OpenShift Container Platform 4.8

Service Mesh

Service Mesh installation, usage, and release notes
OpenShift Container Platform 4.8 Service Mesh

Service Mesh installation, usage, and release notes
Abstract
This document provides information on how to use Service Mesh in OpenShift Container Platform.
Table of Contents

CHAPTER 1. SERVICE MESH 2.X ............................................................ 10
  1.1. ABOUT OPENSHIFT SERVICE MESH ................................. 10
      1.1.1. Introduction to Red Hat OpenShift Service Mesh ... 10
      1.1.1.1. Making open source more inclusive .................. 10
  1.2. SERVICE MESH RELEASE NOTES ..................................... 10
      1.2.1. New features ............................................... 10
      1.2.1.1. Component versions included in Red Hat OpenShift Service Mesh version 2.0.6 ... 10
      1.2.1.2. Red Hat OpenShift Service Mesh on Red Hat OpenShift Dedicated and Microsoft Azure Red Hat OpenShift 11
      1.2.1.3. New features Red Hat OpenShift Service Mesh 2.0.6 ... 11
      1.2.1.4. New features Red Hat OpenShift Service Mesh 2.0.5 ... 11
      1.2.1.5. New features Red Hat OpenShift Service Mesh 2.0.4 ... 11
      1.2.1.5.1. Manual updates required by CVE-2021-29492 and CVE-2021-31920 ... 11
      1.2.1.5.2. Updating the path normalization configuration ........... 12
      1.2.1.5.3. Path normalization configuration examples .......... 13
      1.2.1.5.4. Configuring your SMCP for path normalization ....... 14
      1.2.1.5.5. Configuring for case normalization ................. 14
      1.2.1.6. New features Red Hat OpenShift Service Mesh 2.0.3 ... 15
      1.2.1.7. New features Red Hat OpenShift Service Mesh 2.0.2 ... 15
      1.2.1.8. New features Red Hat OpenShift Service Mesh 2.0.1 ... 15
      1.2.1.9. New features Red Hat OpenShift Service Mesh 2.0 ... 15
      1.2.2. Technology Preview ........................................ 16
      1.2.2.1. OVN-Kubernetes technology preview ................. 16
      1.2.2.2. WebAssesmbly technology preview .................... 16
      1.2.3. Deprecated features ...................................... 17
      1.2.3.1. Deprecated features Red Hat OpenShift Service Mesh 2.0.2 ... 17
      1.2.4. Known issues ........................................... 17
      1.2.4.1. Service Mesh known issues ......................... 18
      1.2.4.2. Kiali known issues ............................... 19
      1.2.4.3. Jaeger known issues .......................... 19
      1.2.5. Fixed issues .......................................... 19
      1.2.5.1. Service Mesh fixed issues ....................... 19
      1.2.5.2. Jaeger fixed issues .......................... 21
  1.3. GETTING SUPPORT ..................................................... 22
      1.3.1. Getting support ....................................... 22
      1.3.2. About the must-gather tool .......................... 22
      1.3.2.1. Prerequisites ..................................... 22
      1.3.3. About collecting service mesh data .................. 23
  1.4. UNDERSTANDING RED HAT OPENSHIFT SERVICE MESH .......... 23
      1.4.1. Understanding service mesh ................................ 23
      1.4.2. Red Hat OpenShift Service Mesh Architecture ........ 24
      1.4.3. Understanding Kiali .................................. 24
      1.4.3.1. Kiali overview .................................. 24
      1.4.3.2. Kiali architecture ................................ 24
      1.4.3.3. Kiali features .................................. 25
      1.4.4. Understanding Jaeger .................................. 26
      1.4.4.1. Jaeger overview .................................. 26
      1.4.4.2. Jaeger architecture ................................ 26
      1.4.4.3. Jaeger features .................................. 27
      1.4.5. Next steps ........................................... 27
  1.5. SERVICE MESH AND ISTIO DIFFERENCES .................... 27
1.5.1. Service Mesh features
1.5.1.1. Command line tool
1.5.1.2. Installation and upgrades
1.5.1.3. Federation and multic和平
1.5.1.4. Automatic injection
1.5.1.5. Istio Role Based Access Control features
1.5.1.6. OpenSSL
1.5.1.7. External workloads
1.5.1.8. Component modifications
1.5.1.9. Envoy services
1.5.1.10. Istio Container Network Interface (CNI) plug-in
1.5.1.11. Routes for Istio Gateways
1.5.1.11.1. Catch-all domains
1.5.1.11.2. Subdomains
1.5.1.11.3. Transport layer security
1.5.1.11.4. WebAssembly Extensions
Additional resources
1.5.2. Multitenant installations
1.5.2.1. Multitenancy versus cluster-wide installations
1.5.2.2. Cluster scoped resources
1.5.3. Kiali and service mesh
1.5.4. Jaeger and service mesh

1.6. PREPARING TO INSTALL RED HAT OPENS SHIFT SERVICE MESH
1.6.1. Prerequisites
1.6.2. Supported configurations
1.6.2.1. Supported network configurations
1.6.2.2. Supported configurations for Kiali
1.6.2.3. Supported configurations for Distributed Tracing
1.6.2.4. Supported Mixer adapters
1.6.3. Next steps

1.7. INSTALLING THE OPERATORS
1.7.1. Operator overview
1.7.2. Installing the Operators
1.7.3. Next steps

1.8. CREATING THE SERVICEMESHCONTROLPLANE
1.8.1. Deploying the control plane from the web console
1.8.2. Deploying the control plane from the CLI
1.8.3. Next steps

1.9. ADDING SERVICES TO A SERVICE MESH
1.9.1. Creating the Red Hat OpenShift Service Mesh member roll
1.9.1.1. Creating the member roll from the web console
1.9.1.2. Creating the member roll from the CLI
1.9.2. Adding or removing projects from the service mesh
1.9.2.1. Adding or removing projects from the member roll using the web console
1.9.2.2. Adding or removing projects from the member roll using the CLI
1.9.3. Bookinfo example application
1.9.3.1. Installing the Bookinfo application
1.9.3.2. Adding default destination rules
1.9.3.3. Verifying the Bookinfo installation
1.9.3.4. Removing the Bookinfo application
1.9.3.4.1. Delete the Bookinfo project
1.9.3.4.2. Remove the Bookinfo project from the Service Mesh member roll
1.9.4. Next steps
1.10. ENABLING SIDECAR INJECTION
1.10.1. Prerequisites
1.10.2. Enabling automatic sidecar injection
1.10.3. Updating your application pods
1.10.4. Setting environment variables on the proxy in applications through annotations
1.10.5. Next steps

1.11. UPGRADING RED HAT OPENSHIFT SERVICE MESH FROM VERSION 1.1 TO VERSION 2.0
1.11.1. Upgrading Red Hat OpenShift Service Mesh
1.11.2. Configuring the 2.0 ServiceMeshControlPlane
   1.11.2.1. Architecture changes
   1.11.2.2. Annotation changes
   1.11.2.3. Behavioral changes
   1.11.2.4. Migration details for unsupported resources
   1.11.2.5. Mixer plugins
   1.11.2.6. Mutual TLS changes
      1.11.2.6.1. Other mTLS Examples
   1.11.3. Configuration recipes
      1.11.3.1. Mutual TLS in a data plane
      1.11.3.2. Custom signing key
      1.11.3.3. Tracing
      1.11.3.4. Visualization
      1.11.3.5. Resource utilization and scheduling
   1.11.4. Next steps for migrating your applications and workflows

1.12. MANAGING USERS AND PROFILES
1.12.1. Creating the Red Hat OpenShift Service Mesh members
1.12.2. Creating control plane profiles
   1.12.2.1. Creating the ConfigMap
   1.12.2.2. Setting the correct network policy

1.13. SECURITY
1.13.1. Mutual Transport Layer Security (mTLS)
   1.13.1.1. Enabling strict mTLS across the service mesh
      1.13.1.1.1. Configuring sidecars for incoming connections for specific services
      1.13.1.1.2. Configuring sidecars for outgoing connections
      1.13.1.1.3. Setting the minimum and maximum protocol versions
   1.13.2. Configuring Role Based Access Control (RBAC)
      1.13.2.1. Configure intra-project communication
      1.13.2.1.1. Restrict access to services outside a namespace
      1.13.2.1.2. Creating allow-all and default deny-all authorization policies
      1.13.2.2. Allow or deny access to the ingress gateway
      1.13.2.3. Restrict access with JSON Web Token
   1.13.3. Configuring cipher suites and ECDH curves
   1.13.4. Adding an external certificate authority key and certificate
      1.13.4.1. Adding an existing certificate and key
      1.13.4.2. Verifying your certificates
      1.13.4.3. Removing the certificates

1.14. CONFIGURING TRAFFIC MANAGEMENT
1.14.1. Routing tutorial
   1.14.1.1. Applying a virtual service
   1.14.1.2. Testing the new route configuration
   1.14.1.3. Route based on user identity
   1.14.2. Routing and managing traffic
   1.14.2.1. Traffic management with virtual services
      1.14.2.1.1. Configuring virtual services
1.14.2.2. Configuring your virtual host
  1.14.2.2.1. Hosts
  1.14.2.2.2. Routing rules
  1.14.2.2.3. Destination rules
    1.14.2.2.3.1. Load balancing options
  1.14.2.2.4. Gateways
  1.14.2.2.5. Service entries

1.14.3. Managing ingress traffic
  1.14.3.1. Determining the ingress IP and ports
  1.14.3.1.1. Determining ingress ports with a load balancer
  1.14.3.1.2. Determining ingress ports without a load balancer

1.14.4. Configuring ingress using a gateway

1.14.5. Automatic routes
  1.14.5.1. Subdomains
  1.14.5.2. Creating subdomain routes
  1.14.5.3. Annotations
  1.14.5.4. Disabling automatic route creation
  1.14.5.5. Sidecar

1.15. METRICS AND TRACES
  1.15.1. Accessing metrics and tracing data from the CLI
  1.15.2. Viewing service mesh data
    1.15.2.1. Working with data in the Kiali console
      1.15.2.1.1. Namespace graphs
  1.15.3. Distributed tracing
    1.15.3.1. Generating example traces and analyzing trace data
    1.15.3.2. Adjusting the sampling rate
    1.15.3.3. Connecting standalone Jaeger
  1.15.4. Accessing Grafana
  1.15.5. Accessing Prometheus

1.16. PERFORMANCE AND SCALABILITY
  1.16.1. Setting limits on compute resources
  1.16.2. Load test results
    1.16.2.1. Control plane performance
    1.16.2.2. Data plane performance
      1.16.2.2.1. CPU and memory consumption
      1.16.2.2.2. Additional latency

1.17. EXTENSIONS
  1.17.1. WebAssembly extensions
    1.17.1.1. Container Format
    1.17.1.2. Example Rust extension
    1.17.1.3. Enabling WebAssembly extension support
    1.17.1.4. Deploying extensions

1.18. CONFIGURING SERVICE MESH FOR PRODUCTION
  1.18.1. Configuring your ServiceMeshControlPlane resource for production

1.19. ADDITIONAL RESOURCES

1.20. USING THE 3SCALE ISTIO ADAPTER
  1.20.1. Integrate the 3scale adapter with Red Hat OpenShift Service Mesh
    1.20.1.1. Generating 3scale custom resources
      1.20.1.1.1. Generate templates from URL examples
    1.20.1.2. Generating manifests from a deployed adapter
    1.20.1.3. Routing service traffic through the adapter
    1.20.2. Configure the integration settings in 3scale
    1.20.3. Caching behavior
1.20.4. Authenticating requests
  1.20.4.1. Applying authentication patterns
    1.20.4.1.1. API key authentication method
    1.20.4.1.2. Application ID and application key pair authentication method
    1.20.4.1.3. OpenID authentication method
    1.20.4.1.4. Hybrid authentication method
  1.20.5. 3scale Adapter metrics
  1.20.6. 3scale backend cache
    1.20.6.1. Advantages of enabling backend cache
    1.20.6.2. Trade-offs for having lower latencies
    1.20.6.3. Backend cache configuration settings
  1.20.7. 3scale Istio Adapter APIcast emulation
  1.20.8. 3scale Istio adapter verification
  1.20.9. 3scale Istio adapter troubleshooting checklist

1.21. UNINSTALLING RED HAT OPENSФFT SERVICE MESH
  1.21.1. Removing the Red Hat OpenShift Service Mesh control plane
    1.21.1.1. Removing the control plane with the web console
    1.21.1.2. Removing the control plane from the CLI
  1.21.2. Removing the installed Operators
    1.21.2.1. Removing the Operators
  1.21.3. Clean up Operator resources

1.22. CONFIGURATION REFERENCE
  1.22.1. Control plane parameters
  1.22.2. 3scale configuration
  1.22.3. More information

CHAPTER 2. SERVICE MESH 1.X ................................................................. 122
  2.1. SERVICE MESH RELEASE NOTES ..................................................... 122
    2.1.1. Introduction to Red Hat OpenShift Service Mesh
      2.1.1.1. Making open source more inclusive
    2.1.2. Getting support
      2.1.2.1. About the must-gather tool
      2.1.2.2. Prerequisites
      2.1.2.3. About collecting service mesh data
    2.1.3. Red Hat OpenShift Service Mesh supported configurations
      2.1.3.1. Supported configurations for Kiali on Red Hat OpenShift Service Mesh
      2.1.3.2. Supported Mixer adapters
    2.1.4. New Features
      2.1.4.1. Component versions included in Red Hat OpenShift Service Mesh version 1.1.16
      2.1.4.2. New features Red Hat OpenShift Service Mesh 1.1.16
      2.1.4.3. New features Red Hat OpenShift Service Mesh 1.1.15
      2.1.4.4. New features Red Hat OpenShift Service Mesh 1.1.14
        2.1.4.4.1. Manual updates required by CVE-2021-29492 and CVE-2021-31920
        2.1.4.4.2. Updating the path normalization configuration
        2.1.4.4.3. Path normalization configuration examples
        2.1.4.4.4. Configuring your SMCP for path normalization
      2.1.4.5. New features Red Hat OpenShift Service Mesh 1.1.13
      2.1.4.6. New features Red Hat OpenShift Service Mesh 1.1.12
      2.1.4.7. New features Red Hat OpenShift Service Mesh 1.1.11
      2.1.4.8. New features Red Hat OpenShift Service Mesh 1.1.10
      2.1.4.9. New features Red Hat OpenShift Service Mesh 1.1.9
      2.1.4.10. New features Red Hat OpenShift Service Mesh 1.1.8
      2.1.4.11. New features Red Hat OpenShift Service Mesh 1.1.7
2.1.4.12. New features Red Hat OpenShift Service Mesh 1.1.6
2.1.4.13. New features Red Hat OpenShift Service Mesh 1.1.5
2.1.4.14. New features Red Hat OpenShift Service Mesh 1.1.4
2.1.4.15. New features Red Hat OpenShift Service Mesh 1.1.3
2.1.4.16. New features Red Hat OpenShift Service Mesh 1.1.2
2.1.4.17. New features Red Hat OpenShift Service Mesh 1.1.1
2.1.4.18. New features Red Hat OpenShift Service Mesh 1.1.0
   2.1.4.18.1. Manual updates from 1.0 to 1.1
2.1.5. Deprecated features
   2.1.5.1. Deprecated features Red Hat OpenShift Service Mesh 1.1.5
2.1.6. Known issues
   2.1.6.1. Service Mesh known issues
   2.1.6.2. Kiali known issues
   2.1.6.3. Jaeger known issues
2.1.7. Fixed issues
   2.1.7.1. Service Mesh fixed issues
   2.1.7.2. Kiali fixed issues
   2.1.7.3. Jaeger fixed issues
2.2. UNDERSTANDING RED HAT OPENSHIFT SERVICE MESH
   2.2.1. Understanding service mesh
   2.2.2. Red Hat OpenShift Service Mesh Architecture
   2.2.3. Understanding Kiali
      2.2.3.1. Kiali overview
      2.2.3.2. Kiali architecture
      2.2.3.3. Kiali features
   2.2.4. Understanding Jaeger
      2.2.4.1. Jaeger overview
      2.2.4.2. Jaeger architecture
      2.2.4.3. Jaeger features
   2.2.5. Next steps
2.3. SERVICE MESH AND ISTIO DIFFERENCES
   2.3.1. Multitenant installations
      2.3.1.1. Multitenancy versus cluster-wide installations
      2.3.1.2. Cluster scoped resources
   2.3.2. Differences between Istio and Red Hat OpenShift Service Mesh
      2.3.2.1. Command line tool
      2.3.2.2. Automatic injection
      2.3.2.3. Istio Role Based Access Control features
      2.3.2.4. OpenSSL
      2.3.2.5. Component modifications
      2.3.2.6. Envoy, Secret Discovery Service, and certificates
      2.3.2.7. Istio Container Network Interface (CNI) plug-in
      2.3.2.8. Routes for Istio Gateways
         2.3.2.8.1. Catch-all domains
         2.3.2.8.2. Subdomains
         2.3.2.8.3. Transport layer security
            Additional resources
   2.3.3. Kiali and service mesh
   2.3.4. Jaeger and service mesh
2.4. PREPARING TO INSTALL RED HAT OPENSHIFT SERVICE MESH
   2.4.1. Prerequisites
2.7.5.2. Subdomains
2.7.6. Links

2.8. DEPLOYING APPLICATIONS ON RED HAT OPENSHIFT SERVICE MESH
2.8.1. Prerequisites
2.8.2. Creating control plane templates
2.8.2.1. Creating the ConfigMap
2.8.3. Enabling automatic sidecar injection
2.8.4. Setting environment variables on the proxy in applications through annotations
2.8.5. Updating Mixer policy enforcement
2.8.5.1. Setting the correct network policy
2.8.6. Bookinfo example application
2.8.6.1. Installing the Bookinfo application
2.8.6.2. Adding default destination rules
2.8.6.3. Verifying the Bookinfo installation
2.8.6.4. Removing the Bookinfo application
2.8.6.4.1. Delete the Bookinfo project
2.8.6.4.2. Remove the Bookinfo project from the Service Mesh member roll
2.8.7. Generating example traces and analyzing trace data

2.9. DATA VISUALIZATION AND OBSERVABILITY
2.9.1. Viewing service mesh data
2.9.2. Working with data in the Kiali console
2.9.2.1. Namespace graphs

2.10. USING THE 3SCALE ISTIO ADAPTER
2.10.1. Integrate the 3scale adapter with Red Hat OpenShift Service Mesh
2.10.1.1. Generating 3scale custom resources
2.10.1.1.1. Generate templates from URL examples
2.10.1.2. Generating manifests from a deployed adapter
2.10.1.3. Routing service traffic through the adapter
2.10.2. Configure the integration settings in 3scale
2.10.3. Caching behavior
2.10.4. Authenticating requests
2.10.4.1. Applying authentication patterns
2.10.4.1.1. API key authentication method
2.10.4.1.2. Application ID and application key pair authentication method
2.10.4.1.3. OpenID authentication method
2.10.4.1.4. Hybrid authentication method
2.10.5. 3scale Adapter metrics
2.10.6. 3scale Istio adapter verification
2.10.7. 3scale Istio adapter troubleshooting checklist

2.11. REMOVING RED HAT OPENSHIFT SERVICE MESH
2.11.1. Removing the Red Hat OpenShift Service Mesh control plane
2.11.1.1. Removing the control plane with the web console
2.11.1.2. Removing the control plane from the CLI
2.11.2. Removing the installed Operators
2.11.2.1. Removing the Operators
2.11.2.2. Clean up Operator resources
CHAPTER 1. SERVICE MESH 2.X

1.1. ABOUT OPENSIGHT SERVICE MESH

1.1.1. Introduction to Red Hat OpenShift Service Mesh

Red Hat OpenShift Service Mesh addresses a variety of problems in a microservice architecture by creating a centralized point of control in an application. It adds a transparent layer on existing distributed applications without requiring any changes to the application code.

Microservice architectures split the work of enterprise applications into modular services, which can make scaling and maintenance easier. However, as an enterprise application built on a microservice architecture grows in size and complexity, it becomes difficult to understand and manage. Service Mesh can address those architecture problems by capturing or intercepting traffic between services and can modify, redirect, or create new requests to other services.

Service Mesh, which is based on the open source Istio project, 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.

1.1.1.1. Making open source more inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message.

1.2. SERVICE MESH RELEASE NOTES

1.2.1. 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.

1.2.1.1. Component versions included in Red Hat OpenShift Service Mesh version 2.0.6
1.2.1.2. Red Hat OpenShift Service Mesh on Red Hat OpenShift Dedicated and Microsoft Azure Red Hat OpenShift

Red Hat OpenShift Service Mesh is now supported through Red Hat OpenShift Dedicated and Microsoft Azure Red Hat OpenShift.

1.2.1.3. New features Red Hat OpenShift Service Mesh 2.0.6

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

1.2.1.4. New features Red Hat OpenShift Service Mesh 2.0.5

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

1.2.1.5. New features Red Hat OpenShift Service Mesh 2.0.4

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

---

**IMPORTANT**

There are manual steps that must be completed to address CVE-2021-29492 and CVE-2021-31920.

1.2.1.5.1. Manual updates required by CVE-2021-29492 and CVE-2021-31920

Istio contains a remotely exploitable vulnerability where an HTTP request path with multiple slashes or escaped slash characters (\%2F or \%5C) could potentially bypass an Istio authorization policy when path-based authorization rules are used.

For example, assume an Istio cluster administrator defines an authorization DENY policy to reject the request at path /admin. A request sent to the URL path //admin will NOT be rejected by the authorization policy.

According to RFC 3986, the path //admin with multiple slashes should technically be treated as a different path from the /admin. However, some backend services choose to normalize the URL paths by merging multiple slashes into a single slash. This can result in a bypass of the authorization policy (//admin does not match /admin), and a user can access the resource at path /admin in the backend; this would represent a security incident.
Your cluster is impacted by this vulnerability if you have authorization policies using **ALLOW action + notPaths field** or **DENY action + paths field** patterns. These patterns are vulnerable to unexpected policy bypasses.

Your cluster is NOT impacted by this vulnerability if:

- You don’t have authorization policies.
- Your authorization policies don’t define **paths** or **notPaths** fields.
- Your authorization policies use **ALLOW action + paths field** or **DENY action + notPaths field** patterns. These patterns could only cause unexpected rejection instead of policy bypasses. The upgrade is optional for these cases.

**NOTE**

The Red Hat OpenShift Service Mesh configuration location for path normalization is different from the Istio configuration.

### 1.2.1.5.2. Updating the path normalization configuration

Istio authorization policies can be based on the URL paths in the HTTP request. **Path normalization**, also known as URI normalization, modifies and standardizes the incoming requests’ paths so that the normalized paths can be processed in a standard way. Syntactically different paths may be equivalent after path normalization.

Istio supports the following normalization schemes on the request paths before evaluating against the authorization policies and routing the requests:

**Table 1.1. Normalization schemes**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Example</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NONE</strong></td>
<td>No normalization is done. Anything received by Envoy will be forwarded exactly as-is to any backend service.</td>
<td>../%2Fa../b is evaluated by the authorization policies and sent to your service.</td>
<td>This setting is vulnerable to CVE-2021-31920.</td>
</tr>
<tr>
<td><strong>BASE</strong></td>
<td>This is currently the option used in the default installation of Istio. This applies the <code>normalize_path</code> option on Envoy proxies, which follows RFC 3986 with extra normalization to convert backslashes to forward slashes.</td>
<td>/a/..b is normalized to /b. \da is normalized to /da.</td>
<td>This setting is vulnerable to CVE-2021-31920.</td>
</tr>
<tr>
<td><strong>MERGE_SLASHES</strong></td>
<td>Slashes are merged after the <code>BASE</code> normalization.</td>
<td>/a//b is normalized to /a/b.</td>
<td>Update to this setting to mitigate CVE-2021-31920.</td>
</tr>
</tbody>
</table>
The strictest setting when you allow all traffic by default. This setting is recommended, with the caveat that you must thoroughly test your authorization policies routes. Percent-encoded slash and backslash characters (\%2F, \%2f, \%5C and \%5c) are decoded to/ or \) before the **MERGE_SLASHES** normalization.

The normalization algorithms are conducted in the following order:

1. Percent-decode \%2F, \%2f, \%5C and \%5c.

2. The **RFC 3986** and other normalization implemented by the `normalize_path` option in Envoy.

3. Merge slashes.

**WARNING**

While these normalization options represent recommendations from HTTP standards and common industry practices, applications may interpret a URL in any way it chooses to. When using denial policies, ensure that you understand how your application behaves.

### 1.2.1.5.3. Path normalization configuration examples

Ensuring Envoy normalizes request paths to match your backend services’ expectations is critical to the security of your system. The following examples can be used as a reference for you to configure your system. The normalized URL paths, or the original URL paths if **NONE** is selected, will be:

1. Used to check against the authorization policies.

2. Forwarded to the backend application.

**Table 1.2. Configuration examples**

<table>
<thead>
<tr>
<th>If your application...</th>
<th>Choose...</th>
</tr>
</thead>
</table>

If your application... | Choose... |
<table>
<thead>
<tr>
<th>If your application...</th>
<th>Choose...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relies on the proxy to do normalization</td>
<td>BASE, MERGE_SLASHES or DECODE_AND_MERGE_SLASHES</td>
</tr>
<tr>
<td>Normalizes request paths based on RFC 3986 and does not merge slashes.</td>
<td>BASE</td>
</tr>
<tr>
<td>Normalizes request paths based on RFC 3986 and merges slashes, but does not decode percent-encoded slashes.</td>
<td>MERGE_SLASHES</td>
</tr>
<tr>
<td>Normalizes request paths based on RFC 3986, decodes percent-encoded slashes, and merges slashes.</td>
<td>DECODE_AND_MERGE_SLASHES</td>
</tr>
<tr>
<td>Processes request paths in a way that is incompatible with RFC 3986.</td>
<td>NONE</td>
</tr>
</tbody>
</table>

1.2.1.5.4. Configuring your SMCP for path normalization

To configure path normalization for Red Hat OpenShift Service Mesh, specify the following in your ServiceMeshControlPlane. Use the configuration examples to help determine the settings for your system.

SMCP v2 pathNormalization

```yaml
spec:
  techPreview:
    global:
      pathNormalization: <option>
```

1.2.1.5.5. Configuring for case normalization

In some environments, it may be useful to have paths in authorization policies compared in a case insensitive manner. For example, treating `https://myurl/get` and `https://myurl/GeT` as equivalent. In those cases, you can use the EnvoyFilter shown below. This filter will change both the path used for comparison and the path presented to the application.

Save the EnvoyFilter to a file and execute the following command:

```
$ oc create -f <myEnvoyFilterFile>
```

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: ingress-case-insensitive
  namespace: istio-system
spec:
cfgPatch:
```
1.2.1.6. New features Red Hat OpenShift Service Mesh 2.0.3

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

In addition, this release has the following new features:

- Added an option to the must-gather data collection tool that gathers information from a specified control plane namespace. For more information, see OSSM-351.

- Improved performance for control planes with hundreds of namespaces

1.2.1.7. New features Red Hat OpenShift Service Mesh 2.0.2

This release of Red Hat OpenShift Service Mesh adds support for IBM Z and IBM Power Systems. It also addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.2.1.8. New features Red Hat OpenShift Service Mesh 2.0.1

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

1.2.1.9. New features Red Hat OpenShift Service Mesh 2.0

This release of Red Hat OpenShift Service Mesh adds support for Istio 1.6.5, Jaeger 1.20.0, Kiali 1.24.2, and the 3scale Istio Adapter 2.0 and OpenShift Container Platform 4.6.

In addition, this release has the following new features:

- Introduces a re-architected control plane. The Mixer component has been deprecated and will be removed in a future release. The other control plane components, Pilot, Galley, Citadel, have been combined into a single binary known as Istiod. The "d" stands for daemon.
  - Simplifies installation, upgrades, and management of the control plane.
Reduces the control plane's resource usage and startup time.

- Improves performance by reducing inter-control plane communication over networking.

- Adds support for Envoy's Secret Discovery Service (SDS). SDS is a more secure and efficient mechanism for delivering secrets to Envoy side car proxies.
  - Removes the need to use Kubernetes Secrets, which have well known security risks.
  - Improves performance during certificate rotation, as proxies no longer require a restart to recognize new certificates.

- Adds support for Istio's Telemetry v2 architecture, which is built using WebAssembly extensions. This new architecture brings significant performance improvements.

- Updates the ServiceMeshControlPlane resource to v2 with a streamlined configuration to make it easier to manage the Control Plane.

- Introduces WebAssembly extensions as a Technology Preview feature.

### 1.2.2. Technology Preview

**IMPORTANT**

Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see the Technology Preview Support Scope.

#### 1.2.2.1. OVN-Kubernetes technology preview

Red Hat OpenShift Service Mesh 2.0.1 introduces technology preview support for the OVN-Kubernetes network type on OpenShift Container Platform 4.6 and 4.7.

#### 1.2.2.2. WebAssembly technology preview

Red Hat OpenShift Service Mesh 2.0.0 introduces support for WebAssembly extensions to Envoy Proxy.

Up through release 1.5, Istio implemented extensions using the Mixer Telemetry and Policy components. In Istio 1.5 Mixer was deprecated and WebAssembly was introduced as the new mechanism for extensions in Istio. Envoy now allows extensions using WebAssembly ("WASM") - a format for executing code written in multiple programming languages. Mixer has been deprecated as of Istio 1.5, and will be removed in 1.8. Going forward, extensions to Istio will be implemented with Envoy plugins written with WebAssembly.

The new Telemetry architecture is based on these WebAssembly extensions. For Service Mesh 2.0, we are introducing WebAssembly extensions as a Tech Preview feature. WebAssembly extensions is the new way of extending Istio functionality, replacing the Mixer component, which has been deprecated and will eventually be removed.
NOTE

Note that built-in Istio WASM extensions are not included in the proxy binary and that WASM filters from the upstream Istio community are not supported in Red Hat OpenShift Service Mesh 2.0.

For more information about WebAssembly extensions, see Extensions.

1.2.3. 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.

1.2.3.1. Deprecated features Red Hat OpenShift Service Mesh 2.0

The Mixer component was deprecated in release 2.0 and will be removed in release 2.1. While using Mixer for implementing extensions was still supported in release 2.0, extensions should have been migrated to the new WebAssembly mechanism.

The following resource types are no longer supported in Red Hat OpenShift Service Mesh 2.0:

- **Policy** (authentication.istio.io/v1alpha1) is no longer supported. Depending on the specific configuration in your Policy resource, you may have to configure multiple resources to achieve the same effect.
  - Use **RequestAuthentication** (security.istio.io/v1beta1)
  - Use **PeerAuthentication** (security.istio.io/v1beta1)
- **ServiceMeshPolicy** (maistra.io/v1) is no longer supported.
  - Use **RequestAuthentication** or **PeerAuthentication**, as mentioned above, but place in the control plane namespace.
- **RbacConfig** (rbac.istio.io/v1alpha1) is no longer supported.
  - Replaced by **AuthorizationPolicy** (security.istio.io/v1beta1), which encompasses behavior of RbacConfig, ServiceRole, and ServiceRoleBinding.
- **ServiceMeshRbacConfig** (maistra.io/v1) is no longer supported.
  - Use **AuthorizationPolicy** as above, but place in control plane namespace.
- **ServiceRole** (rbac.istio.io/v1alpha1) is no longer supported.
- **ServiceRoleBinding** (rbac.istio.io/v1alpha1) is no longer supported.
- In Kiali, the login and LDAP strategies are deprecated. A future version will introduce authentication using OpenID providers.

1.2.4. 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.

• The Bookinfo sample application cannot be installed on IBM Z and IBM Power Systems.

• WebAssembly is unsupported on IBM Z and IBM Power Systems.

### 1.2.4.1. Service Mesh known issues

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

• **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.

• **OSSM-285** When trying to access the Kiali console, receive the following error message "Error trying to get OAuth Metadata". The workaround is to restart the Kiali pod.

• **MAISTRA-1947** *Technology Preview* Updates to ServiceMeshExtensions are not applied. The workaround is to remove and recreate the ServiceMeshExtensions.

• **MAISTRA-1959** *Migration to 2.0* Prometheus scraping (**spec.addons.prometheus.scrape** set to **true**) does not work when mTLS is enabled. Additionally, Kiali displays extraneous graph data when mTLS is disabled. This problem can be addressed by excluding port 15020 from proxy configuration, for example,

```yaml
spec:
  proxy:
    networking:
      trafficControl:
        inbound:
          excludedPorts:
            - 15020
```

• **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.
Applying multiple gateways referencing the same hostname will cause all gateways to stop functioning.

### 1.2.4.2. Kiali known issues

**NOTE**

New issues for Kiali should be created in the OpenShift Service Mesh project with the **Component** set to Kiali.

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.

### 1.2.4.3. Jaeger known issues

These limitations exist in Jaeger:

- Apache Spark is not supported.

- Jaeger streaming via AMQ/Kafka is unsupported on IBM Z and IBM Power Systems.

These are the known issues in Jaeger:

- **BZ-1918920** The Elasticsearch pods does not get restarted automatically after an update. As a workaround, restart the pods manually.

- **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](#).

### 1.2.5. Fixed issues

The following issues been resolved in the current release:

#### 1.2.5.1. Service Mesh fixed issues

- **MAISTRA-2401** CVE-2021-3586 servicemesh-operator: NetworkPolicy resources incorrectly specified ports for ingress resources. The NetworkPolicy resources installed for Red Hat OpenShift Service Mesh did not properly specify which ports could be accessed. This allowed access to all ports on these resources from any pod. Network policies applied to the following resources are affected:

  - Galley

  - Grafana
- Istiod
- Jaeger
- Kiali
- Prometheus
- Sidecar injector

- **MAISTRA-2378** When the cluster is configured to use OpenShiftSDN with `ovs-multitenant` and the mesh contains a large number of namespaces (200+), the OpenShift Container Platform networking plugin is unable to configure the namespaces quickly. Service Mesh times out causing namespaces to be continuously dropped from the service mesh and then reenlisted.

- **MAISTRA-2370** Handle tombstones in listerInformer. The updated cache codebase was not handling tombstones when translating the events from the namespace caches to the aggregated cache, leading to a panic in the go routine.

- **OSSM-449** VirtualService and Service causes an error "Only unique values for domains are permitted. Duplicate entry of domain."

- **OSSM-419** Namespaces with similar names will all show in Kiali namespace list, even though namespaces may not be defined in Service Mesh Member Role.

- **OSSM-296** When adding health configuration to the Kiali custom resource (CR) is it not being replicated to the Kiali configmap.

- **OSSM-291** In the Kiali console, on the Applications, Services, and Workloads pages, the "Remove Label from Filters" function is not working.

- **OSSM-289** In the Kiali console, on the Service Details pages for the 'istio-ingressgateway' and 'jaeger-query' services there are no Traces being displayed. The traces exist in Jaeger.

- **OSSM-287** In the Kiali console there are no traces being displayed on the Graph Service.

- **MAISTRA-2010** AuthorizationPolicy does not support `request.regex.headers` field. The validatingwebhook rejects any AuthorizationPolicy with the field, and even if you disable that, Pilot tries to validate it using the same code, and it does not work.

- **MAISTRA-1979** Migration to 2.0 The conversion webhook drops the following important fields when converting `SMCP.status` from v2 to v1:
  - conditions
  - components
  - observedGeneration
  - annotations

  Upgrading the operator to 2.0 might break client tools that read the SMCP status using the maistra.io/v1 version of the resource.

  This also causes the READY and STATUS columns to be empty when you run `oc get servicemeshcontrolplanes.v1.maistra.io`. 

---

OpenShift Container Platform 4.8 Service Mesh
MAISTRA-1089 Migration to 2.0 Gateways created in a non-control plane namespace are automatically deleted. Users will need to manually delete these resources after removing the gateway definition from the SMCP spec.

MAISTRA-1983 Migration to 2.0 Upgrading to 2.0.0 with an existing invalid ServiceMeshControlPlane cannot easily be repaired. The invalid items in the ServiceMeshControlPlane resource caused an unrecoverable error. The fix makes the errors recoverable. You can delete the invalid resource and replace it with a new one or edit the resource to fix the errors. For more information about editing your resource, see [Configuring the Red Hat OpenShift Service Mesh installation].

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.

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

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.

1.2.5.2. Jaeger fixed issues

- TRACING-1725 Follow-up to TRACING-1631. Additional fix to ensure that Elasticsearch certificates are properly reconciled when there are multiple Jaeger production instances, using same name but within different namespaces. See also BZ-1918920.

- TRACING-1631 Multiple Jaeger production instances, using same name but within different namespaces, causing Elasticsearch certificate issue. When multiple service meshes were installed, all of the Jaeger Elasticsearch instances had the same Elasticsearch secret instead of individual secrets, which prevented the OpenShift Elasticsearch Operator from communicating with all of the Elasticsearch clusters.

- 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.

- 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 an error: Failed
to pull image registry.redhat.io/amq7/amq-streams-kafka-24-rhel7@sha256:f9ceca004f1b7dccb3b82d9a8027961f9fe4104e0ed69752c0bdd8078b4a1076.

1.3. GETTING SUPPORT

1.3.1. 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.

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. You can send that diagnostic information to support for both OpenShift Container Platform and Red Hat OpenShift Service Mesh.

1.3.2. 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.

1.3.2.1. Prerequisites

- Access to the cluster as a user with the cluster-admin role. If you use Red Hat OpenShift Dedicated, you must have an account with the dedicated-admin role.
- The OpenShift Container Platform CLI (oc) installed.
1.3.3. 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-rhel8
```

To collect Red Hat OpenShift Service Mesh data for a specific control plane namespace with `must-gather`, you must specify the Red Hat OpenShift Service Mesh image and namespace. In this example, replace `<namespace>` with your control plane namespace, such as `istio-system`.

```
$ oc adm must-gather --image=registry.redhat.io/openshift-service-mesh/istio-must-gather-rhel8
gather <namespace>
```

1.4. UNDERSTANDING RED HAT OPENSSHIFT 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.

1.4.1. 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
1.4.2. 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.

- **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 Istiod to enforce proxies to route traffic.

Istiod provides service discovery, configuration and certificate management. It converts high-level routing rules to Envoy configurations and propagates them to the sidecars at runtime.

Secret Discovery Service (SDS) distributes certificates and keys to sidecars directly from Istiod.

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.

1.4.3. Understanding Kiali

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

1.4.3.1. 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.

1.4.3.2. 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 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 are 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) 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 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 to view links to the Grafana dashboard and view Grafana data.

**1.4.3.3. 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.

1.4.4. 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 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—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 comprises 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.

1.4.4.1. 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.

1.4.4.2. 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.

1.4.4.3. 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.

1.4.5. Next steps

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

1.5. SERVICE MESH AND ISTIO DIFFERENCES

Red Hat OpenShift Service Mesh differs from an installation of Istio to provide additional features or to handle differences when deploying on OpenShift Container Platform.

1.5.1. Service Mesh features

The following features are different in Service Mesh and Istio.

1.5.1.1. Command line tool

The command line tool for Red Hat OpenShift Service Mesh is `oc`. Red Hat OpenShift Service Mesh does not support `istioctl`.
1.5.1.2. Installation and upgrades

Red Hat OpenShift Service Mesh does not support Istio installation profiles.

Red Hat OpenShift Service Mesh does not support canary upgrades of the service mesh.

1.5.1.3. Federation and multicluster

Red Hat OpenShift Service Mesh does not yet support federated or multiclustered service meshes.

- **Federated**: a set of Service Mesh control planes which interact with each other and are configured independently.

- **Clustered**: a set of Service Mesh control planes which act as a single control plane and are configured as a single entity.

1.5.1.4. 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.

1.5.1.5. 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**

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: httpbin-usernamepolicy
spec:
  action: ALLOW
  rules:
    - when:
      - key: 'request.regex.headers[username]'
        values:
        - "allowed.*"
  selector:
    matchLabels:
      app: httpbin
```
1.5.1.6. 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.

1.5.1.7. External workloads

Red Hat OpenShift Service Mesh does not support external workloads (virtual machines).

1.5.1.8. 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.

1.5.1.9. Envoy services

Red Hat OpenShift Service Mesh does not support QUIC-based services.

1.5.1.10. 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.

1.5.1.11. 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 Automatic route creation.

1.5.1.11.1. 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-admin can customize it. If you use Red Hat OpenShift Dedicated, refer to the Red Hat OpenShift Dedicated the dedicated-admin role.

1.5.1.11.2. 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.

1.5.1.11.3. 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.

1.5.1.11.4. WebAssembly Extensions

Red Hat OpenShift Service Mesh 2.0 introduces WebAssembly extensions to Envoy Proxy as a Technology Preview. Note that WASM extensions are not included in the proxy binary and that WASM filters from the upstream Istio community are not supported in Red Hat OpenShift Service Mesh 2.0.

Additional resources

- Automatic route creation

1.5.2. Multitenant installations

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.

1.5.2.1. 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.

### 1.5.2.2. 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.

### 1.5.3. 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.

- Do not 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. If you use Red Hat OpenShift Dedicated, updating the operator files should be restricted to those users with `dedicated-admin` privileges.

### 1.5.4. 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 when you select either the production or streaming deployment option.

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.

1.6. PREPARING TO INSTALL RED HAT OPENSOURCES SERVICE MESH

Before you can install Red Hat OpenShift Service Mesh, you must subscribe to OpenShift Container Platform and install OpenShift Container Platform in a supported configuration.

1.6.1. Prerequisites

- Maintain 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 4.8 overview.

- Install OpenShift Container Platform 4.8. If you are installing Red Hat OpenShift Service Mesh on a restricted network, follow the instructions for your chosen OpenShift Container Platform infrastructure.
  - Install OpenShift Container Platform 4.8 on AWS
  - Install OpenShift Container Platform 4.8 on user-provisioned AWS
  - Install OpenShift Container Platform 4.8 on bare metal
  - Install OpenShift Container Platform 4.8 on vSphere
  - Install OpenShift Container Platform 4.6 on IBM Z and LinuxONE
  - Install OpenShift Container Platform 4.6 on IBM Power Systems

- 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 4.8, see About the OpenShift CLI.

1.6.2. Supported configurations

The following configurations are supported for the current release of Red Hat OpenShift Service Mesh:

- Red Hat OpenShift Container Platform version 4.x.
NOTE

Red Hat OpenShift Online is 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, IBM Z, and IBM Power Systems.
  - IBM Z is only supported on OpenShift Container Platform 4.6 and later.
  - IBM Power Systems is only supported on OpenShift Container Platform 4.6 and later.
- Configurations where all Service Mesh components are contained in the OpenShift cluster in which it operates. Red Hat OpenShift Service Mesh does not support management of microservices that reside outside of the cluster, or in a multi-cluster scenario.
- Configurations that do not integrate external services such as virtual machines.

For additional information about Red Hat OpenShift Service Mesh lifecycle and supported configurations, refer to the Support Policy.

1.6.2.1. Supported network configurations

Red Hat OpenShift Service Mesh supports the following network configurations.

- OpenShift-SDN

- OVN-Kubernetes is supported as a technology preview in OpenShift Container Platform version 4.7.

1.6.2.2. Supported configurations for Kiali

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

1.6.2.3. Supported configurations for Distributed Tracing

- Jaeger agent as a sidecar is the only supported configuration for Jaeger. Jaeger as a daemonset is not supported for multitenant installations or OpenShift Dedicated.

1.6.2.4. Supported Mixer adapters

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

1.6.3. Next steps

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

1.7. INSTALLING THE OPERATORS
To install Red Hat OpenShift Service Mesh, first install the required Operators on OpenShift Container Platform and then create a ServiceMeshControlPlane resource to deploy the control plane.

**Prerequisites**

- Read the Preparing to install Red Hat OpenShift Service Mesh process.
- An account with the cluster-admin role. If you use Red Hat OpenShift Dedicated, you must have an account with the dedicated-admin role.

The following steps show how to install a basic instance of Red Hat OpenShift Service Mesh on OpenShift Container Platform.

### 1.7.1. Operator overview

Red Hat OpenShift Service Mesh requires the following four Operators:

- **OpenShift Elasticsearch** - (Optional) Provides database storage for tracing and logging with Jaeger. It is based on the open source Elasticsearch project.

- **Jaeger** - Provides tracing to monitor and troubleshoot transactions in complex distributed systems. It is based on the open source Jaeger project.

- **Kiali** - Provides observability for your service mesh. Allows you to view configurations, monitor traffic, and analyze traces in a single console. It is based on the open source Kiali project.

- **Red Hat OpenShift Service Mesh** - Allows you to connect, secure, control, and observe the microservices that comprise your applications. The Service Mesh Operator defines and monitors the ServiceMeshControlPlane resources that manage the deployment, updating, and deletion of the Service Mesh components. It is based on the open source Istio project.

### 1.7.2. Installing the Operators

To install Red Hat OpenShift Service Mesh, install following Operators in this order. Repeat the procedure for each Operator.

1. (Optional) OpenShift Elasticsearch
2. Jaeger
3. Kiali
4. Red Hat OpenShift Service Mesh

**Procedure**

1. Log in to the OpenShift Container Platform web console as a user with the cluster-admin role.
2. In the OpenShift Container Platform web console, click Operators → OperatorHub.
3. Type the name of the Operator into the filter box and select the Red Hat version of the Operator. Community versions of the Operators are not supported.
4. Click Install.
5. On the Install Operator page, select installation options.
a. For the OpenShift Elasticsearch Operator, in the Update Channel section, select **4.6**.

b. For the Jaeger, Kiali, and Red Hat OpenShift Service Mesh Operators, accept the defaults. The Jaeger, Kiali and Red Hat OpenShift Service Mesh are installed in the `openshift-operators` namespace. The OpenShift Elasticsearch Operator is installed in the `openshift-operators-redhat` namespace.

6. Click **Install**. Wait until the Operator has installed before repeating the steps for the next Operator in the list.

7. After all you have installed all four Operators, click **Operators → Installed Operators** to verify that your Operators installed.

### 1.7.3. Next steps

Create a **ServiceMeshControlPlane** resource to configure the components of Service Mesh. For more information, see **Creating the ServiceMeshControlPlane**.

### 1.8. CREATING THE SERVICEMESHCONTROLPLANE

You can deploy a basic installation of the **ServiceMeshControlPlane** by using either the OpenShift Container Platform web console or from the command line using the `oc` client tool.

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

You can deploy a basic **ServiceMeshControlPlane** by using the web console.

**Prerequisites**

- The Red Hat OpenShift Service Mesh Operator must be installed.
- 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. In the **Name** field, enter **istio-system**. The **ServiceMeshControlPlane** resource must be installed in a project that is separate from your microservices and Operators. These steps use **istio-system** as an example, but you can deploy your control plane in any project as long as it is separate from the project that contains your services.
   d. Click **Create**.

3. Navigate to **Operators → Installed Operators**.

4. Click the Red Hat OpenShift Service Mesh Operator, then click **Istio Service Mesh Control Plane**.
5. On the **Istio Service Mesh Control Plane** tab, click **Create ServiceMeshControlPlane**.

6. On the **Create ServiceMeshControlPlane** page, accept the default control plane version to take advantage of the features available in the most current version of the product. The version of the control plane determines the features available regardless of the version of the Operator. You can configure **ServiceMeshControlPlane** settings later. For more information, see Configuring Red Hat OpenShift Service Mesh.
   
   a. Click **Create**. The Operator creates pods, services, and Service Mesh control plane components based on your configuration parameters.

7. To verify the control plane installed correctly, click the **Istio Service Mesh Control Plane** tab.
   
   a. Click the name of the new control plane.

   b. Click the **Resources** tab to see the Red Hat OpenShift Service Mesh control plane resources the Operator created and configured.

1.8.2. Deploying the control plane from the CLI

You can deploy a basic **ServiceMeshControlPlane** from the command line.

**Prerequisites**

- The Red Hat OpenShift Service Mesh Operator must be installed.
- 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 following example. The version of the control plane determines the features available regardless of the version of the Operator.

**Example version 2.0 istio-installation.yaml**

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.0
  tracing:
    type: Jaeger
    sampling: 10000
addons:
```
Run the following command to deploy the control plane, where `<istio_installation.yaml>` includes the full path to your file.

```bash
$ oc create -n istio-system -f <istio_installation.yaml>
```

5. Run the following command to verify the control plane installation.

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

The installation has finished successfully when the `STATUS` column is `ComponentsReady`.

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

### 1.8.3. Next steps

Create a `ServiceMeshMemberRoll` resource to specify the namespaces associated with the Service Mesh. For more information, see Adding services to a service mesh.

### 1.9. ADDING SERVICES TO A SERVICE MESH

After installing the Operators and `ServiceMeshControlPlane` resource, add applications, workloads, or services to your mesh by creating a `ServiceMeshMemberRoll` resource and specifying the namespaces where your content is located. If you already have an application, workflow, or service to add to a `ServiceMeshMemberRoll` resource, use the following steps. Or, to install a sample application called Bookinfo and add it to a `ServiceMeshMemberRoll` resource, skip to the tutorial for installing the Bookinfo example application to see how an application works in Red Hat OpenShift Service Mesh.

The items listed in the `ServiceMeshMemberRoll` resource are the applications and workflows that are managed by the `ServiceMeshControlPlane` resource. The control plane, which includes the Service Mesh Operators, Istiod, and `ServiceMeshControlPlane`, and the data plane, which includes applications and Envoy proxy, must be in separate namespaces.

**NOTE**

After you add the namespace to the `ServiceMeshMemberRoll`, access to services or pods in that namespace will not be accessible to callers outside the service mesh.

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

```yaml
jaeger:
  name: jaeger
  install:
    storage:
      type: Memory
  kiali:
    enabled: true
    name: kiali
  grafana:
    enabled: true
```
The **ServiceMeshMemberRoll** lists the projects that belong 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**, for example `istio-system`.

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

You can add one or more projects to the Service Mesh member roll from the web console.

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

**Procedure**

1. Log in to the OpenShift Container Platform web console.
2. If you do not already have services 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**.
3. Navigate to **Operators → Installed Operators**.
4. Click the **Project** menu and choose the project where your **ServiceMeshControlPlane** resource is deployed from the list, for example `istio-system`.
5. Click the Red Hat OpenShift Service Mesh Operator.
6. Click the **Istio Service Mesh Member Roll** tab.
7. Click **Create ServiceMeshMemberRoll**
8. Click **Members**, then enter the name of your project in the **Value** field. You can add any number of projects, but a project can only belong to one **ServiceMeshMemberRoll** resource.
9. Click **Create**.

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

You can add a project to the **ServiceMeshMemberRoll** from the command line.

**Prerequisites**
- An installed, verified Red Hat OpenShift Service Mesh Operator.
- 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 https://{HOSTNAME}:6443

2. If you do not already have services for your mesh, or you are starting from scratch, create a project. It must be different from `istio-system`.

   $ oc new-project {your-project}

3. To add your projects as members, modify the following example YAML. You can add any number of projects, but a project can only belong to one `ServiceMeshMemberRoll` resource.

   **Example servicemeshmemberroll-default.yaml**

   ```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
   ```

4. Run the following command to upload and create the `ServiceMeshMemberRoll` resource in the `istio-system` namespace.

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

5. Run the following command to verify the `ServiceMeshMemberRoll` was created successfully.

   $ oc get smmr -n istio-system default

   The installation has finished successfully when the `STATUS` column is `Configured`.

1.9.2. Adding or removing projects from the service mesh

You can add or remove projects from 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.

1.9.2.1. Adding or removing projects from the member roll using 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` resource 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.

### 1.9.2.2. Adding or removing projects from the member roll using the CLI

You can 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.
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`

```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
```

1.9.3. Bookinfo example application

The Bookinfo example application allows you to test your Red Hat OpenShift Service Mesh 2.0.6 installation on OpenShift Container Platform.

The Bookinfo application displays information about a book, similar to a single catalog entry of an online book store. The application displays a page that describes 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.

1.9.3.1. Installing the Bookinfo application

This tutorial walks you through how to create a sample application by creating a project, deploying the Bookinfo application to that project, and viewing the running application in Service Mesh.

Prerequisites:
- OpenShift Container Platform 4.1 or higher installed.
- Red Hat OpenShift Service Mesh 2.0.6 installed.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as oc.
- An account with the `cluster-admin` role.

**NOTE**
The Bookinfo sample application cannot be installed on IBM Z and IBM Power Systems.

**Procedure**

1. Log in to the OpenShift Container Platform web console as a user with cluster-admin rights. If you use Red Hat OpenShift Dedicated, you must have an account with the `dedicated-admin` role.

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** tab.
   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 **ServiceMeshMemberRoll**, click **Create ServiceMeshMemberRoll**.

9. Click **Members**, then enter the name of your project in the **Value** field.

10. Click **Create** to save the updated Service Mesh Member Roll.
    - Or, save the following example to a YAML file.

```
Bookinfo ServiceMeshMemberRoll example servicemeshmemberroll-default.yaml

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

11. Run the following command to verify the `ServiceMeshMemberRoll` was created successfully.

```shell
$ oc smmr -n istio-system
```

The installation has finished successfully when the `STATUS` column is `Configured`.

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>1/1</td>
<td>Configured</td>
<td>2m27s</td>
</tr>
</tbody>
</table>

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

```shell
$ oc apply -n bookinfo -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.0/samples/bookinfo/platform/kube/bookinfo.yaml
```

You should see output similar to the following:

- service/details created
- serviceaccount/bookinfo-details created
- deployment.apps/details-v1 created
- service/ratings created
- serviceaccount/bookinfo-ratings created
- deployment.apps/ratings-v1 created
- service/reviews created
- serviceaccount/bookinfo-reviews created
- deployment.apps/reviews-v1 created
- deployment.apps/reviews-v2 created
- deployment.apps/reviews-v3 created
- service/productpage created
- serviceaccount/bookinfo-productpage created
- deployment.apps/productpage-v1 created

13. Create the ingress gateway by applying the `bookinfo-gateway.yaml` file:

```shell
$ oc apply -n bookinfo -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.0/samples/bookinfo/networking/bookinfo-gateway.yaml
```

You should see output similar to the following:

- gateway.networking.istio.io/bookinfo-gateway created
- virtualservice.networking.istio.io/bookinfo created

14. 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`.

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

1.9.3.2. Adding default destination rules

Before you can use the Bookinfo application, you must first 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:
     ```sh
     $ oc apply -n bookinfo -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.0/samples/bookinfo/networking/destination-rule-all.yaml
     ```
   - If you enabled mutual TLS:
     ```sh
     $ oc apply -n bookinfo -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.0/samples/bookinfo/networking/destination-rule-all-mtls.yaml
     ```
   You should see output similar to the following:
   
   ```
   destinationrule.networking.istio.io/productpage created
   destinationrule.networking.istio.io/reviews created
   destinationrule.networking.istio.io/ratings created
   destinationrule.networking.istio.io/details created
   ```

1.9.3.3. Verifying the Bookinfo installation

To confirm that the sample Bookinfo application was successfully deployed, perform the following steps.

**Prerequisites**

- Red Hat OpenShift Service Mesh 2.0.6 installed.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as `oc`.
- Complete the steps for installing the Bookinfo sample app.

**Procedure**

1. Log in to the OpenShift Container Platform CLI.
2. Verify that all pods are ready with this command:
All pods should have a status of **Running**. You should see output similar to the following:

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

3. Run the following command to retrieve the URL for the product page:

```
echo "http://$GATEWAY_URL/productpage"
```

4. Copy and paste the output in a web browser to verify the Bookinfo product page is deployed.

### 1.9.3.4. 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 2.0.6 installed.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as **oc**.

**1.9.3.4.1. Delete the Bookinfo project**

**Procedure**

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

**1.9.3.4.2. Remove the Bookinfo project from the Service Mesh member roll**

**Procedure**

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>details-v1-55b869668-jh7hb</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>12m</td>
</tr>
<tr>
<td>productpage-v1-6fc77f794-nsl8r</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>12m</td>
</tr>
<tr>
<td>ratings-v1-7d7d8d8b56-55scn</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>12m</td>
</tr>
<tr>
<td>reviews-v1-868597db96-bdxgq</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>12m</td>
</tr>
<tr>
<td>reviews-v2-5b64f47978-cvssp</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>12m</td>
</tr>
<tr>
<td>reviews-v3-6dfd49b55b-vcwpf</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>12m</td>
</tr>
</tbody>
</table>
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**. In this example, **istio-system** is the name of the control plane project.

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

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

**1.9.4. Next steps**

- To continue the installation process, you must **enable sidecar injection**.

**1.10. ENABLING SIDECAR INJECTION**

After adding your services to a mesh, enable automatic sidecar injection in the deployment resource for your application. You must enable automatic sidecar injection for each deployment.

If you have installed the Bookinfo sample application, the application was deployed and the sidecars were injected. If you are using your own project and service, deploy your applications on OpenShift Container Platform. For more information, see **Understanding Deployment and DeploymentConfig objects**.

**1.10.1. Prerequisites**

- **Adding services to a service mesh**

- **A deployment resource for your project**

**1.10.2. Enabling automatic sidecar injection**

When deploying an application, you must opt-in to injection by setting the **sidecar.istio.io/inject** annotation to "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.**
1. Open the application’s deployment configuration YAML file in an editor. To find a deployment use the `oc get` command. For example, for an app called `sleep` in the `sleep` namespace, use the following command to see the resource in YAML format.

```
oc get deployment sleep -o yaml
```

2. Add `sidecar.istio.io/inject` to the configuration YAML with a value of “true” in the `spec.template.metadata.annotations.sidecar.istio/inject` field. See the following example for an app called `sleep`.

**Sleep test application example sleep.yaml**

```yaml
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: curlimages/curl
    command: ["/bin/sleep","3650d"]
    imagePullPolicy: IfNotPresent
```

3. Save the configuration file.

4. Add the file back to the project that contains your app. In this example, `sleep` is the name of the project that contains the `sleep` app and `sleep.yaml` is the file you edited.

```
$ oc apply -n sleep -f sleep.yaml
```

5. To verify that the resource uploaded successfully, run the following command.

```
oc get deployment sleep -o yaml
```

**1.10.3. 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:

```bash
$ 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.

1.10.4. 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.

Example `injection-template.yaml`

```yaml
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.

1.10.5. Next steps

Configure Red Hat OpenShift Service Mesh features for your environment.

- **Security**
- **Traffic management**
1.11. UPGRADING RED HAT OPENSSHIFT SERVICE MESH FROM VERSION 1.1 TO VERSION 2.0

Upgrading from version 1.1 to 2.0 requires manual steps that migrate your workloads and application to a new instance of Red Hat OpenShift Service Mesh running the new version.

Prerequisites

- You must upgrade to OpenShift Container Platform 4.7. before you upgrade to Red Hat OpenShift Service Mesh 2.0.

- You must have Red Hat OpenShift Service Mesh version 2.0 operator. If you selected the automatic upgrade path, the operator automatically downloads the latest information. However, there are steps you must take to use the features in Red Hat OpenShift Service Mesh version 2.0.

1.11.1. Upgrading Red Hat OpenShift Service Mesh

To upgrade Red Hat OpenShift Service Mesh, you must create an instance of Red Hat OpenShift Service Mesh `ServiceMeshControlPlane` v2 resource in a new namespace. Then, once it’s configured, move your microservice applications and workloads from your old mesh to the new service mesh.

Procedure

1. Check your v1 `ServiceMeshControlPlane` resource configuration to make sure it is valid.
   a. Run the following command to view your `ServiceMeshControlPlane` resource as a v2 resource.

   ```bash
   $ oc get smcp -o yaml
   $ oc get smcp.v1.maistra.io <smcp_name> > smcp-resource.yaml
   #Edit the smcp-resource.yaml file.
   $ oc replace -f smcp-resource.yaml
   ```
   b. Check the `spec.techPreview.errored.message` field in the output for information about any invalid fields.
   c. If there are invalid fields in your v1 resource, the resource is not reconciled and cannot be edited as a v2 resource. All updates to v2 fields will be overridden by the original v1 settings. To fix the invalid fields, you can replace, patch, or edit the v1 version of the resource. You can also delete the resource without fixing it. After the resource has been fixed, it can be reconciled, and you can to modify or view the v2 version of the resource.
   d. To fix the resource by editing a file, use `oc get` to retrieve the resource, edit the text file locally, and replace the resource with the file you edited.

   ```bash
   $ oc get smcp.v1.maistra.io <smcp_name> > smcp-resource.yaml
   #Edit the smcp-resource.yaml file.
   $ oc replace -f smcp-resource.yaml
   ```
   e. To fix the resource using patching, use `oc patch`.

   ```bash
   $ oc patch smcp.v1.maistra.io <smcp_name> --type json --patch '[["op": "replace","path": "/spec/path/to/bad-setting","value": "corrected-value"]]'
f. To fix the resource by editing with command line tools, use `oc edit`.

```
$ oc edit smcp.v1.maistra.io <smcp_name>
```

2. Back up your control plane configuration. Switch to the project that contains your `ServiceMeshControlPlane` resource.

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

3. Enter the following command to retrieve the current configuration. Your `<smcp_name>` is specified in the metadata of your `ServiceMeshControlPlane` resource, for example `basic-install` or `full-install`.

```
$ oc get servicemeshcontrolplanes.v1.maistra.io <smcp_name> -o yaml > <smcp_name>.v1.yaml
```

4. Convert your `ServiceMeshControlPlane` to a v2 control plane version that contains information about your configuration as a starting point.

```
$ oc get smcp <smcp_name> -o yaml > <smcp_name>.v2.yaml
```

5. Create a project. In the OpenShift Container Platform console Project menu, click New Project and enter a name for your project, `istio-system-upgrade`, for example. Or, you can run this command from the CLI.

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

6. Update the `metadata.namespace` field in your v2 `ServiceMeshControlPlane` with your new project name. In this example, use `istio-system-upgrade`.

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

7. Update the `version` field from 1.1 to 2.0 or remove it in your v2 `ServiceMeshControlPlane`.

8. Create a `ServiceMeshControlPlane` in the new namespace. On the command line, run the following command to deploy the control plane with the v2 version of the `ServiceMeshControlPlane` that you retrieved. In this example, replace `<smcp_name>.v2> with the path to your file.

```
$ oc create -n istio-system-upgrade -f <smcp_name>.v2.yaml
```

Alternatively, you can use the console to create the control plane. In the OpenShift Container Platform web console, click Project. Then, select the project name you just entered.

a. Click Operators → Installed Operators.

b. Click Create ServiceMeshControlPlane.

c. Select YAML view and paste text of the YAML file you retrieved into the field. Check that the `apiVersion` field is set to `maistra.io/v2` and modify the `metadata.namespace` field to use the new namespace, for example `istio-system-upgrade`.

d. Click Create.

### 1.11.2. Configuring the 2.0 ServiceMeshControlPlane
The `ServiceMeshControlPlane` resource has been changed for Red Hat OpenShift Service Mesh version 2.0. After you created a v2 version of the `ServiceMeshControlPlane` resource, modify it to take advantage of the new features and to fit your deployment. Consider the following changes to the specification and behavior of Red Hat OpenShift Service Mesh 2.0 as you’re modifying your `ServiceMeshControlPlane` resource. You can also refer to the Red Hat OpenShift Service Mesh 2.0 product documentation for new information to features you use. The v2 resource must be used for Red Hat OpenShift Service Mesh 2.0 installations.

1.11.2.1. Architecture changes

The architectural units used by previous versions have been replaced by Istiod. In 2.0 the control plane components Mixer, Pilot, Citadel, Galley, and the sidecar injector functionality have been combined into a single component, Istiod.

Although Mixer is no longer supported as a control plane component, Mixer policy and telemetry plugins are now supported through WASM extensions in Istiod. Mixer can be enabled for policy and telemetry if you need to integrate legacy Mixer plugins.

Secret Discovery Service (SDS) is used to distribute certificates and keys to sidecars directly from Istiod. In Red Hat OpenShift Service Mesh version 1.1, secrets were generated by Citadel, which were used by the proxies to retrieve their client certificates and keys.

1.11.2.2. Annotation changes

The following annotations are no longer supported in v2.0. If you are using one of these annotations, you must update your workload before moving it to a v2.0 control plane.

- `sidecar.maistra.io/proxyCPULimit` has been replaced with `sidecar.istio.io/proxyCPULimit`. If you were using `sidecar.maistra.io` annotations on your workloads, you must modify those workloads to use `sidecar.istio.io` equivalents instead.

- `sidecar.maistra.io/proxyMemoryLimit` has been replaced with `sidecar.istio.io/proxyMemoryLimit`.

- `sidecar.maistra.io/discoveryAddress` is no longer supported. Also, the default discovery address has moved from `pilot.<control_plane_namespace>.svc:15010` (or port 15011, if mtls is enabled) to `istiod-<smcp_name>.<control_plane_namespace>.svc:15012`.

- The health status port is no longer configurable and is hard-coded to 15021. If you were defining a custom status port, for example, `status.sidecar.istio.io/port`, you must remove the override before moving the workload to a v2.0 control plane. Readiness checks can still be disabled by setting the status port to 0.

- Kubernetes Secret resources are no longer used to distribute client certificates for sidecars. Certificates are now distributed through Istiod’s SDS service. If you were relying on mounted secrets, they are longer available for workloads in v2.0 control planes.

1.11.2.3. Behavioral changes

Some features in Red Hat OpenShift Service Mesh 2.0 work differently than they did in previous versions.

- The readiness port on gateways has moved from 15020 to 15021.

- The target host visibility includes VirtualService, as well as ServiceEntry resources. It includes any restrictions applied through Sidecar resources.
Automatic mutual TLS is enabled by default. Proxy to proxy communication is automatically configured to use mTLS, regardless of global PeerAuthentication policies in place.

Secure connections are always used when proxies communicate with the control plane regardless of `spec.security.controlPlane.mtls` setting. The `spec.security.controlPlane.mtls` setting is only used when configuring connections for Mixer telemetry or policy.

1.11.2.4. Migration details for unsupported resources

**Policy (authentication.istio.io/v1alpha1)**

Policy resources must be migrated to new resource types for use with v2.0 control planes, PeerAuthentication and RequestAuthentication. Depending on the specific configuration in your Policy resource, you may have to configure multiple resources to achieve the same effect.

**Mutual TLS**

Mutual TLS enforcement is accomplished using the `security.istio.io/v1beta1` PeerAuthentication resource. The legacy `spec.peers.mtls.mode` field maps directly to the new resource’s `spec.mtls.mode` field. Selection criteria has changed from specifying a service name in `spec.targets[x].name` to a label selector in `spec.selector.matchLabels`. In PeerAuthentication, the labels must match the selector on the services named in the targets list. Any port-specific settings will need to be mapped into `spec.portLevelMtls`.

**Authentication**

Additional authentication methods specified in `spec.origins`, must be mapped into a `security.istio.io/v1beta1` RequestAuthentication resource. `spec.selector.matchLabels` must be configured similarly to the same field on PeerAuthentication. Configuration specific to JWT principals from `spec.origins.jwt` items map to similar fields in `spec.rules` items.

- `spec.origins[x].jwt.triggerRules` specified in the Policy must be mapped into one or more `security.istio.io/v1beta1` AuthorizationPolicy resources. Any `spec.selector.labels` must be configured similarly to the same field on RequestAuthentication.

- `spec.origins[x].jwt.triggerRules.excludedPaths` must be mapped into an AuthorizationPolicy whose `spec.action` is set to ALLOW, with `spec.rules[x].to.operation.path` entries matching the excluded paths.

- `spec.origins[x].jwt.triggerRules.includedPaths` must be mapped into a separate AuthorizationPolicy whose `spec.action` is set to ALLOW, with `spec.rules[x].to.operation.path` entries matching the included paths, and `spec.rules[x].from.source.requestPrincipals` entries that align with the specified `spec.origins[x].jwt.issuer` in the Policy resource.

**ServiceMeshPolicy (maistra.io/v1)**

ServiceMeshPolicy was configured automatically for the control plane through the `spec.istio.global.mtls.enabled` in the v1 resource or `spec.security.dataPlane.mtls` in the v2 resource setting. For v2 control planes, a functionally equivalent PeerAuthentication resource is created during installation. This feature is deprecated in Red Hat OpenShift Service Mesh version 2.0

**RbacConfig, ServiceRole, ServiceRoleBinding (rbac.istio.io/v1alpha1)**

These resources were replaced by the `security.istio.io/v1beta1` AuthorizationPolicy resource.

Mimicking RbacConfig behavior requires writing a default AuthorizationPolicy whose settings depend on the `spec.mode` specified in the RbacConfig.
- When `spec.mode` is set to **OFF**, no resource is required as the default policy is ALLOW, unless an AuthorizationPolicy applies to the request.

- When `spec.mode` is set to **ON**, set `spec: {}`. You must create AuthorizationPolicy policies for all services in the mesh.

- `spec.mode` is set to **ON_WITH_INCLUSION**, must create an AuthorizationPolicy with `spec: {}` in each included namespace. Inclusion of individual services is not supported by AuthorizationPolicy. However, as soon as any AuthorizationPolicy is created that applies to the workloads for the service, all other requests not explicitly allowed will be denied.

- When `spec.mode` is set to **ON_WITH_EXCLUSION**, it is not supported by AuthorizationPolicy. A global DENY policy can be created, but an AuthorizationPolicy must be created for every workload in the mesh because there is no allow-all policy that can be applied to either a namespace or a workload.

AuthorizationPolicy includes configuration for both the selector to which the configuration applies, which is similar to the function ServiceRoleBinding provides and the rules which should be applied, which is similar to the function ServiceRole provides.

**ServiceMeshRbacConfig (maistra.io/v1)**

This resource is replaced by using a `security.istio.io/v1beta1` AuthorizationPolicy resource with an empty `spec.selector` in the control plane’s namespace. This policy will be the default authorization policy applied to all workloads in the mesh. For specific migration details, see RbacConfig above.

**1.11.2.5. Mixer plugins**

Mixer components are disabled by default in version 2.0. If you rely on Mixer plugins for your workload, you must configure your version 2.0 `ServiceMeshControlPlane` to include the Mixer components.

To enable the Mixer policy components, add the following snippet to your `ServiceMeshControlPlane`.

```yaml
spec:
  policy:
    type: Mixer
```

To enable the Mixer telemetry components, add the following snippet to your `ServiceMeshControlPlane`.

```yaml
spec:
  telemetry:
    type: Mixer
```

Legacy mixer plugins can also be migrated to WASM and integrated using the new ServiceMeshExtension (maistra.io/valpha1) custom resource.

Built-in WASM filters included in the upstream Istio distribution are not available in Red Hat OpenShift Service Mesh 2.0.

**1.11.2.6. Mutual TLS changes**

When using mTLS with workload specific PeerAuthentication policies, a corresponding DestinationRule is required to allow traffic if the workload policy differs from the namespace/global policy.

Auto mTLS is enabled by default, but can be disabled by setting `spec.security.dataPlane.automtls` to
false in the `ServiceMeshControlPlane` resource. When disabling auto mTLS, DestinationRules may be required for proper communication between services. For example, setting PeerAuthentication to `STRICT` for one namespace may prevent services in other namespaces from accessing them, unless a DestinationRule configures TLS mode for the services in the namespace.

For information about mTLS, see [Enabling mutual Transport Layer Security (mTLS)](enabling-mutual-transport-layer-security-mtls).

### 1.11.2.6.1. Other mTLS Examples

To disable mTLS for `productpage` service in the bookinfo sample application, your Policy resource was configured the following way for Red Hat OpenShift Service Mesh v1.1.

**Example Policy resource**

```yaml
apiVersion: authentication.istio.io/v1alpha1
kind: Policy
metadata:
  name: productpage-mTLS-disable
  namespace: <namespace>
spec:
  targets:
  - name: productpage
```

To disable mTLS for `productpage` service in the bookinfo sample application, use the following example to configure your PeerAuthentication resource for Red Hat OpenShift Service Mesh v2.0.

**Example PeerAuthentication resource**

```yaml
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: productpage-mTLS-disable
  namespace: <namespace>
spec:
  mtls:
    mode: DISABLE
  selector:
    matchLabels:
    # this should match the selector for the "productpage" service
    app: productpage
```

To enable mTLS with JWT authentication for the `productpage` service in the bookinfo sample application, your Policy resource was configured the following way for Red Hat OpenShift Service Mesh v1.1.

**Example Policy resource**

```yaml
apiVersion: authentication.istio.io/v1alpha1
kind: Policy
metadata:
  name: productpage-mTLS-with-JWT
  namespace: <namespace>
spec:
  targets:
  - name: productpage
```
To enable mTLS with JWT authentication for the productpage service in the bookinfo sample application, use the following example to configure your PeerAuthentication resource for Red Hat OpenShift Service Mesh v2.0.

Example PeerAuthentication resource

```yaml
#require mTLS for productpage:9000
apiVersion: security.istio.io/v1beta1
category: PeerAuthentication
metadata:
  name: productpage-mTLS-with-JWT
  namespace: <namespace>
spec:
  selector:
    matchLabels:
      # this should match the selector for the "productpage" service
      app: productpage
  portLevelMtls:
    9000:
      mode: STRICT

#JWT authentication for productpage
apiVersion: security.istio.io/v1beta1
category: RequestAuthentication
metadata:
  name: productpage-mTLS-with-JWT
  namespace: <namespace>
spec:
  selector:
    matchLabels:
      # this should match the selector for the "productpage" service
      app: productpage
  jwtRules:
  - issuer: "https://securetoken.google.com"
    audiences:
      - "productpage"
    jwksUri: "https://www.googleapis.com/oauth2/v1/certs"
    jwtHeaders:
      - name: x-goog-iap-jwt-assertion
    triggerRules:
      - excluded Paths:
        - exact: /health_check
  principal Binding: USE_ORIGIN
```
1.11.3. Configuration recipes

You can configure the following items with these configuration recipes.

1.11.3.1. Mutual TLS in a data plane

Mutual TLS for data plane communication is configured through `spec.security.dataPlane.mtls` in the `ServiceMeshControlPlane` resource, which is `false` by default.

1.11.3.2. Custom signing key

Istiod manages client certificates and private keys used by service proxies. By default, Istiod uses a self-signed certificate for signing, but you can configure a custom certificate and private key. For more information about how to configure signing keys, see Adding an external certificate authority key and certificate

1.11.3.3. Tracing

Tracing is configured in `spec.tracing`. Currently, the only type of tracer that is supported is Jaeger. Sampling is a scaled integer representing 0.01% increments, for example, 1 is 0.01% and 10000 is 100%. The tracing implementation and sampling rate can be specified:

```yaml
...
#Require JWT token to access product page service from
#any client to all paths except /health_check
apiVersion: security.istio.io/v1beta1
document:
    kind: AuthorizationPolicy
data:
    name: productpage-mTLS-with-JWT
    namespace: <namespace>
    spec:
        action: ALLOW
        selector:
            matchLabels:
                # this should match the selector for the "productpage" service
                app: productpage
        rules:
            - to: # require JWT token to access all other paths
                operation:
                    notPaths:
                        - /health_check
                from:
                    - source:
                        # if using principalBinding: USE_PEER in the Policy,
                        # then use principals, e.g.
                        # principals:
                        #   - "*
                        requestPrincipals:
                            - "*
            - to: # no JWT token required to access health_check
                operation:
                    paths:
                        - /health_check
```
Jaeger is configured in the addons section of the `ServiceMeshControlPlane` resource.

```yaml
spec:
  tracing:
    sampling: 100 # 1%
    type: Jaeger

The Jaeger installation can be customized with the `install` field. Container configuration, such as resource limits is configured in `spec.runtime.components.jaeger` related fields. If a Jaeger resource matching the value of `spec.addons.jaeger.name` exists, the control plane will be configured to use the existing installation. Use an existing Jaeger resource to fully customize your Jaeger installation.

```

```yaml
spec:
  addons:
    jaeger:
      name: jaeger
      install:
        storage:
          type: Memory # or Elasticsearch for production mode
          memory: 100000
          elasticsearch: # the following values only apply if storage:type=Elasticsearch
          storage: # specific storageclass configuration for the Jaeger Elasticsearch (optional)
          size: "100G"
          storageClassName: "storageclass"
          nodeCount: 3
          redundancyPolicy: SingleRedundancy
        runtime:
          components:
            tracing.jaeger: {} # general Jaeger specific runtime configuration (optional)
            tracing.jaeger.elasticsearch: #runtime configuration for Jaeger Elasticsearch deployment (optional)
              container:
                resources:
                  requests:
                    memory: "1Gi"
                    cpu: "500m"
                  limits:
                    memory: "1Gi"
```

The Jaeger installation can be customized with the `install` field. Container configuration, such as resource limits is configured in `spec.runtime.components.jaeger` related fields. If a Jaeger resource matching the value of `spec.addons.jaeger.name` exists, the control plane will be configured to use the existing installation. Use an existing Jaeger resource to fully customize your Jaeger installation.

1.11.3.4. Visualization

Kiali and Grafana are configured under the addons section of the `ServiceMeshControlPlane` resource.

```yaml
spec:
  addons:
    grafana:
      enabled: true
      install: {} # customize install
    kiali:
      enabled: true
      name: kiali
      install: {} # customize install
```

The Grafana and Kiali installations can be customized through their respective `install` fields. Container
customization, such as resource limits, is configured in `spec.runtime.components.kiali` and `spec.runtime.components.grafana`. If an existing Kiali resource matching the value of name exists, the control plane configures the Kiali resource for use with the control plane. Some fields in the Kiali resource are overridden, such as the `accessible_namespaces` list, as well as the endpoints for Grafana, Prometheus, and tracing. Use an existing resource to fully customize your Kiali installation.

1.11.3.5. Resource utilization and scheduling

Resources are configured under `spec.runtime.<component>`. The following component names are supported.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Versions supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>security</td>
<td>Citadel container</td>
<td>v1.0/1.1</td>
</tr>
<tr>
<td>galley</td>
<td>Galley container</td>
<td>v1.0/1.1</td>
</tr>
<tr>
<td>pilot</td>
<td>Pilot/Istiod container</td>
<td>v1.0/1.1/2.0</td>
</tr>
<tr>
<td>mixer</td>
<td>istio–telemetry and istio–policy containers</td>
<td>v1.0/1.1</td>
</tr>
<tr>
<td>mixer.policy</td>
<td>istio-policy container</td>
<td>v2.0</td>
</tr>
<tr>
<td>mixer.telemetry</td>
<td>istio–telemetry container</td>
<td>v2.0</td>
</tr>
<tr>
<td>global.oauthproxy</td>
<td>oauth-proxy container used with various addons</td>
<td>v1.0/1.1/2.0</td>
</tr>
<tr>
<td>sidecarInjectorWebhook</td>
<td>sidecar injector webhook container</td>
<td>v1.0/1.1</td>
</tr>
<tr>
<td>tracing.jaeger</td>
<td>general Jaeger container - not all settings may be applied. Complete customization of Jaeger installation is supported by specifying an existing Jaeger resource in the control plane configuration.</td>
<td>v1.0/1.1/2.0</td>
</tr>
<tr>
<td>tracing.jaeger.agent</td>
<td>settings specific to Jaeger agent</td>
<td>v1.0/1.1/2.0</td>
</tr>
<tr>
<td>tracing.jaeger.allInOne</td>
<td>settings specific to Jaeger allInOne</td>
<td>v1.0/1.1/2.0</td>
</tr>
<tr>
<td>tracing.jaeger.collector</td>
<td>settings specific to Jaeger collector</td>
<td>v1.0/1.1/2.0</td>
</tr>
<tr>
<td>tracing.jaeger.elasticsearch</td>
<td>settings specific to Jaeger elasticsearch deployment</td>
<td>v1.0/1.1/2.0</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
<td>Versions supported</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>tracing.jaeger.query</td>
<td>settings specific to Jaeger query</td>
<td>v1.0/1.1/2.0</td>
</tr>
<tr>
<td>prometheus</td>
<td>prometheus container</td>
<td>v1.0/1.1/2.0</td>
</tr>
<tr>
<td>kiali</td>
<td>Kiali container - complete customization of Kiali installation is supported by specifying an existing Kiali resource in the control plane configuration.</td>
<td>v1.0/1.1/2.0</td>
</tr>
<tr>
<td>grafana</td>
<td>Grafana container</td>
<td>v1.0/1.1/2.0</td>
</tr>
<tr>
<td>3scale</td>
<td>3scale container</td>
<td>v1.0/1.1/2.0</td>
</tr>
<tr>
<td>wasmExtensions.cacher</td>
<td>WASM extensions cacher container</td>
<td>v2.0 - tech preview</td>
</tr>
</tbody>
</table>

Some components support resource limiting and scheduling. For more information, see **Performance and scalability**.

### 1.11.4. Next steps for migrating your applications and workflows

Move the application workload to the new mesh and remove the old instances to complete your upgrade.

### 1.12. MANAGING USERS AND PROFILES

#### 1.12.1. Creating the Red Hat OpenShift Service Mesh members

_ServiceMeshMember_ resources provide a way for Red Hat OpenShift Service Mesh administrators to delegate permissions to add projects to a service mesh, even when the respective users don’t have direct access to the service mesh project or member roll. 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. In this example, _istio-system_ is the control plane namespace.

```
$ oc policy add-role-to-user -n istio-system --role-namespace istio-system 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 that it references.

```yaml
apiVersion: maistra.io/v1
kind: ServiceMeshMember
metadata:
  name: default
spec:
```
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**:

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

### 1.12.2. Creating control plane profiles

You can create reusable configurations with **ServiceMeshControlPlane** profiles. Individual users can extend the profiles they create with their own configurations. Profiles can also inherit configuration information from other profiles. 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 profiles with team-specific customization.

When you configure control plane profiles, which follow the same syntax as the **ServiceMeshControlPlane**, users inherit settings in a hierarchical fashion. The Operator is delivered with a **default** profile with default settings for Red Hat OpenShift Service Mesh.

#### 1.12.2.1. Creating the ConfigMap

To add custom profiles, you must first create a ConfigMap named **smcp-templates** in the **openshift-operators** project and then mount the ConfigMap in the Operator container at: `/usr/local/share/istio-operator/templates`.

**Prerequisites**

- An installed, verified Service Mesh Operator.
An account with the **cluster-admin** role. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-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-admin**. If you use Red Hat OpenShift Dedicated, you must have an account with the **dedicated-admin** role.

2. From the CLI, run this command to create the ConfigMap named **smcp-templates** in the **openshift-operators** project and replace `<profiles-directory>` with the location of the **ServiceMeshControlPlane** files on your local disk:

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

3. Locate the Operator ClusterServiceVersion name.

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

   **Example output**

   ```
   maistra.v2.0.0    Red Hat OpenShift Service Mesh  2.0.0    Succeeded
   ```

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

   ```bash
   $ oc edit clusterserviceversion -n openshift-operators servicemeshoperator.v2.0.0.1
   ```

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

   ```yaml
   deployments:
   - name: istio-operator
     spec:
       template:
         spec:
           containers:
             volumeMounts:
             - name: smcp-templates
               mountPath: /usr/local/share/istio-operator/templates/
           volumes:
             - name: smcp-templates
               configMap:
                 name: smcp-templates
   ...
   ```

6. Save your changes and exit the editor.

7. You can use the **profiles** parameter in the **ServiceMeshControlPlane** to specify one or more templates.

   ```yaml
   apiVersion: maistra.io/v2
   ```
1.12.2.2. 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 service mesh that were previously exposed through an OpenShift Container Platform route.

- Traffic into the service mesh must always go through the ingress-gateway for Istio to work properly.
- Deploy services external to the service mesh in separate namespaces that are not in any service 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.

1.13. SECURITY

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 help you manage the complexity of your applications and secure microservices.

Before you begin

If you have a project, add your project to the `ServiceMeshMemberRoll` resource.

If you don’t have a project, install the Bookinfo sample application and add it to the `ServiceMeshMemberRoll` resource. The sample application helps illustrate security concepts.

1.13.1. Mutual Transport Layer Security (mTLS)

Mutual Transport Layer Security (mTLS) is a protocol that enables two parties authenticate each other. 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, mTLS in Red Hat OpenShift Service Mesh is enabled and 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. Then, you can enable strict mTLS across your mesh, namespace, or application.

Enabling mTLS across your mesh at the control plane level secures all the traffic in your service mesh without rewriting your applications and workflows. You can secure namespaces in your mesh at the data plane level in the `ServiceMeshControlPlane` resource. To customize traffic encryption connections, configure namespaces at the application level with `PeerAuthentication` and `DestinationRule` resources.
1.13.1.1. Enabling strict mTLS across the service mesh

If your workloads do not communicate with outside services, you can quickly enable mTLS across your mesh without communication interruptions. You can enable it by setting `spec.security.dataPlane.mtls` to `true` in the `ServiceMeshControlPlane` resource. The Operator creates the required resources.

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  version: v2.0
  security:
    dataPlane:
      mtls: true
```

You can also enable mTLS by using the OpenShift Container Platform web console.

**Procedure**

1. Log in to the web console.
2. Click the Project menu and choose the `istio-system` project from the list.
3. Click Operators → Installed Operators.
4. Click Service Mesh Control Plane under Provided APIs.
5. Click the name of your `ServiceMeshControlPlane` resource, for example, `basic`.

1.13.1.1.1. Configuring sidecars for incoming connections for specific services

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

**Procedure**

1. Create a YAML file using the following example.

   ```yaml
   PeerAuthentication Policy example policy.yaml
   ```

   ```yaml
   apiVersion: security.istio.io/v1beta1
   kind: PeerAuthentication
   metadata:
     name: default
     namespace: <namespace>
   spec:
     mtls:
       mode: STRICT
   a. Replace `<namespace>` with the namespace where the service is located.

2. Run the following command to create the resource in the namespace where the service is located. It must match the `namespace` field in the Policy resource you just created.

   ```bash
   $ oc create -n <namespace> -f <policy.yaml>
   ```
NOTE

If you are not using automatic mTLS and you are setting `PeerAuthentication` to STRICT, you must create a `DestinationRule` resource for your service.

1.13.1.1.2. 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.

Procedure

1. Create a YAML file using the following example.

```yaml
DestinationRule example destination-rule.yaml
```

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

a. Replace `<namespace>` with the namespace where the service is located.

2. Run the following command to create the resource in the namespace where the service is located. It must match the `namespace` field in the `DestinationRule` resource you just created.

```
$ oc create -n <namespace> -f <destination-rule.yaml>
```

1.13.1.1.3. 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.security.controlPlane.tls.minProtocolVersion` or `spec.security.controlPlane.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.

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

Table 1.3. Valid values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS_AUTO</td>
<td>default</td>
</tr>
<tr>
<td>TLSv1_0</td>
<td>TLS version 1.0</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>TLSv1_1</td>
<td>TLS version 1.1</td>
</tr>
<tr>
<td>TLSv1_2</td>
<td>TLS version 1.2</td>
</tr>
<tr>
<td>TLSv1_3</td>
<td>TLS version 1.3</td>
</tr>
</tbody>
</table>

Procedure

1. Log in to the web console.
2. Click the Project menu and choose the istio-system project from the list.
3. Click Operators → Installed Operators.
4. Click Service Mesh Control Plane under Provided APIs.
5. Click the name of your ServiceMeshControlPlane resource, for example, basic.
6. Click the YAML tab.
7. Insert the following code snippet in the YAML editor. Replace the value in the minProtocolVersion with the TLS version value. In this example, the minimum TLS version is set to TLSv1_2.

```
ServiceMeshControlPlane snippet

kind: ServiceMeshControlPlane
spec:
  security:
    controlPlane:
      tls:
        minProtocolVersion: TLSv1_2
```
8. Click Save.
9. Click Refresh to verify that the changes updated correctly.

1.13.2. Configuring Role Based Access Control (RBAC)

Role-based access control (RBAC) objects determine whether a user or service is allowed to perform a given action within a project. You can define mesh-, namespace-, and workload-wide access control for your workloads in the mesh.

To configure RBAC, create an AuthorizationPolicy resource in the namespace for which you are configuring access. If you are configuring mesh-wide access, use istio-system.

For example, with RBAC, you can create policies that:

- Configure intra-project communication.
- Allow or deny full access to all workloads in the default namespace.
- Allow or deny ingress gateway access.
- Require a token for access.

An authorization policy includes a selector, an action, and a list of rules:

- The **selector** field specifies the target of the policy.
- The **action** field specifies whether to allow or deny the request.
- The **rules** field specifies when to trigger the action.
  - The **from** field specifies constraints on the request origin.
  - The **to** field specifies constraints on request target and parameters.
  - The **when** field specifies additional conditions that to apply the rule.

**Procedure**

1. Create your **AuthorizationPolicy** resource. The following example shows a resource that updates the ingress-policy **AuthorizationPolicy** to deny an IP address from accessing the ingress gateway.

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: DENY
  rules:
    - from:
      - source:
        ipBlocks: 
          - "1.2.3.4"
```

2. Run the following command after you write your resource to create your resource in your namespace. The namespace must match your `metadata.namespace` field in your **AuthorizationPolicy** resource.

```
$ oc create -n istio-system -f <filename>
```

**Next steps**

Consider the following examples for other common configurations.

**1.13.2.1. Configure intra-project communication**

You can use **AuthorizationPolicy** to configure your control plane to allow or deny the traffic communicating with your mesh or services in your mesh.

**1.13.2.1.1. Restrict access to services outside a namespace**
You can deny requests from any source that is not in the `bookinfo` namespace with the following `AuthorizationPolicy` resource example.

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: httpbin-deny
  namespace: bookinfo
spec:
  selector:
    matchLabels:
      app: httpbin
      version: v1
  action: DENY
  rules:
    - from:
      - source: 
        notNamespaces: ["bookinfo"]
```

### 1.13.2.1.2. Creating allow-all and default deny-all authorization policies

The following example shows an allow-all authorization policy that allows full access to all workloads in the `bookinfo` namespace.

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-all
  namespace: bookinfo
spec:
  action: ALLOW
  rules:
  - {}
```

The following example shows a policy that denies any access to all workloads in the `bookinfo` namespace.

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: bookinfo
spec:
  {}  
```

### 1.13.2.2. Allow or deny access to the ingress gateway

You can set an authorization policy to add allow or deny lists based on IP addresses.

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
```
1.13.2.3. Restrict access with JSON Web Token

You can restrict what can access your mesh with a JSON Web Token (JWT). After authentication, a user or service can access routes, services that are associated with that token.

Create a `RequestAuthentication` resource, which defines the authentication methods that are supported by a workload. The following example accepts a JWT issued by `http://localhost:8080/auth/realms/master`.

```yaml
apiVersion: "security.istio.io/v1beta1"
kind: "RequestAuthentication"
metadata:
  name: "jwt-example"
  namespace: bookinfo
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: ALLOW
  rules:
    - from:
    - source:
      ipBlocks: ["1.2.3.4", "5.6.7.0/24"]
```

Then, create an `AuthorizationPolicy` resource in the same namespace to work with `RequestAuthentication` resource you created. The following example requires a JWT to be present in the `Authorization` header when sending a request to `httpbin` workloads.

```yaml
apiVersion: "security.istio.io/v1beta1"
kind: "AuthorizationPolicy"
metadata:
  name: "frontend-ingress"
  namespace: bookinfo
spec:
  selector:
    matchLabels:
      app: httpbin
  action: DENY
  rules:
    - from:
    - source:
      notRequestPrincipals: ["]"
```

1.13.3. 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_RSA_WITH_3DES_EDE_CBC_SHA`
- `TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA`

The supported ECDH Curves are:
1.13.4. Adding an external certificate authority key and certificate

By default, Red Hat OpenShift Service Mesh generates a 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

- Install 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.
- Deploy the Bookinfo sample application to verify the results with these instructions.

1.13.4.1. 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 named `ca-cert.pem`, the key is `ca-key.pem`, and the root certificate, which signs `ca-cert.pem`, is named `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 `spec.security.dataPlane.mtls: true` to `true` and configure your certificateAuthority like the following example. The default `rootCADir` is `/etc/cacerts`. You do not need to set the `privateKey` if the key and certs are mounted in the default location. Service Mesh reads the certificates and key from the secret-mount files.

   ```yaml
   apiVersion: maistra.io/v2
   kind: ServiceMeshControlPlane
   spec:
     security:
       dataPlane:
         mtls: true
       certificateAuthority:
         type: Istiod
   ```
istiod:
  type: PrivateKey
  privateKey:
    rootCADir: /etc/cacerts

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.

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

1.13.4.2. 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`.

```bash
$ 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.

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

The file `/tmp/pod-root-cert.pem` contains the root 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.

```bash
$ 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.

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

```
$ 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.
```

Example output

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

### 1.13.4.3. Removing the certificates

To remove the certificates you added, follow these steps.

1. Remove the secret `cacerts`.

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


   ```
   apiVersion: maistra.io/v2
   kind: ServiceMeshControlPlane
   spec:
     dataPlane:
       mtls: true
   ```

### 1.14. CONFIGURING TRAFFIC MANAGEMENT

Red Hat OpenShift Service Mesh allows you to control the flow of traffic and API calls between services. 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.

### 1.14.1. Routing tutorial

The Service Mesh Bookinfo sample application consists of four separate microservices, each with multiple versions. After installing the Bookinfo sample application, three different versions of the `reviews` microservice run concurrently.
When you 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.

Prerequisites:

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

1.14.1.1. Applying a virtual service

In the following procedure, the virtual service routes all traffic to v1 of each micro-service by applying virtual services that set the default version for the micro-services.

Procedure

1. Apply the virtual services.

   $ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.0/samples/bookinfo/networking/virtual-service-all-v1.yaml

2. To verify that you applied the virtual services, display the defined routes with the following command:

   $ oc get virtualservices -o yaml

   That command returns a resource of kind: VirtualService in YAML format.

You have configured Service Mesh to route to the v1 version of the Bookinfo microservices including the reviews service version 1.

1.14.1.2. Testing the new route configuration

Test the new configuration by refreshing the /productpage of the Bookinfo application.

Procedure

1. Set the value for the GATEWAY_URL parameter. You can use this variable to find the URL for your Bookinfo product page later. In this example, istio-system is the name of the control plane project.

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

2. Run the following command to retrieve the URL for the product page.

   echo "http://$GATEWAY_URL/productpage"

3. Open the Bookinfo site in your browser.
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.

1.14.1.3. Route based on user identity

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.

Service Mesh does not 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.

Procedure

1. Run the following command to enable user-based routing in the Bookinfo sample application.

   $ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.0/samples/bookinfo/networkingirtual-service-reviews-test-v2.yaml

2. Run the following command to confirm the rule is created. This command returns all resources of kind: VirtualService in YAML format.

   $ oc get virtualservice reviews -o yaml

3. On the /productpage of the Bookinfo app, log in as user jason with no password.
   a. Refresh the browser. 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. Traffic is now routed to reviews:v1 for all users except Jason.

You have successfully configured the Bookinfo sample application to route traffic based on user identity.

1.14.2. 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.

1.14.2.1. 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. A virtual service enables you to turn a monolithic application into a service comprised of distinct microservices with a seamless consumer experience.

- Configure traffic rules in combination with gateways to control ingress and egress traffic.

1.14.2.1.1. Configuring virtual services

OpenShift Container Platform 4.8 Service Mesh
Requests are routed to 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.

**Procedure**

1. Create a YAML file using the following example to route requests to different versions of the Bookinfo sample application service depending on which user connects to the application.

   **Example VirtualService.yaml**

   ```yaml
   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
   ```

2. Run the following command to apply `VirtualService.yaml`, where `VirtualService.yaml` is the path to the file.

   ```bash
   $ oc apply -f VirtualService.yaml
   ```

**1.14.2.2. Configuring your virtual host**

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

**1.14.2.2.1. Hosts**

The `hosts` field lists the virtual service’s destination address to which the routing rules apply. This is the address(es) that are used to send requests to the service.
The virtual service hostname can be an IP address, a DNS name, or a short name that resolves to a fully qualified domain name.

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

### 1.14.2.2. Routing rules

The **http** section contains the virtual service’s routing rules which describe match conditions and actions for routing HTTP/1.1, HTTP2, and gRPC traffic sent to the destination as specified in the hosts field. A routing rule consists of the destination where you want the traffic to go and any specified match conditions.

**Match condition**

The first routing rule in the example has a condition that 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
```

### 1.14.2.3. 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.
1.14.2.2.3.1. Load balancing options

By default, Red Hat OpenShift Service Mesh uses a round-robin load balancing policy, where each service instance in the 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:

```yaml
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
```

1.14.2.2.4. 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 allow 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 and configure 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 enables you to configure a dedicated exit node for the traffic leaving the mesh. This enables you to limit which services have access to external networks, which adds security control to your service mesh. You can also use a gateway to configure a purely internal proxy.

**Gateway example**

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

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: ext-host-gwy
spec:
  selector:
    istio: ingressgateway
  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:

```yaml
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.

### 1.14.2.2.5. 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 send traffic to the service as if it is a service in your mesh. Service entries allow you to do the following:

- Manage traffic for services that run outside of the service mesh.
- 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.

**NOTE**

Add services from a different cluster to the mesh to configure a multicluster Red Hat OpenShift Service Mesh mesh on Kubernetes.

**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:

```yaml
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:

```yaml
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
```

1.14.3. Managing ingress traffic

In Red Hat OpenShift Service Mesh, the Ingress Gateway enables features such as monitoring, security, and route rules to apply to traffic that enters the cluster. Use a Service Mesh gateway to expose a service outside of the service mesh.
1.14.3.1. Determining the ingress IP and ports

Ingress configuration differs depending on if your environment supports an external load balancer. An external load balancer is set in the ingress IP and ports for the cluster. To determine if your cluster’s IP and ports are configured for external load balancers, run the following command.

```bash
$ 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.

Determine the ingress according to your environment. For an environment with load balancer support, [Determining ingress ports with a load balancer](#). For an environment without load balancer support, [Determining ingress ports without a load balancer](#). After you have determined the ingress ports, see [Configuring ingress using a gateway](#) to complete your configuration.

1.14.3.1.1. Determining ingress ports with a load balancer

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

**Procedure**

1. Run the following command to set the ingress IP and ports. This command sets a variable in your terminal.

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

2. Run the following command to set the ingress port.

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

3. Run the following command to set the secure ingress port.

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

4. Run the following command to set the TCP ingress port.

   ```bash
   $ export TCP_INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="tcp")].port}')
   ```
NOTE

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 an IP address. Instead, it’s a host name, and the previous command fails to set the INGRESS_HOST environment variable.

In that case, 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}')
```

1.14.3.1.2. Determining ingress ports without a load balancer

If your environment does not have an external load balancer, determine the ingress ports and use a node port instead.

Procedure

1. Set the ingress ports.

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

2. Run the following command to set the secure ingress port.

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

3. Run the following command to set the TCP ingress port.

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

1.14.4. Configuring ingress using a gateway

An ingress gateway is a load balancer operating at the edge of the mesh that receives incoming HTTP/TCP connections. It configures exposed ports and protocols but does not include any traffic routing configuration. Traffic routing for ingress traffic is instead configured with routing rules, the same way as for internal service requests.

The following steps show how to create a gateway and configure a VirtualService to expose a service in the Bookinfo sample application to outside traffic for paths /productpage and /login.

Procedure

1. Create a gateway to accept traffic.

   a. Create a YAML file, and copy the following YAML into it.

   ```yaml
   Gateway example gateway.yaml
   
   apiVersion: networking.istio.io/v1alpha3
   ```
2. Create a VirtualService object to rewrite the host header.

   a. Create a YAML file, and copy the following YAML into it.

   **Virtual service example vs.yaml**

   ```yaml
   apiVersion: networking.istio.io/v1alpha3
   kind: VirtualService
   metadata:
      name: bookinfo
   spec:
      hosts: 
        - "*"
      gateways:
        - bookinfo-gateway
      http:
        - match:
            - uri:
                exact: /productpage
            - uri:
                prefix: /static
            - uri:
                exact: /login
            - uri:
                exact: /logout
            - uri:
                prefix: /api/v1/products
        route:
          - destination:
              host: productpage
              port:
                number: 9080
   ```

   b. Apply the YAML file.

   ```bash
   $ oc apply -f vs.yaml
   ```

82
3. Test that the gateway and VirtualService have been set correctly.
   a. Set the Gateway URL.
   
   ```bash
   export GATEWAY_URL=$(oc -n istio-system get route istio-ingressgateway -o jsonpath='{.spec.host}')
   ```
   
   b. Set the port number.
   
   ```bash
   export TARGET_PORT=$(oc -n istio-system get route istio-ingressgateway -o jsonpath='{.spec.port.targetPort}')
   ```
   
   c. Test a page that has been explicitly exposed.
   
   ```bash
   curl -s -I "$GATEWAY_URL/productpage"
   ```
   
   The expected result is **200**.

### 1.14.5. Automatic routes

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

#### 1.14.5.1. 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. Configure an OpenShift Container Platform wildcard policy before configuring a wildcard host Gateway. For more information, see [Using wildcard routes](#).

#### 1.14.5.2. Creating subdomain routes

The following example creates a gateway in the Bookinfo sample application, which creates subdomain routes.

```yaml
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. In this example, `istio-system` is the name of the control plane project.

```bash
$ oc -n istio-system get routes
```

**Expected output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>HOST/PORT</th>
<th>PATH</th>
<th>SERVICES</th>
<th>PORT</th>
<th>TERMINATION</th>
<th>WILDCARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>gateway1-lvlfn</td>
<td>bookinfo.example.com</td>
<td></td>
<td>istio-ingressgateway</td>
<td>&lt;all&gt;</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>gateway1-scqhv</td>
<td><a href="http://www.bookinfo.com">www.bookinfo.com</a></td>
<td></td>
<td>istio-ingressgateway</td>
<td>&lt;all&gt;</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

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.

1.14.5.3. Annotations

Sometimes specific annotations are needed in an OpenShift Route. For example, some advanced features in OpenShift Routes are managed via special annotations. For this and other use cases, Red Hat OpenShift Service Mesh will copy all annotations present in the Istio Gateway resource (with the exception of those starting with `kubectl.kubernetes.io`) into the managed OpenShift Route resource.

If you need specific annotations in the OpenShift Routes created by Service Mesh, create them in the Istio Gateway resource and they will be copied into the OpenShift Route resources managed by the Service Mesh.

1.14.5.4. Disabling automatic route creation

By default, the `ServiceMeshControlPlane` resource automatically synchronizes the Gateway resources with OpenShift routes. Disabling the automatic route creation allows you more flexibility to control routes if you have a special case or prefer to control routes manually.

Disable integration between Istio Gateways and OpenShift Routes by setting the `ServiceMeshControlPlane` field `gateways.openshiftRoute.enabled` to `false`. For example, see the following resource snippet.

```yaml
spec:
gateways:
  openshiftRoute:
    enabled: false
```

1.14.5.5. 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.
NOTE
To optimize performance of your service mesh, consider limiting Envoy proxy configurations.

In the Bookinfo sample application, configure a Sidecar so all services can reach other services running in the same namespace and control plane. This Sidecar configuration is required for using Red Hat OpenShift Service Mesh policy and telemetry features.

Procedure

1. Create a YAML file using the following example to specify that you want a sidecar configuration to apply to all workloads in a particular namespace. Otherwise, choose specific workloads using a workloadSelector.

   **Example sidecar.yaml**

   ```yaml
   apiVersion: networking.istio.io/v1alpha3
   kind: Sidecar
   metadata:
     name: default
     namespace: bookinfo
   spec:
     egress:
       - hosts:
         - "/*"
         - "istio-system/*"

   $ oc apply -f sidecar.yaml
   $ oc get sidecar
   ``

2. Run the following command to apply `sidecar.yaml`, where `sidecar.yaml` is the path to the file.

   ```
   $ oc apply -f sidecar.yaml
   ```

3. Run the following command to verify that the sidecar was created successfully.

   ```
   $ oc get sidecar
   ```

1.15. METRICS AND TRACES

You can view your application’s topology, health, and metrics in the Kiali console. If your service is experiencing problems, the Kiali console allows you to view 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 an application installed. If you do not have your own application installed, you can see how observability works in Red Hat OpenShift Service Mesh by installing the Bookinfo sample application.

1.15.1. Accessing metrics and tracing data from the CLI

Access the Jaeger, Prometheus, and Grafana consoles to view and manage your data.

Procedure
1. Switch to the control plane project. In this example, **istio-system** is the control plane project. Run the following command:

   ```sh
   $ oc project istio-system
   ```

2. Get the routes to Red Hat OpenShift Service Mesh components. Run the following command:

   ```sh
   $ oc get routes
   ```

   This command returns URLs for the web consoles of Kiali, Jaeger, Prometheus, and Grafana, and any other routes in your service mesh.

3. Copy the URL for the component you want from the **HOST/PORT** column into a browser to open the console.

### 1.15.2. Viewing service mesh data

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.

To access the Kiali console you must have Red Hat OpenShift Service Mesh installed and projects configured for the service mesh.

**Procedure**

1. Use the perspective switcher to switch to the Administrator perspective.

2. Click **Home > Projects**.

3. Click the name of your project. For example click **bookinfo**.

4. In the Launcher section, click **Kiali**.

5. Log in to the Kiali console with the same user name and password that you use to access the OpenShift Container Platform console.

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

### 1.15.2.1. Working with data in the Kiali console

From the **Graph** menu in the Kiali console, you can use the following graphs and viewing tools to gain deeper insights about data that travels through your service mesh. These tools can help you identify problems with services or workloads.

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 application. All versions of an application 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 application and version labels. If your application does not use version labels, use this the graph.

The Service graph shows a node for each service in your mesh but excludes all applications 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.

1.15.2.1.1. 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.

Prerequisite

- Install the Bookinfo sample application.

Procedure

1. Send traffic to the mesh by entering the following command several times.

   ```bash
   $ curl "http://$GATEWAY_URL/productpage"
   
   This command simulates a user visiting the productpage microservice of the application.
   ```

2. In the main navigation, click Graph to view a namespace graph.

3. Select bookinfo from the Namespace menu.

1.15.3. Distributed tracing

Distributed Tracing is the process of tracking the performance of individual services in an application by tracing the path of the service calls in the application. Each time a user takes action in an application, a request is executed that might require many services to interact to produce a response. The path of this request is called a distributed transaction.

Red Hat OpenShift Service Mesh uses Jaeger to allow developers to view call flows in a microservice application.

1.15.3.1. Generating example traces and analyzing trace data

Jaeger is an open source distributed tracing system. With Jaeger, you can perform a trace that follows the path of a request through various microservices which make up an application. Jaeger is installed by default as part of the Service Mesh.

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

Prerequisites:

- OpenShift Container Platform 4.1 or higher installed.
Red Hat OpenShift Service Mesh 2.0.6 installed.

- Jaeger enabled during the installation.
- Bookinfo example application installed.

**Procedure**

1. After installing the Bookinfo sample application, send traffic to the mesh. Enter the following command several times.

   ```bash
   $ curl "http://$GATEWAY_URL/productpage"
   
   This command simulates a user visiting the productpage microservice of the application.
   
2. In the OpenShift Container Platform console, navigate to Networking → Routes and search for the Jaeger route, which is the URL listed under Location.

   - Alternatively, use the CLI to query for details of the route. In this example, `istio-system` is the control plane namespace:

   ```bash
   $ export JAEGGER_URL=$(oc get route -n istio-system jaeger -o jsonpath='{.spec.host}')
   
   a. Enter the following command to reveal the URL for the Jaeger console. Paste the result in a browser and navigate to that URL.

   ```bash
   echo $JAEGGER_URL
   ```

3. Log in using the same user name and password as you use to access the OpenShift Container Platform console.

4. In the left pane of the Jaeger dashboard, from the Service menu, select productpage.bookinfo and click the Find Traces button at the bottom of the pane. A list of traces is displayed.

5. Click one of the traces in the list to open a detailed view of that trace. If you click the first one in the list, which is the most recent trace, you see the details that correspond to the latest refresh of the /productpage.

**1.15.3.2. Adjusting the sampling rate**

The distributed tracing sampling rate is set to sample 100% of traces in your service mesh by default. A high sampling rate consumes cluster resources and performance but is useful when debugging issues. Before you deploy Red Hat OpenShift Service Mesh in production, set the value to a smaller proportion of traces.

A trace is an execution path between services in the service mesh. A trace is comprised of one or more spans. A span is a logical unit of work that has a name, start time, and duration.

The sampling rate determines how often a trace is generated. Configure sampling as a scaled integer representing 0.01% increments.

In a basic installation, `spec.tracing.sampling` is set to 10000, which samples 100% of traces. For example:

- Setting the value to 10 samples 0.1% of traces.
Setting the value to 500 samples 5% of traces.

Setting the value to 10000 is useful for debugging, but can affect performance. For production, set spec.tracing.sampling to 100.

Procedure

1. In the OpenShift Container Platform web console, click Operators → Installed Operators.

2. From the Project menu, select the project where you installed the control plane, for example istio-system.

3. Click the Red Hat OpenShift Service Mesh Operator. In the Istio Service Mesh Control Plane column, click the name of your ServiceMeshControlPlane resource, for example basic.

4. To adjust the sampling rate, set a different value for spec.tracing.sampling.
   a. Click the YAML tab.
   b. Set the value for spec.tracing.sampling in your ServiceMeshControlPlane resource. In the following example, set it to 100.

   Jaeger sampling example

   ```yaml
   spec:
     tracing:
       sampling: 100
   ```
   c. Click Save.

5. Click Reload to verify the ServiceMeshControlPlane resource was configured correctly.

1.15.3.3. Connecting standalone Jaeger

If you already use standalone Jaeger for distributed tracing in OpenShift Container Platform, configure your ServiceMeshControlPlane resource to use that standalone Jaeger instance rather than the one installed with Red Hat OpenShift Service Mesh.

Prerequisites

- Configure and deploy a standalone Jaeger instance. For more information, see the Jaeger documentation.

Procedure

1. In the OpenShift Container Platform web console, click Operators → Installed Operators.

2. From the Project menu, select the project where you installed the control plane, for example istio-system.

3. Click the Red Hat OpenShift Service Mesh Operator. In the Istio Service Mesh Control Plane column, click the name of your ServiceMeshControlPlane resource, for example basic.

4. Add the name of your standalone Jaeger instance to the ServiceMeshControlPlane.
   a. Click the YAML tab.
b. Add the name of your standalone Jaeger instance to `spec.addons.jaeger.name` in your `ServiceMeshControlPlane` resource. In the following example, `simple-prod` is the name of your standalone Jaeger instance.

**Standalone Jaeger example**

```yaml
spec:
  addons:
    jaeger:
      name: simple-prod
```

c. Click **Save**.

5. Click **Reload** to verify the `ServiceMeshControlPlane` resource was configured correctly.

For more information about configuring Jaeger, see the [Jaeger documentation](#).

### 1.15.4. Accessing Grafana

Grafana is an analytics tool that you can use to view, query, and analyze your service mesh metrics. To access Grafana, do the following:

**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. Click **Routes**.

4. Click the link in the **Location** column for the **Grafana** row.

5. Log into the Grafana console with your OpenShift Container Platform credentials.

### 1.15.5. Accessing Prometheus

Prometheus is a monitoring and alerting tool that you can use to collect multi-dimensional data about your microservices.

**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. Click **Routes**.

4. Click the link in the **Location** column for the **Prometheus** row.

5. Log into the Prometheus console with your OpenShift Container Platform credentials.

### 1.16. PERFORMANCE AND SCALABILITY

The default `ServiceMeshControlPlane` settings are not intended for production use; they are designed to install successfully on a default OpenShift Container Platform installation, which is a resource-limited
environment. After you have verified a successful SMCP installation, you should modify the settings defined within the SMCP to suit your environment.

1.16.1. Setting limits on compute resources

By default, spec.proxy has the settings cpu: 10m and memory: 128M. If you are using Pilot, spec.runtime.components.pilot has the same default values.

The settings in the following example are based on 1,000 services and 1,000 requests per second. You can change the values for cpu and memory in the ServiceMeshControlPlane.

Procedure

1. In the OpenShift Container Platform web console, click Operators → Installed Operators.

2. From the Project menu, select the project where you installed the ServiceMeshControlPlane resource, for example istio-system.

3. Click the Red Hat OpenShift Service Mesh Operator. In the Istio Service Mesh Control Plane column, click the name of your ServiceMeshControlPlane, for example basic.

4. Add the name of your standalone Jaeger instance to the ServiceMeshControlPlane.
   a. Click the YAML tab.
   b. Set the values for spec.proxy.runtime.container.resources.requests.cpu and spec.proxy.runtime.container.resources.requests.memory in your ServiceMeshControlPlane resource.

Example version 2.0 ServiceMeshControlPlane

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.0
  proxy:
    runtime:
      container:
        resources:
          requests:
            cpu: 600m
            memory: 50Mi
          limits: {}

    runtime:
      components:
        pilot:
          container:
            resources:
              requests:
                cpu: 1000m
                memory: 1.6Gi
              limits: {}
```
c. Click Save.

5. Click Reload to verify the ServiceMeshControlPlane resource was configured correctly.

1.16.2. Load test results

The upstream Istio community load tests mesh consists of 1000 services and 2000 sidecars with 70,000 mesh-wide requests per second. Running the tests using Istio 1.6.8, generated the following results:

- The Envoy proxy uses 0.5 vCPU and 50 MB memory per 1000 requests per second going through the proxy.
- Istiod uses 1 vCPU and 1.5 GB of memory.
- The Envoy proxy adds 3.12 ms to the 90th percentile latency.
- The legacy istio-telemetry service (disabled by default in Service Mesh 2.0) uses 0.6 vCPU per 1000 mesh-wide requests per second for deployments that use Mixer. The data plane components, the Envoy proxies, handle data flowing through the system. The control plane component, Istiod, configures the data plane. The data plane and control plane have distinct performance concerns.

1.16.2.1. Control plane performance

Istiod configures sidecar proxies based on user authored configuration files and the current state of the system. In a Kubernetes environment, Custom Resource Definitions (CRDs) and deployments constitute the configuration and state of the system. The Istio configuration objects like gateways and virtual services, provide the user-authored configuration. To produce the configuration for the proxies, Istiod processes the combined configuration and system state from the Kubernetes environment and the user-authored configuration.

The control plane supports thousands of services, spread across thousands of pods with a similar number of user authored virtual services and other configuration objects. Istiod’s CPU and memory requirements scale with the number of configurations and possible system states. The CPU consumption scales with the following factors:

- The rate of deployment changes.
- The rate of configuration changes.
- The number of proxies connecting to Istiod.

However this part is inherently horizontally scalable.

1.16.2.2. Data plane performance

Data plane performance depends on many factors, for example:

- Number of client connections
- Target request rate
- Request size and response size
- Number of proxy worker threads
• Protocol
• CPU cores
• Number and types of proxy filters, specifically telemetry v2 related filters.

The latency, throughput, and the proxies’ CPU and memory consumption are measured as a function of these factors.

1.16.2.2.1. CPU and memory consumption

Since the sidecar proxy performs additional work on the data path, it consumes CPU and memory. As of Istio 1.1, a proxy consumes about 0.6 vCPU per 1000 requests per second.

The memory consumption of the proxy depends on the total configuration state the proxy holds. A large number of listeners, clusters, and routes can increase memory usage.

Since the proxy normally doesn’t buffer the data passing through, request rate doesn’t affect the memory consumption.

1.16.2.2.2. Additional latency

Since Istio injects a sidecar proxy on the data path, latency is an important consideration. Istio adds an authentication filter, a telemetry filter, and a metadata exchange filter to the proxy. Every additional filter adds to the path length inside the proxy and affects latency.

The Envoy proxy collects raw telemetry data after a response is sent to the client. The time spent collecting raw telemetry for a request does not contribute to the total time taken to complete that request. However, since the worker is busy handling the request, the worker won’t start handling the next request immediately. This process adds to the queue wait time of the next request and affects average and tail latencies. The actual tail latency depends on the traffic pattern.

Inside the mesh, a request traverses the client-side proxy and then the server-side proxy. In the default configuration of Istio 1.6.8 (that is, Istio with telemetry v2), the two proxies add about 3.12 ms and 3.13 ms to the 90th and 99th percentile latency, respectively, over the baseline data plane latency.

1.17. EXTENSIONS

You can use WebAssembly extensions to add new features directly into the Red Hat OpenShift Service Mesh proxies, allowing you to move even more common functionality out of your applications, and implement them in a single language that compiles to WebAssembly bytecode.

1.17.1. WebAssembly extensions

WebAssembly modules can be run on many platforms, including proxies, and has broad language support, fast execution and a sandboxed-by-default security model.

Extension Capabilities

Red Hat OpenShift Service Mesh extensions are Envoy HTTP Filters, giving them a wide range of capabilities:

• Manipulating the body and headers of requests and responses
• Out-of-band HTTP requests to services not in the request path, such as authentication or policy checking
Side-channel data storage and queues for filters to communicate with each other

There are two parts to writing a Red Hat OpenShift Service Mesh extension: you’ll have to write your extension using an SDK that exposes the proxy-wasm API and compile it to a WebAssembly module, and then package it into a container.

**Supported languages**

You can use any language that compiles to WebAssembly bytecode to write a Red Hat OpenShift Service Mesh extension, but the following languages have existing SDKs that expose the proxy-wasm API so that it can be consumed directly.

**Table 1.4. Supported languages**

<table>
<thead>
<tr>
<th>Language</th>
<th>Maintainer</th>
<th>Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssemblyScript</td>
<td>solo.io</td>
<td>solo-io/proxy-runtime</td>
</tr>
<tr>
<td>C++</td>
<td>proxy-wasm team (Istio Community)</td>
<td>proxy-wasm/proxy-wasm-cpp-sdk</td>
</tr>
<tr>
<td>Go</td>
<td>tetratelabs.io</td>
<td>tetratelabs/proxy-wasm-go-sdk</td>
</tr>
<tr>
<td>Rust</td>
<td>proxy-wasm team (Istio Community)</td>
<td>proxy-wasm/proxy-wasm-rust-sdk</td>
</tr>
</tbody>
</table>

**1.17.1.1. Container Format**

You must have a .wasm file containing the bytecode of your WebAssembly module, and a manifest.yaml file in the root of the container filesystem to make your container image a valid extension image.

**manifest.yaml**

```
schemaVersion: 1
name: <your-extension>
description: <description>
version: 1.0.0
phase: PreAuthZ
priority: 100
module: extension.wasm
```

**Table 1.5. Field Reference for manifest.yaml**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaVersion</td>
<td>Used for versioning of the manifest schema. Currently the only possible value is 1.</td>
</tr>
<tr>
<td>name</td>
<td>The name of your extension. This field is just metadata and currently unused.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>The description of your extension. This field is just metadata and currently unused.</td>
</tr>
<tr>
<td>version</td>
<td>The version of your extension. This field is just metadata and currently unused.</td>
</tr>
<tr>
<td>phase</td>
<td>The default execution phase of your extension. This is a required field.</td>
</tr>
<tr>
<td>priority</td>
<td>The default priority of your extension. This is a required field.</td>
</tr>
<tr>
<td>module</td>
<td>The relative path from the container filesystem's root to your WebAssembly module. This is a required field.</td>
</tr>
</tbody>
</table>

### 1.17.1.2. Example Rust extension

For a complete example that was built using the Rust SDK, take a look at the `header-append-filter`. The filter appends a header, called `custom-header`, to all responses, with the value depending on its configuration.

### 1.17.1.3. Enabling WebAssembly extension support

Support for WebAssembly extensions to Red Hat OpenShift Service Mesh is currently in Technology Preview, so it must be explicitly enabled for your `ServiceMeshControlPlane`.

#### Procedure

1. In the OpenShift Container Platform web console, click **Operators** → **Installed Operators**.
2. From the **Project** menu, select the project where you installed the control plane, for example `istio-system`.
3. Click the Red Hat OpenShift Service Mesh Operator. In the **Istio Service Mesh Control Plane** column, click the name of your `ServiceMeshControlPlane` resource, for example `basic`.
4. Click the YAML tab.
5. Set `spec.techPreview.wasmExtensions.enabled` in your `ServiceMeshControlPlane` resource to `true`. For example:

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: openid-connect
  namespace: istio-system
spec:
  techPreview:
    wasmExtensions:
      enabled: true
```
6. Click **Save**.

7. Click **Reload** to verify the **ServiceMeshControlPlane** resource was configured correctly.

### 1.17.1.4. Deploying extensions

Red Hat OpenShift Service Mesh extensions can be enabled using the **ServiceMeshExtension** resource.

**Procedure**

1. Create the following example resource:

   ```yaml
   apiVersion: maistra.io/v1alpha1
   kind: ServiceMeshExtension
   metadata:
     name: header-append
     namespace: istio-system
   spec:
     workloadSelector:
       labels:
         app: httpbin
       config: test
       image: quay.io/maistra-dev/header-append-filter:2.0
       phase: PostAuthZ
       priority: 100
   ```

2. Apply the **extension.yaml** file with the following command:

   ```bash
   $ oc apply -f extension.yaml
   ```

**Table 1.6. ServiceMeshExtension Field Reference**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata.namespace</td>
<td>The <strong>metadata.namespace</strong> of a <strong>ServiceMeshExtension</strong> source has a special semantic: if it equals the Control Plane Namespace, the extension will be applied to all workloads in the Service Mesh that match its <strong>workloadSelector</strong>. When deployed to any other Mesh Namespace, it will only be applied to workloads in that same Namespace.</td>
</tr>
<tr>
<td>spec.workloadSelector</td>
<td>The <strong>spec.workloadSelector</strong> field has the same semantic as the <strong>spec.selector</strong> field of the <strong>Istio Gateway resource</strong>. It will match a workload based on its Pod labels. If no <strong>workloadSelector</strong> is specified, the extension will be applied to all workloads in the namespace.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spec.config</td>
<td>This is a pass-through string field that is handed over to the extension. Syntax and semantics are dependent on the extension that you are deploying.</td>
</tr>
<tr>
<td>spec.image</td>
<td>A container image URI pointing to the image that holds the extension.</td>
</tr>
<tr>
<td>spec.phase</td>
<td>This field defaults to the value set in the <code>manifest.yaml</code> of the extension, but can be overwritten by the user. The phase determines where in the filter chain the extension is injected, in relation to existing Istio functionality like Authentication, Authorization and metrics generation. Valid values are: PreAuthN, PostAuthN, PreAuthZ, PostAuthZ, PreStats, PostStats. This field defaults to the value set in the <code>manifest.yaml</code> of the extension, but can be overwritten by the user.</td>
</tr>
<tr>
<td>spec.priority</td>
<td>If multiple extensions with the same <code>spec.phase</code> are applied to the same workload instance, the <code>spec.priority</code> determines the ordering of execution. Extensions with higher priority will be executed first. This allows for inter-dependent extensions. This field defaults to the value set in the <code>manifest.yaml</code> of the extension, but can be overwritten by the user.</td>
</tr>
</tbody>
</table>

### 1.18. CONFIGURING SERVICE MESH FOR PRODUCTION

When you are ready to move from a basic installation to production, you must configure your control plane, tracing, and security certificates to meet production requirements.

**Prerequisites**

- Install and configure Red Hat OpenShift Service Mesh.
- Test your configuration in a staging environment.

**1.18.1. Configuring your ServiceMeshControlPlane resource for production**

If you have installed a basic `ServiceMeshControlPlane` resource to test Service Mesh, you must configure it to production specification before you use Red Hat OpenShift Service Mesh in production.

You cannot change the `metadata.name` field of an existing `ServiceMeshControlPlane` resource. For production deployments, you must customize the default template.

**Procedure**

1. Configure Jaeger for production.
a. Edit the ServiceMeshControlPlane resource to use the production deployment strategy, by setting spec.addons.jaeger.install.storage.type to Elasticsearch and specify additional configuration options under install. You can create and configure your Jaeger instance and set spec.addons.jaeger.name to the name of the Jaeger instance, for example, jaeger-production.

Default Jaeger parameters including Elasticsearch

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.0
tracing:
  sampling: 100
type: Jaeger
addons:
  jaeger:
    name: jaeger-production
    install:
      storage:
        type: Elasticsearch
      ingress:
        enabled: true
runtime:
  components:
    tracing.jaeger.elasticsearch: # only supports resources and image name
      container:
        resources: {}
```

b. Configure the sampling rate for production. For more information, see the Performance and scalability section.

2. Ensure your security certificates are production ready by installing security certificates from an external certificate authority. For more information, see the Security section.

3. Verify the results. Enter the following command to verify that the ServiceMeshControlPlane resource updated properly. In this example, basic is the name of the ServiceMeshControlPlane resource.

   ```bash
   $ oc get smcp basic -o yaml
   ```

1.19. ADDITIONAL RESOURCES

For more information about tuning Service Mesh for performance, see Performance and Scalability.

1.20. 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.
1.20.1. 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 2.x
- A working 3scale account (SaaS or 3scale 2.9 On-Premises)
- Enabling backend cache requires 3scale 2.9 or greater
- 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.
- Mixer policy and telemetry must be enabled if you are using a mixer plug-in.
  - You will need to properly configure the Service Mesh Control Plane (SMCP) when upgrading.

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 account credentials. You can optionally add a service_id to a handler, but this is kept for backwards compatibility only, since it would render the handler only useful for one service in your 3scale account. If you add service_id to a handler, enabling 3scale for other services requires you to create more handlers with different service_ids.

Use a single handler per 3scale account by following the steps below:

Procedure

1. Create a handler for your 3scale account and specify your account credentials. Omit any service identifier.

```yaml
apiVersion: "config.istio.io/v1alpha2"
kind: handler
metadata:
  name: threescale
spec:
  adapter: threescale
```
params:
  - 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.

2. Edit or patch the Deployment resource of any services belonging to your 3scale account as follows:
   a. Add the "service-mesh.3scale.net/service-id" label with a value corresponding to a valid service_id.
   b. Add the "service-mesh.3scale.net/credentials" label with its value being the name of the handler resource from step 1.

3. Do step 2 to link it to your 3scale account credentials and to its service identifier, whenever you intend to add more services.

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

**Rule configuration example**

```yaml
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
```

1.20.1.1. Generating 3scale custom resources

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

**Table 1.7. Usage**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Produces help output for available options</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>--name</td>
<td>Unique name for this URL, token pair</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Required</td>
<td>Default value</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>---------------</td>
</tr>
<tr>
<td>-n, --namespace</td>
<td>Namespace to generate templates</td>
<td>No</td>
<td>istio-system</td>
</tr>
<tr>
<td>-t, --token</td>
<td>3scale access token</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>-u, --url</td>
<td>3scale Admin Portal URL</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>--backend-url</td>
<td>3scale backend URL. If set, it overrides the value that is read from system configuration</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>-s, --service</td>
<td>3scale API/Service ID</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>--auth</td>
<td>3scale authentication pattern to specify (1=API Key, 2=App Id/App Key, 3=OIDC)</td>
<td>No</td>
<td>Hybrid</td>
</tr>
<tr>
<td>-o, --output</td>
<td>File to save produced manifests to</td>
<td>No</td>
<td>Standard output</td>
</tr>
<tr>
<td>--version</td>
<td>Outputs the CLI version and exits immediately</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

1.20.1.1. Generate templates from URL examples

**NOTE**

- Run the following commands via `oc exec` from the 3scale adapter container image in Generating manifests from a deployed adapter.
- Use the `3scale-config-gen` command to help avoid YAML syntax and indentation errors.
- You can omit the `--service` if you use the annotations.
- This command must be invoked from within the container image via `oc exec`.

**Procedure**

- Use the `3scale-config-gen` command to autogenerate templates files allowing the token, URL pair to be shared by multiple services as a single handler:

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

- The following example generates the templates with the service ID embedded in the handler:
Additional resources

- Tokens.

1.20.1.2. Generating manifests from a deployed adapter

**NOTE**

- **NAME** is an identifier you use to identify with the service you are managing with 3scale.

- The **CREDENTIALS_NAME** reference is an identifier that corresponds to the **match** section in the rule configuration. This is automatically set to the **NAME** identifier if you are using the CLI tool.

- Its value does not need to be anything specific: the label value should just match the contents of the rule. See Routing service traffic through the adapter for more information.

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})" -o jsonpatch -n $({NS})"
```
1.20.1.3. 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.

1.20.2. 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

2. Click Settings.

3. Select the Istio option under Deployment.
   - The API Key (user_key) option under Authentication is selected by default.

4. Click Update Product to save your selection.

5. Click Configuration.

6. Click Update Configuration.

1.20.3. 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.

### 1.20.4. 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.

#### 1.20.4.1. 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"]`.

#### 1.20.4.1.1. 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**

```yaml
apiVersion: "config.istio.io/v1alpha2"
kind: instance
metadata:
  name: threescale-authorization
  namespace: istio-system
spec:
```

OpenShift Container Platform 4.8 Service Mesh
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"]`.

### 1.20.4.1.2. 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

```yaml
apiVersion: "config.istio.io/v1alpha2"
kind: instance
metadata:
  name: threescale-authorization
  namespace: istio-system
spec:
  template:
    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"]`.

### 1.20.4.1.3. 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

```yaml
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 a Request authorization 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 RequestAuthentication defined below, replace issuer, jwksUri, and selector as appropriate.

OpenID Policy example

apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: bookinfo
spec:
  selector:
    matchLabels:
      app: productpage
  jwtRules:
  - 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

1.20.4.1.4. 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:
1.20.5. 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.

**NOTE**

There are incompatible changes in the 3scale Istio Adapter metrics since the previous releases in Service Mesh 1.x.

In Prometheus, metrics have been renamed with one addition for the backend cache, so that the following metrics exist as of Service Mesh 2.0:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threescale_latency</td>
<td>Histogram</td>
<td>Request latency between adapter and 3scale.</td>
</tr>
<tr>
<td>threescale_http_total</td>
<td>Counter</td>
<td>HTTP Status response codes for requests to 3scale backend.</td>
</tr>
<tr>
<td>threescale_system_cache_hits</td>
<td>Counter</td>
<td>Total number of requests to the 3scale system fetched from the configuration cache.</td>
</tr>
<tr>
<td>threescale_backend_cache_hits</td>
<td>Counter</td>
<td>Total number of requests to 3scale backend fetched from the backend cache.</td>
</tr>
</tbody>
</table>

1.20.6. 3scale backend cache

```yaml
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"] | ""
```
The 3scale backend cache provides an authorization and reporting cache for clients of the 3scale Service Management API. This cache is embedded in the adapter to enable lower latencies in responses in certain situations assuming the administrator is willing to accept the trade-offs.

NOTE
3scale backend cache is disabled by default. 3scale backend cache functionality trades inaccuracy in rate limiting and potential loss of hits since the last flush was performed for low latency and higher consumption of resources in the processor and memory.

1.20.6.1. Advantages of enabling backend cache

The following are advantages to enabling the backend cache:

- Enable the backend cache when you find latencies are high while accessing services managed by the 3scale Istio Adapter.

- Enabling the backend cache will stop the adapter from continually checking with the 3scale API manager for request authorizations, which will lower the latency.
  - This creates an in-memory cache of 3scale authorizations for the 3scale Istio Adapter to store and reuse before attempting to contact the 3scale API manager for authorizations. Authorizations will then take much less time to be granted or denied.

- Backend caching is useful in cases when you are hosting the 3scale API manager in another geographical location from the service mesh running the 3scale Istio Adapter.
  - This is generally the case with the 3scale Hosted (SaaS) platform, but also if a user hosts their 3scale API manager in another cluster located in a different geographical location, in a different availability zone, or in any case where the network overhead to reach the 3scale API manager is noticeable.

1.20.6.2. Trade-offs for having lower latencies

The following are trade-offs for having lower latencies:

- Each 3scale adapter’s authorization state updates every time a flush happens.
  - This means two or more instances of the adapter will introduce more inaccuracy between flushing periods.
  - There is a greater chance of too many requests being granted that exceed limits and introduce erratic behavior, which leads to some requests going through and some not, depending on which adapter processes each request.

- An adapter cache that cannot flush its data and update its authorization information risks shutting down or crashing without reporting its information to the API manager.

- A fail open or fail closed policy will be applied when an adapter cache cannot determine whether a request must be granted or denied, possibly due to network connectivity issues in contacting the API manager.

- When cache misses occur, typically right after booting the adapter or after a long period of no connectivity, latencies will grow in order to query the API manager.

- An adapter cache must do much more work on computing authorizations than it would without an enabled cache, which will tax processor resources.
• Memory requirements will grow proportionally to the combination of the amount of limits, applications, and services managed by the cache.

1.20.6.3. Backend cache configuration settings

The following points explain the backend cache configuration settings:

• Find the settings to configure the backend cache in the 3scale configuration options.

• The last 3 settings control enabling of backend cache:

  ○ PARAM_USE_CACHE_BACKEND - set to true to enable backend cache.

  ○ PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS - sets time in seconds between consecutive attempts to flush cache data to the API manager.

  ○ PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED - set whether or not to allow/open or deny/close requests to the services when there is not enough cached data and the 3scale API manager cannot be reached.

1.20.7. 3scale Istio Adapter APIcast emulation

The 3scale Istio Adapter performs as APIcast would when the following conditions occur:

• When a request cannot match any mapping rule defined, the returned HTTP code is 404 Not Found. This was previously 403 Forbidden.

• When a request is denied because it goes over limits, the returned HTTP code is 429 Too Many Requests. This was previously 403 Forbidden.

• When generating default templates via the CLI, it will use underscores rather than dashes for the headers, for example: user_key rather than user-key.

1.20.8. 3scale Istio adapter verification

You might want to check whether the 3scale Istio adapter is working as expected. If your adapter is not working, use the following steps to help troubleshoot the problem.

Procedure

1. Ensure the 3scale-adapter pod is running in the control plane namespace:

   $ oc get pods -n <istio-system>

2. Check that the 3scale-adapter pod has printed out information about itself booting up, such as its version:

   $ oc logs <istio-system>

3. When performing requests to the services protected by the 3scale adapter integration, always try requests that lack the right credentials and ensure they fail. Check the 3scale adapter logs to gather additional information.

Additional resources
• Inspecting pod and container logs.

1.20.9. 3scale Istio adapter troubleshooting checklist

As the administrator installing the 3scale Istio adapter, there are a number of scenarios that might be causing your integration to not function properly. Use the following list to troubleshoot your installation:

• Incorrect YAML indentation.

• Missing YAML sections.

• Forgot to apply the changes in the YAML to the cluster.

• Forgot to label the service workloads with the `service-mesh.3scale.net/credentials` key.

• Forgot to label the service workloads with `service-mesh.3scale.net/service-id` when using handlers that do not contain a `service_id` so they are reusable per account.

• The `Rule` custom resource points to the wrong handler or instance custom resources, or the references lack the corresponding namespace suffix.

• The `Rule` custom resource `match` section cannot possibly match the service you are configuring, or it points to a destination workload that is not currently running or does not exist.

• Wrong access token or URL for the 3scale Admin Portal in the handler.

• The `Instance` custom resource’s `params/subject/properties` section fails to list the right parameters for `app_id`, `app_key`, or `client_id`, either because they specify the wrong location such as the query parameters, headers, and authorization claims, or the parameter names do not match the requests used for testing.

• Failing to use the configuration generator without realizing that it actually lives in the adapter container image and needs `oc exec` to invoke it.

1.21. UNINSTALLING RED HAT OPENSSHIFT SERVICE MESH

To uninstall Red Hat OpenShift Service Mesh from an existing OpenShift Container Platform instance and remove its resources, you must delete the control plane, delete the Operators, and run commands to manually remove some resources.

1.21.1. Removing the Red Hat OpenShift Service Mesh control plane

To uninstall Service Mesh from an existing OpenShift Container Platform instance, you must first delete the control plane and the Operators. Then, you must run commands to manually remove residual resources.

1.21.1.1. Removing the control plane with the web console

You can remove the Red Hat OpenShift Service Mesh control plane by using the web console.

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 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.

1.21.1.2. Removing the control plane from the CLI

You can remove the Red Hat OpenShift Service Mesh control plane by using the CLI.

Procedure

1. Log in to the OpenShift Container Platform CLI.

2. Run this command to retrieve the name of the installed ServiceMeshControlPlane:

   ```
   $ oc get smcp -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 smcp -n istio-system <name_of_custom_resource>
   ```

1.21.2. Removing the installed Operators

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

1.21.2.1. Removing the Operators

Follow this procedure to remove the Operators that make up Red Hat OpenShift Service Mesh. Repeat the steps for each of the following Operators.

- Red Hat OpenShift Service Mesh
- Kiali
- Jaeger
- OpenShift Elasticsearch

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 each Operator. Then, click the Operator name.
3. On the Operator Details page, select Uninstall Operator from the Actions menu. Follow the prompts to uninstall each Operator.

### 1.21.3. Clean up Operator resources

You can manually remove resources left behind after removing the Red Hat OpenShift Service Mesh Operator using the OpenShift console.

**Prerequisites**

- An account with cluster administration access. If you use Red Hat OpenShift Dedicated, you must have an account with the `dedicated-admin` role.
- 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. If you intend to keep using Jaeger as a stand alone service without service mesh, do not delete the Jaeger resources.

```
NOTE

The Openshift Elasticsearch operator is installed in `openshift-operators-redhat` by default. The other Operators are installed in the `openshift-operators` namespace by default. If you installed the Operators in another namespace, replace `openshift-operators` with the name of the project where the Red Hat OpenShift Service Mesh Operator was installed.

$ oc delete validatingwebhookconfiguration/openshift-operators.servicemesh-resources.maistra.io
$ oc delete mutatingwebhookconfigurations/openshift-operators.servicemesh-resources.maistra.io
$ oc delete svc maistra-admission-controller -n openshift-operators
$ oc delete -n openshift-operators 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
```
### 1.22. CONFIGURATION REFERENCE

#### 1.22.1. Control plane parameters

The following table lists the top-level parameters for the `ServiceMeshControlPlane` resource.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>apiVersion</strong></td>
<td>APIVersion defines the versioned schema of this representation of an object. Servers convert recognized schemas to the latest internal value, and may reject unrecognized values. The value for the <code>ServiceMeshControlPlane</code> version 2.0 is <code>maistra.io/v2</code>.</td>
<td>The value for <code>ServiceMeshControlPlane</code> version 2.0 is <code>maistra.io/v2</code>.</td>
</tr>
<tr>
<td><strong>kind</strong></td>
<td>Kind is a string value that represents the REST resource this object represents.</td>
<td><code>ServiceMeshControlPlane</code> is the only valid value for a <code>ServiceMeshControlPlane</code>.</td>
</tr>
<tr>
<td><strong>metadata</strong></td>
<td>Metadata about this <code>ServiceMeshControlPlane</code> instance. You can provide a name for your control plane installation to keep track of your work, for example, <code>basic</code>.</td>
<td>string</td>
</tr>
<tr>
<td><strong>spec</strong></td>
<td>The specification of the desired state of this <code>ServiceMeshControlPlane</code>. This includes the configuration options for all components that comprise the control plane.</td>
<td>For more information, see Table 2.</td>
</tr>
</tbody>
</table>

```bash
$ 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
$ oc delete crds jaegers.jaegertracing.io
$ oc delete secret -n openshift-operators maistra-operator-serving-cert
$ oc delete cm -n openshift-operators maistra-operator-cabundle
```
The current status of this `ServiceMeshControlPlane` and the components that comprise the control plane. For more information, see Table 3.

The following table lists the specifications for the `ServiceMeshControlPlane` resource. Changing these parameters configures Red Hat OpenShift Service Mesh components.

### Table 1.10. `ServiceMeshControlPlane` resource spec

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Configurable parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>addons</strong></td>
<td>The <code>addons</code> parameter configures additional features beyond core control plane components, such as visualization, or metric storage.</td>
<td><code>3scale, grafana, jaeger, kiali, and prometheus</code>.</td>
</tr>
<tr>
<td><strong>cluster</strong></td>
<td>The <code>cluster</code> parameter sets the general configuration of the cluster (cluster name, network name, multi-cluster, mesh expansion, etc.)</td>
<td><code>meshExpansion, multiCluster, name, and network</code></td>
</tr>
<tr>
<td><strong>gateways</strong></td>
<td>You use the <code>gateways</code> parameter to configure ingress and egress gateways for the mesh.</td>
<td><code>enabled, additionalEgress, additionalIngress, egress, ingress, and openshiftRoute</code></td>
</tr>
<tr>
<td><strong>general</strong></td>
<td>The <code>general</code> parameter represents general control plane configuration that does not fit anywhere else.</td>
<td><code>logging and validationMessages</code></td>
</tr>
<tr>
<td><strong>policy</strong></td>
<td>You use the <code>policy</code> parameter to configure policy checking for the control plane. Policy checking can be enabled by setting <code>spec.policy.enabled</code> to true.</td>
<td><code>mixer remote, or type. type can be set to Istiod, Mixer or None.</code></td>
</tr>
<tr>
<td><strong>profiles</strong></td>
<td>You select the <code>ServiceMeshControlPlane</code> profile to use for default values using the <code>profiles</code> parameter.</td>
<td><code>default</code></td>
</tr>
<tr>
<td><strong>proxy</strong></td>
<td>You use the <code>proxy</code> parameter to configure the default behavior for sidecars.</td>
<td><code>accessLogging, adminPort, concurrency, and envoyMetricsService</code></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Configurable parameters</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>runtime</td>
<td>You use the runtime parameter to configure the control plane components.</td>
<td>components, and defaults</td>
</tr>
<tr>
<td>security</td>
<td>The security parameter allows you to configure aspects of security for the control plane.</td>
<td>certificateAuthority, controlPlane, identity, dataPlane and trust</td>
</tr>
<tr>
<td>techPreview</td>
<td>The techPreview parameter enables early access to features that are in technology preview.</td>
<td>N/A</td>
</tr>
<tr>
<td>telemetry</td>
<td>If <code>spec.mixer.telemetry.enabled</code> is set to <code>true</code>, telemetry is enabled.</td>
<td>mixer, remote, and type.type can be set to Istiod, Mixer or None.</td>
</tr>
<tr>
<td>tracing</td>
<td>You use the tracing parameter to enables distributed tracing for the mesh.</td>
<td>sampling, type.type can be set to Jaeger or None.</td>
</tr>
<tr>
<td>version</td>
<td>You use the version parameter to specify what Maistra version of the control plane to install. When creating a ServiceMeshControlPlane with an empty version, the admission webhook sets the version to the current version. New ServiceMeshControlPlanes with an empty version are set to v2.0. Existing ServiceMeshControlPlanes with an empty version keep their setting.</td>
<td>string</td>
</tr>
</tbody>
</table>

**ControlPlaneStatus** represents the current state of your service mesh.

**Table 1.11. ServiceMeshControlPlane resource ControlPlaneStatus**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
</table>


<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotations</td>
<td>The <strong>annotations</strong> parameter stores additional, usually redundant status information, such as the number of components deployed by the <a href="https://openpitrix.org/docs/">ServiceMeshControlPlane</a>. These statuses are used by the command line tool, <code>oc</code>, which does not yet allow counting objects in JSONPath expressions.</td>
<td>Not configurable</td>
</tr>
<tr>
<td>conditions</td>
<td>Represents the latest available observations of the object’s current state. <strong>Reconciled</strong> indicates whether the operator has finished reconciling the actual state of deployed components with the configuration in the <a href="https://openpitrix.org/docs/">ServiceMeshControlPlane</a>. <strong>Installed</strong> indicates whether the control plane has been installed. <strong>Ready</strong> indicates whether all control plane components are ready.</td>
<td>string</td>
</tr>
<tr>
<td>components</td>
<td>Shows the status of each deployed control plane component.</td>
<td>string</td>
</tr>
<tr>
<td>appliedSpec</td>
<td>The resulting specification of the configuration options after all profiles have been applied.</td>
<td>ControlPlaneSpec</td>
</tr>
<tr>
<td>appliedValues</td>
<td>The resulting values.yaml used to generate the charts.</td>
<td>ControlPlaneSpec</td>
</tr>
<tr>
<td>chartVersion</td>
<td>The version of the charts that were last processed for this resource.</td>
<td>string</td>
</tr>
<tr>
<td>observedGeneration</td>
<td>The generation observed by the controller during the most recent reconciliation. The information in the status pertains to this particular generation of the object. The <strong>status.conditions</strong> are not up-to-date if the <strong>status.observedGeneration</strong> field doesn’t match <strong>metadata.generation</strong>.</td>
<td>integer</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>operatorVersion</td>
<td>The version of the operator that last processed this resource.</td>
<td>string</td>
</tr>
<tr>
<td>readiness</td>
<td>The readiness status of components &amp; owned resources.</td>
<td>string</td>
</tr>
</tbody>
</table>

This example `ServiceMeshControlPlane` definition contains all of the supported parameters.

**Example ServiceMeshControlPlane resource**

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  proxy:
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 500m
        memory: 128Mi
    tracing:
      type: Jaeger
gateways:
  ingress: # istio-ingressgateway
    service:
      type: ClusterIP
      ports:
      - name: status-port
        port: 15020
      - name: http2
        port: 80
        targetPort: 8080
      - name: https
        port: 443
        targetPort: 8443
    meshExpansionPorts: []
  egress: # istio-egressgateway
    service:
      type: ClusterIP
      ports:
      - name: status-port
        port: 15020
      - name: http2
        port: 80
        targetPort: 8080
      - name: https
        port: 443
        targetPort: 8443
```
additionalIngress:
    some-other-ingress-gateway: {}
additionalEgress:
    some-other-egress-gateway: {}

policy:
    type: Mixer
    mixer: # only applies if policy.type: Mixer
        enableChecks: true
        failOpen: false

telemetry:
    type: Istiod # or Mixer
    mixer: # only applies if telemetry.type: Mixer, for v1 telemetry
        sessionAffinity: false
    batching:
        maxEntries: 100
        maxTime: 1s
    adapters:
        kubernetesenv: true
        stdio:
            enabled: true
            outputAsJSON: true

addons:
    grafana:
        enabled: true
    install:
        config:
            env: {}
            envSecrets: {}
        service:
            ingress:
                contextPath: /grafana
                tls: termination: reencrypt
    kiali:
        name: kiali
        enabled: true
    install: # install kiali CR if not present
    dashboard:
        viewOnly: false
        enableGrafana: true
        enableTracing: true
        enablePrometheus: true
    service:
        ingress:
            contextPath: /kiali
    jaeger:
        name: jaeger
        install:
            storage:
                type: Elasticsearch # or Memory
                memory:
                    maxTraces: 100000
                elasticsearch:
                    nodeCount: 3
1.22.2. 3scale configuration

The following table explains the parameters for the 3scale Istio Adapter in the ServiceMeshControlPlane resource.

Example 3scale parameters

```
storage: {}
redundancyPolicy: SingleRedundancy
indexCleaner: {}
ingress: {}  # jaeger ingress configuration
runtime:
  components:
    pilot:
      deployment:
        replicas: 2
      pod:
        affinity: {}
        container:
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 500m
              memory: 128Mi
    grafana:
      deployment: {}
      pod: {}
    kiali:
      deployment: {}
      pod: {}

spec:
  addons:
    3Scale:
      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_CLIENT_TIMEOUT_SECONDS: 10
      PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS: 60
      PARAM_USE_CACHED_BACKEND: false
      PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS: 15
      PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED: true
```

Table 1.12. 3scale parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Values</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether to use the 3scale adapter</td>
<td>true/false</td>
<td>false</td>
</tr>
<tr>
<td>PARAM_THREESCALE_LISTEN_ADDR</td>
<td>Sets the listen address for the gRPC server</td>
<td>Valid port number</td>
<td>3333</td>
</tr>
<tr>
<td>PARAM_THREESCALE_LOG_LEVEL</td>
<td>Sets the minimum log output level.</td>
<td>debug, info, warn, error, or none</td>
<td>info</td>
</tr>
<tr>
<td>PARAM_THREESCALE_LOG_JSON</td>
<td>Controls whether the log is formatted as JSON</td>
<td>true/false</td>
<td>true</td>
</tr>
<tr>
<td>PARAM_THREESCALE_LOG_GRPC</td>
<td>Controls whether the log contains gRPC info</td>
<td>true/false</td>
<td>true</td>
</tr>
<tr>
<td>PARAM_THREESCALE_REPORT_METRICS</td>
<td>Controls whether 3scale system and backend metrics are collected and reported to Prometheus</td>
<td>true/false</td>
<td>true</td>
</tr>
<tr>
<td>PARAM_THREESCALE_METRICS_PORT</td>
<td>Sets the port that the 3scale /metrics endpoint can be scrapped from</td>
<td>Valid port number</td>
<td>8080</td>
</tr>
<tr>
<td>PARAM_THREESCALE_CACHE_TTL_SECONDS</td>
<td>Time period, in seconds, to wait before purging expired items from the cache</td>
<td>Time period in seconds</td>
<td>300</td>
</tr>
<tr>
<td>PARAM_THREESCALE_CACHE_REFRESH_SECONDS</td>
<td>Time period before expiry when cache elements are attempted to be refreshed</td>
<td>Time period in seconds</td>
<td>180</td>
</tr>
<tr>
<td>PARAM_THREESCALE_CACHE_ENTRIES_MAX</td>
<td>Max number of items that can be stored in the cache at any time. Set to 0 to disable caching</td>
<td>Valid number</td>
<td>1000</td>
</tr>
<tr>
<td>PARAM_THREESCALE_CACHE_REFRESH_RETRIES</td>
<td>The number of times unreachable hosts are retried during a cache update loop</td>
<td>Valid number</td>
<td>1</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Values</td>
<td>Default value</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>--------</td>
<td>---------------</td>
</tr>
<tr>
<td>PARAM_THREESCALE_ALLOW_INSECURE_CONN</td>
<td>Allow to skip certificate verification when calling 3scale APIs. Enabling this is not recommended.</td>
<td>true/false</td>
<td>false</td>
</tr>
<tr>
<td>PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS</td>
<td>Sets the number of seconds to wait before terminating requests to 3scale System and Backend</td>
<td>Time period in seconds</td>
<td>10</td>
</tr>
<tr>
<td>PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS</td>
<td>Sets the maximum amount of seconds (+/-10% jitter) a connection may exist before it is closed</td>
<td>Time period in seconds</td>
<td>60</td>
</tr>
<tr>
<td>PARAM_USE_CACHE_BACKEND</td>
<td>If true, attempt to create an in-memory apisonator cache for authorization requests</td>
<td>true/false</td>
<td>false</td>
</tr>
<tr>
<td>PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS</td>
<td>If the backend cache is enabled, this sets the interval in seconds for flushing the cache against 3scale</td>
<td>Time period in seconds</td>
<td>15</td>
</tr>
<tr>
<td>PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED</td>
<td>Whenever the backend cache cannot retrieve authorization data, whether to deny (closed) or allow (open) requests</td>
<td>true/false</td>
<td>true</td>
</tr>
</tbody>
</table>

1.22.3. More information

For more information about how to configure the features in the ServiceMeshControlPlane see the following link.

- Security
- Traffic management
- Metrics and traces
2.1.1. Introduction to Red Hat OpenShift Service Mesh

Red Hat OpenShift Service Mesh addresses a variety of problems in a microservice architecture by creating a centralized point of control in an application. It adds a transparent layer on existing distributed applications without requiring any changes to the application code.

Microservice architectures split the work of enterprise applications into modular services, which can make scaling and maintenance easier. However, as an enterprise application built on a microservice architecture grows in size and complexity, it becomes difficult to understand and manage. Service Mesh can address those architecture problems by capturing or intercepting traffic between services and can modify, redirect, or create new requests to other services.

Service Mesh, which is based on the open source Istio project, 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.

2.1.1.1. Making open source more inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message.

2.1.2. 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.

2.1.2.1. 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.

2.1.2.2. Prerequisites

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

2.1.2.3. 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-rhel8
```

To collect Red Hat OpenShift Service Mesh data for a specific control plane namespace with `must-
gather`, you must specify the Red Hat OpenShift Service Mesh image and namespace. In this example,
replace `<namespace>` with your control plane namespace, such as `istio-system`.

```
$ oc adm must-gather --image=registry.redhat.io/openshift-service-mesh/istio-must-gather-rhel8
gather <namespace>
```

2.1.3. 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.

For additional information about Red Hat OpenShift Service Mesh lifecycle and supported configurations, refer to the Support Policy.

2.1.3.1. 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.

2.1.3.2. Supported Mixer adapters

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

2.1.4. 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.

2.1.4.1. Component versions included in Red Hat OpenShift Service Mesh version 1.1.16

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Istio</td>
<td>1.4.8</td>
</tr>
<tr>
<td>Jaeger</td>
<td>1.17.8</td>
</tr>
</tbody>
</table>
2.1.4.2. New features Red Hat OpenShift Service Mesh 1.1.16

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

2.1.4.3. New features Red Hat OpenShift Service Mesh 1.1.15

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

2.1.4.4. New features Red Hat OpenShift Service Mesh 1.1.14

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

**IMPORTANT**

There are manual steps that must be completed to address CVE-2021-29492 and CVE-2021-31920.

2.1.4.4.1. Manual updates required by CVE-2021-29492 and CVE-2021-31920

Istio contains a remotely exploitable vulnerability where an HTTP request path with multiple slashes or escaped slash characters (\%2F or \%5C) could potentially bypass an Istio authorization policy when path-based authorization rules are used.

For example, assume an Istio cluster administrator defines an authorization DENY policy to reject the request at path /admin. A request sent to the URL path /admin will NOT be rejected by the authorization policy.

According to RFC 3986, the path /admin with multiple slashes should technically be treated as a different path from the /admin. However, some backend services choose to normalize the URL paths by merging multiple slashes into a single slash. This can result in a bypass of the authorization policy (/admin does not match /admin), and a user can access the resource at path /admin in the backend; this would represent a security incident.

Your cluster is impacted by this vulnerability if you have authorization policies using ALLOW action + notPaths field or DENY action + paths field patterns. These patterns are vulnerable to unexpected policy bypasses.

Your cluster is NOT impacted by this vulnerability if:

- You don’t have authorization policies.
- Your authorization policies don’t define paths or notPaths fields.
- Your authorization policies use **ALLOW action + paths** field or **DENY action + notPaths** field patterns. These patterns could only cause unexpected rejection instead of policy bypasses. The upgrade is optional for these cases.

**NOTE**

The Red Hat OpenShift Service Mesh configuration location for path normalization is different from the Istio configuration.

### 2.1.4.4.2. Updating the path normalization configuration

Istio authorization policies can be based on the URL paths in the HTTP request. **Path normalization**, also known as URI normalization, modifies and standardizes the incoming requests’ paths so that the normalized paths can be processed in a standard way. Syntactically different paths may be equivalent after path normalization.

Istio supports the following normalization schemes on the request paths before evaluating against the authorization policies and routing the requests:

**Table 2.1. Normalization schemes**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Example</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NONE</strong></td>
<td>No normalization is done. Anything received by Envoy will be forwarded exactly as-is to any backend service.</td>
<td>../%2Fa/../b is evaluated by the authorization policies and sent to your service.</td>
<td>This setting is vulnerable to CVE-2021-31920.</td>
</tr>
<tr>
<td><strong>BASE</strong></td>
<td>This is currently the option used in the default installation of Istio. This applies the normalize_path option on Envoy proxies, which follows RFC 3986 with extra normalization to convert backslashes to forward slashes.</td>
<td>/a/../b is normalized to /b. da is normalized to /da.</td>
<td>This setting is vulnerable to CVE-2021-31920.</td>
</tr>
<tr>
<td><strong>MERGE_SLASHES</strong></td>
<td>Slashes are merged after the BASE normalization.</td>
<td>/a//b is normalized to /a/b.</td>
<td>Update to this setting to mitigate CVE-2021-31920.</td>
</tr>
</tbody>
</table>
DECODE_AND_MERGE_SLASHES

The strictest setting when you allow all traffic by default. This setting is recommended, with the caveat that you must thoroughly test your authorization policies routes. Percent-encoded slash and backslash characters (%2F, %2f, %5C and %5c) are decoded to / or \ before the MERGE_SLASHES normalization.

/a%2fb is normalized to /a/b.

Update to this setting to mitigate CVE-2021-31920. This setting is more secure, but also has the potential to break applications. Test your applications before deploying to production.

The normalization algorithms are conducted in the following order:

1. Percent-decode %2F, %2f, %5C and %5c.
2. The RFC 3986 and other normalization implemented by the normalize_path option in Envoy.
3. Merge slashes.

WARNING
While these normalization options represent recommendations from HTTP standards and common industry practices, applications may interpret a URL in any way it chooses to. When using denial policies, ensure that you understand how your application behaves.

2.1.4.4.3. Path normalization configuration examples

Ensuring Envoy normalizes request paths to match your backend services’ expectations is critical to the security of your system. The following examples can be used as a reference for you to configure your system. The normalized URL paths, or the original URL paths if NONE is selected, will be:

1. Used to check against the authorization policies.
2. Forwarded to the backend application.

Table 2.2. Configuration examples
Relies on the proxy to do normalization

If your application...

<table>
<thead>
<tr>
<th>Choose...</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE, MERGE_SLASHES or</td>
</tr>
<tr>
<td>DECODE_AND_MERGE_SLASHES</td>
</tr>
<tr>
<td>BASE</td>
</tr>
<tr>
<td>MERGE_SLASHES</td>
</tr>
<tr>
<td>DECODE_AND_MERGE_SLASHES</td>
</tr>
<tr>
<td>NONE</td>
</tr>
</tbody>
</table>

Normalizes request paths based on RFC 3986 and does not merge slashes.

Normalizes request paths based on RFC 3986 and merges slashes, but does not decode percent-encoded slashes.

Normalizes request paths based on RFC 3986, decodes percent-encoded slashes, and merges slashes.

Processes request paths in a way that is incompatible with RFC 3986.

2.1.4.4.4. Configuring your SMCP for path normalization

To configure path normalization for Red Hat OpenShift Service Mesh, specify the following in your ServiceMeshControlPlane. Use the configuration examples to help determine the settings for your system.

SMCP v1 pathNormalization

```yaml
spec:
  global:
    pathNormalization: <option>
```

2.1.4.5. New features Red Hat OpenShift Service Mesh 1.1.13

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

2.1.4.6. New features Red Hat OpenShift Service Mesh 1.1.12

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

2.1.4.7. 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.

2.1.4.8. 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.
2.1.4.9. 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.

2.1.4.10. 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.

2.1.4.11. 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.

2.1.4.12. 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.

This release also added support for configuring cipher suites.

2.1.4.13. 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.

2.1.4.14. 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.


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:

```json
{
    "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):

```bash
$ 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**

```yaml
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:

```bash
$ 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**

```yaml
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:
```


2.1.4.14.2. 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>
   ```

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
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
```

2.1.4.15. 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.

2.1.4.16. New features Red Hat OpenShift Service Mesh 1.1.2

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

2.1.4.17. New features Red Hat OpenShift Service Mesh 1.1.1

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

2.1.4.18. 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.

2.1.4.18.1. 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 Service Mesh to install and defaults to the latest available version.
Note that support for Red Hat OpenShift Service Mesh v1.0 ended in October, 2020. You must upgrade to either v1.1 or v2.0.

### 2.1.5. 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.

#### 2.1.5.1. Deprecated features Red Hat OpenShift Service Mesh 1.1.5

The following custom resources were deprecated in release 1.1.5 and were removed in release 1.1.12

- **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

### 2.1.6. 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
- **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.

### 2.1.6.1. Service Mesh known issues

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

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

### 2.1.6.2. Kiali known issues
NOTE

New issues for Kiali should be created in the OpenShift Service Mesh project with the Component set to Kiali.

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.

### 2.1.6.3. Jaeger known issues

These limitations exist in Jaeger:

- Apache Spark is not supported.
- Jaeger streaming via AMQ/Kafka is unsupported on IBM Z and IBM Power Systems.

These are the known issues in Jaeger:

- **BZ-1918920** The Elasticsearch pods does not get restarted automatically after an update. As a workaround, restart the pods manually.
- **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.

### 2.1.7. Fixed issues

The following issues been resolved in the current release:

#### 2.1.7.1. Service Mesh fixed issues

- **MAISTRA-2371** Handle tombstones in listerInformer. The updated cache codebase was not handling tombstones when translating the events from the namespace caches to the aggregated cache, leading to a panic in the go routine.
- **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-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.
• **MAISTA-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.

• **MAISTA-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.

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

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

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

• **MAISTA-833** Pilot stopped delivering configuration after many namespace deletions and re-creations.

• **MAISTA-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.

• **MAISTA-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.

• **MAISTA-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.

• **MAISTA-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.

• **MAISTA-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.

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

• **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 ServiceMeshControlPlane resource, edit the YAML content directly or update the resource from the command line instead of clicking the toggle controls.

**2.1.7.2. 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.

### 2.1.7.3. Jaeger fixed issues

- **TRACING-1725** Follow-up to TRACING-1631. Additional fix to ensure that Elasticsearch certificates are properly reconciled when there are multiple Jaeger production instances, using the same name but within different namespaces. See also BZ-1918920.

- **TRACING-1631** Multiple Jaeger production instances, using the same name but within different namespaces, causing Elasticsearch certificate issue. When multiple service meshes were installed, all of the Jaeger Elasticsearch instances had the same Elasticsearch secret instead of individual secrets, which prevented the OpenShift Elasticsearch Operator from communicating with all of the Elasticsearch clusters.

- **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`.

- **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 an error: Failed to pull image registry.redhat.io/amq7/amq-streams-kafka-24-rhel7@sha256:f9ceca004f1b7dccb3b82d9a8027961f9fe4104e0ed69752c0bddd8078b4a1076.

## 2.2. UNDERSTANDING RED HAT OPENSShift SERVICE MESH

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

### 2.2.1. 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

2.2.2. 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.

### 2.2.3. Understanding Kiali

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

#### 2.2.3.1. 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.

#### 2.2.3.2. 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 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 are 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) 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 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 to view links to the Grafana dashboard and view Grafana data.

### 2.2.3.3. 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.

### 2.2.4. 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 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—to understand a whole chain of events in a distributed transaction. Distributed tracing lets developers visualize call flows in large service oriented architectures. It can be invaluable in understanding serialization, parallelism, and sources of latency.
Jaeger records the execution of individual requests across the whole stack of microservices, and presents them as traces. A trace is a data/execution path through the system. An end-to-end trace is comprised of one or more spans.

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

2.2.4.1. 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.

2.2.4.2. 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.

### 2.2.4.3. 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.

### 2.2.5. Next steps

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

### 2.3. 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:

#### 2.3.1. Multitenant installations

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.

#### 2.3.1.1. 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.

### 2.3.1.2. 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.

### 2.3.2. 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.

#### 2.3.2.1. Command line tool

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

#### 2.3.2.2. 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.

### 2.3.2.3. 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**

```yaml
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**

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

### 2.3.2.4. 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.

### 2.3.2.5. 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.

2.3.2.6. 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.

2.3.2.7. 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.

2.3.2.8. 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 Automatic route creation.

2.3.2.8.1. 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.

2.3.2.8.2. 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.

2.3.2.8.3. 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.

Additional resources


- Automatic route creation

### 2.3.3. 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.
- Do not 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. If you use Red Hat OpenShift Dedicated, updating the operator files should be restricted to those users with dedicated-admin privileges.

### 2.3.4. 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 when you select either the production or streaming deployment option.
- 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.

### 2.4. PREPARING TO INSTALL RED HAT OPENSIGHT SERVICE MESH

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

#### 2.4.1. 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 4.8 overview.

Install OpenShift Container Platform 4.8.

- Install OpenShift Container Platform 4.8 on AWS
- Install OpenShift Container Platform 4.8 on user-provisioned AWS
- Install OpenShift Container Platform 4.8 on bare metal
- Install OpenShift Container Platform 4.8 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 4.8, see About the OpenShift CLI.

## 2.4.2. 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.

For additional information about Red Hat OpenShift Service Mesh lifecycle and supported configurations, refer to the Support Policy.

### 2.4.2.1. 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.

### 2.4.2.2. Supported Mixer adapters

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

### 2.4.3. Operator overview

Red Hat OpenShift Service Mesh requires the following four Operators:

- **OpenShift Elasticsearch** - (Optional) Provides database storage for tracing and logging with Jaeger. It is based on the open source Elasticsearch project.

- **Jaeger** - Provides tracing to monitor and troubleshoot transactions in complex distributed systems. It is based on the open source Jaeger project.

- **Kiali** - Provides observability for your service mesh. Allows you to view configurations, monitor traffic, and analyze traces in a single console. It is based on the open source Kiali project.

- **Red Hat OpenShift Service Mesh** - Allows you to connect, secure, control, and observe the microservices that comprise your applications. The Service Mesh Operator defines and monitors the `ServiceMeshControlPlane` resources that manage the deployment, updating, and deletion of the Service Mesh components. It is based on the open source Istio project.

---

**WARNING**

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

---

### 2.4.4. Next steps

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

---

### 2.5. INSTALLING RED HAT OPENSSHIFT SERVICE MESH

Installing the Service Mesh involves installing the OpenShift 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.

2.5.1. 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 1.1.16, you must install the OpenShift Elasticsearch Operator, the Jaeger Operator, and the Kiali Operator before the Red Hat OpenShift Service Mesh Operator can install the control plane.

2.5.2. Installing the OpenShift 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 and configure a persistent storage option, in this case, Elasticsearch.

**Prerequisites**

- Access to the OpenShift Container Platform web console.
- An account with the `cluster-admin` role. If you use Red Hat OpenShift Dedicated, you must have an account with the `dedicated-admin` role.

**WARNING**

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

**NOTE**

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

1. Log in to the OpenShift Container Platform web console as a user with the `cluster-admin` role. If you use Red Hat OpenShift Dedicated, you must have an account with the `dedicated-admin` role.


3. Type `Elasticsearch` into the filter box to locate the OpenShift Elasticsearch Operator.

4. Click the `OpenShift Elasticsearch Operator` provided by Red Hat to display information about the Operator.

5. Click `Install`.

6. On the Install Operator page, under Installation Mode select All namespaces on the cluster (default). This makes the Operator available to all projects in the cluster.

7. Under Installed Namespaces select `openshift-operators-redhat` from the menu.

   **NOTE**
   The Elasticsearch installation requires the `openshift-operators-redhat` namespace for the OpenShift Elasticsearch Operator. The other Red Hat OpenShift Service Mesh operators are installed in the `openshift-operators` namespace.

8. Select the Update Channel that matches your OpenShift Container Platform installation. For example, if you are installing on OpenShift Container Platform version 4.8, select the 4.8 update channel.

   **NOTE**
   For OpenShift Container Platform version 4.7 select the 4.6 update channel.


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

10. Click `Install`.

11. On the Installed Operators page, select the `openshift-operators-redhat` project. Wait until you see that the OpenShift Elasticsearch Operator shows a status of "InstallSucceeded" before continuing.

**2.5.3. 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 use Red Hat OpenShift Dedicated, you must have an account with the `dedicated-admin` role.
- If you require persistent storage, you must also install the OpenShift 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. If you use Red Hat OpenShift Dedicated, you must have an account with the `dedicated-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 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.
7. 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.
   - 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.

### 2.5.4. 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.
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 Operator Installation page, select the stable Update Channel.
7. 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.
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.

### 2.5.5. Installing the Operators

To install Red Hat OpenShift Service Mesh, install following Operators in this order. Repeat the procedure for each Operator.
1. (Optional) OpenShift Elasticsearch
2. Jaeger
3. Kiali
4. Red Hat OpenShift Service Mesh

Procedure
1. Log in to the OpenShift Container Platform web console as a user with the `cluster-admin` role.
2. In the OpenShift Container Platform web console, click `Operators` → `OperatorHub`.
3. Type the name of the Operator into the filter box and select the Red Hat version of the Operator. Community versions of the Operators are not supported.
4. Click `Install`.
5. On the `Install Operator` page, select installation options.
   a. For the OpenShift Elasticsearch Operator, in the `Update Channel` section, select `4.6`.
   b. For the Jaeger, Kiali, and Red Hat OpenShift Service Mesh Operators, accept the defaults. The Jaeger, Kiali and Red Hat OpenShift Service Mesh are installed in the `openshift-operators` namespace. The OpenShift Elasticsearch Operator is installed in the `openshift-operators-redhat` namespace.
6. Click `Install`. Wait until the Operator has installed before repeating the steps for the next Operator in the list.
7. After all you have installed all four Operators, click `Operators` → `Installed Operators` to verify that your Operators installed.

2.5.6. 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.

2.5.6.1. 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.

**2.5.6.2. 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.
   
   ```bash
   $ oc login https://{HOSTNAME}:6443
   ```

2. Create a project named `istio-system`.
   
   ```bash
   $ 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:
   
   ```bash
   $ oc create -n istio-system -f istio-installation.yaml
   ```

5. Execute the following command to see the status of the control plane installation.
   
   ```bash
   $ oc get smcp -n istio-system
   ```

   The installation has finished successfully when the STATUS column is **InstallSuccessful**.

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic-install</td>
<td>9/9</td>
<td>InstallSuccessful</td>
<td>0</td>
<td>4m25s</td>
</tr>
</tbody>
</table>

6. Run the following command to watch the progress of the Pods during the installation process:
   
   ```bash
   $ oc get pods -n istio-system -w
   ```

   You should see output similar to the following:

   **Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>grafana-7bf5764d9d-2b2f6</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>28h</td>
</tr>
<tr>
<td>istio-citadel-576b9c5bbd-z84z4</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>28h</td>
</tr>
<tr>
<td>istio-egressgateway-5476bc4656-r4zdv</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>28h</td>
</tr>
<tr>
<td>istio-galley-7d57b47bb7-lqdxv</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>28h</td>
</tr>
<tr>
<td>istio-ingressgateway-dbb8f7f46-cf6n5</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>28h</td>
</tr>
<tr>
<td>istio-pilot-546bf69578-ccg5x</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>28h</td>
</tr>
<tr>
<td>istio-policy-77fd498655-7pvjw</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>28h</td>
</tr>
<tr>
<td>istio-sidecar-injector-df45bd899-ctxdt</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>28h</td>
</tr>
<tr>
<td>istio-telemetry-66f697d6d5-cj28l</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>28h</td>
</tr>
<tr>
<td>jaeger-896945cbc-7lqrr</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>11h</td>
</tr>
<tr>
<td>kiali-78d9c5b87c-snjzh</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>22h</td>
</tr>
<tr>
<td>prometheus-6dff867c97-gr2n5</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>28h</td>
</tr>
</tbody>
</table>
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.

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

The ServiceMeshMemberRoll lists the projects that belong 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, for example istio-system.

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

You can add one or more projects to the Service Mesh member roll from the web console.

**Prerequisites**

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

**Procedure**

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

2. If you do not already have services for your mesh, or you are starting from scratch, create a project. It must be different from istio-system.
   b. Enter a name in the Name field.
   c. Click Create.

3. Navigate to Operators → Installed Operators.

4. Click the Project menu and choose the project where your ServiceMeshControlPlane resource is deployed from the list, for example istio-system.

5. Click the Red Hat OpenShift Service Mesh Operator.

6. Click the Istio Service Mesh Member Roll tab.

7. Click Create ServiceMeshMemberRoll

8. Click Members, then enter the name of your project in the Value field. You can add any number of projects, but a project can only belong to one ServiceMeshMemberRoll resource.

9. Click Create.

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

You can add a project to the ServiceMeshMemberRoll from the command line.
Prerequisites

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

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

2. If you do not already have services for your mesh, or you are starting from scratch, create a project. It must be different from `istio-system`.

   ```bash
   $ oc new-project {your-project}
   ``

3. To add your projects as members, modify the following example YAML. You can add any number of projects, but a project can only belong to one `ServiceMeshMemberRoll` resource.

   **Example servicemeshmemberroll-default.yaml**

   ```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
   ```

4. Run the following command to upload and create the `ServiceMeshMemberRoll` resource in the `istio-system` namespace.

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

5. Run the following command to verify the `ServiceMeshMemberRoll` was created successfully.

   ```bash
   $ oc get smmr -n istio-system default
   ``

   The installation has finished successfully when the `STATUS` column is `Configured`.

2.5.8. Adding or removing projects from the service mesh

You can add or remove projects from 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.

2.5.8.1. Adding or removing projects from the member roll using 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 resource 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.

2.5.8.2. Adding or removing projects from the member roll using the CLI

You can 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**

   ```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
   ```

### 2.5.9. 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.

#### 2.5.9.1. 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.

### 2.5.10. Next steps

- Prepare to deploy applications on Red Hat OpenShift Service Mesh.
2.6. 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.

2.6.1. Enabling mutual Transport Layer Security (mTLS)

Mutual Transport Layer Security (mTLS) is a protocol where two parties authenticate each other. 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.

2.6.1.1. 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
```

2.6.1.1.1. 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"
kkind: "Policy"
metadata:
  name: default
  namespace: <NAMESPACE>
spec:
  peers:
    - mtls: {}
```

2.6.1.2. 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.

---
2.6.1.3. 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.security.controlPlane.tls.minProtocolVersion` or `spec.security.controlPlane.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.

Table 2.3. Valid values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS_AUTO</td>
<td>default</td>
</tr>
<tr>
<td>TLSv1_0</td>
<td>TLS version 1.0</td>
</tr>
<tr>
<td>TLSv1_1</td>
<td>TLS version 1.1</td>
</tr>
<tr>
<td>TLSv1_2</td>
<td>TLS version 1.2</td>
</tr>
<tr>
<td>TLSv1_3</td>
<td>TLS version 1.3</td>
</tr>
</tbody>
</table>

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

2.6.2. 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_ECDHE_RSA_WITH_AES_128_CBC_SHA256`
- `TLS_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_RSA_WITH_AES_128_CBC_SHA256`
2.6.3. Adding an external certificate authority key and certificate

By default, Red Hat OpenShift Service Mesh generates a 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

- Install 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.
- Deploy the Bookinfo sample application to verify the results with these instructions.

2.6.3.1. 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 named `ca-cert.pem`, the key is `ca-key.pem`, and the root certificate, which signs `ca-cert.pem`, is named `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`.

   ```bash
   $ 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 `spec.security.dataPlane.mtls: true` to `true` and configure your certificateAuthority like the following example. The default `rootCADir` is `/etc/cacerts`. You do not need to set the `privateKey` if the key and certs are mounted in the default location. Service Mesh reads the certificates and key from the secret-mount files.

   ```yaml
   apiVersion: maistra.io/v2
   kind: ServiceMeshControlPlane
   spec:
     security:
       dataPlane:
         mtls: true
         certificateAuthority:
           type: Istiod
   ```
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.

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

### 2.6.3.2. 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**.

   ```bash
   $ 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.

   ```bash
   $ oc exec -it $RATINGSPOD -c istio-proxy -- /bin/cat /var/run/secrets/istio/root-cert.pem > /tmp/pod-root-cert.pem
   $ oc exec -it $RATINGSPOD -c istio-proxy -- /bin/cat /etc/certs/cert-chain.pem > /tmp/pod-cert-chain.pem
   ``
   The file **/tmp/pod-root-cert.pem** contains the root certificate propagated to the pod.

   ```bash
   $ 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.

   ```bash
   $ 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.

   ```bash
   $ 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

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

2.6.3.3. Removing the certificates

To remove the certificates you added, follow these steps.

1. Remove the secret `cacerts`.

   $ oc delete secret cacerts -n istio-system


   ```yaml
   apiVersion: maistra.io/v2
   kind: ServiceMeshControlPlane
   spec:
     dataPlane:
       mtls: true
   ```

2.7. 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.

2.7.1. 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.

2.7.1.1. 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. A virtual service enables you to turn a monolithic application into a service comprised of distinct microservices with a seamless consumer experience.

- Configure traffic rules in combination with gateways to control ingress and egress traffic.

### 2.7.1.1.1. Configuring virtual services

Requests are routed to 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.

**Procedure**

1. Create a YAML file using the following example to route requests to different versions of a the Bookinfo sample application service depending on which user connects to the application.

   **Example VirtualService.yaml**

   ```yaml
   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
           - destination:
             host: reviews
             subset: v3
   
   $ oc apply -f VirtualService.yaml
   ```

2. Run the following command to apply `VirtualService.yaml`, where `VirtualService.yaml` is the path to the file.

   ```bash
   $ oc apply -f VirtualService.yaml
   ```
2.7.1.2. Configuring your virtual host

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

2.7.1.2.1. Hosts

The **hosts** field lists the virtual service’s destination address to which the routing rules apply. This is the address(es) that are used to send requests to the service.

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

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

2.7.1.2.2. Routing rules

The **http** section contains the virtual service’s routing rules which describe match conditions and actions for routing HTTP/1.1, HTTP2, and gRPC traffic sent to the destination as specified in the hosts field. A routing rule consists of the destination where you want the traffic to go and any specified match conditions.

**Match condition**

The first routing rule in the example has a condition that 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.

```yaml
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:

```yaml
spec:
  hosts:
    - reviews
  http:
    - match:
      - headers:
        end-user:
          exact: jason
  route:
```
2.7.1.2.3. 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.

2.7.1.2.3.1. Load balancing options

By default, Red Hat OpenShift Service Mesh uses a round-robin load balancing policy, where each service instance in the 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
    - name: v3
      labels:
        version: v3
```

2.7.1.2.4. 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 allow 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 and configure 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 enables you to configure a dedicated exit node for the traffic leaving the mesh. This enables you to limit which services have access to external networks, which adds security control to your service mesh. You can also use a gateway to configure a purely internal proxy.

**Gateway example**

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

```yaml
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:

```yaml
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.
2.7.1.2.5. 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 send traffic to the service as if it is a service in your mesh. Service entries allow you to do the following:

- Manage traffic for services that run outside of the service mesh.
- 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.

**NOTE**

Add services from a different cluster to the mesh to configure a multicluster Red Hat OpenShift Service Mesh mesh on Kubernetes.

**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:

```yaml
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:

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: ext-res-dr
spec:
  host: ext-svc.example.com
  trafficPolicy:
    tls:
      mode: MUTUAL
```
### 2.7.2. Configuring ingress using a gateway

An ingress gateway is a load balancer operating at the edge of the mesh that receives incoming HTTP/TCP connections. It configures exposed ports and protocols but does not include any traffic routing configuration. Traffic routing for ingress traffic is instead configured with routing rules, the same way as for internal service requests.

The following steps show how to create a gateway and configure a VirtualService to expose a service in the Bookinfo sample application to outside traffic for paths `/productpage` and `/login`.

**Procedure**

1. Create a gateway to accept traffic.
   - a. Create a YAML file, and copy the following YAML into it.

   **Gateway example gateway.yaml**

   ```yaml
   apiVersion: networking.istio.io/v1alpha3
   kind: Gateway
   metadata:
     name: bookinfo-gateway
   spec:
     selector:
       istio: ingressgateway
     servers:
       - port:
           number: 80
           name: http
           protocol: HTTP
           hosts:
             - "*

   $ oc apply -f gateway.yaml
   
   2. Create a VirtualService object to rewrite the host header.
      - a. Create a YAML file, and copy the following YAML into it.

   **Virtual service example vs.yaml**

   ```yaml
   apiVersion: networking.istio.io/v1alpha3
   kind: VirtualService
   metadata:
     name: bookinfo
   spec:
     hosts:
     - "*
```
b. Apply the YML file.

```
$ oc apply -f vs.yaml
```

3. Test that the gateway and VirtualService have been set correctly.

a. Set the Gateway URL.

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

b. Set the port number.

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

c. Test a page that has been explicitly exposed.

```
curl -s -I "$GATEWAY_URL/productpage"
```

The expected result is 200.

### 2.7.3. Routing tutorial

The Service Mesh Bookinfo sample application consists of four separate microservices, each with multiple versions. After installing the Bookinfo sample application, three different versions of the reviews microservice run concurrently.

When you 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.
Prerequisites:

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

2.7.4. Managing ingress traffic

In Red Hat OpenShift Service Mesh, the Ingress Gateway enables features such as monitoring, security, and route rules to apply to traffic that enters the cluster. Use a Service Mesh gateway to expose a service outside of the service mesh.

2.7.4.1. Determining the ingress IP and ports

Ingress configuration differs depending on if your environment supports an external load balancer. An external load balancer is set in the ingress IP and ports for the cluster. To determine if your cluster’s IP and ports are configured for external load balancers, run the following command.

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.

Determine the ingress according to your environment. For an environment with load balancer support, Determining ingress ports with a load balancer. For an environment without load balancer support, Determining ingress ports without a load balancer. After you have determined the ingress ports, see Configuring ingress using a gateway to complete your configuration.

2.7.4.1.1. Determining ingress ports with a load balancer

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

Procedure

1. Run the following command to set the ingress IP and ports. This command sets a variable in your terminal

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

2. Run the following command to set the ingress port.

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

3. Run the following command to set the secure ingress port.

   $ export SECURE_INGRESS_PORT=$(oc -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].port}')
4. Run the following command to set the TCP ingress port.

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

**NOTE**

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 an IP address.
Instead, it’s a host name, and the previous command fails to set the `INGRESS_HOST`
environment variable.

In that case, 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}')
```

### 2.7.4.1.2. Determining ingress ports without a load balancer

If your environment does not have an external load balancer, determine the ingress ports and use a node
port instead.

**Procedure**

1. Set the ingress ports.

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

2. Run the following command to set the secure ingress port.

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

3. Run the following command to set the TCP ingress port.

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

### 2.7.5. 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.

#### 2.7.5.1. 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
```

### 2.7.5.2. 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. Configure an OpenShift Container Platform wildcard policy before configuring a wildcard host Gateway. For more information, see the "Links" section.

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 <control_plane_namespace> get routes
```

#### Expected output

<table>
<thead>
<tr>
<th>NAME</th>
<th>HOST/PORT</th>
<th>PATH</th>
<th>SERVICES</th>
<th>PORT</th>
<th>TERMINATION</th>
<th>WILDCARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>gateway1-lvlfn</td>
<td>bookinfo.exmple.com</td>
<td></td>
<td>istio-ingressgateway</td>
<td>&lt;all&gt;</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>gateway1-scqhv</td>
<td><a href="http://www.bookinfo.com">www.bookinfo.com</a></td>
<td></td>
<td>istio-ingressgateway</td>
<td>&lt;all&gt;</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

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.
2.7.6. Links

For more information about configuring an OpenShift Container Platform wildcard policy, see Using wildcard routes.

2.8. DEPLOYING APPLICATIONS ON RED HAT OPENSHELFT 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.

2.8.1. Prerequisites

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

2.8.2. 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.

2.8.2.1. 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.

```yaml
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.

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

**2.8.3. Enabling automatic sidecar injection**
When deploying an application, you must opt-in to injection by setting the `sidecar.istio.io/inject` annotation to "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.

**Procedure**

1. Open the application’s deployment configuration YAML file in an editor. To find a deployment use the `oc get` command. For example, for an app called `sleep` in the `sleep` namespace, use the following command to see the resource in YAML format.
   
   ```bash
   oc get deployment sleep -o yaml
   ```

2. Add `sidecar.istio.io/inject` to the configuration YAML with a value of "true" in the `spec.template.metadata.annotations.sidecar.istio/inject` field. See the following example for an app called `sleep`.

   **Sleep test application example sleep.yaml**

   ```yaml
   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: curlimages/curl
               command: ["/bin/sleep","3650d"]
               imagePullPolicy: IfNotPresent
   ```

3. Save the configuration file.

4. Add the file back to the project that contains your app. In this example, `sleep` is the name of the project that contains the `sleep` app and `sleep.yaml` is the file you edited.
   
   ```bash
   $ oc apply -n sleep -f sleep.yaml
   ```

5. To verify that the resource uploaded successfully, run the following command.
   
   ```bash
   $ oc get deployment sleep -o yaml
   ```
2.8.4. 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.

Example injection-template.yaml

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

2.8.5. 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`.

**NOTE**

The examples use `<istio-system>` as the control plane namespace. Replace this value with the namespace where you deployed the Service Mesh Control Plane (SMCP).
**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`.

**2.8.5.1. 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 service mesh that were previously exposed through an OpenShift Container Platform route.

- Traffic into the service mesh must always go through the ingress-gateway for Istio to work properly.
- Deploy services external to the service mesh in separate namespaces that are not in any service 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.

**2.8.6. Bookinfo example application**

The Bookinfo example application allows you to test your Red Hat OpenShift Service Mesh 2.0.6 installation on OpenShift Container Platform.

The Bookinfo application displays information about a book, similar to a single catalog entry of an online book store. The application displays a page that describes 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.

2.8.6.1. Installing the Bookinfo application

This tutorial walks you through how to create a sample application by creating a project, deploying the Bookinfo application to that project, and viewing the running application in Service Mesh.

Prerequisites:

• OpenShift Container Platform 4.1 or higher installed.

• Red Hat OpenShift Service Mesh 2.0.6 installed.

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

• An account with the cluster-admin role.

NOTE

The Bookinfo sample application cannot be installed on IBM Z and IBM Power Systems.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with cluster-admin rights. If you use Red Hat OpenShift Dedicated, you must have an account with the dedicated-admin role.

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 tab.

   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 ServiceMeshMemberRoll, click Create ServiceMeshMemberRoll.
9. Click **Members**, then enter the name of your project in the **Value** field.

10. Click **Create** to save the updated Service Mesh Member Roll.
   
   + Or, save the following example to a YAML file.

   ```yaml
   Bookinfo ServiceMeshMemberRoll example servicemeshmemberroll-default.yaml
   
   apiVersion: maistra.io/v1
   kind: ServiceMeshMemberRoll
   metadata:
     name: default
   spec:
     members:
       - bookinfo
   
   a. Run the following command to upload that file and create the **ServiceMeshMemberRoll** resource in the **istio-system** namespace.

   ```sh
   $ oc create -n istio-system -f servicemeshmemberroll-default.yaml
   
   $ oc get smmr -n istio-system
   
   NAME      READY   STATUS       AGE
   default   1/1     Configured   2m27s
   
   11. Run the following command to verify the **ServiceMeshMemberRoll** was created successfully.

   ```sh
   $ oc get smmr -n istio-system
   
   NAME      READY   STATUS       AGE
   default   1/1     Configured   2m27s
   
   The installation has finished successfully when the **STATUS** column is **Configured**.

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

   ```sh
   $ oc apply -n bookinfo -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.0/samples/bookinfo/platform/kube/bookinfo.yaml
   
   You should see output similar to the following:

   service/details created
   serviceaccount/bookinfo-details created
   deployment.apps/details-v1 created
   service/ratings created
   serviceaccount/bookinfo-ratings created
   deployment.apps/ratings-v1 created
   service/reviews created
   serviceaccount/bookinfo-reviews created
   deployment.apps/reviews-v1 created
   deployment.apps/reviews-v2 created
   deployment.apps/reviews-v3 created
   service/productpage created
   serviceaccount/bookinfo-productpage created
   deployment.apps/productpage-v1 created
13. Create the ingress gateway by applying the `bookinfo-gateway.yaml` file:

```bash
$ oc apply -n bookinfo -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.0/samples/bookinfo/networking/bookinfo-gateway.yaml
```

You should see output similar to the following:

```
gateway.networking.istio.io/bookinfo-gateway created
virtualservice.networking.istio.io/bookinfo created
```

14. 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`.
```

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

2.8.6.2. Adding default destination rules

Before you can use the Bookinfo application, you must first 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:

     ```bash
     $ oc apply -n bookinfo -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.0/samples/bookinfo/networking/destination-rule-all.yaml
     ```

   - If you enabled mutual TLS:

     ```bash
     $ oc apply -n bookinfo -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.0/samples/bookinfo/networking/destination-rule-all-mtls.yaml
     ```

You should see output similar to the following:

```
destinationrule.networking.istio.io/productpage created
destinationrule.networking.istio.io/reviews created
destinationrule.networking.istio.io/ratings created
destinationrule.networking.istio.io/details created
```

2.8.6.3. Verifying the Bookinfo installation

To confirm that the sample Bookinfo application was successfully deployed, perform the following steps.
Prerequisites

- Red Hat OpenShift Service Mesh 2.0.6 installed.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as oc.
- Complete the steps for installing the Bookinfo sample app.

Procedure

1. Log in to the OpenShift Container Platform CLI.

2. Verify that all pods are ready with this command:

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

   All pods should have a status of Running. You should see output similar to the following:

   ```
   NAME               READY   STATUS    RESTARTS   AGE
   details-v1-55b869668-jh7hb   2/2     Running   0      12m
   productpage-v1-6fc77f794-nsl8r 2/2     Running   0      12m
   ratings-v1-7d7d8d8b56-55scn   2/2     Running   0      12m
   reviews-v1-868597db96-bdxgq   2/2     Running   0      12m
   reviews-v2-5b64f47978-cvssp   2/2     Running   0      12m
   reviews-v3-6ddf49b55b-vcwpf   2/2     Running   0      12m
   ```

3. Run the following command to retrieve the URL for the product page:

   ```
   echo "http://$GATEWAY_URL/productpage"
   ```

4. Copy and paste the output in a web browser to verify the Bookinfo product page is deployed.

2.8.6.4. 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 2.0.6 installed.
- Access to the OpenShift Container Platform Command-line Interface (CLI) also known as oc.

2.8.6.4.1. Delete the Bookinfo project

Procedure

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

2. Click to **Home → Projects**.
3. Click 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
```

### 2.8.6.4.2. 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**. In this example, **istio-system** is the name of the control plane project.

```
$ oc -n istio-system patch --type='json' smmr default -p '[["op": "remove", "path": "/spec/members", "value": ["bookinfo"]]]'
```

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

### 2.8.7. Generating example traces and analyzing trace data

Jaeger is an open source distributed tracing system. With Jaeger, you can perform a trace that follows the path of a request through various microservices which make up an application. Jaeger is installed by default as part of the Service Mesh.

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

**Prerequisites:**

- OpenShift Container Platform 4.1 or higher installed.
- Red Hat OpenShift Service Mesh 2.0.6 installed.
- Jaeger enabled during the installation.
- Bookinfo example application installed.
Procedure

1. After installing the Bookinfo sample application, send traffic to the mesh. Enter the following command several times.

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

   This command simulates a user visiting the **productpage** microservice of the application.

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

   - Alternatively, use the CLI to query for details of the route. In this example, **istio-system** is the control plane namespace:

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

   a. Enter the following command to reveal the URL for the Jaeger console. Paste the result in a browser and navigate to that URL.

   ```
   echo $JAEGGER_URL
   ```

3. Log in using the same user name and password as you use to access the OpenShift Container Platform console.

4. In the left pane of the Jaeger dashboard, from the Service menu, select **productpage.bookinfo** and click the Find Traces button at the bottom of the pane. A list of traces is displayed.

5. Click one of the traces in the list to open a detailed view of that trace. If you click the first one in the list, which is the most recent trace, you see the details that correspond to the latest refresh of the /productpage.

### 2.9. 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.

**Before you begin**

You can observe the data flow through your application if you have an application 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.

#### 2.9.1. Viewing service mesh data

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.

To access the Kiali console you must have Red Hat OpenShift Service Mesh installed and projects configured for the service mesh.

**Procedure**
1. Use the perspective switcher to switch to the Administrator perspective.

2. Click Home > Projects.

3. Click the name of your project. For example click bookinfo.

4. In the Launcher section, click Kiali.

5. Log in to the Kiali console with the same user name and password that you use to access the OpenShift Container Platform console.

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

2.9.2. Working with data in the Kiali console

From the Graph menu in the Kiali console, you can use the following graphs and viewing tools to gain deeper insights about data that travels through your service mesh. These tools can help you identify problems with services or workloads.

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 application. All versions of an application 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 application and version labels. If your application does not use version labels, use this the graph.

- The **Service graph** shows a node for each service in your mesh but excludes all applications 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.

2.9.2.1. 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.

**Prerequisite**

- Install the Bookinfo sample application.

**Procedure**

1. Send traffic to the mesh by entering the following command several times.

   ```bash
   $ curl "http://$GATEWAY_URL/productpage"
   
   This command simulates a user visiting the **productpage** microservice of the application.
   
2. In the main navigation, click **Graph** to view a namespace graph.
3. Select **bookinfo** from the **Namespace** menu.

### 2.10. 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.

#### 2.10.1. 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*)
- Enabling backend cache requires 3scale 2.9 or greater
- Red Hat OpenShift Service Mesh prerequisites

**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 account credentials. You can optionally add a **service_id** to a handler, but this is kept for backwards compatibility only, since it would render the handler only useful for one service in your 3scale account. If you add **service_id** to a handler, enabling 3scale for other services requires you to create more handlers with different **service_ids**.

Use a single handler per 3scale account by following the steps below:

**Procedure**

1. Create a handler for your 3scale account and specify your account credentials. Omit any service identifier.

```yaml
apiVersion: "config.istio.io/v1alpha2"
kind: handler
metadata:
  name: threescale
spec:
  adapter: threescale
  params:
    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.

2. Edit or patch the Deployment resource of any services belonging to your 3scale account as follows:
   a. Add the `"service-mesh.3scale.net/service-id"` label with a value corresponding to a valid `service_id`.
   b. Add the `"service-mesh.3scale.net/credentials"` label with its value being the `name of the handler resource` from step 1.

3. Do step 2 to link it to your 3scale account credentials and to its service identifier, whenever you intend to add more services.

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

**Rule configuration example**

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

### 2.10.1.1. Generating 3scale custom resources

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

**Table 2.4. Usage**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Produces help output for available options</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>--name</td>
<td>Unique name for this URL, token pair</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>-n, --namespace</td>
<td>Namespace to generate templates</td>
<td>No</td>
<td>istio-system</td>
</tr>
<tr>
<td>-t, --token</td>
<td>3scale access token</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>-u, --url</td>
<td>3scale Admin Portal URL</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Required</td>
<td>Default value</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>---------------</td>
</tr>
<tr>
<td>--backend-url</td>
<td>3scale backend URL. If set, it overrides the value that is read from system configuration</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>-s, --service</td>
<td>3scale API/Service ID</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>--auth</td>
<td>3scale authentication pattern to specify (1=API Key, 2=App Id/App Key, 3=OIDC)</td>
<td>No</td>
<td>Hybrid</td>
</tr>
<tr>
<td>-o, --output</td>
<td>File to save produced manifests to</td>
<td>No</td>
<td>Standard output</td>
</tr>
<tr>
<td>--version</td>
<td>Outputs the CLI version and exits immediately</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

2.10.1.1. Generate templates from URL examples

**NOTE**
- Run the following commands via `oc exec` from the 3scale adapter container image in [*Generating manifests from a deployed adapter*](#).
- Use the **3scale-config-gen** command to help avoid YAML syntax and indentation errors.
- You can omit the **--service** if you use the annotations.
- This command must be invoked from within the container image via `oc exec`.

**Procedure**
- Use the **3scale-config-gen** command to autogenerate templates files allowing the token, URL pair to be shared by multiple services as a single handler:

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

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

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

**Additional resources**
- [Tokens](#)
2.10.1.2. Generating manifests from a deployed adapter

NOTE

- **NAME** is an identifier you use to identify with the service you are managing with 3scale.

- The **CREDENTIALS_NAME** reference is an identifier that corresponds to the **match** section in the rule configuration. This is automatically set to the **NAME** identifier if you are using the CLI tool.

- Its value does not need to be anything specific: the label value should just match the contents of the rule. See Routing service traffic through the adapter for more information.

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

   ```sh
   $ 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:

   ```sh
   $ export CREDENTIALS_NAME="replace-me"
   export SERVICE_ID="replace-me"
   export DEPLOYMENT="replace-me"
   patch="$(oc get deployment "$DEPLOYMENT" --template='{{ "spec":{{ "metadata":{ "labels":{{ range $k,$v := .spec.template.metadata.labels }}|{{"$k":"$v"}}}}}}' )"
   oc patch deployment "$DEPLOYMENT" --patch """"$patch"
   ```

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

**Prerequisites**

- Credentials and service ID from your 3scale administrator.

**Procedure**

1. Match the rule `destination.labels["service-mesh.3scale.net/credentials"] == "threescale"` that you previously created in the configuration, in the `kind: rule` resource.

2. Add the above label to `PodTemplateSpec` on the Deployment of the target workload to integrate a service. The value, `threescale`, refers to the name of the generated handler. This handler stores the access token required to call 3scale.

3. Add the `destination.labels["service-mesh.3scale.net/service-id"] == "replace-me"` label to the workload to pass the service ID to the adapter via the instance at request time.

**2.10.2. 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`

2. Click Settings.

3. Select the Istio option under Deployment.
   
   - The API Key (user_key) option under Authentication is selected by default.

4. Click Update Product to save your selection.

5. Click Configuration.

6. Click Update Configuration.

**2.10.3. Caching behavior**

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

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

By using the refreshing process, cached values whose hosts become unreachable will be retried before eventually being purged when past their expiry.
2.10.4. 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.

2.10.4.1. 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"].

2.10.4.1.1. 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**

```yaml
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"]`.

### 2.10.4.1.2. 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**

```yaml
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"]`.

### 2.10.4.1.3. OpenID authentication method

To use the [OpenID Connect (OIDC) authentication method](https://openid.net/connect/), 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**

```yaml
apiVersion: "config.istio.io/v1alpha2"
kind: instance
metadata:
  name: threescale-authorization
spec:
  template: threescale-authorization
  params:
    Subject:
      properties:
```

---

OpenShift Container Platform 4.8 Service Mesh

194
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 a Request authorization 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 RequestAuthentication defined below, replace issuer, jwksUri, and selector as appropriate.

OpenID Policy example

```yaml
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: bookinfo
spec:
  selector:
    matchLabels:
      app: productpage
  jwtRules:
  - 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

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"] |
```

2.10.4.1.4. 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

```yaml
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"] | ""
```
2.10.5. 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.

2.10.6. 3scale Istio adapter verification

You might want to check whether the 3scale Istio adapter is working as expected. If your adapter is not working, use the following steps to help troubleshoot the problem.

Procedure

1. Ensure the 3scale-adapter pod is running in the control plane namespace:

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

2. Check that the 3scale-adapter pod has printed out information about itself booting up, such as its version:

   ```
   $ oc logs <istio-system>
   ```

3. When performing requests to the services protected by the 3scale adapter integration, always try requests that lack the right credentials and ensure they fail. Check the 3scale adapter logs to gather additional information.

Additional resources

- [Inspecting pod and container logs](#)

2.10.7. 3scale Istio adapter troubleshooting checklist

As the administrator installing the 3scale Istio adapter, there are a number of scenarios that might be causing your integration to not function properly. Use the following list to troubleshoot your installation:

- Incorrect YAML indentation.
- Missing YAML sections.
- Forgot to apply the changes in the YAML to the cluster.
- Forgot to label the service workloads with the `service-mesh.3scale.net/credentials` key.
- Forgot to label the service workloads with `service-mesh.3scale.net/service-id` when using handlers that do not contain a `service_id` so they are reusable per account.
The Rule custom resource points to the wrong handler or instance custom resources, or the references lack the corresponding namespace suffix.

The Rule custom resource match section cannot possibly match the service you are configuring, or it points to a destination workload that is not currently running or does not exist.

Wrong access token or URL for the 3scale Admin Portal in the handler.

The Instance custom resource’s params/subject/properties section fails to list the right parameters for app_id, app_key, or client_id, either because they specify the wrong location such as the query parameters, headers, and authorization claims, or the parameter names do not match the requests used for testing.

Failing to use the configuration generator without realizing that it actually lives in the adapter container image and needs oc exec to invoke it.

2.11. REMOVING RED HAT OPENSSHIFT SERVICE MESH

To remove Red Hat OpenShift Service Mesh from an existing OpenShift Container Platform instance, remove the control plane before removing the operators.

2.11.1. Removing the Red Hat OpenShift Service Mesh control plane

To uninstall Service Mesh from an existing OpenShift Container Platform instance, you must first delete the control plane and the Operators. Then, you must run commands to manually remove residual resources.

2.11.1.1. Removing the control plane with the web console

You can remove the Red Hat OpenShift Service Mesh control plane by using the web console.

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 Service Mesh Control Plane under Provided APIs.
5. Click the ServiceMeshControlPlane menu.
6. Click Delete Service Mesh Control Plane
7. Click Delete on the confirmation dialog window to remove the ServiceMeshControlPlane.

2.11.1.2. Removing the control plane from the CLI

You can remove the Red Hat OpenShift Service Mesh control plane by using the CLI.

Procedure
1. Log in to the OpenShift Container Platform CLI.

2. Run this command to retrieve the name of the installed ServiceMeshControlPlane:

   ```bash
   $ oc get smcp -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:

   ```bash
   $ oc delete smcp -n istio-system <name_of_custom_resource>
   ```

### 2.11.2. Removing the installed Operators

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

#### 2.11.2.1. Removing the Operators

Follow this procedure to remove the Operators that make up Red Hat OpenShift Service Mesh. Repeat the steps for each of the following Operators.

- Red Hat OpenShift Service Mesh
- Kiali
- Jaeger
- OpenShift Elasticsearch

#### 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 each Operator. Then, click the Operator name.

3. On the **Operator Details** page, select **Uninstall Operator** from the **Actions** menu. Follow the prompts to uninstall each Operator.

#### 2.11.2.2. Clean up Operator resources

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

**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. If you intend to keep using Jaeger as a stand alone service without service mesh, do not delete the Jaeger resources.

**NOTE**

The Operators are installed in the *openshift-operators* namespace by default. If you installed the Operators in another namespace, replace *openshift-operators* with the name of the project where the Red Hat OpenShift Service Mesh Operator was installed.

```bash
$ oc delete validatingwebhookconfiguration/openshift-operators.servicemesh-resources.maistra.io
$ oc delete mutatingwebhookconfigurations/openshift-operators.servicemesh-resources.maistra.io
$ oc delete -n openshift-operators daemonset/istio-node
$ oc delete clusterrole/istio-admin clusterrole/istio-cni clusterrolebinding/istio-cni
$ oc delete clusterrole istio-view istio-edit
$ 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
$ oc delete crds jaegers.jaegertracing.io
$ oc delete svc admission-controller -n <operator-project>
$ oc delete project <istio-system-project>
```