant operator to manage the control plane lifecycle.

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

1.3.1.1. Multitenancy versus cluster-wide installations

The main difference between a multitenant 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**.

Every project in the **ServiceMeshMemberRoll members** list will have a **RoleBinding** for each service account associated with the 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 you require ingress from non-member projects, 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.

1.3.1.2. Cluster scoped resources

Upstream Istio has two cluster scoped resources that it relies on. The **MeshPolicy** and the **ClusterRbacConfig**. These are not compatible with a multitenant cluster and have been replaced as described below.

- *ServiceMeshPolicy* replaces MeshPolicy for configuration of control-plane-wide authentication policies. This must be created in the same project as the control plane.

- *ServicemeshRbacConfig* replaces ClusterRbacConfig for configuration of control-plane-wide role based access control. This must be created in the same project as the control plane.

1.3.2. Differences between Istio and Red Hat OpenShift Service Mesh

An installation of Red Hat OpenShift Service Mesh differs from an installation of Istio 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.

1.3.2.1. Command line tool

The command line tool for Red Hat OpenShift Service Mesh is **oc**. Red Hat OpenShift Service Mesh does not support **istioctl**.

1.3.2.2. 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 opt in to injection using an annotation without labeling projects. This method requires fewer privileges and does not conflict with other OpenShift capabilities such as builder pods. To enable automatic injection you specify the **sidecar.istio.io/inject** annotation as described in the Automatic sidecar injection section.

1.3.2.3. 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>"
```

1.3.2.4. 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.

1.3.2.5. Component modifications

- A *maistra-version* label has been added to all resources.
- All Ingress resources have been converted to OpenShift Route resources.
- Grafana, Tracing (Jaeger), and Kiali are enabled by default and exposed through OpenShift routes.
- Godebug has been removed from all templates
- The **istio-multi** ServiceAccount and ClusterRoleBinding have been removed, as well as the **istio-reader** ClusterRole.

1.3.2.6. Envoy, Secret Discovery Service, and certificates

- Red Hat OpenShift Service Mesh does not support QUIC-based services.
- Deployment of TLS certificates using the Secret Discovery Service (SDS) functionality of Istio is not currently supported in Red Hat OpenShift Service Mesh. The Istio implementation depends on a nodeagent container that uses hostPath mounts.

1.3.2.7. 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.

1.3.2.8. Routes for Istio Gateways

OpenShift routes for Istio Gateways are automatically managed in Red Hat OpenShift Service Mesh. Every time an Istio Gateway is created, updated or deleted inside the service mesh, an OpenShift route is created, updated or deleted.

A Red Hat OpenShift Service Mesh control plane component called Istio OpenShift Routing (IOR) synchronizes the gateway route. For more information see the "Automatic route creation" section.

1.3.2.8.1. Catch-all domains

Catch-all domains ("*") are not supported. If one is found in the Gateway definition, Red Hat OpenShift Service Mesh *will* create the route, but will rely on OpenShift to create a default hostname. This means that the newly created route will *not* be a catch all ("*") route, instead it will have a hostname in the form **<route-name>[-<project>].<suffix>**. Refer to the OpenShift documentation for more information about how default hostnames work and how a cluster administrator can customize it.

1.3.2.8.2. Subdomains

Subdomains (e.g.: "*.domain.com") are supported. However this ability doesn't come enabled by default in OpenShift. This means that Red Hat OpenShift Service Mesh *will* create the route with the subdomain, but it will only be in effect if OpenShift is configured to enable it.

1.3.2.8.3. Transport layer security

Transport Layer Security (TLS) is supported. This means that, if the Gateway contains a **tls** section, the OpenShift Route will be configured to support TLS.

1.3.3. 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.

1.3.4. 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.

1.4. 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:

1.4.1. 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 4.5 overview.
- Install OpenShift Container Platform 4.5.
  - Install OpenShift Container Platform 4.5 on AWS
  - Install OpenShift Container Platform 4.5 on user-provisioned AWS
  - Install OpenShift Container Platform 4.5 on bare metal
  - Install OpenShift Container Platform 4.5 on vSphere

NOTE

If you are installing Red Hat OpenShift Service Mesh on a restricted network, follow the instructions for your chosen OpenShift Container Platform infrastructure.

- 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 4.5, see About the CLI.

1.4.2. Red Hat OpenShift Service Mesh supported configurations

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

- Red Hat OpenShift Container Platform version 4.x.

NOTE

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

- The deployment must be contained to a single OpenShift Container Platform cluster that is not federated.
- This release of Red Hat OpenShift Service Mesh is only available on OpenShift Container Platform x86_64.
- This release only supports configurations where all Service Mesh components are contained in the OpenShift cluster in which it operates. It does not support management of microservices that reside outside of the cluster, or in a multi-cluster scenario.
- This release only supports configurations that do not integrate external services such as virtual machines.

1.4.2.1. Supported configurations for Kiali on Red Hat OpenShift Service Mesh

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

1.4.2.2. Supported Mixer adapters

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

1.4.3. Red Hat OpenShift Service Mesh installation activities

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

- **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.

WARNING

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

1.4.4. Next steps

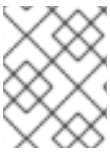
- Install Red Hat OpenShift Service Mesh in your OpenShift Container Platform environment.

1.5. 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.

NOTE

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

1.5.1. Prerequisites

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

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 1.1.11, 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.

1.5.2. Installing the Elasticsearch Operator

The default Jaeger deployment uses in-memory storage because it is designed to be installed quickly for those evaluating Jaeger, giving demonstrations, or using Jaeger in a test environment. If you plan to use Jaeger in production, you must install a persistent storage option, in this case, Elasticsearch.

Prerequisites

- Access to the OpenShift Container Platform web console.
- An account with the **cluster-admin** role.

WARNING

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

NOTE

If you have already installed the Elasticsearch Operator as part of OpenShift cluster logging, you do not need to install the Elasticsearch Operator again. The Jaeger Operator will create the Elasticsearch instance using the installed Elasticsearch Operator.

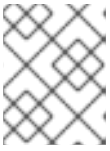
Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.
2. Navigate to **Operators → OperatorHub**.
3. Type **Elasticsearch** into the filter box to locate the Elasticsearch Operator.
4. Click the **Elasticsearch Operator** provided by Red Hat to display information about the Operator.
5. Click **Install**.
6. On the **Install Operator** page, select the **A specific namespace on the cluster**option and then select **openshift-operators-redhat** from the menu.

NOTE

The Elasticsearch installation guide says you must specify the **openshift-operators-redhat** namespace for the Elasticsearch operator for Red Hat OpenShift Service Mesh.

7. Select the **Update Channel** that matches your OpenShift Container Platform installation. For example, if you are installing on OpenShift Container Platform version 4.6, select the 4.6 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 **Install**.
10. On the **Installed Operators** page, select the **openshift-operators-redhat** project. Wait until you see that the Elasticsearch Operator shows a status of "InstallSucceeded" before continuing.

1.5.3. Installing the Jaeger Operator

To install Jaeger you use the OperatorHub to install the Jaeger Operator.

By default the Operator is installed in the **openshift-operators** project.

Prerequisites

- Access to the OpenShift Container Platform web console.
- An account with the **cluster-admin** role.

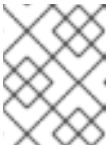
- If you require persistent storage, you must also install the Elasticsearch Operator before installing the Jaeger Operator.

WARNING

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

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.
2. Navigate to **Operators → OperatorHub**.
3. Type **Jaeger** into the filter 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 **Install Operator** 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. This will automatically update Jaeger as new versions are released. If you select a maintenance channel, for example, **1.17-stable**, you will receive bug fixes and security patches for the length of the support cycle for that version.
   - Select an Approval Strategy. You can select **Automatic** or **Manual** updates. If you choose Automatic updates for an installed Operator, when a new version of that Operator is available, the Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without human intervention. If you select Manual updates, when a newer version of an Operator is available, the OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Operator updated to the new version.



NOTE

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

8. Click **Install**.
9. On the **Subscription Overview** page, select the **openshift-operators** project. Wait until you see that the Jaeger Operator shows a status of "InstallSucceeded" before continuing.

1.5.4. Installing the Kiali Operator

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

WARNING

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

Prerequisites

- Access to the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Operators → OperatorHub**.
3. Type **Kiali** into the filter box to find the Kiali Operator.
4. Click the **Kiali Operator** provided by Red Hat to display information about the Operator.
5. Click **Install**.
6. On the **Install Operator** 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 **Install**.
10. The **Installed Operators** page displays the Kiali Operator's installation progress.

1.5.5. Installing the Red Hat OpenShift Service Mesh Operator

Prerequisites

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

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Operators → OperatorHub**.
3. Type **Red Hat OpenShift Service Mesh** into the filter box to find the Red Hat OpenShift Service Mesh Operator.
4. Click the Red Hat OpenShift Service Mesh Operator to display information about the Operator.
5. On the **Install Operator** 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 **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 **Install**.
10. The **Installed Operators** page displays the Red Hat OpenShift Service Mesh Operator's installation progress.

1.5.6. Deploying the Red Hat OpenShift Service Mesh control plane

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

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

1.5.6.1. Deploying the control plane from the web console

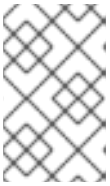
Follow this procedure to deploy the Red Hat OpenShift Service Mesh control plane by using the web console.

Prerequisites

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

Procedure

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



NOTE

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

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

1.5.6.2. Deploying the control plane from the CLI

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

Prerequisites

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

- Access to the OpenShift CLI (**oc**).

Procedure

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

```
$ oc login https://{HOSTNAME}:6443
```

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

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

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

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

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

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

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

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

```
NAME          READY
basic-install   True
```

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

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

You should see output similar to the following:

Example output

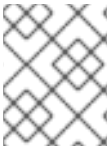
```
NAME                                READY   STATUS            RESTARTS   AGE
grafana-7bf5764d9d-2b2f6            2/2     Running           0          28h
istio-citadel-576b9c5bbd-z84z4      1/1     Running           0          28h
istio-egressgateway-5476bc4656-r4zdv    1/1     Running          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

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

1.5.7. Creating the Red Hat OpenShift Service Mesh member roll

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

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



NOTE

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

1.5.7.1. Creating the member roll from the web console

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

Prerequisites

- An installed, verified Red Hat OpenShift Service Mesh Operator.
- Location of the installed **ServiceMeshControlPlane**.
- List of existing projects to add to the service mesh.

Procedure

1. If you don't already have projects for your mesh, or you are starting from scratch, create a project. It must be different from **istio-system**.
   a. Navigate to **Home → Projects**.
   b. Enter a name in the **Name** field.
   c. Click **Create**.
2. Log in to the OpenShift Container Platform web console.
3. Navigate to **Operators → Installed Operators**.
4. Click the **Project** menu and choose the project where your **ServiceMeshControlPlane** is deployed from the list, for example **istio-system**.
5. Click the Red Hat OpenShift Service Mesh Operator.
6. Click the **All Instances** tab.
7. Click **Create New**, and then select **Create Istio Service Mesh Member Roll**

NOTE

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

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

9. Click **Create** to save the Service Mesh Member Roll.

1.5.7.2. Creating the member roll from the CLI

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

Prerequisites

- An installed, verified Red Hat OpenShift Service Mesh Operator.
- Location of the installed **ServiceMeshControlPlane**.
- List of projects to add to the service mesh.
- Access to the OpenShift CLI (**oc**).

Procedure

1. Log in to the OpenShift Container Platform CLI.

```
$ oc login
```

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

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

Example servicemeshmemberroll-default.yaml

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

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

1.5.7.3. Creating the Red Hat OpenShift Service Mesh members

ServiceMeshMember resources can be created by service mesh users who don't have privileges to add members to the **ServiceMeshMemberRoll** directly. While project administrators are automatically given permission to create the **ServiceMeshMember** resource in their project, they cannot point it to any **ServiceMeshControlPlane** until the service mesh administrator explicitly grants access to the service mesh. Administrators can grant users permissions to access the mesh by granting them the **mesh-user** user role, for example:

```
$ oc policy add-role-to-user -n <control-plane-namespace> --role-namespace <control-plane-namespace> mesh-user <user-name>.
```

Administrators can modify the **mesh user** role binding in the control plane project to specify the users and groups that are granted access. The **ServiceMeshMember** adds the project to the **ServiceMeshMemberRoll** within the control plane project it references.

```
apiVersion: maistra.io/v1
kind: ServiceMeshMember
metadata:
  name: default
spec:
  controlPlaneRef:
    namespace: control-plane-namespace
    name: minimal-install
```

The mesh-users role binding is created automatically after the administrator creates the **ServiceMeshControlPlane** resource. An administrator can use the following command to add a role to a user.

```
$ oc policy add-role-to-user
```

The administrator can also create the **mesh-user** role binding before the administrator creates the **ServiceMeshControlPlane** resource. For example, the administrator can create it in the same **oc apply** operation as the **ServiceMeshControlPlane** resource.

This example adds a role binding for **alice**:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  namespace: control-plane-namespace
  name: mesh-users
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: mesh-user
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: alice
```

1.5.8. Adding or removing projects from the service mesh

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

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

1.5.8.1. Modifying the member roll from the web console

Prerequisites

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

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Operators → Installed Operators**.
3. Click the **Project** menu and choose the project where your **ServiceMeshControlPlane** is deployed from the list, for example **istio-system**.
4. Click the Red Hat OpenShift Service Mesh Operator.
5. Click the **Istio Service Mesh Member Roll**tab.
6. Click the **default** link.
7. Click the YAML tab.
8. Modify the YAML to add or remove projects as members. You can add any number of projects, but a project can only belong to **one ServiceMeshMemberRoll** resource.
9. Click **Save**.
10. Click **Reload**.

1.5.8.2. Modifying the member roll from the CLI

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

Prerequisites

- An installed, verified Red Hat OpenShift Service Mesh Operator.
- An existing **ServiceMeshMemberRoll** resource.
- Name of the project with the **ServiceMeshMemberRoll** resource.

- Names of the projects you want to add or remove from the mesh.
- Access to the OpenShift CLI (**oc**).

Procedure

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

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

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

Example servicemeshmemberroll-default.yaml

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

1.5.9. Manual updates

If you choose to update manually, the Operator Lifecycle Manager (OLM) controls the installation, upgrade, and role-based access control (RBAC) of Operators in a cluster. OLM runs by default in OpenShift Container Platform. OLM uses CatalogSources, which use the Operator Registry API, to query for available Operators as well as upgrades for installed Operators.

- For more information about how OpenShift Container Platform handled upgrades, refer to the Operator Lifecycle Manager documentation.

1.5.9.1. Updating your application pods

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

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

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

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

1.5.10. Next steps

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

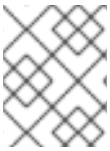
1.6. CUSTOMIZING THE RED HAT OPENSHIFT SERVICE MESH INSTALLATION

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

1.6.1. Prerequisites

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

1.6.2. Red Hat OpenShift Service Mesh custom resources



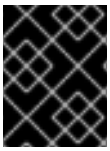
NOTE

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 1.1.11 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).

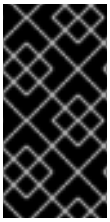
Example istio-installation.yaml

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

mixer:
  policy:
    autoscaleEnabled: false

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

pilot:
  autoscaleEnabled: false
  traceSampling: 100

kiali:
  enabled: true

grafana:
  enabled: true

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

1.6.3. 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.

1.6.3.1. Istio global example

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

NOTE

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

Example global parameters

```
istio:
  global:
    tag: 1.1.0
    hub: registry.redhat.io/openshift-service-mesh/
    proxy:
      resources:
        requests:
          cpu: 10m
          memory: 128Mi
        limits:
    mtls:
      enabled: false
    disablePolicyChecks: true
    policyCheckFailOpen: false
    imagePullSecrets:
      - MyPullSecret
```

Table 1.1. Global parameters

Parameter	Description	Values	Default value
disablePolicyChecks	This parameter enables/disables policy checks.	**true**/**false**	**true**
policyCheckFailOpen	This parameter indicates whether traffic is allowed to pass through to the Envoy sidecar when the Mixer policy service cannot be reached.	**true**/**false**	**false**

Parameter	Description	Values	Default value
tag	The tag that the Operator uses to pull the Istio images.	A valid container image tag.	**1.1.0**
hub	The hub that the Operator uses to pull Istio images.	A valid image repository.	**maistra/** or **registry.redhat.io/openshift-service-mesh/**
mtls	This parameter controls whether to enable/disable 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 1.2. Proxy parameters

Type	Parameter	Description	Values	Default value
Resources	**cpu**	The amount of CPU resources requested for Envoy proxy.	CPU resources, specified in cores or millicores (for example, 200m, 0.5, 1) based on your environment's configuration.	**10m**
	memory	The amount of memory requested for Envoy proxy	Available memory in bytes(for example, 200Ki, 50Mi, 5Gi) based on your environment's configuration.	**1024Mi**

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

1.6.3.2. Istio gateway configuration

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

Example gateway parameters

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

Table 1.3. Istio Gateway parameters

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

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

1.6.3.3. Automatic route creation

OpenShift routes for Istio Gateways are automatically managed in Red Hat OpenShift Service Mesh. Every time an Istio Gateway is created, updated or deleted inside the service mesh, an OpenShift route is created, updated or deleted.

1.6.3.3.1. Enabling Automatic Route Creation

A Red Hat OpenShift Service Mesh control plane component called Istio OpenShift Routing (IOR) synchronizes the gateway route. Enable IOR as part of the control plane deployment.

If the Gateway contains a TLS section, the OpenShift Route will be configured to support TLS.

1. In the **ServiceMeshControlPlane** resource, add the **ior_enabled** parameter and set it to **true**. For example, see the following resource snippet:

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

1.6.3.3.2. Subdomains

Red Hat OpenShift Service Mesh creates the route with the subdomain, but OpenShift Container Platform must be configured to enable it. Subdomains, for example ***.domain.com**, are supported but not by default.

If the following gateway is created:

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

Then, the following OpenShift Routes are created automatically. You can check that the routes are created with the following command.

```
$ oc -n <your-control-plane-namespace> get routes
```

Expected output

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

If the gateway is deleted, Red Hat OpenShift Service Mesh deletes the routes. However, routes created manually are never modified by Red Hat OpenShift Service Mesh.

Cluster administrators can refer to Using wildcard routes for instructions on how to enable subdomains.

1.6.3.4. Istio Mixer configuration

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

Example mixer parameters

```
mixer:
  enabled: true
  policy:
    autoscaleEnabled: false
  telemetry:
    autoscaleEnabled: false
    resources:
    requests:
      cpu: 10m
      memory: 128Mi
      limits:
```

Table 1.4. Istio Mixer policy parameters

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

Table 1.5. 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.	**10m**
	memory	The amount of memory requested for Mixer telemetry.	Available memory in bytes (for example, 200Ki, 50Mi, 5Gi) based on your environment's configuration.	**128Mi**
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 (for example, 200Ki, 50Mi, 5Gi) based on your environment's configuration.	**4G**

1.6.3.5. Istio Pilot configuration

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

Example pilot parameters

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

Table 1.6. Istio Pilot parameters

Parameter	Description	Values	Default value

Parameter	Description	Values	Default value
cpu	The percentage of CPU resources requested for Pilot.	CPU resources in millicores based on your environment's configuration.	**10m**
memory	The amount of memory requested for Pilot.	Available memory in bytes (for example, 200Ki, 50Mi, 5Gi) based on your environment's configuration.	**128Mi**
autoscaleEnabled	This parameter enables/disables autoscaling. Disable this for small environments.	**true**/**false**	**true**
traceSampling	This value controls how often random sampling occurs. **Note:** Increase for development or testing.	A valid percentage.	**1.0**

1.6.4. Configuring Kiali

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

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

Example Kiali parameters

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

Table 1.7. Kiali parameters

Parameter	Description	Values	Default value
enabled	This parameter enables/disables Kiali. Kiali is enabled by default.	**true**/**false**	**true**

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

1.6.4.1. Configuring Kiali for Grafana

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

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

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

Additional Grafana parameters

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

1.6.4.2. Configuring Kiali for Jaeger

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

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

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

Additional Jaeger parameters

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

1.6.5. Configuring Jaeger

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

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

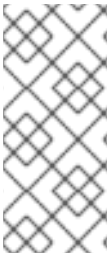
Default **all-in-one** Jaeger parameters

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

Table 1.8. Jaeger parameters

Parameter	Description	Values	Default value
`tracing` `  enabled`	This parameter enables/disables tracing in Service Mesh. Jaeger is installed by default.	**true**/**false**	**true**

Parameter	Description	Values	Default value
jaeger template	This parameter specifies which Jaeger deployment strategy to use.	• **all-in-one**- For development, testing, demonstrations, and proof of concept. • **production-elasticsearch** - For production use.	**all-in-one**



NOTE

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

1.6.5.1. Configuring Elasticsearch

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

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

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

Default "production" Jaeger parameters with Elasticsearch

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

```
    template: production-elasticsearch
    elasticsearch:
      nodeCount: 3
      redundancyPolicy:
      resources:
        requests:
          cpu: "1"
          memory: "16Gi"
        limits:
          cpu: "1"
          memory: "16Gi"
```

Table 1.9. Elasticsearch parameters

Parameter	Description	Values	Default Value	Examples
`tracing: enabled`	This parameter enables/disables tracing in Service Mesh. Jaeger is installed by default.	**true**/**false**	**true**	
`ingress: enabled`	This parameter enables/disables ingress for Jaeger.	**true**/**false**	**true**	
`jaeger template`	This parameter specifies which Jaeger deployment strategy to use.	**all-in-one**/**production-elasticsearch**	**all-in-one**	
`elasticsearch: nodeCount`	Number of Elasticsearch nodes to create.	Integer value.	1	Proof of concept = 1, Minimum deployment =3
`requests: cpu`	Number of central processing units for requests, based on your environment's configuration.	Specified in cores or millicores (for example, 200m, 0.5, 1).	1Gi	Proof of concept = 500m, Minimum deployment =1
`requests: memory`	Available memory for requests, based on your environment's configuration.	Specified in bytes (for example, 200Ki, 50Mi, 5Gi).	500m	Proof of concept = 1Gi, Minimum deployment = 16Gi*

Parameter	Description	Values	Default Value	Examples
limits: cpu	Limit on number of central processing units, based on your environment's configuration.	Specified in cores or millicores (for example, 200m, 0.5, 1).		Proof of concept = 500m, Minimum deployment =1
limits: memory	Available memory limit based on your environment's configuration.	Specified in bytes (for example, 200Ki, 50Mi, 5Gi).		Proof of concept = 1Gi, Minimum deployment = 16Gi*
	* Each Elasticsearch node can operate with a lower memory setting though this is **not** recommended for production deployments. For production use, you should have no less than 16Gi allocated to each pod by default, but preferably allocate as much as you can, up to 64Gi per pod.			

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.
2. Navigate to **Operators → Installed Operators**.
3. Click the Red Hat OpenShift Service Mesh Operator.
4. Click the **Istio Service Mesh Control Plane** tab.
5. Click the name of your control plane file, for example, **basic-install**.
6. Click the **YAML** tab.
7. Edit the Jaeger parameters, replacing the default **all-in-one** template with parameters for the **production-elasticsearch** template, modified for your use case. Ensure that the indentation is correct.
8. Click **Save**.
9. Click **Reload**. OpenShift Container Platform redeploys Jaeger and creates the Elasticsearch resources based on the specified parameters.

1.6.5.2. 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 1.10. 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

For more information about configuring Elasticsearch with OpenShift Container Platform, see Configuring the log store.

1.6.6. 3scale configuration

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

Example 3scale parameters

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

Table 1.11. 3scale parameters

Parameter	Description	Values	Default value

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**
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**

Parameter	Description	Values	Default value
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

1.6.7. Next steps

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

1.7. 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.

1.7.1. Prerequisites

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

1.7.2. Creating control plane templates

You can create reusable configurations with **ServiceMeshControlPlane** templates. Individual users can extend the templates they 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**.

1.7.2.1. Creating the ConfigMap

Follow this procedure to create the ConfigMap.

Prerequisites

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

Procedure

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

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

3. Locate the Operator ClusterServiceVersion name.

```
$ oc get clusterserviceversion -n openshift-operators | grep 'Service Mesh'
```

Example output

```
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
...
```

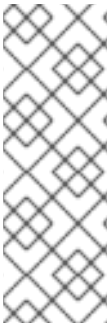
6. Save your changes and exit the editor.

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
```

1.7.3. 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.

1.7.3.1. Setting environment variables on the proxy in applications through annotations

You can set environment variables on the sidecar proxy for applications by adding pod annotations in the deployment in the **injection-template.yaml** file. The environment variables are injected to the sidecar.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: resource
spec:
  replicas: 7
  selector:
    matchLabels:
      app: resource
  template:
    metadata:
```

```
    annotations:
      sidecar.maistra.io/proxyEnv: "{ \"maistra_test_env\": \"env_value\", \"maistra_test_env_2\":
\"env_value_2\" }"
```

WARNING

maistra.io/ labels and annotations should never be included in user-created resources, because they indicate that the resources are generated and managed by the Operator. If you are copying content from an Operator-generated resource when creating your own resources, do not include labels or annotations that start with **maistra.io/** or your resource will be overwritten or deleted by the Operator during the next reconciliation.

1.7.3.2. 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: apps/v1
kind: Deployment
metadata:
  labels:
    app: sleep
  name: sleep
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sleep
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true"
      labels:
```

```
      app: sleep
  spec:
    containers:
    - name: sleep
      image: tutum/curl
      command: ["/bin/sleep","infinity"]
      imagePullPolicy: IfNotPresent
```

3. Save the configuration file.

1.7.4. Updating Mixer policy enforcement

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

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**.

1.7.4.1. Setting the correct network policy

Service Mesh creates network policies in the control plane and member namespaces to allow traffic between them. Before you deploy, consider the following conditions to ensure the services in your mesh that were previously exposed through an OpenShift Container Platform route.

- Traffic into the mesh must always go through the ingress-gateway for Istio to work properly.
- Deploy services external to the mesh in separate namespaces that are not in any mesh.
- Non-mesh services that need to be deployed within a service mesh enlisted namespace should label their deployments **maistra.io/expose-route: "true"**, which ensures OpenShift Container Platform routes to these services still work.

1.7.5. Bookinfo example 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.

1.7.5.1. 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 1.1.9.

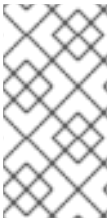
WARNING

The Bookinfo example application allows you to test your Red Hat OpenShift Service Mesh 1.1.9 installation on OpenShift Container Platform.

Red Hat does not provide support for the Bookinfo application.

Prerequisites:

- OpenShift Container Platform 4.1 or higher installed.
- Red Hat OpenShift Service Mesh 1.1.9 installed.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as **oc**.



NOTE

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

Procedure

1. Log in to the OpenShift Container Platform web console as a user with cluster-admin rights.
2. Click to **Home → Projects**.
3. Click **Create Project**.
4. Enter **bookinfo** as the **Project Name**, enter a **Display Name**, and enter a **Description**, then click **Create**.
   - Alternatively, you can run this command from the CLI to create the **bookinfo** project.

     ```
     $ oc new-project bookinfo
     ```
5. Click **Operators → Installed Operators**.
6. Click the **Project** menu and use the control plane namespace. In this example, use **istio-system**.
7. Click the **Red Hat OpenShift Service Mesh** Operator.
8. Click the **Istio Service Mesh Member Roll** link.
   a. If you have already created a Istio Service Mesh Member Roll, click the name, then click the YAML tab to open the YAML editor.
   b. If you have not created a Istio Service Mesh Member Roll, click **Create Service Mesh Member Roll**.

NOTE

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

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

Bookinfo ServiceMeshMemberRoll example

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

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

     ```
     $ oc -n <control_plane_project> patch --type='json' smmr default -p '[{"op": "add", "path": "/spec/members", "value":["""bookinfo"""]}]'
     ```
10. Click **Create** to save the updated Service Mesh Member Roll.

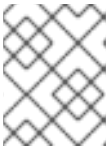
11. From the CLI, deploy the Bookinfo application in the `*bookinfo*` project by applying the **bookinfo.yaml** file:

```
$ oc apply -n bookinfo -f https://raw.githubusercontent.com/Maistra/istio/maistra-1.1/samples/bookinfo/platform/kube/bookinfo.yaml
```

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

```
$ oc apply -n bookinfo -f https://raw.githubusercontent.com/Maistra/istio/maistra-1.1/samples/bookinfo/networking/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. In this example, the control plane project is **istio-system**.

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

1.7.5.2. 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/Maistra/istio/maistra-1.1/samples/bookinfo/networking/destination-rule-all.yaml
```

   - If you enabled mutual TLS:

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

1.7.5.3. 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 1.1.9 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
```

1.7.5.4. 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 1.1.9 installed.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as **oc**.

1.7.5.4.1. 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
```

1.7.5.4.2. Remove the Bookinfo project from the Service Mesh member roll

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Click **Operators → Installed Operators**.
3. Click the **Project** menu and choose **openshift-operators** from the list.

4. Click the **Istio Service Mesh Member Roll**link under **Provided APIS** for the **Red Hat OpenShift Service Mesh** Operator.

5. Click the **ServiceMeshMemberRoll** menu  and select **Edit Service Mesh Member Roll**

6. Edit the default Service Mesh Member Roll YAML and remove **bookinfo** from the **members** list.

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

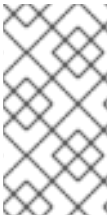
     ```
     $ oc -n <control_plane_project> patch --type='json' smmr default -p '[{"op": "remove", "path": "/spec/members", "value":[""bookinfo""]}]'
     ```

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

1.7.6. Generating example traces and analyzing trace data

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

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



NOTE

The Bookinfo example application allows you to test your Red Hat OpenShift Service Mesh 1.1.9 installation on OpenShift Container Platform.

Red Hat does not provide support for the Bookinfo application.

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

Prerequisites:

- OpenShift Container Platform 4.1 or higher installed.
- Red Hat OpenShift Service Mesh 1.1.9 installed.
- Jaeger enabled during the installation.
- Bookinfo example application installed.

Procedure

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

2. The installation process creates a route to access the Jaeger console.

   a. In the OpenShift Container Platform console, navigate to **Networking → Routes** and search for the Jaeger route, which is the URL listed under **Location**.

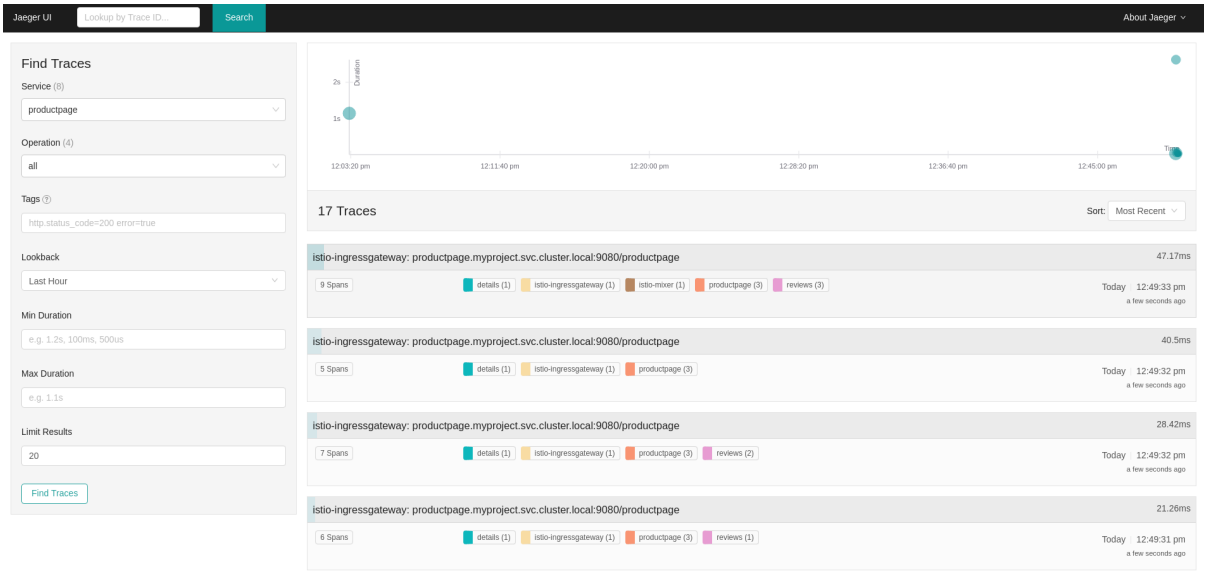
   b. Use the CLI to query for details of the route:

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

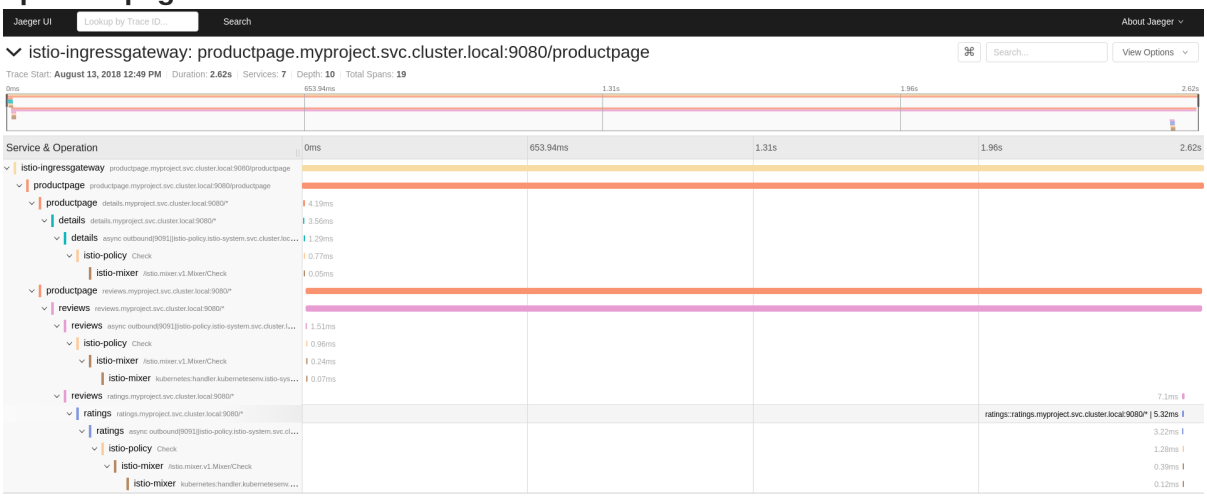
3. Launch a browser and navigate to **https://<JAEGER_URL>**.

4. If necessary, log in using the same user name and password as you use to access the OpenShift Container Platform console.

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



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



The trace in the previous figure consists of a few nested spans, each corresponding to a Bookinfo Service call, all performed in response to a `/**productpage** request. Overall processing time was 2.62s, with the **details** Service taking 3.56ms, the **reviews** Service taking 2.6s, and the

ratings Service taking 5.32ms. Each of the calls to remote Services is represented by a client-side and server-side span. For example, the **details** client-side span is labeled **productpage details.myproject.svc.cluster.local:9080**. The span nested underneath it, labeled **details details.myproject.svc.cluster.local:9080**, corresponds to the server-side processing of the request. The trace also shows calls to **istio-policy**, which reflect authorization checks made by Istio.

1.8. DATA VISUALIZATION AND OBSERVABILITY

You can view your application's topology, health and metrics in the Kiali console. If your service is having issues, the Kiali console offers ways to visualize the data flow through your service. You can view insights about the mesh components at different levels, including abstract applications, services, and workloads. It also provides an interactive graph view of your namespace in real time.

You can observe the data flow through your application if you have one installed. If you don't have your own application installed, you can see how observability works in Red Hat OpenShift Service Mesh by installing the Bookinfo sample application.

After installing the Bookinfo sample application, send traffic to the mesh. Enter the following command a few times:

```
$ curl http://$GATEWAY_URL/productpage
```

If your sample application is configured correctly, this command simulates a user visiting the **productpage** microservice of the application.

1.8.1. Accessing the Kiali console

To access the console, in the menu bar, click the **Application launcher** > **Kiali**.

1. In the OpenShift Container Platform menu bar, click the **Application launcher** > **Kiali**.
2. Log in to the Kiali console with the same user name and password as you use to access the OpenShift Container Platform console.
3. Select the project for your service in the **Namespace** field. If you have installed the Bookinfo example, select **bookinfo**.

Procedure from the command line

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

   ```
   $ oc get routes
   ```

   In the output on the **kiali** line, use the URL in the HOST/PORT column to open the Kiali console. Log in to the Kiali console with the same user name and password as you use to access the OpenShift Container Platform console. Select the project for your service in the **Namespace** field.

When you first log in, you see the Overview page which displays all the namespaces in your mesh that you have permission to view.

1.8.2. Visualizing your service

The Kiali operator works with the telemetry data gathered in Red Hat OpenShift Service Mesh to provide graphs and real time network diagrams of the applications, services, and workloads in your namespace.

The Overview page displays all the namespaces that have services in your mesh. You can reveal deeper insights about the data traveling through your Service mesh or help identify problems with services or workloads in your service mesh with the following graphs and visualizations.

1.8.2.1. Namespace graphs

The namespace graph is a map of the services, deployments and workflows in your namespace and arrows that show how data flows through them. To view a namespace graph:

1. Click **Graph** in the main navigation.
2. Select **bookinfo** from the **Namespace** menu.

If your application uses version tags, like the Bookinfo sample application, you can see a Version graph. Select a graph from the Graph Type drop down menu. There are several graphs to choose from:

- The App graph shows an aggregate workload for all applications that are labeled the same.
- The Versioned App graph shows a node for each version of an app. All versions of an app are grouped together.
- The Workload graph shows a node for each workload in your service mesh. This graph does not require you to use the app and version labels. If your app does not use version labels, use this the graph.
- The Service graph shows a node for each service in your mesh but excludes all apps and workloads from the graph. It provides a high level view and aggregates all traffic for defined services.

To view a summary of metrics, select any node or edge in the graph to display its metric details in the summary details panel.

1.9. CUSTOMIZING SECURITY IN A SERVICE MESH

If your service mesh application is constructed with a complex array of microservices, you can use Red Hat OpenShift Service Mesh to customize the security of the communication between those services. The infrastructure of OpenShift Container Platform along with the traffic management features of Service Mesh can help you manage the complexity of your applications and provide service and identity security for microservices.

1.9.1. Enabling mutual Transport Layer Security (mTLS)

Mutual Transport Layer Security (mTLS) is a protocol where two parties authenticate each other at the same time. It is the default mode of authentication in some protocols (IKE, SSH) and optional in others (TLS).

MTLS can be used without changes to the application or service code. The TLS is handled entirely by the service mesh infrastructure and between the two sidecar proxies.

By default, Red Hat OpenShift Service Mesh is set to permissive mode, where the sidecars in Service Mesh accept both plain-text traffic and connections that are encrypted using mTLS. If a service in your mesh is communicating with a service outside the mesh, strict mTLS could break communication

between those services. Use permissive mode while you migrate your workloads to Service Mesh.

1.9.1.1. Enabling strict mTLS across the mesh

If your workloads do not communicate with services outside your mesh and communication will not be interrupted by only accepting encrypted connections, you can enable mTLS across your mesh quickly. Set **spec.istio.global.mtls.enabled** to **true** in your **ServiceMeshControlPlane** resource. The operator creates the required resources.

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    global:
      mtls:
        enabled: true
```

1.9.1.1.1. Configuring sidecars for incoming connections for specific services

You can also configure mTLS for individual services or namespaces by creating a policy.

```
apiVersion: "authentication.istio.io/v1alpha1"
kind: "Policy"
metadata:
  name: "default"
  namespace: <NAMESPACE>
spec:
  peers:
  - mtls: {}
```

1.9.1.2. Configuring sidecars for outgoing connections

Create a destination rule to configure Service Mesh to use mTLS when sending requests to other services in the mesh.

```
apiVersion: "networking.istio.io/v1alpha3"
kind: "DestinationRule"
metadata:
  name: "default"
  namespace: <CONTROL_PLANE_NAMESPACE>
spec:
  host: "*.local"
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
```

1.9.1.3. Setting the minimum and maximum protocol versions

If your environment has specific requirements for encrypted traffic in your service mesh, you can control the cryptographic functions that are allowed by setting the **spec.istio.global.tls.minProtocolVersion** or **spec.istio.global.tls.maxProtocolVersion** in your **ServiceMeshControlPlane** resource. Those values, configured in your control plane resource, define the minimum and maximum TLS version used by mesh components when communicating securely over TLS.

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    global:
      tls:
        minProtocolVersion: TLSv1_0
```

The default is **TLS_AUTO** and does not specify a version of TLS.

Table 1.12. Valid values

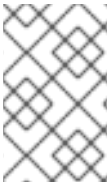
Value	Description
TLS_AUTO	default
TLSv1_0	TLS version 1.0
TLSv1_1	TLS version 1.1
TLSv1_2	TLS version 1.2
TLSv1_3	TLS version 1.3

1.9.2. Configuring cipher suites and ECDH curves

Cipher suites and Elliptic-curve Diffie–Hellman (ECDH curves) can help you secure your service mesh. You can define a comma separated list of cipher suites using **spec.istio.global.tls.cipherSuites** and ECDH curves using **spec.istio.global.tls.ecdhCurves** in your **ServiceMeshControlPlane** resource. If either of these attributes are empty, then the default values are used.

The **cipherSuites** setting is effective if your service mesh uses TLS 1.2 or earlier. It has no effect when negotiating with TLS 1.3.

Set your cipher suites in the comma separated list in order of priority. For example, **ecdhCurves: CurveP256, CurveP384** sets **CurveP256** as a higher priority than **CurveP384**.



NOTE

You must include either **TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256** or **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256** when you configure the cipher suite. HTTP/2 support requires at least one of these cipher suites.

The supported cipher suites are:

- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA

The supported ECDH Curves are:

- CurveP256
- CurveP384
- CurveP521
- X25519

1.9.3. Adding an external certificate authority key and certificate

By default, Red Hat OpenShift Service Mesh generates self-signed root certificate and key, and uses them to sign the workload certificates. You can also use the user-defined certificate and key to sign workload certificates, with user-defined root certificate. This task demonstrates an example to plug certificates and key into Service Mesh.

Prerequisites

- You must have installed Red Hat OpenShift Service Mesh with mutual TLS enabled to configure certificates.
- This example uses the certificates from the Maistra repository. For production, use your own certificates from your certificate authority.
- You must deploy the Bookinfo sample application to verify the results with these instructions.

1.9.3.1. Adding an existing certificate and key

To use an existing signing (CA) certificate and key, you must create a chain of trust file that includes the CA certificate, key, and root certificate. You must use the following exact file names for each of the corresponding certificates. The CA certificate is called **ca-cert.pem**, the key is **ca-key.pem**, and the root certificate, which signs **ca-cert.pem**, is called **root-cert.pem**. If your workload uses intermediate certificates, you must specify them in a **cert-chain.pem** file.

Add the certificates to Service Mesh by following these steps. Save the example certificates from the Maistra repository locally and replace **<path>** with the path to your certificates.

1. Create a secret **cacert** that includes the input files **ca-cert.pem**, **ca-key.pem**, **root-cert.pem** and **cert-chain.pem**.

```
$ oc create secret generic cacerts -n istio-system --from-file=<path>/ca-cert.pem \
    --from-file=<path>/ca-key.pem --from-file=<path>/root-cert.pem \
    --from-file=<path>/cert-chain.pem
```

2. In the **ServiceMeshControlPlane** resource set **global.mtls.enabled** to **true** and **security.selfSigned** set to **false**. Service Mesh reads the certificates and key from the secret-mount files.

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    global:
      mtls:
        enabled: true
    security:
      selfSigned: false
```

3. To make sure the workloads add the new certificates promptly, delete the secrets generated by Service Mesh, named **istio.***. In this example, **istio.default**. Service Mesh issues new certificates for the workloads.

```
$ oc delete secret istio.default
```

1.9.3.2. Verifying your certificates

Use the Bookinfo sample application to verify your certificates are mounted correctly. First, retrieve the mounted certificates. Then, verify the certificates mounted on the pod.

1. Store the pod name in the variable **RATINGSPOD**.

```
$ RATINGSPOD=`oc get pods -l app=ratings -o jsonpath='{.items[0].metadata.name}'`
```

2. Run the following commands to retrieve the certificates mounted on the proxy.

```
$ oc exec -it $RATINGSPOD -c istio-proxy -- /bin/cat /etc/certs/root-cert.pem > /tmp/pod-root-cert.pem
```

The file **/tmp/pod-root-cert.pem** contains the root certificate propagated to the pod.

```
$ oc exec -it $RATINGSPOD -c istio-proxy -- /bin/cat /etc/certs/cert-chain.pem > /tmp/pod-cert-chain.pem
```

The file **/tmp/pod-cert-chain.pem** contains the workload certificate and the CA certificate propagated to the pod.

3. Verify the root certificate is the same as the one specified by the Operator. Replace **<path>** with the path to your certificates.

```
$ openssl x509 -in <path>/root-cert.pem -text -noout > /tmp/root-cert.crt.txt
```

```
$ openssl x509 -in /tmp/pod-root-cert.pem -text -noout > /tmp/pod-root-cert.crt.txt
```

```
$ diff /tmp/root-cert.crt.txt /tmp/pod-root-cert.crt.txt
```

Expect the output to be empty.

4. Verify the CA certificate is the same as the one specified by Operator. Replace **<path>** with the path to your certificates.

```
$ sed '0,/^-----END CERTIFICATE-----/d' /tmp/pod-cert-chain.pem > /tmp/pod-cert-chain-ca.pem
```

```
$ openssl x509 -in <path>/ca-cert.pem -text -noout > /tmp/ca-cert.crt.txt
```

```
$ openssl x509 -in /tmp/pod-cert-chain-ca.pem -text -noout > /tmp/pod-cert-chain-ca.crt.txt
```

```
$ diff /tmp/ca-cert.crt.txt /tmp/pod-cert-chain-ca.crt.txt
```

Expect the output to be empty.

5. Verify the certificate chain from the root certificate to the workload certificate. Replace **<path>** with the path to your certificates.

```
$ head -n 21 /tmp/pod-cert-chain.pem > /tmp/pod-cert-chain-workload.pem
```

```
$ openssl verify -CAfile <(cat <path>/ca-cert.pem <path>/root-cert.pem) /tmp/pod-cert-chain-workload.pem
```

Example output

```
/tmp/pod-cert-chain-workload.pem: OK
```

1.9.3.3. Removing the certificates

To remove the certificates you added, follow these steps.

1. Remove the secret **cacerts**.

```
$ oc delete secret cacerts -n istio-system
```

2. Redeploy Service Mesh with a self-signed root certificate in the **ServiceMeshControlPlane** resource.

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    global:
      mtls:
        enabled: true
    security:
      selfSigned: true
```

1.10. TRAFFIC MANAGEMENT

You can control the flow of traffic and API calls between services in Red Hat OpenShift Service Mesh. For example, some services in your service mesh may need to communicate within the mesh and others may need to be hidden. Manage the traffic to hide specific backend services, expose services, create testing or versioning deployments, or add a security layer on a set of services.

This guide references the Bookinfo sample application to provide examples of routing in an example application. Install the Bookinfo application to learn how these routing examples work.

1.10.1. Routing and managing traffic

Configure your service mesh by adding your own traffic configuration to Red Hat OpenShift Service Mesh with a custom resource definitions in a YAML file.

1.10.1.1. Traffic management with virtual services

You can route requests dynamically to multiple versions of a microservice through Red Hat OpenShift Service Mesh with a virtual service. With virtual services, you can:

- Address multiple application services through a single virtual service. If your mesh uses Kubernetes, for example, you can configure a virtual service to handle all services in a specific namespace. Mapping a single virtual service to many services is particularly useful in facilitating turning a monolithic application into a composite service built out of distinct microservices without requiring the consumers of the service to adapt to the transition.
- Configure traffic rules in combination with gateways to control ingress and egress traffic.

1.10.1.1.1. Configuring virtual services

Requests are routed to a services within a service mesh with virtual services. Each virtual service consists of a set of routing rules that are evaluated in order. Red Hat OpenShift Service Mesh matches each given request to the virtual service to a specific real destination within the mesh.

Without virtual services, Red Hat OpenShift Service Mesh distributes traffic using round-robin load balancing between all service instances. With a virtual service, you can specify traffic behavior for one or more hostnames. Routing rules in the virtual service tell Red Hat OpenShift Service Mesh how to send the traffic for the virtual service to appropriate destinations. Route destinations can be versions of the same service or entirely different services.

The following example routes requests to different versions of a service depending on which user connects to the application. Use this command to apply this example YAML file, or one you create.

```
$ oc apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v3
EOF
```

1.10.1.2. Configuring your virtual host

The following sections explain each field in the YAML file and explain how you can create a virtual host in a virtual service.

1.10.1.2.1. Hosts

The **hosts** field lists the virtual service's user-addressable destination that these routing rules apply to. This is the address or addresses the client uses when sending requests to the service.

The virtual service hostname can be an IP address, a DNS name, or, depending on the platform, a short name that resolves to a fully qualified domain name.

```
spec:
  hosts:
  - reviews
```

1.10.1.2.2. Routing rules

The **http** section contains the virtual service's routing rules, describing match conditions and actions for routing HTTP/1.1, HTTP2, and gRPC traffic sent to the destination specified in the hosts field. A routing rule consists of the destination where you want the traffic to go and zero or more match conditions, depending on your use case.

Match condition

The first routing rule in the example has a condition and begins with the match field. In this example, this routing applies to all requests from the user **jason**. Add the **headers**, **end-user**, and **exact** fields to select the appropriate requests.

```
spec:
```

```
  hosts:
  - reviews
  http:
  - match:
    - headers:
      end-user:
        exact: jason
```

Destination

The **destination** field in the route section specifies the actual destination for traffic that matches this condition. Unlike the virtual service's host, the destination's host must be a real destination that exists in the Red Hat OpenShift Service Mesh service registry. This can be a mesh service with proxies or a non-mesh service added using a service entry. In this example, the host name is a Kubernetes service name:

```
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
      end-user:
        exact: jason
    route:
    - destination:
      host: reviews
      subset: v2
```

1.10.1.2.3. Destination rules

Destination rules are applied after virtual service routing rules are evaluated, so they apply to the traffic's real destination. Virtual services route traffic to a destination. Destination rules configure what happens to traffic at that destination.

1.10.1.2.3.1. Load balancing options

By default, Red Hat OpenShift Service Mesh uses a round-robin load balancing policy, where each service instance in the instance pool gets a request in turn. Red Hat OpenShift Service Mesh also supports the following models, which you can specify in destination rules for requests to a particular service or service subset.

- Random: Requests are forwarded at random to instances in the pool.
- Weighted: Requests are forwarded to instances in the pool according to a specific percentage.
- Least requests: Requests are forwarded to instances with the least number of requests.

Destination rule example

The following example destination rule configures three different subsets for the **my-svc** destination service, with different load balancing policies:

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: my-destination-rule
```

```
spec:
  host: my-svc
  trafficPolicy:
    loadBalancer:
      simple: RANDOM
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
    trafficPolicy:
      loadBalancer:
        simple: ROUND_ROBIN
  - name: v3
    labels:
      version: v3
```

1.10.1.2.4. Gateways

You can use a gateway to manage inbound and outbound traffic for your mesh to specify which traffic you want to enter or leave the mesh. Gateway configurations are applied to standalone Envoy proxies that are running at the edge of the mesh, rather than sidecar Envoy proxies running alongside your service workloads.

Unlike other mechanisms for controlling traffic entering your systems, such as the Kubernetes Ingress APIs, Red Hat OpenShift Service Mesh gateways let you use the full power and flexibility of traffic routing. The Red Hat OpenShift Service Mesh gateway resource can layer 4-6 load balancing properties such as ports to expose, Red Hat OpenShift Service Mesh TLS settings. Instead of adding application-layer traffic routing (L7) to the same API resource, you can bind a regular Red Hat OpenShift Service Mesh virtual service to the gateway and manage gateway traffic like any other data plane traffic in a service mesh.

Gateways are primarily used to manage ingress traffic, but you can also configure egress gateways. An egress gateway lets you configure a dedicated exit node for the traffic leaving the mesh, letting you limit which services have access to external networks, or to enable secure control of egress traffic to add security to your mesh, for example. You can also use a gateway to configure a purely internal proxy.

Gateway example

The following example shows a possible gateway configuration for external HTTPS ingress traffic:

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: ext-host-gwy
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
    number: 443
    name: https
    protocol: HTTPS
    hosts:
```

```
    - ext-host.example.com
    tls:
      mode: SIMPLE
      serverCertificate: /tmp/tls.crt
      privateKey: /tmp/tls.key
```

This gateway configuration lets HTTPS traffic from **ext-host.example.com** into the mesh on port 443, but doesn't specify any routing for the traffic.

To specify routing and for the gateway to work as intended, you must also bind the gateway to a virtual service. You do this using the virtual service's gateways field, as shown in the following example:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: virtual-svc
spec:
  hosts:
  - ext-host.example.com
  gateways:
    - ext-host-gwy
```

You can then configure the virtual service with routing rules for the external traffic.

1.10.1.2.5. Service entries

A service entry adds an entry to the service registry that Red Hat OpenShift Service Mesh maintains internally. After you add the service entry, the Envoy proxies can send traffic to the service as if it was a service in your mesh. Configuring service entries allows you to manage traffic for services running outside of the mesh, including the following tasks:

- Redirect and forward traffic for external destinations, such as APIs consumed from the web, or traffic to services in legacy infrastructure.
- Define retry, timeout, and fault injection policies for external destinations.
- Run a mesh service in a Virtual Machine (VM) by adding VMs to your mesh.
- Logically add services from a different cluster to the mesh to configure a multicluster Red Hat OpenShift Service Mesh mesh on Kubernetes.
- You don't need to add a service entry for every external service that you want your mesh services to use. By default, Red Hat OpenShift Service Mesh configures the Envoy proxies to pass requests through to unknown services. However, you can't use Red Hat OpenShift Service Mesh features to control the traffic to destinations that aren't registered in the mesh.

Service entry examples

The following example mesh-external service entry adds the **ext-resource** external dependency to the Red Hat OpenShift Service Mesh service registry:

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: svc-entry
spec:
```

```
  hosts:
  - ext-svc.example.com
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
```

Specify the external resource using the hosts field. You can qualify it fully or use a wildcard prefixed domain name.

You can configure virtual services and destination rules to control traffic to a service entry in the same way you configure traffic for any other service in the mesh. For example, the following destination rule configures the traffic route to use mutual TLS to secure the connection to the **ext-svc.example.com** external service that is configured using the service entry:

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: ext-res-dr
spec:
  host: ext-svc.example.com
  trafficPolicy:
    tls:
      mode: MUTUAL
      clientCertificate: /etc/certs/myclientcert.pem
      privateKey: /etc/certs/client_private_key.pem
      caCertificates: /etc/certs/rootcacerts.pem
```

1.10.1.2.6. Sidecar

By default, Red Hat OpenShift Service Mesh configures every Envoy proxy to accept traffic on all the ports of its associated workload, and to reach every workload in the mesh when forwarding traffic. You can use a sidecar configuration to do the following:

- Fine-tune the set of ports and protocols that an Envoy proxy accepts.
- Limit the set of services that the Envoy proxy can reach.
- You might want to limit sidecar reachability like this in larger applications, where having every proxy configured to reach every other service in the mesh can potentially affect mesh performance due to high memory usage.

Sidecar example

You can specify that you want a sidecar configuration to apply to all workloads in a particular namespace, or choose specific workloads using a **workloadSelector**. For example, the following sidecar configuration configures all services in the **bookinfo** namespace to only reach services running in the same namespace and the Red Hat OpenShift Service Mesh control plane (currently needed to use the Red Hat OpenShift Service Mesh policy and telemetry features):

```
apiVersion: networking.istio.io/v1alpha3
kind: Sidecar
metadata:
  name: default
```

```
  namespace: bookinfo
spec:
  egress:
  - hosts:
    - "./*"
    - "istio-system/*"
```

1.10.2. Managing ingress traffic

In Red Hat OpenShift Service Mesh, the Ingress Gateway enables Service Mesh features such as monitoring, security, and route rules to be applied to traffic entering the cluster. Configure Service Mesh to expose a service outside of the service mesh using an Service Mesh gateway.

1.10.2.1. Determining the ingress IP and ports

Run the following command to determine if your Kubernetes cluster is running in an environment that supports external load balancers:

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

That command returns the **NAME**, **TYPE**, **CLUSTER-IP**, **EXTERNAL-IP**, **PORT(S)**, and **AGE** of each item in your namespace.

If the **EXTERNAL-IP** value is set, your environment has an external load balancer that you can use for the ingress gateway.

If the **EXTERNAL-IP** value is **<none>**, or perpetually **<pending>**, your environment does not provide an external load balancer for the ingress gateway. You can access the gateway using the service's node port.

Choose the instructions for your environment:

Configuring routing with a load balancer

Follow these instructions if your environment has an external load balancer.

Set the ingress IP and ports:

```
$ export INGRESS_HOST=$(oc -n istio-system get service istio-ingressgateway -o
jsonpath='{.status.loadBalancer.ingress[0].ip}')
```

```
$ export INGRESS_PORT=$(oc -n istio-system get service istio-ingressgateway -o
jsonpath='{.spec.ports[?(@.name=="http2")].port}')
```

```
$ export SECURE_INGRESS_PORT=$(oc -n istio-system get service istio-ingressgateway -o
jsonpath='{.spec.ports[?(@.name=="https")].port}')
```

In some environments, the load balancer may be exposed using a host name instead of an IP address. For that case, the ingress gateway's **EXTERNAL-IP** value is not be an IP address. Instead, it's a host name, and the previous command fails to set the **INGRESS_HOST** environment variable.

Use the following command to correct the **INGRESS_HOST** value:

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

Configuring routing without a load balancer

Follow these instructions if your environment does not have an external load balancer. You must use a node port instead.

Set the ingress ports:

```
$ export INGRESS_PORT=$(oc -n istio-system get service istio-ingressgateway -o
jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')
```

```
$ export SECURE_INGRESS_PORT=$(oc -n istio-system get service istio-ingressgateway -o
jsonpath='{.spec.ports[?(@.name=="https")].nodePort}')
```

1.10.3. Routing example using the bookinfo application

The Service Mesh Bookinfo sample application consists of four separate microservices, each with multiple versions. Three different versions, one of the microservices called **reviews**, have been deployed and are running concurrently.

Prerequisites:

- Deploy the Bookinfo sample application to work with the following examples.

About this task

To illustrate the problem this causes, access the bookinfo app /**product page** in a browser and refresh several times.

Sometimes the book review output contains star ratings and other times it does not. Without an explicit default service version to route to, Service Mesh routes requests to all available versions one after the other.

This tutorial helps you apply rules that route all traffic to **v1** (version 1) of the microservices. Later, you can apply a rule to route traffic based on the value of an HTTP request header.

1.10.3.1. Applying a virtual service

To route to one version only, apply virtual services that set the default version for the micro-services. In the following example, the virtual service routes all traffic to **v1** of each micro-service

1. Run the following command to apply the virtual services:

   ```
   $ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-
   1.1/samples/bookinfo/networking/virtual-service-all-v1.yaml
   ```

2. To test the command was successful, display the defined routes with the following command:

   ```
   $ oc get virtualservices -o yaml
   ```

   That command returns the following YAML file.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: details
  ...
spec:
  hosts:
  - details
  http:
  - route:
    - destination:
      host: details
      subset: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: productpage
  ...
spec:
  gateways:
  - bookinfo-gateway
  - mesh
  hosts:
  - productpage
  http:
  - route:
    - destination:
      host: productpage
      subset: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
  ...
spec:
  hosts:
  - ratings
  http:
  - route:
    - destination:
      host: ratings
      subset: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
  ...
spec:
  hosts:
  - reviews
  http:
  - route:
```

```
    - destination:
      host: reviews
      subset: v1
```

You have configured Service Mesh to route to the **v1** version of the Bookinfo microservices, most importantly the **reviews** service version 1.

1.10.3.2. Test the new routing configuration

You can easily test the new configuration by once again refreshing the /**productpage** of the Bookinfo app.

1. Open the Bookinfo site in your browser. The URL is **http://$GATEWAY_URL/productpage**, where **$GATEWAY_URL** is the External IP address of the ingress.
   The reviews part of the page displays with no rating stars, no matter how many times you refresh. This is because you configured Service Mesh to route all traffic for the reviews service to the version **reviews:v1** and this version of the service does not access the star ratings service.

   Your service mesh now routes traffic to one version of a service.

1.10.3.3. Route based on user identity

Next, change the route configuration so that all traffic from a specific user is routed to a specific service version. In this case, all traffic from a user named **jason** will be routed to the service **reviews:v2**.

Note that Service Mesh doesn't have any special, built-in understanding of user identity. This example is enabled by the fact that the **productpage** service adds a custom **end-user** header to all outbound HTTP requests to the reviews service.

1. Run the following command to enable user-based routing:

```
$ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-
1.1/samples/bookinfo/networking/virtual-service-reviews-test-v2.yaml
```

2. Confirm the rule is created:

```
$ oc get virtualservice reviews -o yaml
```

That command returns the following YAML file.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
  ...
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
      end-user:
        exact: jason
    route:
```

```
    - destination:
        host: reviews
        subset: v2
    - route:
      - destination:
          host: reviews
          subset: v1
```

3. On the /**productpage** of the Bookinfo app, log in as user **jason**. Refresh the browser. What do you see? The star ratings appear next to each review.

4. Log in as another user (pick any name you wish). Refresh the browser. Now the stars are gone. This is because traffic is routed to **reviews:v1** for all users except Jason.

You have successfully configured Service Mesh to route traffic based on user identity.

1.11. USING THE 3SCALE ISTIO ADAPTER

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

1.11.1. 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 version 1.x
- A working 3scale account (SaaS or 3scale 2.5 On-Premises)
- Red Hat OpenShift Service Mesh prerequisites
- Ensure Mixer policy enforcement is enabled. Update Mixer policy enforcement section provides instructions to check the current Mixer policy enforcement status and enable policy enforcement.

NOTE

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

NOTE

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

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

Handler configuration example

```
apiVersion: "config.istio.io/v1alpha2"
kind: handler
```

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

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

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

   Rule configuration example

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

1.11.1.1. Generating 3scale custom resources

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

Table 1.13. 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	

Option	Description	Required	Default value
--backend-url	3scale backend URL. If set, it overrides the value that is read from system configuration	No	
-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	

1.11.1.1.1. 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]"
```

1.11.1.2. 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}""
```

1.11.1.3. 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.

1.11.2. 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**.

1.11.3. 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.

1.11.4. Authenticating requests

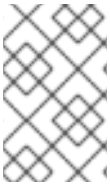
This release supports the following authentication methods:

- **Standard API Keys**: single randomized strings or hashes acting as an identifier and a secret token.
- **Application identifier and key pairs** immutable identifier and mutable secret key strings.
- **OpenID authentication method**: client ID string parsed from the JSON Web Token.

1.11.4.1. Applying authentication patterns

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

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



NOTE

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

1.11.4.1.1. API key authentication method

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

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

API key authentication method example

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

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

1.11.4.1.2. Application ID and application key pair authentication method

Service Mesh looks for the application ID and application key in query parameters and request headers, as specified in the **properties** option in the **subject** custom resource parameter. The application key is optional. It checks the values in the order given in the custom resource file. You can restrict the search for the credentials to either query parameters or request headers by not including the unwanted option.

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

Application ID and application key pair authentication method example

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

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

1.11.4.1.3. OpenID authentication method

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

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

OpenID authentication method example

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

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

In the sample **Policy** defined below, replace **issuer** and **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
```

1.11.4.1.4. Hybrid authentication method

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

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

Hybrid authentication method example

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

1.11.5. 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.

1.12. REMOVING RED HAT OPENSHIFT SERVICE MESH

This process allows you to remove Red Hat OpenShift Service Mesh from an existing OpenShift Container Platform instance. Remove the control plane before removing the operators.

1.12.1. Removing the Red Hat OpenShift Service Mesh member roll

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

1.12.2. 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.

1.12.2.1. 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 **Operators → 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**.

1.12.2.2. 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>
```

1.12.3. 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.

1.12.3.1. Removing the Red Hat OpenShift Service Mesh Operator

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

Prerequisites

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

Procedure

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

1.12.3.2. Removing the Jaeger Operator

Follow this procedure to remove the Jaeger Operator.

Prerequisites

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

Procedure

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

1.12.3.3. Removing the Kiali Operator

Follow this procedure to remove the Kiali Operator.

Prerequisites

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

Procedure

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

1.12.3.4. Removing the Elasticsearch Operator

Follow this procedure to remove the Elasticsearch Operator.

Prerequisites

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

Procedure

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

1.12.3.5. 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 mutatingwebhookconfigurations/<operator-project>.servicemesh-resources.maistra.io
```

```
$ oc delete -n <operator-project> daemonset/istio-node
```

```
$ oc delete clusterrole/istio-admin clusterrole/istio-cni clusterrolebinding/istio-cni
```

```
$ 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
```

```
$ oc get crds -o name | grep '.*\.kiali\.io' | xargs -r -n 1 oc delete
```