Red Hat OpenShift Service on AWS 4

Service Mesh

Service Mesh 的安装、使用和发行注记信息

Last Updated: 2025-11-29

Red Hat OpenShift Service on AWS 4 Service Mesh

Service Mesh 的安装、使用和发行注记信息

Abstract

本文档提供了有关如何在 Red Hat OpenShift Service on AWS 中使用 Service Mesh 的信息.

Table of Contents

第 1 章 SERVICE MESH 2.X

1.1. 关于 OPENSHIFT SERVICE MESH

注意

因为 Red Hat OpenShift Service Mesh 的发行节奏与 Red Hat OpenShift Service on AWS 不同，且 Red Hat OpenShift Service Mesh Operator 支持部署多个版本的 **ServiceMeshControlPlane**，所以 Service Mesh 没有为产品的次版本维护单独的文档。当前文档集适用于 Service Mesh 的最新版本，除非在特定主题或特定功能中明确指定了特定于版本的限制。

如需有关 Red Hat OpenShift Service Mesh 生命周期和支持的平台的更多信息，请参阅平台生命周期政策。

1.1.1. Red Hat OpenShift Service Mesh 简介

Red Hat OpenShift Service Mesh 通过在应用程序中创建集中控制点来解决微服务架构中的各种问题。它在现有分布式应用上添加一个透明层，而无需对应用代码进行任何更改。

微服务架构将企业应用的工作分成模块化服务，从而简化扩展和维护。但是，随着微服务架构上构建的企业应用的规模和复杂性不断增长，理解和管理变得困难。Service Mesh 可以通过捕获或截获服务间的流量来解决这些架构问题，并可修改、重定向或创建新请求到其他服务。

Service Mesh 基于开源 Istio 项目，为创建部署的服务提供发现、负载均衡、服务对服务身份验证、故障恢复、指标和监控的服务网络提供了便捷的方法。服务网格还提供更复杂的操作功能，其中包括 A/B 测试、canary 发行版本、访问控制以及端到端验证。

注意

Red Hat OpenShift Service Mesh 3 已正式发布。如需更多信息，请参阅 Red Hat OpenShift Service Mesh 3.0。

1.1.2. 核心功能

Red Hat OpenShift Service Mesh 在服务网络间提供了实现关键功能的统一方式：

- **流量管理** - 控制服务间的流量和 API 调用，提高调用的可靠性，并使网络在条件不好的情况保持稳定。
- **服务标识和安全性** - 在网格中提供可验证身份的服务，并提供保护服务流量的能力，以便可以通过信任度不同的网络进行传输。
- **策略强制** - 对服务间的交互应用机构策略，确保实施访问策略，并在用户间分配资源。通过配置网格就可以对策略进行更改，而不需要修改应用程序代码。
- **遥测** - 了解服务间的依赖关系以及服务间的网络数据流，从而可以快速发现问题。

1.2. SERVICE MESH 发行注记

1.2.1. Red Hat OpenShift Service Mesh 版本 2.6.7

此 Red Hat OpenShift Service Mesh 发行版本将 Red Hat OpenShift Service Mesh Operator 版本更新至 2.6.7，并包括以下 **ServiceMeshControlPlane** 资源版本更新： 2.6.7 和 2.5.10。

此发行版本解决了 CVE 报告的安全漏洞问题，并在 Red Hat OpenShift Service on AWS 4.14 及更新的版本中被支持。

Red Hat OpenShift Service Mesh Operator 的最新版本可与所有支持的 Service Mesh 版本一起使用。Service Mesh 的版本使用 **ServiceMeshControlPlane** 指定。

您可以使用红帽提供的 Kiali Operator 的最新版本以及所有支持的 Red Hat OpenShift Service Mesh 版本。Service Mesh 的版本使用 **ServiceMeshControlPlane** 资源指定。Service Mesh 的版本会自动确保 Kiali 的兼容版本。

1.2.1.1. 组件更新

组件	版本
Istio	1.20.8
Envoy Proxy	1.28.7
Kiali Server	1.73.20

1.2.2. Red Hat OpenShift Service Mesh 版本 2.5.10

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.7 中，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。此发行版本解决了 CVE 报告的安全漏洞问题。

1.2.2.1. 组件更新

组件	版本
Istio	1.18.7
Envoy Proxy	1.26.8
Kiali Server	1.73.20

1.2.3. Red Hat OpenShift Service Mesh 版本 2.6.6

此 Red Hat OpenShift Service Mesh 发行版本将 Red Hat OpenShift Service Mesh Operator 版本更新至 2.6.6，并包括以下 **ServiceMeshControlPlane** 资源版本更新： 2.6.6, 2.5.9 和 2.4.15。

此发行版本解决了 CVE 报告的安全漏洞问题，并在 Red Hat OpenShift Service on AWS 4.14 及更新的版本中被支持。

您可以使用红帽提供的 Kiali Operator 的最新版本以及所有支持的 Red Hat OpenShift Service Mesh 版本。Service Mesh 的版本使用 **ServiceMeshControlPlane** 资源指定。Service Mesh 的版本会自动确保 Kiali 的兼容版本。

1.2.3.1. 组件更新

组件	版本
Istio	1.20.8
Envoy Proxy	1.28.7
Kiali Server	1.73.19

1.2.3.2. 新功能

- 在这个版本中，Red Hat OpenShift Service Mesh 2.6 的 Operator 被重命名为 Red Hat OpenShift Service Mesh 2，以便与 Red Hat OpenShift Service Mesh 3.0 发行版本保持一致并改进清晰性。

1.2.4. Red Hat OpenShift Service Mesh 版本 2.5.9

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.6 中，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。此发行版本解决了 CVE 报告的安全漏洞问题。

1.2.4.1. 组件更新

组件	版本
Istio	1.18.7
Envoy Proxy	1.26.8
Kiali Server	1.73.19

1.2.5. Red Hat OpenShift Service Mesh 版本 2.4.15

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.6 中，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。

1.2.5.1. 组件更新

组件	版本
Istio	1.16.7
Envoy Proxy	1.24.12
Kiali Server	1.65.20

1.2.6. Red Hat OpenShift Service Mesh 版本 2.6.5

此 Red Hat OpenShift Service Mesh 发行版本将 Red Hat OpenShift Service Mesh Operator 版本更新至 2.6.5，并包括以下 **ServiceMeshControlPlane** 资源版本更新： 2.6.5, 2.5.8 和 2.4.14。

此发行版本解决了 CVE 报告的安全漏洞问题，并在 Red Hat OpenShift Service on AWS 4.14 及更新的版本中被支持。

您可以使用红帽提供的 Kiali Operator 的最新版本以及所有支持的 Red Hat OpenShift Service Mesh 版本。Service Mesh 的版本使用 **ServiceMeshControlPlane** 资源指定。Service Mesh 的版本会自动确保 Kiali 的兼容版本。

1.2.6.1. 组件更新

组件	版本
Istio	1.20.8
Envoy Proxy	1.28.7
Kiali Server	1.73.18

1.2.6.2. 新功能

- IBM Z 现在支持 Red Hat OpenShift distributed Tracing Platform Stack。

1.2.6.3. 修复的问题

- OSSM-8608 以前，在安装过程中终止 Container Network Interface (CNI) pod，同时复制二进制文件可能会在节点文件系统中保留 Istio-CNI 临时文件。重复发生可能会最终填满节点磁盘空间。现在，当在安装过程中终止 CNI pod 时，在复制 CNI 二进制文件前会删除现有临时文件，确保节点文件系统中只有一个临时文件。

1.2.7. Red Hat OpenShift Service Mesh 版本 2.5.8

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.5 中，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。此发行版本解决了 CVE 报告的安全漏洞问题。

1.2.7.1. 组件更新

组件	版本
Istio	1.18.7
Envoy Proxy	1.26.8
Kiali Server	1.73.18

1.2.7.2. 修复的问题

- OSSM-8608 以前，在安装过程中终止 Container Network Interface (CNI) pod，同时复制二进制文件可能会在节点文件系统中保留 Istio-CNI 临时文件。重复发生可能会最终填满节点磁盘空间。现在，当在安装过程中终止 CNI pod 时，在复制 CNI 二进制文件前会删除现有临时文件，确保节点文件系统中只有一个临时文件。

1.2.8. Red Hat OpenShift Service Mesh 版本 2.4.14

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.5 中，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。此发行版本解决了 CVE 报告的安全漏洞问题。

1.2.8.1. 组件更新

组件	版本
Istio	1.16.7
Envoy Proxy	1.24.12
Kiali Server	1.65.19

1.2.8.2. 修复的问题

- OSSM-8608 以前，在安装过程中终止 Container Network Interface (CNI) pod，同时复制二进制文件可能会在节点文件系统中保留 Istio-CNI 临时文件。重复发生可能会最终填满节点磁盘空间。现在，当在安装过程中终止 CNI pod 时，在复制 CNI 二进制文件前会删除现有临时文件，确保节点文件系统中只有一个临时文件。

1.2.9. Red Hat OpenShift Service Mesh 版本 2.6.4

此 Red Hat OpenShift Service Mesh 发行版本将 Red Hat OpenShift Service Mesh Operator 版本更新至 2.6.4，并包括以下 **ServiceMeshControlPlane** 资源版本更新： 2.6.4、2.5.7 和 2.4.13。

此发行版本解决了 CVE 报告的安全漏洞问题，并在 Red Hat OpenShift Service on AWS 4.14 及更新的版本中被支持。

红帽提供的 Kiali Operator 的最新版本可与所有支持的 Red Hat OpenShift Service Mesh 版本一起使用。Service Mesh 的版本使用 **ServiceMeshControlPlane** 资源指定。Service Mesh 的版本会自动确保 Kiali 的兼容版本。

1.2.9.1. 组件更新

组件	版本
Istio	1.20.8
Envoy Proxy	1.28.7

组件	版本
Kiali Server	1.73.17

1.2.10. Red Hat OpenShift Service Mesh 版本 2.5.7

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.4 中，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。此发行版本解决了 CVE 报告的安全漏洞问题。

1.2.10.1. 组件更新

组件	版本
Istio	1.18.7
Envoy Proxy	1.26.8
Kiali Server	1.73.17

1.2.11. Red Hat OpenShift Service Mesh version 2.4.13

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.4 中，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。此发行版本解决了 CVE 报告的安全漏洞问题。

1.2.11.1. 组件更新

组件	版本
Istio	1.16.7
Envoy Proxy	1.24.12
Kiali Server	1.65.18

1.2.12. Red Hat OpenShift Service Mesh version 2.6.3

此 Red Hat OpenShift Service Mesh 发行版本将 Red Hat OpenShift Service Mesh Operator 版本更新至 2.6.3，并包括以下 **ServiceMeshControlPlane** 资源版本更新： 2.6.3, 2.5.6 和 2.4.12。

此发行版本解决了 CVE 报告的安全漏洞问题，并在 Red Hat OpenShift Service on AWS 4.14 及更新的版本中被支持。

红帽提供的 Kiali Operator 的最新版本可与所有支持的 Red Hat OpenShift Service Mesh 版本一起使用。Service Mesh 的版本使用 **ServiceMeshControlPlane** 资源指定。Service Mesh 的版本会自动确保 Kiali 的兼容版本。

1.2.12.1. 组件更新

组件	版本
Istio	1.20.8
Envoy Proxy	1.28.7
Kiali Server	1.73.16

1.2.13. Red Hat OpenShift Service Mesh 版本 2.5.6

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.3 中，解决了 CVE 报告的安全漏洞问题，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。

1.2.13.1. 组件更新

组件	版本
Istio	1.18.7
Envoy Proxy	1.26.8
Kiali Server	1.73.16

1.2.14. Red Hat OpenShift Service Mesh version 2.4.12

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.3 中，解决了 CVE 报告的安全漏洞问题，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。

1.2.14.1. 组件更新

组件	版本
Istio	1.16.7
Envoy Proxy	1.24.12
Kiali Server	1.65.17

1.2.15. Red Hat OpenShift Service Mesh version 2.6.2

此 Red Hat OpenShift Service Mesh 发行版本将 Red Hat OpenShift Service Mesh Operator 版本更新至 2.6.2，并包括以下 **ServiceMeshControlPlane** 资源版本更新： 2.6.2, 2.5.5 和 2.4.11。

此发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。

红帽提供的 Kiali Operator 的最新版本可与所有支持的 Red Hat OpenShift Service Mesh 版本一起使用。Service Mesh 的版本使用 **ServiceMeshControlPlane** 资源指定。Service Mesh 的版本会自动确保 Kiali 的兼容版本。

1.2.15.1. 组件更新

组件	版本
Istio	1.20.8
Envoy Proxy	1.28.7
Kiali Server	1.73.15

1.2.15.2. 新功能

- IBM Power、IBM Z 和 IBM® LinuxONE 支持 Red Hat OpenShift 的 cert-manager Operator。

1.2.15.3. 修复的问题

- OSSM-8099 以前，当端点处于 draining 阶段时，支持持久性会话标签存在问题。现在，有处理有状态标头会话的排空端点的方法。
- OSSM-8001 以前，当 **runAsUser** 和 **runAsGroup** 在 pod 中被设置为相同的值时，代理 GID 会错误地设置为与容器的 GID 匹配，从而导致 Istio CNI 应用 iptables 规则的流量截获问题。现在，容器可以具有与 runAsUser 和 runAsGroup 的值相同，iptables 规则会正确应用。
- OSSM-8074 在以前的版本中，当 Service Mesh 有一个仅限数字的命名空间（如 **12345**）时，Kiali Operator 无法安装 Kiali 服务器。现在，只有 numerals 的命名空间可以正常工作。

1.2.16. Red Hat OpenShift Service Mesh version 2.5.5

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.2 中，解决了 CVE 报告的安全漏洞问题，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。

1.2.16.1. 组件更新

组件	版本
Istio	1.18.7
Envoy Proxy	1.26.8
Kiali Server	1.73.15

1.2.16.2. 修复的问题

- OSSM-8001 以前，当 **runAsUser** 和 **runAsGroup** 参数在 pod 中被设置为相同的值时，代理 GID 会错误地设置为与容器的 GID 匹配，从而导致 Istio CNI 应用 iptables 规则的流量截获问题。现在，容器可以具有与 **runAsUser** 和 **runAsGroup** 参数相同的值，iptables 规则会正确应用。
- OSSM-8074 以前，当 Service Mesh 有数字只限命名空间（如 **12345**）时，红帽提供的 Kiali Operator 无法安装 Kiali 服务器。现在，只有 numerals 的命名空间可以正常工作。

1.2.17. Red Hat OpenShift Service Mesh version 2.4.11

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.2 中，解决了 CVE 报告的安全漏洞问题，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。

1.2.17.1. 组件更新

组件	版本
Istio	1.16.7
Envoy Proxy	1.24.12
Kiali Server	1.65.16

1.2.17.2. 修复的问题

- OSSM-8001 以前，当 **runAsUser** 和 **runAsGroup** 参数在 pod 中被设置为相同的值时，代理 GID 会错误地设置为与容器的 GID 匹配，从而导致 Istio CNI 应用 iptables 规则的流量截获问题。现在，容器可以具有与 **runAsUser** 和 **runAsGroup** 参数相同的值，iptables 规则会正确应用。
- OSSM-8074 以前，当 Service Mesh 有数字只限命名空间（如 **12345**）时，红帽提供的 Kiali Operator 无法安装 Kiali 服务器。现在，只有 numerals 的命名空间可以正常工作。

1.2.18. Red Hat OpenShift Service Mesh version 2.6.1

此 Red Hat OpenShift Service Mesh 发行版本将 Red Hat OpenShift Service Mesh Operator 版本更新至 2.6.1，并包括以下 **ServiceMeshControlPlane** 资源版本更新： 2.6.1, 2.5.4 和 2.4.10。此发行版本解决了 CVE 报告的安全漏洞问题(CVE)，它包含一个程序错误修复，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。

红帽提供的 Kiali Operator 的最新版本可与所有支持的 Red Hat OpenShift Service Mesh 版本一起使用。Service Mesh 的版本使用 **ServiceMeshControlPlane** 资源指定。Service Mesh 的版本会自动确保 Kiali 的兼容版本。

1.2.18.1. 组件更新

组件	版本
Istio	1.20.8

组件	版本
Envoy Proxy	1.28.5
Kiali Server	1.73.14

1.2.18.2. 修复的问题

- OSSM-6766 以前，如果用户想更新命名空间（如启用或禁用注入）或创建任何 Istio 对象（例如，创建流量策略），则 OpenShift Service Mesh Console (OSSMC)插件会失败。现在，如果用户更新命名空间或创建任何 Istio 对象，OpenShift Service Mesh Console (OSSMC)插件不会失败。

1.2.19. Red Hat OpenShift Service Mesh 版本 2.5.4

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.1 中，解决了 CVE 报告的安全漏洞问题，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。

1.2.19.1. 组件更新

组件	版本
Istio	1.18.7
Envoy Proxy	1.26.8
Kiali Server	1.73.14

1.2.20. Red Hat OpenShift Service Mesh version 2.4.10

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.1 中，解决了 CVE 报告的安全漏洞问题，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。

1.2.20.1. 组件更新

组件	版本
Istio	1.16.7
Envoy Proxy	1.24.12
Kiali Server	1.65.15

1.2.21. Red Hat OpenShift Service Mesh version 2.6.0

此 Red Hat OpenShift Service Mesh 发行版本将 Red Hat OpenShift Service Mesh Operator 版本更新至 2.6.0，并包括以下 **ServiceMeshControlPlane** 资源版本更新： 2.6.0, 2.5.3 和 2.4.9。此发行版本添加了新功能，解决了 CVE 报告的安全漏洞问题(CVE)，并受 Red Hat OpenShift Service on AWS 4.14 及更新版本的支持。

此发行版本结束了对 Red Hat OpenShift Service Mesh 版本 2.3 的维护支持。如果使用 Service Mesh 版本 2.3，您应该更新到一个受支持的版本。

重要

Red Hat OpenShift Service Mesh 为 FIPS 设计。Service Mesh 使用已提交到 NIST for FIPS 140-2/140-3 Validation 的 RHEL 加密库，用于 x86_64、ppc64le 和 s390x 架构。有关 NIST 验证程序的更多信息，请参阅加密模块验证程序。有关已提交用于验证的 RHEL 加密库的最新 NIST 状态，请参阅 Compliance Activities 和 Government Standards。

1.2.21.1. 组件更新

组件	版本
Istio	1.20.8
Envoy Proxy	1.28.5
Kiali	1.73.9

1.2.21.2. Istio 1.20 支持

Service Mesh 2.6 基于 Istio 1.20，它提供新功能和产品改进，其中包括：

- Red Hat OpenShift Service on AWS 4.16 或更高版本支持原生 sidecar。

  ServiceMeshControlPlane 资源示例

  ```
  apiVersion: maistra.io/v2
  kind: ServiceMeshControlPlane
  metadata:
    name: basic
  spec:
    runtime:
      components:
        pilot:
          container:
            env:
              ENABLE_NATIVE_SIDECARS: "true"
  ```

- Istio 1.20 中的流量镜像现在支持多个目的地。此功能支持将流量镜像到不同的端点，允许在不同服务版本或配置间同时观察。

虽然 Red Hat OpenShift Service Mesh 支持许多 Istio 1.20 功能，但请注意以下例外：

- 不支持 ambient mesh

- 不支持 Istio 中的 QuickAssist Technology (QAT) PrivateKeyProvider

1.2.21.3. Istio 和 Kiali 捆绑包镜像名称更改

此发行版本更新了 Istio 捆绑包镜像名称和 Kiali 捆绑包镜像名称，以更好地与红帽命名约定保持一致。

- Istio 捆绑包镜像名称： **openshift-service-mesh/istio-operator-bundle**
- Kiali bundle image name: **openshift-service-mesh/kiali-operator-bundle**

1.2.21.4. 与 Red Hat OpenShift distributed Tracing Platform 和红帽构建的 OpenTelemetry 集成

此发行版本引入了对追踪扩展供应商的通用集成，Red Hat OpenShift distributed Tracing Platform 和 Red Hat build of OpenTelemetry。

您可以通过将指定元素和 **opentelemetry** 供应商附加到 **ServiceMehControlPlane** 资源中的 **spec.meshConfig.extensionProviders** 规格中，向 Red Hat OpenShift distributed Tracing Platform 公开追踪数据。然后，遥测自定义资源将 Istio 代理配置为收集 trace span，并将其发送到 OpenTelemetry Collector 端点。

您可以在网格命名空间中创建红帽构建的 OpenTelemetry 实例，并将其配置为将追踪数据发送到追踪平台后端服务。

1.2.21.5. Red Hat OpenShift distributed Tracing Platform (Jaeger)默认设置更改

此发行版本默认为 **ServiceMeshControlPlane** 资源的新实例禁用 Red Hat OpenShift Distributed Tracing Platform (Jaeger)。

将 **ServiceMeshControlPlane** 资源的现有实例更新至 Red Hat OpenShift Service Mesh 版本 2.6 时，分布式 Tracing Platform (Jaeger)会被默认启用。

Red Hat OpenShift Service Mesh 2.6 是最后一个发行版本，包括对 Red Hat OpenShift distributed Tracing Platform (Jaeger)和 OpenShift Elasticsearch Operator 的支持。分布式追踪平台(Jaeger)和 OpenShift Elasticsearch Operator 将在以后的版本中删除。目前使用分布式跟踪平台(Jaeger)和 OpenShift Elasticsearch Operator，则需要切换到 Red Hat OpenShift distributed Tracing Platform 和 Red Hat build of OpenTelemetry。

1.2.21.6. 网关 API 使用通常可用于 Red Hat OpenShift Service Mesh 集群范围的部署

此发行版本引入了在 Red Hat OpenShift Service Mesh 2.6 中使用 Kubernetes 网关 API 版本 1.0.0 的 General Availability。此 API 使用仅限于 Red Hat OpenShift Service Mesh。不支持 Gateway API 自定义资源定义(CRD)。

如果启用了集群范围的模式，网关 API 现在会被默认启用(**spec.mode: ClusterWide**)。即使集群中没有安装自定义资源定义(CRD)，也可以启用它。

> **重要**
>
> 用于多租户网格部署的网关 API 仍只是一个技术预览。

请参阅下表来确定应使用的 OpenShift Service Mesh 版本安装哪个网关 API 版本：

Service Mesh 版本	Istio 版本	网关 API 版本	备注
2.6	1.20.x	1.0.0	N/A
2.5.x	1.18.x	0.6.2	使用实验性分支，因为 v0.6.2 中缺少 **ReferenceGrand**。
2.4.x	1.16.x	0.5.1	对于多租户网格部署，所有网关 API CRD 都必须存在。使用实验性分支。

您可以通过将 **PILOT_ENABLE_GATEWAY_API** 设置为 **false** 来禁用此功能：

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  runtime:
    components:
      pilot:
        container:
          env:
            PILOT_ENABLE_GATEWAY_API: "false"
```

1.2.21.7. 修复的问题

- OSSM-6754 在以前的版本中，在 Red Hat OpenShift Service on AWS 4.15 中，当用户进入到 **Service 详情页面**时，点 **Service Mesh** 选项卡并刷新页面，**Service Mesh 详情页面**仍然处于 Service Mesh 内容信息，即使 active 选项卡是默认的 **Details** 选项卡。现在，在刷新后，用户可以浏览服务详情页面的不同标签页，而无需问题。

- **OSSM-2101 在以前的版本中，Istio Operator 在不再需要时不会删除istio-cni-node DaemonSet 和其他 CNI 资源。现在，在升级 Operator 后，如果集群中至少安装了一个 SMCP，Operator 会协调这个 SMCP，然后删除所有未使用的 CNI 安装（甚至非常旧的 CNI 版本为 v2.0）。**

1.2.21.8. Kiali 已知问题

- **OSSM-6099 在 IPv6 集群上安装 OpenShift Service Mesh 控制台 (OSSMC) 插件失败。临时解决方案：在 IPv4 集群上安装 OSSMC 插件。**

1.2.22. Red Hat OpenShift Service Mesh 版本 2.5.3

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.0 中，解决了 CVE 报告的安全漏洞问题，并受 Red Hat OpenShift Service on AWS 4.12 及更新的版本的支持。

1.2.22.1. 组件更新

组件	版本
Istio	1.18.7
Envoy Proxy	1.26.8
Kiali	1.73.9

1.2.23. Red Hat OpenShift Service Mesh version 2.4.9

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.6.0 中，解决了 CVE 报告的安全漏洞问题，并受 Red Hat OpenShift Service on AWS 4.12 及更新的版本的支持。

1.2.23.1. 组件更新

组件	版本
Istio	1.16.7
Envoy Proxy	1.24.12
Kiali	1.65.11

1.2.24. Red Hat OpenShift Service Mesh version 2.5.2

此 Red Hat OpenShift Service Mesh 发行版本将 Red Hat OpenShift Service Mesh Operator 版本更新至 2.5.2，并包括以下 **ServiceMeshControlPlane** 资源版本更新： 2.5.2、2.4.8 和 2.3.12。此发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.12 及更新的版本的支持。

1.2.24.1. 组件更新

组件	版本
Istio	1.18.7
Envoy Proxy	1.26.8
Kiali	1.73.8

1.2.24.2. 修复的问题

- OSSM-6331 以前，**smcp.general.logging.componentLevels** spec 接受无效的 **LogLevel** 值，**ServiceMeshControlPlane** 资源仍然被创建。现在，当使用无效的值且没有创建 control plane 时，终端会显示一个错误消息。

- OSSM-6290 以前，Istio Config 列表页面的 Project 过滤器下拉列表无法正常工作。所有 **istio config** 项都会从所有命名空间中显示，即使您从下拉菜单中选择一个特定的项目。现在，只会显示属于过滤器下拉菜单中所选项目的 **istio** 配置项。
- OSSM-6298 以前，当您点击 OpenShift Service Mesh 控制台(OSSMC)插件中的项目引用时，控制台有时会在打开所需页面前执行多个重定向。因此，导航到在控制台中打开的上一页会导致您的 Web 浏览器打开错误页面。现在，这些重定向不会发生，在网页浏览器中点 Back 会打开正确的页面。
- OSSM-6299 以前，在 Red Hat OpenShift Service on AWS 4.15 中，当您点击流量图中任何节点菜单的 Node 图形 菜单选项时，不会显示节点图。相反，页面使用相同的流量图刷新。现在，点 Node 图形 菜单选项可以正确地显示节点图。
- OSSM-6267 以前，在 Red Hat OpenShift Service Mesh 2.5 Grafana 中配置数据源会导致数据查询身份验证错误，用户无法在 Istio 服务和工作负载仪表板中查看数据。现在，将现有 2.5 SMCP 升级到 2.5.2 或更高版本可解决 Grafana 错误。

1.2.25. Red Hat OpenShift Service Mesh 版本 2.4.8

此 Red Hat OpenShift Service Mesh 发行版本包含在 Red Hat OpenShift Service Mesh Operator 2.5.2 中，解决了 CVE 报告的安全漏洞问题，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.12 及更新的版本的支持。

Red Hat OpenShift Service Mesh Operator 的最新版本可与所有支持的 Service Mesh 版本一起使用。Service Mesh 的版本使用 **ServiceMeshControlPlane** 指定。

1.2.25.1. 组件更新

组件	版本
Istio	1.16.7
Envoy Proxy	1.24.12
Kiali	1.65.11

1.2.26. Red Hat OpenShift Service Mesh version 2.3.12

此 Red Hat OpenShift Service Mesh 发行版本包含在 Red Hat OpenShift Service Mesh Operator 2.5.2 中，解决了 CVE 报告的安全漏洞问题，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.12 及更新的版本的支持。

Red Hat OpenShift Service Mesh Operator 的最新版本可与所有支持的 Service Mesh 版本一起使用。Service Mesh 的版本使用 **ServiceMeshControlPlane** 资源指定。

1.2.26.1. 组件更新

组件	版本
Istio	1.14.5

组件	版本
Envoy Proxy	1.22.11
Kiali	1.57.14

1.2.27. 之前的版本

之前的版本添加了功能和改进。

1.2.27.1. Red Hat OpenShift Service Mesh 版本 2.5.1 的新功能

此 Red Hat OpenShift Service Mesh 发行版本将 Red Hat OpenShift Service Mesh Operator 版本更新至 2.5.1，并包括以下 **ServiceMeshControlPlane** 资源版本更新： 2.5.1、2.4.7 和 2.3.11。

此发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.12 及更新的版本的支持。

1.2.27.1.1. Red Hat OpenShift Service Mesh 2.5.1 的组件版本

组件	版本
Istio	1.18.7
Envoy Proxy	1.26.8
Kiali	1.73.7

1.2.27.2. Red Hat OpenShift Service Mesh 版本 2.5 的新功能

此 Red Hat OpenShift Service Mesh 发行版本将 Red Hat OpenShift Service Mesh Operator 版本更新至 2.5.0，并包括以下 **ServiceMeshControlPlane** 资源版本更新： 2.5.0, 2.4.6 和 2.3.10。

此发行版本添加了新功能，解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.12 及之后的版本的支持。

此发行版本结束了对 OpenShift Service Mesh 版本 2.2 的维护支持。如果使用 OpenShift Service Mesh 版本 2.2，您应该升级到受支持的版本。

1.2.27.2.1. Red Hat OpenShift Service Mesh 2.5 的组件版本

组件	版本
Istio	1.18.7
Envoy Proxy	1.26.8
Kiali	1.73.4

1.2.27.2.2. Istio 1.18 支持

Service Mesh 2.5 基于 Istio 1.18，它引入了新功能和产品改进。虽然 Red Hat OpenShift Service Mesh 支持许多 Istio 1.18 功能，但请注意以下例外：

- 不支持 ambient mesh
- 不支持 Istio 中的 QuickAssist Technology (QAT) PrivateKeyProvider

1.2.27.2.3. Cluster-Wide 网格迁移

此发行版本添加了从多租户网格迁移到集群范围代理的文档。如需更多信息，请参阅以下文档：

- "关于迁移到集群范围的网格"
- "从集群范围的网格中排除命名空间"
- "定义哪些命名空间在集群范围的网格中接收 sidecar 注入"
- "不包括集群范围的网格中的单个 pod"

1.2.27.2.4. 基于 ARM 的集群中的 Red Hat OpenShift Service Mesh Operator

此发行版本在基于 ARM 的集群上提供了 Red Hat OpenShift Service Mesh Operator，作为正式发布的功能。

1.2.27.2.5. 与 Red Hat OpenShift distributed Tracing Platform (Tempo) Stack 集成

此发行版本引入了对追踪扩展供应商的通用集成。您可以通过将命名元素和 **zipkin** 供应商附加到 **spec.meshConfig.extensionProviders** 规格中，将追踪数据公开给 Red Hat OpenShift distributed Tracing Platform (Tempo)堆栈。然后，遥测自定义资源将 Istio 代理配置为收集 trace 并将其发送到 Tempo 经销商服务端点。

> **注意**
>
> IBM Z 不支持 Red Hat OpenShift distributed Tracing Platform (Tempo) Stack。

1.2.27.2.6. OpenShift Service Mesh 控制台插件

此发行版本引入了 OpenShift Service Mesh 控制台(OSSMC)插件的通用版本。

OSSMC 插件是 OpenShift 控制台的扩展，可为您的 Service Mesh 提供可见性。安装 OSSMC 插件后，web 控制台的导航窗格中提供了一个新的 Service Mesh 菜单选项，以及增强现有 Workloads 和 Service 控制台页面的新 Service Mesh 标签页。

OSSMC 插件的功能与独立 Kiali 控制台的功能非常相似。OSSMC 插件不会替换 Kiali 控制台，在安装 OSSMC 插件后，您仍可以访问独立 Kiali 控制台。

1.2.27.2.7. Istio OpenShift 路由(IOR)默认设置更改

Istio OpenShift 路由(IOR)的默认设置已更改。从这个版本开始，对于 **ServiceMeshControlPlane** 资源的新实例，默认禁用自动路由。

对于 **ServiceMeshControlPlane** 资源的新实例，您可以通过在 **ServiceMeshControlPlane** 资源的 gateway **.openshiftRoute** 规格中将 **enabled** 字段设置为 **true** 来使用自动路由。

ServiceMeshControlPlane 资源示例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  gateways:
    openshiftRoute:
      enabled: true
```

将 ServiceMeshControlPlane 资源的现有实例更新至 Red Hat OpenShift Service Mesh 版本 2.5 时，自动路由会默认启用。

1.2.27.2.8. Istio 代理并发配置增强

networking.istio API 中的 **concurrency** 参数配置 Istio 代理运行的 worker 线程数量。

为了在部署间一致性，Istio 现在根据分配给代理容器的 CPU 限制配置 **concurrency** 参数。例如，限制 2500m 会将 **concurrency** 参数设置为 **3**。如果将 **concurrency** 参数设置为不同的值，则 Istio 会使用该值来配置代理运行的线程数量，而不是使用 CPU 限制。

在以前的版本中，参数的默认设置是 **2**。

1.2.27.2.9. 网关 API CRD 版本

重要

Red Hat OpenShift Service on AWS Gateway API 支持只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

现在提供了网关 API 自定义资源定义(CRD)的新版本。请参阅下表来确定应使用的 OpenShift Service Mesh 版本安装哪个网关 API 版本：

Service Mesh 版本	Istio 版本	网关 API 版本	备注
2.5.x	1.18.x	0.6.2	使用实验性分支，因为 v0.6.2 缺少 **ReferenceGrand**
2.4.x	1.16.x	0.5.1	对于多租户网格部署，所有网关 API CRD 都必须存在。使用实验性分支。

1.2.27.3. Red Hat OpenShift Service Mesh 版本 2.4.7 的新功能

此 Red Hat OpenShift Service Mesh 发行版本包含在 Red Hat OpenShift Service Mesh Operator 2.5.1 中，解决了 CVE 报告的安全漏洞问题，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.12 及更新的版本的支持。

1.2.27.3.1. Red Hat OpenShift Service Mesh 2.4.7 的组件版本

组件	版本
Istio	1.16.7
Envoy Proxy	1.24.12
Kiali	1.65.11

1.2.27.4. Red Hat OpenShift Service Mesh 版本 2.4.6 的新功能

此 Red Hat OpenShift Service Mesh 发行版本包含在 Red Hat OpenShift Service Mesh Operator 2.5.0 中，解决了 CVE 报告的安全漏洞问题，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.12 及更新的版本的支持。

1.2.27.4.1. Red Hat OpenShift Service Mesh 2.4.6 的组件版本

组件	版本
Istio	1.16.7
Envoy Proxy	1.24.12
Kiali	1.65.11

1.2.27.5. Red Hat OpenShift Service Mesh 版本 2.4.5 的新功能

此 Red Hat OpenShift Service Mesh 发行版本将 Red Hat OpenShift Service Mesh Operator 版本更新至 2.4.5，并包括以下 **ServiceMeshControlPlane** 资源版本更新： 2.4.5、2.3.9 和 2.2.12。

此发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.11 及之后的版本的支持。

1.2.27.5.1. Red Hat OpenShift Service Mesh 2.4.5 版中包含的组件版本

组件	版本
Istio	1.16.7
Envoy Proxy	1.24.12
Kiali	1.65.11

1.2.27.6. Red Hat OpenShift Service Mesh 版本 2.4.4 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.11 及更新的版本的支持。

1.2.27.6.1. Red Hat OpenShift Service Mesh 2.4.4 版中包含的组件版本

组件	版本
Istio	1.16.7
Envoy Proxy	1.24.12
Jaeger	1.47.0
Kiali	1.65.10

1.2.27.7. Red Hat OpenShift Service Mesh 版本 2.4.3 的新功能

- 添加了 **envoyExtAuthzGrpc** 字段，该字段用于使用 gRPC API 配置外部授权提供程序。
- 解决了常见的漏洞和风险(CVE)。
- 此发行版本在 Red Hat OpenShift Service on AWS 4.10 及更新的版本中被支持。

1.2.27.7.1. Red Hat OpenShift Service Mesh 2.4.3 版中包含的组件版本

组件	版本
Istio	1.16.7
Envoy Proxy	1.24.10
Jaeger	1.42.0
Kiali	1.65.8

1.2.27.7.2. 远程过程调用(gRPC) API 支持外部授权配置

此功能增强添加了 **envoyExtAuthzGrpc** 字段，以使用 gRPC API 配置外部授权供应商。

1.2.27.8. Red Hat OpenShift Service Mesh 版本 2.4.2 的新功能

此 Red Hat OpenShift Service Mesh 发行版本引入了新功能，解决了 CVE 报告的安全漏洞问题 (CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.10 和更高版本的支持。

1.2.27.8.1. Red Hat OpenShift Service Mesh 2.4.2 版中包含的组件版本

组件	版本
Istio	1.16.7

组件	版本
Envoy Proxy	1.24.10
Jaeger	1.42.0
Kiali	1.65.7

1.2.27.9. Red Hat OpenShift Service Mesh 版本 2.4.1 的新功能

此 Red Hat OpenShift Service Mesh 发行版本引入了新功能，解决了 CVE 报告的安全漏洞问题 (CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.10 和更高版本的支持。

1.2.27.9.1. Red Hat OpenShift Service Mesh 2.4.1 版中包含的组件版本

组件	版本
Istio	1.16.5
Envoy Proxy	1.24.8
Jaeger	1.42.0
Kiali	1.65.7

1.2.27.10. Red Hat OpenShift Service Mesh 版本 2.4 的新功能

此 Red Hat OpenShift Service Mesh 发行版本引入了新功能，解决了 CVE 报告的安全漏洞问题 (CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.10 和更高版本的支持。

1.2.27.10.1. Red Hat OpenShift Service Mesh 2.4 版中包含的组件版本

组件	版本
Istio	1.16.5
Envoy Proxy	1.24.8
Jaeger	1.42.0
Kiali	1.65.6

1.2.27.10.2. 集群范围的部署

此功能增强引入了集群范围的部署的通用版本。集群范围的部署包含一个服务网格 control plane，用于监控整个集群的资源。control plane 在所有命名空间中使用单个查询来监控影响网格配置的每个 Istio 或 Kubernetes 资源。减少集群范围的部署中 control plane 执行的查询数量可提高性能。

1.2.27.10.3. 支持发现选择器

此功能增强引入了一个通用的 **meshConfig.discoverySelectors** 字段版本，该字段可用于集群范围的部署来限制服务网格 control plane 可以发现的服务。

```
spec:
  meshConfig
    discoverySelectors:
    - matchLabels:
        env: prod
        region: us-east1
    - matchExpressions:
      - key: app
        operator: In
        values:
          - cassandra
          - spark
```

1.2.27.10.4. 与 cert-manager istio-csr 集成

在这个版本中，Red Hat OpenShift Service Mesh 与 **cert-manager** 控制器和 **istio-csr** 代理集成。**cert-manager** 在 Kubernetes 集群中将证书和证书签发者添加为资源类型，并简化了获取、续订和使用这些证书的过程。**cert-manager** 为 Istio 提供并轮转中间 CA 证书。与**istio-csr** 集成可让用户将 Istio 代理的签名证书请求委派给 **cert-manager**。**ServiceMeshControlPlane** v2.4 接受 **cert-manager** 提供的 CA 证书作为 **cacerts** secret。

注意

IBM Power®、IBM Z® 和 IBM® LinuxONE 不支持与 **cert-manager** 和 **istio-csr** 集成。

1.2.27.10.5. 与外部授权系统集成

此功能增强引入了一种通用可用的方法，使用 **AuthorizationPolicy** 资源的 **action: CUSTOM** 字段将 Red Hat OpenShift Service Mesh 与外部授权系统集成。使用 **envoyExtAuthzHttp** 字段将访问控制委派给外部授权系统。

1.2.27.10.6. 与外部 Prometheus 安装集成

此功能增强引入了 Prometheus 扩展供应商的通用版本。您可以通过在 **spec.meshConfig** 规格中将 **extensionProviders** 字段的值设置为 **prometheus**，将指标公开给 Red Hat OpenShift Service on AWS 监控堆栈或自定义 Prometheus 安装。Telemetry 对象配置 Istio 代理来收集流量指标。Service Mesh 只支持 Prometheus 指标的 Telemetry API。

```
spec:
  meshConfig:
    extensionProviders:
    - name: prometheus
      prometheus: {}
---
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: enable-prometheus-metrics
spec:
```

```
metrics:
- providers:
  - name: prometheus
```

1.2.27.10.7. 单堆栈 IPv6 支持

此功能增强引进了对单堆栈 IPv6 集群的常用支持，提供对更广泛的 IP 地址的访问。不支持双堆栈 IPv4 或 IPv6 集群。

注意

IBM Power®、IBM Z® 和 IBM® LinuxONE 不提供单一堆栈 IPv6 支持。

1.2.27.10.7.1. 在 AWS 网关 API 上启用 Red Hat OpenShift Service

要在 AWS 网关 API 上启用 Red Hat OpenShift Service，在 **ServiceMeshControlPlane** 资源的 **techPreview.gatewayAPI** 规格中将 **enabled** 字段的值设置为 **true**。

```
spec:
  techPreview:
    gatewayAPI:
      enabled: true
```

在以前的版本中，环境变量用于启用网关 API。

```
spec:
  runtime:
    components:
      pilot:
        container:
          env:
            PILOT_ENABLE_GATEWAY_API: "true"
            PILOT_ENABLE_GATEWAY_API_STATUS: "true"
            PILOT_ENABLE_GATEWAY_API_DEPLOYMENT_CONTROLLER: "true"
```

1.2.27.10.8. Istio 1.16 支持

Service Mesh 2.4 基于 Istio 1.16，它引入了新功能和产品改进。虽然很多 Istio 1.16 功能被支持，但请注意以下例外：

- 对于 sidecar 的 HBONE 协议是一个实验性功能，它不被支持。
- 不支持 ARM64 架构上的 Service Mesh。

1.2.27.11. Red Hat OpenShift Service Mesh 版本 2.3.11 的新功能

此 Red Hat OpenShift Service Mesh 发行版本包含在 Red Hat OpenShift Service Mesh Operator 2.5.1 中，解决了 CVE 报告的安全漏洞问题，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.12 及更新的版本的支持。

1.2.27.11.1. Red Hat OpenShift Service Mesh 2.3.11 的组件版本

组件	版本
Istio	1.14.5
Envoy Proxy	1.22.11
Kiali	1.57.14

1.2.27.12. Red Hat OpenShift Service Mesh 版本 2.3.10 的新功能

此 Red Hat OpenShift Service Mesh 发行版本包含在 Red Hat OpenShift Service Mesh Operator 2.5.0 中，解决了 CVE 报告的安全漏洞问题，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.12 及更新的版本的支持。

1.2.27.12.1. Red Hat OpenShift Service Mesh 2.3.10 的组件版本

组件	版本
Istio	1.14.5
Envoy Proxy	1.22.11
Kiali	1.57.14

1.2.27.13. Red Hat OpenShift Service Mesh 2.3.9 的新功能

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.4.5 中，解决了 CVE 报告的安全漏洞问题，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.11 及之后的版本的支持。

1.2.27.13.1. Red Hat OpenShift Service Mesh 2.3.9 版中包含的组件版本

组件	版本
Istio	1.14.5
Envoy Proxy	1.22.11
Jaeger	1.47.0
Kiali	1.57.14

1.2.27.14. Red Hat OpenShift Service Mesh 版本 2.3.8 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.11 及更新的版本的支持。

1.2.27.14.1. Red Hat OpenShift Service Mesh 2.3.8 版中包含的组件版本

组件	版本
Istio	1.14.5
Envoy Proxy	1.22.11
Jaeger	1.47.0
Kiali	1.57.13

1.2.27.15. Red Hat OpenShift Service Mesh 版本 2.3.7 的新功能

此 Red Hat OpenShift Service Mesh 发行版本引入了新功能，解决了 CVE 报告的安全漏洞问题 (CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.10 和更高版本的支持。

1.2.27.15.1. Red Hat OpenShift Service Mesh 2.3.7 版中包含的组件版本

组件	版本
Istio	1.14.6
Envoy Proxy	1.22.11
Jaeger	1.42.0
Kiali	1.57.11

1.2.27.16. Red Hat OpenShift Service Mesh 版本 2.3.6 的新功能

此 Red Hat OpenShift Service Mesh 发行版本引入了新功能，解决了 CVE 报告的安全漏洞问题 (CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.10 和更高版本的支持。

1.2.27.16.1. Red Hat OpenShift Service Mesh 2.3.6 版中包含的组件版本

组件	版本
Istio	1.14.5
Envoy Proxy	1.22.11
Jaeger	1.42.0
Kiali	1.57.10

1.2.27.17. Red Hat OpenShift Service Mesh 2.3.5 的新功能

此 Red Hat OpenShift Service Mesh 发行版本引入了新功能，解决了 CVE 报告的安全漏洞问题 (CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.10 和更高版本的支持。

1.2.27.17.1. Red Hat OpenShift Service Mesh 2.3.5 版中包含的组件版本

组件	版本
Istio	1.14.5
Envoy Proxy	1.22.9
Jaeger	1.42.0
Kiali	1.57.10

1.2.27.18. Red Hat OpenShift Service Mesh 版本 2.3.4 的新功能

此 Red Hat OpenShift Service Mesh 发行版本引入了新功能，解决了 CVE 报告的安全漏洞问题 (CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.10 和更高版本的支持。

1.2.27.18.1. Red Hat OpenShift Service Mesh 2.3.4 版中包含的组件版本

组件	版本
Istio	1.14.6
Envoy Proxy	1.22.9
Jaeger	1.42.0
Kiali	1.57.9

1.2.27.19. Red Hat OpenShift Service Mesh 版本 2.3.3 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.9 及更新的版本的支持。

1.2.27.19.1. Red Hat OpenShift Service Mesh 2.3.3 版中包含的组件版本

组件	版本
Istio	1.14.5
Envoy Proxy	1.22.9

组件	版本
Jaeger	1.42.0
Kiali	1.57.7

1.2.27.20. Red Hat OpenShift Service Mesh 版本 2.3.2 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.9 及更新的版本的支持。

1.2.27.20.1. Red Hat OpenShift Service Mesh 2.3.2 版中包含的组件版本

组件	版本
Istio	1.14.5
Envoy Proxy	1.22.7
Jaeger	1.39
Kiali	1.57.6

1.2.27.21. Red Hat OpenShift Service Mesh 版本 2.3.1 的新功能

此 Red Hat OpenShift Service Mesh 发行版本引入了新功能，解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.9 及更新的版本的支持。

1.2.27.21.1. Red Hat OpenShift Service Mesh 2.3.1 版中包含的组件版本

组件	版本
Istio	1.14.5
Envoy Proxy	1.22.4
Jaeger	1.39
Kiali	1.57.5

1.2.27.22. New features Red Hat OpenShift Service Mesh version 2.3

此 Red Hat OpenShift Service Mesh 发行版本引入了新功能，解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.9 及更新的版本的支持。

1.2.27.22.1. Red Hat OpenShift Service Mesh 2.3 版中包含的组件版本

组件	版本
Istio	1.14.3
Envoy Proxy	1.22.4
Jaeger	1.38
Kiali	1.57.3

1.2.27.22.2. 新的 Container Network Interface (CNI) DaemonSet 容器和 ConfigMap

openshift-operators 命名空间包括一个新的 istio CNI DaemonSet**istio-cni-node-v2-3** 和一个新的 **ConfigMap** 资源 **istio-cni-config-v2-3**。

当升级到 Service Mesh Control Plane 2.3 时，现有的 **istio-cni-node** DaemonSet 不会被更改，并创建一个新的 **istio-cni-node-v2-3** DaemonSet。

这个名称更改不会影响以前版本，或与之前版本部署的 Service Mesh Control Plane 关联的 **istio-cni-node** CNI DaemonSet。

1.2.27.22.3. 网关注入支持

此发行版本引进了对网关注入的正式支持。网关配置应用到在网格边缘运行的独立 Envoy 代理，而不是与您的服务工作负载一同运行的 sidecar Envoy 代理。这可让自定义网关选项。使用网关注入时，您必须在要运行网关代理的命名空间中创建以下资源： **Service**、**Deployment**、**Role** 和 **RoleBinding**。

1.2.27.22.4. Istio 1.14 支持

Service Mesh 2.3 基于 Istio 1.14，这会带来新功能和产品改进。虽然支持很多 Istio 1.14 功能，但需要注意以下例外：

- ProxyConfig API 支持在 image 字段外。
- SPIRE 运行时不是受支持的功能。

1.2.27.22.4.1. 配置集群范围部署

以下示例 **ServiceMeshControlPlane** 对象配置集群范围的部署。

要为集群范围的部署创建 SMCP，用户必须属于 **cluster-admin** ClusterRole。如果为集群范围的部署配置了 SMCP，则它必须是集群中唯一的 SMCP。您无法将 control plane 模式从多租户改为集群范围的（或从集群范围改为多租户）。如果多租户 control plane 已存在，请删除它并创建一个新 control plane。

这个示例为集群范围的部署配置 SMCP。

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: cluster-wide
  namespace: istio-system
```

```
spec:
  version: v2.3
  techPreview:
    controlPlaneMode: ClusterScoped 1
```

1 启用 Istiod 来监控集群一级的资源，而不是监控每个命名空间。

另外，还必须为集群范围的部署配置 SMMR。这个示例为集群范围的部署配置了 SMMR。

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
spec:
  members:
  - '*' 1
```

1 在网格中添加所有命名空间，包括您随后创建的任何命名空间。以下命名空间不是网格的一部分：kube、openshift、kube-* 和 openshift-*。

1.2.27.23. Red Hat OpenShift Service Mesh 版本 2.2.12 的新功能

此 Red Hat OpenShift Service Mesh 发行版本包括在 Red Hat OpenShift Service Mesh Operator 2.4.5 中，解决了 CVE 报告的安全漏洞问题，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.11 及之后的版本的支持。

1.2.27.23.1. Red Hat OpenShift Service Mesh 2.2.12 版中包含的组件版本

组件	版本
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.47.0
Kiali	1.48.11

1.2.27.24. Red Hat OpenShift Service Mesh 版本 2.2.11 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.11 及更新的版本的支持。

1.2.27.24.1. Red Hat OpenShift Service Mesh 2.2.11 版中包含的组件版本

组件	版本
Istio	1.12.9

组件	版本
Envoy Proxy	1.20.8
Jaeger	1.47.0
Kiali	1.48.10

1.2.27.25. Red Hat OpenShift Service Mesh 版本 2.2.10 的新功能

此 Red Hat OpenShift Service Mesh 发行版本引入了新功能，解决了 CVE 报告的安全漏洞问题 (CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.10 和更高版本的支持。

1.2.27.25.1. Red Hat OpenShift Service Mesh 2.2.10 版中包含的组件版本

组件	版本
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.42.0
Kiali	1.48.8

1.2.27.26. Red Hat OpenShift Service Mesh 版本 2.2.9 的新功能

此 Red Hat OpenShift Service Mesh 发行版本引入了新功能，解决了 CVE 报告的安全漏洞问题 (CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.10 和更高版本的支持。

1.2.27.26.1. Red Hat OpenShift Service Mesh 2.2.9 版中包含的组件版本

组件	版本
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.42.0
Kiali	1.48.7

1.2.27.27. Red Hat OpenShift Service Mesh 版本 2.2.8 的新功能

此 Red Hat OpenShift Service Mesh 发行版本引入了新功能，解决了 CVE 报告的安全漏洞问题 (CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.10 和更高版本的支持。

1.2.27.27.1. Red Hat OpenShift Service Mesh 2.2.8 版中包含的组件版本

组件	版本
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.42.0
Kiali	1.48.7

1.2.27.28. Red Hat OpenShift Service Mesh 版本 2.2.7 的新功能

此 Red Hat OpenShift Service Mesh 发行版本引入了新功能，解决了 CVE 报告的安全漏洞问题 (CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.10 和更高版本的支持。

1.2.27.28.1. Red Hat OpenShift Service Mesh 2.2.7 版中包含的组件版本

组件	版本
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.42.0
Kiali	1.48.6

1.2.27.29. Red Hat OpenShift Service Mesh 版本 2.2.6 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.9 及更新的版本的支持。

1.2.27.29.1. Red Hat OpenShift Service Mesh 2.2.6 版中包含的组件版本

组件	版本
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.39
Kiali	1.48.5

1.2.27.30. New features Red Hat OpenShift Service Mesh version 2.2.5

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.9 及更新的版本的支持。

1.2.27.30.1. Red Hat OpenShift Service Mesh 2.2.5 版中包含的组件版本

组件	版本
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.39
Kiali	1.48.3

1.2.27.31. New features Red Hat OpenShift Service Mesh version 2.2.4

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.9 及更新的版本的支持。

1.2.27.31.1. Red Hat OpenShift Service Mesh 2.2.4 版中包含的组件版本

组件	版本
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.36.14
Kiali	1.48.3

1.2.27.32. Red Hat OpenShift Service Mesh 版本 2.2.3 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)、程序错误修正，并受 Red Hat OpenShift Service on AWS 4.9 及更新版本的支持。

1.2.27.32.1. Red Hat OpenShift Service Mesh 2.2.3 版中包含的组件版本

组件	版本
Istio	1.12.9
Envoy Proxy	1.20.8

组件	版本
Jaeger	1.36
Kiali	1.48.3

1.2.27.33. Red Hat OpenShift Service Mesh 版本 2.2.2 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)、程序错误修正，并受 Red Hat OpenShift Service on AWS 4.9 及更新版本的支持。

1.2.27.33.1. Red Hat OpenShift Service Mesh 2.2.2 版中包含的组件版本

组件	版本
Istio	1.12.7
Envoy Proxy	1.20.6
Jaeger	1.36
Kiali	1.48.2-1

1.2.27.33.2. 复制路由标签

在这个版本中，除了复制注解外，您还可以为 OpenShift 路由复制特定的标签。Red Hat OpenShift Service Mesh 将 Istio 网关资源中存在的所有标签和注解（从 kubectl.kubernetes.io 开始的注解除外）复制到受管 OpenShift Route 资源中。

1.2.27.34. Red Hat OpenShift Service Mesh 版本 2.2.1 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)、程序错误修正，并受 Red Hat OpenShift Service on AWS 4.9 及更新版本的支持。

1.2.27.34.1. Red Hat OpenShift Service Mesh 2.2.1 版中包含的组件版本

组件	版本
Istio	1.12.7
Envoy Proxy	1.20.6
Jaeger	1.34.1
Kiali	1.48.2-1

1.2.27.35. Red Hat OpenShift Service Mesh 2.2 的新功能

此 Red Hat OpenShift Service Mesh 发行版本添加了新的功能和增强，并受 Red Hat OpenShift Service on AWS 4.9 及更新的版本的支持。

1.2.27.35.1. Red Hat OpenShift Service Mesh 2.2 版中包含的组件版本

组件	版本
Istio	1.12.7
Envoy Proxy	1.20.4
Jaeger	1.34.1
Kiali	1.48.0.16

1.2.27.35.2. WasmPlugin API

此发行版本添加了对 **WasmPlugin** API 的支持，并弃用 **ServiceMeshExtension** API。

1.2.27.35.3. ROSA 支持

此发行版本引进了对 AWS(ROSA)上的 Red Hat OpenShift 的服务网格支持，包括多集群联邦。

1.2.27.35.4. istio-node DaemonSet 重命名

在此发行版本中，**istio-node** DaemonSet 被重命名为 **istio-cni-node**，以匹配上游 Istio 中的名称。

1.2.27.35.5. Envoy sidecar 网络更改

Istio 1.10 更新了 Envoy，默认使用 **eth0** 而不是 **lo** 将流量发送到应用程序容器。

1.2.27.35.6. Service Mesh Control Plane 1.1

对于所有平台，此发行版本结束了对基于 Service Mesh 1.1 的 Service Mesh Control Planes 的支持。

1.2.27.35.7. Istio 1.12 支持

Service Mesh 2.2 基于 Istio 1.12，它带来新功能和产品改进。虽然仍然会支持许多 Istio 1.12 功能，但请注意以下不被支持的功能：

- AuthPolicy Dry Run 是一个技术预览功能。
- gRPC Proxyless Service Mesh 是一个技术预览功能。
- Telemetry API 是一个技术预览功能。
- 发现选择器功能不受支持。
- 外部 control plane 不受支持。
- 网关注入不受支持。

1.2.27.35.7.1. 安装网关 API CRD

默认情况下，网关 API CRD 不会在 OpenShift 集群中预安装。在 SMCP 中启用网关 API 支持前安装 CRD。

```
$ kubectl get crd gateways.gateway.networking.k8s.io || { kubectl kustomize
"github.com/kubernetes-sigs/gateway-api/config/crd?ref=v0.4.0" | kubectl apply -f -; }
```

1.2.27.35.7.2. 启用 Kubernetes 网关 API

要启用这个功能，请在 **ServiceMeshControlPlane** 中为 **Istiod** 容器设置以下环境变量：

```
spec:
  runtime:
    components:
      pilot:
        container:
          env:
            PILOT_ENABLE_GATEWAY_API: "true"
            PILOT_ENABLE_GATEWAY_API_STATUS: "true"
            # and optionally, for the deployment controller
            PILOT_ENABLE_GATEWAY_API_DEPLOYMENT_CONTROLLER: "true"
```

使用 **SameNamespace** 或 **All** 设置在 Gateway API 监听器上限制路由附加功能可能。Istio 会忽略 **listeners.allowedRoutes.namespaces** 中标签选择器的使用，并恢复到默认行为 (**SameNamespace**)。

1.2.27.35.7.3. 手动将现有网关链接到网关资源

如果禁用 Kubernetes API 部署控制器，您必须手动部署，然后将入口网关链接到创建的网关资源。

```
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: Gateway
metadata:
  name: gateway
spec:
  addresses:
  - value: ingress.istio-gateways.svc.cluster.local
    type: Hostname
```

1.2.27.36. New features Red Hat OpenShift Service Mesh 2.1.6

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.9 及更新的版本的支持。

1.2.27.36.1. Red Hat OpenShift Service Mesh 2.1.6 版中包含的组件版本

组件	版本
Istio	1.9.9
Envoy Proxy	1.17.5

组件	版本
Jaeger	1.36
Kiali	1.36.16

1.2.27.37. New features Red Hat OpenShift Service Mesh 2.1.5.2

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)，包含程序错误修复，并受 Red Hat OpenShift Service on AWS 4.9 及更新的版本的支持。

1.2.27.37.1. Red Hat OpenShift Service Mesh 2.1.5.2 版中包含的组件版本

组件	版本
Istio	1.9.9
Envoy Proxy	1.17.5
Jaeger	1.36
Kiali	1.24.17

1.2.27.38. New features Red Hat OpenShift Service Mesh 2.1.5.1

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)、程序错误修正，并受 Red Hat OpenShift Service on AWS 4.9 及更新版本的支持。

1.2.27.38.1. Red Hat OpenShift Service Mesh 2.1.5.1 版中包含的组件版本

组件	版本
Istio	1.9.9
Envoy Proxy	1.17.5
Jaeger	1.36
Kiali	1.36.13

1.2.27.39. Red Hat OpenShift Service Mesh 2.1.5 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)、程序错误修正，并受 Red Hat OpenShift Service on AWS 4.9 及更新版本的支持。

1.2.27.39.1. Red Hat OpenShift Service Mesh 2.1.5 版中包含的组件版本

组件	版本
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.36
Kiali	1.36.12-1

1.2.27.40. Red Hat OpenShift Service Mesh 2.1.4 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

1.2.27.40.1. Red Hat OpenShift Service Mesh 2.1.4 版中包含的组件版本

组件	版本
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.30.2
Kiali	1.36.12-1

1.2.27.41. Red Hat OpenShift Service Mesh 2.1.3 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

1.2.27.41.1. Red Hat OpenShift Service Mesh 2.1.3 版中包含的组件版本

组件	版本
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.30.2
Kiali	1.36.10-2

1.2.27.42. Red Hat OpenShift Service Mesh 2.1.2.1 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

1.2.27.42.1. Red Hat OpenShift Service Mesh 2.1.2.1 版中包含的组件版本

组件	版本
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.30.2
Kiali	1.36.9

1.2.27.43. Red Hat OpenShift Service Mesh 2.1.2 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

在这个版本中，Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 被默认安装到 **openshift-distributed-tracing** 命名空间。在以前的版本中，默认安装已在 **openshift-operator** 命名空间中。

1.2.27.43.1. Red Hat OpenShift Service Mesh 2.1.2 版中包含的组件版本

组件	版本
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.30.1
Kiali	1.36.8

1.2.27.44. Red Hat OpenShift Service Mesh 2.1.1 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

此发行版本还添加了禁用自动创建网络策略的功能。

1.2.27.44.1. Red Hat OpenShift Service Mesh 2.1.1 版中包含的组件版本

组件	版本
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.24.1

组件	版本
Kiali	1.36.7

1.2.27.44.2. 禁用网络策略

Red Hat OpenShift Service Mesh 自动在 Service Mesh control plane 和应用程序命名空间中创建和管理多个 NetworkPolicies 资源。这是为了确保应用程序和 control plane 可以相互通信。

如果要禁用自动创建和管理 NetworkPolicies 资源，例如为了强制执行公司安全策略，您可以编辑 ServiceMeshControlPlane，将 spec.security.manageNetworkPolicy 设置设置为 false

注意

当您禁用了 spec.security.manageNetworkPolicy，Red Hat OpenShift Service Mesh 不会创建 任何 NetworkPolicy 对象。系统管理员负责管理网络并修复可能导致的任何问题。

流程

1. 在 Red Hat OpenShift Service on AWS web 控制台中，点 Operators → Installed Operators。
2. 从 Project 菜单中选择安装 Service Mesh control plane 的项目，如 **istio-system**。
3. 点 Red Hat OpenShift Service Mesh Operator。在 Istio Service Mesh Control Plane栏中，点 **ServiceMeshControlPlane** 的名称，如 **basic-install**。
4. 在 Create ServiceMeshControlPlane Details 页中，点 **YAML** 修改您的配置。
5. 将 **ServiceMeshControlPlane** 字段 **spec.security.manageNetworkPolicy** 设置为 **false**，如下例所示。

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  security:
      trust:
      manageNetworkPolicy: false
```

6. 点 Save。

1.2.27.45. Red Hat OpenShift Service Mesh 2.1 的新功能和增强

这个版本的 Red Hat OpenShift Service Mesh 增加了在 Red Hat OpenShift Service on AWS 4.6 EUS, 4.7, 4.8, 4.9 上对 Istio 1.9.8, Envoy Proxy 1.17.1, Jaeger 1.24.1, 和 Kiali 1.36.5 的支持，以及新的功能和增强。

1.2.27.45.1. Red Hat OpenShift Service Mesh 2.1 版中包含的组件版本

组件	版本
Istio	1.9.6
Envoy Proxy	1.17.1
Jaeger	1.24.1
Kiali	1.36.5

1.2.27.45.2. Service Mesh Federation

添加了新的自定义资源定义(CRD)以支持联邦服务网格（federating service mesh）。服务网格可以整合到同一集群中或跨不同的 OpenShift 集群。这些新资源包括：

- **ServiceMeshPeer** - 使用单独的服务网格定义联邦，包括网关配置、root 信任证书配置和状态字段。在一对联邦网格中，每个网格将定义自己的独立 **ServiceMeshPeer** 资源。
- **ExportedServiceMeshSet** - 定义给定 **ServiceMeshPeshPeer** 的服务可用于导入的对等网格。
- **ImportedServiceSet** - 定义给定 **ServiceMeshPeer** 的服务是从 peer 网格中导入的。这些服务还必须由 peer 的 **ExportedServiceMeshSet** 资源提供。

在 AWS (ROSA)或 OpenShift Dedicated (OSD)上的 Red Hat OpenShift Service 上的集群间不支持 Service Mesh Federation。

1.2.27.45.3. OVN-Kubernetes Container Network Interface(CNI)正式发布

OVN-Kubernetes Container Network Interface(CNI)以前在 Red Hat OpenShift Service Mesh 2.0.1 中作为技术预览功能引进，现在包括在 Red Hat OpenShift Service Mesh 2.1 和 2.0.x 中，用于 Red Hat OpenShift Service on AWS 4.7.32、OpenShift Container Platform 4.8.12 和 OpenShift Container Platform 4.9。

1.2.27.45.4. Service Mesh WebAssembly(WASM)扩展

ServiceMeshExtensions 自定义资源定义(CRD)现已正式发布，它首次作为技术预览功能在版本 2.0 中推出。您可以使用 CRD 构建自己的插件，但红帽并不支持您创建的插件。

在 Service Mesh 2.1 中已完全删除 Mixer。如果启用了 Mixer，则会阻止从 Service Mesh 2.0.x 升级到 2.1。混合器插件需要移植到 WebAssembly 扩展。

1.2.27.45.5. 3scale WebAssembly Adapter(WASM)

Mixer 现已正式删除，OpenShift Service Mesh 2.1 不支持 3scale 混合器适配器。在升级到 Service Mesh 2.1 之前，删除基于 Mixer 的 3scale 适配器和任何其他 Mixer 插件。然后，使用 Service **MeshExtension** 资源手动安装和配置使用 Service Mesh 2.1+ 的新 3scale WebAssembly 适配器。

3scale 2.11 引入了基于 **WebAssembly** 的更新 Service Mesh 集成。

1.2.27.45.6. Istio 1.9 支持

Service Mesh 2.1 基于 Istio 1.9，它带来了大量新功能和产品增强。虽然大多数 Istio 1.9 功能被支持，但请注意以下例外：

- 虚拟机集成尚不受支持
- 尚不支持 Kubernetes 网关 API
- 尚不支持远程获取和加载 WebAssembly HTTP 过滤器
- 尚不支持使用 Kubernetes CSR API 的自定义 CA 集成
- 监控流量的请求分类是一个技术预览功能
- 通过授权策略的 CUSTOM 操作与外部授权系统集成是一项技术预览功能

1.2.27.45.7. 改进了 Service Mesh operator 性能

Red Hat OpenShift Service Mesh 在每个 **ServiceMeshControlPlane** 协调结束时用于修剪旧资源的时间已经减少。这会更快地进行 **ServiceMeshControlPlane** 部署，并允许应用到现有 SMCP 的更改更快地生效。

1.2.27.45.8. Kiali 更新

Kiali 1.36 包括以下功能和增强：

- Service Mesh 故障排除功能
  - control plane 和网关监控
  - 代理同步状态
  - Envoy 配置视图
  - 显示 Envoy 代理和应用程序日志处于交集的统一视图
- 支持联邦服务网格视图的命名空间和集群选择
- 新的验证、向导和分布式追踪增强

1.2.27.46. New features Red Hat OpenShift Service Mesh 2.0.11.1

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)、程序错误修正，并受 Red Hat OpenShift Service on AWS 4.9 或更高版本的支持。

1.2.27.46.1. Red Hat OpenShift Service Mesh 2.0.11.1 版中包含的组件版本

组件	版本
Istio	1.6.14
Envoy Proxy	1.14.5
Jaeger	1.36

组件	版本
Kiali	1.24.17

1.2.27.47. Red Hat OpenShift Service Mesh 2.0.11 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题(CVE)、程序错误修正，并受 Red Hat OpenShift Service on AWS 4.9 或更高版本的支持。

1.2.27.47.1. Red Hat OpenShift Service Mesh 2.0.11 版中包含的组件版本

组件	版本
Istio	1.6.14
Envoy Proxy	1.14.5
Jaeger	1.36
Kiali	1.24.16-1

1.2.27.48. Red Hat OpenShift Service Mesh 2.0.10 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

1.2.27.48.1. Red Hat OpenShift Service Mesh 2.0.10 版中包含的组件版本

组件	版本
Istio	1.6.14
Envoy Proxy	1.14.5
Jaeger	1.28.0
Kiali	1.24.16-1

1.2.27.49. Red Hat OpenShift Service Mesh 2.0.9 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

1.2.27.49.1. Red Hat OpenShift Service Mesh 2.0.9 版中包含的组件版本

组件	版本
Istio	1.6.14

组件	版本
Envoy Proxy	1.14.5
Jaeger	1.24.1
Kiali	1.24.11

1.2.27.50. Red Hat OpenShift Service Mesh 2.0.8 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了程序错误修正。

1.2.27.51. Red Hat OpenShift Service Mesh 2.0.7.1 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题。

1.2.27.51.1. Red Hat OpenShift Service Mesh 处理 URI 片段的方式改变

Red Hat OpenShift Service Mesh 包含一个可远程利用的漏洞 CVE-2021-39156，其中 HTTP 请求带有片段（以 # 字符开头的 URI 末尾的一个部分），您可以绕过 Istio URI 基于路径的授权策略。例如，Istio 授权策略拒绝发送到 URI 路径 /user/profile 的请求。在存在安全漏洞的版本中，带有 URI 路径 /user/profile#section1 的请求绕过拒绝策略并路由到后端（通过规范的 URIpath /user/profile%23section1），可能会导致安全事件。

如果您使用带有 DENY 操作和 operation.paths 的授权策略，或者 ALLOW 操作和 operation.notPaths，则会受到此漏洞的影响。

在这个版本中，在授权和路由前会删除请求的 URI 片段部分。这可以防止其 URI 中带有片段的请求绕过基于 URI 且没有片段部分的授权策略。

要从缓解措施中的新行为中选择，将保留 URI 的片段部分。您可以将 ServiceMeshControlPlane 配置为保留 URI 片段。

> **警告**
>
> 如前文所述，禁用新行为将对路径进行规范化，并被视为不安全。在选择保留 URI 片段之前，确保您已将这些内容放入任何安全策略中。

ServiceMeshControlPlane 修改示例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
```

```
techPreview:
  meshConfig:
    defaultConfig:
      proxyMetadata: HTTP_STRIP_FRAGMENT_FROM_PATH_UNSAFE_IF_DISABLED: "false"
```

1.2.27.51.2. 授权策略所需的更新

Istio 为主机名本身和所有匹配端口生成主机名。例如，用于 "httpbin.foo" 主机的虚拟服务或网关会生成匹配 "httpbin.foo 和 httpbin.foo:*" 的配置。但是，完全匹配授权策略仅与为 **hosts** 或 **notHosts** 字段给出的确切字符串匹配。

如果您使用精确字符串比较的 **AuthorizationPolicy** 来确定 主机或非主机，则会影响您的集群。

您必须更新授权策略 规则，以使用前缀匹配而不是完全匹配。例如，在第一个 **AuthorizationPolicy** 示例中，将 **hosts: ["httpbin.com"]** 替换为 **hosts: ["httpbin.com:*"]**。

第一个 AuthorizationPolicy 示例使用前缀匹配

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: httpbin
  namespace: foo
spec:
  action: DENY
  rules:
  - from:
    - source:
        namespaces: ["dev"]
    to:
    - operation:
        hosts: ["httpbin.com","httpbin.com:*"]
```

第二个 AuthorizationPolicy 示例使用前缀匹配

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: httpbin
  namespace: default
spec:
  action: DENY
  rules:
  - to:
    - operation:
        hosts: ["httpbin.example.com:*"]
```

1.2.27.52. Red Hat OpenShift Service Mesh 2.0.7 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

1.2.27.53. Red Hat OpenShift Dedicated 上的 Red Hat OpenShift Service Mesh

Red Hat OpenShift Service Mesh 现在通过 Red Hat OpenShift Dedicated 支持。

1.2.27.54. Red Hat OpenShift Service Mesh 2.0.6 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

1.2.27.55. Red Hat OpenShift Service Mesh 2.0.5 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

1.2.27.56. Red Hat OpenShift Service Mesh 2.0.4 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

重要

要解决 CVE-2021-29492 和 CVE-2021-31920 的问题，则必须完成手动步骤。

1.2.27.56.1. CVE-2021-29492 和 CVE-2021-31920 所需的手动更新

Istio 包含一个可被远程利用的漏洞，当使用基于路径的授权规则时，带有多个斜杠或转义的斜杠字符（%2F 或 %5C）的 HTTP 请求路径可能会绕过 Istio 授权策略。

例如，假设 Istio 集群管理员定义了一个授权 DENY 策略，以便在路径 **/admin** 上拒绝请求。发送到 URL 路径 **//admin** 的请求不会被授权策略拒绝。

根据 RFC 3986，带有多个斜杠的路径 **//admin** 在技术上应被视为与 **/admin** 不同的路径。但是，一些后端服务选择通过将多个斜杠合并成单斜杠来规范 URL 路径。这可能导致绕过授权策略（**//admin** 不匹配 **/admin**），用户可以在后端的路径 **/admin** 上访问资源，这可能会产生安全问题。

如果您使用 **ALLOW action + notPaths** 字段或者 **DENY action + paths** 字段特征，您的集群会受到这个漏洞的影响。这些模式可能会被意外的策略绕过。

在以下情况下，集群不会受到此漏洞的影响：

- 您没有授权策略。
- 您的授权策略没有定义 **paths** 或 **notPaths** 字段。
- 您的授权策略使用 **ALLOW action + paths** 字段或 **DENY action + notPaths** 字段特征。这些模式只会导致意外的拒绝，而不是绕过策略。对于以上情况，升级是可选的。

注意

路径规范化的 Red Hat OpenShift Service Mesh 配置位置与 Istio 配置不同。

1.2.27.56.2. 更新路径规范化配置

Istio 授权策略可能基于 HTTP 请求中的 URL 路径。路径规范化（也称为 URI 规范化）、修改和标准化传入请求的路径，以便能够以标准的方式处理规范化路径。在路径规范化后，同步不同路径可能是等同的。

Istio 在根据授权策略和路由请求前，支持请求路径中的以下规范化方案：

表 1.1. 规范化方案

Option	Description	Example	备注
NONE	没有进行规范化。Envoy 接收的任何内容都会完全按原样转发到任何后端服务。	**../%2Fa../b** 由授权策略评估并发送到您的服务。	此设置会受到 CVE-2021-31920 的影响。
BASE	这是目前 Istio **默认**安装中使用的选项。这在 Envoy 代理上应用 **normalize_path** 选项，该选项在 RFC 3986 之后使用额外的规范化来转换反斜杠来正斜杠。	**/a/../b** 被规范化为 **/b**。**\da** 被规范化为 **/da**。	此设置会受到 CVE-2021-31920 的影响。
MERGE_SLASHES	斜杠会在 *BASE* 规范化后合并。	**/a//b** 被规范化为 **/a/b**。	更新此设置以缓解 CVE-2021-31920 的问题。
DECODE_AND_MERGE_SLASHES	默认允许所有流量时的最严格设置。建议使用此设置，请注意您必须对您的授权策略路由进行彻底测试。Percent 编码的 斜杠和反斜杠字符（**%2F**、**%2f**、**%5C** 和 **%5c**）被解码为 / 或 \，在 **MERGE_SLASHES** 规范化前。	**/a%2fb** 规范化为 **/a/b**。	更新此设置以缓解 CVE-2021-31920 的问题。这个设置更为安全，但可能会破坏应用程序。在部署到生产环境中前测试您的应用程序。

规范化算法按以下顺序进行：

1. 解码百分比 %2F、%2f、%5C 和 %5c。
2. RFC 3986 和其他在 Envoy 中的 **normalize_path** 选项实现的规范化。
3. 合并斜杠。

> **警告**
>
> 虽然这些规范化选项代表来自 HTTP 标准和常见行业实践的建议，但应用程序可能会以它选择的任何方式解释 URL。在使用拒绝策略时，请确保您了解应用程序的行为方式。

1.2.27.56.3. 路径规范配置示例

确保 Envoy 规范化请求路径以匹配后端服务的预期，对您的系统安全至关重要。以下示例可用作配置系统的参考。规范化 URL 路径，如果选择 NONE，则原始 URL 路径为：

1. 用于检查授权策略。
2. 转发到后端应用程序。

表 1.2. 配置示例

如果您的应用程序…	选择…
依赖于代理进行规范化	BASE、MERGE_SLASHES 或 DECODE_AND_MERGE_SLASHES
根据 RFC 3986 规范化请求路径，且不合并斜杠。	BASE
根据 RFC 3986 和合并斜杠规范化请求路径，但不解码使用百分比编码的斜杠。	MERGE_SLASHES
根据 RFC 3986 规范化请求路径，解码 百分比编码的斜杠以及合并斜杠。	DECODE_AND_MERGE_SLASHES
以与 RFC 3986 不兼容的方式处理请求路径。	NONE

1.2.27.56.4. 为路径规范化配置 SMCP

要为 Red Hat OpenShift Service Mesh 配置路径规范化，请在 ServiceMeshControlPlane 中指定以下内容。使用配置示例来帮助确定您的系统设置。

SMCP v2 路径规范化

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

1.2.27.56.5. 配置大小写规范化

在某些环境中，在授权策略中对路径进行比较时不区分大小写可能很有用。例如，把 https://myurl/get 视为与 https://myurl/GeT 一样。在这些情况下，您可以使用如下所示的 EnvoyFilter。此过滤器将更改用于比较的路径以及应用程序显示的路径。在本例中，istio-system 是 Service Mesh control plane 项目的名称。

将 EnvoyFilter 保存到文件中并运行以下命令：

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

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: ingress-case-insensitive
  namespace: istio-system
spec:
  configPatches:
```

```
- applyTo: HTTP_FILTER
  match:
    context: GATEWAY
    listener:
      filterChain:
        filter:
          name: "envoy.filters.network.http_connection_manager"
          subFilter:
            name: "envoy.filters.http.router"
  patch:
    operation: INSERT_BEFORE
    value:
      name: envoy.lua
      typed_config:
        "@type": "type.googleapis.com/envoy.extensions.filters.http.lua.v3.Lua"
        inlineCode: |
          function envoy_on_request(request_handle)
            local path = request_handle:headers():get(":path")
            request_handle:headers():replace(":path", string.lower(path))
          end
```

1.2.27.57. Red Hat OpenShift Service Mesh 2.0.3 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

另外，这个版本有以下新特性：

- 在 must-gather 数据收集工具中添加了一个选项，用于从指定的 Service Mesh control plane 命名空间中收集信息。如需更多信息，请参阅 OSSM-351。
- 提高了带有数百个命名空间的 Service Mesh control plane 的性能

1.2.27.58. Red Hat OpenShift Service Mesh 2.0.2 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

1.2.27.59. Red Hat OpenShift Service Mesh 2.0.1 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

1.2.27.60. Red Hat OpenShift Service Mesh 2.0 的新功能

此 Red Hat OpenShift Service Mesh 发行版本添加了对 Istio 1.6.5、Jaeger 1.20.0、Kiali 1.24.2、3scale Istio Adapter 2.0 和 Red Hat OpenShift Service on AWS 4.6 的支持。

另外，这个版本有以下新特性：

- 简化了 Service Mesh control plane 的安装、升级和管理。
- 减少 Service Mesh control plane 的资源使用情况和启动时间。
- 通过降低网络间 control plane 通讯来提高性能。
    - 添加对 Envoy 的 Secret Discovery Service（SDS）的支持。SDS 是一个更加安全有效地向 Envoy side car proxies 发送 secret 的机制。

- 不再需要使用具有已知安全风险的 Kubernetes Secret。
- 在轮转证书的过程中提高了性能，因为代理不再需要重启来识别新证书。
  - 添加了对 Istio Telemetry v2 架构的支持，该架构是由 WebAssembly 扩展构建的。这个新架构带来了显著的性能改进。
  - 使用简化的配置将 ServiceMeshControlPlane 资源更新至 v2，以便更轻松地管理 Service Mesh Control Plane。

1.2.28. 弃用和删除的功能

之前版本中的一些功能已被弃用或删除。

弃用的功能仍然包含在 Red Hat OpenShift Service on AWS 中，并且仍然被支持。但是，弃用的功能可能会在以后的发行版本中被删除，且不建议在新的部署中使用。

删除的功能不再存在于产品中。

1.2.28.1. Red Hat OpenShift Service Mesh 2.5 中已弃用和删除的功能

v2.2 **ServiceMeshControlPlane** 资源不再被支持。客户应该更新其网格部署，以使用更新的 **ServiceMeshControlPlane** 资源版本。

对 Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 的支持已弃用。要收集追踪范围，请使用 Red Hat OpenShift distributed Tracing Platform (Tempo) Stack。

对 OpenShift Elasticsearch Operator 的支持已弃用。

Istio 将删除对第三方 JSON Web 令牌(JWT)的支持。Istio 仍然支持第三方 JWT。

1.2.28.2. Red Hat OpenShift Service Mesh 2.4 中已弃用和删除的功能

v2.1 **ServiceMeshControlPlane** 资源不再被支持。客户应升级其网格部署，以使用更新的 **ServiceMeshControlPlane** 资源版本。

对 Istio OpenShift 路由(IOR)的支持已弃用，并将在以后的发行版本中删除。

对 Grafana 的支持已弃用，并将在以后的发行版本中删除。

对 Red Hat OpenShift Service Mesh 2.3 中已弃用的以下密码套件的支持已从客户端和服务器端的 TLS 协商中使用的默认密码列表中删除。当从代理发起 TLS 连接时，需要访问需要这些密码套件之一的服务的应用程序将无法连接。

- ECDHE-ECDSA-AES128-SHA
- ECDHE-RSA-AES128-SHA
- AES128-GCM-SHA256
- AES128-SHA
- ECDHE-ECDSA-AES256-SHA
- ECDHE-RSA-AES256-SHA

- AES256-GCM-SHA384
- AES256-SHA

1.2.28.3. Red Hat OpenShift Service Mesh 2.3 中已弃用和删除的功能

对以下密码套件的支持已弃用。在以后的发行版本中，它们将从客户端和服务器端使用 TLS 协商的默认密码列表中删除。

- ECDHE-ECDSA-AES128-SHA
- ECDHE-RSA-AES128-SHA
- AES128-GCM-SHA256
- AES128-SHA
- ECDHE-ECDSA-AES256-SHA
- ECDHE-RSA-AES256-SHA
- AES256-GCM-SHA384
- AES256-SHA

Red Hat OpenShift Service Mesh 2.2 中已弃用的 **ServiceMeshExtension** API 已在 Red Hat OpenShift Service Mesh 2.3 中删除。如果使用 **ServiceMeshExtension** API，则必须迁移到 **WasmPlugin** API 来继续使用 WebAssembly 扩展。

1.2.28.4. Red Hat OpenShift Service Mesh 2.2 中已弃用的功能

ServiceMeshExtension API 从版本 2.2 开始已弃用，并将在以后的版本中删除。虽然 **ServiceMeshExtension** API 仍然在 2.2 版本中被支持，但客户应该开始使用新的**WasmPlugin** API。

1.2.28.5. 删除了 Red Hat OpenShift Service Mesh 2.2 中的功能

对于所有平台，此发行版本结束了对基于 Service Mesh 1.1 的 Service Mesh Control Planes 的支持。

1.2.28.6. 删除了 Red Hat OpenShift Service Mesh 2.1 中的功能

在 Service Mesh 2.1 中，Mixer 组件被删除。程序错误修正和支持会在 Service Mesh 2.0 生命周期结束时提供。

如果启用了 Mixer 插件，则不会从 Service Mesh 2.0.x 升级到 2.1。Mixer 插件必须移植到 WebAssembly 扩展。

1.2.28.7. Red Hat OpenShift Service Mesh 2.0 中已弃用的功能

Mixer 组件在版本 2.0 中已弃用，并将在版本 2.1 中删除。虽然在版本 2.0 中仍支持使用 Mixer 来实现扩展，但扩展应该已迁移到新的 WebAsembly 机制。

Red Hat OpenShift Service Mesh 2.0 不再支持以下资源类型：

- **Policy**（策略）（authentication.istio.io/v1alpha1）不再被支持。根据策略资源中的具体配置，您可能需要配置多个资源来达到同样效果。

- 使用 **RequestAuthentication** (security.istio.io/v1beta1)
- 使用 **PeerAuthentication** (security.istio.io/v1beta1)

- **ServiceMeshPolicy** (maistra.io/v1)不再被支持。
  - 使用上述 **RequestAuthentication** 或 **PeerAuthentication**，但放置在 Service Mesh control plane 命名空间中。
- **RbacConfig**（rbac.istio.io/v1alpha1）不再被支持。
  - 由 **AuthorizationPolicy** (security.istio.io/v1beta1)替代，其中包含 **RbacConfig**、**ServiceRole** 和 **ServiceRoleBinding** 的行为。
- **ServiceMeshRbacConfig**（ maistra.io/v1）不再被支持。
  - 使用上述 **AuthorizationPolicy**，但保留在 Service Mesh control plane 命名空间中。
- **ServiceRole**（rbac.istio.io/v1alpha1）不再被支持。
- **ServiceRoleBinding**（rbac.istio.io/v1alpha1）不再被支持。
- 在 Kiali 中，**login** 和 **LDAP** 策略已被弃用。将来的版本将引入使用 OpenID 供应商的身份验证。

1.2.29. 已知问题

Red Hat OpenShift Service Mesh 中存在以下限制：

- Red Hat OpenShift Service Mesh 还没有完全支持 IPv6。因此，Red Hat OpenShift Service Mesh 不支持双栈集群。
- 图形布局 - Kiali 图形的布局会根据应用程序构架和要显示的数据（图形节点数目及其交互）的不同而有所变化。因为创建一个统一布局的难度较大，所以 Kiali 提供了几种不同布局的选择。要选择不同的布局，可从 Graph Settings 菜单中选择一个不同的 Layout Schema。
- 您第一次从 Kiali 控制台访问相关服务，如分布式 Tracing Platform (Jaeger)和 Grafana，您必须接受证书并使用您的 Red Hat OpenShift Service on AWS 登录凭证重新进行身份验证。这是因为框架如何显示控制台中的内置页面中存在问题。

1.2.29.1. Service Mesh 已知问题

当 istio-system 标签与发现选择器不匹配时，会跳过这些 Red Hat OpenShift Service Mesh 中已知的问题：* OSSM-5556 Gateways 会被跳过。

+ 临时解决方案 ：标记 control plane 命名空间以匹配发现选择器，以避免跳过网关配置。

+ .example **ServiceMeshControlPlane** 资源

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  mode: ClusterWide
  meshConfig:
```

```
    discoverySelectors:
    - matchLabels:
        istio-discovery: enabled
  gateways:
    ingress:
      enabled: true
```

+ 然后在命令行中运行以下命令：

+

```
oc label namespace istio-system istio-discovery=enabled
```

- OSSM-3890 尝试在多租户网格部署中使用网关 API 会生成类似如下的错误消息：

  ```
  2023-05-02T15:20:42.541034Z error watch error in cluster Kubernetes: failed to list
  *v1alpha2.TLSRoute: the server could not find the requested resource (get
  tlsroutes.gateway.networking.k8s.io)
  2023-05-02T15:20:42.616450Z info kube controller
  "gateway.networking.k8s.io/v1alpha2/TCPRoute" is syncing...
  ```

  要在多租户网格部署中支持网关 API，集群中必须存在所有网关 API 自定义资源定义(CRD)文件。

  在多租户网格部署中，禁用了 CRD 扫描，Istio 无法发现集群中存在哪些 CRD。因此，Istio 会尝试监视所有支持的网关 API CRD，但如果这些 CRD 不存在，则会生成错误。

  Service Mesh 2.3.1 及更新的版本支持 **v1alpha2** 和 **v1beta1** CRD。因此，两个 CRD 版本都必须存在才能多租户网格部署来支持网关 API。

  临时解决方案：在以下示例中，**kubectl get** 操作会安装 **v1alpha2** 和 **v1beta1** CRD。请注意，URL 包含额外的 *实验性* 片段，并相应地更新任何现有脚本：

  ```
  $ kubectl get crd gateways.gateway.networking.k8s.io ||   { kubectl kustomize
  "github.com/kubernetes-sigs/gateway-api/config/crd/experimental?ref=v0.5.1" |
  kubectl apply -f -; }
  ```

- OSSM-2042 Deployment of SMCP named**default** 会失败。如果您要创建 SMCP 对象，并将其 version 字段设置为 v2.3，则对象的名称不能 是默认的。如果名称是 默认，则 control plane 无法部署，OpenShift 会生成一个带有以下信息的 **Warning** 事件：
  error processing component mesh-config: error: [mesh-config/templates/telemetryv2_1.6.yaml： 发生内部错误： 调用 **Webhook "rev.validation.istio.io"： post "https://istiod-default.istio-system.svc:443/validate?timeout=10s": x509:** 证书对 **istiod.istio-system.svc**、**istiod-remote.istio-system.svc, istio-pilot.istio-system.svc** 有效，而不是 **istiod-default.istio-system.svc, mesh-config/templates/enable-mesh-permissive.yaml**

- OSSM-1655 Kiali 仪表板在 **SMCP** 中启用 mTLS 后显示错误。
  在 SMCP 中启用 **spec.security.controlPlane.mtls** 设置后，Kiali 控制台会显示以下错误消息 **No subsets defined**。

- OSSM-1505 只有在 OpenShift Container Platform 4.11 中使用**ServiceMeshExtension** 资源时才会发生。当在 OpenShift Container Platform 4.11 上使用 **ServiceMeshExtension** 时，资源永不会变为就绪。如果使用 **oc describe ServiceMeshExtension** 检查问题，您会看到以下错

误：stderr: Error create mount namespace before pivot: function not implemented。
临时解决方案：ServiceMeshExtension 在 Service Mesh 2.2 中已弃用。从 ServiceMeshExtension 迁移到 WasmPlugin 资源。如需更多信息，请参阅从 ServiceMeshExtension 迁移到 WasmPlugin 资源。

- OSSM-1396 如果一个网关资源包含 spec.externalIPs 设置，而不是在 ServiceMeshControlPlane 更新时重新创建，则该网关会被删除且永远不会重新创建。

- OSSM-1168 当以单个 YAML 文件形式创建服务网格资源时，Envoy proxy sidecar 不会可靠地注入 pod。当单独创建 SMCP、SMMR 和 Deployment 资源时，部署可以正常工作。

- OSSM-1115 spec.proxy API的并发 字段没有传播到 istio-proxy。当使用ProxyConfig 设置时，并发 字段可以正常工作。concurrency 字段指定要运行的 worker 线程数量。如果字段设为 0， 则可用的 worker 线程数量等于 CPU 内核数。如果没有设置该字段，则可使用的 worker 线程数量默认为 2。
在以下示例中，concurrency 字段设置为 0。

```
apiVersion: networking.istio.io/v1beta1
kind: ProxyConfig
metadata:
  name: mesh-wide-concurrency
  namespace: <istiod-namespace>
spec:
  concurrency: 0
```

- OSSM-1052 在为服务网格 control plane 中为 ingressgateway 配置 ServiceExternalIP 时，不会创建该服务。SMCP 的 schema 缺少该服务的参数。
临时解决方案：禁用 SMCP spec 中的网关创建，手动管理网关部署（包括 Service、Role 和 RoleBinding）。

- OSSM-882 适用于 Service Mesh 2.1 及更早版本。命名空间位于 accessible_namespace 列表中，但不出现在 Kiali UI 中。默认情况下，Kiali 不会显示任何以"kube"开头的命名空间，因为这些命名空间通常仅供内部使用，而不是网格的一部分。
例如，如果您创建一个名为 'akube-a' 的命名空间并将其添加到 Service Mesh member roll 中，Kiali UI 不会显示这个命名空间。这是因为对于定义的排除特征，软件会排除以定义特征开始或包括定义特征的命名空间。

  临时解决方案：更改 Kiali 自定义资源设置，以便使用尖号 (^) 前缀设置。例如：

```
api:
  namespaces:
    exclude:
    - "^istio-operator"
    - "^kube-.*"
    - "^openshift.*"
    - "^kiali-operator"
```

- MAISTRA-2692 删除了 Mixer，在 Service Mesh 2.0.x 中定义的自定义指标无法在 2.1 中使用。可以使用 EnvoyFilter 配置自定义指标。除非有明确记录，红帽不支持 EnvoyFilter 配置。这是因为与底层 Envoy API 耦合紧密，这意味着无法维护向后兼容性。

- MAISTRA-1959 *迁移到 2.0* Prometheus 提取（spec.addons.prometheus.scrape 设置为 true）在启用 mTLS 时无法正常工作。另外，当禁用 mTLS 时，Kiali 会显示无关的图形数据。
可通过将端口 15020 从代理配置中排除来解决这个问题，例如：

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

- MAISTRA-453 如果创建新项目并立即部署 pod，则不会进行 sidecar 注入。在创建 pod 前，operator 无法添加 maistra.io/member-of，因此必须删除 pod 并重新创建它以执行 sidecar 注入操作。

- MAISTRA-158 应用指向同一主机名的多个网关时，会导致所有网关停止工作。

1.2.29.2. Kiali 已知问题

注意

Kiali 的新问题应该在 OpenShift Service Mesh 项目中创建，Component 设为 Kiali。

Kiali 中已知的问题：

- OSSM-6299 在 Red Hat OpenShift Service on AWS 4.15 中，当您点击流量图中任何节点菜单选项时，不会显示节点图形。相反，页面会使用相同的流量图刷新。目前，这个问题还没有临时解决方案。

- OSSM-6298 当您点 OpenShift Service Mesh Console (OSSMC)插件中的项目引用时，如与特定服务相关的工作负载链接，控制台有时会在打开所需页面前执行多个重定向。如果您在网页浏览器中点 Back，则会打开控制台的不同页面，而不是上一页。作为临时解决方案，点 Back 两次来进入上一页。

- OSSM-6290 for Red Hat OpenShift Service on AWS 4.15, Istio Config 列表页面的 Project 过滤器无法正常工作。即使从下拉菜单选择一个特定的项目，也会显示所有 istio 项。目前，这个问题还没有临时解决方案。

- KIALI-2206 当您第一次访问 Kiali 控制台时，浏览器中没有 Kiali 的缓存数据，Kiali 服务详情页面的 Metrics 标签页中的 “View in grafana” 链接会重定向到错误的位置。只有在第一次访问 Kiali 才会出现这个问题。

- KIALI-507 Kiali 不支持 Internet Explorer 11。这是因为底层框架不支持 Internet Explorer。要访问 Kiali 控制台，请使用 Chrome 、Edge 、Firefox 或 Safari 浏览器的两个最新版本之一。

1.2.30. 修复的问题

在之前的版本中解决了以下问题：

1.2.30.1. Service Mesh 修复的问题

- OSSM-6177 以前，当在 **ServiceMeshControlPlane** (SMCP)中启用验证信息时，istiod 会持续崩溃，除非启用了 **GatewayAPI** 支持。现在，当启用验证信息但没有支持 **GatewayAPI** 时，**istio d** 不会持续崩溃。

- OSSM-6163 解决了以下问题：
  - 在以前的版本中，Service Mesh control plane (SMCP) v2.5 中包含了一个不稳定的 Prometheus 镜像，用户无法访问 Prometheus 仪表板。现在，在 Service Mesh operator 2.5.1 中，Prometheus 镜像已被更新。
  - 在以前的版本中，在 Service Mesh control plane (SMCP)中，Grafana 数据源无法自动设置基本身份验证密码，用户可以在 Grafana mesh 仪表板中查看 Prometheus 的指标。现在，Grafana 数据源密码配置在 secureJsonData 字段下。指标数据会在仪表板中正确显示。
- OSSM-6148 以前，当用户点击 Traffic Graph 页面上任何节点菜单中的任何选项时，OpenShift Service Mesh Console (OSSMC)插件没有响应。现在，插件通过重定向到对应的详情页面来响应菜单中的所选选项。
- OSSM-6099 以前，OpenShift Service Mesh 控制台(OSSMC)插件无法在 IPv6 集群中正确加载。现在，OSSMC 插件配置已被修改，以确保在 IPv6 集群中正确载入。
- OSSM-5960 以前，OpenShift Service Mesh Console (OSSMC)插件没有显示通知信息，如后端错误或 Istio 验证。现在，这些通知会在插件页面的顶部正确显示。
- OSSM-5959 以前，OpenShift Service Mesh Console (OSSMC)插件没有在 Overview 页面中显示 TLS 和 Istio 认证信息。现在，这些信息会被正确显示。
- OSSM-5902 以前，当用户在 Overview 页面中点 Istio 配置健康符号时，OpenShift Service Mesh Console (OSSMC)插件会重定向到 "Not Found Page" 错误。现在，插件会重定向到正确的 Istio 配置 详情页面。
- OSSM-5541 以前，Istio operator pod 在一些重启条件中可能会等待领导租期。现在，领导选举实施已被改进，以避免出现这个问题。
- OSSM-1397 以前，如果您从命名空间中删除了 maistra.io/member-of 标签，Service Mesh Operator 不会自动将标签应用到命名空间。因此，sidecar 注入无法在命名空间中工作。

当您更改 ServiceMeshMember 对象时，Operator 会将标签应用到命名空间，这会触发此 member 对象的协调。

现在，对命名空间的任何更改也会触发 member 对象协调。

- OSSM-3647 以前，在 Service Mesh control plane (SMCP) v2.2 (Istio 1.12) 中，WasmPlugins 仅适用于入站监听程序。从 SMCP v2.3 (Istio 1.14)开始，WasmPlugins 默认应用于入站和出站监听程序，这为 3scale WasmPlugin 的用户引入了回归。现在，添加了环境变量 APPLY_WASM_PLUGINS_TO_INBOUND_ONLY，它允许从 SMCP v2.2 安全地迁移到 v2.3 和 v2.4。

  以下设置应添加到 SMCP 配置中：

```
spec:
  runtime:
    components:
      pilot:
        container:
          env:
            APPLY_WASM_PLUGINS_TO_INBOUND_ONLY: "true"
```

  要确保安全迁移，请执行以下步骤：

  1. 在 SMCP v2.2 中设置 APPLY_WASM_PLUGINS_TO_INBOUND_ONLY。

  2. 升级到 2.4。

  3. 在 WasmPlugins 中设置 spec.match[].mode: SERVER。

  4. 删除之前添加的环境变量。

- OSSM-4851 以前，当 runAsGroup、runAsUser 或 fsGroup 参数为 nil 时，Operator 在网格内部署新 pod 时出错。现在，添加了 yaml 验证以避免 nil 值。

- OSSM-3771 以前，对于 Service Mesh Control Plane (SMCP)中定义的额外入口网关无法

禁用 OpenShift 路由。现在，可以将 routeConfig 块添加到 每个额外的Ingress 网关中，以便为每个网关启用或禁用 OpenShift 路由。

- OSSM-4197 以前，如果您部署了 'ServiceMeshControlPlane' 资源的 v2.2 或 v2.1，则不会创建 /etc/cni/multus/net.d/ 目录。因此，istio-cni pod 无法就绪，istio-cni pod 日志包含以下消息：

  ```
  $ error   Installer exits with open /host/etc/cni/multus/net.d/v2-2-istio-cni.kubeconfig.tmp.841118073: no such file or directory
  ```

  现在，如果您部署 'ServiceMeshControlPlane' 资源的 v2.2 或 v2.1，则 /etc/cni/multus/net.d/ 目录已创建，并且 istio-cni pod 变为 ready。

- OSSM-3993 之前，Kiali 只通过标准 HTTPS 端口 443 上的代理支持 OpenShift OAuth。现在，Kiali 通过非标准 HTTPS 端口支持 OpenShift OAuth。要启用端口，您必须将 spec.server.web_port 字段设置为 Kiali CR 中的代理的非标准 HTTPS 端口。

- OSSM-3936 之前，injection_label_rev 和 injection_label_name 属性的值被硬编码。这导致自定义配置无法在 Kiali 自定义资源定义(CRD)中生效。现在，属性值不会被硬编码。您可以在 spec.istio_labels 规格中自定义 injection_label_rev 和 injection_label_name 属性的值。

- OSSM-3644 以前，联邦 egress-gateway 收到网络网关端点的错误更新，从而导致额外的端点条目。现在，在服务器端更新了 federation-egress 网关，以便它接收正确的网络网关端点。

- OSSM-3595 以前，istio-cni 插件有时会在 RHEL 上失败，因为 SELinux 不允许工具 iptables-restore 打开 /tmp 目录中的文件。现在，SELinux 通过 stdin 输入流而不是通过一个文件传递 iptables-restore。

- OSSM-3586 之前，当 Google Cloud Platform (GCP)元数据服务器不可用时，Istio 代理会较慢。当您升级到 Istio 1.14.6 时，Istio 代理会在 GCP 上按预期启动，即使元数据服务器不可用。

- OSSM-3025 Istiod 有时无法就绪。有时，当网格包含很多成员命名空间时，Istiod pod 未就绪，因为 Istiod 中的死锁。现在，死锁已被解决，pod 现在会如预期启动。

- OSSM-2493 默认 nodeSelector 和 tolerations 不在 SMCP 中传递给 Kiali。您添加到 SMCP.spec.runtime.defaults 的 nodeSelector 和 tolerations 现在被传递给 Kiali 资源。

- OSSM-2492 默认容限没有传递给 Jaeger。您添加到 SMCP.spec.runtime.defaults 的 nodeSelector 和 tolerations 现在被传递给 Jaeger 资源。

- OSSM-2374 如果您删除了其中一个 ServiceMeshMember 资源，则 Service Mesh operator 会删除 ServiceMeshMemberRoll。虽然当您删除最后一个 ServiceMeshMember 时这是预期的行为，但如果它还包含任何成员，Operator 不应该删除 ServiceMeshMemberRoll。这个问题现已解决，Operator 仅在最后一个 ServiceMeshMember 资源被删除时删除 ServiceMeshMemberRoll。

- OSSM-2373 登录时尝试获取 OAuth 元数据的错误。要获取集群版本，请使用 system:anonymous 帐户。使用集群的默认捆绑的 ClusterRole 和 ClusterRoleBinding，匿名帐户可以正确地获取版本。如果 system:anonymous 帐户丢失了获取集群版本的权限，OpenShift 身份验证将不可用。

  这个问题已通过使用 Kiali SA 获取集群版本来解决。这也允许提高集群上的安全性。

- OSSM-2371 虽然 Kiali 配置为 "view-only"，但用户可以通过 Workload details 的 kebab 菜单更改代理日志级别。这个问题已被解决，当 Kiali 配置为 "view-only" 时，"Set Proxy Log Level" 下的选项被禁用。

- OSSM-2344 Restarting Istiod 会导致 Kiali 使用 port-forward 请求大量 CRI-O。当 Kiali 无法连接到 Istiod，Kiali 同时向 istiod 发出大量请求时会出现这种情况。Kiali 现在限制它发送到 istiod 的请求数。

- OSSM-2335 将鼠标指针拖放到 Traces scatterchart 图表上，有时会导致 Kiali 控制台因为并发后端请求停止响应。

- OSSM-2221 以前，ServiceMeshControlPlane 命名空间中的网关注入无法进行，因为 ignore-namespace 标签默认应用到命名空间。

  在创建 v2.4 control plane 时，命名空间不再应用 ignore-namespace 标签，并可进行网关注入。

在以下示例中，oc label 命令从现有部署中的命名空间中删除 ignore-namespace 标签：

```
$ oc label namespace istio-system maistra.io/ignore-namespace-
```

其中：

istio_system

指定 ServiceMeshControlPlane 命名空间的名称。

- OSSM-2053 使用 Red Hat OpenShift Service Mesh Operator 2.2 或 2.3，在 SMCP 协调过程中，SMMR 控制器会从 SMMR.status.configuredMembers 中删除成员命名空间。这会导致成员命名空间中的服务在一些时间不可用。

  使用 Red Hat OpenShift Service Mesh Operator 2.2 或 2.3，SMMR 控制器不再从 SMMR.status.configuredMembers 中删除命名空间。相反，控制器会将命名空间添加到 SMMR.status.pendingMembers 中，以指示它们不是最新的。在协调过程中，因为每个命名空间与 SMCP 同步，命名空间会自动从 SMMR.status.pendingMembers 中删除。

- OSSM-1962 使用 EndpointSlices 联合控制器。联合控制器现在使用 EndpointSlices，这提高了大型部署的可扩展性和性能。默认情况下启用 PILOT_USE_ENDPOINT_SLICE 标志。禁用该标志可防止使用联合部署。

- OSSM-1668 一个新的字段 spec.security.jwksResolverCA 已添加到版本 2.1 SMCP 中，但没有存在于 2.2.0 和 2.2.1 版本中。当从存在此字段的 Operator 版本升级到缺少此字段的 Operator 版本时，则 SMCP 中没有 .spec.security.jwksResolverCA 字段。

- OSSM-1325 istiod pod 崩溃并显示以下出错信息：fatal error: concurrent map iteration and map write。

- OSSM-1211 为故障转移配置联邦服务网格无法正常工作。

  Istiod pilot 日志显示以下错误： envoy connection [C289] TLS error: 337047686:SSL routines:tls_process_server_certificate:certificate verify failed

-

OSSM-1099 Kiali 控制台显示消息 Sorry, there was a problem.Try a refresh or navigate to a different page.

- OSSM-1074 Pod 注解没有在 pod 中注入。

- OSSM-999 Kiali retention 无法按预期工作。仪表板图中的日历时间会被问候。

- OSSM-797 Kiali Operator pod 在安装或更新 Operator 时生成 CreateContainerConfigError。

- 从 kube 开始的 OSSM-722 命名空间从 Kiali 中隐藏。

- OSSM-569 Prometheus istio-proxy 容器没有 CPU 内存限值。Prometheus istio-proxy sidecar 现在使用 spec.proxy.runtime.container 中定义的资源限值。

- OSSM-535 支持 SMCP 中的验证消息。Service Mesh Control Plane 中的 ValidationMessages 字段现在可以设置为 True。这会写入资源状态的日志，这在进行故障排除时很有用。

- OSSM-449 VirtualService 和 Service 会导致一个错误 - "Only unique values for domains are permitted.Duplicate entry of domain."

- 具有类似名称的 OSSM-419 命名空间都显示在 Kiali 命名空间列表中，即使命名空间可能无法在 Service Mesh Member Role 中定义。

- OSSM-296 当在 Kiali 自定义资源(CR)中添加健康配置时，不会将其复制到 Kiali configmap 中。

- OSSM-291 在 Kiali 控制台中，在 Applications、Services 和 Workloads 页面中，"Remove Label from Filters"功能无法正常工作。

- OSSM-289 在 Kiali 控制台中，'istio-ingressgateway' 和 'jaeger-query' 服务的服务详情

页面中没有显示 Traces。Jaeger 中存在 trace。

- OSSM-287 在 Kiali 控制台中没有显示 Graph 服务中的 trace。

- OSSM-285 试图访问 Kiali 控制台时会收到以下错误消息："Error trying to get OAuth Metadata"。

  临时解决方案：重启 Kiali pod。

- MAISTRA-2735 当 Red Hat OpenShift Service Mesh 2.1 中协调 SMCP 更改时，Service Mesh Operator 会删除的资源。在以前的版本中，Operator 删除了带有以下标签的资源：

  - maistra.io/owner

  - app.kubernetes.io/version

  现在，Operator 会忽略没有包括 app.kubernetes.io/managed-by=maistra-istio-operator 标签的资源。如果创建自己的资源，则不应将 app.kubernetes.io/managed-by=maistra-istio-operator 标签添加到其中。

- MAISTRA-2687 Red Hat OpenShift Service Mesh 2.1 联邦网关在使用外部证书时不会发送完整的证书链。Service Mesh 联邦出口网关仅发送客户端证书。因为联邦入口网关只知道 root 证书，所以它无法验证客户端证书，除非您将 root 证书添加到联合导入 ConfigMap 中。

- MAISTRA-2635 替换已弃用的 Kubernetes API。为了保持与 Red Hat OpenShift Service on AWS 4.8 兼容，apiextensions.k8s.io/v1beta1 API 从 Red Hat OpenShift Service Mesh 2.0.8 开始已弃用。

- MAISTRA-2631 WASM 功能不起作用，因为 podman 因 nsenter 二进制不存在而失败。Red Hat OpenShift Service Mesh 生成以下出错信息：Error: error configuring CNI network plugin exec: "nsenter": executable file not found in $PATH。容器镜像现在包含 nsenter，WASM 可以正常工作。

-

MAISTRA-2534 当 istiod 试图为 JWT 规则中指定的签发者获取 JWKS 时，签发者服务会使用 502 响应。这导致代理容器就绪，并导致部署挂起。Service Mesh 2.0.7 版本中包括了对社区程序漏洞的修复。

- MAISTRA-2411 当 Operator 使用 ServiceMeshControlPlane 中的 spec.gateways.additionalIngress 创建新的入口网关时，Operator 不会为额外的入口网关创建一个 NetworkPolicy，如默认的 istio-ingressgateway。这导致了来自新网关路由的 503 响应。

  临时解决方案：在 istio-system 命名空间中手动创建 NetworkPolicy。

- MAISTRA-2401 CVE-2021-3586 servicemesh-operator：NetworkPolicy 资源为 ingress 资源指定错误的端口。为 Red Hat OpenShift Service Mesh 安装的 NetworkPolicy 资源没有正确指定可访问哪些端口。这允许从任何 pod 访问这些资源上的所有端口。应用到以下资源的网络策略会受到影响：
  - Galley
  - Grafana
  - Istiod
  - Jaeger
  - Kiali
  - Prometheus
  - Sidecar injector

- MAISTRA-2378 当集群被配置为使用带有 ovs-multitenant 的 OpenShift SDN，且网格包含大量命名空间(200+)，Red Hat OpenShift Service on AWS 网络插件无法快速配置命名空间。Service Mesh 超时会导致从服务网格中持续丢弃命名空间，然后重新加入。

- MAISTRA-2370 Handle tombstones in listerInformer。在将事件从命名空间缓存转换为聚合缓存时，更新的缓存代码库没有处理 tombstones，从而导致在 go 中出现 panic 的问题。

- MAISTRA-2117 向 operator 添加可选的 ConfigMap 挂载。CSV 现在包含一个可选的 ConfigMap 卷挂载，它会挂载 smcp-templates ConfigMap （如果存在）。如果 smcp-templates ConfigMap 不存在，则挂载的目录为空。创建 ConfigMap 时，目录会填充 ConfigMap 中的条目，并可在 SMCP.spec.profiles 中引用。不需要重启 Service Mesh operator。

  使用带有修改 CSV 的 2.0 operator 挂载 smcp-templates ConfigMap 的用户可升级到 Red Hat OpenShift Service Mesh 2.1。升级后，您可以继续使用现有的 ConfigMap 及其包含的配置集，而无需编辑 CSV。以前使用不同名称的 ConfigMap 的客户需要重命名 ConfigMap 或升级后更新 CSV。

- MAISTRA-2010 AuthorizationPolicy 不支持 request.regex.headers 字段。validatingwebhook 会拒绝任何带有字段的 AuthorizationPolicy，即使您禁用该字段，Pilot 也会尝试使用相同的代码验证它，且它无法正常工作。

- MAISTRA-1979 *迁移至 2.0* 在将 SMCP.status 从 v2 转换为 v1 时，转换 Webhook 会丢弃以下重要字段：

  - conditions
  - components
  - observedGeneration
  - annotations

  将 operator 升级到 2.0 可能会破坏使用 maistra.io/v1 版本读取 SMCP 状态的客户端工具。

  这还会导致在运行 oc get servicemeshcontrolplanes.v1.maistra.io 时 READY 和 STATUS 列为空。

- MAISTRA-1983 *迁移到 2.0* 把带有存在无效 ServiceMeshControlPlane 的系统升级到 2.0.0 的问题无法被简单修复。ServiceMeshControlPlane 资源中的无效项会导致无法恢复的错误。在这个版本中，这个错误可以被恢复。您可以删除无效的资源，并将其替换为新资源或编辑资源来修复错误。有关编辑资源的更多信息，请参阅 [配置 Red Hat OpenShift Service Mesh 安装]。

- MAISTRA-1502 由于在版本 1.0.10 中修复了 CVE，Istio 仪表板将不会出现在 Grafana 的 Home Dashboard 菜单中。要访问 Istio 仪表板，点导航面板中的 Dashboard 菜单，然后选择 Manage 选项卡。

- MAISTRA-1399 Red Hat OpenShift Service Mesh 不再阻止您安装不支持的 CNI 协议。支持的网络配置没有改变。

- MAISTRA-1089 *迁移到在非 control plane 命名空间中创建的 2.0* 网关将自动删除。从 SMCP spec 中删除网关定义后，您需要手动删除这些资源。

- MAISTRA-858 Envoy 日志中以下与 与 Istio 1.1.x 相关的弃用选项和配置相关的信息是正常的：

  - [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 被逐出的 Istio Operator Pod 会导致 mesh 和 CNI 不能被部署。

  临时解决方案：如果在部署 control pane 时 istio-operator pod 被逐出，删除被逐出的 istio-operator pod。

-

MAISTRA-681 当 Service Mesh control plane 有多个命名空间时，可能会导致出现性能问题。

- MAISTRA-193 当为 citadel 启用了健康检查功能时，会出现预期外的控制台信息。

- Bugzilla 1821432 Red Hat OpenShift Service on AWS 自定义资源详情页面中的切换控件无法正确更新 CR。Red Hat OpenShift Service on AWS Web 控制台中的 Service Mesh Control Plane (smcp) Overview 页面中的 UI 切换控制有时会更新资源中的错误字段。要更新 SMCP，直接编辑 YAML 内容，或者从命令行更新资源，而不是点击 toggle 控件。

1.3. 了解 SERVICE MESH

Red Hat OpenShift Service Mesh 提供了一个平台，用于对服务网格（service mesh）中联网的微服务进行行为了解和操作控制。通过使用 Red Hat OpenShift Service Mesh，可以连接、控制并监控 Red Hat OpenShift Service on AWS 环境中的微服务。

1.3.1. 什么是 Red Hat OpenShift Service Mesh？

*服务网格（service mesh）*是一个微服务网络，它用于在一个分布式的微服务架构中构成应用程序，并提供不同微服务间的交互功能。当服务网格的规模和复杂性增大时，了解和管理它就会变得非常困难。

Red Hat OpenShift Service Mesh 基于开源 Istio 项目，它在不需要修改服务代码的情况下，为现有的分布式应用程序添加了一个透明的层。您可以在服务中添加对 Red Hat OpenShift Service Mesh 的支持，方法是将一个特殊的 sidecar 代理服务器部署到用于处理不同微服务之间的所有网络通讯的服务网格中。您可以使用 Service Mesh control plane 功能配置和管理 Service Mesh。

Red Hat OpenShift Service Mesh 可让您轻松创建部署的服务网络，该网络提供：

- 发现

- 负载平衡

- 服务到服务的验证

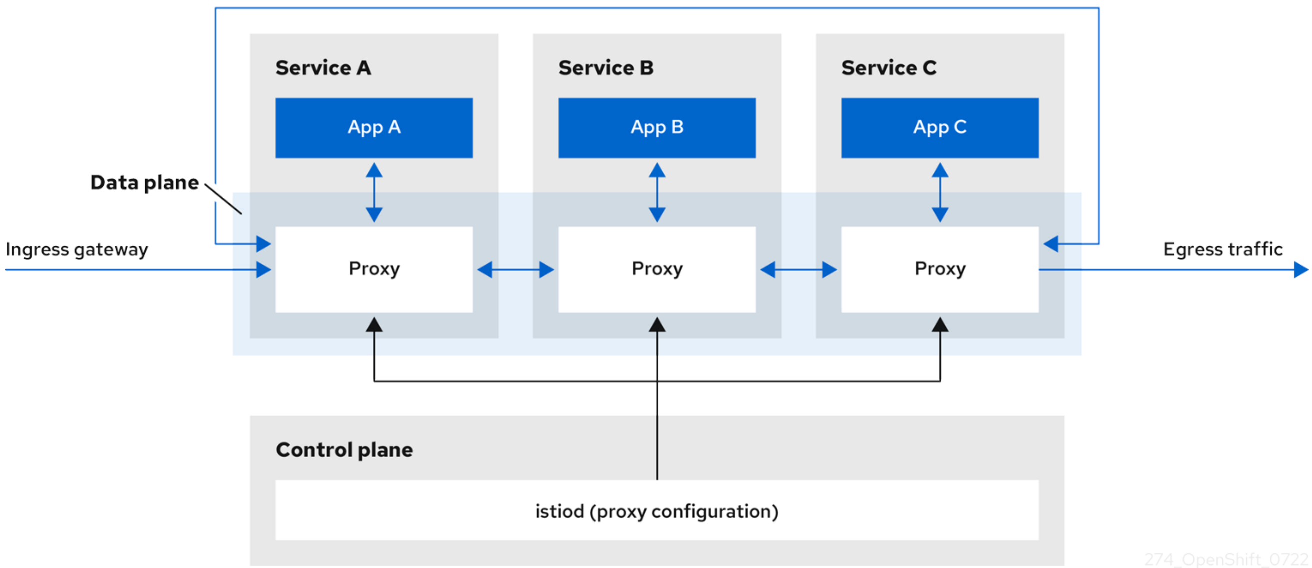
- 故障恢复
- 指标
- 监控

Red Hat OpenShift Service Mesh 还提供更复杂的操作功能，其中包括：

- A/B 测试
- Canary 发行版本
- Access control
- 端到端的验证

1.3.2. Service Mesh 架构

服务网格技术在网络通信级别运作。也就是说，服务网格组件捕获或截获进出微服务的流量，或修改请求、重定向请求或创建新请求到其他服务。



在高级别上，Red Hat OpenShift Service Mesh 由 data plane 和一个 control plane 组成

数据平面是一组智能代理，与 pod 中的应用容器一起运行，用于拦截和控制服务网格中微服务之间的所有入站和出站网络通信。数据平面的实现方式是它会截获所有入站（ingress）和出站（egress）网络流量。Istio 数据平面由与 pod 中侧应用程序容器一起运行的 Envoy 容器组成。Envoy 容器充当代理，控制与 pod 往来的所有网络通信。

- Envoy 代理 是与 data plane 流量交互的唯一 Istio 组件。服务之间的所有传入（ingress）和传出（egress）网络流量通过代理流。Envoy 代理还会收集与网格内服务流量相关的所有指标。Envoy 代理部署为 sidecar，与服务在同一个 pod 中运行。Envoy 代理也用于实现网格网关。
    - sidecar 代理 为其工作负载实例管理入站和出站通信。
    - 网关是作为接收传入或传出 HTTP/TCP 连接的负载平衡器运行的代理。网关配置适用于在网格边缘运行的独立的 Envoy 代理，而不是与您的服务负载一同运行的 sidecar Envoy 代理。您可以使用网关来管理入站和出站流量，允许您指定您要进入或离开网格的流量。
        - Ingress-gateway - 也称为入口控制器，Ingress 网关是一个专用的 Envoy 代理，用于接收和控制进入服务网格的流量。Ingress 网关允许将监控和路由规则等功能应用到进入集群的流量。
        - Egress-gateway - 另外称为出口控制器（Egress Gateway）， Egress 网关是一

个专用的 Envoy 代理，用于管理离开服务网格的流量。Egress 网关允许对流量退出网格应用监控和路由规则等功能。

control plane 管理并配置组成数据平面的代理。它是配置的权威源，管理访问控制和使用策略，并从服务网格中的代理收集指标。

- Istio control plane 由 Istiod 组成，它会将几个之前的 control plane 组件（Citadel、Galley 和 Pilot）整合为一个二进制。Istiod 提供服务发现、配置和证书管理。它将高级别路由规则转换为 Envoy 配置，并在运行时将其传播到 sidecar。
  - Istiod 可以充当证书颁发机构 (CA)，在 data plane 中生成支持安全 mTLS 通信的证书。您还可以使用外部 CA 来实现这一目的。
  - Istiod 负责将 sidecar 代理容器注入到部署到 OpenShift 集群的工作负载中。

Red Hat OpenShift Service Mesh 使用 istio-operator 来管理 control plane 的安装。*Operator* 是一个软件，它可让您实现和自动化 OpenShift 集群中的常见操作。它充当控制器，允许您设置或更改集群中对象的所需状态，本例中为 Red Hat OpenShift Service Mesh 安装。

Red Hat OpenShift Service Mesh 还捆绑以下 Istio 附加组件作为该产品的一部分：

- Kiali - Kiali 是 Red Hat OpenShift Service Mesh 的管理控制台。它提供了仪表板、可观察性以及强大的配置和验证功能。它通过推断流量拓扑显示服务网格的结构，并显示网格的健康状况。Kiali 提供详细的指标、强大的验证、访问 Grafana 以及与分布式 Tracing Platform (Jaeger)的强大集成。
- Prometheus - Red Hat OpenShift Service Mesh 使用 Prometheus 来存储来自服务的遥测信息。Kiali 依靠 Prometheus 获取指标数据、健康状况和网格拓扑。
- Jaeger - Red Hat OpenShift Service Mesh 支持分布式 Tracing Platform (Jaeger)。Jaeger 是一个开源可追踪性服务器，可以集中并显示与多个服务间单一请求关联的 trace。利用分布式跟踪平台(Jaeger)，您可以监控基于微服务的分布式系统并进行故障排除。
- Elasticsearch - Elasticsearch 是一个开源、分布式、基于 JSON 的搜索和分析引擎。分布

式跟踪平台(Jaeger)使用 Elasticsearch 进行持久性存储。

- Grafana - Grafana 为网格管理员提供用于 Istio 数据的高级查询和指标分析和仪表板。另外，Grafana 可以用来分析服务网格指标。

以下 Istio 集成与 Red Hat OpenShift Service Mesh 支持：

- 3scale - Istio 提供与红帽 3scale API 管理解决方案的可选集成。对于 2.1 之前的版本，这个集成是通过 3scale Istio 适配器实现的。对于 2.1 版本，3scale 集成通过 WebAssembly 模块实现。

有关如何安装 3scale 适配器的详情，请参考 3scale Istio 适配器文档

1.3.3. 了解 Kiali

Kiali 通过显示服务网格中的微服务服务以及连接方式，为您提供了一个可视性的服务网格概述。

1.3.3.1. Kiali 概述

Kiali 为在 Red Hat OpenShift Service on AWS 上运行的 Service Mesh 提供了一个观察平台。Kiali 可以帮助您定义、验证并观察 Istio 服务网格。它所提供的拓扑结构可以帮助您了解服务网格的结构，并提供服务网格的健康状况信息。

Kiali 实时提供命名空间的交互式图形视图，可让您了解诸如电路断路器、请求率、延迟甚至流量图等功能。Kiali 提供了从应用程序到服务以及负载等不同级别的组件的了解，并可显示与所选图形节点或边缘的上下文信息和图表的交互。Kiali 还提供了验证 Istio 配置（如网关、目的规则、虚拟服务、网格策略等等）的功能。Kiali 提供了详细的指标数据，并可使用基本的 Grafana 集成来进行高级查询。通过将 Jaeger 集成到 Kiali 控制台来提供分布式追踪。

默认情况下，Kiali 作为 Red Hat OpenShift Service Mesh 的一部分被安装。

1.3.3.2. Kiali 架构

Kiali 基于开源 Kiali 项目。Kiali 由两个组件组成: Kiali 应用程序和 Kiali 控制台。

- Kiali 应用程序 （后端）- 该组件运行在容器应用程序平台中，并与服务网格组件进行通讯，检索和处理数据，并将这些数据提供给控制台。Kiali 应用程序不需要存储。当在集群中部署应用程序时，配置在 ConfigMaps 和 secret 中设置。

- Kiali 控制台 (前端) – Kiali 控制台是一个 Web 应用程序。Kiali 应用程序为 Kiali 控制台提供服务，控制台会查询后端数据并把数据提供给用户。

另外，Kiali 依赖于由容器应用程序平台和 Istio 提供的外部服务和组件。

- Red Hat Service Mesh (Istio) - Kiali 需要 Istio。Istio 是提供和控制服务网格的组件。虽然 Kiali 和 Istio 可以单独安装，但是 Kiali 需要 Istio。如果没有安装 Istio，则无法工作。Kiali 需要检索 Istio 数据和配置，这些数据和配置可以通过 Prometheus 和集群 API 获得。

- Prometheus - 一个专用的 Prometheus 实例作为 Red Hat OpenShift Service Mesh 安装的一部分被包括。启用 Istio 遥测时，指标数据存储在 Prometheus 中。Kiali 使用这个 Prometheus 数据来决定网状拓扑结构、显示指标数据、计算健康状况、显示可能的问题等等。Kiali 与 Prometheus 直接沟通，并假设 Istio Telemetry 使用的数据 schema。Istio 依赖于 Prometheus，Kiali 也依赖于 Prometheus。许多 Kiali 的功能在没有 Prometheus 的情况下将无法工作。

- Cluster API - Kiali 使用 Red Hat OpenShift Service on AWS (cluster API) API 来获取和解析服务网格配置。Kiali 通过查询集群 API 获取信息，如获取命名空间、服务、部署、pod 和其他实体的定义。Kiali 还提供查询来解析不同集群实体之间的关系。另外，还可以通过查询集群 API 以获取 Istio 配置，比如虚拟服务、目的规则、路由规则、网关、配额等等。

- Jaeger - Jaeger 是可选的，但会作为 Red Hat OpenShift Service Mesh 安装的一部分被默认安装。当您作为默认 Red Hat OpenShift Service Mesh 安装的一部分安装 Distributed Tracing Platform (Jaeger)时，Kiali 控制台会包括一个显示分布式追踪数据的标签页。请注意：如果禁用 Istio 的分布式追踪功能，则不会提供追踪数据。另请注意，用户必须可以访问安装 Service Mesh control plane 的命名空间，才能查看追踪数据。

- Grafana - Grafana 是可选的，但作为 Red Hat OpenShift Service Mesh 安装的一部分被默认安装。如果使用了 Grafana，Kiali 的 metrics 页会包括一个链接，用户可以使用它访问 Grafana 中相同的指标数据。请注意，用户必须可以访问安装 Service Mesh control plane 的命名空间，以便查看到 Grafana 仪表板的链接并查看 Grafana 数据。

1.3.3.3. Kiali 的功能

Kiali 控制台与 Red Hat Service Mesh 集成，提供以下功能：

- 健康 – 快速识别应用程序、服务或者工作负载的问题。

- 拓扑 – 以图形的形式显示应用程序、服务或工作负载如何通过 Kiali 进行通信。

- 指标 – 预定义的 metrics dashboard 可为您生成 Go、Node.js、Quarkus 、Spring Boot 、Thonttail 和 Vert.x 的服务网格和应用程序性能图表。。您还可以创建您自己的自定义仪表板。

- 追踪 – 通过与 Jaeger 集成，可以在组成一个应用程序的多个微服务间追踪请求的路径。

- 验证 – 对最常见 Istio 对象（Destination Rules 、Service Entries 、Virtual Services 等等）进行高级验证。

- 配置 – 使用向导创建、更新和删除 Istio 路由配置的可选功能，或者直接在 Kiali Console 的 YAML 编辑器中创建、更新和删除 Istio 路由配置。

1.3.4. 了解分布式追踪

每次用户在某个应用程序中执行一项操作时，一个请求都会在所在的系统上执行，而这个系统可能需要几十个不同服务的共同参与才可以做出相应的响应。这个请求的路径是一个分布式的事务。分布式跟踪平台(Jaeger)可让您执行分布式追踪，它会在组成一个应用程序的多个微服务间追踪请求的路径。

分布式追踪是用来将不同工作单元的信息关联起来的技术，通常是在不同进程或主机中执行的，以便理解分布式事务中的整个事件链。分布式追踪可让开发人员在大型服务架构中视觉化调用流程。它对理解序列化、平行和延迟来源会很有价值。

分布式追踪平台(Jaeger)记录了在微服务的整个堆栈间执行单个请求，并将其显示为 trace。trace是系统的数据/执行路径。端到端追踪包含一个或多个范围。

span 代表具有操作名称、操作的开始时间和持续时间的逻辑工作单元。span 可能会被嵌套并排序以

模拟因果关系。

1.3.4.1. 分布式追踪概述

作为服务所有者，您可以使用分布式追踪来检测您的服务，以收集与服务架构相关的信息。您可以使用 Red Hat OpenShift distributed Tracing Platform 来监控、网络性能分析，并对现代、云原生的基于微服务的应用程序中组件之间的交互进行故障排除。

使用分布式跟踪平台，您可以执行以下功能：

- 监控分布式事务
- 优化性能和延迟时间
- 执行根原因分析

1.3.4.2. Red Hat OpenShift distributed Tracing Platform 架构

Red Hat OpenShift distributed Tracing Platform 由几个组件组成，它们一起收集、存储和显示追踪数据。

- Red Hat OpenShift distributed Tracing Platform - 此组件基于开源 Grafana Tempo 项目。
  - 网关 - 网关处理身份验证、授权和将请求转发到分布式或查询前端服务。
  - 经销商 - Distributor 接受多种格式（包括 Jaeger、OpenTelemetry 和 Zipkin）的 span。它通过哈希 traceID 并将分布式一致的哈希环路由到 Ingester。
  - Ingester - Ingester 将 trace 批处理到块中，创建 bloom 过滤器和索引，然后将其全部刷新到后端。

  - 查询 Frontend - Query Frontend 负责为传入的查询对搜索空间进行分片。然后，搜索查询会发送到 Queriers。Query Frontend 部署通过 Tempo Query sidecar 公开 Jaeger UI。

  - Querier - Querier 负责在 Ingester 或后端存储中查找请求的 trace ID。根据参数，它可以查询 Ingesters，并从后端拉取 Bloom 索引，以便在对象存储中搜索块。

  - compactor - Compactors 流块到后端存储中，以减少块总数。

- 红帽构建的 OpenTelemetry - 此组件基于开源 OpenTelemetry 项目。

  - OpenTelemetry Collector - OpenTelemetry Collector 是一个与厂商无关的方式来接收、处理和导出遥测数据。OpenTelemetry Collector 支持开源可观察数据格式，如 Jaeger 和 Prometheus，发送到一个或多个开源或商业后端。Collector 是默认位置检测库来导出其遥测数据。

- Red Hat OpenShift distributed Tracing Platform (Jaeger) - 此组件基于开源 Jaeger 项目。

重要

Red Hat OpenShift distributed Tracing Platform (Jaeger)是一个已弃用的功能。弃用的功能仍然包含在 Red Hat OpenShift Service on AWS 中，并且仍然被支持。但是，弃用的功能可能会在以后的发行版本中被删除，且不建议在新的部署中使用。

Red Hat OpenShift distributed Tracing Platform Operator (Jaeger)将在以后的版本中从 redhat-operators 目录中删除。如需更多信息，请参阅 OpenShift 中的红帽知识库解决方案 Jaeger Deprecation 和 Removal。

用户必须迁移到 Tempo Operator，以及用于分布式追踪集合和存储的红帽 OpenTelemetry 构建。

  - 客户端 (Jaeger 客户端、跟踪器、报告程序、客户端库)- 分布式跟踪平台(Jaeger)客户

端是 OpenTracing API 的特定语言实现。它们可用于检测应用程序手动或各种现有开源框架（如 Camel (Fuse)、Spring Boot (RHOAR)、MicroProfile (RHOAR/Thorntail)、Wild (EAP)等。

- 代理 (Jaeger 代理, Server Queue, Processor Workers)- 分布式 Tracing Platform (Jaeger)代理是一个网络守护进程，侦听通过用户数据报协议(UDP)发送并发送到 Collector 的 span。这个代理应被放置在要管理的应用程序的同一主机上。这通常是通过容器环境（如 Kubernetes）中的 sidecar 来实现。

- Jaeger Collector (Collector, Queue, Workers)- 与 Jaeger 代理类似，Jaeger Collector 接收 span，并将它们放置在内部队列中进行处理。这允许 Jaeger Collector 立即返回到客户端/代理，而不是等待 span 变为存储。

- Storage (Data Store) - 收集器需要一个持久的存储后端。Red Hat OpenShift distributed Tracing Platform (Jaeger)具有用于 span 存储的可插拔机制。Red Hat OpenShift distributed Tracing Platform (Jaeger)支持 Elasticsearch 存储。

- Query (Query Service) - Query 是一个从存储中检索 trace 的服务。

- Ingester (Ingester Service)- Red Hat OpenShift distributed Tracing Platform 可以使用 Apache Kafka 作为 Collector 和实际的 Elasticsearch 后端存储之间的缓冲。Ingester 是一个从 Kafka 读取数据并写入 Elasticsearch 存储后端的服务。

- Jaeger 控制台 - 使用 Red Hat OpenShift distributed Tracing Platform (Jaeger)用户界面，您可以视觉化您的分布式追踪数据。在搜索页面中，您可以查找 trace，并查看组成一个独立 trace 的 span 详情。

1.3.4.3. Red Hat OpenShift distributed Tracing Platform 功能

Red Hat OpenShift distributed Tracing Platform 提供以下功能：

- 与 Kiali 集成 - 当正确配置时，您可以从 Kiali 控制台查看分布式跟踪平台数据。

- 高可扩展性 - 分布式追踪平台后端旨在没有单点故障，并能够按照业务需求进行扩展。

- 分布式上下文发布 – 允许您通过不同的组件连接数据以创建完整的端到端的 trace。

- 与 Zipkin 的后向兼容性 - Red Hat OpenShift distributed Tracing Platform 有 API，它被用来替代 Zipkin，但红帽在此发行版本中不支持 Zipkin 的兼容性。

1.3.5. 后续步骤

- 准备在 Red Hat OpenShift Service on AWS 环境中安装 Red Hat OpenShift Service Mesh。

1.4. 服务网格部署模型

Red Hat OpenShift Service Mesh 支持几种不同的部署模型，它们可以以不同的方式组合以满足您的业务需求。

在 Istio 中，租户是为一组部署的工作负载共享共同访问权限和特权的用户组。您可以使用租户在不同的团队之间提供一定程度的隔离。您可以使用 istio.io 或服务资源的 NetworkPolicies、AuthorizationPolicies 和 exportTo 注解来隔离对不同租户的访问。

1.4.1. Cluster-Wide (Single Tenant)网格部署模型

集群范围的部署包含一个 Service Mesh Control Plane，用于监控整个集群的资源。监控整个集群的资源与 control plane 在所有命名空间中使用单个查询来监控 Istio 和 Kubernetes 资源的 Istio 功能非常相似。因此，集群范围的部署会减少发送到 API 服务器的请求数。

与 Istio 类似，集群范围的网格默认包括带有 istio-injection=enabled 命名空间标签的命名空间。您可以通过修改 ServiceMeshMemberRoll 资源的 spec.memberSelectors 字段来更改此标签。

1.4.2. 多租户部署模型

Red Hat OpenShift Service Mesh 安装了一个 ServiceMeshControlPlane，它默认配置为多租户。Red Hat OpenShift Service Mesh 使用多租户 Operator 来管理 Service Mesh control plane 生命周期。在网格内，命名空间用于租期。

Red Hat OpenShift Service Mesh 使用 ServiceMeshControlPlane 资源来管理网格安装，该安装范

围默认限制为包含资源的命名空间。您可以使用 ServiceMeshMemberRoll 和 ServiceMeshMember 资源在网格中包含额外的命名空间。命名空间只能包含在单个网格中，多个网格也可以安装到单个 OpenShift 集群中。

典型的服务网格部署使用单一 Service Mesh control plane 来配置网格中服务间的通信。Red Hat OpenShift Service Mesh 支持"软多租户"，其中每个租户有一个 control plane 和一个网格，并且集群中可以有多个独立的 control plane。多租户部署指定可以访问 Service Mesh 的项目，并将 Service Mesh 与其他 control plane 实例隔离。

集群管理员在所有 Istio control plane 间获得控制和可见性，而租户管理员只能控制其特定的 Service Mesh、Kiali 和 Jaeger 实例。

您可以授予团队权限，以便仅将工作负载部署到给定的命名空间或一组命名空间。如果服务网格管理员授予 mesh-user 角色，用户可以创建一个 ServiceMeshMember 资源来将命名空间添加到 ServiceMeshMemberRoll。

1.4.2.1. 关于迁移到集群范围的网格

在集群范围的网格中，一个 ServiceMeshControlPlane (SMCP)会监视整个集群的所有命名空间。您可以使用 Red Hat OpenShift Service Mesh 版本 2.5 或更高版本将现有集群从多租户网格迁移到集群范围的网格。

注意

如果集群必须有多个 SMCP，则无法迁移到集群范围的网格。

默认情况下，集群范围的网格会发现组成集群的所有命名空间。但是，您可以将网格配置为访问有限的命名空间集合。命名空间默认不接收 sidecar 注入。您必须指定哪些命名空间接收 sidecar 注入。

同样，您必须指定哪些 pod 接收 sidecar 注入。接收 sidecar 注入的命名空间中的 Pod 不会继承 sidecar 注入。将 sidecar 注入应用到命名空间和 pod 是独立的操作。

如果您在迁移到集群范围的网格时更改了 Istio 版本，则必须重启应用程序。如果您使用相同的 Istio 版本，应用程序代理将连接到集群范围的网格的新 SMCP，并像多租户网格一样工作。

1.4.2.1.1. 使用 Web 控制台从集群范围的网格中包括和排除命名空间

使用 Red Hat OpenShift Service on AWS Web 控制台，您可以在集群范围的网格中的 ServiceMeshControlPlane 资源中添加发现选择器。发现选择器定义 control plane 可以发现的命名空间。control plane 忽略任何与其中一个发现选择器不匹配的命名空间，这会从网格中排除命名空间。

注意

如果在 control plane 命名空间中安装入口或出口网关，则必须在发现选择器中包含 control plane 命名空间。

先决条件

- 已安装 Red Hat OpenShift Service Mesh Operator。
- 您已部署了 ServiceMeshControlPlane 资源。
- 以具有 cluster-admin 角色的用户身份登录。如果使用 Red Hat OpenShift Dedicated，则以具有 dedicated-admin 角色的用户身份登录。

流程

1. 登陆到 Red Hat OpenShift Service on AWS web 控制台。
2. 导航到 Operators → Installed Operators。
3. 点 Red Hat OpenShift Service Mesh Operator。
4. 点 Istio Service Mesh Control Plane。
5. 点 control plane 的名称。

6. 点 YAML。

7. 修改 YAML 文件，以便 ServiceMeshControlPlane 资源的 spec.meshConfig 字段包含发现选择器。

注意

在配置 Istiod 服务可以发现的命名空间时，排除可能包含不应暴露给网格的敏感服务的命名空间。

在以下示例中，Istiod 服务发现任何标记为 istio-discovery: enabled，或具有名称 info、httpbin 或 istio-system 的命名空间：

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  mode: ClusterWide
  meshConfig:
    discoverySelectors:
    - matchLabels:
        istio-discovery: enabled (1)
    - matchExpressions:
      - key: kubernetes.io/metadata.name (2)
        operator: In
        values:
        - info
        - httpbin
        - istio-system
```

1. 确保网格发现包含标签 istio-discovery: enabled 的命名空间。

2. 确保 mesh 发现命名空间的 info, httpbin 和 istio-system。

如果命名空间与任何发现选择器匹配，则网格会发现命名空间。网格排除与任何发现选择器不匹配的命名空间。

8. 保存该文件。

1.4.2.1.2. 使用 CLI 从集群范围的网格中包括和排除命名空间

使用 Red Hat OpenShift Service on AWS CLI，您可以将发现选择器添加到集群范围的网格中的 ServiceMeshControlPlane 资源中。发现选择器定义 control plane 可以发现的命名空间。control plane 忽略任何与其中一个发现选择器不匹配的命名空间，这会从网格中排除命名空间。

> **注意**
>
> 如果在 control plane 命名空间中安装入口或出口网关，则必须在发现选择器中包含 control plane 命名空间。

先决条件

- 已安装 Red Hat OpenShift Service Mesh Operator。
- 您已部署了 ServiceMeshControlPlane 资源。
- 以具有 cluster-admin 角色的用户身份登录。如果使用 Red Hat OpenShift Dedicated，则以具有 dedicated-admin 角色的用户身份登录。

流程

1. 登陆到 Red Hat OpenShift Service on AWS CLI。
2. 运行以下命令，以 YAML 文件形式打开 ServiceMeshControlPlane 资源：

```
$ oc -n istio-system edit smcp <name> (1)
```

(1) <name> 代表 ServiceMeshControlPlane 资源的名称。

3. 修改 YAML 文件，以便 ServiceMeshControlPlane 资源的 spec.meshConfig 字段包含发现选择器。

注意

在配置 Istiod 服务可以发现的命名空间时，排除可能包含不应暴露给网格的敏感服务的命名空间。

在以下示例中，Istiod 服务发现任何标记为 istio-discovery: enabled 或具有名称 info、httpbin 或 istio-system 的命名空间：

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  mode: ClusterWide
  meshConfig:
    discoverySelectors:
    - matchLabels:
        istio-discovery: enabled (1)
    - matchExpressions:
      - key: kubernetes.io/metadata.name (2)
        operator: In
        values:
        - info
        - httpbin
        - istio-system
```

1. 确保网格发现包含标签 istio-discovery: enabled 的命名空间。

2. 确保网格发现命名空间信息、httpbin 和 istio-system。

如果命名空间与任何发现选择器匹配，则网格会发现命名空间。网格排除与任何发现选择器不匹配的命名空间。

4. 保存文件并退出编辑器。

1.4.2.1.3. 使用 Web 控制台在集群范围的网格中定义哪些命名空间接收 sidecar 注入

默认情况下，Red Hat OpenShift Service Mesh Operator 使用成员选择器来识别哪些命名空间接收 sidecar 注入。与 ServiceMeshMemberRoll 资源中定义的 istio-injection=enabled 标签不匹配的命名空间不会接收 sidecar 注入。

注意

使用发现选择器来决定网格可以发现哪些命名空间对 sidecar 注入没有影响。发现命名空间并配置 sidecar 注入是独立的操作。

前提条件

- 已安装 Red Hat OpenShift Service Mesh Operator。
- 您已部署了带有 mode: ClusterWide 注解的 ServiceMeshControlPlanae 资源。
- 以具有 cluster-admin 角色的用户身份登录。如果使用 Red Hat OpenShift Dedicated，则以具有 dedicated-admin 角色的用户身份登录。

流程

1. 登陆到 Red Hat OpenShift Service on AWS web 控制台。
2. 导航到 Operators → Installed Operators。
3. 点 Red Hat OpenShift Service Mesh Operator。
4. 点 Istio Service Mesh Member Roll。

5. 点 ServiceMeshMemberRoll 资源。

6. 点 YAML。

7. 通过添加与 注入 标签匹配的成员选择器，修改 ServiceMeshMemberRoll 资源中的 spec.memberSelectors 字段。以下示例使用 istio-injection: enabled ：

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
spec:
  memberSelectors:
  - matchLabels:
      istio-injection: enabled (1)
```

(1) 确保命名空间接收 sidecar 注入。

8. 保存该文件。

1.4.2.1.4. 使用 CLI 定义哪些命名空间在集群范围的网格中接收 sidecar 注入

默认情况下，Red Hat OpenShift Service Mesh Operator 使用成员选择器来识别哪些命名空间接收 sidecar 注入。与 ServiceMeshMemberRoll 资源中定义的 istio-injection=enabled 标签不匹配的命名空间不会接收 sidecar 注入。

注意

使用发现选择器来决定网格可以发现哪些命名空间对 sidecar 注入没有影响。发现命名空间并配置 sidecar 注入是独立的操作。

先决条件

- 已安装 Red Hat OpenShift Service Mesh Operator。

- 您已部署了带有 mode: ClusterWide 注解的 ServiceMeshControlPlanae 资源。

- 以具有 cluster-admin 角色的用户身份登录。如果使用 Red Hat OpenShift Dedicated，则以具有 dedicated-admin 角色的用户身份登录。

流程

1. 登陆到 Red Hat OpenShift Service on AWS CLI。

2. 编辑 ServiceMeshMemberRoll 资源。

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

3. 通过添加与 inject 标签匹配的成员选择器，修改 ServiceMeshMemberRoll 资源中的 spec.memberSelectors 字段。以下示例使用 istio-injection: enabled ：

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
spec:
  memberSelectors:
  - matchLabels:
      istio-injection: enabled 1
```

1 确保命名空间接收 sidecar 注入。

4. 保存文件并退出编辑器。

1.4.2.1.5. 使用 Web 控制台从集群范围的网格中排除单个 pod

如果应用 sidecar.istio.io/inject: true 注解，pod 会接收 sidecar 注入，并且 pod 存在于与标签选择器或 ServiceMeshMemberRoll 资源中定义的成员列表匹配的命名空间中。

如果 pod 没有应用 sidecar.istio.io/inject 注解，则无法接收 sidecar 注入。

先决条件

- 已安装 Red Hat OpenShift Service Mesh Operator。
- 您已使用 mode: ClusterWide 注解部署了 ServiceMeshControlPlane 资源。
- 以具有 cluster-admin 角色的用户身份登录。如果使用 Red Hat OpenShift Dedicated，则以具有 dedicated-admin 角色的用户身份登录。

流程

1. 登陆到 Red Hat OpenShift Service on AWS web 控制台。
2. 进入到 Workloads → Deployments。
3. 点部署的名称。
4. 点 YAML。
5. 修改 YAML 文件，以部署一个接收 sidecar 注入的应用程序，以及一个没有设置的应用程序，如下例所示：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: 'true' 1
```

```yaml
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-without-sidecar
spec:
  selector:
    matchLabels:
      app: nginx-without-sidecar
  template:
    metadata:
      labels:
        app: nginx-without-sidecar # 2
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

1 此 pod 应用了 sidecar.istio.io/inject 注解，因此它接收 sidecar 注入。

2 此 pod 没有注解，因此它不会接收 sidecar 注入。

6. 保存该文件。

1.4.2.1.6. 使用 CLI 从集群范围的网格中排除单个 pod

如果应用 sidecar.istio.io/inject: true 注解，pod 会接收 sidecar 注入，并且 pod 存在于与标签选择器或 ServiceMeshMemberRoll 资源中定义的成员列表匹配的命名空间中。

如果 pod 没有应用 sidecar.istio.io/inject 注解，则无法接收 sidecar 注入。

先决条件

- 已安装 Red Hat OpenShift Service Mesh Operator。
- 您已使用 mode: ClusterWide 注解部署了 ServiceMeshControlPlane 资源。
- 以具有 cluster-admin 角色的用户身份登录。如果使用 Red Hat OpenShift Dedicated，则以具有 dedicated-admin 角色的用户身份登录。

流程

1. 登陆到 Red Hat OpenShift Service on AWS CLI。
2. 运行以下命令来编辑部署：

```
$ oc edit deployment -n <namespace> <deploymentName>
```

3. 修改 YAML 文件，以部署一个接收 sidecar 注入的应用程序，以及一个没有设置的应用程序，如下例所示：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: 'true' 1
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
---
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: nginx-without-sidecar
spec:
  selector:
    matchLabels:
      app: nginx-without-sidecar
  template:
    metadata:
      labels:
        app: nginx-without-sidecar 2
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

1. 此 pod 应用了 sidecar.istio.io/inject 注解，因此它接收 sidecar 注入。

2. 此 pod 没有注解，因此它不会接收 sidecar 注入。

4. 保存该文件。

1.4.3. Multimesh 或联邦部署模型

*Federation（联邦）*是一种部署模型，可让您在不同管理域中管理的单独网格间共享服务和工作负载。

Istio 多集群模型需要在网格和远程访问独立网格所在的所有 Kubernetes API 服务器之间具有高度信任。Red Hat OpenShift Service Mesh 联邦针对 Service Mesh 的多集群实施，该方法假设网格之间的信任*最小*。

*联邦网格（federated mesh）*是作为单个网格组成的一组网格。每个网格中的服务可以是独特的服务，例如通过从另一个网格中导入服务的网格添加服务，可以为网格中的相同服务提供额外的工作负载，提供高可用性或两者的组合。加入联邦的所有网格都保持单独管理，您必须明确配置要导出哪些服务并从联邦中的其他网格导入。证书生成、指标和追踪集合等支持功能在其各自网格中保持本地。

1.5. SERVICE MESH 和 ISTIO 的不同

Red Hat OpenShift Service Mesh 与 Istio 安装的不同之处在于提供额外功能或在 Red Hat OpenShift Service on AWS 上部署时处理不同之处。

1.5.1. Istio 和 Red Hat OpenShift Service Mesh 之间的区别

Service Mesh 和 Istio 中的以下功能不同。

1.5.1.1. 命令行工具

Red Hat OpenShift Service Mesh 的命令行工具是 oc。 Red Hat OpenShift Service Mesh 不支持 istioctl。

1.5.1.2. 安装和升级

Red Hat OpenShift Service Mesh 不支持 Istio 安装配置集。

Red Hat OpenShift Service Mesh 不支持 service mesh 的 Canary 升级。

1.5.1.3. 自动注入

上游 Istio 社区安装会在您标记的项目中自动将 sidecar 注入 pod。

Red Hat OpenShift Service Mesh 不会自动将 sidecar 注入任何 pod，而是要求您选择使用没有标记项目的注解注入。这个方法需要较少的权限，且不会与其他 Red Hat OpenShift Service on AWS 功能冲突，如 builder pod。要启用自动注入，请指定 sidecar.istio.io/inject 标签或注解，如*自动 sidecar 注入*部分所述。

表 1.3. sidecar 注入标签和注解设置

	上游 Istio	Red Hat OpenShift Service Mesh
命名空间标签	支持"启用"和"禁用"	支持"禁用"
Pod 标签	支持 "true" 和 "false"	支持 "true" 和 "false"

	上游 Istio	Red Hat OpenShift Service Mesh
Pod 注解	只支持 "false"	支持 "true" 和 "false"

1.5.1.4. Istio 基于角色的访问控制功能

Istio 基于角色的访问控制 (RBAC) 提供了可用来控制对某个服务的访问控制机制。您可以根据用户名或者指定一组属性来识别对象，并相应地应用访问控制。

上游 Istio 社区安装提供的选项包括：标头精确匹配、匹配标头中的通配符，或匹配标头中包括的特定前缀或后缀。

Red Hat OpenShift Service Mesh 使用正则表达式来扩展与请求标头匹配的功能。使用正则表达式指定 request.regex.headers 的属性键。

上游 Istio 社区匹配请求标头示例

```
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
```

1.5.1.5. OpenSSL

Red Hat OpenShift Service Mesh 将 BoringSSL 替换为 OpenSSL。OpenSSL 是包含安全套接字层 (SSL) 和传输层 (TLS) 协议的开源实现的软件库。Red Hat OpenShift Service Mesh Proxy 二进制代

码动态地将 OpenSSL 库（libssl 和 libcrypto）与底层的 Red Hat Enterprise Linux 操作系统进行链接。

1.5.1.6. 外部工作负载

Red Hat OpenShift Service Mesh 不支持外部工作负载，如在裸机服务器上运行的虚拟机。

1.5.1.7. 虚拟机支持

您可以使用 OpenShift Virtualization 将虚拟机部署到 OpenShift。然后，您可以将网格策略（如 mTLS 或 AuthorizationPolicy）应用到这些虚拟机，就像其它属于网格的 pod 一样。

1.5.1.8. 组件修改

- *maistra-version* 标签已添加到所有资源中。
- 所有 Ingress 资源都已转换为 OpenShift Route 资源。
- Grafana、分布式追踪(Jaeger)和 Kiali 会被默认启用，并通过 OpenShift 路由公开。
- Godebug 已从所有模板中删除
- istio-multi ServiceAccount 和 ClusterRoleBinding 已被删除，同时也删除了 istio-reader ClusterRole。

1.5.1.9. Envoy 过滤器

Red Hat OpenShift Service Mesh 不支持 EnvoyFilter 配置，除非明确记录。由于与底层 Envoy API 紧密耦合，因此无法保持向后兼容性。EnvoyFilter 补丁对 Istio 生成的 Envoy 配置格式非常敏感。如果 Istio 生成的配置有变化，则代表可能会破坏 EnvoyFilter 的应用程序。

1.5.1.10. Envoy 服务

Red Hat OpenShift Service Mesh 不支持基于 QUIC 的服务。

1.5.1.11. Istio Container Network Interface (CNI) 插件

Red Hat OpenShift Service Mesh 包括 CNI 插件，它为您提供了配置应用程序 pod 网络的替代方法。CNI 插件替代了 init-container 网络配置，可在不需要提高访问权限的情况下赋予服务帐户和项目对安全上下文约束 (SCC) 的访问。

> **注意**
>
> 默认情况下，Istio Container Network Interface (CNI) pod 在所有 Red Hat OpenShift Service on AWS 节点上创建。要排除在特定节点中创建 CNI pod，请将 maistra.io/exclude-cni=true 标签应用到节点。添加此标签会从节点中删除之前部署的 Istio CNI pod。

1.5.1.12. 全局 mTLS 设置

Red Hat OpenShift Service Mesh 创建一个 PeerAuthentication 资源，在网格内启用或禁用 Mutual TLS 身份验证 (mTLS)。

1.5.1.13. 网关

Red Hat OpenShift Service Mesh 默认安装入口和出口网关。您可以使用以下设置在 ServiceMeshControlPlane (SMCP) 资源中禁用网关安装：

- spec.gateways.enabled=false 可禁用入口和出口网关。
- spec.gateways.ingress.enabled=false 禁用入口网关。
- spec.gateways.egress.enabled=false 禁用出口网关。

> **注意**
>
> Operator 注解了默认网关，以指示它们由 Red Hat OpenShift Service Mesh Operator 生成并管理。

1.5.1.14. 多集群配置

Red Hat OpenShift Service Mesh 对多集群配置的支持仅限于跨多个集群的服务网格的联邦。

1.5.1.15. 自定义证书签名请求 (CSR)

您无法将 Red Hat OpenShift Service Mesh 配置为通过 Kubernetes 证书颁发机构 (CA) 处理 CSR。

1.5.1.16. Istio 网关的路由

Istio 网关的 OpenShift 路由在 Red Hat OpenShift Service Mesh 中被自动管理。每次在 service mesh 中创建、更新或删除 Istio 网关时，都会自动创建、更新或删除 OpenShift 路由。

名为 Istio OpenShift Routing (IOR) 的 Red Hat OpenShift Service Mesh control plane 组件可以用来同步网关路由。如需更多信息，请参阅自动路由创建。

1.5.1.16.1. catch-all 域

不支持 Catch-all（"*"）。如果在网关定义中找到一个，Red Hat OpenShift Service Mesh *将*创建路由，但会依赖于 OpenShift 来创建一个默认主机名。这意味着新创建的路由*不是* catch all ("*") 路由，而是使用 <route-name> [-<project>].<suffix> 格式的主机名。如需有关默认主机名的工作方式以及 cluster-admin 如何自定义它的更多信息，请参阅 Red Hat OpenShift Service on AWS 文档。如果使用 Red Hat OpenShift Dedicated，请参阅 Red Hat OpenShift Dedicated 的 dedicated-admin 角色。

1.5.1.16.2. 子域

支持子域（例如："*.domain.com"）。但是，Red Hat OpenShift Service on AWS 中不默认启用此功能。这意味着，Red Hat OpenShift Service Mesh *将*使用子域创建路由，但只有在 Red Hat OpenShift Service on AWS 被配置为启用它时才有效。

1.5.1.16.3. 传输层安全性

支持传输层安全性（TLS）。这意味着，如果网关包含 tls 部分，OpenShift Route 将配置为支持 TLS。

其他资源

- 自动路由创建

1.5.2. 多租户安装

上游 Istio 采用单一租户方法，Red Hat OpenShift Service Mesh 支持集群中的多个独立的 control plane。Red Hat OpenShift Service Mesh 使用多租户 Operator 来管理 control plane 生命周期。

Red Hat OpenShift Service Mesh 默认安装多租户 control plane。您可以指定可以访问 Service Mesh 的项目，并将 Service Mesh 与其他 control plane 实例隔离。

1.5.2.1. 多租户和集群范围的安装

多租户安装和集群范围安装之间的主要区别在于使用的权限范围。组件不再使用集群范围的 Role Based Access Control（RBAC）资源 ClusterRoleBinding。

ServiceMeshMemberRoll members 列表中的每个项目都将为每个与 control plane 部署关联的服务帐户都有一个 RoleBinding，每个 control plane 部署只会监视这些成员项目。每个成员项目都有一个 maistra.io/member-of 标签，其中 member-of 值是包含 control plane 安装的项目。

Red Hat OpenShift Service Mesh 配置每个成员项目，通过在每个成员项目中创建 NetworkPolicy 资源来确保自身、control plane 和其他成员项目之间的网络访问，允许其他成员和 control plane 到所有 pod 的入口流量。如果从 Service Mesh 中删除了一个成员，则这个 NetworkPolicy 资源会从项目中删除。

注意

这也限制了到成员项目的入站网络数据。如果需要来自非成员项目的入站网络数据，则需要创建一个 NetworkPolicy 来允许这些流量通过。

1.5.2.2. 集群范围内的资源

上游 Istio 会依赖于两个集群范围的资源。MeshPolicy 和 ClusterRbacConfig。它们与多租户集群不兼容并已被替换，如下所述。

- *ServiceMeshPolicy* 替换了用于配置 control-plane-wide 验证策略的 MeshPolicy。这必须与 control plane 在同一个项目中创建。

- *ServicemeshRbacConfig* 替换 ClusterRbacConfig 以配置基于 control-plane 范围角色的访问控制。这必须与 control plane 在同一个项目中创建。

1.5.3. Kiali 和服务网格

通过 Red Hat OpenShift Service on AWS 上的 Service Mesh 安装 Kiali 与社区 Kiali 安装不同。当在 Red Hat OpenShift Service on AWS 上部署时，为了解决问题、提供额外功能或处理不同之处，对 Red Hat OpenShift Service Mesh 的修改有时是必须的。

- Kiali 已被默认启用。

- 默认启用 Ingress 。

- 对 Kiali ConfigMap 进行了更新。

- 对 Kiali 的 ClusterRole 设置进行了更新。

- 不要编辑 ConfigMap，因为您的更改可能会被 Service Mesh 或 Kiali Operator 覆盖。Kiali Operator 管理的文件有 kiali.io/ 标签或注解。更新 Operator 文件应仅限于具有 cluster-admin 权限的用户。如果使用 Red Hat OpenShift Dedicated，则更新 Operator 文件应该仅限于具有 dedicated-admin 权限的用户。

1.5.4. 分布式追踪和服务网格

使用 Red Hat OpenShift Service on AWS 上的 Service Mesh 安装分布式 Tracing Platform (Jaeger)与社区 Jaeger 安装不同。当在 Red Hat OpenShift Service on AWS 上部署时，为了解决问题、提供额外功能或处理不同之处，对 Red Hat OpenShift Service Mesh 的修改有时是必须的。

- Service Mesh 默认启用分布式追踪。
- 为 Service Mesh 默认启用 ingress 。
- Zipkin 端口名称已改为 jaeger-collector-zipkin（从 http）
- 当选择 production 或 streaming 部署选项时，Jaeger 会默认使用 Elasticsearch 作为存储。
- Istio 的社区版本提供了一个通用的 “tracing” 路由。Red Hat OpenShift Service Mesh 使用由 Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 安装的 "jaeger" 路由，并已受到 OAuth 的保护。
- Red Hat OpenShift Service Mesh 为 Envoy proxy 使用 sidecar，Jaeger 也为 Jaeger agent 使用 sidecar。这两个 sidecar 是单独配置的，不应该相互混淆。proxy sidecar 会创建和 pod 的入站和出站相关的 span。agent sidecar 收到应用程序提供的 span ，并将其发送到 Jaeger 收集器。

1.6. 准备安装 SERVICE MESH

在安装 Red Hat OpenShift Service Mesh 之前，您必须订阅 Red Hat OpenShift Service on AWS 并在支持的配置中安装 Red Hat OpenShift Service on AWS。

1.6.1. 前提条件

- 您的红帽帐户上有有效的 Red Hat OpenShift Service on AWS 订阅。如果您没有相关订阅，请联络您的销售代表以获得更多信息。
- 安装与 Red Hat OpenShift Service on AWS 版本匹配的 Red Hat OpenShift Service on AWS 命令行工具(oc 客户端工具)，并将其添加到您的路径中。

如需有关 Red Hat OpenShift Service Mesh 生命周期和支持的平台的更多信息，请参阅支持政策。

1.6.2. 支持的配置

Red Hat OpenShift Service Mesh 当前发行版本支持以下配置。

1.6.2.1. 支持的平台

Red Hat OpenShift Service Mesh Operator 支持 ServiceMeshControlPlane 资源的多个版本。在以下平台版本中支持以下 2.6 Service Mesh control plane：

- Red Hat OpenShift Container Platform 版本 4.10 或更高版本
- Red Hat OpenShift Dedicated 版本 4
- Azure Red Hat OpenShift (ARO) 版本 4
- Red Hat OpenShift Service on AWS (ROSA)

1.6.2.2. 不支持的配置

明确不支持的情形包括：

- OpenShift Online 不支持 Red Hat OpenShift Service Mesh。
- Red Hat OpenShift Service Mesh 不支持在 Service Mesh 集群外部管理微服务。

1.6.2.3. 支持的网络配置

Red Hat OpenShift Service Mesh 支持以下网络配置。

- OpenShift-SDN

- OVN-Kubernetes 在所有支持的 Red Hat OpenShift Service on AWS 版本中可用。

- 在 Red Hat OpenShift Service on AWS 上认证并通过 Service Mesh 一致性测试的第三方 Container Network Interface (CNI)插件。如需更多信息，请参阅认证的 OpenShift CNI 插件。

1.6.2.4. Service Mesh 支持的配置

- 此 Red Hat OpenShift Service Mesh 发行版本仅适用于 Red Hat OpenShift Service on AWS x86_64。

- 将所有 Service Mesh 组件包含在单个 Red Hat OpenShift Service on AWS 集群中的配置。

- 不集成外部服务的配置，如虚拟机。

- Red Hat OpenShift Service Mesh 不支持 EnvoyFilter 配置，除非明确记录。

1.6.2.5. Kiali 支持的配置

- Kiali 控制台只支持 Google Chrome、Microsoft Edge、Mozilla Firefox 或 Apple Safari 浏览器的两个最新版本。

- 当使用 Red Hat OpenShift Service Mesh (OSSM) 部署 Kiali 时，openshift 验证策略是唯一受支持的身份验证配置。openshift 策略根据 Red Hat OpenShift Service on AWS 上的个人基于角色的访问控制(RBAC)角色来控制访问。

1.6.2.6. 分布式追踪支持的配置

- Jaeger 代理是 Jaeger 唯一支持的配置。多租户安装或 OpenShift Dedicated 不支持 Jaeger 作为 daemonset。

1.6.2.7. 支持的 WebAssembly 模块

-

3scale WebAssembly 是唯一提供 WebAssembly 模块。您可以创建自定义 WebAssembly 模块。

1.6.3. 后续步骤

- 在 Red Hat OpenShift Service on AWS 环境中安装 Red Hat OpenShift Service Mesh。

1.7. 安装 OPERATOR

要安装 Red Hat OpenShift Service Mesh，首先在 Red Hat OpenShift Service on AWS 上安装 Red Hat OpenShift Service Mesh Operator 和任何可选 Operator。然后，创建一个 ServiceMeshControlPlane 资源来部署 control plane。

注意

这一基本安装根据默认的 OpenShift 设置进行配置，不并是针对生产环境用途而设计的。 使用此默认安装验证您的安装，然后为特定环境配置服务网格。

前提条件

- 参阅准备安装 Red Hat OpenShift Service Mesh 的过程。
- 具有 cluster-admin 角色的帐户。

以下步骤演示了如何在 Red Hat OpenShift Service on AWS 上安装 Red Hat OpenShift Service Mesh 的基本实例。

重要

从 Red Hat OpenShift Service Mesh 2.5 开始，Red Hat OpenShift distributed Tracing Platform (Jaeger)和 OpenShift Elasticsearch Operator 已被弃用，并将在以后的发行版本中删除。红帽将在当前发行生命周期中为这些功能提供程序错误修正和支持，但此功能将不再获得改进，并将被删除。作为 Red Hat OpenShift distributed Tracing Platform (Jaeger)的替代选择，您可以使用 Red Hat OpenShift distributed Tracing Platform。

1.7.1. Service Mesh Operator 概述

Red Hat OpenShift Service Mesh 需要使用 Red Hat OpenShift Service Mesh Operator，它允许您连接、保护、控制并观察组成应用程序的微服务。您还可以安装其他 Operator 来增强服务网格体验。

> **警告**
>
> 不要安装 Operators 的 Community 版本。不支持社区 Operator。

需要以下 Operator：

Red Hat OpenShift Service Mesh Operator

允许您连接、保护、控制和观察组成应用的微服务。它还定义并监控管理 Service Mesh 组件的部署、更新和删除的 ServiceMeshControlPlane 资源。它基于开源 Istio 项目。

以下 Operator 是可选的：

红帽提供的 Kiali Operator

为您的服务网格提供可观察性。您可以在单个控制台中查看配置、监控流量和分析 trace。它基于开源 Kiali 项目。

Red Hat OpenShift distributed Tracing Platform

提供分布式追踪来监控复杂分布式系统中的事务并进行故障排除。它基于开源 Grafana Tempo 项目。

以下可选 Operator 已被弃用：

重要

从 Red Hat OpenShift Service Mesh 2.5 开始，Red Hat OpenShift distributed Tracing Platform (Jaeger)和 OpenShift Elasticsearch Operator 已被弃用，并将在以后的发行版本中删除。红帽将在当前发行生命周期中为这些功能提供程序错误修正和支持，但此功能将不再获得改进，并将被删除。作为 Red Hat OpenShift distributed Tracing Platform (Jaeger)的替代选择，您可以使用 Red Hat OpenShift distributed Tracing Platform。

Red Hat OpenShift distributed Tracing Platform (Jaeger)

提供分布式追踪来监控复杂分布式系统中的事务并进行故障排除。它基于开源 Jaeger 项目。

OpenShift Elasticsearch Operator

使用分布式跟踪平台(Jaeger)提供用于追踪和日志记录的数据库存储。它基于开源 Elasticsearch 项目。

1.7.2. 安装 Operator

要安装 Red Hat OpenShift Service Mesh，您必须安装 Red Hat OpenShift Service Mesh Operator。对您要安装的每个额外 Operator 重复这个过程。

其他 Operator 包括：

- 红帽提供的 Kiali Operator
- Tempo Operator

弃用的额外 Operator 包括：

重要

从 Red Hat OpenShift Service Mesh 2.5 开始，Red Hat OpenShift distributed Tracing Platform (Jaeger)和 OpenShift Elasticsearch Operator 已被弃用，并将在以后的发行版本中删除。红帽将在当前发行生命周期中为这些功能提供程序错误修正和支持，但此功能将不再获得改进，并将被删除。作为 Red Hat OpenShift distributed Tracing Platform (Jaeger)的替代选择，您可以使用 Red Hat OpenShift distributed Tracing Platform。

- Red Hat OpenShift distributed Tracing Platform (Jaeger)
- OpenShift Elasticsearch Operator

注意

如果您已经安装了 OpenShift Elasticsearch Operator 作为 OpenShift Logging 的一部分，则不需要再次安装 OpenShift Elasticsearch Operator。Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 使用已安装的 OpenShift Elasticsearch Operator 创建 Elasticsearch 实例。

流程

1. 以具有 dedicated-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS 控制台。
2. 在 Red Hat OpenShift Service on AWS web 控制台中，点 Operators → OperatorHub。
3. 在过滤器框中输入 Operator 名称，再选择 Operator 的 Red Hat 版本。不支持 Operator 的社区版本。
4. 点 Install。
5. 在每个 Operator 的 Install Operator 页面中，接受默认设置。

6. 点 Install。等待 Operator 安装，然后为要安装的下一个 Operator 重复步骤。

- Red Hat OpenShift Service Mesh Operator 在 openshift-operators 命名空间中安装，并可用于集群中的所有命名空间。

- 由红帽在 openshift-operators 命名空间中安装的 Kiali Operator，并可用于集群中的所有命名空间。

- Tempo Operator 安装在 openshift-tempo-operator 命名空间中，并可用于集群中的所有命名空间。

- Red Hat OpenShift distributed Tracing Platform (Jaeger)安装在 openshift-distributed-tracing 命名空间中，并可用于集群中的所有命名空间。

重要

从 Red Hat OpenShift Service Mesh 2.5 开始，Red Hat OpenShift distributed Tracing Platform (Jaeger)已被弃用，并将在以后的发行版本中删除。红帽将在当前发行生命周期中提供对这个功能的程序漏洞修复和支持，但这个功能将不再获得改进，并将被删除。作为 Red Hat OpenShift distributed Tracing Platform (Jaeger)的替代选择，您可以使用 Red Hat OpenShift distributed Tracing Platform。

- OpenShift Elasticsearch Operator 安装在 openshift-operators-redhat 命名空间中，并可用于集群中的所有命名空间。

重要

从 Red Hat OpenShift Service Mesh 2.5 开始，OpenShift Elasticsearch Operator 已被弃用，并将在以后的发行版本中删除。红帽将在当前发行生命周期中提供对这个功能的程序漏洞修复和支持，但这个功能将不再获得改进，并将被删除。

验证

-

安装完所有四个 Operator 后，点 Operators → Installed Operators 来验证是否安装了 Operator。

1.7.3. 后续步骤

- 在部署 Service Mesh control plane 前，Red Hat OpenShift Service Mesh Operator 不会创建 Service Mesh 自定义资源定义 (CRD)。您可以使用 ServiceMeshControlPlane 资源来安装和配置 Service Mesh 组件。如需更多信息，请参阅创建 ServiceMeshControlPlane。

1.8. 创建 SERVICEMESHCONTROLPLANE

1.8.1. 关于 ServiceMeshControlPlane

control plane 包括 Istiod、Ingress 和 Egress 网关，以及其他组件，如 Kiali 和 Jaeger。control plane 必须部署到与 Service Mesh Operator 和 data plane 应用程序和服务不同的命名空间中。您可以从 Red Hat OpenShift Service on AWS Web 控制台或使用 oc 客户端工具从命令行部署 ServiceMeshControlPlane(SMCP)的基本安装。

注意

这个基本安装会根据默认的 Red Hat OpenShift Service on AWS 设置进行配置，它不用于生产环境。使用此默认安装来验证安装，然后为您的环境配置 ServiceMeshControlPlane 设置。

如果您要在 Red Hat OpenShift Service on AWS 上部署 control plane，请参阅红帽知识库文章 OpenShift 服务网格 operator Istio 的基本内容没有因为身份验证错误而启动，它讨论了添加新项目并启动 pod。

1.8.1.1. 从 web 控制台部署 Service Mesh control plane

您可以使用 Web 控制台部署基本 ServiceMeshControlPlane。在本例中，istio-system 是 Service Mesh control plane 项目的名称。

前提条件

- 必须安装 Red Hat OpenShift Service Mesh Operator。

- 以具有 dedicated-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS 控制台。

流程

1. 以具有 cluster-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS 控制台。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 dedicated-admin 角色的帐户。
2. 创建一个名为 istio-system 的项目。
   a. 浏览至 Home → Project。
   b. 点击 Create Project。
   c. 在 Name 字段中输入 istio-system。ServiceMeshControlPlane 资源必须安装在 istio-system 项目中，并与您的微服务和 Operator 分开。
   d. 点 Create。
3. 导航到 Operators → Installed Operators。
4. 点 Red Hat OpenShift Service Mesh Operator，然后点 Istio Service Mesh Control Plane。
5. 在 Istio Service Mesh Control Plane 选项卡中，点 Create ServiceMeshControlPlane。
   a. 接受默认的 Service Mesh control plane 版本，以利用该产品的最新版本中提供的功能。control plane 的版本决定了与 Operator 版本无关的可用功能。
   b. 添加 Red Hat OpenShift Service on AWS 所需的

spec.security.identity.type.ThirdParty 字段。

c. 点 Create。

Operator 根据您的配置参数创建 pod、服务和 Service Mesh control plane 组件。您可以稍后配置 ServiceMeshControlPlane 设置。

验证

- 要验证 control plane 是否已正确安装，请点击 Istio Service Mesh Control Plane 标签页。

  a. 点新的 control plane 的名称。

  b. 点 Resources 标签页来查看由 Operator 创建并配置的 Red Hat OpenShift Service Mesh control plane 资源。

1.8.1.2. 使用 CLI 部署 Service Mesh control plane

您可以使用命令行部署基本的 ServiceMeshControlPlane。

前提条件

- 必须安装 Red Hat OpenShift Service Mesh Operator。
- 访问 OpenShift CLI（oc）。
- 以具有 dedicated-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS。

流程

1. 创建一个名为 istio-system 的项目。

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

ServiceMeshControlPlane 资源必须安装在 istio-system 项目中，并与您的微服务和 Operator 分开。

2. 使用以下示例，创建一个名为 istio-installation.yaml 的 ServiceMeshControlPlane 文件。Service Mesh control plane 的版本决定了与 Operator 版本无关的可用功能。

ServiceMeshControlPlane 资源示例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.6
  security:
    identity:
      type: ThirdParty (1)
  tracing:
    type: None
    sampling: 10000
  policy:
    type: Istiod
  addons:
    grafana:
      enabled: true
    kiali:
      enabled: true
    prometheus:
      enabled: true
  telemetry:
    type: Istiod
```

(1) 指定 Red Hat OpenShift Service on AWS 的必要设置。

3. 运行以下命令来部署 Service Mesh control plane，其中 <istio_installation.yaml> 包含到

您的文件的完整路径。

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

4. 要观察 pod 部署的进度，请运行以下命令：

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

您应该看到类似如下的输出：

```
NAME                                   READY   STATUS    RESTARTS   AGE
grafana-b4d59bd7-mrgbr                 2/2     Running   0          65m
istio-egressgateway-678dc97b4c-wrjkp   1/1     Running   0          108s
istio-ingressgateway-b45c9d54d-4qg6n   1/1     Running   0          108s
istiod-basic-55d78bbbcd-j5556          1/1     Running   0          108s
kiali-6476c7656c-x5msp                 1/1     Running   0          43m
prometheus-58954b8d6b-m5std            2/2     Running   0          66m
```

1.8.1.3. 使用 CLI 验证 SMCP 安装

您可以从命令行验证 ServiceMeshControlPlane 创建。

1. 先决条件

- 必须安装 Red Hat OpenShift Service Mesh Operator。
- 访问 OpenShift CLI（oc）。
- 以具有 dedicated-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS。

流程

1. 运行以下命令，以验证 Service Mesh control plane 安装，其中 istio-system 是安装 Service Mesh control plane 的命名空间。

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

当 STATUS 列是 ComponentsReady 时，安装成功完成。

```
NAME    READY   STATUS            PROFILES      VERSION   AGE
basic   10/10   ComponentsReady   ["default"]   2.6.7     66m
```

1.8.2. 关于 control plane 和集群范围的部署

集群范围的部署包含一个 Service Mesh Control Plane，它监控整个集群的资源。监控整个集群的资源与 control plane 在所有命名空间中使用单个查询来监控 Istio 和 Kubernetes 资源的 Istio 功能非常相似。因此，集群范围的部署会减少发送到 API 服务器的请求数。

您可以使用 Red Hat OpenShift Service on AWS web 控制台或 CLI 为集群范围的部署配置 Service Mesh Control Plane。

1.8.2.1. 使用 web 控制台为集群范围的部署配置 control plane

您可以使用 Red Hat OpenShift Service on AWS Web 控制台为集群范围的部署配置 ServiceMeshControlPlane 资源。在本例中，istio-system 是 Service Mesh control plane 项目的名称。

前提条件

- 安装了 Red Hat OpenShift Service Mesh Operator。
- 以具有 dedicated-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS。

流程

1. 创建一个名为 istio-system 的项目。
   a. 浏览至 Home → Project。

b. 点击 **Create Project**。

c. 在 **Name** 字段中输入 **istio-system**。**ServiceMeshControlPlane** 资源必须安装在独立于您的微服务和 Operator 的项目中。

这些步骤使用 **istio-system** 作为示例。只要 Service Mesh control plane 与包含服务的项目分开，就可以将 Service Mesh control plane 部署到任何项目中。

d. 点 **Create**。

2. 导航到 **Operators** → **Installed Operators**。

3. 点 Red Hat OpenShift Service Mesh Operator，然后点 **Istio Service Mesh Control Plane**。

4. 在 **Istio Service Mesh Control Plane** 选项卡中，点 **Create ServiceMeshControlPlane**。

5. 点 **YAML** 视图。Service Mesh control plane 的版本决定了与 Operator 版本无关的可用功能。

6. 修改 **spec.mode** 字段并添加 **spec.security.identity.type.ThirdParty** 字段：

ServiceMeshControlPlane 资源示例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.6
  mode: ClusterWide 1
  security:
    identity:
```

```
    type: ThirdParty 2
  tracing:
    type: Jaeger
    sampling: 10000
  policy:
    type: Istiod
  addons:
    grafana:
      enabled: true
    jaeger:
      install:
        storage:
          type: Memory
    kiali:
      enabled: true
    prometheus:
      enabled: true
  telemetry:
    type: Istiod
```

1

指定资源用于集群范围的部署。

2

指定 Red Hat OpenShift Service on AWS 的必要设置。

7. 点 Create。Operator 根据您的配置参数创建 pod、服务和 Service Mesh control plane 组件。如果 ServiceMeshMemberRoll 不存在，Operator 也会创建 ServiceMeshMemberRoll。

验证

- 验证 control plane 是否已正确安装：

  a. 点 Istio Service Mesh Control Plane 标签页。

  b. 点新 ServiceMeshControlPlane 对象的名称。

  c.

点 Resources 选项卡，查看 Operator 创建和配置的 Red Hat OpenShift Service Mesh control plane 资源。

1.8.2.2. 使用 CLI 为集群范围的部署配置 control plane

您可以使用 CLI 为集群范围的部署配置 ServiceMeshControlPlane 资源。在本例中，istio-system 是 Service Mesh control plane 命名空间的名称。

前提条件

- 安装了 Red Hat OpenShift Service Mesh Operator。
- 您可以访问 OpenShift CLI(oc)。
- 以具有 dedicated-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS。

流程

1. 创建一个名为 istio-system 的项目。

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

2. 使用以下示例，创建一个名为 istio-installation.yaml 的 ServiceMeshControlPlane 文件：

ServiceMeshControlPlane 资源示例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.6
  mode: ClusterWide (1)
```

```
security:
  identity:
    type: ThirdParty 2
```

1

指定资源用于集群范围的部署。

2

指定 Red Hat OpenShift Service on AWS 所需的设置。

3. 运行以下命令来部署 Service Mesh control plane：

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

其中：

<istio_installation.yaml>

指定文件的完整路径。

验证

1. 要监控 pod 部署的进度，请运行以下命令：

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

您应该看到类似以下示例的输出：

输出示例

```
NAME                        READY   STATUS    RESTARTS   AGE
grafana-b4d59bd7-mrgbr            2/2     Running   0          65m
```

```
istio-egressgateway-678dc97b4c-wrjkp   1/1     Running   0          108s
istio-ingressgateway-b45c9d54d-4qg6n   1/1     Running   0          108s
istiod-basic-55d78bbbcd-j5556          1/1     Running   0          108s
jaeger-67c75bd6dc-jv6k6                2/2     Running   0          65m
kiali-6476c7656c-x5msp                 1/1     Running   0          43m
prometheus-58954b8d6b-m5std            2/2     Running   0          66m
```

1.8.2.3. 为集群范围的网格自定义 member roll

在集群范围的模式中，当您创建 ServiceMeshControlPlane 资源时，也会创建 ServiceMeshMemberRoll 资源。您可以在创建 ServiceMeshMemberRoll 资源后修改它。修改资源后，Service Mesh Operator 不再更改它。如果您使用 Red Hat OpenShift Service on AWS Web 控制台修改 ServiceMeshMemberRoll 资源，请接受提示来覆盖修改。

另外，您可以在部署 ServiceMeshControlPlane 资源前创建一个 ServiceMeshMemberRoll 资源。在创建 ServiceMeshControlPlane 资源时，Service Mesh Operator 不会修改 ServiceMeshMemberRoll。

注意

ServiceMeshMemberRoll 资源名称必须命名为 default，且必须与 ServiceMeshControlPlane 资源在同一项目命名空间中创建。

将命名空间添加到网格的方法有两种。您可以通过在 spec.members 列表中指定名称来添加命名空间，或者将一组命名空间标签选择器配置为根据其标签包含或排除命名空间。

注意

无论您在 ServiceMeshMemberRoll 资源中如何指定成员，您也可以通过在每个命名空间中创建 ServiceMeshMember 资源，将成员添加到网格中。

1.8.3. 使用 Kiali 验证 SMCP 安装

您可以使用 Kiali 控制台验证 Service Mesh 安装。Kiali 控制台提供了一种方式来验证您的 Service Mesh 组件是否已正确部署和配置。

1. 前提条件

   - 必须安装 Red Hat OpenShift Service Mesh Operator。
   - 访问 OpenShift CLI（oc）。
   - 以具有 dedicated-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS。

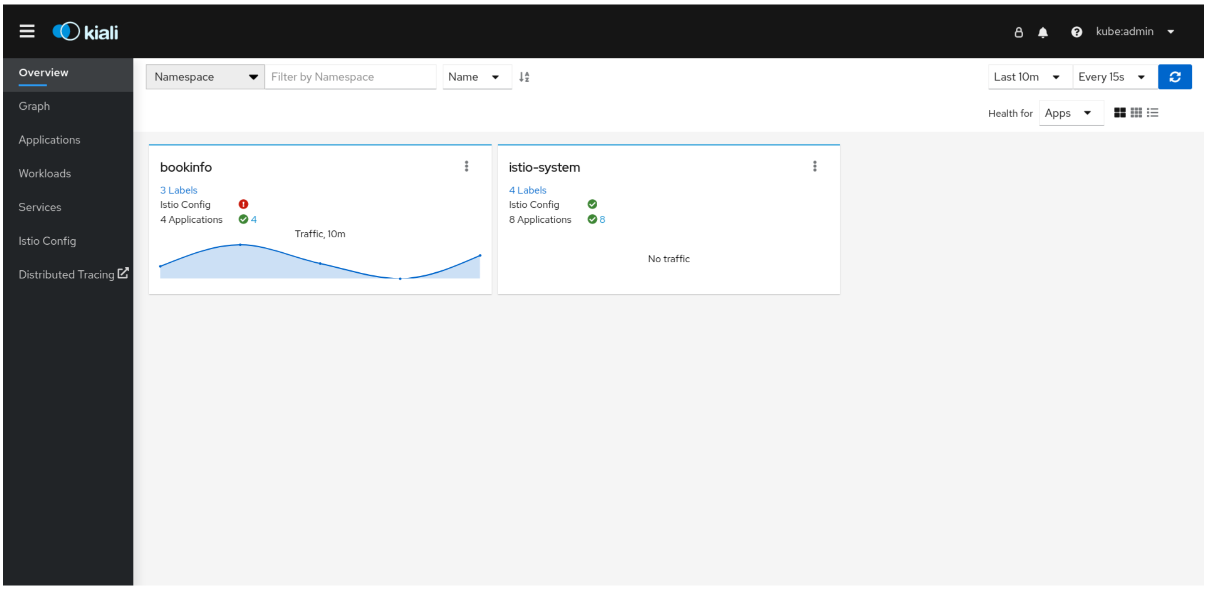
流程

1. 在 Red Hat OpenShift Service on AWS web 控制台中进入 Networking → Routes。
2. 在 Routes 页面中，从 Namespace 菜单中选择 Service Mesh control plane 项目，如 istio-system。

   Location 列显示每个路由的链接地址。
3. 如果需要，使用过滤器查找 Kiali 控制台的路由。单击路由 位置 以启动控制台。
4. 单击 Log In With OpenShift。

   第一次登录到 Kiali 控制台时，您会看到 Overview 页面，它会显示服务网格中您有权查看的所有命名空间。当 Overview 页中显示多个命名空间，Kiali 会首先显示具有健康或验证问题的命名空间。

图 1.1. Kiali Overview 页



每个命名空间的 tile 会显示标签数量、Istio 配置健康、和 应用程序 健康状态的数量，以及命名空间的流量。如果您验证了控制台安装，且命名空间还没有添加到网格中，则可能无法显示 istio-system 以外的任何数据。

5. Kiali 有四个仪表板，专门用于安装了 Service Mesh control plane 的命名空间。要查看这些仪表板，请点击 control plane 命名空间的标题 中的 Options 菜单，如 istio-system，然后选择以下选项之一：

- Istio Mesh Dashboard
- Istio Control Plane Dashboard
- Istio Performance Dashboard
- Istio Wasm Exetension Dashboard

图 1.2. Grafana Istio Control Plane Dashboard

Kiali 还会安装两个额外的 Grafana 仪表板，它们可从 Grafana Home 页面获得：

- Istio Workload Dashboard
- Istio Service Dashboard

6. 要查看 Service Mesh control plane 节点，点 Graph 页面，从菜单中选择安装 ServiceMeshControlPlane 的命名空间，如 istio-system。

   a. 如有必要，请单击 Display idle nodes。

   b. 要了解更多有关 Graph 页面的信息，请点击 Graph tour 链接。

   c. 要查看网格拓扑，请从 Namespace 菜单中从 Service Mesh Member Roll 中选择一个或多个附加命名空间。

7. 要查看 istio-system 命名空间中的应用程序列表，请点击 Applications 页面。Kiali 显示应用程序的健康状况。

a. 将鼠标指针悬停在信息图标上，以查看 Details 列中记下的任何其他信息。

8. 要在 istio-system 命名空间中查看工作负载列表，请点击 Workloads 页面。Kiali 显示工作负载的运行状况。

 a. 将鼠标指针悬停在信息图标上，以查看 Details 列中记下的任何其他信息。

9. 要查看 istio-system 命名空间中的服务列表，点 Services 页面。Kiali 显示服务和配置的健康状态。

 a. 将鼠标指针悬停在信息图标上，以查看 Details 列中记下的任何其他信息。

10. 要查看 istio-system 命名空间中的 Istio Configuration 对象列表，点 Istio Config 页面。Kiali 显示配置的健康状况。

 a. 如果出现配置错误，点行，Kiali 会打开配置文件并突出显示错误。

1.8.4. 其他资源

Red Hat OpenShift Service Mesh 支持集群中的多个独立 control plane。您可以使用 ServiceMeshControlPlane 配置集创建可重复使用的配置。如需更多信息，请参阅创建 control plane 配置集。

1.8.5. 后续步骤

- 在 Service Mesh 中添加项目，以便应用程序可用。如需更多信息，请参阅在服务网格中添加服务。

1.9. 在服务网格中添加服务

项目包含服务；但是，只有在将项目添加到服务网格时，服务才可用。

1.9.1. 关于将项目添加到服务网格中

安装 Operator 并创建 ServiceMeshControlPlane 资源后，将一个或多个项目添加到服务网格中。

注意

在 Red Hat OpenShift Service on AWS 中，项目基本上是一个带有额外注解的 Kubernetes 命名空间，如项目中可以使用的用户 ID 范围。通常，Red Hat OpenShift Service on AWS Web 控制台使用术语项目，CLI 使用术语命名空间，但术语基本上是合成性。

您可以使用 Red Hat OpenShift Service on AWS Web 控制台或 CLI 将项目添加到现有服务网格中。将项目添加到服务网格中有三种方法：

- 在 ServiceMeshMemberRoll 资源中指定项目名称。
- 在 ServiceMeshMemberRoll 资源的 spec.memberSelectors 字段中配置标签选择器。
- 在项目中创建 ServiceMeshMember 资源。

如果使用第一个方法，您必须创建 ServiceMeshMemberRoll 资源。

1.9.2. 创建 Red Hat OpenShift Service Mesh member roll

ServiceMeshMemberRoll 列出属于 Service Mesh control plane 的项目。只有 ServiceMeshMemberRoll 中列出的项目会受到 control plane 的影响。在将项目添加到特定 control plane 部署的 member roll 之前，项目不属于服务网格。

您必须在 ServiceMeshControlPlane 所在的同一个项目中创建一个名为 default 的 ServiceMeshMemberRoll 资源，如 istio-system。

1.9.2.1. 从 Web 控制台创建 member roll

您可从 web 控制台在 Service Mesh member roll 中添加一个或多个项目。在本例中，istio-system 是 Service Mesh control plane 项目的名称。

前提条件

- 已安装并验证的 Red Hat OpenShift Service Mesh Operator。
- 要添加到服务网格的现存项目列表。

流程

1. 登陆到 Red Hat OpenShift Service on AWS web 控制台。
2. 如果您还没有网格服务，或者您从头开始，请为您的应用程序创建一个项目。它必须与安装 Service Mesh control plane 的项目不同。
   a. 浏览至 Home → Project。
   b. 在 Name 字段中输入一个名称。
   c. 点 Create。
3. 导航到 Operators → Installed Operators。
4. 点 Project 菜单，从列表中选择部署 ServiceMeshControlPlane 资源的项目，如 istio-system。
5. 点 Red Hat OpenShift Service Mesh Operator。
6. 点 Istio Service Mesh Member Roll 选项卡。

7. 点 Create ServiceMeshMemberRoll

8. 单击 Members，然后在 Value 字段中输入项目名称。您可以添加任意数量的项目，但一个项目只能属于一个 ServiceMeshMemberRoll 资源。

9. 点 Create。

1.9.2.2. 通过 CLI 创建 member roll

您可以使用命令行将项目添加到 ServiceMeshMemberRoll 中。

前提条件

- 已安装并验证的 Red Hat OpenShift Service Mesh Operator。

- 要添加到服务网格的项目列表。

- 访问 OpenShift CLI（oc）。

流程

1. 登陆到 Red Hat OpenShift Service on AWS CLI。

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. 如果您还没有网格服务，或者您从头开始，请为您的应用程序创建一个项目。它必须与安装 Service Mesh control plane 的项目不同。

```
$ oc new-project <your-project>
```

3. 要添加项目作为成员，请修改以下示例 YAML:您可以添加任意数量的项目，但一个项目只能属于一个 ServiceMeshMemberRoll 资源。在本例中，istio-system 是 Service Mesh control

plane 项目的名称。

servicemeshmemberroll-default.yaml 示例

```
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. 运行以下命令，在 istio-system 命名空间中上传并创建 ServiceMeshMemberRoll 资源。

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

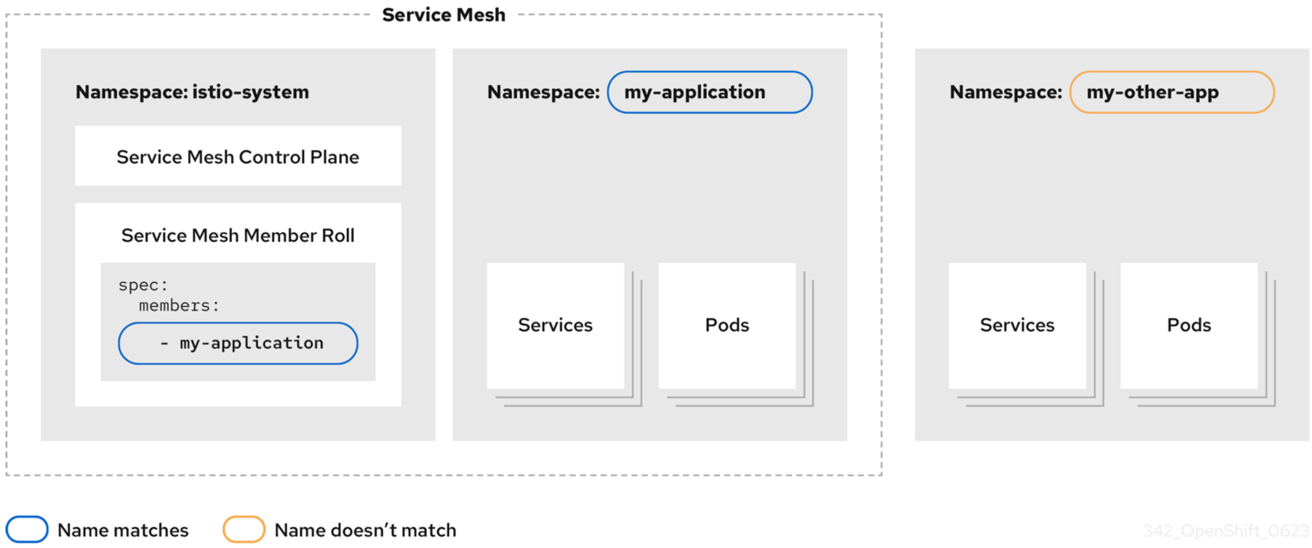
5. 运行以下命令，以验证 ServiceMeshMemberRoll 是否已成功创建。

```
$ oc get smmr -n istio-system default
```

当 STATUS 列为 Configured 时，安装成功完成。

1.9.3. 关于使用 ServiceMeshMemberRoll 资源添加项目

使用 ServiceMeshMemberRoll 资源是将项目添加到服务网格的最简单方法。要添加项目，请在 ServiceMeshMemberRoll 资源的 spec.members 字段中指定项目名称。ServiceMeshMemberRoll 资源指定哪个项目由 ServiceMeshControlPlane 资源控制。



注意

使用此方法添加项目需要用户具有相关项目中的 update servicemeshmemberrolls 和 update pod 特权。

- 如果您已有要添加到服务网格中的应用程序、工作负载或服务，请查看以下操作：
  - 在 web 控制台中使用 ServiceMeshMemberRoll 资源从网格中添加或删除项目
  - 通过 CLI 使用 ServiceMeshMemberRoll 资源从网格中添加或删除项目
- 另外，要安装一个名为 Bookinfo 的示例应用程序并将其添加到 ServiceMeshMemberRoll 资源中，请参阅 Bookinfo 示例应用程序教程。

1.9.3.1. 在 web 控制台中使用 ServiceMeshMemberRoll 资源从网格中添加或删除项目

您可以使用 Red Hat OpenShift Service on AWS Web 控制台的 ServiceMeshMemberRoll 资源从网格中添加或删除项目。您可以添加任意数量的项目，但项目只能属于一个网格。

当它对应的 ServiceMeshControlPlane 资源被删除后，ServiceMeshMemberRoll 资源也会被删除。

前提条件

- 已安装并验证的 Red Hat OpenShift Service Mesh Operator。
- 现有 ServiceMeshMemberRoll 资源
- 带有 ServiceMeshMemberRoll 资源的项目名称。
- 要从网格中添加或删除的项目的名称。

流程

1. 登陆到 Red Hat OpenShift Service on AWS web 控制台。
2. 导航到 Operators → Installed Operators。
3. 点 Project 菜单，从列表中选择部署了 ServiceMeshControlPlane 资源的项目。例如 istio-system。
4. 点 Red Hat OpenShift Service Mesh Operator。
5. 点 Istio Service Mesh Member Roll 选项卡。
6. 点 default 链接。
7. 点 YAML 标签。
8. 修改 YAML 以添加项目作为成员（或删除项目以删除现有成员）。您可以添加任意数量的项目，但一个项目只能属于一个 ServiceMeshMemberRoll 资源。

servicemeshmemberroll-default.yaml 示例

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

9. 点 **Save**。

10. 点 **Reload**。

1.9.3.2. 使用 CLI 的 ServiceMeshMemberRoll 资源从网格中添加或删除项目

您可以使用 CLI 的 ServiceMeshMemberRoll 资源将一个或多个项目添加到网格中。您可以添加任意数量的项目，但项目只能属于一个网格。

当它对应的 ServiceMeshControlPlane 资源被删除后，ServiceMeshMemberRoll 资源也会被删除。

前提条件

- 已安装并验证的 Red Hat OpenShift Service Mesh Operator。
- 现有 ServiceMeshMemberRoll 资源
- 带有 ServiceMeshMemberRoll 资源的项目名称。

- 要从网格中添加或删除的项目的名称。
- 访问 OpenShift CLI（oc）。

流程

1. 登陆到 Red Hat OpenShift Service on AWS CLI。
2. 编辑 ServiceMeshMemberRoll 资源。

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

3. 修改 YAML 以添加或删除作为成员的项目。您可以添加任意数量的项目，但一个项目只能属于一个 ServiceMeshMemberRoll 资源。

servicemeshmemberroll-default.yaml 示例

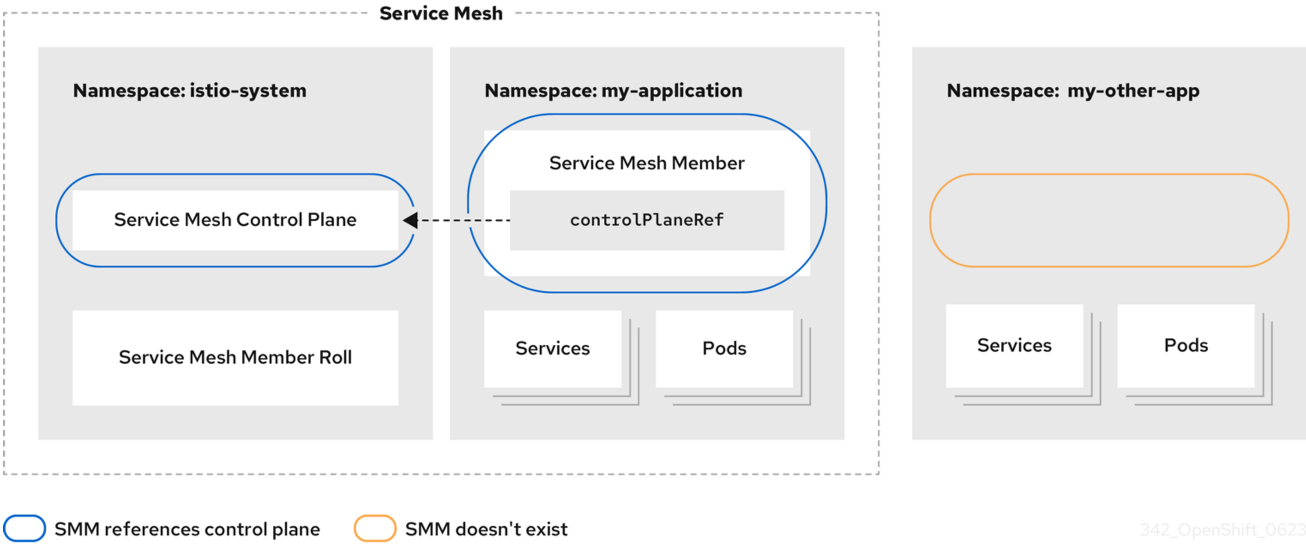
```yaml
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
  namespace: istio-system #control plane project
spec:
  members:
    # a list of projects joined into the service mesh
    - your-project-name
    - another-project-name
```

4. 保存文件并退出编辑器。

1.9.4. 关于使用 ServiceMeshMember 资源添加项目

ServiceMeshMember 资源提供了一种在不修改 ServiceMeshMemberRoll 资源的情况下将项目添加

到服务网格的方法。要添加项目，请在您要添加到服务网格的项目中创建 ServiceMeshMember 资源。当 Service Mesh Operator 处理 ServiceMeshMember 对象时，项目会出现在 ServiceMeshMemberRoll 资源的 status.members 列表中。然后，属于项目中的服务对网格提供。



网格管理员需要为每个网关用户授予引用 ServiceMeshMember 资源中的 ServiceMeshControlPlane 资源的权限。使用这个权限，网格用户可以将项目添加到网格中，即使该用户没有服务网格项目或 ServiceMeshMemberRoll 资源的直接访问权限。如需更多信息，请参阅创建 Red Hat OpenShift Service Mesh 成员。

1.9.4.1. 在 web 控制台中使用 ServiceMeshMember 资源将项目添加到网格中

您可以使用 Red Hat OpenShift Service on AWS Web 控制台的 ServiceMeshMember 资源将一个或多个项目添加到网格中。

前提条件

- 已安装 Red Hat OpenShift Service Mesh Operator。
- 您知道 ServiceMeshControlPlane 资源的名称以及资源所属的项目的名称。
- 您知道您要添加到网格中的项目名称。
- 服务网格管理员必须明确授予服务网格的访问权限。管理员可以使用 RoleBinding 或 ClusterRoleBinding 为用户授予 mesh-user 角色 的访问权限。如需更多信息，请参阅*创建 Red*

Hat OpenShift Service Mesh 成员。

流程

1. 登陆到 Red Hat OpenShift Service on AWS web 控制台。

2. 导航到 Operators → Installed Operators。

3. 点 Project 菜单，然后从下拉列表中选择您要添加到网格的项目。例如： istio-system。

4. 点 Red Hat OpenShift Service Mesh Operator。

5. 点 Istio Service Mesh Member 选项卡。

6. 点 Create ServiceMeshMember

7. 接受 ServiceMeshMember 的默认名称。

8. 点击以展开 ControlPlaneRef。

9. 在 Namespace 字段中，选择 ServiceMeshControlPlane 资源所属的项目。例如： istio-system。

10. 在 Name 字段中输入此命名空间所属的 ServiceMeshControlPlane 资源的名称。例如，基本。

11. 点 Create。

验证

1. 通过以下步骤确认 ServiceMeshMember 资源已创建，并且项目已添加到网格中：

   a. 单击资源名称，如 default。

   b. 查看屏幕末尾显示的 Conditions 部分。

   c. 确认 Reconciled 和 Ready 条件的 Status 为 True。

      如果 Status 为 False，请参阅 Reason 和 Message 列以了解更多信息。

1.9.4.2. 通过 CLI 使用 ServiceMeshMember 资源将项目添加到网格

您可以通过 CLI 使用 ServiceMeshMember 资源将一个或多个项目添加到网格中。

前提条件

- 已安装 Red Hat OpenShift Service Mesh Operator。

- 您知道 ServiceMeshControlPlane 资源的名称及其所属项目的名称。

- 您知道您要添加到网格中的项目名称。

- 服务网格管理员必须明确授予服务网格的访问权限。管理员可以使用 RoleBinding 或 ClusterRoleBinding 为用户授予 mesh-user 角色 的访问权限。如需更多信息，请参阅*创建 Red Hat OpenShift Service Mesh 成员*。

流程

1. 登陆到 Red Hat OpenShift Service on AWS CLI。

2. 为 ServiceMeshMember 清单创建 YAML 文件。清单将 my-application 项目添加到由

istio-system 命名空间中部署的 ServiceMeshControlPlane 资源创建的服务网格中：

```
apiVersion: maistra.io/v1
kind: ServiceMeshMember
metadata:
  name: default
  namespace: my-application
spec:
  controlPlaneRef:
    namespace: istio-system
    name: basic
```

3. 应用 YAML 文件以创建 ServiceMeshMember 资源：

```
$ oc apply -f <file-name>
```

验证

- 运行以下命令，验证命名空间是否为网格的一部分。确认值 True 出现在 READY 列中。

```
$ oc get smm default -n my-application
```

输出示例

```
NAME     CONTROL PLANE        READY   AGE
default  istio-system/basic   True    2m11s
```

- 另外，查看 ServiceMeshMemberRoll 资源，以确认 my-application 命名空间显示在 ServiceMeshMemberRoll 资源的 status.members 和 status.configuredMembers 字段中。

```
$ oc describe smmr default -n istio-system
```

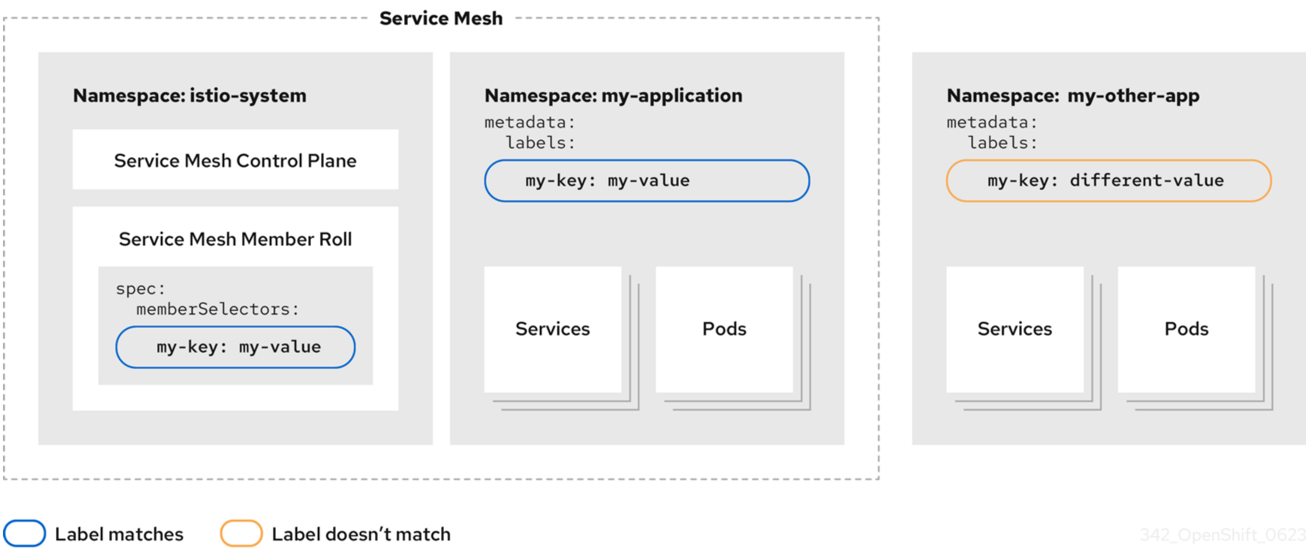
输出示例

```
Name:        default
Namespace:   istio-system
```

```
Labels:       <none>
# ...
Status:
# ...
  Configured Members:
    default
    my-application
# ...
  Members:
    default
    my-application
```

1.9.5. 关于使用标签选择器添加项目

对于集群范围的部署，您可以使用标签选择器将项目添加到网格中。ServiceMeshMemberRoll 资源中指定的标签选择器可让 Service Mesh Operator 根据命名空间标签向网格添加或删除命名空间。与可以用来指定单个标签选择器的 AWS 资源的其他标准 Red Hat OpenShift Service 不同，您可以使用 ServiceMeshMemberRoll 资源指定多个标签选择器。



如果命名空间标签与 ServiceMeshMemberRoll 资源中指定的任何选择器匹配，则命名空间会包含在网格中。

注意

在 Red Hat OpenShift Service on AWS 中，项目基本上是一个带有额外注解的 Kubernetes 命名空间，如项目中可以使用的用户 ID 范围。通常，Red Hat OpenShift Service on AWS Web 控制台使用术语 *项目*，CLI 使用术语 *命名空间*，但术语本质上是相同的。

1.9.5.1. 使用带有 Web 控制台的标签选择器将项目添加到网格中

您可以使用标签选择器使用 Red Hat OpenShift Service on AWS Web 控制台将项目添加到 Service Mesh。

前提条件

- 已安装 Red Hat OpenShift Service Mesh Operator。
- 部署有一个现有的 ServiceMeshMemberRoll 资源。
- 以具有 dedicated-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS web 控制台。

流程

1. 导航到 Operators → Installed Operators。
2. 点 Project 菜单，从下拉菜单中选择部署 ServiceMeshMemberRoll 资源的项目。例如：istio-system。
3. 点 Red Hat OpenShift Service Mesh Operator。
4. 点 Istio Service Mesh Member Roll 选项卡。
5. 单击 Create ServiceMeshMember Roll。

6. 接受 ServiceMeshMemberRoll 的默认名称。

7. 在 Labels 字段中输入键值对，以定义标识服务网格中要包含哪些命名空间的标签。如果项目命名空间具有选择器指定的标签，则项目命名空间会包含在服务网格中。您不需要同时包含这两个标签。

   例如，输入 mykey=myvalue 包括具有此标签的所有命名空间作为网格的一部分。当选择器标识匹配项时，项目命名空间将添加到服务网格中。

   输入 myotherkey=myothervalue 包括具有此标签的所有命名空间作为网格的一部分。当选择器标识匹配项时，项目命名空间将添加到服务网格中。

8. 点 Create。

1.9.5.2. 使用带有 CLI 的标签选择器将项目添加到网格中

您可以使用标签选择器通过 CLI 将项目添加到 Service Mesh 中。

前提条件

- 已安装 Red Hat OpenShift Service Mesh Operator。
- 部署有一个现有的 ServiceMeshMemberRoll 资源。
- 以具有 dedicated-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS。

流程

1. 登陆到 Red Hat OpenShift Service on AWS CLI。
2. 编辑 ServiceMeshMemberRoll 资源。

```
$ oc edit smmr default -n istio-system
```

您可以将 Service Mesh control plane 部署到任何与包含服务的项目分开的项目。

3. 修改 YAML 文件，以在 ServiceMeshMemberRoll 资源的 spec.memberSelectors 字段中包含命名空间标签选择器。

注意

您还可以使用选择器中的 matchExpressions 字段，而不使用 matchLabels 字段。

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
  namespace: istio-system
spec:
  memberSelectors: 1
  - matchLabels: 2
      mykey: myvalue 3
  - matchLabels: 4
      myotherkey: myothervalue 5
```

1 包含用于标识服务网格中包含的项目命名空间的标签选择器。如果项目命名空间具有选择器指定的标签，则项目命名空间会包含在服务网格中。项目命名空间不需要同时包含这两个标签。

2 3 使用 mykey=myvalue 标签指定所有命名空间。当选择器标识匹配项时，项目命名空间将添加到服务网格中。

4 5 使用 myotherkey=myothervalue 标签指定所有命名空间。当选择器标识匹配项时，项目命名空间将添加到服务网格中。

1.9.6. Bookinfo 示例应用程序

您可以使用 Bookinfo 示例应用程序来测试 Red Hat OpenShift Service on AWS 中的 Red Hat OpenShift Service Mesh 2.6.7 安装。

Bookinfo 应用程序显示一本书的信息，类似于在线书店的单一目录条目。应用会显示一个页面，其中描述了图书详细信息（ISBN、页数和其他信息）以及图书的评论。

Bookinfo 应用程序由这些微服务组成：

- productpage 微服务调用 details 和 reviews 微服务来产生页面信息。
- details 微服务包括了书的信息。
- review 微服务包括了书的评论。它同时还会调用 ratings 微服务。
- ratings微服务包括了带有对本书的评论信息的评分信息。

reviews 微服务有三个版本：

- 版本 v1 不调用 ratings 服务。
- 版本 v2 调用 ratings 服务，并以一到五个黑色星来代表对本书的评分。
- 版本 v3 调用 ratings 服务，并以一到五个红色星来代表对本书的评分。

1.9.6.1. 安装 Bookinfo 应用程序

本教程介绍了如何创建项目、将 Bookinfo 应用程序部署到该项目并在 Service Mesh 中查看正在运行的应用程序来创建示例应用程序。

前提条件

- 安装了 Red Hat OpenShift Service on AWS 4.1 或更高版本。
- 安装了 Red Hat OpenShift Service Mesh 2.6.7。
- 访问 OpenShift CLI（oc）。
- 以具有 dedicated-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS。

> **注意**
>
> 本节中的命令假设 Service Mesh control plane 项目为 istio-system。如果在另一个命名空间中安装了 control plane，在运行前编辑每个命令。

流程

1. 点 Home → Projects。
2. 点击 Create Project。
3. 在 Project Name 中输入 info，输入 Display Name 及 Description，然后点 Create。
   - 或者，也可以通过 CLI 运行这个命令来创建 info 项目。

     ```
     $ oc new-project info
     ```
4. 点 Operators → Installed Operators。
5. 点 Project 菜单，使用 Service Mesh control plane 命名空间。在这个示例中，使用 istio-system。

6. 点 Red Hat OpenShift Service Mesh Operator。

7. 点 Istio Service Mesh Member Roll 选项卡。

   a. 如果您已经创建了 Istio Service Mesh Member Roll，请名称，然后点击 YAML 标签来打开 YAML 编辑器。

   b. 如果您还没有创建 ServiceMeshMemberRoll，点 Create ServiceMeshMemberRoll。

8. 单击 Members，然后在 Value 字段中输入项目名称。

9. 点 Create 保存更新的 Service Mesh Member Roll。

   a. 或者，将以下示例保存到 YAML 文件中。

Bookinfo ServiceMeshMemberRoll 示例 servicemeshmemberroll-default.yaml

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

   b. 运行以下命令上传该文件，并在 istio-system 命名空间中创建 ServiceMeshMemberRoll 资源。在本例中，istio-system 是 Service Mesh control plane 项目的名称。

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

10. 运行以下命令，以验证 ServiceMeshMemberRoll 是否已成功创建。

```
$ oc get smmr -n istio-system -o wide
```

当 STATUS 列为 Configured 时，安装成功完成。

```
NAME     READY   STATUS       AGE   MEMBERS
default  1/1     Configured   70s   ["info"]
```

11. 在 CLI 中，通过应用 bookinfo.yaml 文件在 `*info*`项目中部署 Bookinfo：

```
$ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.6/samples/bookinfo/platform/kube/bookinfo.yaml
```

您应该看到类似如下的输出：

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

12. 通过应用 info-gateway.yaml 文件创建入站网关 ：

```
$ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.6/samples/bookinfo/networking/bookinfo-gateway.yaml
```

您应该看到类似如下的输出：

```
gateway.networking.istio.io/info-gateway created
virtualservice.networking.istio.io/info created
```

13. 设置 GATEWAY_URL 参数的值：

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

1.9.6.2. 添加默认目的地规则

在使用 Bookinfo 应用程序前，您必须首先添加默认目的地规则。根据您是否启用了 mutual TLS 验证，预先配置两个 YAML 文件。

流程

1. 要添加目的地规则，请运行以下命令之一：

 - 如果没有启用 mutual TLS：

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

 - 如果启用了 nutual TLS：

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

 您应该看到类似如下的输出：

```
destinationrule.networking.istio.io/productpage created
destinationrule.networking.istio.io/reviews created
destinationrule.networking.istio.io/ratings created
destinationrule.networking.istio.io/details created
```

1.9.6.3. 验证 Bookinfo 安装

要确认示例 Bookinfo 应用程序已被成功部署，请执行以下步骤。

前提条件

- 安装了 Red Hat OpenShift Service Mesh。
- 完成安装 Bookinfo 示例应用程序的步骤。
- 以具有 dedicated-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS。

通过 CLI 的步骤

1. 验证所有 pod 是否都与此命令就绪：

```
$ oc get pods -n info
```

   所有容器集的状态都应为 Running。您应该看到类似如下的输出：

```
NAME                              READY   STATUS    RESTARTS   AGE
details-v1-55b869668-jh7hb        2/2     Running   0          12m
productpage-v1-6fc77ff794-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-v3-6dfd49b55b-vcwpf       2/2     Running   0          12m
```

2. 运行以下命令来检索产品页面的 URL:

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

3. 在网页浏览器中复制并粘贴输出以验证是否已部署了 Bookinfo 产品页面。

来自 Kiali web 控制台的步骤

1. 获取 Kiali web 控制台的地址。

   a. 登陆到 Red Hat OpenShift Service on AWS web 控制台。

b. 进入 **Networking** → **Routes**。

c. 在 **Routes** 页面中，从 **Namespace** 菜单中选择 Service Mesh control plane 项目，如 **istio-system**。

Location 列显示每个路由的链接地址。

d. 点 Kiali 的 **Location** 列中的链接。

e. 单击 **Log In With OpenShift**。Kiali **Overview** 屏幕显示每个项目命名空间的标题。

2. 在 Kiali 中，点 **Graph**。

3. 从 **Namespace** 列表中选择 info，从 **Graph Type** 列表中选择 **App graph**。

4. 从 **Display** 菜单中选择 **Display idle nodes**。

这将显示定义的节点，但尚未收到或发送请求。它可以确认应用已正确定义，但未报告任何请求流量。

- 使用 **Duration** 菜单增加时间段，以帮助确保捕获旧的流量。

- 使用 **Refresh Rate** 菜单刷新流量频率或更小，或者根本不刷新流量。

5. 点 Services、Workloads 或 Istio Config 查看 info 组件的列表视图，并确认它们健康。

1.9.6.4. 删除 Bookinfo 应用程序

按照以下步骤删除 Bookinfo 应用程序。

前提条件

- 安装了 Red Hat OpenShift Service on AWS 4.1 或更高版本。
- 安装了 Red Hat OpenShift Service Mesh 2.6.7。
- 访问 OpenShift CLI（oc）。

1.9.6.4.1. 删除 Bookinfo 项目

流程

1. 登陆到 Red Hat OpenShift Service on AWS web 控制台。
2. 点 Home → Projects。
3. 点 info 菜单

   ，然后点 Delete Project。
4. 在确认对话框中键入 info，然后点 Delete。

   - 或者，您可以使用 CLI 运行这个命令来创建 info 项目。

```
$ oc delete project info
```

1.9.6.4.2. 从 Service Mesh member roll 中删除 Bookinfo 项目

流程

1. 登陆到 Red Hat OpenShift Service on AWS web 控制台。

2. 点 **Operators** → **Installed Operators**。

3. 点 **Project** 菜单，从列表中选 **istio-system**。

4. 为 Red Hat OpenShift Service Mesh Operator 在 **Provided APIS** 下点 **Istio Service Mesh Member Roll** 链接。

5. 点 **ServiceMeshMemberRoll** 菜单
并选择 **Edit Service Mesh Member Roll**。

6. 编辑默认的 Service Mesh Member Roll YAML 并从 **members** 列表中删除 **info**。

   - 或者，您可以使用 CLI 运行这个命令从 **ServiceMeshMemberRoll** 中删除 **info** 项目。在本例中，**istio-system** 是 Service Mesh control plane 项目的名称。

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

7. 点 **Save** 更新 Service Mesh Member Roll。

1.9.7. 后续步骤

- 要继续安装过程，您必须启用 sidecar 注入。

1.10. 启用 SIDECAR 注入

将包含您的服务的命名空间添加到网格后，下一步是在应用程序的 Deployment 资源中启用自动 sidecar 注入功能。您必须为每个部署启用自动 sidecar 注入。

如果您已安装 Bookinfo 示例应用程序，则会部署应用程序，并作为安装过程的一部分注入 sidecar。如果您使用自己的项目和服务，请在 Red Hat OpenShift Service on AWS 上部署应用程序。

注意

通过初始容器启动的流量（在 pod 中应用程序容器之前运行的专用容器）默认无法在服务网格外传输。任何需要在网格外建立网络流量连接的操作初始容器执行都会失败。

有关将初始容器连接到服务的更多信息，请参阅 注入 Service Mesh sidecar 的 pod 中的 CrashLoopBackOff 中的红帽知识库解决方案 initContainer

1.10.1. 先决条件

- 部署到网格中的服务，如 Bookinfo 示例应用程序。
- 部署资源文件。

1.10.2. 启用自动 sidecar 注入

在部署应用程序时，您必须通过将 deployment 对象中的 spec.template.metadata.labels 中的标签 sidecar.istio.io/inject 配置为 true 来选择注入。选择确保 sidecar 注入不会影响 Red Hat OpenShift Service on AWS 的其他功能，如 Red Hat OpenShift Service on AWS 生态系统中的多个框架使用的 builder pod。

前提条件

- 识别作为服务网格一部分的命名空间，以及需要自动 sidecar 注入的部署。

流程

1. 要查找部署，请使用 oc get 命令。

```
$ oc get deployment -n <namespace>
```

例如，若要查看 info 命名空间中 'ratings-v1' 微服务的 Deployment YAML 文件，请使用以下命令以 YAML 格式查看资源：

```
oc get deployment -n info ratings-v1 -o yaml
```

2. 在编辑器中打开应用程序的 Deployment YAML 文件。

3. 将 spec.template.metadata.annotations.sidecar.istio/inject 添加到 Deployment YAML 中，并将 sidecar.istio.io/inject 设置为 true，如下例所示。

info deployment-ratings-v1.yaml 中的代码片段示例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ratings-v1
  namespace: info
  labels:
    app: ratings
    version: v1
spec:
  template:
    metadata:
      labels:
        sidecar.istio.io/inject: 'true'
```

注意

在启用自动 sidecar 注入时使用 annotations 参数已被弃用，应使用 labels 参数替代。

4. 保存 Deployment YAML 文件。

5. 将文件添加回包含应用程序的项目。

```
$ oc apply -n <namespace> -f deployment.yaml
```

在本例中，info 是包含 ratings-v1 应用程序和 deployment-ratings-v1.yaml 的项目的名称，这是您编辑的文件。

```
$ oc apply -n info -f deployment-ratings-v1.yaml
```

6. 若要验证资源上传成功，请运行以下命令：

```
$ oc get deployment -n <namespace> <deploymentName> -o yaml
```

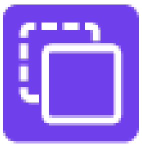
例如，

```
$ oc get deployment -n info ratings-v1 -o yaml
```

1.10.3. 验证 sidecar 注入

Kiali 控制台提供了多种方式来验证应用程序、服务和工作负载是否有 sidecar 代理。

图 1.3. 缺少 sidecar badge



Graph 页面显示一个节点徽标，它显示了以下图形中的 Missing Sidecar ：

- 应用程序图
- 版本的应用程序图
- 工作负载图

图 1.4. 缺少 sidecar 图标

Applications 页面在没有 sidecar 的命名空间中任何应用程序的 Details 列中显示一个 Missing Sidecar 图标。

Workloads 页面中为没有 sidecar 的任何应用程序的 Details 列中显示一个 Missing Sidecar 图标。

Services 页面在没有 sidecar 的命名空间中任何应用程序的 Details 列中显示一个 Missing Sidecar 图标。当服务有多个版本时，您可以使用 Service Details 页面查看 Missing Sidecar 图标。

Workload Details 页面有一个特殊的统一 Logs 选项卡，可让您查看和关联应用程序和代理日志。您可以将 Envoy 日志视为验证应用程序工作负载的 sidecar 注入的另一种方式是查看。

Workload Details 页面还具有作为 Envoy 代理或被注入 Envoy 代理的任何工作负载的 Envoy 标签页。此选项卡显示内置的 Envoy 仪表板，其中包含 Clusters、Listeners、Routes、Bootstrap、Config 和 Metrics 的子选项卡。

有关启用 Envoy 访问日志的详情，请参考故障排除部分。

有关查看 Envoy 日志的详情，请参考 Kiali 控制台中的日志

1.10.4. 通过注解设置代理环境变量

Envoy sidecar 代理的配置由 ServiceMeshControlPlane 管理。

您可以通过在 injection-template.yaml 文件中的部署中添加 pod 注解来为应用程序设置 sidecar 代理的环境变量。环境变量注入 sidecar。

injection-template.yaml 示例

```
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\" }"
```

> **警告**
>
> 在创建自己的自定义资源时，您绝不应包含 maistra.io/ 标签和注解。这些标签和注解表示资源由 Operator 生成和管理。如果您在创建自己的资源时从 Operator 生成的资源复制内容，请不要包含以 maistra.io/ 开头的标签或注解。在下一个协调过程中，Operator 将覆盖或删除这些标签或注解的资源。

1.10.5. 更新 sidecar 代理

要更新 sidecar 代理的配置，应用程序管理员必须重启应用程序 pod。

如果您的部署使用了自动 sidecar 注入功能，则可以通过添加或修改注解来更新部署中的 pod 模板。运行以下命令来重新部署 pod：

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

如果您的部署没有使用自动 sidecar 注入功能，则必须通过修改部署或 pod 中指定的 sidecar 容器镜

像来手动更新 sidecar，然后重启 pod。

1.10.6. 后续步骤

为您的环境配置 Red Hat OpenShift Service Mesh 功能。

- 安全性
- 流量管理
- 指标、日志和追踪

1.11. 升级 SERVICE MESH

要访问 Red Hat OpenShift Service Mesh 的最当前功能，请升级到当前版本 2.6.7。

1.11.1. 了解版本

红帽在产品版本中使用语义版本。语义版本包括 3 个组件号，格式为 X.Y.Z，其中：

- X 代表主版本。主发行版本通常表示有主要的变化：架构更改、API 更改、模式更改以及类似的重大更新。
- Y 代表次版本。次发行版本包含了新功能，同时保持向后兼容性。
- z 代表一个补丁版本（也称为 z-stream 版本）。补丁版本用于提供解决常见漏洞和风险 (CVE) 的解决方案，以及版本中的程序错误修复。新特性通常不会作为补丁版本的一部分发布。

1.11.1.1. 版本对 Service Mesh 升级的影响

根据您要进行的更新版本，升级过程会有所不同。

- 补丁更新 - 由 Operator Lifecycle Manager(OLM)管理补丁升级；在更新 Operator 时会自动进行。

- 次版本更新 - 只需要升级到最新的 Red Hat OpenShift Service Mesh Operator 版本，并手动修改 ServiceMeshControlPlane 资源中的 spec.version 值。

- 主版本更新 - 主版本升级需要更新到最新的 Red Hat OpenShift Service Mesh Operator 版本，并手动修改 ServiceMeshControlPlane 资源中的 spec.version 值。因为主版本的更新可能包含不向后兼容的更改，所以可能需要进行额外的手动更改。

1.11.1.2. 了解 Service Mesh 版本

要了解您在系统上部署的 Red Hat OpenShift Service Mesh 版本，您需要了解如何管理各个组件版本。

- Operator 版本 - 最新版本为 2.6.7。Operator 版本号仅指示当前安装的 Operator 的版本。因为 Red Hat OpenShift Service Mesh Operator 支持 Service Mesh control plane 的多个版本，所以 Operator 的版本不会决定部署的 ServiceMeshControlPlane 资源的版本。

  重要

  升级到最新的 Operator 版本会自动应用补丁更新，但不会自动将 Service Mesh control plane 升级到最新的次版本。

- ServiceMeshControlPlane 版本 - ServiceMeshControlPlane 版本决定您使用的 Red Hat OpenShift Service Mesh 版本。ServiceMeshControlPlane 资源中的 spec.version 字段的值控制用于安装和部署 Red Hat OpenShift Service Mesh 的架构和配置设置。创建 Service Mesh control plane 时，您可以使用以下两种方式之一设置版本：

  - 要在 Form View 中配置，请从 Control Plane Version 菜单中选择版本。

  - 要在 YAML View 中配置，请在 YAML 文件中设置 spec.version 的值。

Operator Lifecycle Manager(OLM)不管理 Service Mesh control plane 升级，因此，Operator 和

ServiceMeshControlPlane (SMCP)的版本号可能不匹配，除非您手动升级 SMCP。

1.11.2. 升级注意事项

maistra.io/ 标签或注解不应用于用户创建的自定义资源，因为它表示资源由 Red Hat OpenShift Service Mesh Operator 生成并应该由 Red Hat OpenShift Service Mesh Operator 管理。

> **警告**
>
> 在升级过程中，Operator 进行更改（包括删除或替换文件）到包含以下标签或注解来指示 Operator 管理资源的资源。

在升级检查用户创建的自定义资源前，其中包含以下标签或注解：

- maistra.io/ 和 app.kubernetes.io/managed-by 标签设置为 maistra-istio-operator (Red Hat OpenShift Service Mesh)
- kiali.io/ (Kiali)
- jaegertracing.io/ （Red Hat OpenShift distributed Tracing Platform (Jaeger)
- logging.openshift.io/ (Red Hat Elasticsearch)

在升级前，检查用户为标签或注解创建自定义资源，以指示用户管理的 Operator。从您不想由 Operator 管理的自定义资源中删除标签或注解。

当升级到版本 2.0 时，Operator 只删除与 SMCP 相同的命名空间中使用这些标签的资源。

当升级到 2.1 版本时，Operator 会删除所有命名空间中使用这些标签的资源。

1.11.2.1. 可能会影响升级的已知问题

已知的可能影响升级的问题包括：

- 当升级 Operator 时，可能会恢复 Jaeger 或 Kiali 的自定义配置。在升级 Operator 前，请注意 Service Mesh production 部署中 Jaeger 或 Kiali 对象的任何自定义配置设置，以便您可以重新创建它们。

- Red Hat OpenShift Service Mesh 不支持使用 EnvoyFilter 配置，除非明确记录。这是因为与底层 Envoy API 耦合紧密，这意味着无法维护向后兼容性。如果您使用 Envoy Filters，并且因为升级 ServiceMeshControlPlane 引进了的 Envoy 版本导致 Istio 生成的配置已更改，则可能会破坏您可能已经实现的 EnvoyFilter。

- OSSM-1505 ServiceMeshExtension 无法用于 Red Hat OpenShift Service on AWS 版本 4.11。因为 ServiceMeshExtension 在 Red Hat OpenShift Service Mesh 2.2 中已弃用，所以这个已知问题不会被修复，且您必须将扩展迁移到 WasmPluging

- OSSM-1396 如果一个网关资源包含 spec.externalIPs 设置，而不是在 ServiceMeshControlPlane 更新时重新创建，则该网关会被删除且永远不会重新创建。

- OSSM-1052 在为服务网格 control plane 中为 ingressgateway 配置 Service ExternalIP 时，不会创建该服务。SMCP 的 schema 缺少该服务的参数。

  临时解决方案：禁用 SMCP spec 中的网关创建，手动管理网关部署（包括 Service、Role 和 RoleBinding）。

1.11.3. 升级 Operator

要让您的 Service Mesh 使用最新的安全修复、程序错误修正和软件更新进行补丁，您需要保持 Operator 为最新版本。您可以通过升级 Operator 来启动补丁更新。

重要

Operator 的版本 不会 决定服务网格的版本。部署的 Service Mesh control plane 的版本决定了您的 Service Mesh 版本。

因为 **Red Hat OpenShift Service Mesh Operator** 支持 **Service Mesh control plane** 的多个版本，所以更新 **Red Hat OpenShift Service Mesh Operator** *不会更新*部署的 **ServiceMeshControlPlane** 的 **spec.version** 值。另请注意，**spec.version** 值是一个双数字，如 **2.2**，补丁更新（如 **2.2.1**）不会反映在 **SMCP** 版本值中。

Operator Lifecycle Manager (OLM) 能够控制集群中 **Operator** 的安装、升级和基于角色的访问控制 **(RBAC)**。**OLM** 默认在 **Red Hat OpenShift Service on AWS** 中运行。**OLM** 会查询可用的 **Operator** 以及已安装的 **Operator** 的升级。

升级 **Operator** 的具体方式取决于您在安装 **Operator** 时选择的设置。安装每个 **Operator** 时，您可以选择一个更新频道和一个批准策略。这两个设置的组合决定了 **Operator** 的更新时间和方式。

表 1.4. 更新频道和批准策略的交互

	版本化频道	"Stable" 或 "Preview" 频道
自动	仅为该版本的次版本自动更新 Operator。将不会自动更新到下一个主要版本（即，从 2.0 升级到 3.0）。手动更改为升级到下一个主要版本所需的 Operator 订阅。	为所有主版本、次版本和补丁版本自动更新 Operator。
Manual（手动）	指定版本的次要和补丁版本需要手动更新。手动更改为升级到下一个主要版本所需的 Operator 订阅。	所有主版本、次版本和补丁版本需要手动更新。

更新 **Red Hat OpenShift Service Mesh Operator** 时，**Operator Lifecycle Manager(OLM)**会删除旧的 **Operator pod** 并启动新的 **pod**。新的 **Operator pod** 启动后，协调过程会检查 **ServiceMeshControlPlane (SMCP)**，如果存在可用于任何 **Service Mesh control plane** 组件的更新容器镜像，它会将这些 **Service Mesh control plane pod** 替换为使用新容器镜像的用户。

当您升级 **Kiali** 和 **Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator** 时，**OLM** 协调过程会扫描集群，并将受管实例升级到新 **Operator** 的版本。例如，如果您将 **Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator** 从 **1.30.2** 更新至 **1.34.1**，**Operator** 会扫描运行分布式 **Tracing Platform (Jaeger)**实例并将其升级到 **1.34.1**。

要保留 Red Hat OpenShift Service Mesh 的特定补丁版本，您需要禁用自动更新，并保留在该 Operator 的特定版本。

1.11.4. 升级 control plane

对于次版本和主版本，您必须手动更新 control plane。社区 Istio 项目建议进行金丝雀（Canary）升级，但 Red Hat OpenShift Service Mesh 只支持原位升级。Red Hat OpenShift Service Mesh 要求您按顺序升级到下一个次版本。例如，您必须从 2.0 升级到 2.1 版本，然后再升级到 2.2 版本。您无法直接从 Red Hat OpenShift Service Mesh 2.0 更新至 2.2。

升级服务网格 control plane 时，所有 Operator 受管资源（如网关）也会被升级。

虽然您可以在同一集群中部署多个 control plane 版本，但 Red Hat OpenShift Service Mesh 不支持服务网格的金丝雀升级。也就是说，您可以使用有不同的 spec.version 值的不同 SCMP 资源，但它们无法管理同一个网格。

有关迁移扩展的更多信息，请参阅从 ServiceMeshExtension 迁移到 WasmPlugin 资源。

1.11.4.1. 将版本 2.5 升级到 2.6 版本

1.11.4.1.1. Red Hat OpenShift distributed Tracing Platform (Jaeger)默认设置更改

此发行版本默认为 ServiceMeshControlPlane 资源的新实例禁用 Red Hat OpenShift Distributed Tracing Platform (Jaeger)。

将 ServiceMeshControlPlane 资源的现有实例更新至 Red Hat OpenShift Service Mesh 版本 2.6 时，分布式 Tracing Platform (Jaeger)会被默认启用。

Red Hat OpenShift Service Mesh 2.6 是最后一个发行版本，包括对 Red Hat OpenShift distributed Tracing Platform (Jaeger)和 OpenShift Elasticsearch Operator 的支持。分布式追踪平台 (Jaeger)和 OpenShift Elasticsearch Operator 将在以后的版本中删除。目前使用分布式跟踪平台 (Jaeger)和 OpenShift Elasticsearch Operator，则必须迁移到 Red Hat OpenShift distributed Tracing Platform 和 Red Hat build of OpenTelemetry。

1.11.4.1.2. Envoy sidecar 容器默认设置更改

为增强 pod 启动时间，Istio 现在默认在 sidecar 容器中包括 startupProbe。在 Envoy sidecar 启动前，pod 的就绪度探测不会启动。

1.11.4.2. 将更改从版本 2.4 升级到 2.5

1.11.4.2.1. Istio OpenShift 路由 (IOR) 默认设置更改

Istio OpenShift 路由 (IOR) 的默认设置已更改。现在默认禁用此设置。

您可以通过在 ServiceMeshControlPlane 资源的 spec.gateways.openshiftRoute 规格中将 enabled 字段设置为 true 来使用 IOR。

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  gateways:
    openshiftRoute:
      enabled: true
```

1.11.4.2.2. Istio 代理并发配置增强

为了维护部署间的一致性，Istio 现在根据分配给代理容器的 CPU 限制配置 concurrency 参数。例如，限制 2500m 会将 concurrency 参数设置为 3。如果将 concurrency 参数设置为一个值，Istio 将使用该值来配置代理运行的线程数量，而不是使用 CPU 限制。

在以前的版本中，参数的默认设置是 2。

1.11.4.3. 将更改从 2.3 升级到 2.4 版本

将 Service Mesh control plane 从 2.3 升级到 2.4 带来了以下行为更改：

- 对 Istio OpenShift 路由 (IOR) 的支持已弃用。IOR 功能仍处于启用状态，但将在以后的发行版本中删除。

- 以下密码套件不再被支持，并从客户端和服务器侧 TLS 协商中使用的密码列表中删除。

- ECDHE-ECDSA-AES128-SHA
- ECDHE-RSA-AES128-SHA
- AES128-GCM-SHA256
- AES128-SHA
- ECDHE-ECDSA-AES256-SHA
- ECDHE-RSA-AES256-SHA
- AES256-GCM-SHA384
- AES256-SHA

当代理启动 TLS 连接时，需要访问使用这些密码套件的服务的应用程序将无法连接。

1.11.4.4. 将更改从 2.2 升级到 2.3

将 Service Mesh control plane 从 2.2 升级到 2.3 包括以下行为更改：

- 此发行版本需要使用 WasmPlugin API。现在，在 2.2 中弃用的 ServiceMeshExtension API 的支持已被删除。如果您在使用 ServiceMeshExtension API 时尝试升级，则升级会失败。

1.11.4.5. 从版本 2.1 升级到 2.2 的变化

将 Service Mesh control plane 从 2.1 升级到 2.2 引进了以下行为更改：

- istio-node DaemonSet 被重命名为 istio-cni-node，以匹配上游 Istio 中的名称。

- Istio 1.10 更新了 Envoy，默认使用 eth0 而不是 lo 将流量发送到应用程序容器。

- 此发行版本添加了对 WasmPlugin API 的支持，并弃用 ServiceMeshExtension API。

1.11.4.6. 将版本 2.0 升级到 2.1 版

将 Service Mesh control plane 从 2.0 升级到 2.1 引进了以下架构和行为更改。

构架更改

在 Red Hat OpenShift Service Mesh 2.1 中已完全删除 Mixer。如果启用了 Mixer，则无法从 Red Hat OpenShift Service Mesh 2.0.x 版本升级到 2.1。

如果您在从 v2.0 升级到 v2.1 时看到以下信息，请在更新 .spec.version 字段前将现有 Mixer 类型更新为 Istiod 类型：

```
An error occurred
admission webhook smcp.validation.maistra.io denied the request: [support for policy.type
"Mixer" and policy.Mixer options have been removed in v2.1, please use another alternative,
support for telemetry.type "Mixer" and telemetry.Mixer options have been removed in v2.1,
please use another alternative]”
```

例如：

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  policy:
    type: Istiod
  telemetry:
    type: Istiod
  version: v2.6
```

行为更改

- **AuthorizationPolicy 更新：**
  - 使用 PROXY 协议时，如果您使用 ipBlocks 和 notIpBlocks 指定远程 IP 地址，请更新配置以使用 remoteIpBlocks 而不是 RemoteIpBlocks。
  - 添加了对嵌套 JSON Web Token(JWT)声明的支持。
- **EnvoyFilter 破坏更改>**
  - 必须使用 typed_config
  - XDS v2 不再被支持
  - 弃用的过滤器名称
- 较旧版本的代理可能会在从更新的代理接收 1xx 或 204 状态代码时报告 503 状态代码。

1.11.4.7. 升级 Service Mesh control plane

要升级 Red Hat OpenShift Service Mesh，您必须更新 Red Hat OpenShift Service Mesh ServiceMeshControlPlane v2 资源的版本字段。然后，在配置和应用后，重启应用程序 pod 以更新每个 sidecar 代理及其配置。

前提条件

- 您已在 AWS 4.9 或更高版本上运行 Red Hat OpenShift Service。
- 您有最新的 Red Hat OpenShift Service Mesh Operator。

流程

1.

切换到包含 ServiceMeshControlPlane 资源的项目。在本例中，istio-system 是 Service Mesh control plane 项目的名称。

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

2. 检查 v2 ServiceMeshControlPlane 资源配置以验证其是否有效。

   a. 运行以下命令，将您的 ServiceMeshControlPlane 资源视为 v2 资源。

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

   提示

   备份 Service Mesh control plane 配置。

3. 更新 .spec.version 字段并应用配置。

   例如：

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
```

另外，您可以使用 Web 控制台编辑 Service Mesh control plane，而不是使用命令行。在 Red Hat OpenShift Service on AWS Web 控制台中，点 Project 并选择您刚才输入的项目名称。

   a. 点 Operators → Installed Operators。

   b. 查找 ServiceMeshControlPlane 实例。

c. 选择 YAML view 并更新 YAML 文件的文本，如上例中所示。

d. 点 Save。

1.11.4.8. 将 Red Hat OpenShift Service Mesh 从版本 1.1 迁移到版本 2.0

从版本 1.1 升级到 2.0 需要手动步骤将工作负载和应用程序迁移到运行新版本的 Red Hat OpenShift Service Mesh 的新实例中。

前提条件

- 在升级到 Red Hat OpenShift Service Mesh 2.0 之前，您必须升级到 Red Hat OpenShift Service on AWS 4.7。
- 您必须具有 Red Hat OpenShift Service Mesh 版本 2.0 operator。如果选择了 自动 升级路径，Operator 将自动下载最新信息。但是，您必须执行一些步骤来使用 Red Hat OpenShift Service Mesh 版本 2.0 中的功能。

1.11.4.8.1. 升级 Red Hat OpenShift Service Mesh

要升级 Red Hat OpenShift Service Mesh，必须在新命名空间中创建一个 Red Hat OpenShift Service Mesh ServiceMeshControlPlane v2 资源实例。然后，配置完成后，将微服务应用程序和工作负载从旧的网格移到新服务网格中。

流程

1. 检查 v1 ServiceMeshControlPlane 资源配置，以确保它有效。

   a. 运行以下命令，将您的 ServiceMeshControlPlane 资源视为 v2 资源。

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

   b. 查看输出中的 spec.techPreview.errored.message 字段，以了解有关任何无效字段的信息。

c. 如果您的 v1 资源中存在无效字段，则该资源不会被协调，且无法作为 v2 资源编辑。v2 字段的所有更新都会被原始 v1 设置覆盖。要修复无效字段，可以替换、补丁或编辑资源的 v1 版本。您还可以在不修复的情况下删除资源。在资源修复后，它可以被协调，您可以修改或查看资源的 v2 版本。

d. 要通过编辑文件来修复资源，请使用 oc get 检索资源，在本地编辑文本文件，并将资源替换为您编辑的文件。

```
$ oc get smcp.v1.maistra.io <smcp_name> > smcp-resource.yaml
#Edit the smcp-resource.yaml file.
$ oc replace -f smcp-resource.yaml
```

e. 要使用补丁修复资源，请使用 oc patch。

```
$ oc patch smcp.v1.maistra.io <smcp_name> --type json --patch '[{"op":
"replace","path":"/spec/path/to/bad/setting","value":"corrected-value"}]'
```

f. 要通过使用命令行工具编辑资源，请使用 oc edit。

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

2. 备份 Service Mesh control plane 配置。切换到包含 ServiceMeshControlPlane 资源的项目。在本例中，istio-system 是 Service Mesh control plane 项目的名称。

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

3. 输入以下命令来检索当前的配置。您的 <smcp_name> 在您的 ServiceMeshControlPlane 资源元数据中指定，如 basic-install 或 full-install。

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

4. 将 ServiceMeshControlPlane 转换为 v2 control plane 版本，其包含您的配置信息作为起点。

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

5.

创建一个项目。在 Red Hat OpenShift Service on AWS 控制台项目菜单中，点 New Project，并为项目输入名称如 istio-system-upgrade。或者，您可以通过 CLI 运行这个命令。

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

6. 使用新项目名称更新 v2 ServiceMeshControlPlane 中的 metadata.namespace 字段。在本例中，使用 istio-system-upgrade。

7. 将 version 字段从 1.1 更新至 2.0，或将其从 v2 ServiceMeshControlPlane 中删除。

8. 在新命名空间中创建一个 ServiceMeshControlPlane。在命令行中，运行以下命令，使用您检索的 ServiceMeshControlPlane 的 v2 版本部署 control plane。在这个示例中将 '<smcp_name.v2> ' 替换为您的文件的路径。

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

另外，您可以使用控制台创建 Service Mesh control plane。在 Red Hat OpenShift Service on AWS web 控制台中，点 Project。然后选择您刚刚输入的项目名称。

a. 点 Operators → Installed Operators。

b. 点 Create ServiceMeshControlPlane。

c. 选择 YAML view，把获取的 YAML 文件的内容复制到这个项。检查 apiVersion 字段是否已设置为 maistra.io/v2，并修改 metadata.namespace 字段以使用新命名空间，如 istio-system-upgrade。

d. 点 Create。

1.11.4.8.2. 配置 2.0 ServiceMeshControlPlane

为 Red Hat OpenShift Service Mesh 版本 2.0 更改了 ServiceMeshControlPlane 资源。创建 ServiceMeshControlPlane 资源 v2 版本后，修改该资源以利用新功能并适合您的部署。在修改 ServiceMeshControlPlane 资源时，考虑对 Red Hat OpenShift Service Mesh 2.0 规范和行为进行以

下更改。您还可以参阅 Red Hat OpenShift Service Mesh 2.0 产品文档来获取您使用的功能的新信息。v2 资源必须用于 Red Hat OpenShift Service Mesh 2.0 安装。

1.11.4.8.2.1. 构架更改

之前的版本使用的架构单元已被 Istiod 替代。在 2.0 中，Service Mesh control plane 组件 Mixer、Pilot、Citadel、Galley 和 sidecar 注入程序功能已合并为一个组件 Istiod。

虽然 Mixer 不再作为 control plane 组件支持，但 Mixer 策略和遥测插件现在可以通过 Istiodi 中的 WASM 扩展支持。如果您需要集成旧的 Mixer 插件，则可为策略和遥测启用混合程序。

安全发现服务 Service（SDS）用于直接从 Istiod 向 sidecar 分发证书和密钥。在 Red Hat OpenShift Service Mesh 1.1 中，secret 由 Citadel 生成，代理使用它来检索其客户端证书和密钥。

1.11.4.8.2.2. 注解更改

v2.0 不再支持以下注解。如果使用其中一个注解，则必须更新工作负载，然后将其移至 v2.0 Service Mesh control plane。

- sidecar.maistra.io/proxyCPULimit 已被 sidecar.istio.io/proxyCPULimit 替代。如果您在工作负载上使用 sidecar.maistra.io 注解，则必须修改这些工作负载，使其使用相应的 sidecar.istio.io。

- sidecar.maistra.io/proxyMemoryLimit 已替换为 sidecar.istio.io/proxyMemoryLimit

- 不再支持 sidecar.istio.io/discoveryAddress。另外，默认发现地址已经从 pilot.<control_plane_namespace>.svc:15010（如果启用 mtls，使用端口 15011）变为 istiod-<smcp_name>.<control_plane_namespace>.svc:15012。

- 健康状态端口不再可以被配置，它被硬编码为 15021。* 如果您要定义自定义状态端口，如 status.sidecar.istio.io/port，则必须在将工作负载移至 v2.0 Service Mesh control plane 前删除覆盖。就绪度检查仍然可以通过将状态端口设置为 0 来禁用。

- Kubernetes Secret 资源不再用于为 sidecar 发布客户端证书。证书现在通过 Istiod 的 SDS 服务发布。如果您需要依赖挂载的 secret，则 v2.0 Service Mesh control plane 中的工作

负载将无法使用它们。

1.11.4.8.2.3. 行为更改

Red Hat OpenShift Service Mesh 2.0 中的一些功能与之前的版本不同。

- 网关上的就绪度端口已从 15020 移到 15021。
- 目标主机可见性包括 VirtualService 以及 ServiceEntry 资源。它包括所有通过 Sidecar 资源实施的限制。
- 默认启用自动 mutual TLS。代理到代理通信会自动配置为使用 mTLS，而不管是否有全局的验证策略。
- 当代理与 Service Mesh control plane 通讯时，无论 spec.security.controlPlane.mtls 设置如何，都始终使用安全连接。spec.security.controlPlane.mtls 设置仅在配置 Mixer 遥测或策略的连接时使用。

1.11.4.8.2.4. 不支持资源的迁移详情

Policy（策略）（authentication.istio.io/v1alpha1）不再被支持。

策略资源必须迁移到 v2.0 Service Mesh control planes、PeerAuthentication 和 RequestAuthentication 的新资源类型。根据策略资源中的具体配置，您可能需要配置多个资源来达到同样效果。

双向 TLS

使用 security.istio.io/v1beta1 PeerAuthentication 资源可以实现双向 TLS 强制。传统的 spec.peers.mtls.mode 字段会直接映射到新资源的 spec.mtls.mode 字段。选择条件已从在 spec.targets[x].name 中指定服务名称改为 spec.selector.matchLabels 中的标签选择器。在 PeerAuthentication 中，标签必须与目标列表中指定的服务选择器匹配。任何特定于端口的设置都需要映射到 spec.portLevelMtls。

身份验证

在 spec.origins 中指定的附加验证方法必须映射到 security.istio.io/v1beta1 RequestAuthentication 资源中。spec.selector.matchLabels 必须与 PeerAuthentication 上的相同字

段配置相类似。spec.origins.jwt 项中特定于 JWT 主体的配置会映射到 spec.rules 项中的类似字段。

- spec.origins[x].jwt.triggerRules 必须映射到一个或多个 security.istio.io/v1beta1 AuthorizationPolicy 资源。任何 spec.selector.labels 都必须配置为 RequestAuthentication 上的相同字段。
- spec.origins[x].jwt.triggerRules.excludedPaths 必须映射到一个 AuthorizationPolicy 中，其 spec.action 设置为 ALLOW，带有与排除路径匹配的 spec.rules[x].to.operation.path 条目。
- spec.origins[x].jwt.triggerRules.includedPaths 必须映射为一个独立的 AuthorizationPolicy，它的 spec.action 设置为 ALLOW，使用与包含的路径匹配的 spec.rules[x].to.operation.path 条目，以及与 Policy 资源中指定的 spec.origins[x].jwt.issuer 相对应的 spec.rules.[x].from.source.requestPrincipals 的条目。

ServiceMeshPolicy (maistra.io/v1)

ServiceMeshPolicy 通过 spec.istio.global.mtls.enabled（v1 资源）或 spec.security.dataPlane.mtls（v2 资源）自动为 Service Mesh control plane 配置。对于 v2 control plane，在安装过程中创建了一个功能等同的 PeerAuthentication 资源。此功能在 Red Hat OpenShift Service Mesh 版本 2.0 中已弃用

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

这些资源由 security.istio.io/v1beta1 AuthorizationPolicy 资源替代。

模拟 RbacConfig 行为需要编写默认的 AuthorizationPolicy，其设置取决于 RbacConfig 中指定的 spec.mode。

- 当将 spec.mode 设置为 OFF 时，不需要任何资源，因为默认策略是 ALLOW，除非对请求应用 AuthorizationPolicy。
- 当 spec.mode 设为 ON 时，设置 spec: {}。您必须为网格中的所有服务创建 AuthorizationPolicy 策略。
- spec.mode 设置为 ON_WITH_INCLUSION，必须为每个包含的命名空间中创建一个带有

spec: {} 的 AuthorizationPolicy。AuthorizationPolicy 不支持包括单独的服务。但是，当创建任何适用于该服务的工作负载的 AuthorizationPolicy 后，未明确允许的所有其他请求都将被拒绝。

- 当 spec.mode 设置为 ON_WITH_EXCLUSION 时，AuthorizationPolicy 不支持它。可创建一个全局 DENY 策略，但必须为每个网格中的每个工作负载创建一个 AuthorizationPolicy，因为没有可用于命名空间或工作负载的 allow-all 策略。

AuthorizationPolicy 包括配置适用的选择器的配置，它类似于 ServiceRoleBinding 提供的函数 ServiceRoleBinding，这与所提供函数 ServiceRole 相似。

ServiceMeshRbacConfig (maistra.io/v1)

这个资源由使用 Service Mesh control plane 命名空间中带有空 spec.selector 的 security.istio.io/v1beta1 AuthorizationPolicy 资源替换。该策略是应用于网格中所有工作负载的默认授权策略。如需具体迁移详情，请参阅上面的 RbacConfig。

1.11.4.8.2.5. Mixer 插件

在版本 2.0 中默认禁用 Mixer 组件。如果您的工作负载依赖 Mixer 插件，必须将 2.0 ServiceMeshControlPlane 版本配置为包含 Mixer 组件。

要启用 Mixer 策略组件，在 ServiceMeshControlPlane 中添加以下片断。

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

要启用 Mixer 遥测组件，在 ServiceMeshControlPlane 中添加以下片断。

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

旧版混合程序插件也可以迁移到 WASM，并使用新的 ServiceMeshExtension（maistra.io/v1alpha1）自定义资源进行集成。

Red Hat OpenShift Service Mesh 2.0 不提供上游 Istio 发行本中的内置 WASM 过滤器。

1.11.4.8.2.6. 双向 TLS 的变化

当使用带有特定工作负载 PeerAuthentication 策略的 mTLS 时，如果工作负载策略与命名空间/全局策略不同，则需要一个对应的 DestinationRule 来允许流量。

auto mTLS 默认启用，但可以通过将 ServiceMeshControlPlane 资源中的 spec.security.dataPlane.automtls 设置为 false 来禁用。禁用自动 mTLS 时，可能需要 DestinationRules 进行服务间的正常通信。例如，将一个命名空间的 PeerAuthentication 设置为 STRICT 可能会阻止其他命名空间中的服务访问它们，除非 DestinationRule 为命名空间中的服务配置 TLS 模式。

有关 mTLS 的详情，请参考启用 mutual Transport Layer Security (mTLS)

1.11.4.8.2.6.1. 其他 mTLS 示例

要在 info 示例应用程序中禁用 mTLS For productpage 服务，您的 Policy 资源需要为 Red Hat OpenShift Service Mesh v1.1 进行以下配置。

策略资源示例

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

要在 info 示例应用程序中禁用 mTLS For productpage 服务，请使用以下示例为 Red Hat OpenShift Service Mesh v2.0 配置 PeerAuthentication 资源。

PeerAuthentication 资源示例

```
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
```

为了在 info 示例应用程序中为 **productpage** 服务启用 mTLS，您的 **Policy** 资源被配置为 Red Hat OpenShift Service Mesh v1.1 如下。

策略资源示例

```
apiVersion: authentication.istio.io/v1alpha1
kind: Policy
metadata:
  name: productpage-mTLS-with-JWT
  namespace: <namespace>
spec:
  targets:
  - name: productpage
    ports:
    - number: 9000
  peers:
  - mtls:
  origins:
  - jwt:
      issuer: "https://securetoken.google.com"
      audiences:
      - "productpage"
      jwksUri: "https://www.googleapis.com/oauth2/v1/certs"
      jwtHeaders:
      - "x-goog-iap-jwt-assertion"
      triggerRules:
      - excludedPaths:
        - exact: /health_check
  principalBinding: USE_ORIGIN
```

要在 info 示例应用程序中为 productpage 服务启用 mTLS，使用以下示例为 Red Hat OpenShift Service Mesh v2.0 配置 PeerAuthentication 资源。

PeerAuthentication 资源示例

```
#require mtls for productpage:9000
apiVersion: security.istio.io/v1beta1
kind: 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
kind: 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"
    fromHeaders:
    - name: "x-goog-iap-jwt-assertion"
---
#Require JWT token to access product page service from
#any client to all paths except /health_check
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  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
```

1.11.4.8.3. 配置方案

您可以使用这些配置方案配置以下项目。

1.11.4.8.3.1. data plane 中的双向 TLS

通过 ServiceMeshControlPlane 资源中的 spec.security.dataPlane.mtls 为 data plane 通信配置双向 TLS，默认为 false。

1.11.4.8.3.2. 自定义签名密钥

Istiod 管理服务代理使用的客户端证书和私钥。默认情况下， Istiod 使用自签名证书作为签名，但您可以配置自定义证书和私钥。有关如何配置签名密钥的更多信息，请参阅添加外部证书颁发机构密钥和证书

1.11.4.8.3.3. Tracing

Tracing 在 spec.tracing 中配置。目前，Jaeger 是唯一支持的 tracer 类型。sampling 是一个缩放整数，代表 0.01% 增长，例如： 1 为 0.01%，10000 为 100%。可以指定追踪实施和抽样率：

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

Jaeger 在 ServiceMeshControlPlane 资源的 addons 部分进行配置。

```
spec:
  addons:
    jaeger:
      name: jaeger
      install:
        storage:
          type: Memory # or Elasticsearch for production mode
          memory:
            maxTraces: 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"
```

您可以使用 install 字段自定义 Jaeger 安装。容器配置，如资源限值是在 spec.runtime.components.jaeger 相关字段中配置的。如果存在与 spec.addons.jaeger.name 值匹配的 Jaeger 资源，Service Mesh control plane 将配置为使用现有安装。使用现有的 Jaeger 资源来完全自定义 Jaeger 安装。

1.11.4.8.3.4. 视觉化

Kiali 和 Grafana 在 ServiceMeshControlPlane 资源的 addons 部分进行配置。

```
spec:
  addons:
    grafana:
      enabled: true
```

```
    install: {} # customize install
  kiali:
    enabled: true
    name: kiali
    install: {} # customize install
```

Grafana 和 Kiali 安装可以通过相应的 install 字段自定义。容器自定义（如资源限制）在 spec.runtime.components.kiali 和 spec.runtime.components.grafana 中配置。如果存在与名称值匹配的现有 Kiali 资源，Service Mesh control plane 会配置用于 control plane 的 Kiali 资源。Kiali 资源中的一些字段会被覆盖，如 access_namespaces 列表，以及 Grafana、Prometheus 和追踪的端点。使用现有资源来完全自定义 Kiali 安装。

1.11.4.8.3.5. 资源使用和调度

资源在 spec.runtime.<component> 下配置。支持以下组件名称。

组件	描述	支持的版本
安全	Citadel 容器	v1.0/1.1
galley	Galley 容器	v1.0/1.1
pilot	Pilot/Istiod 容器	v1.0/1.1/2.0
mixer	istio-telemetry 和 istio-policy 容器	v1.0/1.1
mixer.policy	istio-policy 容器	v2.0
mixer.telemetry	istio-telemetry 容器	v2.0
global.oauthproxy	与不同附加组件搭配使用的 oauth-proxy 容器	v1.0/1.1/2.0
sidecarInjectorWebhook	Sidecar injector Webhook 容器	v1.0/1.1
trace.jaeger	常规 Jaeger 容器 - 可能不会应用所有设置。通过在 Service Mesh control plane 配置中指定现有 Jaeger 资源，支持完全自定义 Jaeger 安装。	v1.0/1.1/2.0
trace.jaeger.agent	特定于 Jaeger 代理的设置	v1.0/1.1/2.0
tracing.jaeger.allInOne	特定于 Jaeger allInOne 的设置	v1.0/1.1/2.0

组件	描述	支持的版本
tracing.jaeger.collector	针对 Jaeger 收集器的设置	v1.0/1.1/2.0
tracing.jaeger.elasticsearch	特定于 Jaeger elasticsearch 部署的设置	v1.0/1.1/2.0
trace.jaeger.query	特定于 Jaeger 查询的设置	v1.0/1.1/2.0
prometheus	prometheus 容器	v1.0/1.1/2.0
kiali	Kiali 容器 - 通过在 Service Mesh control plane 配置中指定现有 Kiali 资源来支持 Kiali 安装的完整自定义。	v1.0/1.1/2.0
grafana	Grafana 容器	v1.0/1.1/2.0
3scale	3scale 容器	v1.0/1.1/2.0
wasmExtensions.cacher	WASM 扩展缓存容器	v2.0 - 技术预览

一些组件支持资源限制和调度。如需更多信息，请参阅性能和可扩展性。

1.11.4.8.4. 迁移应用程序和工作负载的后续步骤

将应用程序工作负载移到新网格中，删除旧实例以完成您的升级。

1.11.5. 升级数据平面（data plane）

升级 control plane 后，您的数据平面仍然可以正常工作。但是，为了对 Envoy 代理应用更新以及代理配置的任何更改，您必须重启应用程序 pod 和工作负载。

1.11.5.1. 更新应用程序和工作负载

要完成迁移，重启网格中的所有应用程序 pod，以升级 Envoy sidecar 代理及其配置。

要对部署执行滚动更新，请使用以下命令：

```
$ oc rollout restart <deployment>
```

您必须对所有组成网格的应用程序执行滚动更新。

1.12. 管理用户和配置集

1.12.1. 创建 Red Hat OpenShift Service Mesh 成员

ServiceMeshMember 资源为 Red Hat OpenShift Service Mesh 管理员提供了一种将项目添加到服务网格的权限，即使对应用户没有服务网格项目或 member roll 的直接访问权限。虽然项目管理员被自动授予在其项目中创建 ServiceMeshMember 资源的权限，但它们不能将其指向任何 ServiceMeshControlPlane，直到服务网格管理员显式授予服务网格访问权限。管理员可以通过授予 mesh-user 用户角色来授予用户访问网格的权限。在本例中，istio-system 是 Service Mesh control plane 项目的名称。

```
$ oc policy add-role-to-user -n istio-system --role-namespace istio-system mesh-user
<user_name>
```

管理员可以修改 Service Mesh control plane 项目中的 mesh-user 角色绑定，以指定授予访问权限的用户和组。ServiceMeshMember 会将项目添加到它引用的 Service Mesh control plane 项目中的 ServiceMeshMemberRoll。

```
apiVersion: maistra.io/v1
kind: ServiceMeshMember
metadata:
  name: default
spec:
  controlPlaneRef:
    namespace: istio-system
    name: basic
```

mesh-users 角色绑定在管理员创建 ServiceMeshControlPlane 资源后自动创建。管理员可使用以下命令为用户添加角色。

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

管理员也可以在创建 ServiceMeshControlPlane 资源前创建 mesh-user 角色绑定。例如，管理员可以在与 ServiceMeshControlPlane 资源相同的 oc apply 操作中创建它。

本例为 alice添加一个角色绑定：

```
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. 创建 Service Mesh control plane 配置集

您可以使用 ServiceMeshControlPlane 配置集创建可重复使用的配置。个人用户可以根据自己的配置扩展他们创建的配置集。配置集也可以从其他配置集继承配置信息。例如，您可以为财务团队创建一个财务 control plane，为市场团队创建一个市场 control plane。如果您创建了一个开发模板和一个产品模板，则市场团队成员和财务团队成员就可以根据自己团队的情况对开发模板和生成环境配置集进行扩展。

当您配置 Service Mesh control plane 配置集时，它遵循与 ServiceMeshControlPlane 相同的语法，用户以分级方式继承设置。Operator 附带一个 默认 配置集，带有 Red Hat OpenShift Service Mesh 的默认设置。

1.12.2.1. 创建 ConfigMap

要添加自定义配置集，您必须在 openshift-operators 项目中创建一个名为 smcp-templates 的 ConfigMap。Operator 容器会自动挂载 ConfigMap。

前提条件

- 已安装并验证的 Service Mesh Operator。
- 具有 cluster-admin 角色的帐户。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 dedicated-admin 角色的帐户。

- Operator 部署的位置。

- 访问 OpenShift CLI（oc）。

流程

1. 以 cluster-admin 身份登录到 Red Hat OpenShift Service on AWS CLI。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 dedicated-admin 角色的帐户。

2. 在 CLI 中运行这个命令，在 openshift-operators 项目中创建名为 smcp-templates 的 ConfigMap，并将 <profiles-directory> 替换成本地磁盘上的 ServiceMeshControlPlane 文件的位置：

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

3. 您可以使用 ServiceMeshControlPlane 中的 profile 参数指定一个或多个模板。

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  profiles:
  - default
```

1.12.2.2. 设置正确的网络策略

Service Mesh 在 Service Mesh control plane 和成员命名空间中创建网络策略，以允许它们之间的流量。在部署前，请考虑以下条件，以确保之前通过 Red Hat OpenShift Service on AWS 路由公开的服务网格中的服务。

- 进入服务网格的流量必须总是经过 ingress-gateway 才能使 Istio 正常工作。

- 在不在任何服务网格中的独立命名空间中为服务网格部署服务。

-

需要在服务网格列出的命名空间中部署的非 mesh 服务应该标记其 maistra.io/expose-route: "true"，这可以确保 Red Hat OpenShift Service on AWS 路由到这些服务仍可以正常工作。

1.13. 安全性

如果您的服务网格应用程序由 一 组复杂的微服务组成，您可以使用 Red Hat OpenShift Service Mesh 来定制这些服务间的通信安全性。Red Hat OpenShift Service on AWS 的基础架构以及 Service Mesh 的流量管理功能可帮助您管理应用程序的复杂性和安全微服务。

开始前

如果您有一个项目，请将项目添加到 ServiceMeshMemberRoll 资源中。

如果您没有项目，请安装 Bookinfo 示例应用程序 并将其添加到 ServiceMeshMemberRoll 资源中。示例应用程序可以帮助演示安全概念。

1.13.1. 关于 mutual Transport Layer Security(mTLS)

Mutual Transport Layer Security(mTLS)是一个协议，可让双方相互验证。在一些协议（IKE、SSH）中，它是身份验证的默认模式，在其他协议中（TLS）是可选的。您可以在不更改应用程序或服务代码的情况下使用 mTLS。TLS 完全由服务网格基础架构处理，并在两个 sidecar 代理之间进行处理。

默认情况下，Red Hat OpenShift Service Mesh 中的 mTLS 被启用并设置为 permissive 模式，Service Mesh 中的 sidecar 接受明文流量和使用 mTLS 加密的连接。如果将网格中的服务配置为使用严格的 mTLS 与网格外的服务通信，则这些服务间的通信可能会中断，因为严格的 mTLS 需要客户端和服务器能够相互验证识别。在将工作负载迁移到 Service Mesh 时使用 permissive 模式。然后，您可以在网格、命名空间或应用程序间启用严格的 mTLS。

在 Service Mesh control plane 级别启用 mTLS 可保护服务网格中的所有流量，而无需重写应用程序和工作负载。您可以在 ServiceMeshControlPlane 资源中的 data plane 级别保护网格中的命名空间。要自定义流量加密连接，请使用 PeerAuthentication 和 DestinationRule 资源在应用级别上配置命名空间。

1.13.1.1. 在服务网格中启用严格的 mTLS

如果您的工作负载没有与外部服务通信，您可以在网格间快速启用 mTLS，而不中断通信。您可以通过在 ServiceMeshControlPlane 资源中将 spec.security.dataPlane.mtls 设置为 true 来启用它。

Operator 会创建所需资源。

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  version: v2.6
  security:
    dataPlane:
      mtls: true
```

您还可以使用 Red Hat OpenShift Service on AWS web 控制台启用 mTLS。

流程

1. 登录到 web 控制台。

2. 点 Project 菜单，选择安装 Service Mesh control plane 的项目，如 istio-system。

3. 点 Operators → Installed Operators。

4. 点 Provided APIs 下的 Service Mesh Control Plane。

5. 点 ServiceMeshControlPlane 资源的名称，例如 basic。

6. 在 Details 页面中，单击 Data Plane Security 的 Security 部分中的切换。

1.13.1.1.1. 为特定服务的入站连接配置 sidecar

您还可以通过创建策略为各个服务配置 mTLS。

流程

1. 使用以下示例创建 YAML 文件：

PeerAuthentication 策略示例 policy.yaml

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: <namespace>
spec:
  mtls:
    mode: STRICT
```

a. 将 <namespace> 替换为该服务所在的命名空间。

2. 运行以下命令，在服务所在的命名空间中创建资源。它必须与您刚才创建的 Policy 资源中的 namespace 字段匹配。

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

注意

如果您不使用自动 mTLS，并且要将 PeerAuthentication 设置为 STRICT，则必须为您的服务创建一个 DestinationRule 资源。

1.13.1.1.2. 为出站连接配置 sidecar

创建一个目标规则将 Service Mesh 配置为在向网格中的其他服务发送请求时使用 mTLS。

流程

1. 使用以下示例创建 YAML 文件：

DestinationRule 示例 destination-rule.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. 将 <namespace> 替换为该服务所在的命名空间。

2. 运行以下命令，在服务所在的命名空间中创建资源。它必须与您刚才创建的 DestinationRule 资源中的 namespace 字段匹配。

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

1.13.1.1.3. 设置最小和最大协议版本

如果您的环境对服务网格中的加密流量有具体要求，可以通过在 ServiceMeshControlPlane 资源中设置 spec.security.controlPlane.tls.minProtocolVersion 或 spec.security.controlPlane.tls.maxProtocolVersion 来控制允许的加密功能。这些值在 Service Mesh control plane 资源中配置，定义网格组件在通过 TLS 安全通信时使用的最小和最大 TLS 版本。

默认为 TLS_AUTO，且不指定 TLS 版本。

表 1.5. 有效值

值	描述
TLS_AUTO	default
TLSv1_0	TLS 版本 1.0
TLSv1_1	TLS 版本 1.1
TLSv1_2	TLS 版本 1.2

值	描述
TLSv1_3	TLS 版本 1.3

流程

1. **登录到 web 控制台。**

2. **点 Project 菜单，选择安装 Service Mesh control plane 的项目，如 istio-system。**

3. **点 Operators → Installed Operators。**

4. **点 Provided APIs 下的 Service Mesh Control Plane。**

5. **点 ServiceMeshControlPlane 资源的名称，例如 basic。**

6. **点 YAML 标签。**

7. **在 YAML 编辑器中插入以下代码片段：将 minProtocolVersion 中的值替换为 TLS 版本值。在本例中，最小 TLS 版本设置为 TLSv1_2。**

ServiceMeshControlPlane 代码片段

```
kind: ServiceMeshControlPlane
spec:
  security:
    controlPlane:
      tls:
        minProtocolVersion: TLSv1_2
```

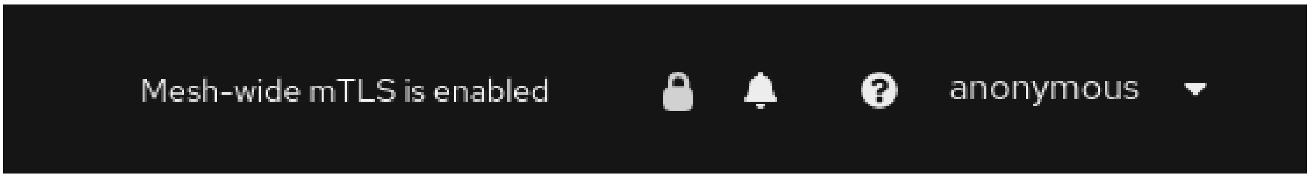
8. 点 **Save**。

9. 单击 **Refresh** 以验证更改是否已正确更新。

1.13.1.2. 使用 Kiali 验证加密

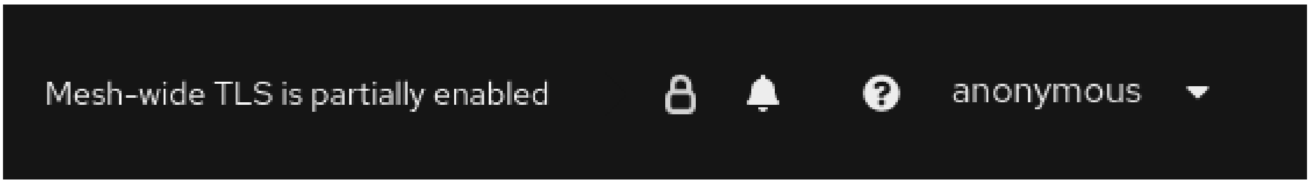
Kiali 控制台提供了多种方式来验证应用程序、服务和工作负载是否启用了 mTLS 加密。

图 1.5. masthead 图标网格范围 mTLS



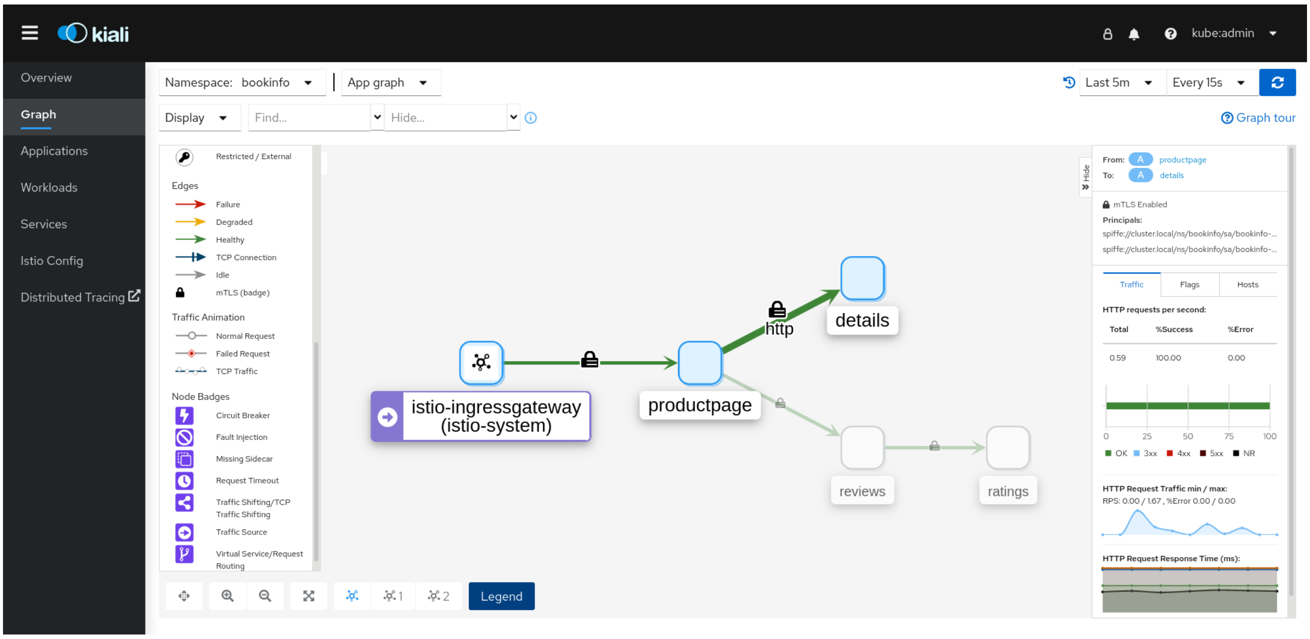
在 masthead 右侧，Kiali 显示一个锁定图标，当网格为整个服务网格启用了 mTLS 时。这意味着网格中的所有通信都使用 mTLS。

图 1.6. masthead 图标网格范围 mTLS 部分启用



当网格以 PERMISSIVE 模式或者网格范围 mTLS 配置出现错误时，Kiali 会显示 hollow 锁定图标。

图 1.7. 安全徽标



Graph 页面有选项，可以在图形边缘上显示 Security badge 来指示启用 mTLS。要在图形上启用安全徽标，请从 Display 菜单的 显示 Badges 下选择 安全 复选框。当边缘显示锁定图标时，它表示至少有一个启用了 mTLS 的请求。如果同时存在 mTLS 和非mTLS 请求，则 side-panel 会显示使用 mTLS 的请求百分比。

Applications details Overview 页会在图形边缘上显示一个 Security 图标，其中至少有一个启用了 mTLS 的请求。

Workloads details Overview 页会在图形边缘上显示一个 Security 图标，其中至少有一个启用了 mTLS 的请求。

Services Details Overview 页会在图形边缘上显示一个 Security 图标，其中至少有一个启用了 mTLS 的请求。另外请注意，Kiali 在 Network 部分显示为 mTLS 配置的端口旁的锁定图标。

1.13.2. 配置基于角色的访问控制（RBAC）

基于角色的访问控制 (RBAC) 对象决定是否允许用户或服务在项目内执行给定的操作。您可以为网格中的工作负载定义 mesh-、namespace- 和工作负载范围访问控制。

要配置 RBAC，在您要配置访问权限的命名空间中创建一个 AuthorizationPolicy 资源。如果要配置网格范围访问，请使用在其中安装 Service Mesh control plane 的项目，如 istio-system。

例如，对于 RBAC，您可以创建以下策略：

- 配置项目内部通信。
- 允许或拒绝对默认命名空间中所有工作负载的完全访问。
- 允许或拒绝入口网关访问。
- 需要令牌才能访问。

授权策略包括选择器、操作和规则列表：

- selector 字段指定策略的目标。
- action 字段指定是否允许或拒绝请求。
- rules 字段指定何时触发操作。
  - from 字段指定请求来源的限制。
  - to 字段指定请求目标和参数的限制。
  - when 字段指定应用该规则的其他条件。

流程

1. 创建 AuthorizationPolicy 资源。以下示例显示了一个更新 ingress-policy AuthorizationPolicy 的资源，以拒绝 IP 地址访问入口网关。

```
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. 在编写资源以便在命名空间中创建资源后运行以下命令。命名空间必须与 AuthorizationPolicy 资源中的 metadata.namespace 字段匹配。

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

后续步骤

考虑以下示例用于其他通用配置。

1.13.2.1. 配置项目内部通信

您可以使用 AuthorizationPolicy 配置 Service Mesh control plane 来允许或拒绝与网格中的网格或服务通信的流量。

1.13.2.1.1. 限制对命名空间外服务的访问

您可以使用以下 AuthorizationPolicy 资源示例拒绝来自 info 命名空间中没有的源的请求。

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: httpbin-deny
 namespace: info
spec:
 selector:
   matchLabels:
     app: httpbin
     version: v1
 action: DENY
 rules:
```

```
- from:
  - source:
      notNamespaces: ["info"]
```

1.13.2.1.2. 创建 allow-all 和 default deny-all 授权策略

以下示例显示了一个 allow-all 授权策略，允许对 info 命名空间中的所有工作负载进行完全访问。

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

以下示例显示了拒绝对 info 命名空间中所有工作负载的访问的策略。

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

1.13.2.2. 允许或拒绝对入口网关的访问

您可以设置一个授权策略来根据 IP 地址添加 allow 或 deny 列表。

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: ALLOW
  rules:
  - from:
    - source:
        ipBlocks: ["1.2.3.4", "5.6.7.0/24"]
```

1.13.2.3. 限制使用 JSON Web 令牌的访问

您可以使用 JSON Web Token（JWT）限制可以访问您的网格的内容。验证后，用户或服务可以访问路由，与该令牌关联的服务。

创建 RequestAuthentication 资源，用于定义工作负载支持的身份验证方法。下面的例子接受由 http://localhost:8080/auth/realms/master 发布的 JWT。

```
apiVersion: "security.istio.io/v1beta1"
kind: "RequestAuthentication"
metadata:
  name: "jwt-example"
  namespace: info
spec:
  selector:
    matchLabels:
      app: httpbin
  jwtRules:
  - issuer: "http://localhost:8080/auth/realms/master"
    jwksUri: "http://keycloak.default.svc:8080/auth/realms/master/protocol/openid-connect/certs"
```

然后，在同一命名空间中创建一个 AuthorizationPolicy 资源，以用于您创建的 RequestAuthentication 资源。以下示例在向 httpbin 工作负载发送请求时，需要在 Authorization 标头中有一个 JWT。

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

1.13.3. 配置密码套件和 ECDH curves（策展）

密码套件和 Elliptic-curve Diffie-Hellman（ECDH 策展）可以帮助您保护服务网格的安全。您可以使用 spec.security.controlplane.tls.cipherSuites 和 ECDH 策展在 ServiceMeshControlPlane 资源中使

用 spec.istio.global.tls.ecdhCurves 定义以逗号隔开的密码套件列表。如果其中任何一个属性为空，则使用默认值。

如果您的服务网格使用 TLS 1.2 或更早版本， cipherSuites 设置就会有效。它在使用 TLS 1.3 时无效。

在以逗号分开的列表中设置密码组合，以优先级顺序进行排列。例如，ecdhCurves: CurveP256, CurveP384 把 CurveP256 设置为比 CurveP384 有更高的优先级。

注意

在配置加密套件时，需要包括 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 或 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256。HTTP/2 的支持至少需要其中一个加密套件。

支持的加密套件是：

- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256

- **TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA**
- **TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256**
- **TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA**
- **TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA**
- **TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA**
- **TLS_RSA_WITH_AES_128_GCM_SHA256**
- **TLS_RSA_WITH_AES_256_GCM_SHA384**
- **TLS_RSA_WITH_AES_128_CBC_SHA256**
- **TLS_RSA_WITH_AES_128_CBC_SHA**
- **TLS_RSA_WITH_AES_256_CBC_SHA**
- **TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA**
- **TLS_RSA_WITH_3DES_EDE_CBC_SHA**

支持的 ECDH Curves 是：

- CurveP256
- CurveP384
- CurveP521
- X25519

1.13.4. 配置 JSON Web 密钥设置解析器证书颁发机构

您可以从 ServiceMeshControlPlane (SMCP) spec 配置您自己的 JSON Web Key Sets (JWKS)解析器证书颁发机构(CA)。

流程

1. 编辑 ServiceMeshControlPlane spec 文件：

```
$ oc edit smcp <smcp-name>
```

2. 通过将 ServiceMeshControlPlane spec 中的 mtls 字段设置为 true，为数据平面启用 mtls，如下例所示：

```
spec:
  security:
    dataPlane:
      mtls: true # enable mtls for data plane
    # JWKSResolver extra CA
    # PEM-encoded certificate content to trust an additional CA
    jwksResolverCA: |
      -----BEGIN CERTIFICATE-----
      [...]
      [...]
      -----END CERTIFICATE-----
...
```

3. 保存更改。Red Hat OpenShift Service on AWS 会自动应用它们。

一个 ConfigMap（如 pilot-jwks-cacerts-<SMCP name>）是由 CA .pem data 创建的。

ConfigMap pilot-jwks-cacerts-<SMCP name> 示例

```
kind: ConfigMap
apiVersion: v1
data:
  extra.pem: |
      -----BEGIN CERTIFICATE-----
      [...]
      [...]
      -----END CERTIFICATE-----
```

1.13.5. 添加外部证书颁发机构密钥和证书

默认情况下，Red Hat OpenShift Service Mesh 生成自签名 root 证书和密钥，并使用它们为工作负载证书签名。您还可以使用用户定义的证书和密钥使用用户定义的 root 证书为工作负载证书签名。此任务演示了一个将证书和密钥插入 Service Mesh 的示例。

前提条件

- 在启用了 mutual TLS 配置证书的情况下安装 Red Hat OpenShift Service Mesh。
- 本例使用 Maistra 仓库中的证书。对于生产环境，请使用您自己的证书颁发机构提供的证书。
- 部署 Bookinfo 示例应用程序以按照以下说明验证结果。
- 需要 openssl 才能验证证书。

1.13.5.1. 添加一个现有证书和密钥

要使用现有签名（CA）证书和密钥，必须创建一个信任文件链，其中包括 CA 证书、密钥和 root 证

书。您必须为每个对应证书使用以下准确文件名称。CA 证书名为 ca-cert.pem，密钥是 ca-key.pem，签名 ca-cert.pem 的 root 证书名为 root-cert.pem。如果您的工作负载使用中间证书，则必须在 cert-chain.pem 文件中指定它们。

1. 本地保存 Maistra repo 中的示例证书,，将 <path> 替换为证书的路径。

2. 创建名为 cacert 的 secret，其中包含输入文件 ca-cert.pem、ca-key.pem、root-cert.pem 和 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
```

3. 在 ServiceMeshControlPlane 资源中将 spec.security.dataPlane.mtls 设置为 true，并配置 certificateAuthority 字段，如下例所示。默认 rootCADir 是 /etc/cacerts。如果在默认位置挂载了密钥和证书，则不需要设置 privateKey。Service Mesh 从 secret-mount 文件中读取证书和密钥。

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

4. 创建/更改/取消设置 cacert secret 后，Service Mesh control plane istiod 和 gateway pod 必须重启，以便更改生效。使用以下命令重启 pod：

```
$ oc -n istio-system delete pods -l 'app in (istiod,istio-ingressgateway, istio-egressgateway)'
```

Operator 会在 pod 被删除后自动重新创建。

5. 重启 info 应用程序 pod，以便 sidecar 代理获取 secret 的更改。使用以下命令重启 pod：

```
$ oc -n info delete pods --all
```

您应该看到类似如下的输出：

```
pod "details-v1-6cd699df8c-j54nh" deleted
pod "productpage-v1-5ddcb4b84f-mtmf2" deleted
pod "ratings-v1-bdbcc68bc-kmng4" deleted
pod "reviews-v1-754ddd7b6f-lqhsv" deleted
pod "reviews-v2-675679877f-q67r2" deleted
pod "reviews-v3-79d7549c7-c2gjs" deleted
```

6. 使用以下命令验证 pod 是否已创建并就绪：

```
$ oc get pods -n info
```

1.13.5.2. 验证您的证书

使用 Bookinfo 示例应用程序来验证工作负载证书是否由插入到 CA 的证书签名。这个过程要求您在机器上安装 openssl。

1. 要从 info 工作负载中提取证书，请使用以下命令：

```
$ sleep 60
$ oc -n info exec "$(oc -n bookinfo get pod -l app=productpage -o jsonpath=
{.items..metadata.name})" -c istio-proxy -- openssl s_client -showcerts -connect
details:9080 > bookinfo-proxy-cert.txt
$ sed -n '/-----BEGIN CERTIFICATE-----/{:start /-----END CERTIFICATE-----/!{N;b
start};/.*/p}' info-proxy-cert.txt > certs.pem
$ awk 'BEGIN {counter=0;} /BEGIN CERT/{counter++} { print > "proxy-cert-" counter
".pem"}' < certs.pem
```

运行此命令后，您的工作目录中应当有三个文件： proxy-cert-1.pem、proxy-cert-2.pem 和 proxy-cert-3.pem。

2. 验证 root 证书是否与管理员指定证书相同。将 <path> 替换为证书的路径。

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

在终端窗口中运行以下语法：

```
$ openssl x509 -in ./proxy-cert-3.pem -text -noout > /tmp/pod-root-cert.crt.txt
```

通过在终端窗口中运行以下命令来比较证书。

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

您应看到以下结果：Files /tmp/root-cert.crt.txt and /tmp/pod-root-cert.crt.txt are identical

3. 验证 CA 证书是否与管理员指定证书相同。将 <path> 替换为证书的路径。

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

在终端窗口中运行以下语法：

```
$ openssl x509 -in ./proxy-cert-2.pem -text -noout > /tmp/pod-cert-chain-ca.crt.txt
```

通过在终端窗口中运行以下命令来比较证书。

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

您应看到以下结果：Files /tmp/ca-cert.crt.txt and /tmp/pod-cert-chain-ca.crt.txt are identical.

4. 从 root 证书到工作负载证书验证证书链。将 <path> 替换为证书的路径。

```
$ openssl verify -CAfile <(cat <path>/ca-cert.pem <path>/root-cert.pem) ./proxy-cert-1.pem
```

您应看到以下结果：./proxy-cert-1.pem: OK

1.13.5.3. 删除证书

要删除您添加的证书，请按照以下步骤操作。

1. 删除 secret cacert。在本例中，istio-system 是 Service Mesh control plane 项目的名称。

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

2. 在 ServiceMeshControlPlane 资源中使用自签名 root 证书重新部署 Service Mesh。

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

1.13.6. 关于将 Service Mesh 与 cert-manager 和 istio-csr 集成

cert-manager 工具是 Kubernetes 上 X.509 证书管理的解决方案。它提供了一个统一的 API，将应用程序与私钥或公钥基础架构 (PKI) 集成，如 Vault、Google Cloud Certificate Authority Service、Ret 的 Encrypt 和其他供应商。

cert-manager 工具通过尝试在配置的时间尝试续订证书，确保证书有效且最新。

对于 Istio 用户，cert-manager 还提供与 istio-csr 集成，它是一个处理 Istio 代理的证书签名请求 (CSR) 的证书颁发机构 (CA) 服务器。然后，服务器将签名委托给 cert-manager，它将 CSR 转发到配置的 CA 服务器。

注意

红帽支持与 istio-csr 和 cert-manager 集成。红帽不提供对 istio-csr 或社区 cert-manager 组件的直接支持。此处所示的社区 cert-manager 的使用仅用于演示目的。

前提条件

- 这些 cert-manager 版本之一：

  - cert-manager Operator for Red Hat OpenShift 1.10 或更高版本
  - 社区 cert-manager Operator 1.11 或更高版本
  - cert-manager 1.11 或更高版本
- OpenShift Service Mesh Operator 2.4 或更高版本
- istio-csr 0.6.0 或更高版本

> **注意**
>
> 为了避免在使用 jetstack/cert-manager-istio-csr Helm chart 安装 istio-csr 服务器时在所有命名空间中创建配置映射，在 istio-csr.yaml 文件中使用以下设置：app.controller.configmapNamespaceSelector: "maistra.io/member-of: <istio-namespace>"。

1.13.6.1. 安装 cert-manager

您可以安装 cert-manager 工具来管理 TLS 证书的生命周期，并确保它们有效且最新。如果您在您的环境中运行 Istio，您还可以安装 istio-csr 证书颁发机构 (CA) 服务器，该服务器处理 Istio 代理中的证书签名请求 (CSR)。istio-csr CA 将签名委派给 cert-manager 工具，该工具委托给配置的 CA。

流程

1. 创建 root 集群签发者：

   a. 创建 cluster-issuer 对象，如下例所示：

cluster-issuer.yaml 示例

```
apiVersion: cert-manager.io/v1
```

```yaml
kind: Issuer
metadata:
  name: selfsigned-root-issuer
  namespace: cert-manager
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: root-ca
  namespace: cert-manager
spec:
  isCA: true
  duration: 21600h # 900d
  secretName: root-ca
  commonName: root-ca.my-company.net
  subject:
    organizations:
    - my-company.net
  issuerRef:
    name: selfsigned-root-issuer
    kind: Issuer
    group: cert-manager.io
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: root-ca
spec:
  ca:
    secretName: root-ca
```

注意

selfsigned-root-issuer issuer 和 root-ca 证书的命名空间是 cert-manager，因为 root-ca 是集群签发者，因此 cert-manager 会在其自己的命名空间中查找引用的 secret。在 Red Hat OpenShift 的 cert-manager Operator 时，命名空间称为 cert-manager。

b. 使用以下命令创建对象：

```
$ oc apply -f cluster-issuer.yaml
```

c. 创建 istio-ca 对象，如下例所示：

istio-ca.yaml 示例

```yaml
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: istio-ca
  namespace: istio-system
spec:
  isCA: true
  duration: 21600h
  secretName: istio-ca
  commonName: istio-ca.my-company.net
  subject:
    organizations:
    - my-company.net
  issuerRef:
    name: root-ca
    kind: ClusterIssuer
    group: cert-manager.io
---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: istio-ca
  namespace: istio-system
spec:
  ca:
    secretName: istio-ca
```

d. 使用以下命令来创建对象：

```
$ oc apply -n istio-system -f istio-ca.yaml
```

2. 安装 istio-csr ：

```
$ helm install istio-csr jetstack/cert-manager-istio-csr \
    -n istio-system \
    -f deploy/examples/cert-manager/istio-csr/istio-csr.yaml
```

istio-csr.yaml 示例

```yaml
replicaCount: 2

image:
  repository: quay.io/jetstack/cert-manager-istio-csr
  tag: v0.6.0
  pullSecretName: ""

app:
  certmanager:
    namespace: istio-system
    issuer:
      group: cert-manager.io
      kind: Issuer
      name: istio-ca

  controller:
    configmapNamespaceSelector: "maistra.io/member-of=istio-system"
    leaderElectionNamespace: istio-system

  istio:
    namespace: istio-system
    revisions: ["basic"]

  server:
    maxCertificateDuration: 5m

  tls:
    certificateDNSNames:
    # This DNS name must be set in the SMCP spec.security.certificateAuthority.cert-manager.address
    - cert-manager-istio-csr.istio-system.svc
```

3. 部署 SMCP：

```
$ oc apply -f mesh.yaml -n istio-system
```

mesh.yaml 示例

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
```

```
  name: basic
spec:
  addons:
    grafana:
      enabled: false
    kiali:
      enabled: false
    prometheus:
      enabled: false
  proxy:
    accessLogging:
      file:
        name: /dev/stdout
  security:
    certificateAuthority:
      cert-manager:
        address: cert-manager-istio-csr.istio-system.svc:443
      type: cert-manager
    dataPlane:
      mtls: true
    identity:
      type: ThirdParty
  tracing:
    type: None
---
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
spec:
  members:
  - httpbin
  - sleep
```

注意

当配置了 security.certificateAuthority.type: cert-manager 时必须设置 security.identity.type: ThirdParty。

验证

使用示例 httpbin 服务和 sleep 应用程序来检查来自入口网关的 mTLS 流量，并验证安装了 cert-manager 工具。

1. 部署 HTTP 和 sleep 应用程序：

```
$ oc new-project <namespace>
```

```
$ oc apply -f https://raw.githubusercontent.com/maistra/istio/maistra-2.4/samples/httpbin/httpbin.yaml
```

```
$ oc apply -f https://raw.githubusercontent.com/maistra/istio/maistra-2.4/samples/sleep/sleep.yaml
```

2. 验证 sleep 是否可以访问 httpbin 服务：

```
$ oc exec "$(oc get pod -l app=sleep -n <namespace> \
  -o jsonpath={.items..metadata.name})" -c sleep -n <namespace> -- \
  curl http://httpbin.<namespace>:8000/ip -s -o /dev/null \
  -w "%{http_code}\n"
```

输出示例

```
200
```

3. 检查来自 ingress 网关到 httpbin 服务的 mTLS 流量：

```
$ oc apply -n <namespace> -f https://raw.githubusercontent.com/maistra/istio/maistra-2.4/samples/httpbin/httpbin-gateway.yaml
```

4. 获取 istio-ingressgateway 路由：

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

5. 验证从 ingress 网关到 httpbin 服务的 mTLS 流量：

```
$ curl -s -I http://$INGRESS_HOST/headers -o /dev/null -w "%{http_code}" -s
```

1.13.7. 其他资源

有关如何为 Red Hat OpenShift Service on AWS 安装 cert-manager Operator 的详情，请参考：为 Red Hat OpenShift 安装 cert-manager Operator。

1.14. 管理服务网格中的流量

使用 Red Hat OpenShift Service Mesh，您可以控制服务间的流量和 API 调用流。服务网格中的一些服务可能需要在网格内进行通信，其他服务则需要隐藏。您可以管理流量来隐藏特定后端服务、公开服务、创建测试或版本部署，或者在一组服务中添加安全层。

1.14.1. 使用网关

您可以使用网关来管理入站和出站流量，以指定您想要进入或离开网格的流量。网关配置适用于在网格边缘运行的独立的 Envoy 代理，而不是与您的服务负载一同运行的 sidecar Envoy 代理。

与控制进入系统的其他流量的机制不同，如 Kubernetes Ingress API，Red Hat OpenShift Service Mesh 网关可让您获得流量路由所具有的优点和灵活性。

Red Hat OpenShift Service Mesh 网关资源可使用层 4-6 负载均衡属性，比如要公开和配置 Red Hat OpenShift Service Mesh TLS 设置的端口。您可以将常规 Red Hat OpenShift Service Mesh 虚拟服务绑定到网关，并像服务网格中的其它数据平面流量一样管理网关流量，而不将应用程序层流量路由(L7)添加到相同的 API 资源中。

网关主要用于管理入口流量，但您也可以配置出口网关。出口网关可让您为离开网格的流量配置专用退出节点。这可让您限制哪些服务可以访问外部网络，这会为您的服务网格增加安全控制。您还可以使用网关配置纯内部代理。

网关示例

网关资源描述了在网格边缘运行的负载均衡器，接收进入或传出的 HTTP/TCP 连接。该规范描述应当公开的一组端口，要使用的协议类型，以及负载均衡器的 SNI 配置等。

以下示例显示了外部 HTTPS 入口流量的网关配置示例：

```
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
```

这个网关配置允许来自 ext-host.example.com 的 HTTPS 流量通过端口 443 进入网格，但没有为流量指定路由。

要指定路由并让网关按预期工作，还必须将网关绑定到虚拟服务。您可以使用虚拟服务的网关字段进行此操作，如下例所示：

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

然后，您可以使用外部流量的路由规则配置虚拟服务。

1.14.1.1. 管理入口流量

在 Red Hat OpenShift Service Mesh 中，Ingress Gateway 允许监控、安全性和路由规则等功能应用到进入集群的流量。使用 Service Mesh 网关在服务网格外公开服务。

1.14.1.1.1. 决定入口 IP 和端口

入口配置根据您的环境是否支持外部负载均衡器而有所不同。在集群的入口 IP 和端口中设置一个外部负载均衡器。要确定是否为外部负载均衡器配置了集群的 IP 和端口，请运行以下命令。在本例中，istio-system 是 Service Mesh control plane 项目的名称。

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

该命令会返回命名空间中每个项目的 NAME、TYPE、CLUSTER-IP、EXTERNAL-IP、PORT(S) 和 AGE。

如果设置了 EXTERNAL-IP 值，您的环境会有一个外部负载均衡器，供您用于入口网关。

如果 EXTERNAL-IP 值是 <none>，或 <pending>，则您的环境不会为入口网关提供外部负载均衡器。

1.14.1.1.1.1. 使用负载均衡器确定入口端口

如果您的环境有外部负载均衡器，请按照以下步骤操作。

流程

1. 运行以下命令来设置入口 IP 和端口。此命令在终端中设置变量。

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

2. 运行以下命令来设置入口端口。

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

3. 运行以下命令来设置安全入口端口。

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

4. 运行以下命令来设置 TCP 入口端口。

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

注意

在某些情况下，负载均衡器可能会使用主机名而不是 IP 地址公开。在这种情况下，入口网关的 EXTERNAL-IP 值不是一个 IP 地址。相反，这是一个主机名，上一命令无法设置 INGRESS_HOST 环境变量。

在这种情况下，使用以下命令更正 INGRESS_HOST 值：

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

1.14.1.1.1.2. 确定没有负载均衡器的入口端口

如果您的环境没有外部负载均衡器，请确定入口端口并改用节点端口。

流程

1. 设置入口端口。

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

2. 运行以下命令来设置安全入口端口。

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

3. 运行以下命令来设置 TCP 入口端口。

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

1.14.1.2. 配置入口网关

入口网关是在网格边缘运行的负载均衡器，接收传入的 HTTP/TCP 连接。它配置公开的端口和协议，但不包括任何流量路由配置。入口流量的流量路由改为使用路由规则配置，这与内部服务请求相同。

以下步骤演示了如何创建网关并配置 VirtualService，以在 Bookinfo 示例应用程序中将服务公开给路径 /productpage 和 /login 的外部流量。

流程

1. 创建网关以接受流量。

   a. 创建 YAML 文件，并将以下 YAML 复制到其中：

   网关 gateway.yaml 示例

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: info-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

   b. 应用 YAML 文件。

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

2. 创建 VirtualService 对象来重写主机标头。

   a. 创建 YAML 文件，并将以下 YAML 复制到其中：

   虚拟服务示例

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: info
spec:
  hosts:
  - "*"
  gateways:
  - info-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. 应用 YAML 文件。

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

3. 测试网关和 VirtualService 已正确设置。

a. 设置网关 URL。

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

b. 设置端口号。在本例中，istio-system 是 Service Mesh control plane 项目的名称。

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

c. 测试已明确公开的页面。

```
curl -s -I "$GATEWAY_URL/productpage"
```

预期的结果为 200。

1.14.2. 了解自动路由

重要

Istio OpenShift 路由 (IOR) 是一个已弃用的功能。弃用的功能仍然包含在 Red Hat OpenShift Service on AWS 中，并且仍然被支持。但是，弃用的功能可能会在以后的发行版本中被删除，且不建议在新的部署中使用。

有关 Red Hat OpenShift Service on AWS 中已弃用或删除的主要功能的最新列表，请参阅 Red Hat OpenShift Service on AWS *发行注记中已弃用和删除的功能* 部分。

在 Service Mesh 中自动管理网关的 OpenShift 路由。每次在 service mesh 中创建、更新或删除 Istio 网关时，都会自动创建、更新或删除 OpenShift 路由。

注意

从 Service Mesh 2.5 开始，对于 ServiceMeshControlPlane 资源的新实例，默认禁用自动路由。

1.14.2.1. 使用子域的路由

Red Hat OpenShift Service Mesh 使用子域创建路由，但必须配置 Red Hat OpenShift Service on AWS 才能启用它。支持子域，如 *.domain.com，但默认情况下不受支持。在配置通配符主机网关前，配置 Red Hat OpenShift Service on AWS 通配符策略。

1.14.2.2. 创建子域路由

以下示例在 Bookinfo 示例应用程序中创建了一个网关，它会创建子域路由。

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

Gateway 资源创建以下 OpenShift 路由：您可以使用以下命令来检查是否创建了路由。在本例中，istio-system 是 Service Mesh control plane 项目的名称。

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

预期输出

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

如果删除了网关，Red Hat OpenShift Service Mesh 会删除路由。但是，您手动创建的路由不会被 Red Hat OpenShift Service Mesh 修改。

1.14.2.3. 路由标签和注解

有时，OpenShift 路由中需要特定的标签或注解。

对于这个和其他用例，Red Hat OpenShift Service Mesh 会将 Istio 网关资源中的所有标签和注解

（除 kubectl.kubernetes.io开始的注解外）复制到受管 OpenShift Route 资源中。

如果您在 Service Mesh 创建的 OpenShift Routes 中需要特定的标签或注解，在 Istio 网关资源中创建它们，并将其复制到由 Service Mesh 管理的 OpenShift Route 资源中。

1.14.2.4. 禁用自动路由创建

默认情况下，ServiceMeshControlPlane 资源会自动将 Istio 网关资源与 OpenShift 路由同步。禁用自动路由创建功能，如果您有特殊情况或更喜欢手动控制路由，您可以更灵活地控制路由。

1.14.2.4.1. 为特定情况禁用自动路由创建

如果要禁用为特定 Istio 网关的 OpenShift 路由自动管理，您必须将注解 maistra.io/manageRoute: false 添加到网关元数据定义中。Red Hat OpenShift Service Mesh 将忽略带有这个注解的 Istio 网关，同时保持其它 Istio 网关的自动管理。

1.14.2.4.2. 为所有情况禁用自动路由创建

您可以为网格中的所有网关禁用自动管理 OpenShift 路由。

通过将 ServiceMeshControlPlane 字段 gateways.openshiftRoute.enabled 设置为 false 来禁用 Istio 网关和 OpenShift 路由之间的集成。例如，查看以下资源片断。

```yaml
apiVersion: maistra.io/v1alpha1
kind: ServiceMeshControlPlane
metadata:
  namespace: istio-system
spec:
  gateways:
    openshiftRoute:
      enabled: false
```

1.14.3. 了解服务条目

服务条目在由 Red Hat OpenShift Service Mesh 内部维护的服务 registry 中添加一个条目。添加服务条目后，Envoy 代理将流量发送到该服务，就像是网格中的服务一样。服务条目允许您进行以下操作：

- 管理服务网格外运行的服务的流量。
- 重定向和转发外部目的地的流量，如来自 web 的 API 调用，或转发到旧基础架构中服务的流量。
- 为外部目的地定义重新尝试、超时和错误注入策略。
- 在虚拟机 (VM) 中运行网格服务，方法是在网格中添加虚拟机。

> **注意**
>
> 将服务从不同集群添加到网格，以便在 Kubernetes 上配置多集群 Red Hat OpenShift Service Mesh 网格。

服务条目示例

以下示例 mesh-external 服务条目将 ext-resource 外部依赖关系添加到 Red Hat OpenShift Service Mesh 服务 registry:

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: svc-entry
spec:
  hosts:
  - ext-svc.example.com
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
```

使用 hosts 字段指定外部资源。您可以完全限定名，也可以使用通配符前缀域名。

您可以配置虚拟服务和目的地规则，以控制到服务条目的流量，其方式与您为网格中的任何其他服务配置流量相同。例如，以下目的地规则将流量路由配置为使用 mutual TLS 来保护到 ext-svc.example.com 外部服务的连接。它被配置为使用服务项：

```
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
```

1.14.4. 使用 VirtualServices

您可以使用虚拟服务通过 Red Hat OpenShift Service Mesh 将请求动态路由到微服务的不同版本。使用虚拟服务，您可以：

- 通过单一虚拟服务处理多个应用程序服务。如果网格使用 Kubernetes，您可以配置虚拟服务来处理特定命名空间中的所有服务。虚拟服务使您能够将单体式应用转变为由具有无缝消费者体验的不同微服务组成的服务。

- 配置流量规则与网关相结合，以控制入口和出口流量。

1.14.4.1. 配置 VirtualServices

使用虚拟服务将请求路由到服务网格中的服务。每个虚拟服务由一组路由规则组成，并按顺序应用。Red Hat OpenShift Service Mesh 会将每个给定给虚拟服务的请求与网格内的特定实际目的地匹配。

如果没有虚拟服务，Red Hat OpenShift Service Mesh 会在所有服务实例间使用最少请求（least requests）负载均衡分配流量。使用虚拟服务时，您可以指定一个或多个主机名的流量行为。虚拟服务的路由规则告知 Red Hat OpenShift Service Mesh 如何将虚拟服务的流量发送到适当的目的地。路由目的地可以是同一服务版本，也可以是完全不同的服务。

流程

1. 使用以下示例创建 YAML 文件，根据用户连接到应用程序的不同版本将请求路由到 Bookinfo 示例应用程序服务的不同版本。

VirtualService.yaml 示例

```
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. 运行以下命令以应用 VirtualService.yaml，其中 VirtualService.yaml 是文件的路径。

```
$ oc apply -f <VirtualService.yaml>
```

1.14.4.2. VirtualService 配置参考

参数	描述
`spec:` `  hosts:`	**hosts** 字段列出了路由规则应用到的虚拟服务的目标地址。这是用于向服务发送请求的地址。虚拟服务主机名可以是解析为完全限定域名的 IP 地址、DNS 名称或简短名称。

参数	描述

`spec:`
`  http:`
`  - match:`
	http 部分包含虚拟服务的路由规则，这些规则描述路由 HTTP/1.1、HTTP2 和 gRPC 流量与 hosts 字段中指定的目的地的匹配条件和操作。路由规则由您希望流量到达的目的地以及任何指定的匹配条件组成。示例中的第一个路由规则有一个以 match 字段开头的条件。在这个示例中，这个路由适用于来自用户 **jason** 的所有请求。添加 **headers**、**end-user** 和 **exact** 项来选择适当的请求。

`spec:`
`  http:`
`  - match:`
`    - destination:`
	route 部分的 **destination** 字段指定与这个条件匹配的流量的实际目的地。与虚拟服务的主机不同，目的地的主机必须是 Red Hat OpenShift Service Mesh 服务 registry 中存在的真实目的地。这可以是带有代理的网格服务，或使用 service 条目添加的一个非网格服务。在本例中，主机名是一个 Kubernetes 服务名称：

1.14.5. 了解目的地规则

目的地规则在评估虚拟服务路由规则后应用，它们应用到流量的真实目的地。虚拟服务将流量路由到目的地。目的地规则配置该目的地的网络流量。

默认情况下，Red Hat OpenShift Service Mesh 使用最的请求负载均衡策略，其中池中的服务实例最少活跃连接数接收请求。Red Hat OpenShift Service Mesh 还支持以下模型，您可以在目的地规则中指定对特定服务或服务子集的请求。

- Random： 请求会随机转发到池里的实例。
- Weighted： 根据特定百分比将请求转发到池中的实例。
- Least requests： 将请求转发到请求数量最少的实例。

目的地规则示例

以下目的地规则示例为 my-svc 目的地服务配置三个不同的子集，具有不同的负载均衡策略：

```
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
```

1.14.6. 了解网络策略

Red Hat OpenShift Service Mesh 自动在 Service Mesh control plane 和应用程序命名空间中创建和管理多个 NetworkPolicies 资源。这是为了确保应用程序和 control plane 可以相互通信。

例如，如果您已将 Red Hat OpenShift Service on AWS 集群配置为使用 SDN 插件，Red Hat OpenShift Service Mesh 会在每个成员项目中创建 NetworkPolicy 资源。这可让从其他网格成员和 control plane 对网格中的所有 pod 的入站网络数据。这也限制了到成员项目的入站网络数据。如果需要来自非成员项目的入站网络数据，则需要创建一个 NetworkPolicy 来允许这些流量通过。如果您从 Service Mesh 中删除命名空间，则此 NetworkPolicy 资源会从项目中删除。

1.14.6.1. 禁用自动 NetworkPolicy 创建

如果要禁用 NetworkPolicy 资源自动创建和管理，例如强制实现公司安全策略，或者允许直接访问网格中的 pod，您可以这样做。您可以编辑 ServiceMeshControlPlane，并将 spec.security.manageNetworkPolicy 设置为 false。

注意

当您禁用了 spec.security.manageNetworkPolicy，Red Hat OpenShift Service Mesh 不会创建 任何 NetworkPolicy 对象。系统管理员负责管理网络并修复可能导致的任何问题。

前提条件

- 安装了 Red Hat OpenShift Service Mesh Operator 2.1.1 或更高版本。

- ServiceMeshControlPlane 资源更新至 2.1 或更高版本。

流程

1. 在 Red Hat OpenShift Service on AWS web 控制台中，点 Operators → Installed Operators。

2. 从 Project 菜单中选择安装 Service Mesh control plane 的项目，如 istio-system。

3. 点 Red Hat OpenShift Service Mesh Operator。在 Istio Service Mesh Control Plane 栏中，点 ServiceMeshControlPlane 的名称，如 basic-install。

4. 在 Create ServiceMeshControlPlane Details 页中，点 YAML 修改您的配置。

5. 将 ServiceMeshControlPlane 字段 spec.security.manageNetworkPolicy 设置为 false，如下例所示。

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  security:
    manageNetworkPolicy: false
```

6. 点 Save。

1.14.7. 为流量管理配置 sidecar

默认情况下，Red Hat OpenShift Service Mesh 配置每个 Envoy 代理，在其相关负载的所有端口上接收流量，并在转发流量时到达网格中的每个工作负载。您可以使用 sidecar 配置进行以下操作：

- 微调 Envoy 代理接受的端口和协议集合。

- 限制 Envoy 代理可访问的服务集合。

> **注意**
>
> 要优化服务网格的性能，请考虑限制 Envoy 代理配置。

在 Bookinfo 示例应用程序中，配置 Sidecar 以便所有服务都可以访问在同一命名空间和 control plane 中运行的其他服务。使用 Red Hat OpenShift Service Mesh 策略和遥测功能需要这个 Sidecar 配置。

流程

1. 使用以下示例创建 YAML 文件，以指定您希望 sidecar 配置应用到特定命名空间中的所有工作负载。否则，使用 workloadSelector 选择特定的工作负载。

   sidecar.yaml 示例

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

2. 运行以下命令以应用 sidecar.yaml，其中 sidecar.yaml 是文件的路径。

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

3. 运行以下命令，以验证 sidecar 是否已成功创建。

   -

```
$ oc get sidecar
```

1.14.8. 路由教程

本指南使用 Bookinfo 示例应用程序来提供示例应用程序中的路由示例。安装 Bookinfo 应用程序以了解这些路由示例如何工作。

1.14.8.1. Bookinfo 路由指南

Service Mesh Bookinfo 示例应用程序包含四个独立的微服务，每个服务都有多个版本。安装 Bookinfo 示例应用程序后，reviews 微服务的三个不同版本同时运行。

当您在浏览器中访问 Bookinfo 应用 /product 页面并多次刷新时，有时书的评论输出中会包含星号分级，而其它时候则没有。如果没有可路由的显式默认服务版本，Service Mesh 会将请求路由到所有可用版本。

本教程可帮助您应用将所有流量路由到微服务的 v1 （版本 1）的规则。之后，您可以根据 HTTP 请求标头值应用一条规则来路由流量。

先决条件

- 部署 Bookinfo 示例应用程序以使用以下示例。

1.14.8.2. 应用虚拟服务

在以下流程中，虚拟服务通过应用为微服务设定默认版本的虚拟服务，将所有流量路由到每个微服务的 v1。

流程

1. 应用虚拟服务。

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

2. 要验证是否应用了虚拟服务，请使用以下命令显示定义的路由：

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

该命令返回一个 kind: VirtualService 资源，采用 YAML 格式。

您已将 Service Mesh 配置为路由到 Bookinfo 微服务的 v1 版本，包括 reviews 服务版本 1。

1.14.8.3. 测试新路由配置

通过刷新 Bookinfo 应用程序的 /productpage 来测试新配置。

流程

1. 设置 GATEWAY_URL 参数的值。您可以在以后使用这个变量查找 Bookinfo 产品页面的 URL。在本例中，istio-system 是 control plane 项目的名称。

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

2. 运行以下命令，以检索产品页面的 URL:

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

3. 在浏览器中打开 Bookinfo 网站。

页面的评论部分显示没有分级星，无论您刷新多少次。这是因为您已将 Service Mesh 配置为将 reviews 服务的所有流量路由到版本 review:v1，此版本的服务无法访问星表分级服务。

您的服务网格现在将流量路由到服务的一个版本。

1.14.8.4. 基于用户身份的路由

更改路由配置，以便特定用户的所有流量都路由到特定的服务版本。在这种情况下，所有来自名为 Jason 的用户的流量都会被路由到服务的 review:v2 中。

Service Mesh 对用户身份没有任何特殊的内置了解。这个示例是启用的，因为 productpage 服务为到 reviews 服务的所有传出 HTTP 请求都添加了一个自定义的 end-user 标头。

流程

1. 运行以下命令在 Bookinfo 示例应用程序中启用基于用户的路由。

```
$ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.6/samples/info/networking/virtual-service-reviews-test-v2.yaml
```

2. 运行以下命令，以确认创建了该规则。此命令返回 YAML 格式的所有 kind: VirtualService。

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

3. 在 Bookinfo 应用程序的 /productpage 中，以用户 jason 身份在无需密码的情况下进行登录。

4. 刷新浏览器。星级分级会出现在每条评论旁。

5. 以其他用户身份登录（选择任意名称）。刷新浏览器。现在就不会出现星级评分。现在，除 Jason 外，所有用户的流量都会被路由到 review :v1。

您已成功配置了 Bookinfo 示例应用程序，以根据用户身份路由流量。

1.15. 指标、日志和追踪

将应用程序添加到网格后，您可以观察通过应用程序的数据流。如果您没有安装自己的应用程序，可以通过安装 Bookinfo 示例应用程序来了解 Red Hat OpenShift Service Mesh 中的可观察性如何工作。

1.15.1. 发现控制台地址

Red Hat OpenShift Service Mesh 提供以下控制台来查看您的服务网格数据：

- Kiali 控制台 - Kiali 是 Red Hat OpenShift Service Mesh 的管理控制台。
- Jaeger console - Jaeger 是 Red Hat OpenShift distributed Tracing Platform 的管理控制台。
- Grafana 控制台 - Grafana 为网格管理员提供 Istio 数据的高级查询和指标分析和仪表板。另外，Grafana 可以用来分析服务网格指标。
- Prometheus 控制台 - Red Hat OpenShift Service Mesh 使用 Prometheus 存储来自服务的遥测信息。

安装 Service Mesh control plane 时，它会为每个安装的组件自动生成路由。获得路由地址后，您可以访问 Kiali、Jaeger、Prometheus 或 Grafana 控制台来查看和管理您的服务网格数据。

前提条件

- 必须启用并安装组件。例如，如果您没有安装分布式追踪，您将无法访问 Jaeger 控制台。

从 OpenShift 控制台的步骤

1. 以具有 cluster-admin 权限的用户身份登录Red Hat OpenShift Service on AWS Web。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 dedicated-admin 角色的帐户。
2. 进入 Networking → Routes。
3. 在 Routes 页面中，从 Namespace 菜单中选择 Service Mesh control plane 项目，如 istio-system。

   Location 列显示每个路由的链接地址。

4. 如有必要，使用过滤器来查找您要访问的路由的组件控制台。单击路由 位置 以启动控制台。

5. 单击 Log In With OpenShift。

通过 CLI 操作的步骤

1. 以具有 cluster-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS CLI。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 dedicated-admin 角色的帐户。

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. 切换到 Service Mesh control plane 项目。在本例中，istio-system 是 Service Mesh control plane 项目。运行以下命令：

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

3. 要获取各种 Red Hat OpenShift Service Mesh 控制台的路由，请运行以下命令：

```
$ oc get routes
```

这个命令返回 Kiali、Jaeger、Prometheus 和 Grafana web 控制台以及服务网格中任何其他路由的 URL。您应该看到类似如下的输出：

```
NAME                    HOST/PORT                         SERVICES              PORT    TERMINATION
info-gateway            bookinfo-gateway-yourcompany.com  istio-ingressgateway
http2
grafana                 grafana-yourcompany.com           grafana               <all>
reencrypt/Redirect
istio-ingressgateway    istio-ingress-yourcompany.com     istio-ingressgateway  8080
jaeger                  jaeger-yourcompany.com            jaeger-query          <all>   reencrypt
kiali                   kiali-yourcompany.com             kiali                 20001   reencrypt/Redirect
prometheus              prometheus-yourcompany.com        prometheus            <all>
reencrypt/Redirect
```

4. 将您要从 HOST/PORT 列访问的控制台的 URL 复制到浏览器中以打开控制台。

5. 单击 **Log In With OpenShift**。

1.15.2. 访问 Kiali 控制台

您可以在 Kiali 控制台中查看应用程序的拓扑、健康和指标。如果您的服务遇到问题，Kiali 控制台可让您通过服务查看数据流。您可以查看不同级别中的与网格组件相关的信息，包括抽象应用程序、服务以及负载。Kiali 还会实时提供命名空间的互动图形视图。

要访问 Kiali 控制台，您必须安装并配置了 Red Hat OpenShift Service Mesh。

安装过程创建了访问 Kiali 控制台的路由。

如果您知道 Kiali 控制台的 URL，您可以直接访问它。如果您不知道 URL，请使用以下指示：

管理员的步骤

1. 使用管理员角色登录到 Red Hat OpenShift Service on AWS Web 控制台。
2. 点 **Home** → **Projects**。
3. 如有必要，在 **Projects** 页面上，使用过滤器来查找项目的名称。
4. 点项目的名称，例如 **info**。
5. 在 **Project details** 页面中，点 **Launcher** 部分的 **Kiali** 链接。
6. 使用与访问 Red Hat OpenShift Service on AWS 控制台相同的用户名和密码登录到 Kiali 控制台。

   第一次登录到 Kiali 控制台时，您会看到 **Overview** 页面，它会显示服务网格中您有权查看的所有命名空间。

如果您验证了控制台安装，且命名空间还没有添加到网格中，则可能无法显示 istio-system 以外的任何数据。

开发人员的步骤

1. 使用开发者角色登录 Red Hat OpenShift Service on AWS Web 控制台。

2. 单击 Project。

3. 如有必要，在 Project Details 页面上，使用过滤器来查找项目的名称。

4. 点项目的名称，例如 info。

5. 在 Project 页面中，点 Launcher 部分的 Kiali 链接。

6. 单击 Log In With OpenShift。

1.15.3. 在 Kiali 控制台中查看服务网格数据

Kiali Graph 为您的网格流量提供了强大的视觉化功能。拓扑将实时请求流量与您的 Istio 配置信息相结合，可让您快速发现服务网格的行为。多种图形类型允许您将流量视觉化为高级别服务拓扑、低级工作负载拓扑或应用程序级别拓扑。

可以选择的几个图：

- App 图显示所有标记相同应用程序的总工作负载。

- Service 图显示网格中各个服务的节点，但所有应用程序和工作负载都不包括在这个图中。它提供了一个高级别的视图，并聚合了定义的服务的所有流量。

- Versioned App 图 显示每个应用版本的节点。应用程序的所有版本都分组在一起。

- Workload 图显示服务网格中每个工作负载的节点。此图不要求您使用应用程序和版本标签。如果您的应用程序没有使用 version 标签，请使用此图。

图形节点使用各种信息进行解码，指向虚拟服务和服务条目等各种路由路由选项，以及故障注入和断路器等特殊配置。它可以识别 mTLS 问题、延迟问题、错误流量等。Graph 高度可配置，可以显示流量动画，并具有强大的 Find 和 Hide 功能。

单击 Legend 按钮，以查看图中显示的有关图形、颜色、箭头和徽标的信息。

要查看指标的概述信息，请在图形中选择任意节点或边缘以便在概述详情面板中显示其指标详情。

1.15.3.1. 在 Kiali 中更改图形布局

Kiali 图形的布局可能会根据您的应用程序架构和要显示的数据的不同而有所不同。例如，图形节点的数量及其交互可以决定 Kiali 图形的呈现方式。因为无法创建出适合每种情况的单一布局，Kiali 提供了几种不同布局的选择。

先决条件

- 如果您没有安装自己的应用程序，请安装 Bookinfo 示例应用程序。然后，通过多次输入以下命令为 Bookinfo 应用程序生成流量。

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

  此命令模拟访问应用的 productpage 微服务的用户。

流程

1. 启动 Kiali 控制台。

2. 单击 Log In With OpenShift。

3. 在 Kiali 控制台中，点 Graph 查看命名空间图。

4. 在 Namespace 菜单中选择应用程序命名空间，例如 info。

5. 要选择不同的图形布局，请执行以下任一操作：

   - 从图顶部的菜单中选择不同的图形数据分组。

     - 应用程序图

     - 服务图

     - 版本化应用图（默认）

     - 工作负载图

   - 从图形底部的图标中选择不同的图形布局。

     - 布局默认 dagre

     - 布局 1 cose-bilkent

     - 布局 2 cola

1.15.3.2. 在 Kiali 控制台中查看日志

您可以在 Kiali 控制台中查看工作负载的日志。Workload Detail 页面包含一个 Logs 选项卡，显示一个可显示应用程序和代理日志的统一日志视图。您可以选择在 Kiali 中显示日志的频率。

要更改 Kiali 中显示的日志的日志级别，您可以更改工作负载或代理的日志配置。

前提条件

- 安装和配置 Service Mesh。
- 已安装并配置了 Kiali。
- Kiali 控制台的地址。
- 在网格中添加了应用程序或 Bookinfo 示例应用程序。

流程

1. 启动 Kiali 控制台。
2. 单击 **Log In With OpenShift**。

   Kiali Overview 页面会显示添加到具有权限的网格中的命名空间。
3. 单击 **Workloads**。
4. 在 **Workloads** 页面中，从 **Namespace** 菜单中选择项目。
5. 如有必要，使用过滤器来查找您要查看的日志的工作负载。单击工作负载名称。例如，单击 **ratings-v1**。
6. 在 **Workload Details** 页面中，单击 **Logs** 选项卡来查看工作负载的日志。

提示

如果没有看到任何日志条目，您可能需要调整 Time Range 或 Refresh 间隔。

1.15.3.3. 在 Kiali 控制台中查看指标

您可以在 Kiali 控制台中查看应用程序、工作负载和服务的入站和出站指标。详情页面包括以下标签页：

- 入站应用程序指标
- 出站应用指标
- 入站工作负载指标
- 出站工作负载指标
- 入站服务指标

这些标签页显示预定义的指标仪表板，它们根据相关应用程序、工作负载或服务级别进行定制。应用程序和工作负载详情视图显示请求和响应指标，如卷、持续时间、大小或 TCP 流量。服务详情视图仅显示入站流量的请求和响应指标。

Kiali 允许您选择图表的尺寸来自定义 chart。Kiali 还可以显示源或目标代理指标报告的指标。另外，Kiali 可以覆盖指标上的 trace。

前提条件

- 安装和配置 Service Mesh。
- 已安装并配置了 Kiali。

- Kiali 控制台的地址。
- （可选）安装和配置了分布式追踪。

流程

1. 启动 Kiali 控制台。
2. 单击 Log In With OpenShift。

   Kiali Overview 页面会显示添加到具有权限的网格中的命名空间。
3. 单击 Applications、Workloads 或 Services。
4. 在 Applications、Workloads 或 Services 页面上，从 Namespace 菜单中选择项目。
5. 如有必要，使用过滤器来查找您要查看其日志的应用程序、工作负载或服务。单击 Name。
6. 在 Application Detailstail、Workload Details 或 Service Details 页面中，单击 Inbound Metrics 或 Outbound Metrics 选项卡来查看指标。

1.15.4. 分布式追踪

分布式追踪是通过跟踪应用中服务调用的路径来跟踪应用中各个服务的性能的过程。每次用户在应用中采取行动时，将执行请求，该请求可能需要许多服务进行交互来生成响应。此请求的路径称为分布式事务。

Red Hat OpenShift Service Mesh 使用 Red Hat OpenShift distributed Tracing Platform 来查看微服务应用程序中的调用流。

1.15.4.1. 配置 Red Hat OpenShift distributedracing Platform 和 Red Hat build of OpenTelemetry

您可以通过将命名元素和 opentelemetry 供应商附加到 ServiceMeshControlPlane 中的 spec.meshConfig.extensionProviders 规格中，向 Red Hat OpenShift distributed Tracing Platform 公开追踪数据。然后，遥测自定义资源将 Istio 代理配置为收集 trace span，并将其发送到 OpenTelemetry Collector 端点。

您可以在网格命名空间中创建红帽构建的 OpenTelemetry 实例，并将其配置为将追踪数据发送到追踪平台后端服务。

先决条件

- 您在 tracing-system 命名空间中使用 Red Hat Tempo Operator 创建 TempoStack 实例。如需更多信息，请参阅"安装 Red Hat OpenShift distributed Tracing Platform"。

- 您在推荐的命名空间或 openshift-operators 命名空间中安装了 Red Hat build of OpenTelemetry Operator。如需更多信息，请参阅"安装Red Hat build of OpenTelemetry"。

- 如果使用 Red Hat OpenShift Service Mesh 2.5 或更早版本，请将 ServiceMeshControlPlane 资源的 spec.tracing.type 参数设置为 None，以便追踪数据可以发送到 OpenTelemetry Collector。

流程

1. 在网格命名空间中创建 OpenTelemetry Collector 实例。这个示例使用 info 命名空间：

   OpenTelemetry Collector 配置示例

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otel
  namespace: info (1)
spec:
  mode: deployment
  config: |
    receivers:
      otlp:
        protocols:
          grpc:
            endpoint: 0.0.0.0:4317
```

```yaml
  exporters:
    otlp:
      endpoint: "tempo-sample-distributor.tracing-system.svc.cluster.local:4317" 2
      tls:
        insecure: true
  service:
    pipelines:
      traces:
        receivers: [otlp]
        processors: []
        exporters: [otlp]
```

1 在 ServiceMeshMemberRoll 成员列表中包含命名空间。

2 在本例中，TempoStack 实例在 tracing-system 命名空间中运行。您不必在 ServiceMeshMemberRoll 成员列表中包含 TempoStack 命名空间，如 'tracing-system'。

注意

- 在其中一个 ServiceMeshMemberRoll 成员命名空间中创建 OpenTelemetry Collector 的单一实例。
- 您可以通过在 OpenTelemetryCollector 资源中添加 sidecar.istio.io/inject: 'true' 来添加 otel-collector 作为网格的一部分。

2. 检查 otel-collector pod 日志并验证 pod 是否正在运行：

otel-collector pod 日志检查示例

```
$ oc logs -n info  -l app.kubernetes.io/name=otel-collector
```

3. 在 istio-system 命名空间中创建或更新现有 ServiceMeshControlPlane 自定义资源(CR)：

SMCP 自定义资源示例

```
kind: ServiceMeshControlPlane
apiVersion: maistra.io/v2
metadata:
  name: basic
  namespace: istio-system
spec:
  addons:
    grafana:
      enabled: false
    kiali:
      enabled: true
    prometheus:
      enabled: true
  meshConfig:
    extensionProviders:
      - name: otel
        opentelemetry:
          port: 4317
          service: otel-collector.info.svc.cluster.local
  policy:
    type: Istiod
  telemetry:
    type: Istiod
  version: v2.6
```

注意

当从 SMCP 2.5 升级到 2.6 时，将 spec.tracing.type 参数设置为 None ：

SMCP spec.tracing.type 参数示例

```
spec:
  tracing:
    type: None
```

4. 在 istio-system 命名空间中创建 Telemetry 资源：

Telemetry 资源示例

```
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: mesh-default
  namespace: istio-system
spec:
  tracing:
  - providers:
    - name: otel
    randomSamplingPercentage: 100
```

5. 验证 istiod 日志。

6. 配置 Kiali 资源规格以启用 Kiali 工作负载跟踪仪表板。您可以使用仪表板查看追踪查询结果。

Kiali 资源示例

```yaml
apiVersion: kiali.io/v1alpha1
kind: Kiali
# ...
spec:
  external_services:
    tracing:
      query_timeout: 30 (1)
      enabled: true
      in_cluster_url: 'http://tempo-sample-query-frontend.tracing-system.svc.cluster.local:16685'
      url: '[Tempo query frontend Route url]'
      use_grpc: true (2)
```

1. 默认的 query_timeout 整数值为 30 秒。如果将值设为 30 秒，您必须更新 Kiali CR 中的 .spec.server.write_timeout，并将注解 haproxy.router.openshift.io/timeout=50s 添加到 Kiali 路由。.spec.server.write_timeout 和 haproxy.router.openshift.io/timeout= 必须大于 query_timeout。

2. 如果您没有使用默认的 HTTP 或 gRPC 端口，请将 in_cluster_url: 端口替换为您的自定义端口。

注意

Kiali 1.73 使用 Jaeger Query API，这会导致根据 Tempo 资源限制造成较长的响应时间。如果您在 Kiali UI 中看到 Could not fetch span 错误消息，请检查 Tempo 配置，或减少 Kiali 中每个查询的限制。

7. 将请求发送到您的应用。

8. 验证 istiod pod 日志和 otel-collector pod 日志。

1.15.4.1.1. 在 mTLS 加密 Service Mesh 成员命名空间中配置 OpenTelemetryCollector

当您启用 Service Mesh dataPlane mTLS 加密时，所有流量都是 TLS 加密。

要启用网格与 OpenTelemetryCollector 服务通信，请通过为 OpenTelemetryCollector 服务应用 DestinationRule 来禁用 TLS trafficPolicy ：

DestinationRule Tempo CR 示例

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: otel-disable-tls
spec:
  host: "otel-collector.info.svc.cluster.local"
  trafficPolicy:
    tls:
      mode: DISABLE
```

1.15.4.1.2. 在 mTLS 加密 Service Mesh 成员命名空间中配置 Red Hat OpenShift distributed Tracing Platform

注意

如果您在不是 Service Mesh 成员命名空间的命名空间中创建了 TempoStack 实例，则不需要这个额外的 DestinationRule 配置。

当您启用 Service Mesh dataPlane mTLS 加密并在 Service Mesh 成员命名空间中创建 TempoStack 实例时，所有流量都是 TLS 加密，如 tracing-system-mtls。这个加密没有预期来自 Tempo 分布式服务，并返回一个 TLS 错误。

要修复 TLS 错误，请通过为 Tempo 和 Kiali 应用 DestinationRule 来禁用 TLS trafficPolicy ：

DestinationRule Tempo 示例

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: tempo
  namespace: tracing-system-mtls
spec:
  host: "*.tracing-system-mtls.svc.cluster.local"
  trafficPolicy:
    tls:
      mode: DISABLE
```

DestinationRule Kiali 示例

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: kiali
  namespace: istio-system
spec:
  host: kiali.istio-system.svc.cluster.local
  trafficPolicy:
    tls:
      mode: DISABLE
```

1.15.4.2. 连接现有的分布式追踪 Jaeger 实例

如果您已在 Red Hat OpenShift Service on AWS 中已有现有的 Red Hat OpenShift distributed Tracing Platform (Jaeger)实例，您可以将 ServiceMeshControlPlane 资源配置为使用该实例进行分布式跟踪平台。

> **重要**
>
> 从 Red Hat OpenShift Service Mesh 2.5 开始，Red Hat OpenShift distributed Tracing Platform (Jaeger)和 OpenShift Elasticsearch Operator 已被弃用，并将在以后的发行版本中删除。红帽将在当前发行生命周期中为这些功能提供程序错误修正和支持，但此功能将不再获得改进，并将被删除。作为 Red Hat OpenShift distributed Tracing Platform (Jaeger)的替代选择，您可以使用 Red Hat OpenShift distributed Tracing Platform。

先决条件

- 安装和配置 Red Hat OpenShift distributed Tracing Platform 实例。

流程

1. 在 Red Hat OpenShift Service on AWS web 控制台中，点 Operators → Installed Operators。

2. 点 Project 菜单，选择安装 Service Mesh control plane 的项目，如 istio-system。

3. 点 Red Hat OpenShift Service Mesh Operator。在 Istio Service Mesh Control Plane 列中，点 ServiceMeshControlPlane 资源的名称，例如 basic。

4. 将分布式跟踪平台(Jaeger)实例的名称添加到 ServiceMeshControlPlane。

   a. 点 YAML 标签。

   b. 将分布式 Tracing Platform (Jaeger)实例的名称添加到 ServiceMeshControlPlane 资源中的 spec.addons.jaeger.name 中。在以下示例中，distr-tracing-production 是分布式 Tracing Platform (Jaeger)实例的名称。

   分布式追踪配置示例

   ```
   spec:
     addons:
       jaeger:
         name: distr-tracing-production
   ```

   c. 点 Save。

5. 点 Reload 来验证 ServiceMeshControlPlane 资源已被正确配置。

1.15.4.3. 调整抽样率

trace 是服务网格中服务间的执行路径。一个 trace 由一个或多个范围组成。span 是具有名称、开始时间和持续时间的逻辑工作单元。抽样率决定了 trace 的持久性频率。

Envoy 代理抽样率默认设置为服务网格中 trace 的 100%。高抽样率会消耗集群资源和性能，但在调试问题时很有用。在生产环境中部署 Red Hat OpenShift Service Mesh 前，请将值设置为较小的 trace 部分。例如，将 spec.tracing.sampling 设置为 100 来示例 trace 的 1%。

将 Envoy 代理抽样率配置为代表 0.01% 增量的扩展整数。

在基本安装中，spec.tracing.sampling 设置为 10000，这代表 100% 的 trace 采样。例如：

- 将值设置为 10 个 trace 的 0.1% 样本。
- 将值设为 500 个样本 5% 的 trace.

注意

Envoy 代理抽样率适用于 Service Mesh 可用的应用程序，并使用 Envoy 代理。这个抽样率决定了 Envoy 代理收集并跟踪的数据量。

Jaeger 远程抽样率适用于 Service Mesh 外部的应用程序，不要使用 Envoy 代理，如数据库。这种抽样率决定了分布式追踪系统收集和存储的数据量。

流程

1. 在 Red Hat OpenShift Service on AWS web 控制台中，点 Operators → Installed Operators。

2. 点 Project 菜单并选择安装 control plane 的项目，如 istio-system。

3. 点 Red Hat OpenShift Service Mesh Operator。在 Istio Service Mesh Control Plane 列中，点 ServiceMeshControlPlane 资源的名称，例如 basic。

4. 要调整抽样率，请为 spec.tracing.sampling 设置不同的值。

   a. 点 YAML 标签。

   b. 为 ServiceMeshControlPlane 资源中的 spec.tracing.sampling 设置值。在以下示例中，将它设置为 100。

      Jaeger 抽样示例

```
spec:
  tracing:
    sampling: 100
```

   c. 点 Save。

5. 点 Reload 来验证 ServiceMeshControlPlane 资源已被正确配置。

1.15.5. 访问 Jaeger 控制台

要访问 Jaeger 控制台，您必须安装并配置了 Red Hat OpenShift Service Mesh。

安装过程会创建路由来访问 Jaeger 控制台。

如果您知道 Jaeger 控制台的 URL，您可以直接访问它。如果您不知道 URL，请使用以下指示：

重要

从 Red Hat OpenShift Service Mesh 2.5 开始，Red Hat OpenShift distributed Tracing Platform (Jaeger)和 OpenShift Elasticsearch Operator 已被弃用，并将在以后的发行版本中删除。红帽将在当前发行生命周期中提供对这个功能的程序漏洞修复和支持，但这个功能将不再获得改进，并将被删除。作为 Red Hat OpenShift distributed Tracing Platform (Jaeger)的替代选择，您可以使用 Red Hat OpenShift distributed Tracing Platform。

从 OpenShift 控制台的步骤

1. 以具有 cluster-admin 权限的用户身份登录Red Hat OpenShift Service on AWS Web。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 dedicated-admin 角色的帐户。

2. 进入 Networking → Routes。

3. 在 Routes 页面中，从 Namespace 菜单中选择 Service Mesh control plane 项目，如 istio-system。

   Location 列显示每个路由的链接地址。

4. 如有必要，使用过滤器来查找 jaeger 路由。单击路由 位置 以启动控制台。

5. 单击 Log In With OpenShift。

Kiali 控制台的步骤

1. 启动 Kiali 控制台。

2. 单击左侧导航窗格中的 Distributed Tracing。

3.

3. 单击 **Log In With OpenShift**。

通过 CLI 操作的步骤

1. 以具有 cluster-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS CLI。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 dedicated-admin 角色的帐户。

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. 要使用命令行查询路由详情，请输入以下命令。在本例中，istio-system 是 Service Mesh control plane 命名空间。

```
$ oc get route -n istio-system jaeger -o jsonpath='{.spec.host}'
```

3. 启动浏览器并进入 https://<JAEGER_URL>，其中 <JAEGER_URL> 是您在上一步中发现的路由。

4. 使用您用来访问 Red Hat OpenShift Service on AWS 控制台的同一用户名和密码登录。

5. 如果您已将服务添加到服务网格中并生成了 trace，您可以使用过滤器和 **Find Traces** 按钮搜索 trace 数据。

   如果您要验证控制台安装，则不会显示 trace 数据。

1.15.6. 访问 Grafana 控制台

Grafana 是一个分析工具，可用于查看、查询和分析服务网格指标。在本例中，istio-system 是 Service Mesh control plane 命名空间。要访问 Grafana，请执行以下操作：

流程

1. 登陆到 Red Hat OpenShift Service on AWS web 控制台。

2. 点 Project 菜单，选择安装 Service Mesh control plane 的项目，如 istio-system。

3. 单击 Routes。

4. 点击 Grafana 行的 Location 列中的链接。

5. 使用 Red Hat OpenShift Service on AWS 凭证登录到 Grafana 控制台。

1.15.7. 访问 Prometheus 控制台

Prometheus 是一个监控和警报工具，可用于收集微服务相关的多维数据。在本例中，istio-system 是 Service Mesh control plane 命名空间。

流程

1. 登陆到 Red Hat OpenShift Service on AWS web 控制台。

2. 点 Project 菜单，选择安装 Service Mesh control plane 的项目，如 istio-system。

3. 单击 Routes。

4. 单击 Prometheus 行的 Location 列中的链接。

5. 使用 Red Hat OpenShift Service on AWS 凭证登录到 Prometheus 控制台。

1.15.8. 与用户工作负载监控集成

默认情况下，Red Hat OpenShift Service Mesh (OSSM)使用专用的 Prometheus 实例安装 Service Mesh control plane (sMCP)，用于从网格收集指标。但是，生产系统需要更高级的监控系统，如 Red Hat OpenShift Service on AWS 监控用户定义的项目。

以下步骤演示了如何将 Service Mesh 与 user-workload 监控集成。

前提条件

- 启用 user-workload 监控。
- 安装了 Red Hat OpenShift Service Mesh Operator 2.4。
- 已安装 Kiali Operator 1.65。

流程

1. 将 cluster-monitoring-view 角色授予 Kiali 服务帐户：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kiali-monitoring-rbac
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-monitoring-view
subjects:
- kind: ServiceAccount
  name: kiali-service-account
  namespace: istio-system
```

2. 为 user-workload 监控配置 Kiali：

```
apiVersion: kiali.io/v1alpha1
kind: Kiali
metadata:
  name: kiali
  namespace: istio-system
spec:
  auth:
    strategy: openshift
  deployment:
    accessible_namespaces:   #restricted setting for ROSA
      - istio-system
```

```
image_pull_policy: ''
ingress_enabled: true
namespace: istio-system
```

- 如果使用 Istio Operator 2.4，请使用此配置为 user-workload 监控配置 Kiali：

```
apiVersion: kiali.io/v1alpha1
kind: Kiali
metadata:
  name: kiali-user-workload-monitoring
  namespace: istio-system
spec:
  external_services:
    istio:
      config_map_name: istio-<smcp-name>
      istio_sidecar_injector_config_map_name: istio-sidecar-injector-<smcp-name>
      istiod_deployment_name: istiod-<smcp-name>
      url_service_version: 'http://istiod-<smcp-name>.istio-system:15014/version'
    prometheus:
      auth:
        token: secret:thanos-querier-web-token:token
        type: bearer
        use_kiali_token: false
      query_scope:
        mesh_id: "basic-istio-system"
      thanos_proxy:
        enabled: true
      url: https://thanos-querier.openshift-monitoring.svc.cluster.local:9091
  version: v1.65
```

注意

Red Hat OpenShift Service on AWS 对创建资源并不允许在 Red Hat managed 命名空间中创建 Kiali 资源方面存在额外的限制。

这意味着 OpenShift 专用集群中不允许使用 spec.deployment.access_namespaces 的以下通用设置：

- ['**']（所有命名空间）
- default
- codeready-*
- openshift-*
- redhat-*

验证错误消息提供了所有受限命名空间的完整列表。

3. 为外部 Prometheus 配置 SMCP：

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  addons:
    prometheus:
      enabled: false 1
    grafana:
      enabled: false 2
    kiali:
      name: kiali-user-workload-monitoring
  meshConfig:
```

```
  extensionProviders:
  - name: prometheus
    prometheus: {}
```

1. 禁用 OSSM 提供的默认 Prometheus 实例。

2. 禁用 Grafana。外部 Prometheus 实例不支持它。

4. 应用自定义网络策略以允许来自监控命名空间的入口流量：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: user-workload-access
  namespace: istio-system (1)
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: monitoring
  podSelector: {}
  policyTypes:
  - Ingress
```

1. 自定义网络策略必须应用到所有命名空间。

5. 应用 Telemetry 对象以启用 Istio 代理中的流量指标：

```
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: enable-prometheus-metrics
  namespace: istio-system (1)
spec:
  selector: (2)
    matchLabels:
      app: info
```

```
metrics:
- providers:
  - name: prometheus
```

1
在 control plane 命名空间中创建的 Telemetry 对象应用到网格中的所有工作负载。要将遥测应用到一个命名空间，请在目标命名空间中创建对象。

2
可选：设置 selector.matchLabels spec，将 Telemetry 对象应用到目标命名空间中的特定工作负载。

6. 应用 ServiceMonitor 对象来监控 Istio control plane：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: istiod-monitor
  namespace: istio-system 1
spec:
  targetLabels:
  - app
  selector:
    matchLabels:
      istio: pilot
  endpoints:
  - port: http-monitoring
    interval: 30s
    relabelings:
    - action: replace
      replacement: "basic-istio-system" 2
      targetLabel: mesh_id
```

1
在 Istio control plane 命名空间中创建此 ServiceMonitor 对象，因为它监控 Istiod 服务。在本例中，命名空间是 istio-system。

2
字符串 "basic-istio-system" 是 SMCP 名称及其命名空间的组合，但只要使用集群中用户工作负载监控的每个网格都是唯一的，可以使用任何标签。第 2 步中配置的 Kiali 资源的 spec.prometheus.query_scope 需要匹配这个值。

注意

如果使用 user-workload 监控只有一个网格，则 Kiali 资源中的 mesh_id 重新标记和 spec.prometheus.query_scope 字段都是可选的（但如果 mesh_id 标签被删除，则此处给出的 query_scope 字段应该被删除）。

如果集群中的多个网格实例可能会使用 user-workload 监控，则需要 Kiali 资源中的 mesh_id relabelings 和 spec.prometheus.query_scope 字段。这样可确保 Kiali 只查看其关联的网格中的指标。

如果没有部署 Kiali，您仍然可以应用 mesh_id 重新标记，以便来自不同网格的指标可以区分不同的网格。

7. 应用 PodMonitor 对象从 Istio 代理收集指标：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: istio-proxies-monitor
  namespace: istio-system (1)
spec:
  selector:
    matchExpressions:
    - key: istio-prometheus-ignore
      operator: DoesNotExist
  podMetricsEndpoints:
  - path: /stats/prometheus
    interval: 30s
    relabelings:
    - action: keep
      sourceLabels: [__meta_kubernetes_pod_container_name]
      regex: "istio-proxy"
    - action: keep
      sourceLabels:
[__meta_kubernetes_pod_annotationpresent_prometheus_io_scrape]
    - action: replace
      regex: (\d+);(([A-Fa-f0-9]{1,4}::?){1,7}[A-Fa-f0-9]{1,4})
      replacement: '[$2]:$1'
      sourceLabels: [__meta_kubernetes_pod_annotation_prometheus_io_port,
      __meta_kubernetes_pod_ip]
      targetLabel: __address__
    - action: replace
      regex: (\d+);((([0-9]+?)(\.|$)){4})
      replacement: $2:$1
      sourceLabels: [__meta_kubernetes_pod_annotation_prometheus_io_port,
      __meta_kubernetes_pod_ip]
```

```
    targetLabel: __address__
  - action: labeldrop
    regex: "__meta_kubernetes_pod_label_(.+)"
  - sourceLabels: [__meta_kubernetes_namespace]
    action: replace
    targetLabel: namespace
  - sourceLabels: [__meta_kubernetes_pod_name]
    action: replace
    targetLabel: pod_name
  - action: replace
    replacement: "basic-istio-system" 2
    targetLabel: mesh_id
```

1

因为 Red Hat OpenShift Service on AWS 监控会忽略 ServiceMonitor 和 PodMonitor 对象中的 namespaceSelector 规格，所以您必须在所有网格命名空间中应用 PodMonitor 对象，包括 control plane 命名空间。

2

字符串 "basic-istio-system" 是 SMCP 名称及其命名空间的组合，但只要使用集群中用户工作负载监控的每个网格都是唯一的，可以使用任何标签。第 2 步中配置的 Kiali 资源的 spec.prometheus.query_scope 需要匹配这个值。

注意

如果使用 user-workload 监控只有一个网格，则 Kiali 资源中的 mesh_id 重新标记和 spec.prometheus.query_scope 字段都是可选的（但如果 mesh_id 标签被删除，则此处给出的 query_scope 字段应该被删除）。

如果集群中的多个网格实例可能会使用 user-workload 监控，则需要 Kiali 资源中的 mesh_id relabelings 和 spec.prometheus.query_scope 字段。这样可确保 Kiali 只查看其关联的网格中的指标。

如果没有部署 Kiali，您仍然可以应用 mesh_id 重新标记，以便来自不同网格的指标可以区分不同的网格。

8. 打开 Red Hat OpenShift Service on AWS Web 控制台，检查指标是否可见。

1.16. 性能和可扩展性

默认 ServiceMeshControlPlane 设置不适用于生产环境，它被设计为在一个默认的 Red Hat OpenShift Service on AWS 安装中成功安装，默认的 OpenShift Container Platform 安装是一个有限的资源环境。在验证了成功安装 SMCP 后，您应该修改 SMCP 中定义的设置以适应您的环境。

1.16.1. 设置计算资源的限制

默认情况下, spec.proxy 具有设置 cpu:10m 和 memory:128M。如果使用 Pilot，spec.runtime.components.pilot 具有相同的默认值。

以下示例中的设置基于 1000 个服务以及每秒1000 个请求。您可以更改 ServiceMeshControlPlane 中的 cpu 和 memory 的值。

流程

1. 在 Red Hat OpenShift Service on AWS web 控制台中，点 Operators → Installed Operators。

2. 点 Project 菜单，选择安装 Service Mesh control plane 的项目，如 istio-system。

3. 点 Red Hat OpenShift Service Mesh Operator。在 Istio Service Mesh Control Plane 列中，点 ServiceMeshControlPlane 的名称，例如 basic。

4. 将独立 Jaeger 实例的名称添加到 ServiceMeshControlPlane。

   a. 点 YAML 标签。

   b. 在 ServiceMeshControlPlane 资源中设置 spec.proxy.runtime.container.resources.requests.cpu,spec.proxy.runtime.container.resources.requests.memory,components.kiali.container, 和 components.global.oauthproxy 的值。

   版本 2.6 ServiceMeshControlPlane 示例

   -

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.6
  proxy:
    runtime:
      container:
        resources:
          requests:
            cpu: 600m
            memory: 50Mi
          limits: {}
  runtime:
    components:
      pilot:
        container:
          resources:
            requests:
              cpu: 1000m
              memory: 1.6Gi
            limits: {}
      kiali:
        container:
          resources:
            limits:
              cpu: "90m"
              memory: "245Mi"
            requests:
              cpu: "30m"
              memory: "108Mi"
      global.oauthproxy:
        container:
          resources:
            requests:
              cpu: "101m"
              memory: "256Mi"
            limits:
              cpu: "201m"
              memory: "512Mi"
```

c. 要为 **Red Hat OpenShift distributed Tracing Platform (Jaeger)**设置值，请参阅**"配置和部署分布式追踪平台 Jaeger"。**

d. 点击 **Save。**

验证

- 点 Reload 来验证 ServiceMeshControlPlane 资源已被正确配置。

1.16.2. 加载测试结果

其他资源

上游 Istio 社区负载测试网格由 1000 个服务和 2000 个 sidecars，带有 70,000 个网格范围请求每秒组成。使用 Istio 1.12.3 运行测试，生成以下结果：

- Envoy 代理每秒每 1000 个通过代理的请求使用 0.35 vCPU 和 40 MB 内存。
- Istiod 使用 1 vCPU 和 1.5 GB 内存。
- Envoy 代理对 90th percentile 延迟增加了 2.65 ms。
- 传统的 istio-telemetry 服务（在 Service Mesh 2.0 中默认禁用）用于使用 Mixer 的部署，每 1000 网格范围内请求每秒使用 0.6 vCPU 请求。数据平面组件（Envoy 代理）处理通过系统的数据流。Service Mesh control plane 组件 Istiod 配置数据平面（data plane）。data plane 和 control plane 有不同的性能问题。

1.16.2.1. Service Mesh Control plane 性能

Istiod 根据用户发布的配置文件和系统当前状态配置 sidecar 代理。在 Kubernetes 环境中，自定义资源定义（CRD）和部署由系统的配置和状态组成。Istio 配置对象，比如网关和虚拟服务，提供用户授权的配置。要生成代理的配置，Istiod 从 Kubernetes 环境和用户授权的配置处理组合配置和系统状态。

Service Mesh control plane 支持数千个服务，分布到成千上万的 pod，它们的用户作者虚拟服务和其他配置对象数量相似。Istiod 的 CPU 和内存要求扩展，以及配置数量和可能的系统状态。CPU 消耗扩展有以下因素：

- 部署更改率。

- 配置更改率。
- 连接到 Istiod 的代理数量。

但这部分本质上是可横向扩展的。

1.16.2.2. data plane 性能

data plane 的性能取决于多个因素，例如：

- 客户端连接数
- 目标请求率
- 请求大小和响应大小
- 代理 worker 线程的数量
- 协议
- CPU 内核
- 代理过滤器的数量和类型，特别是遥测 v2 相关的过滤器。

延迟、吞吐量以及代理的 CPU 和内存消耗作为这些因素的功能来测量。

1.16.2.2.1. CPU 和内存消耗

因为 sidecar 代理对数据路径执行额外的工作，所以它会消耗 CPU 和内存。从 Istio 1.12.3 开始，

代理每秒每 1000 个请求大约消耗 0.5 个 vCPU。

代理的内存消耗取决于代理拥有的总配置状态。大量监听器、集群和路由可以增加内存用量。

因为代理通常不会缓冲传输的数据，所以请求率不会影响内存消耗。

1.16.2.2.2. 额外的延迟

因为 Istio 在数据路径上注入 sidecar 代理，所以延迟是一个重要因素。Istio 向代理添加验证过滤器、遥测过滤器和元数据交换过滤器。每个附加过滤器都会添加到代理内的路径长度中，并影响延迟。

Envoy 代理会在向客户端发送响应后收集原始遥测数据。为请求收集原始遥测所花的时间不会造成完成该请求的总时间。但是，由于 worker 忙于处理请求，因此 worker 不会立即开始处理下一个请求。这个过程为下一个请求的队列等待时间添加，并影响平均延迟和尾部延迟。实际的尾部延迟取决于流量模式。

在网格中，请求会绕过客户端代理，然后是服务器端代理。在 Istio 1.12.3（Istio 带有 telemetry v2）的默认配置中，两个代理会对 90th 和 99th percentile 延迟分布增加 1.7 ms 和 2.7 ms（超过基准数据平面的延迟）。

1.17. 为生产环境配置 SERVICE MESH

当您准备从基本安装迁移到生产环境时，您必须配置 control plane、追踪和安全证书以满足生产要求。

前提条件

- 安装和配置 Red Hat OpenShift Service Mesh。
- 在暂存环境中测试您的配置。

1.17.1. 为生产环境配置 ServiceMeshControlPlane 资源

如果您已安装了一个基本的 ServiceMeshControlPlane 资源来测试 Service Mesh，则必须将其配置

为生产环境中的 Red Hat OpenShift Service Mesh。

您无法更改现有 ServiceMeshControlPlane 资源的 metadata.name 字段。对于生产环境部署，您必须自定义默认模板。

流程

1. 为生产环境配置分布式跟踪平台(Jaeger)。

   a. 编辑 ServiceMeshControlPlane 资源以使用 production 部署策略，方法是将 spec.addons.jaeger.install.storage.type 设置为 Elasticsearch，并在 install 中指定额外的配置选项。您可以创建并配置 Jaeger 实例，并将 spec.addons.jaeger.name 设置为 Jaeger 实例的名称。

   默认 Jaeger 参数，包括 Elasticsearch

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
  tracing:
    sampling: 100
    type: Jaeger
  addons:
    jaeger:
      name: MyJaeger
      install:
        storage:
          type: Elasticsearch
        ingress:
          enabled: true
  runtime:
    components:
      tracing.jaeger.elasticsearch: # only supports resources and image name
        container:
          resources: {}
```

b. 为生产环境配置抽样率。如需更多信息，请参阅性能和可扩展性部分。

2. 通过从外部证书颁发机构安装安全证书，确保您的安全证书已就绪。如需更多信息，请参阅安全部分。

验证

1. 输入以下命令验证 ServiceMeshControlPlane 资源是否已正确更新。在本例中，basic 是 ServiceMeshControlPlane 资源的名称。

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

1.17.2. 其他资源

- 有关性能调整 Service Mesh 的更多信息，请参阅性能和可扩展性。

1.18. 连接服务网格

Federation（联邦）是一种部署模型，可让您在不同管理域中管理的单独网格间共享服务和工作负载。

1.18.1. 联邦概述

Federation（联邦）是一组可让您在独立网格间连接服务的功能，允许在多个不同的管理域中使用 Service Mesh 功能，如身份验证、授权和流量管理。

通过实施联邦网格，您可以运行、管理和观察在多个 OpenShift 集群中运行的单个服务网格。Red Hat OpenShift Service Mesh 联邦针对 Service Mesh 的多集群实施，该方法假设网格之间的信任*最小*。

Service Mesh federation 假设每个网格都单独管理，并保留自己的管理员。默认的行为是不允许任何通信，且网格之间没有共享信息。在网格间共享信息是基于明确选择的。联邦网格中的任何内容都不是共享的，除非为共享进行了配置。证书生成、指标和追踪集合等支持功能在其各自网格中保持本地。

您可以在每个服务网格中配置 ServiceMeshControlPlane，以创建专用于联合的入口和出口网关，并

为网格指定信任域。

联邦还包括创建额外的联邦文件。以下资源用于在两个或多个网格间配置联合。

- **ServiceMeshPeer** 资源声明一对服务网格之间的联邦。
- **ExportedServiceSet** 资源声明网格中的一个或多个服务可供对等网格使用。
- **ImportedServiceSet** 资源声明对等网格导出的服务将导入到网格中。

1.18.2. 联邦特性

Red Hat OpenShift Service Mesh 联邦方法加入网格的功能包括：

- 支持每个网格的通用 root 证书。
- 支持每个网格的不同根证书。
- Mesh 管理员必须手动配置证书链、服务发现端点、信任域等，以用于 RU mesh 之外的网格。
- 仅导出/导入您要在网格间共享的服务。
    - 默认为不与联邦中其他网格共享已部署的工作负载的信息。可以 导出 服务使其对其他网格可见，并允许来自其自身网格外的工作负载的请求。
    - 已导出的服务可以被导入到另一个网格中，使网格上的工作负载能够将请求发送到导入的服务。

- 对网格之间的通信进行加密。

- 支持在本地部署的工作负载和在联邦中部署的另一个网格中部署的工作负载之间配置负载平衡。

当网格加入到另一个网格时，它可以执行以下操作：

- 向联邦网格提供有关自身的信任详情。

- 发现联邦网格的信任详情。

- 向联邦网格提供有关其自身导出服务的信息。

- 发现联邦网格导出的服务的信息。

1.18.3. 联邦安全

Red Hat OpenShift Service Mesh 联邦针对 Service Mesh 的多集群实施，该方法假设网格之间的信任最小。数据安全性作为联邦功能的一部分而建立。

- 每个网格被视为唯一租户，具有独特的管理。

- 您可以在联邦中为每个网格创建一个唯一的信任域。

- 联邦网格之间的流量使用 mutual Transport Layer Security(mTLS)自动加密。

- Kiali 图仅显示您导入的网格和服务。您无法看到还没有导入到网格中的其他网格或服务。

1.18.4. 联邦限制

Red Hat OpenShift Service Mesh 联邦方法加入网格有以下限制：

- OpenShift Dedicated 不支持网格绑定。

1.18.5. 联邦先决条件

Red Hat OpenShift Service Mesh 联邦方法加入网格需要以下先决条件：

- 两个或更多个 Red Hat OpenShift Service on AWS 4.6 或更高版本的集群。
- 联邦（Federation）是在 Red Hat OpenShift Service Mesh 2.1 或更高版本中引入的。您必须已在您要联合的每个网格上安装了 Red Hat OpenShift Service Mesh 2.1 或更高的 Operator。
- 您必须在您要联邦的每个网格上部署 2.1 或更高版本的 ServiceMeshControlPlane。
- 您必须配置支持与联邦网关关联的服务的负载均衡器，以支持原始 TLS 流量。联合流量包括用于发现的 HTTPS 和用于服务流量的原始加密 TCP。
- 在导出并导入它们前，您应该部署要公开给另一个网格的服务。但这不是严格的要求。您可以指定导出/导入尚不存在的服务名称。当您部署在 ExportedServiceSet 和 ImportedServiceSet 中命名的的服务时，它们会自动提供给导出/导入。

1.18.6. 规划网格联邦

在开始配置网格联邦前，您应该需要一些时间来规划您的实施。

- 您计划将多少网格加入到联邦？您可能想从有限数量的网格开始，可能是两个或三个网格。
- 您计划为每个网格使用哪些命名约定？使用预定义的命名约定有助于配置和故障排除。本文档中的示例为每个网格使用不同的颜色。您应该决定一个命名约定，它可帮助您确定谁拥有和管

理每个网格，以及以下联合资源：

- 集群名称
- 集群网络名称
- Mesh 名称和命名空间
- Federation ingress 网关
- Federation egress 网关
- 安全信任域

注意

联邦中的每个网格都必须有自己的唯一信任域。

- 您计划从每个网格中导出哪些服务到联邦网格？每个服务都可以单独导出，也可以指定标签或使用通配符。
  - 是否要将别名用于服务命名空间？
  - 是否要将别名用于导出的服务？
- 每个网格计划导入哪些导出的服务？每个网格只导入它所需的服务。
  - 是否要将别名用于导入的服务？

1.18.7. 集群间的 Mesh 联合

要将 OpenShift Service Mesh 的一个实例与在不同集群中运行的实例连接，这个过程与连接同一集群中部署的两个网格的过程有很大不同。但是，一个网格的 ingress 网关必须可以被另一个网格访问。确保这一点的一种方法是，如果集群支持这种类型的服务，将网关服务配置为 LoadBalancer 服务。

该服务必须通过在 OSI 模型的第 4 层运行的负载平衡器公开。

1.18.7.1. 在裸机上运行的集群上公开联合入口

如果集群在裸机上运行并完全支持 LoadBalancer 服务，则 ingress 网关 Service 对象的 .status.loadBalancer.ingress.ip 字段中的 IP 地址应指定为 ServiceMeshPeer 对象的 .spec.remote.addresses 字段中的条目之一。

如果集群不支持 LoadBalancer 服务，则如果节点可从运行其他网格的集群访问，则可以使用 NodePort 服务。在 ServiceMeshPeer 对象中，在 .spec.remote.addresses 字段中指定节点的 IP 地址，并在 .spec.remote.discoveryPort 和 .spec.remote.servicePort 字段中指定服务的节点端口。

1.18.7.2. 在 Amazon Web Services(AWS)上公开联邦入口。

默认情况下，在 AWS 上运行的集群中的 LoadBalancer 服务不支持 L4 负载均衡。为了使 Red Hat OpenShift Service Mesh 联邦可以正常工作，必须在 ingress 网关服务中添加以下注解：

```
service.beta.kubernetes.io/aws-load-balancer-type: nlb
```

在 ingress 网关 Service 对象的 .status.loadBalancer.ingress.hostname 字段中的完全限定域名应指定为 ServiceMeshPeer 对象的 .spec.remote.addresses 字段中的条目之一。

1.18.8. 联邦实施清单

联邦服务网格包括以下活动：

- ❑ 在您要联邦的集群间配置网络。

- ❑ 配置支持与联邦网关关联的服务的负载均衡器，以支持原始 TLS 流量。

❑ 在集群中安装 Red Hat OpenShift Service Mesh 版本 2.1 或更高版本的 Operator。

❑ 为每个集群部署版本 2.1 或更高版本的 ServiceMeshControlPlane。

❑ 为您要联邦的每个网格配置 SMCP：

- ❑ 为您要联邦的每个网格创建一个联邦出口网关
- ❑ 为您要联邦的每个网格创建一个联邦入口网关
- ❑ 配置唯一的信任域。

❑ 通过为每个网格对创建 ServiceMeshPeer 资源来联邦两个或多个网格。

❑ 通过创建一个 ExportedServiceSet 资源导出服务，使等网网格可以访问这些服务。

❑ 通过创建一个 ImportedServiceSet 资源来导入服务，导入网格对等点共享的服务。

1.18.9. 为联邦配置 Service Mesh control plane

在网格可以被联邦前，您必须为网格联邦配置 ServiceMeshControlPlane。因为属于联邦成员的所有网格都是相等的，并且每个网格都独立管理，所以您必须 *为每个* 加入联邦的网格配置 SMCP。

在以下示例中，red-mesh 的管理员将配置 SMCP 以同时使用 green-mesh 和 blue-mesh 进行联邦。

Red-mesh 的 SMCP 示例

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: red-mesh
  namespace: red-mesh-system
spec:
  version: v2.6
  runtime:
    defaults:
      container:
        imagePullPolicy: Always
  gateways:
    additionalEgress:
      egress-green-mesh:
        enabled: true
        requestedNetworkView:
        - green-network
        routerMode: sni-dnat
        service:
          metadata:
            labels:
              federation.maistra.io/egress-for: egress-green-mesh
          ports:
          - port: 15443
            name: tls
          - port: 8188
            name: http-discovery  #note HTTP here
      egress-blue-mesh:
        enabled: true
        requestedNetworkView:
        - blue-network
        routerMode: sni-dnat
        service:
          metadata:
            labels:
              federation.maistra.io/egress-for: egress-blue-mesh
          ports:
          - port: 15443
            name: tls
          - port: 8188
            name: http-discovery  #note HTTP here
    additionalIngress:
      ingress-green-mesh:
        enabled: true
        routerMode: sni-dnat
        service:
          type: LoadBalancer
          metadata:
            labels:
              federation.maistra.io/ingress-for: ingress-green-mesh
          ports:
          - port: 15443
            name: tls
```

```
        - port: 8188
          name: https-discovery  #note HTTPS here
      ingress-blue-mesh:
        enabled: true
        routerMode: sni-dnat
        service:
          type: LoadBalancer
          metadata:
            labels:
              federation.maistra.io/ingress-for: ingress-blue-mesh
          ports:
          - port: 15443
            name: tls
          - port: 8188
            name: https-discovery  #note HTTPS here
  security:
    identity:
      type: ThirdParty
    trust:
      domain: red-mesh.local
```

表 1.6. ServiceMeshControlPlane 联邦配置参数

参数	描述	值	默认值

`spec:`
`  cluster:`
`    name:`
	集群的名称。您不需要指定集群名称，但有助于进行故障排除。	字符串	N/A

`spec:`
`  cluster:`
`    network:`
	集群网络的名称。您不必为网络指定名称，但对配置和故障排除很有帮助。	字符串	N/A

1.18.9.1. 了解联邦网关

您可以使用 网关 来管理入站和出站流量，允许您指定您要进入或离开网格的流量。

您可以使用入口和出口网关来管理进入和离开服务网格（North-South 流量）的流量。当您创建联邦网格时，您可以创建额外的入口/出口网关，以便联邦网格间的服务发现、联邦网格之间的通信，以及管理服务网格（East-West 流量）之间的流量流。

为了避免网格间的命名冲突，您必须为每个网格创建单独的出口和入口网关。例如，**red-mesh** 将具有单独的出口网关，用于发送到 **green-mesh** 和 **blue-mesh** 的流量。

表 1.7. 联邦网关参数

参数	描述	值	默认值

spec:
  gateways:
    additionalEgress:
      <egress_name>:
	在联邦中为每个网格对等点定义额外的出口网关。		

spec:
  gateways:
    additionalEgress:
      <egress_name>:
        enabled:
	这个参数启用或禁用联邦出口。	**true**/**false**	**true**

spec:
  gateways:
    additionalEgress:
      <egress_name>:
        requestedNetworkView:
	与导出的服务关联的网络。	设置为网格 SMCP 中的 **spec.cluster.network** 值，否则使用 <ServiceMeshPeer-name>-network。例如，如果那个网格的 **ServiceMeshPeer** 资源命名为 **west**，则该网络将命名为 **west-network**。	

spec:
  gateways:
    additionalEgress:
      <egress_name>:
        service:
          metadata:
            labels:
              federation.maistra.io/egress-for:
	为网关指定一个唯一标签，以防止联邦流量通过集群的默认系统网关流。		

参数	描述	值	默认值

spec:
  gateways:
 	
additionalEgress:
 	
<egress_name>:
    service:
      ports:
	指定用于 TLS 和服务发现的 **port:** 和 **name:**。联邦流量由服务流量的原始加密 TCP 组成。	将 TLS 服务请求发送到联邦中的其他网格需要端口 **15443**。将服务发现请求发送到联邦中的其他网格需要端口 **8188**。	

spec: gateways: additionalIngress:	在联邦中为*每*个网格对等点定义额外的入口网关网关。		

spec:
  gateways:
    additionalIgress:
 	
<ingress_name>:
    enabled:
	此参数启用或禁用联邦入口。	**true/false**	**true**

spec:
  gateways:
 	
additionalIngress:
 	
<ingress_name>:
    service:
      type:
	入口网关服务必须通过在 OSI 模型的第 4 层运行并公开可用的负载均衡器公开。	**LoadBalancer**	

spec:
  gateways:
 	
additionalIngress:
 	
<ingress_name>:
    service:
      type:
	如果集群不支持 **LoadBalancer** 服务，则可以通过 **NodePort** 服务公开入口网关服务。	**NodePort**	

参数	描述	值	默认值

`spec:`
`  gateways:`
`    additionalIngress:`
`      <ingress_name>:`
`        service:`
`          metadata:`
`            labels:`
`              federation.maistra.io/ingress-for:`
	为网关指定一个唯一标签，以防止联邦流量通过集群的默认系统网关流。		

`spec:`
`  gateways:`
`    additionalIngress:`
`      <ingress_name>:`
`        service:`
`          ports:`
	指定用于 TLS 和服务发现的 **port:** 和 **name:**。联邦流量由服务流量的原始加密 TCP 组成。联邦流量由 HTTPS 用于发现。	在向联邦中的其他网格接收 TLS 服务请求时，需要端口 **15443**。在向联邦中的其他网格接收服务发现请求时，需要端口 **8188**。	

`spec:`
`  gateways:`
`    additionalIngress:`
`      <ingress_name>:`
`        service:`
`          ports:`
`            nodePort:`
	用于指定 **nodePort：** 如果集群不支持 **LoadBalancer** 服务。	如果指定，除了 **port:** 和 **name:** 外，还需要用于 TLS 和服务发现。**NodePort：** 范围需要为 **30000**-**32767**。	

在以下示例中，管理员使用 NodePort 服务与 green-mesh 来配置 SMCP。

NodePort 的 SMCP 示例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: green-mesh
  namespace: green-mesh-system
```

```
spec:
# ...
  gateways:
    additionalIngress:
     ingress-green-mesh:
       enabled: true
       service:
         type: NodePort
         metadata:
           labels:
             federation.maistra.io/ingress-for: ingress-green-mesh
         ports:
         - port: 15443
           nodePort: 30510
           name: tls
         - port: 8188
           nodePort: 32359
           name: https-discovery
```

1.18.9.2. 了解联邦信任域参数

联邦中的每个网格都必须有自己的唯一信任域。这个值用于在 ServiceMeshPeer 资源中配置 mesh federation。

```
kind: ServiceMeshControlPlane
metadata:
  name: red-mesh
  namespace: red-mesh-system
spec:
  security:
    trust:
      domain: red-mesh.local
```

表 1.8. 联邦安全参数

参数	描述	值	默认值

`spec:`
`  security:`
`    trust:`
`      domain:`
	用于为网格指定信任域的唯一名称。域对于联邦中的每个网格都必须是唯一的。	<mesh-name>.local	N/A

控制台的步骤

按照以下步骤，使用 Red Hat OpenShift Service on AWS Web 控制台编辑

ServiceMeshControlPlane。本例使用 red-mesh 作为示例。

1. 以具有 cluster-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS 控制台。

2. 导航到 Operators → Installed Operators。

3. 点 Project 菜单，选择安装 Service Mesh control plane 的项目。例如： red-mesh-system。

4. 点 Red Hat OpenShift Service Mesh Operator。

5. 在 Istio Service Mesh Control Plane 选项卡中，点击 ServiceMeshControlPlane 的名称，如 red-mesh。

6. 在 Create ServiceMeshControlPlane Details 页中，点 YAML 修改您的配置。

7. 修改 ServiceMeshControlPlane 以添加联合入口和出口网关，并指定信任域。

8. 点 Save。

通过 CLI 操作的步骤

按照以下步骤使用命令行创建或编辑 ServiceMeshControlPlane。本例使用 red-mesh 作为示例。

1. 以具有 cluster-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS CLI。输入以下命令。然后在提示时输入您的用户名和密码。

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. 切换到安装 Service Mesh control plane 的项目，如 red-mesh-system。

```
$ oc project red-mesh-system
```

3. 编辑 ServiceMeshControlPlane 文件，添加联合入口和出口网关，并指定信任域。

4. 运行以下命令编辑 Service Mesh control plane，其中 red-mesh-system 是系统命名空间，red-mesh 是 ServiceMeshControlPlane 对象的名称：

```
$ oc edit -n red-mesh-system smcp red-mesh
```

5. 输入以下命令，其中 red-mesh-system 是系统命名空间，以查看 Service Mesh control plane 安装的状态。

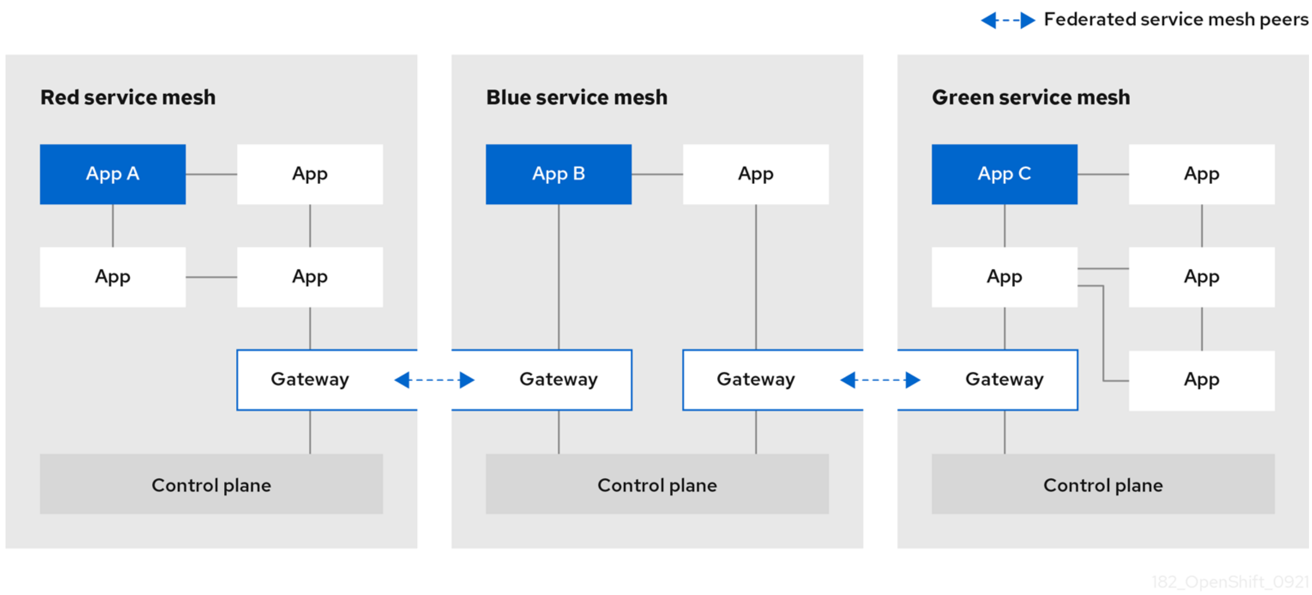
```
$ oc get smcp -n red-mesh-system
```

当 READY 列指出所有组件都已就绪时，安装成功完成。

```
NAME      READY   STATUS            PROFILES      VERSION   AGE
red-mesh   10/10   ComponentsReady   ["default"]   2.1.0     4m25s
```

1.18.10. 加入联邦网格

您可以通过创建 ServiceMeshPeer 资源来声明两个网格之间的联邦。ServiceMeshPeer 资源定义了两个网格之间的联邦，您可以使用它为对等网格配置发现功能，访问对等网格网格，以及用于验证其他网格客户端的证书。



网格以一对一为基础联邦，因此每对等点都需要一对 ServiceMeshPeer 资源指定与其他服务网格的联邦连接。例如，联合名为 red 和 green 的两个网格需要两个 ServiceMeshPeer 文件。

1. 在 red-mesh-system 上，为绿色网格创建一个 ServiceMeshPeer。

2. 在 green-mesh-system 上，为红色网格创建一个 ServiceMeshPeer。

对名为 red, blue, 和 green 的三个网格进行联邦会需要六个 ServiceMeshPeer 文件。

1. 在 red-mesh-system 上，为绿色网格创建一个 ServiceMeshPeer。

2. 在 red-mesh-system 上，为蓝网格创建一个 ServiceMeshPeer。

3. 在 green-mesh-system 上，为红色网格创建一个 ServiceMeshPeer。

4. 在 green-mesh-system 上，为蓝网格创建一个 ServiceMeshPeer。

5. 在 blue-mesh-system 上，为红色网格创建一个 ServiceMeshPeer。

6. 在 blue-mesh-system 上，为绿色网格创建一个 ServiceMeshPeer。

ServiceMeshPeer 资源中的配置包括以下：

- 其他网格的 ingress 网关的地址，用于发现和服务请求。
- 用于与指定对等网格交互的本地入口和出口网关名称。
- 将请求发送到此网格时由其他网格使用的客户端 ID。
- 其他网格使用的信任域。
- 包含根证书的 ConfigMap 名称，用于验证由其他网格使用的信任域中的客户端证书。

在以下示例中，red-mesh 的管理员使用 green-mesh 配置联邦。

red-mesh 的 ServiceMeshPeer 资源示例

```
kind: ServiceMeshPeer
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  remote:
    addresses:
    - ingress-red-mesh.green-mesh-system.apps.domain.com
  gateways:
    ingress:
      name: ingress-green-mesh
    egress:
      name: egress-green-mesh
  security:
    trustDomain: green-mesh.local
    clientID: green-mesh.local/ns/green-mesh-system/sa/egress-red-mesh-service-account
```

```
certificateChain:
  kind: ConfigMap
  name: green-mesh-ca-root-cert
```

表 1.9. ServiceMeshPeer 配置参数

参数	描述	值
`metadata:` `  name:`	此资源配置联合的对等网格名称。	字符串
`metadata:` `  namespace:`	此网格的系统命名空间，即安装了 Service Mesh control plane。	字符串

`spec:`
`  remote:`
`    addresses:`
	对等网格提供请求的 ingress 网关的公共地址列表。	

`spec:`
`  remote:`
`    discoveryPort:`
	地址处理发现请求的端口。	默认值为 8188

`spec:`
`  remote:`
`    servicePort:`
	地址处理服务请求的端口。	默认值为 15443

`spec:`
`  gateways:`
`    ingress:`
`      name:`
	此网格上为从 peer 网格接收的请求提供服务的网格的 ingress 名称。例如：**ingress-green-mesh**。	

`spec:`
`  gateways:`
`    egress:`
`      name:`
	此网格上为发送到 peer 网格的请求提供服务的出口名称。例如，**egress-green-mesh**。	

参数	描述	值

spec:
  security:
    trustDomain:
	peer 网格使用的信任域。	<peerMeshName>.local

spec:
  security:
    clientID:
	对等网格在调用此网格时使用的客户端 ID。	<peerMeshTrustDomain>/ns/<peerMeshSystem>/sa/<peerMeshEgressGatewayName>-service-account

spec:
  security:
    certificateChain:
      kind: ConfigMap
      name:
	包含根证书的资源的 kind（如 ConfigMap）和名称，用于验证由 peer 网格提供给这个网格的客户端和服务器证书。包含证书的配置映射条目的密钥应当是 **root-cert.pem**。	kind: ConfigMap name: <peerMesh>-ca-root-cert

1.18.10.1. 创建 ServiceMeshPeer 资源

前提条件

- 两个或更多个 Red Hat OpenShift Service on AWS 4.6 或更高版本的集群。
- 集群必须已经联网。
- 支持与联邦网关关联的服务的负载均衡器必须配置为支持原始 TLS 流量。
- 每个集群都必须配置 2.1 或更高版本的 ServiceMeshControlPlane 来支持部署的联邦。
- 具有 cluster-admin 角色的帐户。

通过 CLI 操作的步骤

按照以下步骤，从命令行创建 ServiceMeshPeer 资源。本例演示，red-mesh 为 green-mesh 创建一个对等资源。

1.

以具有 cluster-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS CLI。输入以下命令。然后在提示时输入您的用户名和密码。

```
$ oc login --username=<NAMEOFUSER> <API token> https://<HOSTNAME>:6443
```

2. 切换到安装 control plane 的项目，如 red-mesh-system。

```
$ oc project red-mesh-system
```

3. 根据以下示例为您要联合的两个网格创建一个 ServiceMeshPeer 文件。

red-mesh 到 green-mesh 的 ServiceMeshPeer 资源示例

```
kind: ServiceMeshPeer
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  remote:
    addresses:
    - ingress-red-mesh.green-mesh-system.apps.domain.com
  gateways:
    ingress:
      name: ingress-green-mesh
    egress:
      name: egress-green-mesh
  security:
    trustDomain: green-mesh.local
    clientID: green-mesh.local/ns/green-mesh-system/sa/egress-red-mesh-service-account
    certificateChain:
      kind: ConfigMap
      name: green-mesh-ca-root-cert
```

4. 运行以下命令来部署资源，其中 red-mesh-system 是系统命名空间，servicemeshpeer.yaml 包含您编辑的文件的完整路径：

```
$ oc create -n red-mesh-system -f servicemeshpeer.yaml
```

5. 要确认在红色网格和绿色网格间建立了连接，请在 red-mesh-system命名空间中检查 green-mesh ServiceMeshPeer 的状态：

```
$ oc -n red-mesh-system get servicemeshpeer green-mesh -o yaml
```

red-mesh 和 green-mesh 之间的 ServiceMeshPeer 连接示例

```yaml
status:
 discoveryStatus:
  active:
  - pod: istiod-red-mesh-b65457658-9wq5j
   remotes:
   - connected: true
     lastConnected: "2021-10-05T13:02:25Z"
     lastFullSync: "2021-10-05T13:02:25Z"
     source: 10.128.2.149
   watch:
     connected: true
     lastConnected: "2021-10-05T13:02:55Z"
     lastDisconnectStatus: 503 Service Unavailable
     lastFullSync: "2021-10-05T13:05:43Z"
```

status.discoveryStatus.active.remotes 字段显示 peer mesh 中的 istiod（在本例中为绿色网格）连接到当前网格中的 istiod（本例中为红色网格）。

status.discoveryStatus.active.watch 字段显示当前网格中的 istiod 连接到对等网格中的 istiod。

如果在 green-mesh-system 中检查名为 red-mesh 的 servicemeshpeer，您会从绿色网格的角度找到有关同一两个连接的信息。

当两个网格之间没有建立连接时，ServiceMeshPeshPeer 状态在 status.discoveryStatus.inactive 字段中显示此状态。

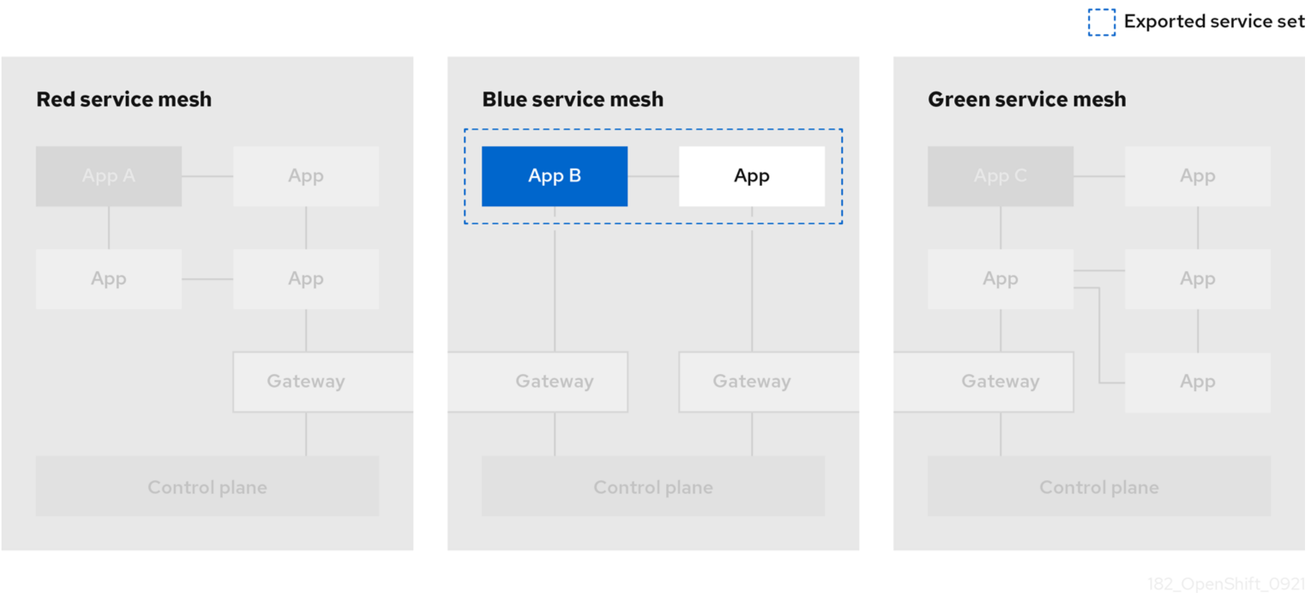
有关连接尝试失败的更多信息，请检查 Istiod 日志，访问日志处理对等网络中的出口流量，以及处理对等网格中当前网格的入口流量的 ingress 网关。

例如，如果红色网格无法连接到绿色网格，请检查以下日志：

- red-mesh-system 中的 Istiod-red-mesh
- red-mesh-system 中的 egress-green-mesh
- green-mesh-system 中的 ingress-red-mesh

1.18.11. 从联邦网格导出服务

导出服务允许网格与联邦网格的另一个成员共享一个或多个服务。



您可以使用 ExportedServiceSet 资源，在一个网格中声明您要提供给联邦网格中的另一个对等点的服务。您必须明确声明每个服务要与同级服务器共享。

- 您可以根据命名空间或名称选择服务。

- 您可以使用通配符来选择服务；例如，导出命名空间中的所有服务。

- 您可以使用别名导出服务。例如，您可以将 foo/bar 服务导出为 custom-ns/bar。

- 您只能导出对网格系统命名空间可见的服务。例如：在另一个命名空间中，将 networking.istio.io/exportTo 标签设置为 '.' 的服务不会是导出的候选者。

- 对于导出的服务，它们的目标服务将只看到来自入口网关的流量，而不是原始请求者（即，它们不会看到来自其他网格的出口网关的客户端 ID 或源自请求的工作负载）

以下示例是 red-mesh 导出至 green-mesh 的服务。

ExportedServiceSet 资源示例

```
kind: ExportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  exportRules:
  # export ratings.mesh-x-info as ratings.bookinfo
  - type: NameSelector
    nameSelector:
      namespace: red-mesh-info
      name: red-ratings
      alias:
        namespace: info
        name: ratings
  # export any service in red-mesh-info namespace with label export-service=true
  - type: LabelSelector
    labelSelector:
      namespace: red-mesh-info
      selector:
        matchLabels:
          export-service: "true"
      aliases: # export all matching services as if they were in the info namespace
      - namespace: "*"
        name: "*"
        alias:
          namespace: info
```

表 1.10. ExportedServiceSet 参数

参数	描述	值
metadata: name:	将此服务公开给的 ServiceMeshPeshPeer 的名称。	必须与 **ServiceMeshPeer** 资源中的网格 **name** 值匹配。
metadata: namespace:	包含此资源的项目/命名空间的名称（应该是网格的系统命名空间）。	

spec:
  exportRules:
  - type:
	监管此服务的导出的规则类型。为服务找到的第一个匹配规则将用于导出。	**NameSelector**, **LabelSelector**

spec:
  exportRules:
  - type: NameSelector
    nameSelector:
      namespace:
      name:
	要创建 **NameSelector** 规则，请指定服务 **命名空间** 以及 **Service** 资源中定义的服务 **名称**。	

spec:
  exportRules:
  - type: NameSelector
    nameSelector:
      alias:
        namespace:
        name:
	要在为服务创建使用别名的 **NameSelector** 规则，在为服务指定 **命名空间** 和 **名称** 后，请指定 **命名空间** 的别名以及用于服务**名称**的别名。	

spec:
  exportRules:
  - type: LabelSelector
    labelSelector:
      namespace:
<exportingMesh>
      selector:
        matchLabels:
          <labelKey>:
<labelValue>
	要创建 **LabelSelector** 规则，请指定服务的**命名空间**并指定 **Service** 资源中定义的 **label**。在上例中，标签是 **export-service**。	

参数	描述	值

<pre>spec:
  exportRules:
  - type: LabelSelector
    labelSelector:
      namespace:
<exportingMesh>
      selector:
        matchLabels:
          <labelKey>:
<labelValue>
      aliases:
      - namespace:
        name:
        alias:
          namespace:
          name:</pre>
	要为服务创建使用别名的 **LabelSelector** 规则，在指定 **选择器** 后，请指定用于服务**名称**或**命名空间**的别名。在上例中，所有匹配服务的命名空间别名都是 **info**。	

使用名称"勘误"的服务从红色的所有命名空间导出到 blue-mesh。

```
kind: ExportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: blue-mesh
  namespace: red-mesh-system
spec:
  exportRules:
  - type: NameSelector
    nameSelector:
      namespace: "*"
      name: ratings
```

将 west-data-center 命名空间中的所有服务导出到 green-mesh

```
kind: ExportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  exportRules:
```

```
- type: NameSelector
  nameSelector:
    namespace: west-data-center
    name: "*"
```

1.18.11.1. 创建 ExportedServiceSet

您可以创建一个 ExportedServiceSet 资源来显式声明您要提供给网格对等的服务。

服务导出为 <export-name>.<export-namespace>.svc.<ServiceMeshPeer.name>-exports.local，它将自动路由到目标服务。这是导出的服务在导出网格中已知的名称。当入口网关收到用于此名称的请求时，它将被路由到要导出的实际服务。例如，如果名为 ratings.red-mesh-info 的服务导出至 green-mesh 作为 ratings.bookinfo，则服务将在名称 ratings.bookinfo.svc.green-mesh-exports.local 下导出，由该主机名的 ingress 网关接收的流量将路由到 ratings.red-mesh-bookinfo 服务。

注意

当您将 importAsLocal 参数设置为 true 以将远程端点与本地服务聚合时，您必须为该服务使用别名。当您设置参数 false 时，不需要别名。

先决条件

- 为网格联邦配置了集群和 ServiceMeshControlPlane。
- 具有 cluster-admin 角色的帐户。

注意

您可以配置服务以进行导出，即使这些服务尚不存在。当部署与 ExportedServiceSet 中指定的值匹配的服务时，将自动导出该服务。

通过 CLI 操作的步骤

按照以下步骤，从命令行创建 ExportedServiceSet。

1. 以具有 cluster-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS CLI。输入以下命令。然后在提示时输入您的用户名和密码。

```
$ oc login --username=<NAMEOFUSER> <API token> https://<HOSTNAME>:6443
```

2. 切换到安装 Service Mesh control plane 的项目，如 red-mesh-system。

```
$ oc project red-mesh-system
```

3. 根据以下示例创建 ExportedServiceSet 文件，其中 red-mesh 将服务导出到 green-mesh。

从 red-mesh 到 green-mesh 的 ExportedServiceSet 资源示例

```
apiVersion: federation.maistra.io/v1
kind: ExportedServiceSet
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  exportRules:
  - type: NameSelector
    nameSelector:
      namespace: red-mesh-info
      name: ratings
      alias:
        namespace: info
        name: red-ratings
  - type: NameSelector
    nameSelector:
      namespace: red-mesh-info
      name: reviews
```

4. 运行以下命令，在 red-mesh-system 命名空间中上传并创建 ExportedServiceSet 资源。

```
$ oc create -n <ControlPlaneNamespace> -f <ExportedServiceSet.yaml>
```

例如：

```
$ oc create -n red-mesh-system -f export-to-green-mesh.yaml
```

5. 根据需要为联合网格中的每个网格对等创建额外的 ExportedServiceSets。

验证

- 运行以下命令来验证红色导出与 green-mesh 共享的服务：

```
$ oc get exportedserviceset <PeerMeshExportedTo> -o yaml
```

例如：

```
$ oc -n red-mesh-system get exportedserviceset green-mesh -o yaml
```

从红色网格导出的服务验证与绿色网格共享的示例。

```
status:
  exportedServices:
  - exportedName: red-ratings.info.svc.green-mesh-exports.local
    localService:
      hostname: ratings.red-mesh-info.svc.cluster.local
      name: ratings
      namespace: red-mesh-info
  - exportedName: reviews.red-mesh-info.svc.green-mesh-exports.local
    localService:
      hostname: reviews.red-mesh-info.svc.cluster.local
      name: reviews
      namespace: red-mesh-info
```

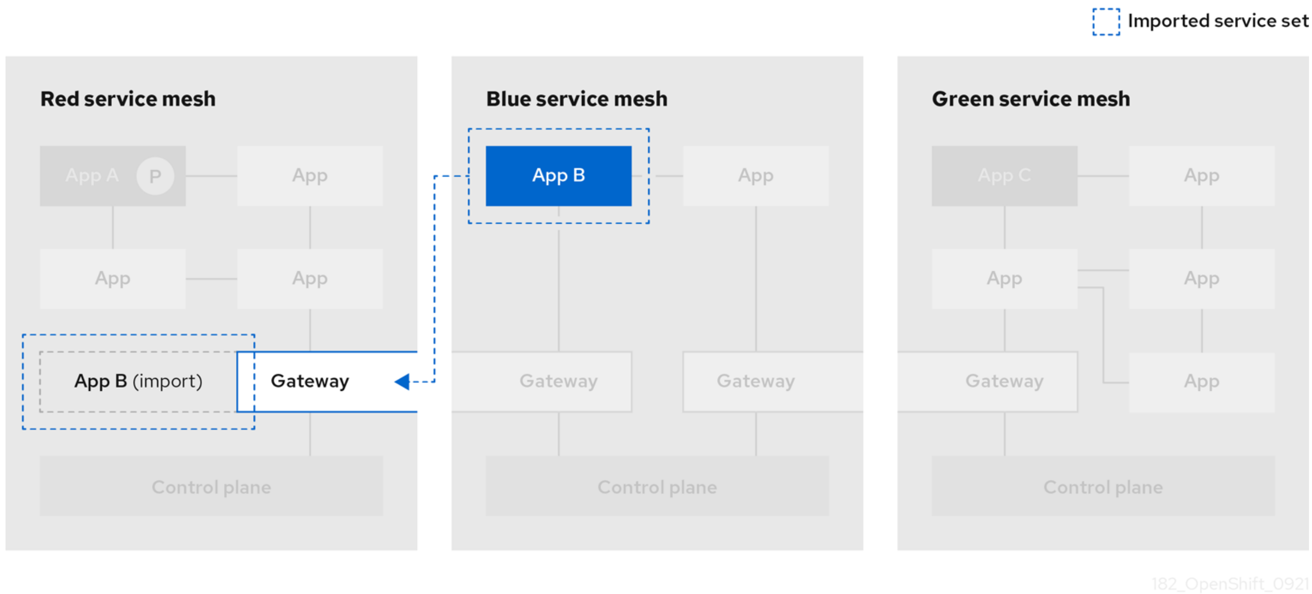
status.exportedServices 数组列出了当前导出的服务（这些服务与 ExportedServiceSet 对象中的导出规则匹配）。数组中的每个条目都指明导出的服务的名称，以及所导出的本地服务

的详细信息。

如果缺少您要导出的服务，请确认 Service 对象存在，其名称或标签与 ExportedServiceSet 对象中定义的 exportRules 匹配，并且 Service 对象的命名空间被配置为使用 ServiceMeshMemberRoll 或 ServiceMeshMember 对象作为服务网格的成员。

1.18.12. 将服务导入到联邦网格中

导入服务可让您明确指定从另一个网格导出的服务应在服务网格内访问。



您可以使用 ImportedServiceSet 资源来选择导入的服务。网格只可使用由网格 peer 导出并明确导入的服务。您没有显式导入的服务不会在网格中提供。

- 您可以根据命名空间或名称选择服务。
- 您可以使用通配符来选择服务，例如，将导出的所有服务导入到命名空间。
- 您可以使用标签选择器（可以是全局到网格）或作用于特定成员命名空间来选择用于导出的服务。

- 您可以使用别名导入服务。例如，您可以将 custom-ns/bar 服务导入为 other-mesh/bar。

- 您可以指定一个自定义域后缀，该后缀将附加到所导入服务的 name.namespace 的完全限定域名中；例如 bar.other-mesh.imported.local。

以下示例是 green-mesh 导入 通过 red-mesh 导出的服务。

ImportedServiceSet 示例

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh #name of mesh that exported the service
  namespace: green-mesh-system #mesh namespace that service is being imported into
spec:
  importRules: # first matching rule is used
  # import ratings.info as ratings.bookinfo
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: info
      name: ratings
      alias:
        # service will be imported as ratings.info.svc.red-mesh-imports.local
        namespace: info
        name: ratings
```

表 1.11. ImportedServiceSet 参数

参数	描述	值
`metadata:` `  name:`	将服务导出到联邦网格的 ServiceMeshPeer 的名称。	
`metadata:` `  namespace:`	包含 ServiceMeshPeer 资源（网格系统命名空间）的命名空间名称。	

参数	描述	值

`spec:`
`  importRules:`
`  - type:`
	监管该服务导入的规则类型。为服务找到的第一个匹配规则将用于导入。	**NameSelector**

`spec:`
`  importRules:`
`  - type: NameSelector`
`    nameSelector:`
`      namespace:`
`      name:`
	要创建 **NameSelector** 规则，请指定导出的服务的**命名空间**和**名称**。	

`spec:`
`  importRules:`
`  - type: NameSelector`
`    importAsLocal:`
	设置为 **true**，以将远程端点与本地服务聚合。当将 **true** 服务导入为 **<name>.<namespace>.svc.cluster.local** 时。为 **true** 时，需要一个别名。为 **false** 时，不需要别名。	**true/false**

`spec:`
`  importRules:`
`  - type: NameSelector`
`    nameSelector:`
`      namespace:`
`      name:`
`      alias:`
`        namespace:`
`        name:`
	要在为服务创建使用别名的 **NameSelector** 规则，在为服务指定 **命名空间** 和 **名称** 后，请指定 **命名空间** 的别名以及用于服务**名称**的别名。	

将"info/ratings"服务从红色导入到 blue-mesh

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh
  namespace: blue-mesh-system
spec:
  importRules:
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: info
      name: ratings
```

将 red-mesh 的 west-data-center 命名空间中的所有服务导入到 green-mesh 命名空间中。这些服务可作为 <name>.west-data-center.svc.red-mesh-imports.local 访问

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh
  namespace: green-mesh-system
spec:
  importRules:
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: west-data-center
      name: "*"
```

1.18.12.1. 创建 ImportedServiceSet

您可以创建一个 ImportedServiceSet 资源来显式声明您要导入到网格中的服务。

服务使用名称 <exported-name>.<exported-namespace>.svc.<ServiceMeshPeer.name>.remote 导入，它是一个"hidden"服务，仅在出口网关命名空间中可见，并与导出的服务主机名相关联。默认情况下，该服务将本地作为 <export-name>.<export-namespace>.<domainSuffix> 提供，其中 domainSuffix 是 svc.<ServiceMeshPeshPeer.name>-imports.local， 除非 importAsLocal 设置为 true，否则 domainSuffix 为 svc.cluster.local。如果 importAsLocal 设为 false，则会应用导入规则中的域后缀。您可以像网格中的任何其他服务一样对待本地导入。它通过出口网关自动路由，它将重定向到导出服务的远程名称。

前提条件

- 为网格联邦配置了集群和 ServiceMeshControlPlane。
- 具有 cluster-admin 角色的帐户。

注意

您可以配置用于导入的服务，即使它们尚未导出。当部署并导出与 ImportedServiceSet 中指定的值匹配的服务时，它会被自动导入。

流程

按照以下步骤，通过命令行创建 ImportedServiceSet。

1. 以具有 cluster-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS CLI。输入以下命令。然后在提示时输入您的用户名和密码。

```
$ oc login --username=<NAMEOFUSER> <API token> https://<HOSTNAME>:6443
```

2. 切换到安装 Service Mesh control plane 的项目，如 green-mesh-system。

```
$ oc project green-mesh-system
```

3. 根据以下示例创建一个 ImportedServiceSet 文件，其中 green-mesh 导入之前通过 red-mesh 导出的服务。

从 red-mesh 到 green-mesh 的 ImportedServiceSet 资源示例

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh
  namespace: green-mesh-system
spec:
  importRules:
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: info
      name: red-ratings
      alias:
        namespace: info
        name: ratings
```

4. 运行以下命令在 green-mesh-system 命名空间中上传并创建 ImportedServiceSet 资源。

```
$ oc create -n <ControlPlaneNamespace> -f <ImportedServiceSet.yaml>
```

例如：

```
$ oc create -n green-mesh-system -f import-from-red-mesh.yaml
```

5. 根据需要为联邦网格中的每个网格对等创建额外的 ImportedServiceSet 资源。

验证

- 运行以下命令，以验证服务是否已导入到 green-mesh 中：

```
$ oc get importedserviceset <PeerMeshImportedInto> -o yaml
```

使用 importedserviceset/red-mesh' object in the 'green-mesh-system 命名空间中的 status 部分验证从红色网格导出的服务是否已导入到绿色网格中

```
$ oc -n green-mesh-system get importedserviceset/red-mesh -o yaml
```

```
status:
  importedServices:
  - exportedName: red-ratings.info.svc.green-mesh-exports.local
    localService:
      hostname: ratings.info.svc.red-mesh-imports.local
      name: ratings
      namespace: info
  - exportedName: reviews.red-mesh-info.svc.green-mesh-exports.local
    localService:
      hostname: ""
      name: ""
      namespace: ""
```

在上例中，仅导入 ratings 服务，如 localService 下的填充字段所示。reviews 服务可用于导入，但目前并不导入，因为它与 ImportedServiceSet 对象中的任何 importRules 不匹配。

1.18.13. 为故障转移配置一个联邦网格

故障转移功能可以实现自动、无缝地切换到可靠的备份系统，例如切换到另一台服务器。如果是联邦网格，您可以在一个网格中配置服务，以便在另一个网格中切换到服务。

您可以通过在 ImportedServiceSet 资源中设置 importAsLocal 和 locality 设置来配置故障转移的联邦，然后配置 DestinationRule，将服务被配置为 ImportedServiceSet 中指定的本地性。

前提条件

- 两个或多个 Red Hat OpenShift Service on AWS 4.6 或更高版本的集群已进行了联网和联邦。
- 已为联邦网格中的每个网格中的网格对等点创建 ExportedServiceSet 资源。
- 已为联邦网格中的每个网格中的网格对等点创建 ImportedServiceSet 资源。
- 具有 cluster-admin 角色的帐户。

1.18.13.1. 为故障转移配置 ImportedServiceSet

管理员可以利用本地方式管理负载平衡，根据流量的来源和终止位置控制流量到端点的分布。这些本地设置使用任意标签指定，它以 {region}/{zone}/{sub-zone} 的形式指定了一个分层级结构的本地设置。

在本节示例中，green-mesh 位于 us-east 地区，而 red-mesh 位于 us-west 区域。

从 red-mesh 到 green-mesh 的ImportedServiceSet 资源示例

```
kind: ImportedServiceSet
```

```
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh #name of mesh that exported the service
  namespace: green-mesh-system #mesh namespace that service is being imported into
spec:
  importRules: # first matching rule is used
  # import ratings.info as ratings.bookinfo
  - type: NameSelector
    importAsLocal: true
    nameSelector:
      namespace: info
      name: ratings
      alias:
        # service will be imported as ratings.info.svc.red-mesh-imports.local
        namespace: info
        name: ratings
  #Locality within which imported services should be associated.
  locality:
    region: us-west
```

表 1.12. ImportedServiceLocality 字段表

名称	描述	类型
区域：	导入的服务所在的区域。	字符串
子区：	导入服务的子区(zone)位于其中。I Subzone 被指定，还必须指定 Zone。	字符串
zone：	导入的服务所在的区。如果指定了 Zone，还必须指定 Region。	字符串

流程

1. **以具有 cluster-admin 角色的用户身份登录 Red Hat OpenShift Service on AWS CLI，请输入以下命令：**

```
$ oc login --username=<NAMEOFUSER> <API token> https://<HOSTNAME>:6443
```

2. **进入到安装 Service Mesh control plane 的项目，请输入以下命令：**

```
$ oc project <smcp-system>
```

例如： green-mesh-system。

```
$ oc project green-mesh-system
```

3. 编辑 ImportedServiceSet 文件，其中 <ImportedServiceSet.yaml> 包含您要编辑的文件的完整路径，请输入以下命令：

```
$ oc edit -n <smcp-system> -f <ImportedServiceSet.yaml>
```

例如，如果要将从 red-mesh-system 导入到 green-mesh-system 的文件（如前面的 ImportedServiceSet 示例所示）。

```
$ oc edit -n green-mesh-system -f import-from-red-mesh.yaml
```

4. 修改该文件：

   a. 将 spec.importRules.importAsLocal 设置为 true。

   b. 将 spec.locality 设置为 region, zone, 或 subzone。

   c. 保存您的更改。

1.18.13.2. 为故障转移配置 DestinationRule

创建配置以下内容的 DestinationRule 资源：

- 服务的 Outlier 检测。需要此项才能使故障转移正常工作。特别是，它会配置 sidecar 代理，以获知服务端点处于不健康状态，并最终触发对下一个位置的故障转移。

- 各地区之间的故障转移策略。这样可确保除区域边界外的故障切换将具有可预见的。

流程

1. 以具有 cluster-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS CLI。输入以下命令。然后在提示时输入您的用户名和密码。

```
$ oc login --username=<NAMEOFUSER> <API token> https://<HOSTNAME>:6443
```

2. 切换到安装 Service Mesh control plane 的项目。

```
$ oc project <smcp-system>
```

例如：green-mesh-system。

```
$ oc project green-mesh-system
```

3. 根据以下示例创建一个 DestinationRule 文件，如果 green-mesh 不可用，则流量应从 us-east 区域中的 green-mesh 路由到 us-west 中的 red-mesh。

DestinationRule示例

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: default-failover
  namespace: info
spec:
  host: "ratings.info.svc.cluster.local"
  trafficPolicy:
    loadBalancer:
      localityLbSetting:
        enabled: true
        failover:
          - from: us-east
            to: us-west
    outlierDetection:
      consecutive5xxErrors: 3
      interval: 10s
      baseEjectionTime: 1m
```

4.

部署 DestinationRule，其中 <DestinationRule> 包含到您的文件的完整路径，请输入以下命令：

```
$ oc create -n <application namespace> -f <DestinationRule.yaml>
```

例如：

```
$ oc create -n info -f green-mesh-us-west-DestinationRule.yaml
```

1.18.14. 从联邦网格中删除服务

如果您需要从联邦网格中删除服务，例如，如果服务已过时或者已被其他服务替换，您可以这样做。

1.18.14.1. 从单个网格中删除服务

从不再应该访问该服务的网格对等点的 ImportedServiceSet 资源中删除服务条目。

1.18.14.2. 从整个联邦网格中删除服务

从拥有该服务的网格的 ExportedServiceSet 资源中删除服务条目。

1.18.15. 从联邦网格中删除网格

如果您需要从联邦中删除网格，您可以这样做。

1. 编辑删除的网格的 ServiceMeshControlPlane 资源，以删除所有对等网格的联邦入口网关。

2. 对于已删除网格的每个网格对等点，请执行以下操作：

   a. 删除链接两个网格的 ServiceMeshPeer 资源。

   b. 编辑 peer mesh 的 ServiceMeshControlPlane 资源，以删除服务已删除网格的出口网关。

1.19. 扩展

您可以使用 WebAsembly 扩展直接将新功能添加到 Red Hat OpenShift Service Mesh 代理中。这可让您从应用程序中移出更多常见的功能，并使用编译到 WebAssembly 字节代码的单一语言实现它们。

1.19.1. WebAssembly 模块概述

WebAsembly 模块可以在很多平台上运行，包括代理，并有广泛语言支持、快速执行以及沙盒安全模型。

Red Hat OpenShift Service Mesh 扩展是 Envoy HTTP Filters，为它们提供广泛的功能：

- 控制请求和响应的正文和标头。
- 对不在请求路径中的服务（如认证或策略检查）的带外 HTTP 请求。
- 用来相互通信的 sidechannel 数据存储和过滤器队列。

> **注意**
>
> 在创建新的 WebAsembly 扩展时，请使用 WasmPlugin API。ServiceMeshExtension API 在 Red Hat OpenShift Service Mesh 版本 2.2 中已弃用，并在 Red Hat OpenShift Service Mesh 版本 2.3 中删除。

编写 Red Hat OpenShift Service Mesh 扩展有两个部分：

1. 您必须使用提供 proxy-wasm API 的 SDK 编写扩展，并将其编译到 WebAssembly 模块。
2. 然后，您必须将模块打包到容器中。

支持的语言

您可以使用任何编译到 WebAssembly 字节码的语言来编写 Red Hat OpenShift Service Mesh 扩展，但以下语言具有公开 proxy-wasm API 的现有 SDK，以便直接使用它。

表 1.13. 支持的语言

语言	Maintainer	软件仓库
AssemblyScript	solo.io	solo-io/proxy-runtime
C++	proxy-wasm 团队（Istio 社区）	proxy-wasm/proxy-wasm-cpp-sdk
Go	tetrate.io	tetratelabs/proxy-wasm-go-sdk
Rust	proxy-wasm 团队（Istio 社区）	proxy-wasm/proxy-wasm-rust-sdk

1.19.2. WasmPlugin 容器格式

Istio 在其 Wasm Plugin 机制中支持 OCI（Open Container Initiative） 镜像。您可以将 Wasm 插件分发为容器镜像，您可以使用 spec.url 字段来引用容器 registry 位置。例如： quay.io/my-username/my-plugin:latest。

因为 WASM 模块的每个执行环境(runtime)都可以有特定于运行时的配置参数，因此 WASM 镜像由两个层组成：

- plugin.wasm （必需）- 内容层。这个层包含一个包含 WebAssembly 模块字节码的 .wasm 二进制文件，它由运行时加载。您必须将此文件命名为 plugin.wasm。
- runtime-config.json （可选）- 配置层。这个层包含一个 JSON 格式的字符串，用于描述目标运行时模块的元数据。根据目标运行时，配置层也可以包含其他数据。例如， WASM Envoy Filter 的配置包含过滤器上的 root_ids。

1.19.3. WasmPlugin API 参考

WasmPlugins API 提供了通过 WebAssembly 过滤器扩展 Istio 代理提供的功能的机制。

您可以部署多个 WasmPlugins。phase 和 priority 设置决定了执行顺序（作为 Envoy 的过滤器链的一部分），允许对用户提供的 WasmPlugin 和 Istio 的内部过滤器配置复杂交互。

在以下示例中，身份验证过滤器实施 OpenID 流，并使用 JSON Web Token(JWT)填充 Authorization 标头。Istio 身份验证会消耗此令牌并将其部署到入口网关。WasmPlugin 文件在代理 sidecar 文件系统中存在。请注意字段 url。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: openid-connect
  namespace: istio-ingress
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  url: file:///opt/filters/openid.wasm
  sha256: 1ef0c9a92b0420cf25f7fe5d481b231464bc88f486ca3b9c83ed5cc21d2f6210
  phase: AUTHN
  pluginConfig:
    openid_server: authn
    openid_realm: ingress
```

以下是相同的示例，但这一次使用 OCI 镜像而不是文件系统中的文件。记录 url、imagePullPolicy、imagePullSecret 和 imagePullSecret 字段。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: openid-connect
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  url: oci://private-registry:5000/openid-connect/openid:latest
  imagePullPolicy: IfNotPresent
  imagePullSecret: private-registry-pull-secret
  phase: AUTHN
  pluginConfig:
    openid_server: authn
    openid_realm: ingress
```

表 1.14. WasmPlugin 字段参考

字段	类型	描述	必需
spec.selector	WorkloadSelector	用于选择应该应用此插件配置的特定 pod/VM 集合的条件。如果省略，此配置将应用于同一命名空间中的所有工作负载实例。如果 config root 命名空间中存在 **WasmPlugin** 字段，它将应用于任何命名空间中的所有适用工作负载。	否
spec.url	字符串	Wasm 模块或 OCI 容器的 URL。如果没有方案，则默认为 **oci://**，代表 OCI 镜像。其他有效的方案是 **file://**（用于引用代理容器内的本地 .wasm 模块文件）；**http[s]://** 用于远程托管的 .wasm 模块文件 。	否
spec.sha256	字符串	用于验证 Wasm 模块或 OCI 容器的 SHA256 checksum。如果 **url** 字段已引用 SHA256（使用 **@sha256:** 表示法），它必须与此字段的值匹配。如果 tag 引用了 OCI 镜像，并且设置了此字段，则在拉取后将根据此字段的内容验证其校验和。	否
spec.imagePullPolicy	PullPolicy	获取 OCI 镜像时要应用的拉取行为。只有在通过标签而不是 SHA 引用镜像时才相关。默认为 **IfNotPresent** 值，除非在 **url** 字段中引用了 OCI 镜像并且使用了 **latest** 标签时，这种情况下默认值为 **Always**，镜像 K8s 的行为。如果 **url** 字段直接使用 **file://** 或 **http[s]://** 直接引用 Wasm 模块，则忽略设置。	否

字段	类型	描述	必需
spec.imagePullSecret	字符串	用于 OCI 镜像拉取的凭证。与 **WasmPlugin** 对象相同的命名空间中的 secret 名称，其中包含用于在拉取镜像时对 registry 进行身份验证的 pull secret。	否
spec.phase	PluginPhase	决定过滤器链中注入这个 **WasmPlugin** 对象的位置。	否
spec.priority	**int64**	决定有同一 **phase** 值的 **WasmPlugins** 对象的顺序。当多个 **WasmPlugins** 对象应用于同一阶段的同一工作负载时，它们将按优先级和降序应用。如果没有设置 **priority** 字段，或者两个具有相同值的 **WasmPlugins** 对象，则排序将从 **WasmPlugins** 对象的名称和命名空间决定。默认值为 **0**。	否
spec.pluginName	字符串	Envoy 配置中使用的插件名称。有些 Wasm 模块可能需要这个值来选择要执行的 Wasm 插件。	否
spec.pluginConfig	Struct	将要传递给插件的配置。	否
spec.pluginConfig.verificationKey	字符串	用于验证签名 OCI 镜像或 Wasm 模块的公钥。必须以 PEM 格式提供。	否

WorkloadSelector 对象指定用于确定过滤器是否可应用于代理的条件。匹配条件包括与代理关联的元数据、工作负载实例信息，如附加到 pod/VM 的标签，或代理在初始握手期间向 Istio 提供的任何其他信息。如果指定了多个条件，则所有条件都需要匹配才能选择工作负载实例。目前，只支持基于标签的选择机制。

表 1.15. WorkloadSelector

字段	类型	描述	必需
matchLabels	map<string, string>	指定应应用策略的特定 pod/VM 集合的一个或多个标签。标签搜索范围仅限于存在资源的配置命名空间。	是

PullPolicy 对象指定要在获取 OCI 镜像时应用的 pull 行为。

表 1.16. PullPolicy

值	描述
<empty>	默认值为 **IfNotPresent**，但使用标签 latest 的 OCI 镜像除外，其默认值为 **Always**。
IfNotPresent	如果在之前拉取了镜像的现有版本，则会使用它。如果本地没有镜像版本，我们将拉取最新版本。
Always	应用此插件时，始终拉取镜像的最新版本。

Struct 代表结构化数据值，由映射到动态输入的值的字段组成。在某些语言中，Struct 可能受到原生表示的支持。例如，在脚本语言中，JavaScript astruct 等脚本语言表示为对象。

表 1.17. Struct

字段	类型	描述
fields	map<string, Value>	动态输入的值的映射。

PluginPhase 指定将注入插件的过滤器链中的阶段。

表 1.18. PluginPhase

字段	描述
<empty>	control plane 决定插入插件的位置。这通常位于过滤器链的末尾，在路由器前面。如果插件独立于其他插件，则不要指定 PluginPhase。
AUTHN	在 Istio 身份验证过滤器前插入插件。

字段	描述
AUTHZ	在 Istio 授权过滤器和 Istio 身份验证过滤器后插入插件。
STATS	在 Istio stats 过滤器和 Istio 授权过滤器后插入插件。

1.19.3.1. 部署 WasmPlugin 资源

您可以使用 WasmPlugin 资源启用 Red Hat OpenShift Service Mesh 扩展。在本例中，istio-system 是 Service Mesh control plane 项目的名称。以下示例创建了一个 openid-connect 过滤器，它将执行 OpenID Connect 流来验证用户。

流程

1. 创建以下示例资源：

plugin.yaml 示例

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: openid-connect
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  url: oci://private-registry:5000/openid-connect/openid:latest
  imagePullPolicy: IfNotPresent
  imagePullSecret: private-registry-pull-secret
  phase: AUTHN
  pluginConfig:
    openid_server: authn
    openid_realm: ingress
```

2. 使用以下命令应用 plugin.yaml 文件：

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

1.19.4. ServiceMeshExtension 容器格式

您必须有包含 WebAssembly 模块字节码的 .wasm 文件，以及容器文件系统根中的 manifest.yaml 文件，以使您的容器镜像成为有效的扩展镜像。

注意

在创建新的 WebAsembly 扩展时，请使用 WasmPlugin API。ServiceMeshExtension API 在 Red Hat OpenShift Service Mesh 版本 2.2 中已弃用，并在 Red Hat OpenShift Service Mesh 版本 2.3 中删除。

manifest.yaml

```
schemaVersion: 1

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

表 1.19. manifest.yml 的字段参考

字段	描述	必需
schemaVersion	用于清单架构的版本。目前唯一可能的值是 **1**。	这个为必填字段。
name	扩展名。	这个字段只是元数据且目前没有使用。
description	扩展的描述。	这个字段只是元数据且目前没有使用。
version	扩展名的版本。	这个字段只是元数据且目前没有使用。
phase	扩展的默认执行阶段。	这个为必填字段。

字段	描述	必需
priority	扩展的默认优先级。	这个为必填字段。
module	容器文件系统的 root 到 WebAssembly 模块的相对路径。	这个为必填字段。

1.19.5. ServiceMeshExtension 参考

ServiceMeshExtension API 提供了通过 WebAssembly 过滤器扩展 Istio 代理提供的功能的机制。编写 WebAssembly 扩展有两个部分：

1. **使用提供 proxy-wasm API 的 SDK 编写扩展，并将其编译到 WebAssembly 模块。**

2. **将它打包到容器中。**

注意

在创建新的 WebAsembly 扩展时，请使用 WasmPlugin API。Red Hat OpenShift Service Mesh 2.2 中已弃用的 ServiceMeshExtension API 已在 Red Hat OpenShift Service Mesh 2.3 中删除。

表 1.20. ServiceMeshExtension 字段参考

字段	描述
metadata.namespace	**ServiceMeshExtension** 源的 **metadata.namespace** 项具有特殊的语义：如果与 Control Plane 命名空间相等，扩展将应用到 Service Mesh 中与 **workloadSelector** 值匹配的所有工作负载。当部署到任何其他 Mesh 命名空间时，它只适用于同一命名空间中的工作负载。
spec.workloadSelector	**spec.workloadSelector** 字段的语义与 Istio Gateway 资源 的 **spec.selector** 字段相同。它将根据其 Pod 标签匹配工作负载。如果没有指定 **workloadSelector** 值，扩展将应用到命名空间中的所有工作负载。
spec.config	这是一个结构化字段，将被移交给扩展名，其语义取决于您要部署的扩展名。

字段	描述
spec.image	指向包含扩展的镜像的容器镜像 URI。
spec.phase	该阶段根据现有 Istio 功能，如身份验证、授权和指标生成，决定过滤器链中的扩展是否被注入。有效值为：PreAuthN、PostAuthN、PreAuthZ、PostAuthZ、PreStats、PostStats。此字段默认为扩展名的 **manifest.yaml** 文件中设置的值，但可以被用户覆盖。
spec.priority	如果将具有相同 **spec.phase** 值的多个扩展应用到同一工作负载实例，则 **spec.priority** 值会决定执行顺序。优先级更高的扩展将首先执行。这允许相互依赖的扩展。此字段默认为扩展名的 **manifest.yaml** 文件中设置的值，但可以被用户覆盖。

1.19.5.1. 部署 ServiceMeshExtension 资源

您可以使用 ServiceMeshExtension 资源启用 Red Hat OpenShift Service Mesh 扩展。在本例中，istio-system 是 Service Mesh control plane 项目的名称。

注意

在创建新的 WebAsembly 扩展时，请使用 WasmPlugin API。ServiceMeshExtension API 在 Red Hat OpenShift Service Mesh 版本 2.2 中已弃用，并在 Red Hat OpenShift Service Mesh 版本 2.3 中删除。

有关使用 Rust SDK 构建的完整示例，请看标题附加过滤器。这是一个简单的过滤器，它会将一个或多个标头附加到 HTTP 响应中，其名称和值从扩展的 config 字段中获取。请参阅以下代码片段中的示例配置。

流程

1. 创建以下示例资源：

ServiceMeshExtension 资源 extension.yaml 示例

```
apiVersion: maistra.io/v1
```

```
kind: ServiceMeshExtension
metadata:
  name: header-append
  namespace: istio-system
spec:
  workloadSelector:
    labels:
      app: httpbin
  config:
    first-header: some-value
    another-header: another-value
  image: quay.io/maistra-dev/header-append-filter:2.1
  phase: PostAuthZ
  priority: 100
```

2. 使用以下命令应用 extension.yaml 文件：

```
$ oc apply -f <extension>.yaml
```

1.19.6. 从 ServiceMeshExtension 迁移到 WasmPlugin 资源

ServiceMeshExtension API 在 Red Hat OpenShift Service Mesh 版本 2.2 中已弃用，在 Red Hat OpenShift Service Mesh 版本 2.3 中删除。如果使用 ServiceMeshExtension API，则必须迁移到 WasmPlugin API 以继续使用 WebAssembly 扩展。

API 非常相似。迁移由两个步骤组成：

1. 重命名您的插件文件并更新模块打包。

2. 创建引用更新的容器镜像的 WasmPlugin 资源。

1.19.6.1. API 更改

新的 WasmPlugin API 与 ServiceMeshExtension 类似，但有一些区别，特别是在字段名称中：

表 1.21. ServiceMeshExtensions 和 WasmPlugin之间的字段变化

ServiceMeshExtension	WasmPlugin
spec.config	**spec.pluginConfig**
spec.workloadSelector	**spec.selector**
spec.image	**spec.url**
spec.phase 有效值： PreAuthN、PostAuthN、PreAuthZ、PostAuthZ、PreStats、PostStats	**spec.phase** 有效值： <empty>、AUTHN、AUTHZ、STATS

以下是如何将 ServiceMeshExtension 资源转换为 WasmPlugin 资源的示例。

ServiceMeshExtension 资源

```
apiVersion: maistra.io/v1
kind: ServiceMeshExtension
metadata:
  name: header-append
  namespace: istio-system
spec:
  workloadSelector:
    labels:
      app: httpbin
  config:
    first-header: some-value
    another-header: another-value
  image: quay.io/maistra-dev/header-append-filter:2.2
  phase: PostAuthZ
  priority: 100
```

新的 WasmPlugin 资源等同于上面的 ServiceMeshExtension

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: header-append
  namespace: istio-system
spec:
  selector:
```

```
  matchLabels:
    app: httpbin
url: oci://quay.io/maistra-dev/header-append-filter:2.2
phase: STATS
pluginConfig:
  first-header: some-value
  another-header: another-value
```

1.19.6.2. 容器镜像格式更改

新的 WasmPlugin 容器镜像格式与 ServiceMeshExtensions 类似，其区别如下：

- ServiceMeshExtension 容器格式需要在容器文件系统的根目录中名为 manifest.yaml 的元数据文件。WasmPlugin 容器格式不需要 manifest.yaml 文件。
- .wasm 文件（实际插件）以前可能具有任何文件名，现在必须命名为 plugin.wasm，且必须位于容器文件系统的根目录中。

1.19.6.3. 迁移到 WasmPlugin 资源

要将 WebAsembly 扩展从 ServiceMeshExtension API 升级到 WasmPlugin API，您可以重命名您的插件文件。

前提条件

- ServiceMeshControlPlane 升级到 2.2 或更高版本。

流程

1. 更新您的容器镜像。如果插件已在容器内的 /plugin.wasm 中，则跳至下一步。如果没有：
   a. 确保插件文件名为 plugin.wasm。您需要将扩展文件命名为 plugin.wasm。
   b. 确保插件文件位于 root (/)目录中。您必须将扩展文件存储在容器文件系统的根目录

中。

c. 重新构建容器镜像并将其推送到容器 registry。

2. 删除 ServiceMeshExtension 资源，并创建一个 WasmPlugin 资源来引用您构建的新容器镜像。

1.20. 使用 3SCALE WEBASSEMBLY 模块

注意

threescale-wasm-auth 模块在 3scale API 管理 2.11 或更高版本与 Red Hat OpenShift Service Mesh 2.1.0 或更高版本的集成时运行。

threescale-wasm-auth 模块是一个 WebAssembly 模块，它使用一组接口，称为应用二进制接口 (*ABI*)。这由 *Proxy-WASM* 规范定义，驱动实施 ABI 的任何软件，以便它能够授权 HTTP 请求针对 3scale。

作为 ABI 规范，Proxy-WASM 定义名为 *host* 的软件与另一个命名 *模块*、*程序*或*扩展* 名之间的交互。主机公开一组供模块用于执行任务的服务，本例中用于处理代理请求。

主机环境由 WebAsembly 虚拟机组成，与软件进行交互，本例中为 HTTP 代理。

模块本身与外部世界隔离运行，除了它在虚拟机上运行的指令和 Proxy-WASM 指定的 ABI 除外。这是为软件提供扩展点的安全方法：扩展只能以明确定义的方式与虚拟机和主机进行交互。该交互提供了一种计算模型和与代理旨在具有的外部的连接。

1.20.1. 兼容性

threescale-wasm-auth 模块设计为与 *Proxy-WASM ABI* 规范的所有实施完全兼容。然而，此时只经过了全面测试才能与 Envoy 反向代理配合使用。

1.20.2. 使用作为独立模块

由于其自包含的设计，可以将此模块配置为与独立于 Service Mesh 的 Proxy-WASM 代理以及 3scale Istio 适配器部署配合使用。

1.20.3. 配置 threescale-wasm-auth 模块

Red Hat OpenShift Service on AWS 上的集群管理员可以配置 threescale-wasm-auth 模块，通过应用程序二进制接口(ABI)授权 HTTP 请求进行 3scale API 管理。ABI 定义主机和模块之间的交互，公开主机服务，并允许您使用模块来处理代理请求。

1.20.3.1. WasmPlugin API 扩展

Service Mesh 提供了一个自定义资源定义，用于为 sidecar 代理（称为 WasmPlugin ）指定并应用 Proxy-WASM 扩展。Service Mesh 将此自定义资源应用到需要使用 3scale 管理 HTTP API 的工作负载集合。

注意

配置 WebAssembly 扩展目前是一个手动过程。以后的发行版本中将提供对 3scale 系统获取服务配置的支持。

前提条件

- 识别要应用此模块的 Service Mesh 部署上的 Kubernetes 工作负载和命名空间。

- 您必须有一个 3scale 租户帐户。请参阅 SaaS 或 3scale 2.11 On-Premises，其中定义了匹配的服务和相关应用程序和指标。

  - 以下示例是 threescale-wasm-auth 模块的自定义资源的 YAML 格式。本例指的是 Service Mesh 的上游 Maistra 版本 WasmPlugin API。您必须声明部署 threescale-wasm-auth 模块的命名空间，以及一个选择器来标识模块要应用到的应用程序集合：

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
  namespace: <info> 1
spec:
  selector: 2
```

```
  labels:
    app: <product_page>
pluginConfig: <yaml_configuration>
url: oci://registry.redhat.io/3scale-amp2/3scale-auth-wasm-rhel8:0.0.3
phase: AUTHZ
priority: 100
```

1. namespace。

2. 选择器。

- spec.pluginConfig 字段取决于模块配置，且没有在上例中填充。这个示例使用 <yaml_configuration> 占位符值。您可以使用此自定义资源示例的格式。
  - spec.pluginConfig 字段因应用程序而异。所有其他字段在该自定义资源的多个实例之间都存在。例如：
    - url：仅在部署较新版本的模块时更改。
    - phase ：保留相同的模块，因为此模块需要在代理完成任何本地授权后调用，如验证 OpenID Connect(OIDC)令牌。
- 在 spec.pluginConfig 和其余自定义资源中配置了模块后，使用 oc apply 命令应用它：

```
$ oc apply -f threescale-wasm-auth-info.yaml
```

其他资源

- 从 ServiceMeshExtension 迁移到 WasmPlugin 资源
- 自定义资源

1.20.4. 应用 3scale 外部 ServiceEntry 对象

要让 threescale-wasm-auth 模块授权针对 3scale 的请求，该模块必须有权访问 3scale 服务。您可以通过应用外部 ServiceEntry 对象和用于 TLS 配置的对应的 DestinationRule 对象来在 Red Hat OpenShift Service Mesh 中执行此操作，以使用 HTTPS 协议。

自定义资源(CR)设置服务条目和目的地规则，以便从 Service Mesh 内安全访问 3scale 托管(SaaS)，用于服务管理 API 和帐户管理 API 的后端和系统组件。Service Management API 接收每个请求的授权状态查询。帐户管理 API 为您的服务提供 API 管理配置设置。

流程

1. 对集群的由 3scale 托管的后端应用以下外部 ServiceEntry CR 和相关的 DestinationRule CR：

   a. 将 ServiceEntry CR 添加到名为 service-entry-threescale-saas-backend.yml 的文件：

      ServiceEntry CR

```
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: service-entry-threescale-saas-backend
spec:
  hosts:
  - su1.3scale.net
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
```

   b. 将 DestinationRule CR 添加到名为 destination-rule-threescale-saas-backend.yml 的文件：

DestinationRule CR

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: destination-rule-threescale-saas-backend
spec:
  host: su1.3scale.net
  trafficPolicy:
    tls:
      mode: SIMPLE
      sni: su1.3scale.net
```

c. 运行以下命令，为 3scale 托管后端应用并保存外部 ServiceEntry CR：

```
$ oc apply -f service-entry-threescale-saas-backend.yml
```

d. 运行以下命令，为 3scale Hosted 后端应用并保存外部 DestinationRule CR：

```
$ oc apply -f destination-rule-threescale-saas-backend.yml
```

2. 对集群的由 3scale 托管的系统应用以下外部 ServiceEntry CR 和相关的 DestinationRule CR：

a. 将 ServiceEntry CR 添加到名为 service-entry-threescale-saas-system.yml 的文件：

ServiceEntry CR

```
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: service-entry-threescale-saas-system
spec:
  hosts:
  - multitenant.3scale.net
```

```
ports:
- number: 443
  name: https
  protocol: HTTPS
location: MESH_EXTERNAL
resolution: DNS
```

b. 将 DestinationRule CR 添加到名为 destination-rule-threescale-saas-system.yml 的文件：

DestinationRule CR

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: destination-rule-threescale-saas-system
spec:
  host: multitenant.3scale.net
  trafficPolicy:
    tls:
      mode: SIMPLE
      sni: multitenant.3scale.net
```

c. 运行以下命令，为 3scale 托管系统应用并保存外部 ServiceEntry CR：

```
$ oc apply -f service-entry-threescale-saas-system.yml
```

d. 运行以下命令，为 3scale Hosted 系统应用并保存外部 DestinationRule CR：

```
$ oc apply -f <destination-rule-threescale-saas-system.yml>
```

或者，您可以部署一个mesh 3scale 服务。要部署 in-mesh 3scale 服务，请通过部署 3scale 并链接到部署来更改 CR 中服务的位置。

其他资源

- 服务条目和目的地规则文档

1.20.5. 3scale WebAssembly 模块配置

WasmPlugin 自定义资源规格提供了 Proxy-WASM 模块从中读取的配置。

该 spec 嵌入主机中，并由 Proxy-WASM 模块读取。通常，配置采用要解析的模块的 JSON 文件格式，但 WasmPlugin 资源可以将 spec 值解释为 YAML，并将其转换为 JSON 以供模块使用。

如果您在独立模式中使用 Proxy-WASM 模块，则必须使用 JSON 格式编写配置。使用 JSON 格式意味着在 host 配置文件中根据需要使用转义和引用，如 Envoy。当您将 WebAssembly 模块与 WasmPlugin 资源搭配使用时，配置采用 YAML 格式。在这种情况下，无效的配置会强制模块根据其 JSON 表示将诊断显示到 sidecar 的日志记录流。

重要

EnvoyFilter 自定义资源不是受支持的 API，虽然可在 3scale Istio 适配器或 Service Mesh 版本中使用。不建议使用 EnvoyFilter 自定义资源。使用 WasmPlugin API 而不是 EnvoyFilter 自定义资源。如果需要使用 EnvoyFilter 自定义资源，则必须以 JSON 格式指定 spec。

1.20.5.1. 配置 3scale WebAssembly 模块

3scale WebAssembly 模块配置的架构取决于 3scale 帐户和授权服务，以及要处理的服务列表。

前提条件

在所有情形中，先决条件都是一组最低必填字段：

- 对于 3scale 帐户和授权服务：backend-listener URL。
- 要处理的服务列表：服务 ID 和至少一个凭据查找方法以及查找方法。

- 您将找到处理 userkey、以及带有 appkey 的 appid，以及 OpenID Connect(OIDC)模式的示例。

- WebAssembly 模块使用您在静态配置中指定的设置。例如，如果您向模块中添加映射规则配置，它将始终适用，即使 3scale 管理门户没有这样的映射规则。其余的 WasmPlugin 资源围绕 spec.pluginConfig YAML 条目存在。

1.20.5.2. 3scale WebAssembly 模块 api 对象

3scale WebAssembly 模块的 api 顶级字符串定义模块要使用的配置版本。

注意

api 对象的不存在或不受支持的版本会导致 3scale WebAssembly 模块无法正常运行。

api 顶级字符串示例

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
  namespace: <info>
spec:
  pluginConfig:
    api: v1
# ...
```

api 条目定义配置的其余值。唯一接受的值是 v1。破坏与当前配置兼容性或需要更多使用 v1 的模块无法处理的逻辑的新设置将需要不同的值。

1.20.5.3. 3scale WebAssembly 模块系统对象

system 顶级对象指定如何访问特定帐户的 3scale 帐户管理 API。upstream 字段是对象最重要的部

分。upstream 对象是可选的，但建议使用，除非您要为 3scale WebAssembly 模块提供完全静态的配置，如果您不想提供到 3scale 的 *system* 组件的连接，则此选项是一个选项。

当您在 system 对象之外提供静态配置对象时，静态配置对象始终优先。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    system:
      name: <saas_porta>
      upstream: <object>
      token: <my_account_token>
      ttl: 300
# ...
```

表 1.22. system 对象字段

名称	描述	必需
name	3scale 服务的标识符，目前没有在别处引用。	选填
upstream	要联系的网络主机的详细信息。**upstream** 代表 3scale 帐户管理 API 主机，称为 system。	是
token	具有读取权限的 3scale 个人访问令牌。	是
ttl	在尝试获取新更改之前，将从此主机检索到的配置视为有效的最少秒数。默认为 600 秒（10 分钟）。**注意：**没有最大值，但模块通常会在此 TTL apsaps 之后合理时间段内获取任何配置。	选填

1.20.5.4. 3scale WebAssembly 模块上游对象

upstream 对象描述代理可以对其执行调用的外部主机。

```
apiVersion: maistra.io/v1
upstream:
  name: outbound|443||multitenant.3scale.net
```

```
  url: "https://myaccount-admin.3scale.net/"
  timeout: 5000
# ...
```

表 1.23. upstream 对象字段

名称	描述	必需
name	name 不是自由格式的标识符。它是外部主机的标识符，如代理配置中所定义。对于独立 Envoy 配置，它会映射到一个集群的名称，在其他代理中也称为 **上游（upstream）**。**注：** 这个字段的值，因为 Service Mesh 和 3scale Istio 适配器 control plane 根据使用竖线(|)作为多个字段分隔符的格式来配置名称。对于此集成，请始终使用格式：**outbound|<port>||<hostname>**。	是
url	用于访问所描述服务的完整 URL。除非被方案所暗示，否则您必须包含 TCP 端口。	是
Timeout（超时）	超时时间（毫秒），使得响应时间超过响应时间的连接将被视为错误。默认值为 1000 秒。	选填

1.20.5.5. 3scale WebAssembly 模块后端对象

backend 顶级对象指定如何访问 3scale Service Management API 来授权和报告 HTTP 请求。此服务由 3scale 的 *Backend* 组件提供。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
# ...
    backend:
      name: backend
      upstream: <object>
# ...
```

表 1.24. backend 对象字段

名称	描述	必需
name	3scale 后端的标识符，目前没有在别处引用。	选填
upstream	要联系的网络主机的详细信息。这必须引用 3scale 帐户管理 API 主机，即已知系统。	是。最重要和必填字段。

1.20.5.6. 3scale WebAssembly 模块服务对象

services 顶级对象指定由 **module** 的特定实例处理哪些服务标识符。

由于帐户具有多个服务，您必须指定处理哪些服务。其余的配置会围绕如何配置服务。

services 字段是必需的。它是必须至少包含一个服务的数组，才可使用。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
# ...
    services:
    - id: "2555417834789"
      token: service_token
      authorities:
        - "*.app"
        - 0.0.0.0
        - "0.0.0.0:8443"
      credentials: <object>
      mapping_rules: <object>
# ...
```

services 数组中的每个元素代表 3scale 服务。

表 1.25. **services** 对象字段

名称	描述	必需
ID	此 3scale 服务的标识符，目前没有在别处引用。	是

名称	描述	必需
token	此 **token** 可以在您的系统中的服务的代理配置中找到，也可以使用以下 **curl** 命令从系统检索它： **curl https://<system_host>/admin/api/services/<service_id>/proxy/configs/production/latest.json?access_token=<access_token>" | jq '.proxy_config.content.backend_authentication_value**	选填
authorities	一个字符串数组，每个字符串代表要匹配的 *URL* 的*颁发机构*。这些字符串接受支持星号(*)加号(+)和问号(?)匹配器的 glob 模式。	是
credentials	定义要查找和在哪里查找的凭据的对象。	是
mapping_rules	代表要命中映射规则和 3scale 方法的一组对象。	选填

1.20.5.7. 3scale WebAssembly 模块凭证对象

credentials 对象是 **service** 对象的组件。**credentials** 指定要查找的凭证类型，以及执行此操作的步骤。

所有字段均为可选，但您必须至少指定一个 **user_key** 或 **app_id**。指定每个凭据的顺序无关紧要，因为它由模块预先建立。仅指定每个凭证的一个实例。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
# ...
    services:
    - credentials:
        user_key: <array_of_lookup_queries>
        app_id: <array_of_lookup_queries>
        app_key: <array_of_lookup_queries>
# ...
```

表 1.26. credentials 对象字段

名称	描述	必需
user_key	这是一组查询，用于定义 3scale 用户密钥。用户密钥通常称为 API 密钥。	选填
app_id	这是一组查询，用于定义 3scale 应用标识符。应用程序标识符由 3scale 提供，或使用 Red Hat Single Sign-On (RH-SSO) 或 OpenID Connect(OIDC)等身份提供程序来提供。此处指定的查找查询的解析（只要成功并解析为两个值），它会设置 **app_id** 和 **app_key**。	选填
app_key	这是一组用于定义 3scale 应用键的查询。没有解析的 **app_id** 的应用程序密钥是无用的，因此仅在指定 **app_id** 时指定此字段。	选填

1.20.5.8. 3scale WebAssembly 模块查找查询

lookup query 对象是 **credentials** 对象中任何字段的一部分。它指定如何查找和处理给定凭证字段。评估之后，成功解析意味着找到一个或多个值。失败的解决方案意味着没有找到任何值。

lookup queries 的数组描述了一个短电路或关系：成功解析其中一个查询会停止评估任何剩余查询，并将值或值分配到指定的凭证类型。数组中的每个查询相互独立。

lookup queries 由单个字段（一个源对象）组成，它可以是多个源类型之一。请参见以下示例：

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
# ...
    services:
    - credentials:
        user_key:
          - <source_type>: <object>
          - <source_type>: <object>
# ...
        app_id:
```

```
      - <source_type>: <object>
  # ...
    app_key:
      - <source_type>: <object>
  # ...
```

1.20.5.9. 3scale WebAssembly 模块源对象

source 对象作为任何 credentials 对象字段中的源数组的一部分存在。对象字段名称，称为 source 类型代表以下任意一个：

- header ：查找查询接收 HTTP 请求标头作为输入。
- query_string ：lookup query 接收 URL 查询字符串参数作为输入。
- filter：lookup query 接收过滤器元数据作为输入。

所有 source 类型对象至少具有以下两个字段：

表 1.27. source 类型对象字段

名称	描述	必需
keys	一个字符串数组，各自对应一个 **key**，引用输入数据中找到的条目。	是
ops	用于执行 **key** 项匹配的操作数组。该数组是操作在下一个操作上接收输入并生成输出的管道。如果 **operation** 无法提供一个输出将会被解析为 **lookup query** 失败。操作的管道顺序决定了评估顺序。	选填

filter 字段名称具有所需的 path 条目，用于显示用于查找数据的元数据中的路径。

当 key 与输入数据匹配时，不会评估其余的密钥，而且源解析算法会跳转到执行指定的操作（ops），如果存在。如果没有指定 ops，则返回匹配 key 的结果值（若有）。

Operations 提供了一种方式，用于您在第一阶段查找 key 后为输入指定某些条件和转换。当您需要转换、解码和断言属性时，请使用 Operations，但它们不提供成熟的语言来满足所有需求并缺少 *Turing-completeness*。

存储 operations 输出的堆栈。评估时，lookup query 通过在堆栈的底部分配值或值来完成，具体取决于凭据使用的值。

1.20.5.10. 3scale WebAssembly 模块操作对象

属于特定 source type 的 ops 数组中的每个元素都是 operation 对象，可以应用转换到值或执行测试。用于此类对象的字段名称是 operation 本身的名称，任何值都是 operation 的参数，可以是结构对象，例如，带有字段和值、列表或字符串的映射。

大多数 operation 都参与一个或多个输入，产生一个或多个输出。当它们消耗输入或生成输出时，它们与一个堆栈相关：操作消耗的每个值都从堆栈中弹出，最初填充任何 source 匹配。它们输出的值将推送到堆栈。其他 operations 没有使用或生成的输出不是声明的特定属性，但您检查值的堆栈。

注意

完成解析后，下一步获取的值，例如将值分配给 app_id、app_key 或 user_key，取自堆栈的底部值。

有几个不同的 operations 类别：

- decode ：通过解码来转换输入值，使其获得不同的格式。
- string ：这取字符串值作为输入，并对字符串执行转换和检查。
- stack ：它们取输入中的一组值，执行多个堆栈转换，以及堆栈中特定位置的选择。
- check ：这声明了以没有副作用的方式处理一组操作的属性。

- control ：它们执行允许修改评估流的操作。
- format ：它解析输入值的格式特定结构，并在其中查找值。

所有操作都由名称标识符以字符串形式指定。

其他资源

- 可用的操作

1.20.5.11. 3scale WebAssembly 模块 mapping_rules 对象

mapping_rules 对象是 service 对象的一部分。它指定一组 REST 路径模式和相关 3scale 指标，并在模式匹配时指定要使用的递增数。

如果 system 顶级对象中没有提供动态配置，则需要该值。如果对象在 system 顶级条目外提供，则首先评估 mapping_rules 对象。

mapping_rules 是一个数组对象。该数组的每个元素都是 mapping_rule 对象。传入请求上评估的匹配映射规则提供了一组 3scale methods，用于授权并向 *APIManager* 报告。当多个匹配规则指代相同的 methods 时，调用 3scale 时会有一个 deltas 的总结。例如，如果两个规则使用 deltas 1 和 3 将 *Hits* 方法增加两次，则报告至 3scale 的 Hits 的单一方法条目的 delta 为 4。

1.20.5.12. 3scale WebAssembly 模块 mapping_rule 对象

mapping_rule 对象是 mapping_rules 对象中的数组的一部分。

mapping_rule 对象字段指定以下信息：

- 要匹配的 *HTTP 请求方法*。

- 匹配路径的模式.

- 要报告的 3scale 方法以及要报告的数量。指定字段的顺序决定了评估顺序。

表 1.28. mapping_rule 对象字段

名称	描述	必需
方法	指定代表 HTTP 请求方法的字符串，也称为 verb。接受的值与接受的 HTTP 方法名称之一匹配，不区分大小写。任何方法都匹配的特殊值。	是
pattern	与 HTTP 请求的 URI 路径组件匹配的模式。此模式遵循与 3scale 中记录的相同语法。它允许使用大括号（如 **{this}**）之间的任意字符序列使用通配符（使用星号(*)字符）。	是
usages	**usage** 对象列表。当规则匹配时，所有带有其 **deltas** 的方法都会添加到发送到 3scale 的方法列表中，以进行授权和报告。 使用以下必填字段嵌入 **usages** 对象 ： • **name**: 要报告的 **method** 系统名。 • **delta**: **method** 增加的数量。	是
last	当成功与此规则匹配，是否应停止评估更多映射规则。	可选布尔值。默认值为 **false**

以下示例独立于 3scale 中方法之间的现有层次结构。也就是说，在 3scale 侧运行的任何内容都不会受到影响。例如，*Hits* 指标可以是全部的父项，因此它存储了 4 个命中，因为授权请求中的所有报告方法总和，并调用 3scale Authrep API 端点。

以下示例使用到匹配所有规则的路径 /products/1/sold 的 GET 请求。

mapping_rules GET 请求示例

```yaml
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
# ...
    mapping_rules:
      - method: GET
        pattern: /
        usages:
          - name: hits
            delta: 1
      - method: GET
        pattern: /products/
        usages:
          - name: products
            delta: 1
      - method: ANY
        pattern: /products/{id}/sold
        usages:
          - name: sales
            delta: 1
          - name: products
            delta: 1
# ...
```

所有 usages 都会添加到模块执行的请求中使用用量数据 3scale，如下所示：

- 命中：1
- 产品：2 个
- 销售：1

1.20.6. 凭证用例的 3scale WebAssembly 模块示例

您将花费大部分时间应用配置步骤，在请求您的服务中获取凭证。

以下是 credentials 示例，您可以对其进行修改以符合特定用例的要求。

您可以组合使用它们，尽管当您指定多个源对象和自己的 lookup queries 时，会按照顺序对它们进行评估，直到其中一个成功解析为止。

1.20.6.1. 查询字符串参数中的 API 键 (user_key)

以下示例在查询字符串参数或相同名称的标头中查找 user_key ：

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
# ...
    services:
# ...
      credentials:
        user_key:
          - query_string:
              keys:
                - <user_key>
          - header:
              keys:
                - <user_key>
# ...
```

1.20.6.2. 应用程序 ID 和密钥

以下示例在查询或标头中查找 app_key 和 app_id 凭据。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
# ...
    services:
# ...
      credentials:
        app_id:
          - query_string:
```

```
          keys:
            - <app_id>
        - header:
          keys:
            - <app_id>
      app_key:
        - query_string:
          keys:
            - <app_key>
        - header:
          keys:
            - <app_key>
# ...
```

1.20.6.3. 授权标头

请求在 authorization 标头中包含 app_id 和 app_key。如果末尾至少输出了一个或两个值，您可以分配 app_key。

如果末尾输出了一两个或两个，此处的解决方法将分配 app_key。

authorization 标头使用授权类型指定值，其值编码为 Base64。这意味着，您可以通过空格字符来划分值，取第二个输出，然后使用冒号(:)作为分隔符再次分割它。例如，如果您使用这种格式 app_id:app_key，则标头类似以下示例 credential ：

```
aladdin:opensesame:  Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
```

您必须使用小写标头字段名称，如下例所示：

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
# ...
    services:
# ...
      credentials:
        app_id:
          - header:
            keys:
              - authorization
            ops:
              - split:
                  separator: " "
                  max: 2
```

```
            - length:
                min: 2
            - drop:
                head: 1
            - base64_urlsafe
            - split:
                max: 2
      app_key:
        - header:
            keys:
              - app_key
# ...
```

以上用例示例查看 authorization 标头：

1. 它接受字符串值并通过空格分割，检查它是否至少生成两个 credential 类型和 credential 本身，然后丢弃 credential 类型。

2. 然后，它会解码包含所需数据的第二个值，并使用冒号(:)字符进行拆分，使其具有一个包含 app_id 的操作堆栈，然后解码 app_key （若存在）。

   a. 如果授权标头中不存在 app_key，则将检查其特定源，例如本例中带有键 app_key 的标头。

3. 要向 credentials 添加额外条件，允许 Basic 授权，其中 app_id 是 aladdin 或 admin，或者任何 app_id 长度至少为 8 个字符。

4. app_key 必须包含一个值，并且至少具有 64 个字符，如下例所示：

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
# ...
    services:
# ...
      credentials:
        app_id:
          - header:
              keys:
                - authorization
              ops:
```

```
          - split:
              separator: " "
              max: 2
          - length:
              min: 2
          - reverse
          - glob:
            - Basic
          - drop:
              tail: 1
          - base64_urlsafe
          - split:
              max: 2
           - test:
              if:
                length:
                  min: 2
             then:
                - strlen:
                    max: 63
                - or:
                    - strlen:
                        min: 1
                    - drop:
                        tail: 1
          - assert:
            - and:
              - reverse
              - or:
                - strlen:
                    min: 8
                - glob:
                  - aladdin
                  - admin
# ...
```

5. 选取 authorization 标头值后，您可以通过淘汰堆栈来获取 Basic credential 类型，使类型放置在顶部。

6. 在其上运行通配匹配。验证凭据并且凭据被解码和分割后，您将获得堆栈底部的 app_id，还可能获得顶部的 app_key。

7. 运行 测试： 如果堆栈中有两个值，表示已获取 app_key。

   a. 确保字符串长度介于 1 到 63 之间，包括 app_id 和 app_key。如果密钥的长度为零，则将其丢弃，并像不存在密钥一样继续。如果只有一个 app_id 且没有 app_key，则缺少的其他分支表示测试和评估成功。

assert，最后一个操作表示它使它进入堆栈没有副作用。然后您可以修改堆栈：

1. 颠倒堆栈，使 app_id 位于顶部。

   a. 无论是否存在 app_key，取代堆栈可确保 app_id 处于顶级。

2. 使用 and 在测试期间保留堆栈的内容。

   然后使用以下可能性之一：

   - 确保 app_id 的字符串长度至少为 8。
   - 确保 app_id 与 aladdin 或 admin 匹配。

1.20.6.4. OpenID Connect(OIDC)用例

对于 Service Mesh 和 3scale Istio 适配器，您必须部署一个 RequestAuthentication，如下例所示，填入您自己的工作负载数据和 jwtRules:

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: info
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
```

应用 RequestAuthentication 时，它会使用原生插件配置 Envoy 以验证 JWT 令牌。代理会在运行模块前验证所有内容，因此任何失败的请求都不会将其发送到 3scale WebAssembly 模块。

验证 JWT 令牌时，代理将其内容存储在内部元数据对象中，其键取决于插件的具体配置。通过这个用例，您可以通过包含未知密钥名称的单一条目来查找结构对象。

OIDC 的 3scale app_id 与 OAuth client_id 匹配。这可在 JWT 令牌的 azp 或 aud 字段中找到。

要从 Envoy 的原生 JWT 身份验证过滤器获取 app_id 字段，请参阅以下示例：

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
# ...
    services:
# ...
      credentials:
        app_id:
          - filter:
              path:
                - envoy.filters.http.jwt_authn
                - "0"
              keys:
                - azp
                - aud
              ops:
                - take:
                    head: 1
# ...
```

示例指示模块使用 filter 源类型从 Envoy特定的 JWT 身份验证原生插件中查找对象的过滤器元数据。此插件包含 JWT 令牌，作为具有单个条目和预配置名称的结构对象的一部分。使用 0 指定您将仅访问单个条目。

生成值是一个结构，您要解析以下两个字段：

- azp ：找到 app_id 的值。
- aud: 也可以找到这个信息的值。

该操作可确保仅保留一个值进行分配。

1.20.6.5. 从标头中选取 JWT 令牌

一些设置可能具有 JWT 令牌的验证流程，验证令牌可通过 JSON 格式的标头访问此模块。

要获得 app_id，请参阅以下示例：

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
# ...
    services:
# ...
      credentials:
        app_id:
          - header:
              keys:
                - x-jwt-payload
              ops:
                - base64_urlsafe
                - json:
                  - keys:
                    - azp
                    - aud
                - take:
                    head: 1
# ,,,
```

1.20.7. 3scale WebAssembly 模块最小工作配置

以下是 3scale WebAssembly 模块最小工作配置的示例：您可以复制并粘贴此内容，并编辑它以便使用自己的配置。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  url: oci://registry.redhat.io/3scale-amp2/3scale-auth-wasm-rhel8:0.0.3
  imagePullSecret: <optional_pull_secret_resource>
  phase: AUTHZ
  priority: 100
  selector:
```

```
    labels:
      app: <product_page>
  pluginConfig:
    api: v1
    system:
      name: <system_name>
      upstream:
        name: outbound|443||multitenant.3scale.net
        url: https://istiodevel-admin.3scale.net/
        timeout: 5000
      token: <token>
    backend:
      name: <backend_name>
      upstream:
        name: outbound|443||su1.3scale.net
        url: https://su1.3scale.net/
        timeout: 5000
      extensions:
      - no_body
    services:
    - id: '2555417834780'
      authorities:
      - "*"
      credentials:
        user_key:
          - query_string:
              keys:
                - <user_key>
          - header:
              keys:
                - <user_key>
        app_id:
          - query_string:
              keys:
                - <app_id>
          - header:
              keys:
                - <app_id>
        app_key:
          - query_string:
              keys:
                - <app_key>
          - header:
              keys:
                - <app_key>
```

1.21. 使用 3SCALE ISTIO 适配器

3scale Istio 适配器是一个可选适配器，允许您在 Red Hat OpenShift Service Mesh 中标记运行的服务，并将该服务与 3scale API 管理解决方案集成。Red Hat OpenShift Service Mesh 不需要该适配器。

重要

您只能在 Red Hat OpenShift Service Mesh 版本 2.0 及以下中使用 3scale Istio 适配器。Mixer 组件在版本 2.0 中已弃用，并在版本 2.1 中删除。对于 Red Hat OpenShift Service Mesh 2.1.0 及之后的版本，您应该使用 3scale WebAsembly 模块。

如果要使用 3scale Istio 适配器启用 3scale 后端缓存，还必须启用 Mixer 策略和 Mixer 遥测。请参阅部署 Red Hat OpenShift Service Mesh control plane。

1.21.1. 将 3scale 适配器与 Red Hat OpenShift Service Mesh 集成

您可以使用这些示例来配置对使用 3scale Istio 适配器的服务的请求。

先决条件

- Red Hat OpenShift Service Mesh 版本 2.x
- 一个有效的 3scale 帐户（SaaS 或 3scale 2.9 On-Premises）
- 启用后端缓存需要 3scale 2.9 或更高版本
- Red Hat OpenShift Service Mesh 的先决条件
- 启用了 Mixer 强制功能。“更新 Mixer 策略强制”部分提供了检查当前 Mixer 策略实施状态和启用策略强制状态的信息。
- 如果您使用混合器插件，则必须启用混合器策略和遥测。
    - 升级时，您需要正确配置 Service Mesh Control Plane（SMCP）。

注意

要配置 3scale Istio 适配器，请参考“Red Hat OpenShift Service Mesh 自定义资源”来获得在自定义资源文件中添加适配器参数的说明。

注意

请特别注意 kind: handler 资源。您必须使用 3scale 帐户凭证更新它。您可以选择将 service_id 添加到处理程序，但这仅用于向后兼容性，因为它会使处理程序仅对 3scale 帐户中的一个服务有用。如果将 service_id 添加到处理程序，则为其他服务启用 3scale 需要使用不同的 service_ids 创建更多处理程序。

按照以下步骤，每个 3scale 帐户使用一个处理器：

流程

1. 为您的 3scale 帐户创建一个处理程序，并指定您的帐户凭证。省略任何服务标识符。

```
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"
```

   您可以选择在 *params* 部分提供一个 backend_url 字段来覆盖 3scale 配置提供的 URL。如果适配器与 3scale 内部实例在同一集群中运行，且您希望利用内部集群 DNS，这可能很有用。

2. 编辑或修补属于 3scale 帐户的所有服务的 Deployment 资源，如下所示：

   a. 添加 "service-mesh.3scale.net/service-id" 标签，其值与有效的 service_id 对应。

   b. 添加 "service-mesh.3scale.net/credentials" 标签，其值为第 1 步中的 *handler 资源*

的名称。

3. 每当您想要添加更多服务时，请执行第 2 步将其链接到您的 3scale 帐户凭证及其服务标识符。

4. 用 3scale 配置来修改规则配置，将规则发送到 3scale 处理器。

规则配置示例

```
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
```

1.21.1.1. 生成 3scale 自定义资源

适配器包括了一个可以用来生成 handler、instance 和 rule 自定义资源的工具。

表 1.29. 使用方法

选项	描述	必需的	默认值
-h, --help	显示可用选项的帮助信息	不是	
--name	这个 URL 的唯一名称，令牌对	是	
-n, --namespace	生成模板的命名空间	不是	istio-system
-t, --token	3scale 访问令牌	是	
-u, --url	3scale Admin Portal URL	是	

选项	描述	必需的	默认值
--backend-url	3scale 后端 URL。如果设定，它会覆盖从系统配置中读取的值。	不是	
-s, --service	3scale API/Service ID	不是	
--auth	3scale 的认证方法（1=API Key, 2=App Id/App Key, 3=OIDC）	不是	混合
-o, --output	保存产生的清单的文件	不是	标准输出
--version	输出 CLI 版本并立即退出	不是	

1.21.1.1.1. 从 URL 示例生成模板

注意

- 通过 **oc exec** 运行以下命令从 3scale adapter 容器镜像生成清单（请参阅从一个部署的 adapter 生成清单）。
- 使用 **3scale-config-gen** 命令帮助避免 YAML 语法和缩进错误。
- 如果使用注解，可以省略 **--service**。
- 此命令必须通过 **oc exec** 从容器镜像内调用。

流程

- 使用 **3scale-config-gen** 命令自动生成模板文件，允许令牌、URL 对作为单个处理器由多个服务共享：

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

- 以下示例生成带有嵌入在处理器中的服务 ID 的模板：

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

其他资源

- 令牌.

1.21.1.2. 从部署的适配器生成清单

注意

- **NAME 是用于标识您使用 3scale 管理的服务的标识符。**
- **CREDENTIALS_NAME 引用是一个标识符，对应于规则配置中的 match 部分。如果您使用 CLI 工具，这会自动设置为 NAME 标识符。**
- **其值不需要任何特定内容：**标签值应当仅与规则的内容匹配。**如需更多信息，**请参阅通过适配器路由服务流量。

1. 运行这个命令从在 **istio-system**命名空间中部署的适配器生成清单：

```
$ 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. 这将在终端中输出示例。如果需要，请编辑这些样本，并使用 **oc create** 命令创建对象。

3. 当请求到达适配器时，适配器需要知道服务如何被映射到一个 3scale 中的 API。您可以以两种方式提供这个信息：

   a. 标记（label）工作负载（推荐）

b. 硬编码处理器为 **service_id**

4. 使用所需注解更新工作负载：

注意

如果服务 ID 没有被嵌入到处理器中，您只需要更新本示例中的服务 ID。处理器中的设置会优先使用。

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

1.21.1.3. 通过适配器的路由服务流量

按照以下步骤，通过 3scale 适配器为您的服务驱动流量。

先决条件

- 3scale 管理员的凭据和服务 ID。

流程

1. 匹配在以前创建的配置中的 **destination.labels["service-mesh.3scale.net/credentials"] == "threescale"**（在 **kind: rule** 资源中）。

2. 在部署目标负载时为 **PodTemplateSpec**添加上面的标签以集成服务 。**threescale**是生成的处理器的名称。这个处理器存储了调用 3scale 所需的访问令牌。

3. 为工作负载添加 **destination.labels["service-mesh.3scale.net/service-id"] == "replace-me"** 标签，以便在请求时通过实例将服务 ID 传递给适配器。

1.21.2. 在 3scale 中配置集成设置

按照以下步骤配置 3scale 集成设置。

> **注意**
>
> 对于 3scale SaaS 客户，Red Hat OpenShift Service Mesh 作为 Early Access 项目的一部分被启用。

流程

1. 进入 **[your_API_name] → Integration**
2. 单击 **Settings**。
3. 在 ***Deployment*** 下选择 **Istio** 选项。
   - ***Authentication*** 下的 **API Key (user_key)** 选项会被默认选择。
4. 单击 **Update Product** 以保存您的选择。
5. 单击 **Configuration**。
6. 单击 **Update Configuration**。

1.21.3. 缓存行为

在适配器中，来自 3scale System API 的响应会被默认缓存。当条目存在的时间超过 cacheTTLSeconds 所指定的值时，条目会从缓存清除 。另外，默认情况下，根

据cacheRefreshSeconds 的值，在缓存的条目过期前会尝试进行自动刷新。您可以通过设置高于 cacheTTLSeconds 的值来禁用自动刷新。

把 cacheEntriesMax 设置为一个非正数的值可以完全禁用缓存。

通过使用刷新功能，那些代表已无法访问的主机的缓存项，会在其过期并最终被删除前重新尝试进行检索。

1.21.4. 身份验证请求

这个发行版本支持以下验证方法：

- 标准 API 键：使用一个随机字符串或哈希值作为标识符和 secret 令牌。
- 应用程序标识符和键对：不可改变的标识符和可变的密键字符串。
- OpenID 验证方法：从 JSON Web Token 解析的客户端 ID 字符串。

1.21.4.1. 应用验证模式

如以下验证方法示例所示，修改 instance 自定义资源来配置身份验证的方法。您可以接受以下身份验证凭证：

- 请求的标头（header）
- 请求参数
- 请求标头和查询参数

注意

当通过标头指定值时，必须为小写。例如，如果需要发送一个标头为 User-Key，则需要在配置中使用 request.headers["user-key"] 来指代它。

1.21.4.1.1. API 键验证方法

根据在 subject 自定义资源参数的 user 选项，Service Mesh 在查询参数和请求标头中查找 API 键。它会按照自定义资源文件中给出的顺序检查这些值。您可以通过跳过不需要的选项，把对 API 键的搜索限制为只搜索查询参数或只搜索请求标头。

在这个示例中，Service Mesh 在 user_key 查询参数中查找 API 键。如果 API 键不在查询参数中，Service Mesh 会检查 user-key 标头。

API 键验证方法示例

```
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"
```

如果您希望适配器检查不同的查询参数或请求标头，请根据情况更改名称。例如：要在名为 "key" 的查询参数中检查 API 键，把 request.query_params["user_key"] 改为 query_params["key"]。

1.21.4.1.2. 应用程序 ID 和应用程序键对验证方法

根据 subject 自定义资源参数中的 properties 选项，Service Mesh 在查询参数和请求标头中查找应用程序 ID 和应用程序键。应用程序键是可选的。它会按照自定义资源文件中给出的顺序检查这些值。通

过不使用不需要的选项，可以将搜索凭证限制为只搜索查询参数或只搜索请求标头。

在这个示例中，Service Mesh 会首先在查询参数中查找应用程序 ID 和应用程序键，如果需要，再对请求标头进行查找。

应用程序 ID 和应用程序键对验证方法示例

```
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"
```

如果您希望适配器检查不同的查询参数或请求标头，请根据情况更改名称。例如，要在名为 identification 的查询参数中查找应用程序 ID，把 request.query_params["app_id"] 改为 request.query_params["identification"]。

1.21.4.1.3. OpenID 验证方法

要使用 *OpenID Connect (OIDC) 验证方法*，使用 subject 字段中的 properties 值设定 client_id 及可选的 app_key。

您可以使用前面描述的方法操作这个对象。在下面的示例配置中，客户标识符（应用程序 ID）是从标签 *azp* 下的 JSON Web Token (JWT) 解析出来的。您可以根据需要修改它。

OpenID 验证方法示例

```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"] | ""
```

要使这个集成服务可以正常工作，OIDC 必须在 3scale 中完成，以便客户端在身份提供者 (IdP) 中创建。您应该为您要保护的服务在与该服务相同的命名空间中创建 Request 授权。JWT 由请求的 Authorization 标头传递。

在下面定义的 RequestAuthentication 示例中，根据情况替换 issuer、jwksUri 和 selector。

OpenID 策略示例

```yaml
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: info
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
```

1.21.4.1.4. 混合验证方法

您可以选择不强制使用一个特定的验证方法，而是接受任何有效的凭证。如果 API 键和应用程序 ID/应用程序键对都被提供，则 Service Mesh 会使用 API 键。

在这个示例中，Service Mesh 在查询参数中检查一个 API 键，然后是请求标头。如果没有 API 键，则会在查询参数中检查应用程序 ID 和键，然后查询请求标头。

混合验证方法示例

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

1.21.5. 3scale Adapter 指标数据

在默认情况下，适配器默会通过 /metrics 端点的端口 8080 提供各种 Prometheus 指标数据。这些指标可让您了解适配器和 3scale 之间的交互是如何执行的。该服务被标记为由 Prometheus 自动发现和弃用。

注意

与之前的 Service Mesh 1.x 版本相比，3scale Istio 适配器指标有不兼容的更改。

在 Prometheus 中，指标被重命名为后端缓存的新增名称，以便在 Service Mesh 2.0 中存在以下指标：

表 1.30. Prometheus 指标

指标	类型	描述
threescale_latency	Histogram	适配器和 3scale 之间的请求延迟。
threescale_http_total	计数	到 3scale 后端的请求的 HTTP 状态响应代码。
threescale_system_cache_hits	计数	从配置缓存获取到 3scale 系统的请求总数。
threescale_backend_cache_hits	计数	从后端缓存获取到 3scale 后端的请求总数。

1.21.6. 3scale 后端缓存

3scale 后端缓存为 3scale 服务管理 API 的客户端提供授权和报告缓存。这个缓存是嵌入到适配器中，在某些情况下可以降低响应速度，假设管理员接受利弊取而代之。

注意

默认情况下禁用 3scale 后端缓存。3scale 后端缓存功能会带来速率限制，以及因为处理器和内存中资源消耗较低延迟和较高消耗，自上次同步以来可能会丢失点击。

1.21.6.1. 启用后端缓存的优点

以下是启用后端缓存的优点：

- 当您在访问由 3scale Istio Adapter 管理的服务时，当您找到时，启用后端缓存。
- 启用后端缓存将停止适配器不断与 3scale API 管理器签注请求授权，这样可降低延迟。
    - 这会创建一个 3scale Istio 适配器的 3scale 授权内存缓存，以便在试图联系 3scale

API Manager 获取授权前进行存储和重复使用。这样一来，授予或拒绝授权的时间要短得多。

- 当您从运行 3scale Istio 适配器的服务网格的其他地理位置托管 3scale API manager 时，后端缓存很有用。
    - 这通常是 3scale 托管(SaaS)平台的情况。另外,当用户将 3scale API Manager 托管在另一个群集中时,位于不同地理位置,位于不同的可用区,或者在需要到达 3scale API 管理器的网络开销的情况下也是如此。

1.21.6.2. 具有较低延迟的利弊得失

以下是具有较低延迟的利弊得失：

- 每个 3scale 适配器的授权状态会在每次 flush 发生时更新。
    - 这意味着，两个或者多个适配器实例将在刷新周期之间引入更多不准确。
    - 授予太多请求的几率超过限制并引入异常行为，这会导致一些请求被处理，以及一些请求没有完成，这取决于每个适配器处理的请求。
- 如果适配器缓存无法清除其数据并更新其授权信息，而不向 API Manager 报告其信息，则可能会关闭或崩溃。
- 当适配器缓存无法决定请求是否被授予或拒绝时，会应用一个失败的打开策略或已关闭策略，这可能是因为与 API Manager 联系时存在网络连接问题。
- 当缓存丢失时（通常是在引导适配器后或者长时间没有连接后），查询 API 管理器时会增大延迟。
- 适配器缓存必须在计算授权上做很多工作，而不必启用缓存，这会消耗处理器资源。

- 内存需求会与由缓存管理的限值、应用程序和服务的数量成比例增长。

1.21.6.3. 后端缓存配置设置

以下解释了后端缓存配置设置：

- 在 3scale 配置选项中找到用于配置后端缓存的设置。
- 最后的 3 设置控制启用后端缓存：
  - PARAM_USE_CACHE_BACKEND - 设置为 true 以启用后端缓存。
  - PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS - 以秒为单位设置连续尝试刷新 API 管理器缓存数据的时间（以秒为单位）。
  - PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED - 当没有足够的缓存数据且无法到达 3scale API Manager 时，设定是否允许/打开或拒绝对服务的请求。

1.21.7. 3scale Istio Adapter APIcast emulation

当出现以下情况时，3scale Istio Adapter 会以 APIcast 的形式执行：

- 当请求无法与定义的任何映射规则匹配时，返回的 HTTP 代码为 404 Not Found。之前是 403 Forbidden。
- 当一个请求因为超过限制而被拒绝时，返回的 HTTP 代码为 429 Too Many Requests。之前是 403 Forbidden。
- 在通过 CLI 生成默认模板时，对于标头使用下划线而不是横线，例如： user_key 而不是 user-key。

1.21.8. 3scale Istio 适配器验证

您可能需要检查 3scale Istio 适配器是否按预期工作。如果您的适配器无法正常工作，请按照以下步骤帮助排除此问题。

流程

1. 确保 *3scale-adapter* pod 在 Service Mesh control plane 命名空间中运行：

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

2. 检查 *3scale-adapter* pod 是否已输出有关自身引导的信息，比如它的版本：

   ```
   $ oc logs istio-system
   ```

3. 在对 3scale 适配器集成保护的服务执行请求时，请始终尝试缺少正确凭证的请求，并确保它们失败。检查 3scale 适配器日志来收集其他信息。

其他资源

- 检查容器集和容器日志.

1.21.9. 3scale Istio 适配器故障排除清单

作为管理员安装 3scale Istio 适配器，在有些情况下可能会导致您的集成无法正常工作。使用以下列表排除安装故障：

- YAML 缩进不正确。
- 缺少 YAML 部分。
- 忘记将 YAML 中的更改应用到集群。

- 忘记使用 service-mesh.3scale.net/credentials 键标记服务工作负载。

- 当使用不包含 service_id 的处理程序时，忘记使用 service-mesh.3scale.net/service-id 标记服务工作负载，导致每个帐户可以重复使用它们。

- *Rule* 自定义资源指向错误的处理程序或实例自定义资源，或者引用缺少对应的命名空间后缀。

- *Rule* 自定义资源的 match 部分可能无法与您配置的服务匹配，或者指向当前没有运行或不存在的目标工作负载。

- 处理程序中 3scale 管理门户的访问令牌或 URL 错误。

- *Instance* 自定义资源的 params/subject/properties 部分无法列出 app_id、app_key 或 client_id 的正确参数，因为它们指定了错误的位置，如查询参数、标头和授权声明，或者参数名称与用于测试的请求不匹配。

- 在未意识到它实际存放在适配器容器镜像中，并且需要 oc exec 调用它的情况下，未能使用配置生成器。

1.22. 服务网格故障排除

本节论述了如何识别和解决 Red Hat OpenShift Service Mesh 中的常见问题。在 Red Hat OpenShift Service on AWS 上部署 Red Hat OpenShift Service Mesh 时，请使用以下部分帮助排除故障并调试问题。

1.22.1. 了解 Service Mesh 版本

要了解您在系统上部署的 Red Hat OpenShift Service Mesh 版本，您需要了解如何管理各个组件版本。

- Operator 版本 - 最新版本为 2.6.7。Operator 版本号仅指示当前安装的 Operator 的版本。因为 Red Hat OpenShift Service Mesh Operator 支持 Service Mesh control plane 的多个版

本，所以 Operator 的版本不会决定部署的 ServiceMeshControlPlane 资源的版本。

重要

升级到最新的 Operator 版本会自动应用补丁更新，但不会自动将 Service Mesh control plane 升级到最新的次版本。

- ServiceMeshControlPlane 版本 - ServiceMeshControlPlane 版本决定您使用的 Red Hat OpenShift Service Mesh 版本。ServiceMeshControlPlane 资源中的 spec.version 字段的值控制用于安装和部署 Red Hat OpenShift Service Mesh 的架构和配置设置。创建 Service Mesh control plane 时，您可以使用以下两种方式之一设置版本：
  - 要在 Form View 中配置，请从 Control Plane Version 菜单中选择版本。
  - 要在 YAML View 中配置，请在 YAML 文件中设置 spec.version 的值。

Operator Lifecycle Manager(OLM)不管理 Service Mesh control plane 升级，因此，Operator 和 ServiceMeshControlPlane (SMCP)的版本号可能不匹配，除非您手动升级 SMCP。

1.22.2. 安装故障排除 Operator

1.22.2.1. 验证 Operator 安装

安装 Red Hat OpenShift Service Mesh Operator 时，OpenShift 会在成功安装 Operator 时自动创建以下对象：

- 配置映射
- 自定义资源定义
- 部署

- **pods**
- **副本集**
- **角色**
- **角色绑定**
- **secrets**
- 服务帐户
- **services**

从 Red Hat OpenShift Service on AWS 控制台

您可以使用 Red Hat OpenShift Service on AWS 控制台验证 **Operator pod 是否可用并**运行。

1. 导航到 **Workloads → Pods。**
2. 选择 **openshift-operators 命名空间。**
3. 验证以下 **pod 是否存在，状态是否为 running ：**
   - **istio-operator**
   - **jaeger-operator**

- **kiali-operator**

4. 选择 **openshift-operators-redhat** 命名空间。

5. 验证 **elasticsearch-operator** pod 是否存在，状态是否为 **running**。

从命令行

1. **使用以下命令，**验证 **Operator pod 可用并在 openshift-operators** 命名空间中运行：

```
$ oc get pods -n openshift-operators
```

输出示例

```
NAME                               READY   STATUS    RESTARTS   AGE
istio-operator-bb49787db-zgr87     1/1     Running   0          15s
jaeger-operator-7d5c4f57d8-9xphf   1/1     Running   0          2m42s
kiali-operator-f9c8d84f4-7xh2v     1/1     Running   0          64s
```

2. **使用以下命令**验证 **Elasticsearch Operator：**

```
$ oc get pods -n openshift-operators-redhat
```

输出示例

```
NAME                                      READY   STATUS    RESTARTS   AGE
elasticsearch-operator-d4f59b968-796vq    1/1     Running   0          15s
```

1.22.2.2. 服务网格 Operator 故障排除

如果遇到 Operator 问题：

- 验证 Operator 订阅状态。
- 验证您安装的 Operator 是受支持的红帽版本，而不是社区版本。
- 验证您有 cluster-admin 角色来安装 Red Hat OpenShift Service Mesh。
- 检查 Operator pod 日志中是否存在与安装 Operator 相关的问题。

> **注意**
>
> 您只能通过 OpenShift 控制台安装 Operator，无法从命令行访问 OperatorHub。

1.22.2.2.1. 查看 Operator pod 日志

您可以使用 oc logs 命令查看 Operator 日志。红帽可能会要求提供日志来帮助解决支持问题单。

流程

- 要查看 Operator pod 日志，请输入以下命令：

```
$ oc logs -n openshift-operators <podName>
```

例如，

```
$ oc logs -n openshift-operators istio-operator-bb49787db-zgr87
```

1.22.3. control plane 故障排除

Service Mesh *control plane* 由 Istiod 组成，它会将几个以前的 control plane 组件（Citadel、Galley、Pilot）整合为一个二进制文件。部署 ServiceMeshControlPlane 还会创建组成 Red Hat

OpenShift Service Mesh 的其他组件，如架构主题所述。

1.22.3.1. 验证 Service Mesh control plane 安装

在创建 Service Mesh control plane 时，Service Mesh Operator 使用您在 ServiceMeshControlPlane 资源文件中指定的参数进行以下操作：

- 创建 Istio 组件并部署以下 pod：
  - istiod
  - istio-ingressgateway
  - istio-egressgateway
  - grafana
  - prometheus
- 调用 Kiali Operator 根据 SMCP 或 Kiali 自定义资源中的配置创建 Kaili 部署。

  注意

  您可以查看 Kiali Operator 中的 Kiali 组件，而不是 Service Mesh Operator。

- 调用 Red Hat OpenShift distributedracing Platform (Jaeger) Operator，以根据 SMCP 或 Jaeger 自定义资源中的配置创建分布式 Tracing Platform (Jaeger)组件。

注意

您可以在 Red Hat OpenShift Distributed Tracing Platform (Jaeger) Operator 和 Red Hat Elasticsearch Operator 下的 Elasticsearch 组件中查看 Jaeger 组件，而不是 Service Mesh Operator。

从 Red Hat OpenShift Service on AWS 控制台

您可以在 Red Hat OpenShift Service on AWS web 控制台中验证 Service Mesh control plane 安装。

1. 导航到 **Operators** → **Installed Operators**。

2. 选择 **istio-system** 命名空间。

3. 选择 Red Hat OpenShift Service Mesh Operator。

    a. 点 **Istio Service Mesh Control Plane** 标签页。

    b. 点 control plane 的名称，例如 **basic**。

    c. 若要查看部署所创建的资源，可单击 **Resources** 选项卡。您可以使用过滤器来缩小您的视图，例如，检查所有 **Pod** 的状态是否为 **running**。

    d. 如果 SMCP 状态指示任何问题，请检查 YAML 文件中的 **status:** 输出以了解更多信息。

4. 返回到 **Operators** → **Installed Operators**。

5. 选择 OpenShift Elasticsearch Operator。

a. 点 **Elasticsearch** 标签页。

b. 点部署的名称，如 **elasticsearch**。

c. 若要查看部署创建的资源，请点 **Resources** 选项卡。

d. 如果状态列有任何问题，请检查 **YAML** 选项卡中的 **status:** 输出以了解更多信息。

6. 返回到 **Operators** → **Installed Operators**。

7. 选择 **Red Hat OpenShift Distributed Tracing Platform (Jaeger) Operator**。

a. 点 **Jaeger** 标签页。

b. 点部署的名称，如 **jaeger**。

c. 若要查看部署所创建的资源，可单击 **Resources** 选项卡。

d. 如果状态列显示任何问题，请检查 **YAML** 选项卡中的 **status:** 输出以了解更多信息。

8. 导航到 **Operators** → **Installed Operators**。

9. 选择 **Kiali Operator**。

a. 点 **Istio Service Mesh Control Plane** 标签页。

b. 点部署的名称，如 kiali。

c. 若要查看部署所创建的资源，可单击 Resources 选项卡。

d. 如果状态列有任何问题，请检查 YAML 选项卡中的 status: 输出以了解更多信息。

从命令行

1. 运行以下命令，以查看 Service Mesh control plane pod 是否可用并正在运行，其中 istio-system 是安装 SMCP 的命名空间。

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

输出示例

```
NAME                                   READY   STATUS    RESTARTS   AGE
grafana-6776785cfc-6fz7t               2/2     Running   0          102s
istio-egressgateway-5f49dd99-l9ppq     1/1     Running   0          103s
istio-ingressgateway-6dc885c48-jjd8r   1/1     Running   0          103s
istiod-basic-6c9cc55998-wg4zq          1/1     Running   0          2m14s
jaeger-6865d5d8bf-zrfss                2/2     Running   0          100s
kiali-579799fbb7-8mwc8                 1/1     Running   0          46s
prometheus-5c579dfb-6qhjk              2/2     Running   0          115s
```

2. 使用以下命令检查 Service Mesh control plane 部署的状态。使用部署 SMCP 的命名空间替换 istio-system。

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

当 STATUS 列是 ComponentsReady 时，安装成功完成。

输出示例

```
NAME    READY   STATUS            PROFILES      VERSION   AGE
basic   10/10   ComponentsReady   ["default"]   2.1.3     4m2s
```

如果修改并重新部署了 Service Mesh control plane，其状态应该会显示 UpdateSuccessful。

输出示例

```
NAME            READY     STATUS             TEMPLATE   VERSION   AGE
basic-install   10/10     UpdateSuccessful   default     v1.1     3d16h
```

3. 如果 SMCP 状态指示了 ComponentsReady 以外的任何内容，请检查 SCMP 资源中的 status: 输出以获取更多信息。

```
$ oc describe smcp <smcp-name> -n <controlplane-namespace>
```

输出示例

```
$ oc describe smcp basic -n istio-system
```

4. 使用以下命令检查 Jaeger 部署的状态，其中 istio-system 是部署 SMCP 的命名空间。

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

输出示例

```
NAME    STATUS   VERSION  STRATEGY  STORAGE  AGE
jaeger  Running  1.30.0   allinone  memory   15m
```

5. 使用以下命令检查 Kiali 部署的状态，其中 istio-system 是部署 SMCP 的命名空间。

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

输出示例

```
NAME   AGE
kiali  15m
```

1.22.3.1.1. 访问 Kiali 控制台

您可以在 Kiali 控制台中查看应用程序的拓扑、健康和指标。如果您的服务遇到问题，Kiali 控制台可让您通过服务查看数据流。您可以查看不同级别中的与网格组件相关的信息，包括抽象应用程序、服务以及负载。Kiali 还会实时提供命名空间的互动图形视图。

要访问 Kiali 控制台，您必须安装并配置了 Red Hat OpenShift Service Mesh。

安装过程创建了访问 Kiali 控制台的路由。

如果您知道 Kiali 控制台的 URL，您可以直接访问它。如果您不知道 URL，请使用以下指示：

管理员的步骤

1. 使用管理员角色登录到 Red Hat OpenShift Service on AWS Web 控制台。

2. 点 **Home** → **Projects**。

3. 如有必要，在 **Projects** 页面上，使用过滤器来查找项目的名称。

4. 点项目的名称，例如 **info**。

5. 在 **Project details** 页面中，点 **Launcher** 部分的 **Kiali** 链接。

6. 使用与访问 Red Hat OpenShift Service on AWS 控制台相同的用户名和密码登录到 Kiali 控制台。

   第一次登录到 Kiali 控制台时，您会看到 **Overview** 页面，它会显示服务网格中您有权查看的所有命名空间。

   如果您验证了控制台安装，且命名空间还没有添加到网格中，则可能无法显示 **istio-system** 以外的任何数据。

开发人员的步骤

1. 使用开发者角色登录 Red Hat OpenShift Service on AWS Web 控制台。

2. 单击 **Project**。

3. 如有必要，在 **Project Details** 页面上，使用过滤器来查找项目的名称。

4. 点项目的名称，例如 **info**。

5. 在 **Project** 页面中，点 **Launcher** 部分的 **Kiali** 链接。

6.

6. 单击 **Log In With OpenShift**。

1.22.3.1.2. 访问 Jaeger 控制台

要访问 Jaeger 控制台，您必须安装并配置了 Red Hat OpenShift Service Mesh。

安装过程会创建路由来访问 Jaeger 控制台。

如果您知道 Jaeger 控制台的 URL，您可以直接访问它。如果您不知道 URL，请使用以下指示：

重要

从 Red Hat OpenShift Service Mesh 2.5 开始，Red Hat OpenShift distributed Tracing Platform (Jaeger)和 OpenShift Elasticsearch Operator 已被弃用，并将在以后的发行版本中删除。红帽将在当前发行生命周期中提供对这个功能的程序漏洞修复和支持，但这个功能将不再获得改进，并将被删除。作为 Red Hat OpenShift distributed Tracing Platform (Jaeger)的替代选择，您可以使用 Red Hat OpenShift distributed Tracing Platform。

从 OpenShift 控制台的步骤

1. 以具有 cluster-admin 权限的用户身份登录Red Hat OpenShift Service on AWS Web。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 dedicated-admin 角色的帐户。

2. 进入 **Networking** → **Routes**。

3. 在 **Routes** 页面中，从 **Namespace** 菜单中选择 Service Mesh control plane 项目，如 istio-system。

   Location 列显示每个路由的链接地址。

4. 如有必要，使用过滤器来查找 jaeger 路由。单击路由 **位置** 以启动控制台。

5.

5. 单击 **Log In With OpenShift**。

Kiali 控制台的步骤

1. 启动 Kiali 控制台。

2. 单击左侧导航窗格中的 **Distributed Tracing**。

3. 单击 **Log In With OpenShift**。

通过 CLI 操作的步骤

1. 以具有 **cluster-admin** 角色的用户身份登录到 Red Hat OpenShift Service on AWS CLI。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 **dedicated-admin** 角色的帐户。

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. 要使用命令行查询路由详情，请输入以下命令。在本例中，**istio-system** 是 Service Mesh control plane 命名空间。

```
$ oc get route -n istio-system jaeger -o jsonpath='{.spec.host}'
```

3. 启动浏览器并进入 **https://<JAEGER_URL>**，其中 **<JAEGER_URL>** 是您在上一步中发现的路由。

4. 使用您用来访问 Red Hat OpenShift Service on AWS 控制台的同一用户名和密码登录。

5. 如果您已将服务添加到服务网格中并生成了 trace，您可以使用过滤器和 **Find Traces** 按钮搜索 trace 数据。

   如果您要验证控制台安装，则不会显示 trace 数据。

1.22.3.2. Service Mesh control plane 故障排除

如果您在部署 Service Mesh control plane 时遇到问题，

- 确保 ServiceMeshControlPlane 资源安装在一个与您的服务和 Operator 分开的项目中。本文档使用 istio-system 项目作为示例，但只要它与包含 Operator 和服务的项目分开，就可以在任何项目中部署 control plane。

- 确保 ServiceMeshControlPlane 和 Jaeger 自定义资源已部署在同一项目中。例如，两者都使用 istio-system 项目。

1.22.4. 对数据平面进行故障排除

数据平面 是一组智能代理，用于拦截和控制服务网格中服务之间的所有入站和出站网络通信。

Red Hat OpenShift Service Mesh 依赖于应用程序 pod 中的 proxy sidecar 来为应用程序提供服务网格功能。

1.22.4.1. sidecar 注入故障排除

Red Hat OpenShift Service Mesh 不会自动将代理 sidecar 注入 pod。您必须选择 sidecar 注入。

1.22.4.1.1. Istio sidecar 注入故障排除

检查您的应用程序的部署中是否启用了自动注入。如果为 Envoy 代理启用了自动注入，则 spec.template.metadata.annotations 的 Deployment 资源中应该有一个 sidecar.istio.io/inject:"true" 注解。

1.22.4.1.2. Jaeger 代理 sidecar 注入故障排除

检查您的应用程序的部署中是否启用了自动注入。如果启用了 Jaeger 代理的自动注入，Deployment 资源中应该有一个 sidecar.jaegertracing.io/inject:"true" 注解。

如需有关 sidecar 注入的更多信息，请参阅启用自动注入

1.23. ENVOY 代理故障排除

Envoy 代理会截获服务网格中所有服务的入站和出站流量。Envoy 还会收集并报告服务网格上的遥测。Envoy 会在同一个 pod 中被部署为相关服务的 sidecar。

1.23.1. 启用 Envoy 访问日志

Envoy 访问日志可用于诊断流量故障和流，并帮助端到端流量流分析。

要为所有 istio-proxy 容器启用访问日志，请编辑 ServiceMeshControlPlane (SMCP) 对象为日志输出添加文件名。

流程

1. 以具有 cluster-admin 角色的用户身份登录 OpenShift Container Platform CLI。输入以下命令。然后在提示时输入您的用户名和密码。

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. 切换到安装 Service Mesh control plane 的项目，如 istio-system。

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

3. 编辑 ServiceMeshControlPlane 文件。

```
$ oc edit smcp <smcp_name>
```

4. 如以下示例所示，使用 name 指定代理日志的文件名。如果没有为 name 指定值，则不会写入日志条目。

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  proxy:
    accessLogging:
      file:
        name: /dev/stdout     #file name
```

1.23.2. 获取支持

如果您在执行本文档所述的某个流程或 Red Hat OpenShift Service on AWS 时遇到问题，请访问 红帽客户门户网站。

通过红帽客户门户网站：

- 搜索或者浏览红帽知识库，了解与红帽产品相关的文章和解决方案。
- 提交问题单给红帽支持。
- 访问其他产品文档。

要识别集群中的问题，您可以在 OpenShift Cluster Manager 中使用 Insights。Insights 提供了问题的详细信息，并在有可用的情况下，提供了如何解决问题的信息。

如果您对本文档有任何改进建议，或发现了任何错误，请为相关文档组件提交 JIRA 问题。请提供具体详情，如章节名称和 Red Hat OpenShift Service on AWS 版本。

1.23.2.1. 关于红帽知识库

红帽知识库提供丰富的内容以帮助您最大程度地利用红帽的产品和技术。红帽知识库包括文章、产品文档和视频，概述了安装、配置和使用红帽产品的最佳实践。另外，您还可以搜索已知问题的解决方案，其提供简洁的根原因描述和补救措施。

1.23.2.2. 搜索红帽知识库

如果出现 Red Hat OpenShift Service on AWS 问题，您可以先进行搜索，以确定红帽知识库中是否已存在相关的解决方案。

先决条件

- 您有红帽客户门户网站帐户。

流程

1. 登录到 红帽客户门户网站。
2. 点 Search。
3. 在搜索字段中，输入与问题相关的关键字和字符串，包括：
   - Red Hat OpenShift Service on AWS 组件（如 etcd）
   - 相关步骤（比如 安装）
   - 警告、错误消息和其他与输出与特定的问题相关
4. 点 Enter 键。
5. 可选： 选择 Red Hat OpenShift Service on AWS 产品过滤器。
6. 可选： 选择 Documentation 内容类型过滤器。

1.23.2.3. 提交支持问题单

先决条件

- 您可以使用具有 dedicated-admin 角色的用户访问集群。
- 已安装 OpenShift CLI（oc）。
- 您可以访问 Red Hat OpenShift Cluster Manager。

流程

1. 登录到红帽客户门户网站的客户支持 页面。

2. 点 Get support。

3. 在 客户支持 页面的 Cases 选项卡中：

    a. 可选：根据需要更改预先填充的帐户和所有者详情。

    b. 为您的问题选择适当的类别，如 Bug 或 Defect，然后单击 Continue。

4. 输入以下信息：

    a. 在 Summary 字段中，输入简要但描述性问题概述，以及有关所经历的症状的详细信息，以及您的预期。

    b. 从 Product 下拉菜单中选择 Red Hat OpenShift Service on AWS。

5. 查看推荐的红帽知识库解决方案列表，它们可能会与您要报告的问题相关。如果建议的文章没有解决这个问题，请点 Continue。

6. 查看更新的推荐红帽知识库解决方案列表，它们可能会与您要报告的问题相关。这个列表的范围会缩小，因为您在创建问题单的过程中提供了更多信息。如果建议的文章没有解决这个问题，请点 Continue。

7. 请确保提供的帐户信息是正确的，如果需要，请相应调整。

8. 检查自动填充的 Red Hat OpenShift Service on AWS 集群 ID 是否正确。如果不正确，请手动提供集群 ID。

- 使用 Red Hat OpenShift Service on AWS Web 控制台手动获得集群 ID：

  a. 导航到 Home → Overview。

  b. 该值包括在 Details 中的 Cluster ID 项中。

- 另外，也可以通过 Red Hat OpenShift Service on AWS Web 控制台直接创建新的支持问题单，并自动填充集群 ID。

  a. 从工具栏导航至 （？）help → Open Support Case。

  b. Cluster ID 的值会被自动填充 。

- 要使用 OpenShift CLI（oc）获取集群 ID，请运行以下命令：

```
$ oc get clusterversion -o jsonpath='{.items[].spec.clusterID}{"\n"}'
```

9. 完成以下提示的问题，点 Continue:

- 您遇到什么情况？您期望发生什么情况？

- 定义对您或业务的价值或影响。

- 您在哪里遇到此行为？什么环境？

- 此行为何时发生？发生频率？重复发生？是否只在特定时间发生？

10. 上传相关的诊断数据文件并点击 Continue。

11. 输入相关问题单管理详情，点 **Continue**。

12. 预览问题单详情，点 **Submit**。

1.24. SERVICE MESH CONTROL PLANE 配置参考

您可以通过修改默认的 ServiceMeshControlPlane (SMCP) 资源或创建完全自定义 SMCP 资源来自定义 Red Hat OpenShift Service Mesh。本参考部分记录了可用于 SMCP 资源的配置选项。

1.24.1. Service Mesh Control plane 参数

下表列出了 ServiceMeshControlPlane 资源的顶级参数。

表 1.31. ServiceMeshControlPlane 资源参数

名称	描述	类型
apiVersion	APIVersion 定义对象的这个表示法的版本化的 schema。服务器将识别的模式转换为最新的内部值，并可拒绝未识别的值。**ServiceMeshControlPlane** 版本 2.0 的值为 **maistra.io/v2**。	**ServiceMeshControlPlane** 版本 2.0 的值为 **maistra.io/v2**。
kind	kind 是一个字符串值，代表此对象所代表的 REST 资源。	**ServiceMeshControlPlane** 是 ServiceMeshControlPlane 的唯一有效值。
metadata	关于这个 **ServiceMeshControlPlane** 实例的元数据。您可以为 Service Mesh control plane 安装提供一个名称来跟踪您的工作，例如 **basic**。	字符串
spec	此 **ServiceMeshControlPlane** 所需状态的规格。这包括组成 Service Mesh control plane 的所有组件的配置选项。	如需更多信息，请参阅表 2。

名称	描述	类型
status	此 **ServiceMeshControlPlane** 的当前状态以及组成 Service Mesh control plane 的组件。	如需更多信息，请参阅表 3。

下表列出了 **ServiceMeshControlPlane** 资源规格。更改这些参数配置 Red Hat OpenShift Service Mesh 组件。

表 1.32. ServiceMeshControlPlane 资源规格

名称	描述	可配置参数
附加组件	**addons** 参数配置除 Service Mesh control plane 组件之外的额外功能，如视觉化或指标存储。	**3scale**、**grafana**、**jaeger**、**kiali** 和 **prometheus**。
cluster	**cluster** 参数设置集群的常规配置（集群名称、网络名称、多集群、网格扩展等等）	**meshExpansion**、**multiCluster**、**name** 和 **网络**
gateways	您可以使用 **gateways** 参数为网格配置入口和出口网关。	**enabled**、**additionalEgress**、**additionalIngress**、**egress**、**ingress** 和 **openshiftRoute**
general	**general** 参数代表在其它任何位置都不适用的常规 Service Mesh control plane 配置。	**logging** 和 **validationMessages**
policy	您可以使用 **policy** 参数为 Service Mesh control plane 配置策略检查。通过将 **spec.policy.enabled** 设置为 **true** 来启用策略检查。	**mixer remote** 或 **type**。**type** 可以被设置为 **Istiod**、**Mixer** 或 **None**。
profiles	您可以使用 **profile** 参数设置用于默认值的 **ServiceMeshControlPlane** 配置集。	**default**
proxy	您可以使用 **proxy** 参数来配置 sidecar 的默认行为。	**accessLogging**、**adminPort**、**concurrency** 和 **envoyMetricsService**
runtime	您可以使用 **runtime** 参数配置 Service Mesh control plane 组件。	**components** 和 **defaults**

名称	描述	可配置参数
安全	**security** 参数允许您为 Service Mesh control plane 配置安全性方面。	**certificateAuthority**、**control Plane**、**identity**、**dataPlane** 和 **trust**
techPreview	**techPreview** 参数允许早期访问技术预览中的功能。	N/A
Telemetry	如果 **spec.mixer.telemetry.enabled** 被设置为 **true**，则遥测会被启用。	**mixer**, **remote**, 和 **type**.**type** 可以被设置为 **Istiod**、**Mixer** 或 **None**。
tracing	您可以使用 **tracing** 参数为网格启用分布式追踪。	**sampling**, **type**.**type** 可以被设置为 **Jaeger** 或 **None**。
version	您可以使用 **version** 参数指定要安装的 Service Mesh control plane 的 Maistra 版本。当使用空版本创建 **ServiceMeshControlPlane** 时，准入 Webhook 会将版本设置为当前版本。带有空版本的新的 **ServiceMeshControlPlanes** 设置为 **v2.0**。现有带有空版本的 **ServiceMeshControlPlanes** 会保留其设置。	字符串

ControlPlaneStatus 代表服务网格的当前状态。

表 1.33. ServiceMeshControlPlane 资源 ControlPlaneStatus

名称	描述	类型
annotations	**annotations** 参数存储额外的、通常多余的状态信息，如 **ServiceMeshControlPlane** 部署的组件数量。命令行工具 **oc** 使用这些状态，它还不允许在 JSONPath 表达式中计数对象。	无法配置

名称	描述	类型
conditions	代表对象当前状态的最新可用影响。**Recoveryd** 表示 Operator 是否已完成与 **ServiceMeshControlPlane** 资源中的配置协调已部署组件的实际状态。**Installed** 显示是否安装了 Service Mesh control plane。**Ready** 显示是否所有 Service Mesh control plane 组件都已就绪。	字符串
components	显示每个部署的 Service Mesh control plane 组件的状态。	字符串
appliedSpec	应用所有配置集后生成的配置选项规格。	**ControlPlaneSpec**
appliedValues	用于生成 chart 的 values.yaml。	**ControlPlaneSpec**
chartVersion	最后一次为此资源处理的图表版本。	字符串
observedGeneration	控制器在最新协调期间观察到的生成。状态中的信息与对象的特定生成有关。如果 **status.observedGeneration** 字段与 **metadata.generation** 不匹配，则 **status.conditions** 不会被最新。	整数
operatorVersion	最后处理此资源的 operator 版本。	字符串
readiness	组件和拥有资源的就绪状态。	字符串

这个示例 **ServiceMeshControlPlane** 定义包含所有支持的参数。

ServiceMeshControlPlane 资源示例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
```

```yaml
version: v2.6
proxy:
  runtime:
    container:
      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: {}
      persistence:
        enabled: true
        storageClassName: ""
        accessMode: ReadWriteOnce
        capacity:
          requests:
            storage: 5Gi
      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
          storage: {}
          redundancyPolicy: SingleRedundancy
          indexCleaner: {}
      ingress: {} # jaeger ingress configuration
runtime:
  components:
    pilot:
      deployment:
        replicas: 2
      pod:
```

```yaml
        affinity: {}
      container:
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 500m
            memory: 128Mi
    grafana:
      deployment: {}
      pod: {}
    kiali:
      deployment: {}
      pod: {}
```

1.24.2. spec 参数

1.24.2.1. 常规参数

下面是一个示例，它演示了 ServiceMeshControlPlane 对象的 spec.general 参数，以及可用参数和值的信息。

常规参数示例

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  general:
    logging:
      componentLevels: {}
        # misc: error
      logAsJSON: false
    validationMessages: true
```

表 1.34. Istio 常规参数

参数	描述	值	默认值
logging:	用于为 Service Mesh control plane 组件配置日志记录。		N/A
logging: componentLevels:	用于指定组件日志级别。	可能的值有：**debug**、**info**、**warn**、**error**、**fatal**.	N/A
logging: logAsJSON:	用于启用或禁用 JSON 日志。	**true**/**false**	N/A
validationMessages:	用于在 istio.io 资源的状态字段中启用或禁用验证信息。这对于检测资源中的配置错误非常有用。	**true**/**false**	N/A

1.24.2.2. 配置集参数

您可以使用 ServiceMeshControlPlane 对象配置集创建可重复使用的配置。如果没有配置 profile 设置，Red Hat OpenShift Service Mesh 将使用默认配置集。

下面是一个示例，它演示了 ServiceMeshControlPlane 对象的 spec.profiles 参数：

配置集参数示例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  profiles:
  - YourProfileName
```

有关创建配置集的详情，请参阅创建 control plane 配置集。

有关安全配置的详情，请参阅 Mutual Transport Layer Security(mTLS)。

1.24.2.3. 追踪参数

以下示例演示了 ServiceMeshControlPlane 对象的 spec.tracing 参数，以及可用参数和值的信息。

追踪参数示例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
  tracing:
    sampling: 100
    type: Jaeger
```

表 1.35. Istio 追踪参数

参数	描述	值	默认值
`tracing:` `  sampling:`	抽样率决定了 Envoy 代理生成 trace 的频率。您可以使用抽样率来控制向追踪系统报告的请求百分比。	0 到 10000 之间的整数值，代表 0.01% 的增长（0 到 100%）。例如，将值设置为 **10** 会抽样 0.1% 的请求，将值设为 **100** 会抽样 1% 的请求，设置为 **500** 会抽样 5% 的请求，设置 **10000** 会抽样 100% 的请求。	**10000** （100% trace）
`tracing:` `  type:`	目前 **Jaeger** 是唯一支持的追踪类型。默认启用 Jaeger。要禁用追踪，请将 **type** 参数设置为 **None**。	**None**, **Jaeger**	**Jaeger**

1.24.2.4. version 参数

Red Hat OpenShift Service Mesh Operator 支持安装不同版本的 ServiceMeshControlPlane。您可以使用 version 参数指定要安装的 Service Mesh control plane 版本。如果您在创建 SMCP 时没有指定 version 参数，Operator 会将值设置为最新版本：(2.6)。现有 ServiceMeshControlPlane 对象会在 Operator 升级过程中保留其版本设置。

1.24.2.5. 3scale 配置

下表解释了 ServiceMeshControlPlane 资源中的 3scale Istio 适配器的参数。

3scale 参数示例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
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_CACHE_REFRESH_RETRIES: 1
      PARAM_THREESCALE_ALLOW_INSECURE_CONN: false
      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
# ...
```

表 1.36. 3scale 参数

参数	描述	值	默认值
enabled	是否使用 3scale 适配器	true/false	false

参数	描述	值	默认值
PARAM_THREESCALE_LISTEN_ADDR	为 gRPC 服务器设定侦听地址	有效端口号	**3333**
PARAM_THREESCALE_LOG_LEVEL	设置最小日志输出级别。	**debug**、**info**、**warn**、**error**或 **none**	**info**
PARAM_THREESCALE_LOG_JSON	是否将日志格式转化为 JSON	**true/false**	**true**
PARAM_THREESCALE_LOG_GRPC	日志是否包含 gRPC 信息	**true/false**	**true**
PARAM_THREESCALE_REPORT_METRICS	是否收集 3scale 系统和后端的指标数据并报告给 Prometheus	**true/false**	**true**
PARAM_THREESCALE_METRICS_PORT	设置 3scale **/metrics** 端点可以从中分离的端口	有效端口号	**8080**
PARAM_THREESCALE_CACHE_TTL_SECONDS	在从缓存中移除过期项目前等待的时间（以秒为单位）	时间间隔（以秒为单位）	**300**
PARAM_THREESCALE_CACHE_REFRESH_SECONDS	尝试刷新缓存元素的过期时间	时间间隔（以秒为单位）	**180**
PARAM_THREESCALE_CACHE_ENTRIES_MAX	在任何时间可以保存在缓存中的最大项目数。设为 **0** 会禁用缓存	有效数量	**1000**
PARAM_THREESCALE_CACHE_REFRESH_RETRIES	在缓存更新循环中检索无法访问的主机的次数	有效数量	**1**
PARAM_THREESCALE_ALLOW_INSECURE_CONN	在调用 **3scale** API 时允许跳过证书验证。不推荐启用此功能。	**true/false**	**false**
PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS	终止到 3scale 系统和后端请求前等待的秒数	时间间隔（以秒为单位）	**10**

参数	描述	值	默认值
PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS	在连接关闭前设置连接的最大秒数（+/-10% 抖动）	时间间隔（以秒为单位）	60
PARAM_USE_CACHE_BACKEND	如果为 true，则尝试为授权请求创建一个内存 apisonator 缓存	**true/false**	**false**
PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS	如果启用了后端缓存，这会在 3scale 中设置刷新缓存的时间间隔（以秒为单位）	时间间隔（以秒为单位）	15
PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED	每当后端缓存无法检索授权数据时，无论是拒绝（已关闭）还是允许（打开）请求	**true/false**	**true**

1.24.3. 状态参数

status 参数描述了服务网格的当前状态。这些信息由 Operator 生成，且为只读。

表 1.37. Istio 状态参数

名称	描述	类型
observedGeneration	控制器在最新协调期间观察到的生成。状态中的信息与对象的特定生成有关。如果 **status.observedGeneration** 字段与 **metadata.generation** 不匹配，则 **status.conditions** 不会被最新。	整数
annotations	**annotations** 参数存储额外的、通常多余的状态信息，如由 **ServiceMeshControlPlane** 对象部署的组件数量。命令行工具 **oc** 使用这些状态，它还不允许在 JSONPath 表达式中计数对象。	无法配置
readiness	组件和拥有资源的就绪状态。	字符串
operatorVersion	最后处理此资源的 Operator 版本。	字符串

名称	描述	类型
components	显示每个部署的 Service Mesh control plane 组件的状态。	字符串
appliedSpec	应用所有配置集后生成的配置选项规格。	**ControlPlaneSpec**
conditions	代表对象当前状态的最新可用影响。**Recoveryd** 表示 Operator 已完成与 **ServiceMeshControlPlane** 资源中的配置协调已部署组件的实际状态。**Installed** 表示安装了 Service Mesh control plane。**Ready** 表示所有 Service Mesh control plane 组件都已就绪。	字符串
chartVersion	最后一次为此资源处理的图表版本。	字符串
appliedValues	生成的 **values.yaml** 文件，用于生成 chart。	**ControlPlaneSpec**

1.24.4. 其他资源

- **有关如何在 ServiceMeshControlPlane 资源中配置功能的更多信息，请参阅以下链接：**
  - 安全性
  - 流量管理
  - 指标和追踪

1.25. KIALI 配置参考

当 Service Mesh Operator 创建 ServiceMeshControlPlane 时，它也会处理 Kiali 资源。然后，当 Kiali Operator 创建 Kiali 实例时会使用这个对象。

1.25.1. 在 SMCP 中指定 Kiali 配置

您可以在 ServiceMeshControlPlane 资源的 addons 部分配置 Kiali。默认情况下启用 Kiali 。要禁用 Kiali，将 spec.addons.kiali.enabled 设置为 false。

您可以通过以下两种方式之一指定 Kiali 配置：

- 在 spec.addons.kiali.install 下指定 ServiceMeshControlPlane 资源中的 Kiali 配置。这个方法有一些限制，因为 SMCP 中没有 Kiali 配置的完整列表。

- 配置和部署 Kiali 实例，并将 Kiali 资源的名称指定为 ServiceMeshControlPlane 资源中的 spec.addons.kiali.name 的值。您必须在与 Service Mesh control plane 相同的命名空间中创建 CR，如 istio-system。如果存在与 name 值匹配的 Kiali 资源，control plane 将配置该 Kiali 资源以用于 control plane。这个方法可让您在 Kiali 资源中完全自定义 Kiali 配置。请注意，使用此方法，Kiali 资源的不同字段会被 Service Mesh Operator 覆盖，特别是 accessible_namespaces 列表，以及 Grafana、Prometheus 和追踪的端点。

Kiali 的 SMCP 参数示例

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  addons:
    kiali:
      name: kiali
      enabled: true
      install:
        dashboard:
          viewOnly: false
          enableGrafana: true
          enableTracing: true
          enablePrometheus: true
        service:
          ingress:
            contextPath: /kiali
```

表 1.38. ServiceMeshControlPlane Kiali 参数

参数	描述	值	默认值

spec:
  addons:
    kiali:
      name:
	Kiali 自定义资源的名称。如果存在与 **name** 值匹配的 Kiali CR，Service Mesh Operator 会将该 CR 用于安装。如果没有 Kiali CR，Operator 将使用这个 **名称**和 SMCP 中指定的配置选项创建一个。	字符串	**kiali**

kiali: enabled:	这个参数启用或禁用 Kiali。默认情况下启用 Kiali 。	**true/false**	**true**
kiali: install:	如果命名 Kiali 资源不存在，请安装 Kiali 资源。如果 **addons.kiali.enabled** 设为 **false**，则 **install** 部分会被忽略。		

kiali:
  install:
    dashboard:
	Kiali 提供的仪表板的配置参数。		

kiali:
  install:
    dashboard:
      viewOnly:
	为 Kiali 控制台启用或禁用只读视图模式。启用只读视图模式时，用户无法使用 Kiali 控制台来更改 Service Mesh。	**true/false**	**false**

kiali:
  install:
    dashboard:
      enableGrafana:
	基于 **spec.addons.grafana** 配置的 Grafana 端点。	**true/false**	**true**

kiali:
  install:
    dashboard:
 	
enablePrometheus:
	基于 **spec.addons.prometheus** 配置的 Prometheus 端点。	**true/false**	**true**

参数	描述	值	默认值
kiali: install: dashboard: enableTracing:	追踪根据 Jaeger 自定义资源配置配置的端点。	**true/false**	**true**
kiali: install: service:	与 Kiali 安装关联的 Kubernetes 服务的配置参数。		
kiali: install: service: metadata:	用于指定应用到资源的额外元数据。	N/A	N/A
kiali: install: service: metadata: annotations:	用于指定可应用到组件的服务的额外注解。	字符串	N/A
kiali: install: service: metadata: labels:	用于指定要应用到组件的服务的额外标签。	字符串	N/A
kiali: install: service: ingress:	用于指定通过 OpenShift Route 访问组件的服务的详细信息。	N/A	N/A
kiali: install: service: ingress: metadata: annotations:	用于指定可应用到组件的服务入口的额外注解。	字符串	N/A

参数	描述	值	默认值

kiali:
  install:
    service:
      ingress:
        metadata:
          labels:
	用于指定可应用到组件的服务入口的额外标签。	字符串	N/A

kiali:
  install:
    service:
      ingress:
        enabled:
	用于为与组件关联的服务自定义 OpenShift Route。	**true**/**false**	**true**

kiali:
  install:
    service:
      ingress:
        contextPath:
	用于指定服务的上下文路径。	字符串	N/A

install:
  service:
    ingress:
      hosts:
	用于为每个 OpenShift 路由指定一个主机名。空主机名表示路由的默认主机名。	字符串	N/A

install:
  service:
    ingress:
      tls:
	用于为 OpenShift 路由配置 TLS。		N/A

kiali:
  install:
    service:
      nodePort:
	用于为组件的服务 **Values.<component>.service.nodePort.port** 指定 **nodePort**	整数	N/A

1.25.2. 在 Kiali 自定义资源中指定 Kiali 配置

您可以通过在 Kiali 自定义资源(CR)而不是 ServiceMeshControlPlane (SMCP)资源中配置 Kiali 部署来完全自定义 Kiali 部署。这个配置有时被称为 "外部 Kiali"，因为配置是在 SMCP 之外指定的。

注意

您必须在同一命名空间中部署 ServiceMeshControlPlane 和 Kiali 自定义资源。例如： istio-system。

您可以配置和部署 Kiali 实例，然后将 Kiali 资源的 name 指定为 SMCP 资源中的 spec.addons.kiali.name 的值。如果存在与 name 值匹配的 Kiali CR，Service Mesh control plane 将使用现有安装。这个方法可让您完全自定义 Kiali 配置。

1.26. JAEGER 配置参考

当 Service Mesh Operator 部署 ServiceMeshControlPlane 资源时，它还可以为分布式追踪创建资源。Service Mesh 使用 Jaeger 进行分布式追踪。

重要

- Jaeger 不使用经 FIPS 验证的加密模块。
- 从 Red Hat OpenShift Service Mesh 2.5 开始，Red Hat OpenShift distributed Tracing Platform (Jaeger)已被弃用，并将在以后的发行版本中删除。红帽将在当前发行生命周期中提供对这个功能的程序漏洞修复和支持，但这个功能将不再获得改进，并将被删除。作为 Red Hat OpenShift distributed Tracing Platform (Jaeger)的替代选择，您可以使用 Red Hat OpenShift distributed Tracing Platform。

1.26.1. 启用和禁用追踪

您可以通过在 ServiceMeshControlPlane 资源中指定追踪类型和抽样率来启用分布式追踪。

默认的 all-in-one Jaeger 参数

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
```

```
tracing:
  sampling: 100
  type: Jaeger
```

在 Red Hat OpenShift Service Mesh 2.6 中，追踪类型 Jaeger 已弃用，并会被默认禁用。

在 Red Hat OpenShift Service Mesh 2.5 及更早版本中，追踪类型 Jaeger 会被默认启用。要禁用 Jaeger 追踪，将 ServiceMeshControlPlane 资源的 spec.tracing.type 参数设置为 None。

抽样率决定了 Envoy 代理生成 trace 的频率。您可以使用抽样率选项来控制向追踪系统报告的请求百分比。您可以根据网格中的流量以及您要收集的追踪数据量来配置此设置。您可以将 sampling 配置为一个缩放整数，代表 0.01% 增长。例如，将值设置为 10 会抽样 0.1% trace，将值设置为 500 代表抽样 5% trace，设置为 10000 代表抽样 100% trace。

注意

SMCP 抽样配置选项控制 Envoy 抽样率。您可以在 Jaeger 自定义资源中配置 Jaeger 追踪抽样率。

1.26.2. 在 SMCP 中指定 Jaeger 配置

您可以在 ServiceMeshControlPlane 资源的 addons 部分配置 Jaeger。但是，您可以在 SMCP 中配置的内容有一些限制。

当 SMCP 将配置信息传递给 Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 时，它会触发三种部署策略之一： allInOne、production 或 streaming。

1.26.3. 部署分布式追踪平台

分布式追踪平台(Jaeger)具有预定义的部署策略。您可以在 Jaeger 自定义资源 (CR) 文件中指定部署策略。当您创建分布式跟踪平台(Jaeger)的实例时，Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 会使用此配置文件创建部署所需的对象。

Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 目前支持以下部署策略：

- allInOne（默认）- 此策略旨在用于开发、测试和演示目的，它不用于生产环境。主要的后端组件 Agent、Collector 和 Query 服务都打包成单个可执行文件，（默认）配置为使用内存存储。您可以在 SMCP 中配置此部署策略。

  注意

  内存存储不是持久性的，这意味着如果 Jaeger 实例关闭、重启或被替换，您的 trace 数据将会丢失。此外，内存存储无法扩展，因为每个 Pod 都有自己的内存。对于持久性存储，您必须使用 production 或 streaming 策略，这些策略使用 Elasticsearch 作为默认存储。

- production - production 策略主要用于生产环境，在生产环境中，对 trace 数据进行长期存储非常重要，同时需要更容易扩展和高度可用的构架。因此，每个后端组件都会单独部署。Agent 可以作为检测应用程序上的 sidecar 注入。Query 和 Collector 服务被配置为使用一个受支持的存储类型 - 当前为 Elasticsearch。可以根据性能和恢复能力的需要提供每个组件的多个实例。您可以在 SMCP 中配置此部署策略，但为了完全自定义，您必须在 Jaeger CR 中指定您的配置，并链接到 SMCP。

- streaming - streaming 策略旨在通过提供 Collector 和 Elasticsearch 后端存储之间的流功能来增强 production 策略。这样做的好处是在高负载情况下降低后端存储压力，并允许其他 trace 后处理功能直接从流传输平台 (AMQ Streams/ Kafka) 中利用实时 span 数据。您无法在 SMCP 中配置此部署策略；您必须配置 Jaeger CR 并链接到 SMCP。

注意

streaming 策略需要额外的 AMQ Streams 订阅。

1.26.3.1. 默认分布式跟踪平台(Jaeger)部署

如果没有指定 Jaeger 配置选项，ServiceMeshControlPlane 资源将默认使用 allInOne Jaeger 部署策略。使用默认的 allInOne 部署策略时，请将 spec.addons.jaeger.install.storage.type 设置为 Memory。您可接受默认选项，也可以在 install 中指定附加配置选项。

control plane 默认 Jaeger 参数(Memory)

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
  tracing:
    sampling: 10000
    type: Jaeger
  addons:
    jaeger:
      name: jaeger
      install:
        storage:
          type: Memory
```

1.26.3.2. 生产环境分布式跟踪平台(Jaeger)部署（最小）

要使用 production 部署策略的默认设置，请将 spec.addons.jaeger.install.storage.type 设置为 Elasticsearch，并在 install 中指定额外的配置选项。请注意，SMCP 只支持配置 Elasticsearch 资源和镜像名称。

control plane 默认 Jaeger 参数 (Elasticsearch)

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
  tracing:
    sampling: 10000
    type: Jaeger
  addons:
    jaeger:
      name: jaeger  #name of Jaeger CR
      install:
        storage:
          type: Elasticsearch
        ingress:
          enabled: true
  runtime:
    components:
```

```
    tracing.jaeger.elasticsearch: # only supports resources and image name
     container:
      resources: {}
```

1.26.3.3. 生产环境分布式跟踪平台(Jaeger)部署（完全自定义）

SMCP 仅支持最小的 Elasticsearch 参数。要完全自定义生产环境并访问所有 Elasticsearch 配置参数，请使用 Jaeger 自定义资源 (CR) 来配置 Jaeger。

创建并配置 Jaeger 实例，并将 spec.addons.jaeger.name 设置为 Jaeger 实例的名称，在本例中为 MyJaegerInstance。

带有链接 Jaeger production CR 的 control plane

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
  tracing:
    sampling: 1000
    type: Jaeger
  addons:
    jaeger:
      name: MyJaegerInstance #name of Jaeger CR
      install:
        storage:
          type: Elasticsearch
        ingress:
          enabled: true
```

1.26.3.4. 流 Jaeger 部署

要使用 streaming 部署策略，请首先创建和配置 Jaeger 实例，然后将 spec.addons.jaeger.name 设置为 Jaeger 实例的名称，在本例中为 MyJaegerInstance。

带有链接 Jaeger streaming CR 的 control plane

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
  tracing:
    sampling: 1000
    type: Jaeger
  addons:
    jaeger:
      name: MyJaegerInstance  #name of Jaeger CR
```

1.26.4. 在 Jaeger 自定义资源中指定 Jaeger 配置

您可以通过在 Jaeger 自定义资源 (CR) 中而不是 ServiceMeshControlPlane (SMCP) 资源中配置 Jaeger 来完全自定义 Jaeger 部署。此配置有时被称为 "外部 Jaeger"，因为配置是在 SMCP 之外指定的。

> **注意**
>
> 您必须在同一命名空间中部署 SMCP 和 Jaeger CR。例如： istio-system。

您可以配置和部署独立 Jaeger 实例，然后将 Jaeger 资源的 name 指定为 SMCP 资源中的 spec.addons.jaeger.name 的值。如果存在与 name 值匹配的 Jaeger CR，Service Mesh control plane 将使用现有安装。这种方法可让您完全自定义 Jaeger 配置。

1.26.4.1. 部署最佳实践

- Red Hat OpenShift distributed Tracing Platform 实例名称必须是唯一的。如果要有多个 Red Hat OpenShift distributed Tracing Platform (Jaeger)实例，且正在使用 sidecar 注入的代理，则 Red Hat OpenShift distributed Tracing Platform (Jaeger)实例应该具有唯一的名称，注入注解应明确指定 Red Hat OpenShift distributed Tracing Platform (Jaeger)实例，应该报告追踪数据。

- 如果您有多租户实现，且租户由命名空间分开，请将 Red Hat OpenShift distributed Tracing Platform (Jaeger)实例部署到每个租户命名空间中。

1.26.4.2. 为服务网格配置分布式追踪安全性

分布式追踪平台(Jaeger)使用 OAuth 进行默认身份验证。但是，Red Hat OpenShift Service Mesh 使用名为 htpasswd 的 secret 来实现依赖服务（如 Grafana、Kiali 和分布式 Tracing Platform (Jaeger)之间的通信）。当您在 ServiceMeshControlPlane 中配置分布式跟踪平台(Jaeger)时，Service Mesh 会自动配置安全设置以使用 htpasswd。

如果要在 Jaeger 自定义资源中指定分布式跟踪平台(Jaeger)配置，您必须手动配置 htpasswd 设置，并确保 htpasswd secret 挂载到 Jaeger 实例中，以便 Kiali 能够与其通信。

1.26.4.2.1. 从 web 控制台为服务网格配置分布式追踪安全性

您可以修改 Jaeger 资源来配置分布式 Tracing Platform (Jaeger)安全性，以便在 web 控制台中用于 Service Mesh。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 dedicated-admin 角色的帐户。
- 必须安装 Red Hat OpenShift Service Mesh Operator。
- 部署到集群的 ServiceMeshControlPlane。
- 您可以访问 Red Hat OpenShift Service on AWS Web 控制台。

流程

1. 以具有 cluster-admin 角色的用户身份登录到 Red Hat OpenShift Service on AWS 控制台。

2. 导航到 **Operators** → **Installed Operators**。

3. 点 **Project** 菜单，从列表中选择部署 `ServiceMeshControlPlane` 资源的项目，如 **istio-system**。

4. 点 **Red Hat OpenShift Distributed Tracing Platform (Jaeger) Operator**。

5. 在 **Operator Details** 页面中，点 **Jaeger** 标签页。

6. 点 Jaeger 实例的名称。

7. 在 Jaeger 详情页面上，点 **YAML** 选项卡修改您的配置。

8. 编辑 **Jaeger** 自定义资源文件，以添加 **htpasswd** 配置，如下例所示。

   - **spec.ingress.openshift.htpasswdFile**

   - **spec.volumes**

   - **spec.volumeMounts**

显示 htpasswd 配置的 Jaeger 资源示例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
spec:
  ingress:
    enabled: true
    openshift:
      htpasswdFile: /etc/proxy/htpasswd/auth
      sar: '{"namespace": "istio-system", "resource": "pods", "verb": "get"}'
    options: {}
```

```
    resources: {}
    security: oauth-proxy
  volumes:
    - name: secret-htpasswd
      secret:
        secretName: htpasswd
    - configMap:
        defaultMode: 420
        items:
          - key: ca-bundle.crt
            path: tls-ca-bundle.pem
        name: trusted-ca-bundle
        optional: true
      name: trusted-ca-bundle
  volumeMounts:
    - mountPath: /etc/proxy/htpasswd
      name: secret-htpasswd
    - mountPath: /etc/pki/ca-trust/extracted/pem/
      name: trusted-ca-bundle
      readOnly: true
# ...
```

9. 点 **Save**。

1.26.4.2.2. 使用命令行为服务网格配置分布式追踪安全性

您可以通过运行 OpenShift CLI (oc)来修改 Jaeger 资源，以配置分布式 Tracing Platform (Jaeger) 安全性，以便从命令行用于 Service Mesh。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 dedicated-admin 角色的帐户。
- 必须安装 Red Hat OpenShift Service Mesh Operator。
- 部署到集群的 ServiceMeshControlPlane。

- 您可以访问与 Red Hat OpenShift Service on AWS 版本匹配的 OpenShift CLI (oc)。

流程

1. 运行以下命令，以具有 cluster-admin 角色的用户身份登录 OpenShift CLI (oc)。如果使用 Red Hat OpenShift Dedicated，则必须有一个具有 dedicated-admin 角色的帐户。

```
$ oc login https://<HOSTNAME>:6443
```

2. 输入以下命令来更改安装 control plane 的项目，如 istio-system ：

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

3. 运行以下命令来编辑 Jaeger 自定义资源文件：

```
$ oc edit -n openshift-distributed-tracing -f jaeger.yaml
```

4. 编辑 Jaeger 自定义资源文件，以添加 htpasswd 配置，如下例所示。

   - spec.ingress.openshift.htpasswdFile
   - spec.volumes
   - spec.volumeMounts

   显示 htpasswd 配置的 Jaeger 资源示例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
spec:
  ingress:
    enabled: true
    openshift:
      htpasswdFile: /etc/proxy/htpasswd/auth
```

```
      sar: '{"namespace": "istio-system", "resource": "pods", "verb": "get"}'
    options: {}
    resources: {}
    security: oauth-proxy
  volumes:
    - name: secret-htpasswd
      secret:
        secretName: htpasswd
    - configMap:
        defaultMode: 420
        items:
          - key: ca-bundle.crt
            path: tls-ca-bundle.pem
        name: trusted-ca-bundle
        optional: true
      name: trusted-ca-bundle
  volumeMounts:
    - mountPath: /etc/proxy/htpasswd
      name: secret-htpasswd
    - mountPath: /etc/pki/ca-trust/extracted/pem/
      name: trusted-ca-bundle
      readOnly: true
```

5. 运行以下命令来监控 pod 部署的进度：

```
$ oc get pods -n openshift-distributed-tracing
```

1.26.4.3. 分布式追踪默认配置选项

Jaeger 自定义资源(CR)定义创建分布式 Tracing Platform (Jaeger)资源时要使用的架构和设置。您可以修改这些参数以根据您的业务需求自定义分布式 Tracing Platform (Jaeger)实施。

Jaeger CR 的通用 YAML 示例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: name
spec:
  strategy: <deployment_strategy>
  allInOne:
    options: {}
    resources: {}
  agent:
```

```
    options: {}
    resources: {}
  collector:
    options: {}
    resources: {}
  sampling:
    options: {}
  storage:
    type:
    options: {}
  query:
    options: {}
    resources: {}
  ingester:
    options: {}
    resources: {}
  options: {}
```

表 1.39. Jaeger 参数

参数	描述	值	默认值
apiVersion：	创建对象时要使用的 API 版本。	**jaegertracing.io/v1**	**jaegertracing.io/v1**
kind：	定义要创建的 Kubernetes 对象的种类。	**jaeger**	
metadata：	有助于唯一标识对象的数据，包括 **name** 字符串、**UID** 和可选 **namespace**。		Red Hat OpenShift Service on AWS 会自动生成 **UID** 并使用创建对象的项目名称完成 **namespace**。
name：	对象的名称。	分布式追踪平台(Jaeger)实例的名称。	**jaeger-all-in-one-inmemory**
spec：	要创建的对象的规格。	包含分布式跟踪平台(Jaeger)实例的所有配置参数。当需要所有 Jaeger 组件的通用定义时，会在 **spec** 节点下定义它。当该定义与单个组件相关时，它将放置在 **spec/<component>** 节点下。	N/A

参数	描述	值	默认值
strategy：	Jaeger 部署策略	**allInOne**、**production** 或 **streaming**	**allInOne**
allInOne：	因为 **allInOne** 镜像在单个 pod 中部署了 Agent、Collector、Query、Ingester 和 Jaeger UI，所以此部署的配置必须在 **allInOne** 参数下嵌套组件配置。		
agent：	定义代理的配置选项。		
collector：	定义 Jaeger Collector 的配置选项。		
sampling：	定义用于追踪的抽样策略的配置选项。		
storage：	定义存储的配置选项。所有与存储相关的选项都必须放在 **storage** 下，而不是放在 **allInOne** 或其他组件选项下。		
query：	定义 Query 服务的配置选项。		
ingester：	定义 Ingester 服务的配置选项。		

以下示例 YAML 是使用默认设置创建 Red Hat OpenShift distributed Tracing Platform (Jaeger)部署的最低要求。

最低要求示例 dist-tracing-all-in-one.yaml

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-all-in-one-inmemory
```

1.26.4.4. Jaeger Collector 配置选项

Jaeger Collector 组件负责接收 tracer 捕获的 span，在使用 production 策略时将其写入持久性 Elasticsearch 存储，或使用 streaming 策略时将其写入 AMQ Streams。

Collector 是无状态的，因此很多 Jaeger Collector 实例可以并行运行。除了 Elasticsearch 集群的位置，收集器几乎不需要任何配置。

表 1.40. Operator 用来定义 Jaeger Collector 的参数

参数	描述	值
`collector:` `  replicas:`	指定要创建的 Collector 副本数。	整数，如 **5**。

表 1.41. 传递给 Collector 的配置参数

参数	描述	值

`spec:`
` collector:`
`  options: {}`
	定义 Jaeger Collector 的配置选项。	

`options:`
`  collector:`
`    num-workers:`
	从队列中拉取的 worker 数量。	整数，如 **50**。

`options:`
`  collector:`
`    queue-size:`
	Collector 队列的大小。	整数，如 **2000**。

`options:`
`  kafka:`
`    producer:`
`      topic: jaeger-spans`
	topic 参数标识收集器用来生成消息的 Kafka 配置以及要使用消息的 ingester。	producer 的标签。

参数	描述	值

`options:`
`  kafka:`
`    producer:`
`      brokers: my-cluster-kafka-brokers.kafka:9092`
	标识 Collector 用来生成消息的 Kafka 配置。如果没有指定代理，并且安装了 AMQ Streams 1.4.0+，Red Hat OpenShift Distributed Tracing Platform (Jaeger) Operator 将自助置备 Kafka。	

`options:` `  log-level:`	Collector 的日志记录级别。	可能的值有：**debug**、**info**、**warn**、**error**、**fatal**、**panic**。

`options:`
`  otlp:`
`    enabled: true`
`    grpc:`
`      host-port: 4317`
`      max-connection-age: 0s`
`      max-connection-age-grace: 0s`
`      max-message-size: 4194304`
`      tls:`
`        enabled: false`
`        cert: /path/to/cert.crt`
`        cipher-suites: "TLS_AES_256_GCM_SHA384,TLS_CHACHA20_POLY1305_SHA256"`
`        client-ca: /path/to/cert.ca`
`        reload-interval: 0s`
`        min-version: 1.2`
`        max-version: 1.3`
	要接受 OTLP/gRPC，请明确启用 **otlp**。所有其他选项都是可选的。	

参数	描述	值

options:
  otlp:
    enabled: true
    http:
      cors:
        allowed-headers: [<header-name>[, <header-name>]*]
        allowed-origins: *
      host-port: 4318
      max-connection-age: 0s
      max-connection-age-grace: 0s
      max-message-size: 4194304
      read-timeout: 0s
      read-header-timeout: 2s
      idle-timeout: 0s
      tls:
        enabled: false
        cert: /path/to/cert.crt
        cipher-suites: "TLS_AES_256_GCM_SHA384,TLS_CHACHA20_POLY1305_SHA256"
        client-ca: /path/to/cert.ca
        reload-interval: 0s
        min-version: 1.2
        max-version: 1.3
	要接受 OTLP/HTTP，请明确启用 **otlp**。所有其他选项都是可选的。	

1.26.4.5. 分布式追踪抽样配置选项

Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 可用于定义抽样策略，以提供给配置为使用远程 sampler 的 tracer。

虽然生成了所有 trace，但只有几个会被抽样。对某个 trace 进行抽样会标记该 trace 用于进一步处理和存储。

注意

如果一个 trace 由 Envoy 代理启动，则不会相关，因为抽样决定是在那里做出的。只有在应用程序使用客户端启动 trace 时，Jaeger 抽样决定才相关。

当服务收到不包含 trace 上下文的请求时，客户端会启动新的 trace，为它分配一个随机 trace ID，并根据当前安装的抽样策略做出抽样决定。抽样决定被传播到 trace 中的所有后续请求，以便其他服务不会再次做出抽样决定。

分布式追踪平台(Jaeger)库支持以下抽样：

- Probabilistic（概率） - sampler 做出一个随机抽样决定，其抽样的概率等于 sampling.param 属性的值。例如，使用 sampling.param=0.1 代表大约为 10 个 trace 抽样 1 次。

- Rate Limiting（速率限制） - sampler 使用泄漏存储桶速率限制器来确保 trace 使用某种恒定速率进行抽样。例如，使用 sampling.param=2.0 抽样请求，速率为每秒 2 个 trace。

表 1.42. Jaeger 抽样选项

参数	描述	值	默认值
spec: sampling: options: {} default_strategy: service_strategy:	定义用于追踪的抽样策略的配置选项。		如果没有提供配置，Collector 会返回默认的概率抽样策略，所有服务都为 0.001(0.1%)概率。
default_strategy: type: service_strategy: type:	要使用的抽样策略。请参阅上述描述。	有效值是 **probabilistic** 和 **ratelimiting**。	**probabilistic**
default_strategy: param: service_strategy: param:	所选抽样策略的参数。	小数值和整数值 (0, .1, 1, 10)	1

这个示例定义了一种概率性的默认抽样策略，trace 实例被抽样的几率为 50%。

概率抽样示例

```yaml
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: with-sampling
spec:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 0.5
      service_strategies:
        - service: alpha
          type: probabilistic
          param: 0.8
          operation_strategies:
            - operation: op1
              type: probabilistic
              param: 0.2
            - operation: op2
              type: probabilistic
              param: 0.4
        - service: beta
          type: ratelimiting
          param: 5
```

如果没有用户提供的配置，分布式 Tracing Platform (Jaeger)将使用以下设置：

默认抽样

```yaml
spec:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 1
```

1.26.4.6. 分布式追踪存储配置选项

您可以在 spec.storage 下为 Collector、Ingester 和 Query 服务配置存储。可以根据性能和恢复能力的需要提供每个组件的多个实例。

表 1.43. Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 用来定义分布式追踪存储的一般存储参数

参数	描述	值	默认值

`spec:`
`  storage:`
`    type:`
	要在部署中使用的存储类型。	**memory** 或 **elasticsearch**。内存存储仅适用于开发、测试、演示和验证概念环境，因在关闭 pod 时，数据不会保留。对于生产环境，分布式跟踪平台(Jaeger) 支持 Elasticsearch 进行持久性存储。	**memory**

`storage:` `  secretname:`	secret 名称，例如 **tracing-secret**。		N/A
`storage:` `  options: {}`	定义存储的配置选项。		

表 1.44. Elasticsearch 索引清理参数

参数	描述	值	默认值

`storage:`
`  esIndexCleaner:`
`    enabled:`
	当使用 Elasticsearch 存储时，默认会创建一个任务来清理索引中的旧 trace。这个参数用于启用或禁用索引清理任务。	**true/ false**	**true**

`storage:`
`  esIndexCleaner:`
`    numberOfDays:`
	删除索引前等待的天数。	整数值	**7**

`storage:`
`  esIndexCleaner:`
`    schedule:`
	为 Elasticsearch 索引的清理频率定义调度。	Cron 表达式	"55 23 * * *"

1.26.4.6.1. 自动置备 Elasticsearch 实例

部署 Jaeger 自定义资源时，Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 使用 OpenShift Elasticsearch Operator 根据自定义资源文件的 storage 部分中提供的配置创建一个 Elasticsearch 集群。如果设置了以下配置，Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 将置备 Elasticsearch：

- spec.storage:type 设置为 elasticsearch
- spec.storage.elasticsearch.doNotProvision 设置为 false
- spec.storage.options.es.server-urls 没有被定义，即没有连接到 OpenShift Elasticsearch Operator 未置备的 Elasticsearch 实例。

置备 Elasticsearch 时，Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 会将 Elasticsearch 自定义资源名称设置为 Jaeger 自定义资源的 spec.storage.elasticsearch. name 的值。如果没有为 spec.storage.elasticsearch.name 指定一个值，Operator 会使用 elasticsearch。

限制

- 每个命名空间只能有一个具有自助置备 Elasticsearch 实例的分布式跟踪平台(Jaeger)。Elasticsearch 集群旨在专用于单个分布式 Tracing Platform (Jaeger)实例。
- 每个命名空间只能有一个 Elasticsearch。

注意

如果您已经安装了 Elasticsearch 作为 OpenShift Logging 的一部分，Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 可以使用已安装的 OpenShift Elasticsearch Operator 来置备存储。

以下配置参数用于一个 *自置备的* Elasticsearch 实例，这是由 Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 使用 OpenShift Elasticsearch Operator 创建的实例。在配置文件中，您可以在 spec:storage:elasticsearch 下为自助置备 Elasticsearch 指定配置选项。

表 1.45. Elasticsearch 资源配置参数

参数	描述	值	默认值
elasticsearch: properties: doNotProvision:	用于指定 Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 是否应该置备 Elasticsearch 实例。	**true/false**	**true**
elasticsearch: properties: name:	Elasticsearch 实例的名称。Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 使用此参数中指定的 Elasticsearch 实例连接到 Elasticsearch。	string	**elasticsearch**
elasticsearch: nodeCount:	Elasticsearch 节点数量。对于高可用性，需要至少 3 个节点。不要只使用 2 个节点，因为可能会出现"脑裂"问题。	整数值。例如，概念验证 = 1，最小部署 = 3	3
elasticsearch: resources: requests: cpu:	根据您的环境配置，请求的 CPU 数量。	以内核数或 millicores 指定，例如 200m, 0.5, 1。例如，概念证明 = 500m，最小部署 =1	1
elasticsearch: resources: requests: memory:	根据您的环境配置，可用于请求的内存。	以字节为单位指定，例如 200Ki, 50Mi, 5Gi。例如，概念证明 = 1Gi，最小部署 = 16Gi*	16Gi
elasticsearch: resources: limits: cpu:	根据您的环境配置，CPU 数量的限值。	以内核数或 millicores 指定，例如 200m, 0.5, 1。例如，概念证明 = 500m，最小部署 =1	
elasticsearch: resources: limits: memory:	根据您的环境配置，可用的内存限值。	以字节为单位指定，例如 200Ki, 50Mi, 5Gi。例如，概念证明 = 1Gi，最小部署 = 16Gi*	

参数	描述	值	默认值
elasticsearch: redundancyPolicy:	数据复制策略定义如何在集群中的数据节点之间复制 Elasticsearch 分片：如果没有指定，Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 会自动根据节点数量决定最合适的复制。	**ZeroRedundancy**（无副本分片）、**SingleRedundancy**（一个副本分片）、**MultipleRedundancy**（每个索引分散于一半的 Data 节点）、**FullRedundancy**（每个索引在集群中的每个 Data 节点上完全复制）。	
elasticsearch: useCertManagement:	用于指定分布式跟踪平台 (Jaeger)是否应使用 OpenShift Elasticsearch Operator 的证书管理功能。此功能被添加到 Red Hat OpenShift Service on AWS 4.7 中的 {logging-title} 5.2 中，它是新 Jaeger 部署的首选设置。	**true/false**	**true**

通过这个设置可以使每个 Elasticsearch 节点使用较低内存进行操作，但对于生产环境部署，不建议这样做。对于生产环境，您应该默认为每个 pod 分配不少于 16Gi 内存，但最好为每个 pod 最多分配 64Gi 内存。

生产环境存储示例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 3
      resources:
        requests:
          cpu: 1
          memory: 16Gi
        limits:
          memory: 16Gi
```

具有持久性存储的存储示例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 1
      storage: # 1
        storageClassName: gp2
        size: 5Gi
      resources:
        requests:
          cpu: 200m
          memory: 4Gi
        limits:
          memory: 4Gi
      redundancyPolicy: ZeroRedundancy
```

1 持久性存储配置。在本例中，AWS gp2 的大小为 5Gi。如果没有指定值，分布式跟踪平台 (Jaeger)将使用 emptyDir。OpenShift Elasticsearch Operator 置备 PersistentVolumeClaim 和 PersistentVolume，它们不会在分布式 Tracing Platform (Jaeger)实例中删除。如果您创建具有相同名称和命名空间的分布式 Tracing Platform (Jaeger)实例，则可以挂载同一卷。

1.26.4.6.2. 连接到现有 Elasticsearch 实例

您可以使用现有 Elasticsearch 集群进行分布式跟踪平台存储。现有的 Elasticsearch 集群（也称为*外部* Elasticsearch 实例）是由 Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 或 OpenShift Elasticsearch Operator 安装的实例。

部署 Jaeger 自定义资源时，如果设置了以下配置，Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 不会置备 Elasticsearch：

- **spec.storage.elasticsearch.doNotProvision** 设置为 **true**

- **spec.storage.options.es.server-urls** 有一个值

- **spec.storage.elasticsearch.name** 具有一个值，或者 Elasticsearch 实例名称是 **elasticsearch**。

Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 使用 **spec.storage.elasticsearch.name** 中指定的 Elasticsearch 实例连接到 Elasticsearch。

限制

- 您不能将 Red Hat OpenShift Service on AWS 日志记录 Elasticsearch 实例与分布式 Tracing Platform (Jaeger)共享或重复使用。Elasticsearch 集群旨在专用于单个分布式 Tracing Platform (Jaeger)实例。

以下配置参数适用于已经存在的 Elasticsearch 实例，也称为*外部* Elasticsearch 实例。在本例中，您可以在自定义资源文件中的 **spec:storage:options:es** 下为 Elasticsearch 指定配置选项。

表 1.46. 常规 ES 配置参数

参数	描述	值	默认值
es: server-urls:	Elasticsearch 实例的 URL。	Elasticsearch 服务器的完全限定域名。	http://elasticsearch.<namespace>.svc:9200
es: max-doc-count:	从 Elasticsearch 查询返回的最大文档数量。这也适用于聚合。如果同时设置了 **es.max-doc-count** 和 **es.max-num-spans**，Elasticsearch 将使用两者中的较小的值。		10000

参数	描述	值	默认值
es: max-num-spans:	[**已弃用** - 将在以后的版本中删除，使用 **es.max-doc-count** 代替。] 在 Elasticsearch 中每个查询每次抓取的最大 span 数量。如果同时设置了 **es.max-num-spans** 和 **es.max-doc-count**，Elasticsearch 将使用两者中的较小的值。		10000
es: max-span-age:	Elasticsearch 中 span 的最大查询。		72h0m0s
es: sniffer:	Elasticsearch 的侦察器配置。客户端使用侦察过程自动查找所有节点。默认禁用此选项。	**true**/ **false**	**false**
es: sniffer-tls-enabled:	在监控 Elasticsearch 集群时启用 TLS 的选项。客户端使用侦察过程自动查找所有节点。默认禁用	**true**/ **false**	**false**
es: timeout:	用于查询的超时。当设为零时，则没有超时。		0s
es: username:	Elasticsearch 所需的用户名。如果指定，基本身份验证也会加载 CA。另请参阅 **es.password**。		
es: password:	Elasticsearch 所需的密码。另请参阅 **es.username**。		
es: version:	主要的 Elasticsearch 版本。如果没有指定，则该值将从 Elasticsearch 中自动探测到。		0

表 1.47. ES 数据复制参数

参数	描述	值	默认值
es: num-replicas:	Elasticsearch 中每个索引的副本数。		1
es: num-shards:	Elasticsearch 中每个索引的分片数量。		5

表 1.48. ES 索引配置参数

参数	描述	值	默认值
es: create-index-templates:	设置为 **true** 时，应用程序启动时自动创建索引模板。手动安装模板时，设置为 **false**。	**true**/ **false**	**true**
es: index-prefix:	分布式追踪平台(Jaeger)索引的可选前缀。例如，将其设置为 "production" 会创建名为 "production-tracing-*" 的索引。		

表 1.49. ES 批量处理器配置参数

参数	描述	值	默认值
es: bulk: actions:	在批量处理器决定向磁盘提交更新前可添加到队列的请求数。		1000
es: bulk: flush-interval:	**提交批量请求的时间**. 要禁用批量处理器清除间隔，请将其设置为零。		200ms
es: bulk: size:	在批量处理器决定提交更新之前，批量请求可以处理的字节数。		5000000

参数	描述	值	默认值

es:
  bulk:
    workers:
	可以接收并将批量请求提交 Elasticsearch 的 worker 数量。		1

表 1.50. ES TLS 配置参数

参数	描述	值	默认值

es:
  tls:
    ca:
	用于验证远程服务器的 TLS 证书颁发机构(CA)文件的路径。		默认将使用系统信任存储。

es:
  tls:
    cert:
	TLS 证书文件的路径，用来识别此进程到远程服务器。		

es:
  tls:
    enabled:
	与远程服务器对话时启用传输层安全(TLS)。默认禁用此选项。	**true/ false**	**false**

es:
  tls:
    key:
	TLS 私钥文件的路径，用来识别此进程到远程服务器。		

es:
  tls:
    server-name:
	覆盖远程服务器证书中预期的 TLS 服务器名称。		

es: token-file:	包含 bearer 令牌的文件的路径。如果指定该标志，该标志也会载入认证机构（CA）文件。		

表 1.51. ES 归档配置参数

参数	描述	值	默认值

es-archive:
  bulk:
    actions:
	在批量处理器决定向磁盘提交更新前可添加到队列的请求数。		0

es-archive:
  bulk:
    flush-interval:
	提交批量请求的时间. 要禁用批量处理器清除间隔，请将其设置为零。		0s

es-archive:
  bulk:
    size:
	在批量处理器决定提交更新之前，批量请求可以处理的字节数。		0

es-archive:
  bulk:
    workers:
	可以接收并将批量请求提交 Elasticsearch 的 worker 数量。		0

es-archive: create-index-templates:	设置为 **true** 时，应用程序启动时自动创建索引模板。手动安装模板时，设置为 **false**。	**true**/ **false**	**false**
es-archive: enabled:	启用额外的存储。	**true**/ **false**	**false**
es-archive: index-prefix:	分布式追踪平台(Jaeger)索引的可选前缀。例如，将其设置为 "production" 会创建名为 "production-tracing-*" 的索引。		
es-archive: max-doc-count:	从 Elasticsearch 查询返回的最大文档数量。这也适用于聚合。		0
es-archive: max-num-spans:	[**已弃用** - 将在以后的版本中删除，使用 **es-archive.max-doc-count** 替代。] Elasticsearch 中的每个查询一次获取的最大 span 数量。		0

参数	描述	值	默认值
es-archive: max-span-age:	Elasticsearch 中 span 的最大查询。		0s
es-archive: num-replicas:	Elasticsearch 中每个索引的副本数。		0
es-archive: num-shards:	Elasticsearch 中每个索引的分片数量。		0
es-archive: password:	Elasticsearch 所需的密码。另请参阅 **es.username**。		
es-archive: server-urls:	以逗号分隔的 Elasticsearch 服务器列表。必须指定为完全限定的 URL，例如 **http://localhost:9200**。		
es-archive: sniffer:	Elasticsearch 的侦察器配置。客户端使用侦察过程自动查找所有节点。默认禁用此选项。	**true**/ **false**	**false**
es-archive: sniffer-tls-enabled:	在监控 Elasticsearch 集群时启用 TLS 的选项。客户端使用侦察过程自动查找所有节点。默认禁用此选项。	**true**/ **false**	**false**
es-archive: timeout:	用于查询的超时。当设为零时，则没有超时。		0s
es-archive: tls: ca:	用于验证远程服务器的 TLS 证书颁发机构(CA)文件的路径。		默认将使用系统信任存储。

参数	描述	值	默认值
es-archive: tls: cert:	TLS 证书文件的路径，用来识别此进程到远程服务器。		
es-archive: tls: enabled:	与远程服务器对话时启用传输层安全(TLS)。默认禁用此选项。	**true**/ **false**	**false**
es-archive: tls: key:	TLS 私钥文件的路径，用来识别此进程到远程服务器。		
es-archive: tls: server-name:	覆盖远程服务器证书中预期的 TLS 服务器名称。		
es-archive: token-file:	包含 bearer 令牌的文件的路径。如果指定该标志，该标志也会载入认证机构（CA）文件。		
es-archive: username:	Elasticsearch 所需的用户名。如果指定，基本身份验证也会加载 CA。请参阅 **es-archive.password**。		
es-archive: version:	主要的 Elasticsearch 版本。如果没有指定，则该值将从 Elasticsearch 中自动探测到。		0

使用卷挂载的存储示例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
```

```
      type: elasticsearch
      options:
        es:
          server-urls: https://quickstart-es-http.default.svc:9200
          index-prefix: my-prefix
          tls:
            ca: /es/certificates/ca.crt
      secretName: tracing-secret
    volumeMounts:
      - name: certificates
        mountPath: /es/certificates/
        readOnly: true
    volumes:
      - name: certificates
        secret:
          secretName: quickstart-es-http-certs-public
```

以下示例显示了使用从存储在 secret 中的卷和用户/密码挂载了 TLS CA 证书的外部 Elasticsearch 集群的 Jaeger CR。

外部 Elasticsearch 示例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200 1
        index-prefix: my-prefix
        tls: 2
          ca: /es/certificates/ca.crt
    secretName: tracing-secret 3
  volumeMounts: 4
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
  volumes:
    - name: certificates
      secret:
        secretName: quickstart-es-http-certs-public
```

1

在默认命名空间中运行的 Elasticsearch 服务 URL。

2

TLS 配置。在这种情况下，只有 CA 证书，但在使用 mutual TLS 时，它也可以包含 es.tls.key 和 es.tls.cert。

3

定义环境变量 ES_PASSWORD 和 ES_USERNAME 的 Secret。由 kubectl create secret generic tracing-secret --from-literal=ES_PASSWORD=changeme --from-literal=ES_USERNAME=elastic 创建

4

被挂载到所有存储组件的卷挂载和卷。

1.26.4.7. 查询配置选项

Query 是一个从存储中检索 trace 并托管用户界面来显示它们的服务。

表 1.52. Red Hat OpenShift distributed Tracing Platform (Jaeger) Operator 用来定义 Query 的参数

参数	描述	值	默认值

spec:
  query:
    replicas:
	指定要创建的 Query 副本数。	整数，如 2。	

表 1.53. 传递给 Query 的配置参数

参数	描述	值	默认值

spec:
  query:
    options: {}
	定义 Query 服务的配置选项。		

参数	描述	值	默认值
options: log-level:	Query 的日志记录级别。	可能的值有：**debug**、**info**、**warn**、**error**、**fatal**、**panic**。	

options:
  query:
    base-path:
	所有 jaeger-query HTTP 路由的基本路径都可设置为非 root 值，例如，**/jaeger** 将导致所有 UI URL 以 **/jaeger** 开头。当在反向代理后面运行 Jaeger-query 时，这很有用。	/<path>	

示例 Query 配置

```
apiVersion: jaegertracing.io/v1
kind: "Jaeger"
metadata:
  name: "my-jaeger"
spec:
  strategy: allInOne
  allInOne:
    options:
      log-level: debug
      query:
        base-path: /jaeger
```

1.26.4.8. Ingester 配置选项

Ingester 是一个从 Kafka 主题读取并写入 Elasticsearch 存储后端的服务。如果您使用 allInOne 或 production 部署策略，则不需要配置 Ingester 服务。

表 1.54. 传递给 Ingester 的 Jaeger 参数

参数	描述	值

`spec:`
`  ingester:`
`    options: {}`
	定义 Ingester 服务的配置选项。	

`options:` `  deadlockInterval:`	指定 Ingester 在终止前必须等待消息的时间间隔（以秒为单位）。默认情况下，死锁时间间隔被禁用（设置为 **0**），以避免在系统初始化过程中没有信息到达 Ingester。	分钟和秒，例如 **1m0s**。默认值为 **0**。

`options:`
`  kafka:`
`    consumer:`
`      topic:`
	topic 参数标识收集器用来生成消息的 Kafka 配置，并使用 Ingester。	consumer 的标签例如，**jaeger-spans**.

`options:`
`  kafka:`
`    consumer:`
`      brokers:`
	Ingester 用来使用消息的 Kafka 配置的标识。	代理的标签，如 **my-cluster-kafka-brokers.kafka:9092**。

`options:` `  log-level:`	Ingester 的日志记录级别。	可能的值有：**debug**,**info**,**warn**,**error**,**fatal**,**dpanic**,**panic**.

流传输 Collector 和 Ingester 示例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-streaming
spec:
  strategy: streaming
  collector:
    options:
      kafka:
        producer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
  ingester:
    options:
      kafka:
        consumer:
          topic: jaeger-spans
```

```
        brokers: my-cluster-kafka-brokers.kafka:9092
      ingester:
        deadlockInterval: 5
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: http://elasticsearch:9200
```

1.27. 卸载 SERVICE MESH

要从现有的 Red Hat OpenShift Service on AWS 实例卸载 Red Hat OpenShift Service Mesh 并删除其资源，您必须删除 control plane、删除 Operator，并运行命令来手动删除某些资源。

1.27.1. 删除 Red Hat OpenShift Service Mesh control plane

要从现有的 Red Hat OpenShift Service on AWS 实例卸载 Service Mesh，首先删除 Service Mesh control plane 和 Operator。然后，您将运行命令来删除剩余的资源。

1.27.1.1. 使用 Web 控制台删除 Service Mesh control plane

您可以使用 Web 控制台删除 Red Hat OpenShift Service Mesh control plane。

流程

1. 登陆到 Red Hat OpenShift Service on AWS web 控制台。

2. 点 Project 菜单，选择安装 Service Mesh control plane 的项目，如 istio-system。

3. 导航到 Operators → Installed Operators。

4. 点 Provided APIs 下的 Service Mesh Control Plane。

5.

点 ServiceMeshControlPlane 菜单

.

6. 点 Delete Service Mesh Control Plane。

7. 在确认窗口中点 Delete 删除 ServiceMeshControlPlane。

1.27.1.2. 使用 CLI 删除 Service Mesh control plane

您可以使用 CLI 删除 Red Hat OpenShift Service Mesh control plane。在本例中，istio-system 是 control plane 项目的名称。

流程

1. 登陆到 Red Hat OpenShift Service on AWS CLI。

2. 运行以下命令以删除 ServiceMeshMemberRoll 资源。

```
$ oc delete smmr -n istio-system default
```

3. 运行这个命令来获得安装的 ServiceMeshControlPlane 的名称：

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

4. 使用以上命令中的输出替换 <name_of_custom_resource>，运行这个命令来删除自定义资源：

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

1.27.2. 删除安装的 Operator

您必须删除 Operator 才可以成功删除 Red Hat OpenShift Service Mesh。删除 Red Hat OpenShift Service Mesh Operator 后，您必须删除 Kiali Operator、Red Hat OpenShift Distributed Tracing

Platform (Jaeger) Operator 和 OpenShift Elasticsearch Operator。

1.27.2.1. 删除 Operator

按照以下步骤删除组成 Red Hat OpenShift Service Mesh 的 Operator。对以下每个 Operator 重复上述步骤。

- Red Hat OpenShift Service Mesh
- Kiali
- Red Hat OpenShift distributed Tracing Platform (Jaeger)
- OpenShift Elasticsearch

流程

1. 登陆到 Red Hat OpenShift Service on AWS web 控制台。
2. 在 **Operators** → **Installed Operators** 页面中，滚动页面或在 **Filter by name** 中输入关键字以查找每个 Operator。然后点击 Operator 名称。
3. 在 **Operator Details** 页面中，从 **Actions** 菜单中选择 **Uninstall Operator**。按照提示卸载每个 Operator。

1.27.3. 清理 Operator 资源

您可以使用 Red Hat OpenShift Service on AWS Web 控制台手动删除 Red Hat OpenShift Service Mesh Operator 后保留的资源。

前提条件

- 具有集群管理访问权限的帐户。如果使用 Red Hat OpenShift Dedicated，则必须有一个具

有 dedicated-admin 角色的帐户。

- 访问 OpenShift CLI（oc）。

流程

1. 以集群管理员身份登录 Red Hat OpenShift Service on AWS CLI。

2. 在卸载 Operators 后运行以下命令清理资源。如果您要继续使用分布式跟踪平台(Jaeger)作为没有服务网格的独立服务，请不要删除 Jaeger 资源。

注意

OpenShift Elasticsearch Operator 默认安装在 openshift-operators-redhat 中。其他 Operator 默认安装在 openshift-operators 命名空间中。如果在另一个命名空间中安装了 Operator，将 openshift-operators 替换为安装了 Red Hat OpenShift Service Mesh Operator 的项目的名称。

```
$ oc delete svc maistra-admission-controller -n openshift-operators
```

```
$ oc -n openshift-operators delete ds -lmaistra-version
```

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

```
$ oc delete clusterrole istio-view istio-edit
```

```
$ oc delete clusterrole jaegers.jaegertracing.io-v1-admin jaegers.jaegertracing.io-v1-crdview jaegers.jaegertracing.io-v1-edit jaegers.jaegertracing.io-v1-view
```

```
$ oc delete cm -n openshift-operators maistra-operator-cabundle
```

```
$ oc delete cm -n openshift-operators istio-cni-config istio-cni-config-v2-3
```

```
$ oc delete sa -n openshift-operators istio-cni
```