OpenShift Container Platform 4.3

Service Mesh

Service Mesh 的安装、使用和发行注记信息

Last Updated: 2020-11-30

OpenShift Container Platform 4.3 Service Mesh

Service Mesh 的安装、使用和发行注记信息

法律通告

摘要

本文档提供了有关如何在 OpenShift Container Platform 中使用 Service Mesh 的信息

目录

第 1 章 SERVICE MESH 1.X

1.1. SERVICE MESH 发行注记

1.1.1. Red Hat OpenShift Service Mesh

Red Hat OpenShift Service Mesh 是一个提供对服务网格（service mesh）的行为信息和操作控制的平台，它为用户提供了一个连接、管理和监控微服务应用程序的统一方法。

术语 *服务网格（service mesh）*代表在分布式微服务架构中组成应用程序的微服务网络，以及这些微服务间的交互。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。

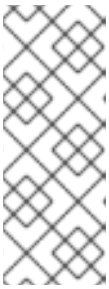
Red Hat OpenShift Service Mesh 基于开源 Istio 项目，它在不需要修改服务代码的情况下，为现有的分布式应用程序添加了一个透明的层。您可以在服务中添加对 Red Hat OpenShift Service Mesh 的支持，方法是将一个特殊的 sidecar 代理服务器部署到用于处理不同微服务之间的所有网络通讯的环境中。您可以使用 control plane 功能配置和管理 Service Mesh。

Red Hat OpenShift Service Mesh 提供了一个方便的方法来创建一个部署的服务网络，它可提供发现、负载平衡、服务对服务验证、故障恢复、指标和监控的功能。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、速率限制、访问控制以及端到端验证。

1.1.2. 获取支持

如果您在执行本文档所述的某个流程时遇到问题，请访问红帽客户门户。您可通过该客户门户：

- 搜索或浏览红帽知识库，了解有关红帽产品的技术支持文章。
- 提交问题单给红帽支持。



注意

在提交问题单的同时提供您的集群信息，可以帮助红帽支持为您进行排除故障。

  - 这类信息可使用 **oc adm must-gather** 命令来收集。
  - 唯一的集群 ID。进入 **（？） Help → Open Support Case**，在提交问题单时自动填充集群 ID。

- 访问其他产品文档。

如果您对本文档有任何改进建议，或发现了任何错误，请访问 Bugzilla，针对 **OpenShift Container Platform** 产品的 **Documentation**组件提交 Bugzilla 报告。请提供具体详情，如章节名称和 OpenShift Container Platform 版本。

在提交问题单时同时提供您的集群信息，可以帮助红帽支持为您进行排除故障。

您可使用 **must-gather** 工具来收集有关 OpenShift Container Platform 集群的诊断信息，包括虚拟机数据以及其他与 Red Hat OpenShift Service Mesh 相关的数据。

为了获得快速支持，请提供 OpenShift Container Platform 和 Red Hat OpenShift Service Mesh的诊断信息。

1.1.2.1. 关于 must-gather 工具

oc adm must-gather CLI 命令可收集最有助于解决问题的集群信息，如：

- 资源定义
- 审计日志
- 服务日志

您在运行该命令时，可通过包含 **--image** 参数来指定一个或多个镜像。指定镜像后，该工具便会收集有关相应功能或产品的信息。

您在运行 **oc adm must-gather** 时，集群上会创建一个新 Pod。在该 Pod 上收集数据，并保存至以 **must-gather.local** 开头的一个新目录中。此目录在当前工作目录中创建。

1.1.2.2. 先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。
- 已安装 OpenShift Container Platform CLI (**oc**)。

1.1.2.3. 关于收集服务网格数据

您可使用 **oc adm must-gather** CLI 命令来收集有关集群的信息，包括与 Red Hat OpenShift Service Mesh 相关的功能和对象：

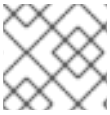
要使用 **must-gather**收集 Red Hat OpenShift Service Mesh 数据，您必须指定 Red Hat OpenShift Service Mesh 镜像：

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

1.1.3. Red Hat OpenShift Service Mesh 支持的配置

以下是 Red Hat OpenShift Service Mesh 唯一支持的配置：

- Red Hat OpenShift Container Platform 版本 4.x。



注意

OpenShift Online 和 OpenShift Dedicated 不支持 Red Hat OpenShift Service Mesh。

- 部署必须包含在一个独立的 OpenShift Container Platform 集群中。
- 此版本的 Red Hat OpenShift Service Mesh 仅适用于 OpenShift Container Platform x86_64。
- 此发行版本只支持在 OpenShift 集群中包含所有 Service Mesh 组件的配置。它不支持在集群之外或在多集群场景中管理微服务。
- 这个版本只支持没有集成外部服务的配置，比如虚拟机。

1.1.3.1. Red Hat OpenShift Service Mesh 支持的 Kiali 配置

- Kiali 观察控制台只支持 Chrome 、Edge 、Firefox 或 SDomain 浏览器的最新的两个版本。

1.1.3.2. 支持的 Mixer 适配器

- 此发行版本只支持以下 Mixer 适配器：
  - 3scale Istio Adapter

1.1.4. 新功能

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。

1.1.4.1. Red Hat OpenShift Service Mesh 1.1.10 版中包含的组件版本

组件	版本
Istio	1.4.8
Jaeger	1.17.4
Kiali	1.12.7
3scale Istio Adapter	1.0.0

1.1.4.2. Red Hat OpenShift Service Mesh 1.1.10 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。

1.1.4.3. Red Hat OpenShift Service Mesh 1.1.9 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。

1.1.4.4. Red Hat OpenShift Service Mesh 1.1.8 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以及程序错误。

1.1.4.5. Red Hat OpenShift Service Mesh 1.1.7 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。

1.1.4.6. Red Hat OpenShift Service Mesh 1.1.6 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。

1.1.4.7. Red Hat OpenShift Service Mesh 1.1.5 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以及程序错误。

此发行版本还添加了对配置密码套件的支持。

1.1.4.8. Red Hat OpenShift Service Mesh 1.1.4 的新功能

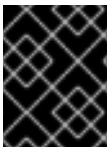
此 Red Hat OpenShift Service Mesh 发行版本解决了 CVE 报告的安全漏洞问题以及程序错误。

注意

要解决 CVE-2020-8663 的问题，则必须完成手动步骤。

1.1.4.8.1. CVE-2020-8663 所需的手动更新

对于 CVE-2020-8663**: envoy: Resource exhaustion when accepting too many connections** 的问题，为下游连接添加了一个可配置的限制。必须配置这个限制的配置选项来减轻这个安全漏洞的影响。



重要

无论您使用 1.1 版本还是使用 Red Hat OpenShift Service Mesh 1.0 版本，需要手动步骤来缓解这个 CVE。

这个新配置选项称为 **overload.global_downstream_max_connections**，它作为一个代理的 **runtime** 设置可以进行配置。在 Ingress Gateway 上执行以下步骤配置限制。

流程

1. 使用以下文本创建名为 **bootstrap-override.json** 的文件，以强制代理覆盖 bootstrap 模板并从磁盘加载运行时配置：

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

2. 从 **bootstrap-override.json** 文件创建 secret，将 <SMCPnamespace> 替换为您在其中创建服务网格 control plane（SMCP）的命名空间：

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

3. 更新 SMCP 配置来激活覆盖。

更新的 SMCP 配置示例 #1

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    gateways:
```

```
      istio-ingressgateway:
        env:
          ISTIO_BOOTSTRAP_OVERRIDE: /var/lib/istio/envoy/custom-bootstrap/bootstrap-
override.json
        secretVolumes:
        - mountPath: /var/lib/istio/envoy/custom-bootstrap
          name: custom-bootstrap
          secretName: gateway-bootstrap
```

4. 要设置新的配置选项，创建一个带有适当的 **overload.global_downstream_max_connections** 设置的 secret。以下示例使用 **10000**：

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

5. 再次更新 SMCP，将 secret 挂载到 Envoy 查找运行时配置的位置：

更新的 SMCP 配置示例 #2

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  template: default
#Change the version to "v1.0" if you are on the 1.0 stream.
  version: v1.1
  istio:
    gateways:
      istio-ingressgateway:
        env:
          ISTIO_BOOTSTRAP_OVERRIDE: /var/lib/istio/envoy/custom-bootstrap/bootstrap-override.json
        secretVolumes:
        - mountPath: /var/lib/istio/envoy/custom-bootstrap
          name: custom-bootstrap
          secretName: gateway-bootstrap
        # below is the new secret mount
        - mountPath: /var/lib/istio/envoy/runtime
          name: gateway-settings
          secretName: gateway-settings
```

1.1.4.8.2. 从 Elasticsearch 5 升级到 Elasticsearch 6

从 Elasticsearch 5 更新至 Elasticsearch 6 时，必须删除 Jaeger 实例。然后，因为证书存在问题，需要重新创建 Jaeger 实例。重新创建 Jaeger 实例会触发新证书集。如果正在使用持久性存储，只要新 Jaeger 实例的 Jaeger 名称和命名空间与已删除的 Jaeger 实例相同，就可以挂载相同的卷。

Jaeger 作为 Red Hat Service Mesh 的一部分安装的流程

1. 确定 Jaeger 自定义资源文件的名称：

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

您应该看到类似如下的内容：

```
NAME     AGE
jaeger   3d21h
```

2. 将生成的自定义资源文件复制到临时目录中：

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

3. 删除 Jaeger 实例：

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

4. 从自定义资源文件的副本重新创建 Jaeger 实例：

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

5. 删除生成的自定义资源文件的副本：

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

Jaeger 没有作为 Red Hat Service Mesh 的一部分安装的流程

在开始前，创建 Jaeger 自定义资源文件的副本。

1. 通过删除自定义资源文件来删除 Jaeger 实例：

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

例如：

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

2. 从自定义资源文件的备份副本重新创建 Jaeger 实例：

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

3. 验证您的 Pod 已重启：

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

1.1.4.9. Red Hat OpenShift Service Mesh 1.1.3 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。

1.1.4.10. Red Hat OpenShift Service Mesh 1.1.2 的新功能

此 Red Hat OpenShift Service Mesh 发行版本解决了一个安全漏洞问题。

1.1.4.11. Red Hat OpenShift Service Mesh 1.1.1 的新功能

此 Red Hat OpenShift Service Mesh 发行版本增加了对断开连接的安装的支持。

1.1.4.12. Red Hat OpenShift Service Mesh 1.1.0 的新功能

此 Red Hat OpenShift Service Mesh 发行版本添加了对 Istio 1.4.6 和 Jaeger 1.17.1 的支持。

1.1.4.12.1. 从 1.0 手动更新到 1.1

如果要从 Red Hat OpenShift Service Mesh 1.0 更新至 1.1，您必须更新 **ServiceMeshControlPlane** 资源，以便将 control plane 组件更新至新版本。

1. 在 web 控制台中，点 Red Hat OpenShift Service Mesh Ooperator。
2. 点 **Project** 菜单，然后从列表中选择部署了 **ServiceMeshControlPlane** 的项目，如 **istio-system**。
3. 点 control plane 的名称，例如 **basic-install**。
4. 点 YAML，并将版本字段添加到 **ServiceMeshControlPlane** 资源的 **spec:** 中。例如，要升级到 Red Hat OpenShift Service Mesh 1.1.0，请添加 **version: v1.1**。

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

version 字段指定要安装的 ServiceMesh 版本，默认是最新的可用版本。

1.1.4.13. Red Hat OpenShift Service Mesh 1.0.11 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以及程序错误。

注意

要解决 CVE-2020-8663 的问题，则必须完成手动步骤。请参阅上述步骤。

1.1.4.14. Red Hat OpenShift Service Mesh 1.0.10 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。

1.1.4.15. Red Hat OpenShift Service Mesh 1.0.9 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。

1.1.4.16. Red Hat OpenShift Service Mesh 1.0.8 的新功能

此版本的 Red Hat OpenShift Service Mesh 解决了与 OpenShift Container Platform 4.4 间的兼容性问题。在将 OpenShift Container Platform 4.3 升级到 4.4 之前，您必须将 Red Hat OpenShift Service Mesh 升级到 1.0.8。

1.1.4.17. Red Hat OpenShift Service Mesh 1.0.7 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。

1.1.4.18. Red Hat OpenShift Service Mesh 1.0.6 的新功能

这个版本包括了内部改进。

1.1.4.19. Red Hat OpenShift Service Mesh 1.0.5 的新功能

这个版本包括了内部改进。

1.1.4.20. Red Hat OpenShift Service Mesh 1.0.4 的新功能

此 Red Hat OpenShift Service Mesh 发行版本添加了对 Kiali 1.0.9 的支持，并解决了 CVE（Common Vulnerabilities and Exposures）报告的安全漏洞问题。

1.1.4.21. Red Hat OpenShift Service Mesh 1.0.3 的新功能

此 Red Hat OpenShift Service Mesh 发行版本添加了对 Kiali 1.0.8 的支持，并解决了 CVE 报告的安全漏洞问题。

1.1.4.22. Red Hat OpenShift Service Mesh 1.0.2 的新功能

此 Red Hat OpenShift Service Mesh 发行版本添加了对 Istio 1.1.17、Jaeger 1.13.1、Kiali 1.0.7、3scale Istio Adapter 1.0 和 OpenShift Container Platform 4.2 的支持。

1.1.4.22.1. 从 1.0.1 升级到 1.0.2 所需的手动更新

更新 Red Hat OpenShift Service Mesh 需要额外的步骤才能将 OpenShift Container Platform 更新至版本 4.2。在将 OpenShift Container Platform 4.1.x 升级到 4.2 之前，您必须将 Red Hat OpenShift Service Mesh 升级到 1.0.2。

先决条件

- Red Hat OpenShift Service Mesh 版本 1.0.1
- OpenShift Container Platform 版本 4.1

流程

1. 通过运行以下 **oc patch** 命令配置现有的 SMCP 资源请求。将 <smcp_namespace> 和 <smcp_name> 替换为您的具体名称：

```
$ oc patch -n <smcp_namespace> smcp <smcp_name> \ (1)
    --type=merge -p \
    '{"spec": {"istio": {"global": {"defaultResources": {"requests": {"cpu": "10m","memory":
"128Mi"},"limits":{}},"proxy": {"resources": {"requests": {"cpu": "10m","memory":
"128Mi"},"limits":{}}},"defaultPodDisruptionBudget": {"enabled": false}},"security": {"resources":
{"requests": {"cpu": "10m","memory": "128Mi"}}},"galley": {"resources": {"requests": {"cpu":
"10m","memory": "128Mi"}}},"pilot": {"resources": {"requests": {"cpu": "10m","memory":
"128Mi"}}},"mixer": {"telemetry": {"resources": {"requests": {"cpu": "10m","memory":
"128Mi"}}}},"gateways": {"istio-egressgateway": {"resources": {"requests": {"cpu":
"10m","memory": "128Mi"}}},"istio-ingressgateway": {"resources": {"requests": {"cpu":
"10m","memory": "128Mi"}}}},"prometheus": {"resources": {"requests": {"cpu":
"10m","memory": "128Mi"}}}}}}'
```

(1) 例如 **basic-install**。

运行此命令后，请等到 SMCP 命名空间中的所有 SMCP Pod 替换完毕。

2. Pod 在 SMCP 命名空间中运行后，重新部署您的 Data Plane 应用程序，如 **bookinfo**。

3. 以 **cluster-admin** 用户身份（如 **kubeadmin**）登录，然后运行以下命令删除 CNI **istio-node** DaemonSet。如果您的 Red Hat OpenShift Service Mesh Operator 没有安装在默认的 **openshift-operators** 命名空间中，请替换 **openshift-operators**：

```
$ oc delete -n openshift-operators daemonset istio-node
```

4. 将 Red Hat OpenShift Service Mesh Operator 和 SMCP 升级至 1.0.2。所有 Pod 在 SMCP 命名空间中运行后，请通过为每个部署运行以下命令来修补 Data Plane 应用程序：

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

5. 使用 OpenShift Container Platform Web 控制台升级 OpenShift Container Platform。

1.1.4.23. Red Hat OpenShift Service Mesh 1.0.1 的新功能

此 Red Hat OpenShift Service Mesh 发行版本添加了对 Istio 1.1.11、Jaeger 1.13.1、Kiali 1.0.6、3scale Istio Adapter 1.0 和 OpenShift Container Platform 4.1 的支持。

1.1.4.24. Red Hat OpenShift Service Mesh 1.0 的新功能

此 Red Hat OpenShift Service Mesh 发行版本添加了对 Istio 1.1.11、Jaeger 1.13.1、Kiali 1.0.5、3scale Istio Adapter 1.0 和 OpenShift Container Platform 4.1 的支持。

本发行版本中的其他显著变化包括：

- Kubernetes Container Network Interface (CNI) 插件一直被启用。
- control plane 默认配置为多租户。单租户、集群范围内的 control plane 配置功能已弃用。
- 通过 OperatorHub 安装 Elasticsearch 、Jaeger 、Kiali 和 Service Mesh Operators。
- 您可以创建并指定 control plane 模板。
- 这个版本删除了自动路由创建功能。

1.1.5. 已过时的功能

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 OpenShift Container Platform 中，并将继续被支持。但是，这个功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。

1.1.5.1. Red Hat OpenShift Service Mesh 1.1.5 已弃用的功能

以下自定义资源在此发行版本中已弃用，并将在以后的发行版本中删除。

- **Policy** - **Policy** 资源已弃用，并将在以后的版本中由 **PeerAuthentication** 资源替代。
- **MeshPolicy** - **MeshPolicy** 资源已弃用，并将在以后的版本中由 **PeerAuthentication** 资源替代。
- **v1alpha1** RBAC API - v1alpha1 RBAC 策略已弃用，使用 v1beta1 **AuthorizationPolicy**。RBAC（Role Based Access Control）定义 **ServiceRole** 和 **ServiceRoleBinding** 对象。

    - **ServiceRole**
    - **ServiceRoleBinding**
- **RbacConfig** - **RbacConfig** 实施自定义资源定义来控制 Istio RBAC 行为。
    - **ClusterRbacConfig**（Red Hat OpenShift Service Mesh 1.0 以前的版本）
    - **ServiceMeshRbacConfig**（Red Hat OpenShift Service Mesh 版本 1.0 及更新版本）
- 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。

本发行版本中还弃用了以下组件，并将在以后的版本中被 **Istiod** 组件替代。

- **Mixer** - 访问控制及使用策略
- **Pilot** - 服务发现和代理配置
- **Citadel** - 证书生成
- **Galley** - 配置验证和发布

1.1.6. 已知问题

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

- Red Hat OpenShift Service Mesh 不支持 IPv6，因为上游 Istio 项目不支持它，OpenShift 也不完全支持它。
- 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 Jaeger 和 Grafana）时，必须使用 OpenShift Container Platform 登录凭证接受证书并重新进行身份验证。这是因为框架如何显示控制台中的内置页面中存在问题。

1.1.6.1. Service Mesh 已知问题

Red Hat OpenShift Service Mesh 有以下已知的问题：

- 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。Istio 仪表板仍然存在。要访问它们，在导航框中点 **Dashboard** 菜单，并选 **Manage** 标签页。
- Bug 1821432 OpenShift Container Platform Control Resource details 页面中的 Toggle 控件无法正确更新 CR。OpenShift Container Platform 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。
- 在对安装了 Service Mesh 1.0.x 的 Jaeger 或 Kiali Operator 进行升级时，Jaeger/Kiali Operator 的升级过程可能会无法完成，Operator 的状态会显示为 Pending。这个问题的解决方案正在开发中，您可以使用一个临时解决方案。详情请查看链接的知识库文章。
- Istio-14743 因为此 Red Hat OpenShift Service Mesh 版本所基于的 Istio 版本的限制，目前有一些应用程序与 Service Mesh 还不兼容。详参阅社区的相关链接。

- 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。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 control plane 有多个命名空间时，可能会导致出现性能问题。
- MAISTRA-465 Maistra Operator 无法为 operator 指标数据创建服务。
- 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-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。
- 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。

1.1.6.2. Kiali 已知问题

Kiali 中已知的问题：

- 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。

1.1.6.3. Jaeger 已知问题

Jaeger 中存在的限制：

- 虽然 Kafka publisher 是 Jaeger 的一部分，但它并不被支持。
- 不支持 Apache spark。
- 仅支持自置备的 Elasticsearch 实例。此发行版本不支持外部 Elasticsearch 实例。

Jaeger 中已知的问题：

- TRACING-1166 目前无法在断开网络连接的环境中使用 Jaeger 流策略。当一个 Kafka 集群被置备时，它会产生一个错误：**Failed to pull image registry.redhat.io/amq7/amq-streams-kafka-24-rhel7@sha256:f9ceca004f1b7DCCB3b82d9a8027961f9fe4104e0ed69752c0bdd8078b4a1076**。

- TRACING-809 Jaeger Ingester 与 Kafka 2.3 不兼容。当存在两个或多个 Jaeger Ingester 实例时，它会不断在日志中生成重新平衡信息。这是由于在 Kafka 2.3 里存在一个程序错误，它已在 Kafka 2.3.1 中修复。如需更多信息，请参阅 Jaegertracing-1819。

1.1.7. 修复的问题

在当前发行本中解决了以下问题：

1.1.7.1. Service Mesh 修复的问题

- MAISTRA-1352 Cert-manager 自定义资源定义(CRD)已针对这个发行版本和以后的版本被删除。如果您已经安装了 Red Hat OpenShift Service Mesh，如果没有使用 cert-manager，则必须手动删除 CRD。
  要删除 CRD，请运行以下命令：

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

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

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

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

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

- MAISTRA-1649 不同命名空间中的无标头服务会有冲突。在不同命名空间中部署无头服务时，端点配置会被合并，并导致推送到 sidecar 的 Envoy 配置无效。
- 当控制器在拥有者引用中未设置时， kubernetesenv 中的 Maistra-1541 会导致 Panic。如果 pod 没有指定控制器的 ownerReference，则会导致 **kubernetesenv cache.go** 代码出现 panic。
- 在使用 Istio sidecar 时，在 Agent 和 Collector 间的连接会出现 TRACING-1300 失败。对 Jaeger Operator 的更新默认启用了 Jaeger sidecar 代理和 Jaeger Collector 之间的 TLS 通信。
- TRACING-1208 访问 Jaeger UI 时的身份验证 "500 Internal Error" 错误。当尝试使用 OAuth 验证 UI 时，会得到 500 错误，因为 oauth-proxy sidecar 不信任安装时使用 additionalTrustBundle 定义的自定义 CA 捆绑包。
- OSSM-99 从没有标签（label）的直接 Pod 产生的工作负载可能会导致 Kiali 崩溃。
- OSSM-93 IstioConfigList 无法根据两个或者更多名称进行过滤。
- OSSM-92 在 VS/DR YAML 编辑页面中取消未保存的更改不会取消更改。
- OSSM-90 trace 没有包括在服务详情页中。
- MAISTRA-1001 关闭 HTTP/2 连接可能会导致 **istio-proxy** 中的分段错误。
- MAISTRA-932 添加了 **requires** 元数据，以添加 Jaeger operator 和 Elasticsearch operator 之间的依赖关系。确保在安装 Jaeger operator 时，它将自动部署 Elasticsearch operator（如果不可用）。
- MAISTRA-862 Galley 在多次命名空间删除和重新创建后丢弃了监控并停止了向其他组件提供配置。

- MAISTRA-833 Pilot 在多次命名空间删除和重新创建后停止了交付配置。
- MAISTRA-684 **istio-operator** 中默认的 Jaeger 版本为 1.12.0，它与 Red Hat OpenShift Service Mesh 0.12.TechPreview 提供的 Jaeger 版本 1.13.1 不匹配。
- MAISTRA-622 在 Maistra 0.12.0/TP12 中，permissive 模式无法正常工作。用户可以使用 Plain text 模式或 Mutual TLS 模式，但不能使用 permissive 模式。
- MAISTRA-572 Jaeger 无法与 Kiali 一起使用。在这个版本中，Jaeger 被配置为使用 OAuth 代理，但它被配置为只能通过浏览器进行配置，且不允许服务访问。Kiali 无法正确与 Jaeger 端点沟通，它会认为 Jaeger 被禁用。请参阅 TRACING-591。
- MAISTRA-357 在 OpenShift 4 Beta on AWS 中，默认无法通过端口 80 之外的 ingress 网关访问 TCP 或 HTTPS 服务。AWS 负载均衡器有一个健康检查，它可验证服务端点中的端口 80 是否活跃。如果服务没有在端口 80 中运行，负载均衡器健康检查就会失败。
- MAISTRA-348 OpenShift 4 Beta on AWS 不支持端口 80 或 443 之外的 ingress 网关流量。如果您将 ingress 网关配置为使用 80 或 443 以外的端口号处理 TCP 流量，作为临时解决方案，您必须使用 AWS 负载均衡器提供的服务主机名，而不是使用 OpenShift 路由器。

1.1.7.2. Kiali 修复的问题

- KIALI-3239 如果一个 Kiali Operator pod 失败且状态为 "Evicted"，它会阻塞 Kiali operator 的部署。解决办法是删除被逐出的 pod，并重新部署 Kiali operator。
- KIALI-3118 当对 ServiceMeshMemberRoll 进行修改后（例如，添加或删除了项目），Kiali pod 会重新启动，并在 Kiali pod 重新启动的过程中在 Graph 页中显示错误信息。
- KIALI-3096 Runtime metrics 在 Service Mesh 中失败。在 Service Mesh 和 Prometheus 之间有一个 OAuth 过滤器，需要向 Prometheus 传递一个 bearer 令牌才会授予访问权限。Kiali 已被更新为在与 Prometheus 服务器通讯时使用这个令牌，但应用程序的 metrics 当前会有 403 错误。
- KIALI-3070 此程序错误只会影响自定义 dashboard，它不会影响默认的 dashboard。当您在 metrics 设置中选择标签并刷新页面时，会在菜单中保留您的选择，但您的选择不会在图表中显示。
- KIALI-2686 当 control plane 有多个命名空间时，可能会导致出现性能问题。

1.2. 了解 RED HAT OPENSHIFT 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 OpenShift Container Platform 环境中的微服务。

1.2.1. 了解服务网格

*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 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
- 速率限制
- 访问控制
- 端到端的验证

1.2.2. Red Hat OpenShift Service Mesh 架构

Red Hat OpenShift Service Mesh 在逻辑上被分成一个 data plane 和一个 control plane：

data plane 是一组作为 sidecar 部署的智能代理。这些代理会接收并控制服务网格内不同微服务之间的所有入站和出站网络数据。Sidecar 代理还和 Mixer（通用的策略和遥测系统） 进行沟通。

- **Envoy 代理**可以截获服务网格内所有服务的入站和出站流量。Envoy 会在同一个 pod 中被部署为相关服务的 sidecar。

control plane 可以为路由流量管理和配置代理，并配置 Mixer 来强制执行策略并收集遥测数据。

- **Mixer** 强制执行访问控制和使用策略（如授权、速率限制、配额、验证和请求追踪），并从 Mixer 代理服务器和其它服务收集遥测数据。
- **Pilot** 在运行时配置代理。Pilot 为 Envoy sidecars 提供服务发现，智能路由的流量管理功能（例如 A/B 测试或 canary 部署），以及弹性（超时、重试和电路断路器）。
- **Citadel** 用于发布并轮转证书。Citadel 通过内置的身份和凭证管理功能提供了强大的服务到服务（service-to-service）的验证功能及对最终用户的验证功能。您可以使用 Citadel 提升服务网格中未加密的网络流量的安全性。Operator 可根据服务身份而不是使用 Citadel 进行网络控制来强制实施策略。
- **Galley** 用来管理服务网格配置，然后验证、处理和发布配置。Galley 用来保护其他服务网格组件，使它们与从 OpenShift Container Platform 获取的用户配置详情相隔离。

Red Hat OpenShift Service Mesh 还使用 **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。它相当于一个控制器，用于设置或更改集群中对象的所需状态。

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

Red Hat OpenShift Service Mesh 安装与 Istio 安装在多个方面都有所不同。当在 OpenShift 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。

1.2.3.1. 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。

1.2.3.2. 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 OpenShift 功能冲突，比如 builder pod。要启用自动注入，您可以指定 **sidecar.istio.io/inject** 注解，如自动 sidecar 注入部分所述。

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

Red Hat OpenShift Service Mesh 使用正则表达式匹配请求标头

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

1.2.3.4. 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。

1.2.3.5. 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、Tracing(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。

1.2.3.6. Envoy、Secret Discovery Service 和证书

- Red Hat OpenShift Service Mesh 不支持基于 QUIC 的服务。
- Red Hat OpenShift Service Mesh 目前还不支持使用 Istio 的 Secret Discovery Service (SDS) 功能部署 TLS 证书。Istio 的实施取决于使用 hostPath 挂载的 nodeagent 容器。

1.2.3.7. 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。

1.2.3.8. 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"部分。

1.2.3.8.1. 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。如需了解有关默认主机名的工作方式以及集群管理员如何自定义它的更多信息，请参阅 OpenShift 文档。

1.2.3.8.2. 子域

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

1.2.3.8.3. 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。

1.3. 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。

1.3.1. Kiali 概述

Kiali 为在 OpenShift Container Platform 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 。

1.3.2. Kiali 架构

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。当启用 Istio 遥测时，指标数据保存在 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 Telemetery 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 OpenShift Container Platform (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。当作为 Red Hat OpenShift Service Mesh 安装的一部分默认安装了 Jaeger，Kiali 控制台会包括一个标签页来显示 Jaeger 的追踪数据。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。同时请注意，为了查看 Jaeger 数据，用户必须可以访问安装 control plane 的命名空间。

- **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 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. 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。

1.4. JAEGER 介绍

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。Jaeger 提供了分布式追踪功能，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。

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。

Jaeger 在微服务的整个堆栈中记录了独立请求的执行过程，并将其显示为 trace。**trace**是系统的数据/执行路径。一个端到端的 trace 由一个或者多个 span 组成。

span 代表 Jaeger 中的逻辑工作单元，它包含操作名称、操作的开始时间和持续时间。span 可能会被嵌套并排序以模拟因果关系。

1.4.1. Jaeger 概述

作为服务所有者，您可以使用 Jaeger 来检测您的服务，以收集与服务架构相关的信息。Jaeger 是一个开源的分布式追踪平台，可用来对现代的、云原生的基于微服务的应用程序中组件间的交互进行监控、创建网络配置集并进行故障排除。

使用 Jaeger 可让您执行以下功能：

- 监控分布式事务
- 优化性能和延迟时间
- 执行根原因分析

Jaeger 基于厂商中立的 OpenTracing API 和工具。

1.4.2. Jaeger 架构

Jaeger 由几个组件组成，它们一起收集、存储和显示追踪数据。

- **Jaeger Client** （Tracer、Reporter、instrumented application, client libraries）- Jaeger client 是 OpenTracing API 的具体语言实现。它们可以用来为各种现有开源框架（如 Camel (Fuse)、Spring Boot (RHOAR)、MicroProfile (RHOAR/Thorntail)、Wilfly (EAP) 等提供分布式追踪工具。
- **Jaeger Agent** （Server Queue, Processor Workers）- Jaeger 代理是一个网络守护进程，它会监听通过 User Datagram Protocol (UDP) 发送的 span，并发送到收集程序。这个代理应被放置在要管理的应用程序的同一主机上。这通常是通过如 Kubernetes 等容器环境中的 sidecar 来实现的。
- **Jaeger Collector** （Que,Weue）- 与代理类似，该收集器可以接收 span，并将其放入内部队列以便进行处理。这允许收集器立即返回到客户端/代理，而不需要等待 span 进入存储。
- **Storage** (Data Store) - 收集器需要一个持久的存储后端。Jaeger 带有一个可插入的机制用于 span 存储。请注意：在这个发行本中，唯一支持的存储是 Elasticsearch。
- **Query** (Query Service) - Query 是一个从存储中检索 trace 的服务。
- **Ingester** (Ingester Service) - Jaeger 可以使用 Apache Kafka 作为收集器和实际后备存储 (Elasticsearch) 之间的缓冲。Ingester 是一个从 Kafka 读取数据并写入另一个存储后端 (Elasticsearch) 的服务。
- **Jaeger Console** – Jaeger 提供了一个用户界面，可让您可视觉地查看所分发的追踪数据。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。

1.4.3. Jaeger 特性

Jaeger 追踪提供以下功能：

- 与 Kiali 集成 – 当正确配置时，您可以从 Kiali 控制台查看 Jaeger 数据。
- 高可伸缩性 – Jaeger 后端被设计为没有单一故障点，并可根据需要进行缩放。
- 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 - Jaeger 带有 API，让可以作为 Zipkin 的替代，但红帽在此发行版本中不支持 Zipkin 的兼容性。

1.4.4. 后续步骤

- 准备在 OpenShift Container Platform 环境中安装 Red Hat OpenShift Service Mesh 。

1.5. SERVICE MESH 和 ISTIO 的不同

Red Hat OpenShift Service Mesh 安装与上游 Istio 社区安装有许多不同。当在 OpenShift Container Platform 上进行部署时，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。

Red Hat OpenShift Service Mesh 的当前发行版本与当前上游 Istio 社区发行版本的不同：

1.5.1. Red Hat OpenShift Service Mesh 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。

1.5.1.1. 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 control plane 部署使用的权限范围，比如 Galley 和 Pilot。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 配置每个成员项目以确保自身、control plane 和其它成员项目间的网络连接。具体的配置根据 OpenShift 软件定义的网络 (SDN) 配置的不同而有所不同。更多详情请参阅"关于 OpenShift SDN"。

如果 OpenShift Container Platform 集群被配置为使用 SDN 插件：

- **NetworkPolicy**: Red Hat OpenShift Service Mesh 在每个成员项目中创建一个 **NetworkPolicy** 资源，允许从其它成员和 control plane 到 pod 的入站网络数据。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。

- **Multitenant**: Red Hat OpenShift Service Mesh 将每个成员项目的 **NetNamespace** 加入到 control plane 项目的 **NetNamespace**（相当于运行 **oc adm pod-network join-projects --to control-plane-project member-project**）。如果您从 Service Mesh 中删除一个成员，它的 **NetNamespace** 与 control plane 分离（相当于运行 **oc adm pod-network is isolatedate-projects member-project**）。

- **Subnet**：没有执行其他配置。

1.5.1.2. 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。

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

Red Hat OpenShift Service Mesh 安装与 Istio 安装在多个方面都有所不同。当在 OpenShift 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。

1.5.2.1. 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。

1.5.2.2. 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 OpenShift 功能冲突，比如 builder pod。要启用自动注入，您可以指定 **sidecar.istio.io/inject** 注解，如自动 sidecar 注入部分所述。

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

Red Hat OpenShift Service Mesh 使用正则表达式匹配请求标头

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

1.5.2.4. 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。

1.5.2.5. 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、Tracing(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。

1.5.2.6. Envoy、Secret Discovery Service 和证书

- Red Hat OpenShift Service Mesh 不支持基于 QUIC 的服务。
- Red Hat OpenShift Service Mesh 目前还不支持使用 Istio 的 Secret Discovery Service (SDS) 功能部署 TLS 证书。Istio 的实施取决于使用 hostPath 挂载的 nodeagent 容器。

1.5.2.7. 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。

1.5.2.8. 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"部分。

1.5.2.8.1. 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。如需了解有关默认主机名的工作方式以及集群管理员如何自定义它的更多信息，请参阅 OpenShift 文档。

1.5.2.8.2. 子域

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

1.5.2.8.3. 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。

1.5.3. Kiali 和服务网格

通过 OpenShift Container Platform 上的 Service Mesh 安装 Kiali 与社区 Kiali 安装不同。为了解决问题、提供额外功能或处理不同之处，这些不同有时是必须的。

- 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。
- 用户不应该手动编辑 ConfigMap 或 Kiali 自定义资源文件，因为这些更改可能会被 Service Mesh 或 Kiali 覆盖。所有在 Red Hat OpenShift Service Mesh 上运行的 Kiali 配置都是在 **ServiceMeshControlPlane** 自定义资源文件中进行的，且只有有限的配置选项。更新 operator 文件应该仅限于具有 cluster-admin 权限的用户。

1.5.4. Jaeger 和 服务网格

通过 OpenShift Container Platform 上的 Service Mesh 安装的 Jaeger 与社区版的 Jaeger 安装有所不同。为了解决问题、提供额外功能或处理不同之处，这些不同有时是必须的。

- Service Mesh 默认启用 Jaeger。
- 为 Service Mesh 默认启用 ingress 。
- Zipkin 端口名称已改为 Jaeger-collector-zipkin（从 http）
- Jaeger 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 Jaeger operator 安装的 "Jaeger" 路由，且已受到 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。

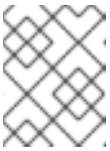
1.6. 准备安装 RED HAT OPENSHIFT SERVICE MESH

在安装 Red Hat OpenShift Service Mesh 前，请查看安装所需的操作，确保满足以下条件：

1.6.1. 先决条件

- 您的红帽帐户中拥有活跃的 OpenShift Container Platform 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。
- 查看 OpenShift Container Platform 4.3 概述 。
- 安装 OpenShift Container Platform 4.3。
  - 在 AWS 上安装 OpenShift Container Platform 4.3
  - 在用户置备的 AWS 上安装 OpenShift Container Platform 4.3
  - 在裸机上安装 OpenShift Container Platform 4.3

- 在 vSphere 上安装 OpenShift Container Platform 4.3



注意

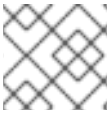
如果您要在受限网络上安装 Red Hat OpenShift Service Mesh，请按照所选 OpenShift Container Platform 基础架构的说明进行操作。

- 安装与 OpenShift Container Platform 版本匹配的 OpenShift Container Platform 命令行工具（**oc** 客户端工具），并将其添加到执行路径中。
  - 如果使用 OpenShift Container Platform 4.3，请参阅关于 CLI。

1.6.2. Red Hat OpenShift Service Mesh 支持的配置

以下是 Red Hat OpenShift Service Mesh 唯一支持的配置：

- Red Hat OpenShift Container Platform 版本 4.x。



注意

OpenShift Online 和 OpenShift Dedicated 不支持 Red Hat OpenShift Service Mesh。

- 部署必须包含在一个独立的 OpenShift Container Platform 集群中。
- 此版本的 Red Hat OpenShift Service Mesh 仅适用于 OpenShift Container Platform x86_64。
- 此发行版本只支持在 OpenShift 集群中包含所有 Service Mesh 组件的配置。它不支持在集群之外或在多集群场景中管理微服务。
- 这个版本只支持没有集成外部服务的配置，比如虚拟机。

1.6.2.1. Red Hat OpenShift Service Mesh 支持的 Kiali 配置

- Kiali 观察控制台只支持 Chrome 、Edge 、Firefox 或 SDomain 浏览器的最新的两个版本。

1.6.2.2. 支持的 Mixer 适配器

- 此发行版本只支持以下 Mixer 适配器：
  - 3scale Istio Adapter

1.6.3. Red Hat OpenShift Service Mesh 安装操作

要安装 Red Hat OpenShift Service Mesh Operator，需首先安装以下 Operator：

- **Elasticsearch** - 基于开源的 Elasticsearch 项目，您可以配置和管理 Elasticsearch 集群来使用 Jaeger 进行追踪和日志。
- **Jaeger** - 基于开源 Jaeger 项目，可让您执行追踪来监控复杂分布式系统中的事务并进行故障排除。
- **Kiali** - 基于开源的 Kiali 项目，提供了对服务网格进行观察的功能。通过使用 Kiali，您可以查看配置、监控流量，并在单一控制台中查看和分析 trace。

安装 Elasticsearch 、Jaeger 和 Kiali Operator 后，请安装 Red Hat OpenShift Service Mesh Operator。Service Mesh Operator 定义并监控管理 **ServiceMeshControlPlane** 资源，这个资源用来管理 Service Mesh 组件的部署、更新和删除操作。

- **Red Hat OpenShift Service Mesh**- 基于开源 Istio 项目，可让您连接、控制并观察组成应用程序的微服务。

警告

如需了解在生产环境中为 Elasticsearch 配置默认 Jaeger 的详情，请参阅配置 Elasticsearch。

1.6.4. 后续步骤

- 在OpenShift Container Platform 环境中安装 Red Hat OpenShift Service Mesh 。

1.7. RED HAT OPENSHIFT SERVICE MESH

安装 Service Mesh 包括安装 Elasticsearch 、Jaeger 、Kiali 和 Service Mesh Operators，创建和管理一个**ServiceMeshControlPlane** 资源 以部署 control plane，创建一个 **ServiceMeshMemberRoll** 资源以指定与 Service Mesh 关联的命名空间。

注意

在默认情况下，Mixer 的策略强制功能被禁用。您必须启用它才能运行策略任务。有关 启用 Mixer 策略强制执行的步骤,请参阅更新 Mixer 策略强制。

注意

从 Red Hat OpenShift Service Mesh 1.0 开始，多租户 control plane 安装是默认配置。

注意

Service Mesh 文档使用 **istio-system** 作为示例项目，但您可以将服务网格部署到任何项目中。

1.7.1. 先决条件

- 按照准备安装 Red Hat OpenShift Service Mesh 的过程进行操作。
- 具有 **cluster-admin** 角色的帐户。

Service Mesh 安装过程使用 OperatorHub 在 **openshift-operators** 项目内安装 **ServiceMeshControlPlane** 自定义资源。Red Hat OpenShift Service Mesh 定义并监控与部署、更新和删除 control plane 相关的 **ServiceMeshControlPlane**。

从 Red Hat OpenShift Service Mesh 1.1.10 开始，您必须安装 Elasticsearch Operator、Jaeger Operator 和 Kiali Operator，然后才能安装 control plane。

1.7.2. 安装 Elasticsearch Operator

默认 Jaeger 部署使用内存存储，这可以使那些评估 Jaeger 、演示或者在测试环境中使用 Jaeger 的用户快速地进行安装。如果要在生产环境中使用 Jaeger，则必须安装持久性存储选项，即 Elasticsearch。

先决条件

- 访问 OpenShift Container Platform Web 控制台。
- 具有 **cluster-admin** 角色的帐户。

警告

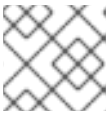
不要安装 Operators 的 Community 版本。不支持 Community 版本的 Operator。

注意

如果您已安装 Elasticsearch Operator 作为 OpenShift 集群日志记录的一部分，则不需要再次安装 Elasticsearch Operator。Jaeger Operator 将使用已安装的 Elasticsearch Operator 创建 Elasticsearch 实例。

流程

1. 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform web 控制台。
2. 进入 **Operators → OperatorHub**。
3. 在过滤器框中键入 **Elasticsearch** 以找到 Elasticsearch Operator。
4. 点由红帽提供的 **Elasticsearch Operator** 来显示有关 Operator 的信息。
5. 点击 **Install**。
6. 在 **Create Operator Subscription** 页面中，选择 **A specific namespace on the cluster**选项，然后从菜单中选择 **openshift-operators-redhat**。
7. 选择与 OpenShift Container Platform 安装匹配的**更新频道**。例如，如果您要在 OpenShift Container Platform 版本 4.5 上安装，请选择 4.5 更新频道。
8. 选择 **Automatic** 批准策略。



注意

手动批准策略需要拥有适当凭证的用户批准 Operator 的安装和订阅过程。

9. 点 **Subscribe**.
10. 在 **Installed Operators** 页面中，选择 **openshift-operators-redhat** 项目。等待 Elasticsearch Operator 的状态显示为 "InstallSucceeded" 后再继续进行操作。

1.7.3. 安装 Jaeger Operator

要安装 Jaeger，您需要使用 OperatorHub 来安装 Jaeger Operator。

默认情况下，Operator 安装在 **openshift-operators** 项目中。

先决条件

- 访问 OpenShift Container Platform Web 控制台。
- 具有 **cluster-admin** 角色的帐户。
- 如果需要持久性存储，则必须在安装 Jaeger Operator 前安装 Elasticsearch Operator。

警告

不要安装 Operators 的 Community 版本。不支持 Community 版本的 Operator。

流程

1. 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform web 控制台。
2. 进入 **Operators → OperatorHub**。
3. 在过滤器框中键入 **Jaeger** 来找到 Jaeger Operator。
4. 点由红帽提供的 **Jaeger Operator** 来显示有关 Operator 的信息。
5. 点 **Install**.
6. 在 **Create Operator Subscription** 页中选 **All namespaces on the cluster (default)**。这会在默认的 **openshift-operators** 项目中安装 Operator，并使其可以被集群中的所有项目使用。
7. 选择 **stable** 更新频道。这可在发布新版本时自动更新 Jaeger。如果您选择维护频道，例如 **1.17-stable**，则会在支持周期内接收程序错误修复和安全补丁。
   - 选择一个批准策略您可以选择 **Automatic** 或 **Manual** 更新。如果选择自动更新某个已安装的 Operator，则当相应 Operator 有可用的新版本时，Operator Lifecycle Manager（OLM）将自动升级该 Operator 的运行实例，而无需人为干预。如果选择手动更新，则当有新版 Operator 可用时，OLM 会创建更新请求。作为集群管理员，您必须手动批准该更新请求，才可将 Operator 更新至新版本。

注意

手动批准策略需要拥有适当凭证的用户批准 Operator 的安装和订阅过程。

8. 点 **Subscribe**.
9. 在 **Subscription Overview** 页面中，选择 **openshift-operators** 项目。等待 Jaeger Operator 的状态显示为 "InstallSucceeded" 后再继续进行操作。。

1.7.4. 安装 Kiali Operator

必须为 Red Hat OpenShift Service Mesh Operator 安装 Kiali Operator 以安装 control plane。

警告

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

先决条件

- 访问 OpenShift Container Platform Web 控制台。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 进入 **Operators → OperatorHub**。
3. 在过滤器框中键入 **Kiali** 来查找 Kiali Operator。
4. 点由红帽提供的 **Kiali Operator** 来显示有关 Operator 的信息。
5. 点击 **Install**。
6. 在 **Install Operator** 页面中，选择 **All namespaces on the cluster(default)**。这会在默认的 **openshift-operators** 项目中安装 Operator，并使其可以被集群中的所有项目使用。
7. 选择 **stable** 更新频道。
8. 选择 **Automatic** 批准策略。

注意

手动批准策略需要拥有适当凭证的用户批准 Operator 的安装和订阅过程。

9. 点击 **Install**。
10. **Installed Operators** 页会显示 Kiali Operator 的安装进度。

1.7.5. 安装 Red Hat OpenShift Service Mesh Operator

先决条件

- 访问 OpenShift Container Platform Web 控制台。
- 必须安装 Elasticsearch Operator。
- 必须安装 Jaeger Operator。
- 必须安装 Kiali Operator。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 进入 **Operators → OperatorHub**。
3. 在过滤器框中键入 **Red Hat OpenShift Service Mesh** 来查找 Red Hat OpenShift Service Mesh Operator。
4. 点 Red Hat OpenShift Service Mesh Operator 来显示有关 Operator 的信息。
5. 在 **Install Operator** 页面中，选择 **All namespaces on the cluster(default)**。这会在默认的 **openshift-operators** 项目中安装 Operator，并使其可以被集群中的所有项目使用。
6. 点击 **Install**。
7. 选择 **stable** 更新频道。
8. 选择 **Automatic** 批准策略。

注意

手动批准策略需要拥有适当凭证的用户批准 Operator 的安装和订阅过程。

9. 点击 **Install**。
10. **Installed Operators** 页会显示 Red Hat OpenShift Service Mesh Operator 的安装进度。

1.7.6. 部署 Red Hat OpenShift Service Mesh control plane

ServiceMeshControlPlane 资源定义要在安装过程中使用的配置。您可以部署红帽提供的默认配置，或者自定义 **ServiceMeshControlPlane** 文件以满足您的业务需求。

您可以使用 OpenShift Container Platform web 控制台或使用 **oc** 客户端工具从命令行部署 Service Mesh control plane。

1.7.6.1. 从 Web 控制台部署 control plane

按照以下步骤，使用 Web 控制台部署 Red Hat OpenShift Service Mesh control plane。

先决条件

- 必须安装 Red Hat OpenShift Service Mesh Operator。
- 查看有关如何自定义 Red Hat OpenShift Service Mesh 安装的说明。
- 具有 **cluster-admin** 角色的帐户。

流程

1. 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform web 控制台。
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**。

d. 点击 **Create**。

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

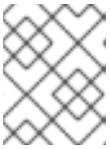
4. 如果需要，请在 Project 菜单中选择 **istio-system**。您可能需要等待一些时间，让 Operator 复制到新项目中。

5. 点 Red Hat OpenShift Service Mesh Operator。在 **Provided APIs** 下，Operator 提供了创建两个资源类型的链接：

   - **ServiceMeshControlPlane** 资源
   - **ServiceMeshMemberRoll** 资源

6. 在 **Istio Service Mesh Control Plane**下点 **Create ServiceMeshControlPlane**。

7. 在 **Create Service Mesh Control Plane**页面中，根据需要修改默认 **ServiceMeshControlPlane** 模板的 YAML。



注意

如需有关自定义 control plane 的更多信息，请参阅“自定义 Red Hat OpenShift Service Mesh 安装”。对于生产环境，您*必须*更改默认的 Jaeger 模板。

8. 点 **Create** 来创建 control plane。Operator 根据您的配置参数创建 Pod、服务和 Service Mesh control plane 组件。

9. 点 **Istio Service Mesh Control Plane**标签页。

10. 点新的 control plane 的名称。

11. 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。

1.7.6.2. 通过 CLI 部署 control plane

按照以下步骤，使用命令行部署 Red Hat OpenShift Service Mesh control plane。

先决条件

- 必须安装 Red Hat OpenShift Service Mesh Operator。
- 查看有关如何自定义 Red Hat OpenShift Service Mesh 安装的说明。
- 具有 **cluster-admin** 角色的帐户。
- 访问 OpenShift CLI（**oc**）。

流程

1. 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform CLI。

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

2. 创建一个名为 **istio-system** 的项目。

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

3. 使用“自定义 Red Hat OpenShift Service Mesh 安装”中的示例，创建一个名为 **istio-installation.yaml** 的 **ServiceMeshControlPlane** 文件。您可以根据需要自定义值来匹配您的用例。对于生产环境，您*必须*更改默认的 Jaeger 模板。

4. 运行以下命令来部署 control plane：

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

5. 执行以下命令查看 control plane 安装的状态。

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

当 READY 列为 true 时，安装已成功完成。

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

6. 在安装过程中运行以下命令来监控 Pod 的进度：

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

您应该看到类似如下的输出：

输**出示例**

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

对于多租户环境，Red Hat OpenShift Service Mesh 支持集群中有多个独立 control plane。您可以使用 **ServiceMeshControlPlane** 模板生成可重复使用的配置。如需更多信息，请参阅创建 control plane 模板。

1.7.7. 创建 Red Hat OpenShift Service Mesh member roll

ServiceMeshMemberRoll 列出了属于 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。

注意

只有在 Service Mesh control plane 安装成功时才会更新成员项目。

1.7.7.1. 从 Web 控制台创建 member roll

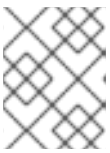
按照此流程，使用 Web 控制台将一个或多个项目添加到 Service Mesh member roll 中。

先决条件

- 已安装并验证的 Red Hat OpenShift Service Mesh Operator。
- 安装的 **ServiceMeshControlPlane** 的位置。
- 要添加到服务网格的现存项目列表。

流程

1. 如果您还没有网格的项目，或者您从头开始，请创建一个项目。它必须与 **istio-system**不同。
   a. 浏览至 **Home → Project**。
   b. 在 **Name** 字段中输入一个名称。
   c. 点击 **Create**。
2. 登陆到 OpenShift Container Platform Web 控制台。
3. 导航到 **Operators → Installed Operators**。
4. 点 **Project** 菜单，然后从列表中选择部署了 **ServiceMeshControlPlane** 的项目，如 **istio-system**。
5. 点 Red Hat OpenShift Service Mesh Operator。
6. 点 **All Instances** 标签。
7. 点 **Create New**，然后选择 **Create Istio Service Mesh Member Roll**。



注意

Operator 完成复制资源的过程需要一定的时间，因此您可能需要刷新屏幕才能看到 **Create Istio Service Mesh Member Roll**按钮。

8. 在 **Create Service Mesh Member Roll**页面中，修改 YAML 以添加项目作为成员。您可以添加多个项目，但每个项目只能属于**一个 ServiceMeshMemberRoll** 资源。
9. 点 **Create** 保存 Service Mesh Member Roll。

1.7.7.2. 通过 CLI 创建 member roll

按照此流程，从命令行将项目添加到 **ServiceMeshMemberRoll**。

先决条件

- 已安装并验证的 Red Hat OpenShift Service Mesh Operator。
- 安装的 **ServiceMeshControlPlane** 的位置。
- 要添加到服务网格的项目列表。
- 访问 OpenShift CLI（**oc**）。

流程

1. 登录 OpenShift Container Platform CLI。

```
$ oc login
```

2. 在 **ServiceMeshControlPlane** 资源所在的同一个项目中创建 **ServiceMeshMemberRoll** 资源，在本例中为 **istio-system**。资源必须被命名为 **default**。

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

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

3. 修改默认 YAML 以添加您的项目作为 **members**。您可以添加多个项目，但每个项目只能属于一个 **ServiceMeshMemberRoll** 资源。

1.7.7.3. 创建 Red Hat OpenShift Service Mesh 成员

ServiceMeshMember 资源可由服务网格用户创建，即使这些用户没有权限直接将成员添加到 **ServiceMeshMemberRoll** 中。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对网格的访问权限，例如：

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

管理员可修改 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** 会把项目添加到它所指代的 control plane 项目中的 **ServiceMeshMemberRoll**。

```
apiVersion: maistra.io/v1
```

```
kind: ServiceMeshMember
metadata:
  name: default
spec:
  controlPlaneRef:
    namespace: control-plane-namespace
    name: minimal-install
```

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

1.7.8. 为服务网格添加或删除项目

按照此流程，使用 Web 控制台修改现有 Service Mesh **ServiceMeshMemberRoll** 资源。

- 您可以添加多个项目，但每个项目只能属于一个 **ServiceMeshMemberRoll** 资源。
- 当它对应的 **ServiceMeshControlPlane** 资源被删除后，**ServiceMeshMemberRoll** 资源也会被删除。

1.7.8.1. 从 web 控制台修改 member roll

先决条件

- 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 。
- 您要为网格添加或删除的项目的名称。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 导航到 **Operators → Installed Operators**。
3. 点 **Project** 菜单，然后从列表中选择部署了 **ServiceMeshControlPlane** 的项目，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 以添加或删除作为成员的项目。您可以添加多个项目，但每个项目只能属于一个 **ServiceMeshMemberRoll** 资源。
9. 点 **Save**。
10. 点 **Reload**。

1.7.8.2. 通过 CLI 修改 member roll

按照此流程，使用命令行修改现有的 Service Mesh member roll。

先决条件

- 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 。
- 您要为网格添加或删除的项目的名称。
- 访问 OpenShift CLI（**oc**）。

流程

1. 登录 OpenShift Container Platform 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。您可以添加多个项目，但每个项目只能属于一个 **ServiceMeshMemberRoll** 资源。

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

```
# a list of projects joined into the service mesh
- your-project-name
- another-project-name
```

1.7.9. 手动更新

如果您选择使用手工更新，Operator Lifecycle Manager (OLM) 会控制集群中 Operator 的安装、升级和基于角色的访问控制 (RBAC)。OLM 在 OpenShift Container Platform 中默认运行。OLM 使用 CatalogSource，而 CatalogSources 使用 Operator Registry API 来查询是否有可用的 Operator 及已安装 Operator 是否有升级版本。

- 如需了解有关 OpenShift Container Platform 如何处理升级的更多信息，请参阅 Operator Lifecycle Manager 文档。

1.7.9.1. 更新应用程序 pod

如果您在安装 Operators 时选择了 Automatic 批准策略，那么 Operator 会自动更新 control plane，但不会更新您的应用程序。现有的应用程序将仍是网格的一部分并可以正常工作。应用程序管理员必须重启应用程序来升级 sidecar。

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，则必须通过修改在部署或 pod 中指定的 sidecar 容器镜像来手动更新 sidecar。

1.8. 删除 RED HAT OPENSHIFT SERVICE MESH

这个过程允许您从现有的 OpenShift Container Platform 实例中删除 Red Hat OpenShift Service Mesh。在删除 Operator 前移除 control plane。

1.8.1. 删除 Red Hat OpenShift Service Mesh member roll

ServiceMeshMemberRoll 资源会在与其关联的 **ServiceMeshControlPlane** 资源删除时自动删除。

1.8.2. 删除 Red Hat OpenShift Service Mesh control plane

您可以使用 OpenShift Container Platform web 控制台或 CLI 删除 Service Mesh control plane。

1.8.2.1. 使用 Web 控制台删除 control plane

按照以下步骤，使用 Web 控制台删除 Red Hat OpenShift Service Mesh control plane。

先决条件

- 已部署了 Red Hat OpenShift Service Mesh control plane。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。

2. 点 **Project** 菜单，从列表中选择 **istio-system** 项目。
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**。

1.8.2.2. 通过 CLI 删除 control plane

按照以下步骤，使用 CLI 删除 Red Hat OpenShift Service Mesh control plane。

先决条件

- 已部署了 Red Hat OpenShift Service Mesh control plane。
- 访问 OpenShift Container Platform 命令行界面 (CLI) 也称为 **oc**。

流程

当删除 **ServiceMeshControlPlane** 时，Service Mesh 会通知 Operator 开始卸载已安装的所有内容。

提示

您可以使用一个短的别名 **smcp** 来替换 **servicemeshcontrolplane**。

1. 登录 OpenShift Container Platform CLI。
2. 运行这个命令来获得安装的 **ServiceMeshControlPlane** 的名称：

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

3. 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 servicemeshcontrolplanes -n istio-system <name_of_custom_resource>
```

1.8.3. 删除安装的 Operator

您必须删除 Operator 才可以成功删除 Red Hat OpenShift Service Mesh。在移除 Red Hat OpenShift Service Mesh Operator 后，必须删除 Jaeger Operator 、Kiali Operator 和 Elasticsearch Operator。

1.8.3.1. 删除 Red Hat OpenShift Service Mesh Operator

按照以下步骤删除 Red Hat OpenShift Service Mesh Operator。

先决条件

- 访问 OpenShift Container Platform Web 控制台。
- 必须安装 Red Hat OpenShift Service Mesh Operator。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 在 **Operators → Installed Operators**页中，滚动页或使用 **Filter by name** 找到 Red Hat OpenShift Service Mesh Operator。然后点击它。
3. 在 **Operator Details** 页面右侧，从 **Actions** 下拉菜单中选择 **Uninstall Operator**。
4. 如果要删除所有安装相关组件，则在看到 **Remove Operator Subscription** 窗口提示时，勾选 **Also completely remove the Operator from the selected namespace**复选框。这会删除 CSV，并删除与 Operator 关联的 Pod 、部署 、crds 和 CR。

1.8.3.2. 删除 Jaeger Operator

按照以下步骤删除 Jaeger Operator。

先决条件

- 访问 OpenShift Container Platform Web 控制台。
- 必须安装 Jaeger Operator。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 在 **Operators → Installed Operators** 页中，通过滚动页或使用 **Filter by name** 来找到 Jaeger Operator。然后点击它。
3. 在 **Operator Details** 页面右侧，从 **Actions** 下拉菜单中选择 **Uninstall Operator**。
4. 如果要删除所有安装相关组件，则在看到 **Remove Operator Subscription** 窗口提示时，勾选 **Also completely remove the Operator from the selected namespace**复选框。这会删除 CSV，并删除与 Operator 关联的 Pod 、部署 、crds 和 CR。

1.8.3.3. 删除 Kiali Operator

按照以下步骤删除 Kiali Operator。

先决条件

- 访问 OpenShift Container Platform Web 控制台。
- 必须安装 Kiali Operator。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。

2. 在 **Operators → Installed Operators** 页中，通过滚动页或使用 **Filter by name** 找到 Kiali Operator。然后点击它。

3. 在 **Operator Details** 页面右侧，从 **Actions** 下拉菜单中选择 **Uninstall Operator**。

4. 如果要删除所有安装相关组件，则在看到 **Remove Operator Subscription** 窗口提示时，勾选 **Also completely remove the Operator from the selected namespace** 复选框。这会删除 CSV，并删除与 Operator 关联的 Pod 、部署 、crds 和 CR。

1.8.3.4. 删除 Elasticsearch Operator

按照以下步骤删除 Elasticsearch Operator。

先决条件

- 访问 OpenShift Container Platform Web 控制台。
- 必须安装 Elasticsearch Operator。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。

2. 在 **Operators → Installed Operators** 页中，通过滚动页或使用 **Filter by name** 找到 Elasticsearch Operator。然后点击它。

3. 在 **Operator Details** 页面右侧，从 **Actions** 下拉菜单中选择 **Uninstall Operator**。

4. 如果要删除所有安装相关组件，则在看到 **Remove Operator Subscription** 窗口提示时，勾选 **Also completely remove the Operator from the selected namespace** 复选框。这会删除 CSV，并删除与 Operator 关联的 Pod 、部署 、crds 和 CR。

1.8.3.5. 清理 Operator 资源

在使用 OperatorHub 接口删除 Red Hat OpenShift Service Mesh Operator 后会剩下一些没有被删除的资源。按照以下步骤手工删除这些资源。

先决条件

- 具有集群管理访问权限的帐户。
- 访问 OpenShift Container Platform 命令行界面 (CLI) 也称为 **oc**。

流程

1. 以集群管理员身份登录到 OpenShift Container Platform CLI。

2. 在卸载 Operators 后运行以下命令清理资源：

注意

用安装了 Red Hat OpenShift Service Mesh Operator 的项目的名称替换 **<operator-project>** 。这通常是 **openshift-operators**。

```
$ oc delete validatingwebhookconfiguration/<operator-project>.servicemesh-resources.maistra.io
```

```
$ oc delete mutatingwebhookconfigurations/<operator-project>.servicemesh-resources.maistra.io
```

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

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

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

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

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

1.8.4. 后续步骤

- 自定义 Red Hat OpenShift Service Mesh 安装 。
- 准备在 Red Hat OpenShift Service Mesh 上部署应用程序 。

1.9. 自定义 RED HAT OPENSHIFT SERVICE MESH 安装

您可以通过修改默认的 Service Mesh 自定义资源或者创建新的自定义资源来定制 Red Hat OpenShift Service Mesh。

1.9.1. 先决条件

- 具有 **cluster-admin** 角色的帐户。
- 完成了准备安装 Red Hat OpenShift Service Mesh 的过程。
- 已安装了 operator。

1.9.2. Red Hat OpenShift Service Mesh 自定义资源

注意

在整个 Service Mesh 文档中，使用 **istio-system** 项目作为一个示例，您可以根据需要使用其他项目。

自定义资源 允许您在 Red Hat OpenShift Service Mesh 项目或集群中扩展 API。当部署 Service Mesh 时，它会创建一个默认的 **ServiceMeshControlPlane**，可以修改它来更改项目参数。

Service Mesh operator 可以通过添加 **ServiceMeshControlPlane** 资源类型来扩展 API，这可让您在项目中创建 **ServiceMeshControlPlane** 对象。通过创建一个 **ServiceMeshControlPlane** 对象，指示 Operator 将一个 Service Mesh control plane 安装到项目中，并使用在 **ServiceMeshControlPlane** 中设置的参数。

这个示例 **ServiceMeshControlPlane** 定义包含所有支持的参数，并部署基于 Red Hat Enterprise Linux（RHEL）的 Red Hat OpenShift Service Mesh 1.1.10 镜像。

重要

3scale Istio 适配器在自定义资源文件中被部署并配置。它还需要一个可以正常工作的 3scale 帐户（SaaS 或 On-Premises）。

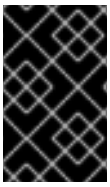
istio-installation.yaml 的示例

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
metadata:
  name: basic-install
spec:

  istio:
    global:
      proxy:
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 500m
            memory: 128Mi

    gateways:
      istio-egressgateway:
        autoscaleEnabled: false
      istio-ingressgateway:
        autoscaleEnabled: false
        ior_enabled: false

    mixer:
      policy:
        autoscaleEnabled: false

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

    pilot:
      autoscaleEnabled: false
      traceSampling: 100

    kiali:
      enabled: true
```

```
  grafana:
    enabled: true

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

1.9.3. ServiceMeshControlPlane 参数

以下示例演示了使用 **ServiceMeshControlPlane** 参数，并提供了有关支持参数的附加信息。



重要

您可以根据 OpenShift 集群的配置，使用这些参数为 Red Hat OpenShift Service Mesh 配置资源，其中包括 CPU、内存和 pod 的数量。根据当前集群配置中的可用资源配置这些参数。

1.9.3.1. Istio 全局示例

下面是一个示例，它演示了**ServiceMeshControlPlane** 的 Istio 全局参数，以及可用参数和值的信息。

注意

为了使 3scale Istio 时配器可以正常工作，**disablePolicyChecks** 必须为 **false**。

全局参数示例

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

表 1.1. 全局参数

参数	描述	值	默认值
disablePolicyChecks	启用/禁用策略检查。	**true/false**	**true**

参数	描述	值	默认值
policyCheckFailOpen	指定在 Mixer 策略服务无法访问时，是否允许流量传递给 Envoy sidecar。	**true/false**	**false**
tag	Operator 用来抓取 Istio 镜像的 tag。	有效的容器镜像 tag。	**1.1.0**
hub	Operator 用来抓取 Istio 镜像的中心。	有效的镜像仓库。	**maistra/** or **registry.redhat.io/openshift-service-mesh/**
mtls	控制是否默认在服务间启用/禁用传输层安全 (mTLS) 。	**true/false**	**false**
imagePullSecrets	如果对提供 Istio 镜像的 registry 的访问是安全的，在这里列出一个 imagePullSecret 。	redhat-registry-pullSecret 或 quay-pullSecret	无

这些参数专用于全局参数的代理子集。

表 1.2. 代理参数

类型	参数	描述	值	默认值
Resources	**cpu**	为 Envoy proxy 要求的 CPU 资源量。	基于环境配置的 CPU 资源，以 cores 或 millicores 为单位（例如，200m 、0.5 、1）指定。	**10m**
	memory	Envoy proxy 内存量请求	可用内存，以字节为单位（例如: 200Ki, 50Mi, 5Gi），基于您的环境配置。	**1024Mi**
Limits	**cpu**	为 Envoy proxy 请求的最大 CPU 资源量。	基于环境配置的 CPU 资源，以 cores 或 millicores 为单位（例如，200m 、0.5 、1）指定。	**2000m**

类型	参数	描述	值	默认值
	memory	Envoy proxy 允许使用的最大内存数量。	可用内存，以字节为单位（例如: 200Ki, 50Mi, 5Gi），根据您的环境配置而定。	**128Mi**

1.9.3.2. Istio 网关配置

下面是一个示例，它演示了 **ServiceMeshControlPlane** 的 Istio 网关参数 以及相关的信息。

网关参数示例

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

表 1.3. Istio 网关参数

类型	参数	描述	值	默认值
istio-egressgateway	**autoscaleEnabled**	启用/禁用自动扩展。	**true/false**	**true**
	autoscaleMin	根据 **autoscaleEnabled**，为出站网关部署的最少的 pod 数量。	基于环境配置的可分配 pods 的有效数量。	**1**
	autoscaleMax	根据 **autoscaleEnabled** 设置，为出站网关部署的最大 pod 数量 。	基于环境配置的可分配 pods 的有效数量。	**5**
istio-ingressgateway	**autoscaleEnabled**	启用/禁用自动扩展。	**true/false**	**true**

类型	参数	描述	值	默认值
	autoscaleMin	根据 **autoscaleEnabled**，为入站网关部署的最少的 pod 数量。	基于环境配置的可分配 pods 的有效数量。	**1**
	autoscaleMax	根据 **autoscaleEnabled**，为入站网关部署的最大的 pod 数量。	基于环境配置的可分配 pods 的有效数量。	**5**
	ior_enabled	控制是否启用自动路由创建。	**true**/**false**	**false**

1.9.3.3. 自动路由创建

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。

1.9.3.3.1. 启用自动路由创建

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。作为 control plane 部署的一部分启用 IOR。

如果网关包含一个 TLS 部分，则 OpenShift Route 将被配置为支持 TLS。

1. 在 **ServiceMeshControlPlane** 资源中添加 **ior_enabled** 参数，并将其设置为 **true**。例如，请查看以下资源片断：

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

1.9.3.3.2. 子域

Red Hat OpenShift Service Mesh 使用子域创建路由，但必须配置 OpenShift Container Platform 才能启用它。子域，如 ***.domain.com**，被支持，但不是默认支持。

如果创建了以下网关：

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

然后，会自动创建以下 OpenShift 路由。您可以使用以下命令来检查是否创建了路由：

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

预期输出

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

如果删除了网关，Red Hat OpenShift Service Mesh 会删除路由。但是，手动创建的路由都不会被 Red Hat OpenShift Service Mesh 修改。

集群管理员可以参阅使用通配符路由来获得如何启用子域的说明。

1.9.3.4. Istio Mixer 配置

下面是一个示例，它演示了**ServiceMeshControlPlane** 的 Mixer 参数，以及可用参数和值的信息。

Mixer 参数示例

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

表 1.4. Istio Mixer 策略参数

参数	描述	值	默认值
enabled	参数启用/禁用 Mixer。	**true/false**	**true**
autoscaleEnabled	启用/禁用自动扩展。在小型环境中禁用它。	**true/false**	**true**
autoscaleMin	根据 **autoscaleEnabled**，部署的最少的 pod 数量。	基于环境配置的可分配 pods 的有效数量。	**1**
autoscaleMax	根据 **autoscaleEnabled**，部署的最大的 pod 数量。	基于环境配置的可分配 pods 的有效数量。	**5**

表 1.5. Istio Mixer 遥测参数

类型	参数	描述	值	默认
Resources	**cpu**	Mixer 遥测所需的 CPU 资源百分比。	基于环境配置的 CPU 资源（以毫秒为单位）。	**10m**
	memory	Mixer 遥测所需的内存量。	可用内存，以字节为单位（例如: 200Ki, 50Mi, 5Gi），根据您的环境配置而定。	**128Mi**
Limits	**cpu**	Mixer 遥测可以使用的 CPU 资源的最大百分比。	基于环境配置的 CPU 资源（以毫秒为单位）。	**4800m**
	memory	Mixer 遥测允许使用的最大内存数量。	可用内存，以字节为单位（例如: 200Ki, 50Mi, 5Gi），根据您的环境配置而定。	**4G**

1.9.3.5. Istio Pilot 配置

下面是一个示例，它演示了**ServiceMeshControlPlane** 的 Istio Pilot 参数，以及可用参数和值的信息。

pilot 参数示例

```
pilot:
  resources:
    requests:
```

```
      cpu: 100m
      memory: 128Mi
  autoscaleEnabled: false
  traceSampling: 100
```

表 1.6. Istio Pilot 参数

参数	描述	值	默认值
cpu	Pilot 请求的 CPU 资源的百分比。	基于环境配置的 CPU 资源（以毫秒为单位）。	**10m**
memory	Pilot 请求的内存量。	可用内存，以字节为单位（例如: 200Ki, 50Mi, 5Gi），根据您的环境配置而定。	**128Mi**
autoscaleEnabled	启用/禁用自动扩展。在小型环境中禁用它。	**true/false**	**true**
traceSampling	这个值控制随机抽样的频率。**注:** 在开发或测试时可以增加这个值。	有效百分比。	**1.0**

1.9.4. 配置 Kiali

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。

在 **ServiceMeshControlPlane** 中指定的默认 Kiali 参数如下：

Kiali 参数示例

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

表 1.7. Kiali 参数

参数	描述	值	默认值
enabled	启用/禁用 Kiali。默认情况下启用 Kiali 。	**true/false**	**true**

参数	描述	值	默认值
dashboard viewOnlyMode	为 Kiali 控制台启用/禁用只读视图模式。启用只读视图模式时，用户无法使用控制台来更改 Service Mesh。	**true/false**	**false**
ingress enabled	为 Kiali 启用/禁用 ingress。	**true/false**	**true**

1.9.4.1. 为 Grafana 配置 Kiali

当将 Kiali 和 Grafana 作为 Red Hat OpenShift Service Mesh 的一部分安装时，Operator 会默认配置以下内容：

- Grafana 作为 Kiali 的外部服务启用
- Kiali 控制台的 Grafana 授权
- Kiali 控制台的 Grafana URL

Kiali 可自动检测 Grafana URL。然而，如果您有不能轻易被 Kiali 自动探测到的自定义 Grafana 安装，则需要更新 **ServiceMeshControlPlane** 资源中的 URL 值。

额外的 Grafana 参数

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

1.9.4.2. 为 Jaeger 配置 Kiali

当您将 Kiali 和 Jaeger 作为 Red Hat OpenShift Service Mesh 的一部分安装时，Operator 会默认配置以下内容：

- Jaeger 作为 Kiali 的外部服务启用
- Kiali 控制台的 Jaeger 授权
- Kiali 控制台的 Jaeger URL

Kiali 可以自动检测 Jaeger URL。然而，如果您有不能轻易被 Kiali 自动探测到的自定义 Jaeger 安装，则需要更新 **ServiceMeshControlPlane** 资源中的 URL 值。

额外的 Jaeger 参数

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

1.9.5. 配置 Jaeger

当 Service Mesh Operator 创建 **ServiceMeshControlPlane** 资源时，它还会创建 Jaeger 资源。然后，Jaeger Operator 在创建 Jaeger 实例时会使用这个对象。

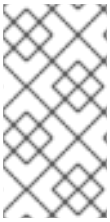
ServiceMeshControlPlane 中指定的默认 Jaeger 参数如下：

默认的 all-in-one Jaeger 参数

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

表 1.8. Jaeger 参数

参数	描述	值	默认值
tracing enabled	在 Service Mesh 中启用/禁用追踪。Jaeger 被默认安装。	**true/false**	**true**
jaeger template	指定使用哪个 Jaeger 部署策略。	• **all-in-one**- 用于开发、测试、演示和概念验证。 • **production-elasticsearch** - 用于产品环境。	**all-in-one**



注意

ServiceMeshControlPlane 资源中的默认模板是 **all-in-one** 部署策略，它使用 in-memory 存储。对于生产环境，唯一支持的存储选项是 Elasticsearch，因此您必须配置 **ServiceMeshControlPlane** 来在生产环境中部署 Service Mesh 时请求 **production-elasticsearch** 模板。

1.9.5.1. 配置 Elasticsearch

默认的 Jaeger 部署策略使用 **all-in-one** 模板，以便可使用最小资源完成安装。但是，因为 **all-in-one**模板使用 in-memory 存储，所以只建议用于开发、演示或者测试目的。在生产环境中不应该使用它。

如果要在产品环境中部署 Service Mesh 和 Jaeger，则需要将模板改为 **production-elasticsearch** 模板，该模板使用 Elasticsearch 来满足 Jaeger 的存储需要。

elasticsearch 是一个需要消耗大量内存的应用程序。在默认的 OpenShift Container Platform 安装中指定的初始节点可能不足以支持 Elasticsearch 集群。您应该修改默认的 Elasticsearch 配置，使其与您的用例和为 OpenShift Container Platform 安装请求的资源相匹配。您可以使用有效的 CPU 和内存值来修改每个组件的 CPU 和内存限值。如果要使用推荐的内存数量（或更多）运行，则必须在集群中添加额外的节点。请确定没有超过 OpenShift Container Platform 安装所请求的资源。

Elasticsearch 默认的 "生产环境" Jaeger 参数

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

表 1.9. elasticsearch 参数

参数	描述	值	默认值	例子
`tracing: enabled`	在 Service Mesh 中启用/禁用追踪。Jaeger 被默认安装。	**true/false**	**true**	
`ingress: enabled`	为 Jaeger 启用/禁用 ingress。	**true/false**	**true**	
`jaeger template`	指定使用哪个 Jaeger 部署策略。	**all-in-one/production-elasticsearch**	**all-in-one**	

参数	描述	值	默认值	例子
elasticsearch: nodeCount	要创建的 Elasticsearch 节点数量。	整数值。	1	概念验证 = 1, 最小部署 =3
requests: cpu	根据您的环境配置，请求的 CPU 数量。	以 core 或者 millicores 指定（例如: 200m, 0.5, 1）。	1Gi	概念证明 = 500m, 最小部署 =1
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）。	500m	概念证明 = 1Gi, 最小部署 = 16Gi*
limits: cpu	根据您的环境配置，CPU 数量的限值。	以 core 或者 millicores 指定（例如: 200m, 0.5, 1）。		概念证明 = 500m, 最小部署 =1
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）。		概念证明 = 1Gi, 最小部署 = 16Gi*
	* 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。			

流程

1. 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform 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**标签页。
5. 点 control plane 文件的名称，例如 **basic-install**。
6. 点 **YAML** 标签。
7. 编辑 Jaeger 参数，根据您的具体用例，使用 **production-elasticsearch** 模板参数替换默认的 **all-in-one** 模板。确定缩进格式正确。
8. 点 **Save**。
9. 点 **Reload**。OpenShift Container Platform 重新部署 Jaeger，并根据指定的参数创建 Elasticsearch 资源。

1.9.5.2. 配置 Elasticsearch 索引清理任务

当 Service Mesh Operator 创建 **ServiceMeshControlPlane** 时，它还会为 Jaeger 创建自定义资源 (CR)。Jaeger operator 在创建 Jaeger 实例时使用这个 CR。

当使用 Elasticsearch 存储时，默认会创建一个任务来清理旧的 trace。要配置这个任务的选项，请编辑 Jaeger 自定义资源 (CR) 以便为您的用例进行定制。以下列出了相关的选项。

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
spec:
  strategy: production
  storage:
    type: elasticsearch
    esIndexCleaner:
      enabled: false
      numberOfDays: 7
      schedule: "55 23 * * *"
```

表 1.10. Elasticsearch 索引清理参数

参数	值	描述
enabled	true/ false	启用或者禁用索引清理任务。
numberOfDays	整数值	删除索引前等待的天数。
schedule	"55 23 * * *"	运行任务的 cron 设置

有关在 OpenShift Container Platform 中配置 Elasticsearch 的详情，请参考配置 Elasticsearch。

1.9.6. 3scale 配置

下面是一个示例，它演示了 Red Hat OpenShift Service Mesh 自定义资源的 3scale Istio 适配器参数，以及可用参数和值的信息。

3scale 参数示例

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

表 1.11. 3scale 参数

参数	描述	值	默认值
enabled	是否使用 3scale 适配器	**true/false**	**false**
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**
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

1.9.7. 后续步骤

- 准备在 Red Hat OpenShift Service Mesh 上部署应用程序 。

1.10. 在 RED HAT OPENSHIFT SERVICE MESH 上部署应用程序

当将应用程序部署到 Service Mesh 后，在 Istio 上游社区版本的应用程序的行为和在 Red Hat OpenShift Service Mesh 中的应用程序的行为有几个不同之处。

1.10.1. 先决条件

- 查看 Red Hat OpenShift Service Mesh 和上游 Istio 社区安装的比较
- 查看安装 Red Hat OpenShift Service Mesh

1.10.2. 创建 control plane 模板

您可以使用 **ServiceMeshControlPlane** 模板生成可重复使用的配置。用户可根据自己的配置对创建的模板进行扩展。模板也可以从其他模板继承配置信息。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产品模板进行扩展。

当配置 control plane 模板（与 **ServiceMeshControlPlane** 语法相同）时，用户以分级方式继承设置。Operator 附带一个具有 Red Hat OpenShift Service Mesh 默认设置的 **default** 模板 。要添加自定义模板，您必须在 **openshift-operators** 项目中创建一个名为 **smcp-templates** 的 ConfigMap，并在 Operator 容器的 **/usr/local/share/istio-operator/templates**中挂载 ConfigMap 。

1.10.2.1. 创建 ConfigMap

按照以下步骤创建 ConfigMap。

先决条件

- 已安装并验证的 Service Mesh Operator。
- 具有 **cluster-admin** 角色的帐户。
- Operator 部署的位置。
- 访问 OpenShift Container Platform 命令行界面 (CLI) 也称为 **oc**。

流程

1. 以集群管理员身份登录到 OpenShift Container Platform CLI。
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 **<templates-directory>** 替换成本地磁盘上的 **ServiceMeshControlPlane** 文件的位置：

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

3. 找到 Operator 集群服务版本名称。

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

输**出示例**

```
maistra.v1.0.0          Red Hat OpenShift Service Mesh   1.0.0          Succeeded
```

4. 编辑 Operator 集群服务版本，指定 Operator 使用 **smcp-templates** ConfigMap。

```
$ oc edit clusterserviceversion -n openshift-operators maistra.v1.0.0
```

5. 在 Operator 部署中添加卷挂载和卷。

```
deployments:
  - name: istio-operator
    spec:
      template:
        spec:
          containers:
            volumeMounts:
              - name: discovery-cache
                mountPath: /home/istio-operator/.kube/cache/discovery
              - name: smcp-templates
                mountPath: /usr/local/share/istio-operator/templates/
          volumes:
            - name: discovery-cache
              emptyDir:
                medium: Memory
            - name: smcp-templates
              configMap:
                name: smcp-templates
...
```

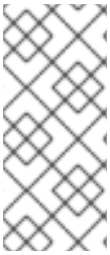
6. 保存更改并退出编辑器。

7. 现在，您可以使用 **ServiceMeshControlPlane** 中的 **template** 参数来指定模板。

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
metadata:
  name: minimal-install
spec:
  template: default
```

1.10.3. Red Hat OpenShift Service Mesh 的 sidecar 注入

Red Hat OpenShift Service Mesh 依靠应用程序 pod 中的 proxy sidecar 来为应用程序提供 Service Mesh 功能。您可以使用自动 sidecar 注入功能或手动管理它。红帽推荐通过注释使用自动注入功能，而不需要标记项目。这样可保证您的应用程序在部署时包含正确的 Service Mesh 配置。这个方法需要较少的权限，且不会与其他 OpenShift 功能冲突，比如 builder pod。



注意

如果您已经标记了项目，Istio 的上游版本会默认注入 sidecar。Red Hat OpenShift Service Mesh 要求您选择为一个部署自动注入 sidecar，因此您不需要标记项目。这可避免在不需要的情况下（例如，在构建或部署 pod 中）注入 sidecar。

webhook 会检查部署到所有项目的 pod 的配置，看它们是否选择使用适当的注解注入。

1.10.3.1. 通过注解在应用程序中设置代理的环境变量

您可以通过在 **injection-template.yaml** 文件中的部署中添加 pod 注解，为应用程序在 sidecar 代理上设置环境变量。环境变量注入 sidecar。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: resource
spec:
  replicas: 7
  selector:
    matchLabels:
      app: resource
  template:
    metadata:
      annotations:
        sidecar.maistra.io/proxyEnv: "{ \"maistra_test_env\": \"env_value\", \"maistra_test_env_2\":
\"env_value_2\" }"
```

1.10.3.2. 启用自动 sidecar 注入

在 Red Hat OpenShift Service Mesh 中部署应用程序时，需要为**sidecar.istio.io/inject** 注解指定一个 **"true"** 值来选择进行注入。这可以确保 sidecar 注入不会影响 OpenShift 的其他功能，如被 OpenShift 生态系统中的很多框架使用的 builder pod。

先决条件

- 识别要启用自动插入的部署。
- 找到应用程序的 YAML 配置文件。

流程

1. 在编辑器中打开应用程序的配置 YAML 文件。
2. 把值为 **"true"** 的 **sidecar.istio.io/inject** 添加到配置 YAML 中，如下所示：

   休眠测试程序示例

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: sleep
   spec:
     replicas: 1
     template:
   ```

```
  metadata:
    annotations:
      sidecar.istio.io/inject: "true"
    labels:
      app: sleep
  spec:
    containers:
    - name: sleep
      image: tutum/curl
      command: ["/bin/sleep","infinity"]
      imagePullPolicy: IfNotPresent
```

3. 保存配置文件。

1.10.4. 更新 Mixer 策略强制功能

在之前的 Red Hat OpenShift Service Mesh 版本中，Mixer 的策略强制功能被默认启用。现在，Mixer 的策略强制功能被默认禁用。您需要在运行策略任务前启用它。

先决条件

- 访问 OpenShift Container Platform 命令行界面 (CLI) 也称为 **oc**。

流程

1. 登录 OpenShift Container Platform CLI。
2. 运行这个命令检查当前的 Mixer 策略强制状态：

```
$ oc get cm -n istio-system istio -o jsonpath='{.data.mesh}' | grep disablePolicyChecks
```

3. 如果为 **disablePolicyChecks: true**，编辑 Service Mesh ConfigMap：

```
$ oc edit cm -n istio-system istio
```

4. 在 ConfigMap 中找到 **disablePolicyChecks: true**，把它的值改为 **false**。
5. 保存配置并退出编辑器。
6. 重新检查 Mixer 策略强制状态以确保其已被设置为 **false**。

1.10.4.1. 设置正确的网络策略

Service Mesh 在 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 OpenShift Container Platform 路由公开的网格中的服务。

- 进入网格的流量必须总是通过 ingress 网关以使 Istio 可以正常工作。
- 在没有处于任何网格的单独命名空间中为网格部署服务。
- 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 OpenShift Container Platform 路由到这些服务仍可以正常工作。

1.10.5. Bookinfo 示例应用程序

上游 Istio 项目有一个名为 Bookinfo 的示例，它由四个独立的微服务组成，用来演示各种 Istio 特性。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。页面上显示的是对书、书目详情（ISBN 、页数以及其它信息）以及对本书的评论。

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 v2 调用 **ratings** 服务，并以一到五个红色星来代表对本书的评分。

1.10.5.1. 安装 Bookinfo 应用程序

本教程介绍了创建 Bookinfo 项目的步骤，包括部署 Bookinfo 应用程序，以及在使用 Service Mesh 1.1.9 的 OpenShift Container Platform 上运行 Bookinfo。

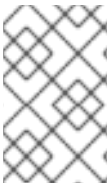
警告

您可以使用 Bookinfo 示例应用程序来测试 OpenShift Container Platform 中的 Red Hat OpenShift Service Mesh 1.1.9 安装。

红帽不提供对 Bookinfo 应用程序的支持。

先决条件

- 安装了 OpenShift Container Platform 4.1 或更高版本。
- 安装了 Red Hat OpenShift Service Mesh 1.1.9。
- 访问 OpenShift Container Platform 命令行界面 (CLI) 也称为 **oc**。



注意

Red Hat OpenShift Service Mesh 的自动注入与上游 Istio 项目不同，因此这里介绍的过程所使用的 **bookinfo.yaml** 文件版本已被注解为为 Red Hat OpenShift Service Mesh 启用 Istio sidecar 自动注入功能。

流程

1. 以具有 cluster-admin 权限的用户身份登录到 OpenShift Container Platform web 控制台。

2. 点 **Home → Projects**。
3. 点击 **Create Project**。
4. 在 **Project Name** 中输入 **bookinfo**，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 **bookinfo** 项目。

     ```
     $ oc new-project bookinfo
     ```
5. 点 **Operators → Installed Operators**。
6. 点击 **Project** 菜单，并使用 control plane 命名空间。在这个示例中，使用 **istio-system**。
7. 点 **Red Hat OpenShift Service Mesh** Operator。
8. 点 **Istio Service Mesh Member Roll**链接。
   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 Istio Service Mesh Member Roll，请点击 **Create Service Mesh Member Roll**。

   注意

   您需要 cluster-admin 权限来编辑 Istio Service Mesh Member Roll。
9. 编辑默认的 Service Mesh Member Roll YAML 并把 **bookinfo** 添加到 **members** 列表中。

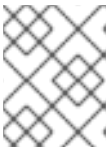
   ```
   apiVersion: maistra.io/v1
   kind: ServiceMeshMemberRoll
   metadata:
     name: default
   spec:
     members:
     - bookinfo
   ```
   - 另外，您还可以通过 CLI 运行这个命令，将 **bookinfo** 项目添加到 **ServiceMeshMemberRoll** 中。用 control plane 项目的名称来替换 **<control_plane_project>** 。

     ```
     $ oc -n <control_plane_project> patch --type='json' smmr default -p '[{"op": "add", "path": "/spec/members", "value":["""bookinfo"""]}]'
     ```
10. 点 **Create** 保存更新的 Service Mesh Member Roll。
11. 在 CLI 中，通过应用 **bookinfo.yaml** 文件在 `*bookinfo*` 项目中部署 Bookinfo：

    ```
    $ oc apply -n bookinfo -f https://raw.githubusercontent.com/Maistra/istio/maistra-1.1/samples/bookinfo/platform/kube/bookinfo.yaml
    ```
12. 通过应用 **bookinfo-gateway.yaml** 文件创建入站网关 ：

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

13. 设置 **GATEWAY_URL** 参数的值：



注意

用 control plane 项目的名称来替换 **<control_plane_project>** 。在本例中，control plane 项目为 **istio-system**。

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

1.10.5.2. 添加默认目的地规则

在使用 Bookinfo 应用程序前，您必须添加默认的目的地规则。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 bookinfo -f https://raw.githubusercontent.com/Maistra/istio/maistra-1.1/samples/bookinfo/networking/destination-rule-all.yaml
     ```

   - 如果启用了 nutual TLS：

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

1.10.5.3. 验证 Bookinfo 安装

在配置应用程序前，请确定它已被成功部署。

先决条件

- 安装了 OpenShift Container Platform 4.1 或更高版本。
- 安装了 Red Hat OpenShift Service Mesh 1.1.9。
- 访问 OpenShift Container Platform 命令行界面 (CLI) 也称为 **oc**。

流程

1. 登录 OpenShift Container Platform CLI。
2. 运行这个命令确认已部署了 Bookinfo：

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

- 另外，也可以通过在浏览器中打开 **http://$GATEWAY_URL/productpage** 来确认。
- 您还可以确认所有 pod 都可以使用这个命令：

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

1.10.5.4. 删除 Bookinfo 应用程序

按照以下步骤删除 Bookinfo 应用程序。

先决条件

- 安装了 OpenShift Container Platform 4.1 或更高版本。
- 安装了 Red Hat OpenShift Service Mesh 1.1.9。
- 访问 OpenShift Container Platform 命令行界面 (CLI) 也称为 **oc**。

1.10.5.4.1. 删除 Bookinfo 项目

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 点 **Home → Projects**。
3. 点击 **bookinfo** 菜单  ，然后点击 **Delete Project**。
4. 在确认对话框中输入 **bookinfo**，然后点 **Delete**。
   - 或者，也可以通过 CLI 运行这个命令来创建 **bookinfo** 项目。

```
$ oc delete project bookinfo
```

1.10.5.4.2. 从 Service Mesh member roll 中删除 Bookinfo 项目

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 点 **Operators → Installed Operators**。
3. 点 **Project** 菜单，从列表中选择 **openshift-operators**。
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 **bookinfo**。

- 另外，您还可以通过 CLI 运行这个命令从 **ServiceMeshMemberRoll** 中删除 **bookinfo** 项目。用 control plane 项目的名称来替换 **<control_plane_project>** 。

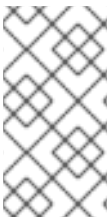
```
$ oc -n <control_plane_project> patch --type='json' smmr default -p '[{"op": "remove", "path": "/spec/members", "value":[""bookinfo""]}]'
```

7. 点 **Save** 更新 Service Mesh Member Roll。

1.10.6. 生成追踪示例并分析 trace 数据

Jaeger 是一个开源分布式追踪系统。您可以使用 Jaeger 监控基于微服务的分布式系统并进行故障排除。使用 Jaeger，您可以执行一个追踪（trace），该 trace 会追踪一个请求在组成应用程序的各种微服务间执行的路径。默认安装 Jaeger 作为 Service Mesh 的一部分。

本教程使用 Service Mesh 和 bookinfo 指南来演示如何使用 Jaeger 来执行分布式追踪。



注意

您可以使用 Bookinfo 示例应用程序来测试 OpenShift Container Platform 中的 Red Hat OpenShift Service Mesh 1.1.9 安装。

红帽不提供对 Bookinfo 应用程序的支持。

本教程使用 Service Mesh 和 Bookinfo 指南来演示如何使用 Red Hat OpenShift Service Mesh 的 Jaeger 组件来执行追踪。

先决条件

- 安装了 OpenShift Container Platform 4.1 或更高版本。
- 安装了 Red Hat OpenShift Service Mesh 1.1.9。
- 安装过程中启用了 Jaeger 。
- 已安装 Bookinfo 示例应用程序。

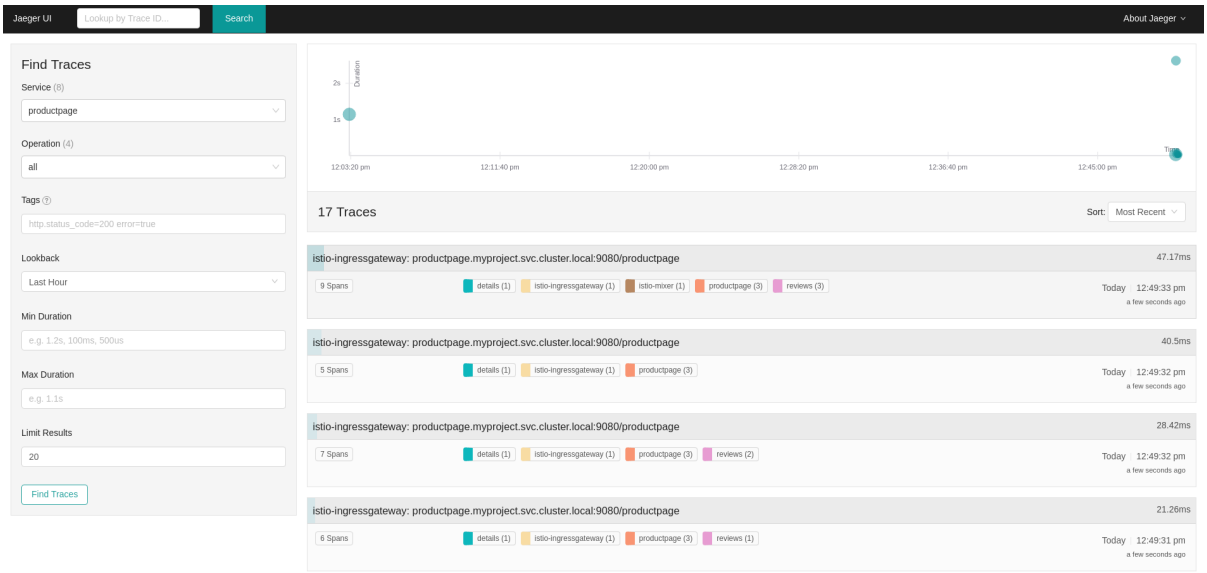
流程

1. 在部署了 Bookinfo 应用程序后，您需要生成对 Bookinfo 应用程序的调用，以便获得一些要分析的追踪数据。访问 http://<GATEWAY_URL>/productpage ,刷新该页面几次以生成一些跟踪数据。
2. 安装过程会创建路由来访问 Jaeger 控制台。
   a. 在 OpenShift Container Platform 控制台中，进入 **Networking → Routes** 并搜索 Jaeger 路由，它是 **Location** 项下列出的 URL。
   b. 使用 CLI 查询路由详情：

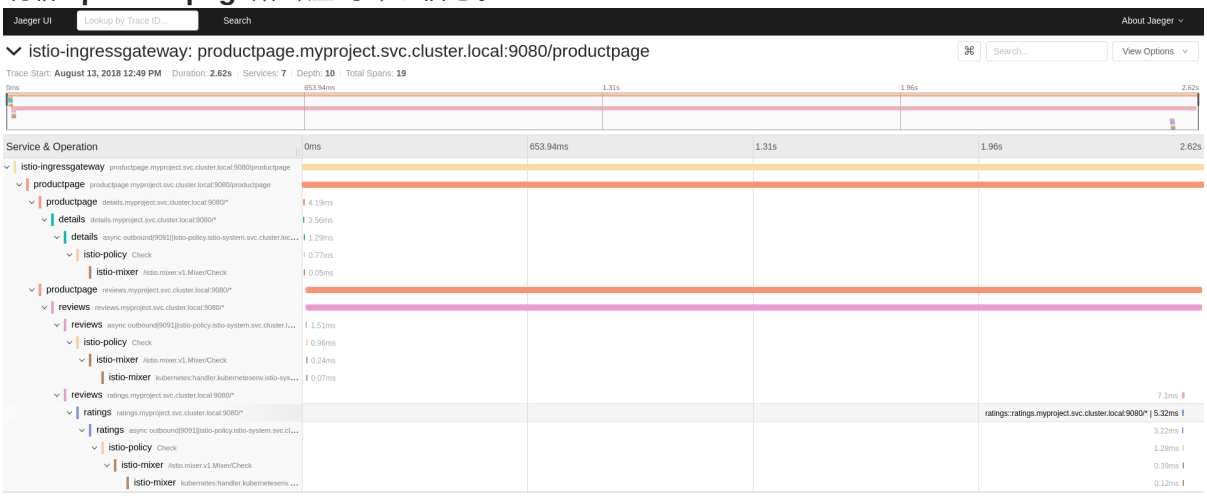
```
$ export JAEGER_URL=$(oc get route -n bookinfo jaeger-query -o jsonpath='{.spec.host}')
```

3. 启动浏览器并访问 **https://<JAEGER_URL>**。
4. 如有必要，使用与您用来访问 OpenShift Container Platform 控制台相同的用户名和密码登录。

5. 在 Jaeger dashboard 左侧的 dashboard 中，从 Service 菜单中选择 "productpage" 并点击面板底部的 **Find Traces** 按钮。此时会显示一个跟踪列表，如下所示：



6. 点击列表中的某个跟踪打开那个追踪的详细视图。如果点击顶部（最新）追踪，您会看到与最新刷新 **'/productpage**所对应的详细信息。



上图中的 trace 由几个嵌套的 spans 组成，每个 spans 对应一个 Bookinfo Service 调用，它们都是针对 **'/productpage** 请求执行的。总的处理时间是 2.62s，**details** Service 为 3.56ms，**reviews** Service 使用了 2.6s，**ratings** Service 使用了 5.32ms。每个远程服务调用都由客户端和服务器端的 span 代表。例如，**details** 客户端 span 被标记为 **productpage details.myproject.svc.cluster.local:9080**。在它下面嵌套的标记为 **details.myproject.svc.cluster.local:9080** 的 span 与服务器端的请求相对应。该 trace 还显示对 **istio-policy** 的调用，它反映了 Istio 的验证检查。

1.11. 数据可视化和可观察性

您可以在 Kiali 控制台中查看应用程序的拓扑、健康和指标。如果您的服务出现问题，Kiali 控制台提供了一种通过服务可视化数据流的方法。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。它还提供了您的命名空间中的实时互动图形视图。

如果您安装了一个应用程序，您可以观察到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 中的可观察性的信息。

安装 Bookinfo 示例应用程序后，将流量发送到网格。请多次输入以下命令：

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

如果您的示例应用程序配置正确，这个命令会模拟访问应用程序 **productpage** 微服务的用户。

1.11.1. 访问 Kiali 控制台

要访问控制台，在菜单栏中点击 **Application launcher** > **Kiali**。

1. 在 OpenShift Container Platform 菜单栏中,点击 **Application launcher** > **Kiali**。
2. 使用与您用来访问 OpenShift Container Platform 控制台相同的用户名和密码登录到 Kiali 控制台。
3. 在 **Namespace** 字段中为您的服务选择项目。如果您已经安装了 Bookinfo 示例，请选择 **bookinfo**。

使用命令行的步骤

1. 从 CLI 运行这个命令获得路由和 Kiali URL：

   ```
   $ oc get routes
   ```

   在 **kiali** 行的输出中，使用 HOST/PORT 栏中的 URL 来打开 Kiali 控制台。使用与您用来访问 OpenShift Container Platform 控制台相同的用户名和密码登录到 Kiali 控制台。在 **Namespace** 字段中为您的服务选择项目。

第一次登录时，您会看到 Overview 页，它会显示网格中的您有权查看的所有命名空间。

1.11.2. 视觉化您的服务

Kiali Operator 会处理 Red Hat OpenShift Service Mesh 收集的遥测数据，以提供命名空间中应用程序、服务和工作负载的数据图形和实时网络图。

Overview 页显示了您在网格中存在服务的所有命名空间。您可以对使用您的服务网格中的数据进行深入了解，或者通过以下图形和视觉化识别服务网格中的服务或工作负载的问题。

1.11.2.1. 命名空间图

命名空间图是命名空间中的服务、部署和工作流的映射，其中的箭头显示数据如何在其中进行传输。查看命名空间图：

1. 在主导航中点击 **Graph**。
2. 从 **Namespace** 菜单中选 **bookinfo**。

如果您的应用程序使用版本标签（如 Bookinfo 示例应用程序），您可以查看 Version 图。从 Graph Type 下拉菜单中选择图形。可以选择的几个图：

- App 图显示所有带有相同标记（label）的应用程序的总工作负载。
- Versioned App 图显示每个应用程序版本的节点。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 app 和 version 标签。如果您的应用程序没有使用 version 标签，请使用此图。

- 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。

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。

1.12. 在 SERVICE MESH 中自定义安全性

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。OpenShift Container Platform 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，并为微服务提供服务和身份安全。

1.12.1. 启用 mutual Transport Layer Security（mTLS）

Mutual Transport Layer Security（mTLS）是一个双方可以同时相互验证对方的协议。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。

MTLS 可在不更改应用程序或服务代码的情况下使用。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 设置为 permissive 模式，Service Mesh 的 sidecar 接受明文网络流量以及使用 mTLS 加密的网络连接。如果网格中的服务需要与网格外的服务进行通信，则 strict 模式的 mTLS 可能会破坏这些服务之间的通信。在将工作负载迁移到 Service Mesh 时使用 permissive 模式。

1.12.1.1. 在网格中启用 strict 模式的 mTLS

如果您的工作负载没有与网格之外的服务进行通信，且只接受加密的连接不会破坏这些通信，则可以在您的网格间快速启用 mTLS。在 **ServiceMeshControlPlane** 资源中将 **spec.istio.global.mTLS.enabled** 设置为 **true**。operator 创建所需资源。

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

1.12.1.1.1. 为特定服务的入站连接配置 sidecar

您可以通过创建一个策略来为各个服务或命名空间配置 mTLS。

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

1.12.1.2. 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。

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

1.12.1.3. 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您可以通过设置 **ServiceMeshControlPlane** 资源中的 **spec.istio.global.tls.minProtocolVersion** 或 **spec.istio.global.tls.maxProtocolVersion** 来控制允许的加密功能。这些值在 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。

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

默认为 **TLS_AUTO**，且不指定 TLS 版本。

表 1.12. 有效值

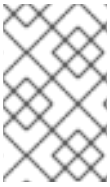
值	描述
TLS_AUTO	default
TLSv1_0	TLS 版本 1.0
TLSv1_1	TLS 版本 1.1
TLSv1_2	TLS 版本 1.2
TLSv1_3	TLS 版本 1.3

1.12.2. 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.istio.global.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

1.12.3. 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 Operator 指定的证书和密钥使用 Operator 指定的 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。

先决条件

- 您必须已安装了启用了 mutual TLS 配置证书的 Red Hat OpenShift Service Mesh。
- 这个示例使用 Maistra repo 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。
- 您必须部署 Bookinfo 示例应用程序以按照以下说明验证结果。

1.12.3.1. 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。

按照以下步骤将证书添加到 Service Mesh。本地保存 Maistra repo 中的证书，并将 **<path>** 替换为您的证书的路径。

1. 创建一个 secret **cacert**，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
```

2. 在 **ServiceMeshControlPlane** 资源中将 **global.mtls.enabled** 设置为 **true**，并将 **security.selfSigned** 设置为 **false**。Service Mesh 从 secret-mount 文件中读取证书和密钥。

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

3. 要确保工作负载迅速添加新证书，请删除名为 **istio.*** 的 Service Mesh 生成的 secret。在这个示例中, **istio.default**。Service Mesh 为工作负载发布新证书。

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

1.12.3.2. 验证您的证书

使用 Bookinfo 示例应用程序验证您的证书被正确挂载。首先，检索挂载的证书。然后，验证 pod 上挂载的证书。

1. 将 pod 名称存储在 **RATINGSPOD** 变量中。

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

运行以下命令以检索代理上挂载的证书。

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

文件 **/tmp/pod-root-cert.pem** 包含向 pod 传播的根证书。

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

文件 **/tmp/pod-cert-chain.pem** 包含向 pod 传播的工作负载证书和 CA 证书。

2. 验证 root 证书与 Operator 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
```

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

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

预期输出为空。

3. 验证 CA 证书与 Operator 指定证书相同。将 **<path>** 替换为证书的路径。

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

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

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

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

预期输出为空。

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。

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

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

输出示例

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

1.12.3.3. 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**。

```
$ 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/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    global:
      mtls:
        enabled: true
    security:
      selfSigned: true
```

1.13. 流量管理

您可以控制 Red Hat OpenShift Service Mesh 中服务间的流量和 API 调用。例如，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。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，或者在一组服务上添加安全层。

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。

1.13.1. 路由和管理流量

通过在 YAML 文件中使用自定义资源定义在 Red Hat OpenShift Service Mesh 中添加您自己的流量配置来配置服务网格。

1.13.1.1. 使用虚拟服务的流量管理

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。将单一虚拟服务映射到很多服务特别有助于将单个应用程序变成由不同微服务构建的复合服务，而无需该服务的使用者进行任何改变。
- 配置流量规则与网关相结合，以控制入口和出口流量。

1.13.1.1.1. 配置虚拟服务

使用虚拟服务将请求路由到服务网格中的服务中。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 round-robin 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。

以下示例将请求路由到服务的不同版本，具体取决于用户连接到哪个应用程序。使用此命令来应用此示例 YAML 文件，或您创建的 YAML 文件。

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

1.13.1.2. 配置您的虚拟主机

以下小节解释了 YAML 文件中的每个字段，并解释了如何在虚拟服务中创建虚拟主机。

1.13.1.2.1. Hosts

hosts 字段列出了这些路由规则应用到的虚拟服务的用户可访问的地址。这是向服务发送请求时使用的地址。

虚拟服务主机名可以是 IP 地址、DNS 名称，或者根据平台而定，可以被是解析为完全限定域名的简短名称。

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

1.13.1.2.2. 路由规则

http 项包含虚拟服务的路由规则，描述路由 HTTP/1.1、HTTP2 和 gRPC 流量与 hosts 字段中指定的目的地匹配的条件和动作。路由规则包含您希望流量访问的目的地，根据您的用例，可能有零个或更多匹配条件。

匹配条件

示例中的第一个路由规则有一个条件，并以 match 字段开头。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:
  hosts:
  - reviews
  http:
  - match:
    - headers:
      end-user:
        exact: jason
```

目的地

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，主机名一个是 Kubernetes 服务名称：

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

1.13.1.2.3. 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。

1.13.1.2.3.1. 负载平衡选项

默认情况下，Red Hat OpenShift Service Mesh 使用 round-robin 负载均衡策略，其中实例池中的每个服务实例依次获得请求。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
```

1.13.1.2.4. 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 网关资源可层 4-6 负载均衡属性，比如要公开的端口、Red Hat OpenShift Service Mesh TLS 设置。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，允许您限制哪些服务可访问外部网络，或者启用安全控制出口流量以为网格添加安全性。您还可以使用网关配置纯内部代理。

网关示例

以下示例显示了外部 HTTPS 入口流量的可能网关配置：

```
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。

1.13.1.2.5. 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 代理可将流量发送到该服务，就像在您的网格中是一个服务一样。配置服务条目可让您管理网格外运行的服务的流量，包括以下任务：

- 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。
- 在逻辑上，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。
- 您不需要为每个希望使用网格服务的外部服务添加服务条目。默认情况下，Red Hat OpenShift Service Mesh 配置 Envoy 代理，将请求传递给未知服务。但是，您无法使用 Red Hat OpenShift Service Mesh 功能来控制发送到没有在网格中注册的到目的地的网络流量。

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
```

1.13.1.2.6. 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。
- 您可能想要限制大型应用程序中的 sidecar 可访问性，每个代理都配置为访问网格中的所有其他服务都可能会影响高内存用量的网格性能。

Sidecar 示例

您可以指定是否希望 sidecar 配置应用到特定命名空间中的所有工作负载，或使用 **workloadSelector** 选择特定工作负载。例如，以下 sidecar 配置将 **bookinfo** 命名空间中的所有服务配置为只访问在同一命名空间和 Red Hat OpenShift Service Mesh control plane 中运行的服务（当前需要使用 Red Hat OpenShift Service Mesh 策略和 telemetry 功能）：

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

1.13.2. 管理入口流量

在 Red Hat OpenShift Service Mesh 中，Ingress Gateway 启用了 Service Mesh 功能，如监控、安全和路由规则，用于进入集群的流量。将 Service Mesh 配置为使用 Service Mesh 网关在服务网格外公开服务。

1.13.2.1. 决定入口 IP 和端口

运行以下命令，以确定您的 Kubernetes 集群是否在支持外部负载均衡器的环境中运行：

```
$ 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。您可以使用服务的 节点端口 访问网关。

选择您的环境说明：

使用负载均衡器配置路由

如果您的环境有外部负载均衡器，请按照以下步骤操作。

设置入口 IP 和端口：

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

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

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

在一些环境中，需要使用主机名而不是 IP 地址来公开负载均衡器。在那种情况下，入口网关的 **EXTERNAL-IP** 值不是一个 IP 地址。相反，它是一个主机名，而上一个命令将无法设置 **INGRESS_HOST** 环境变量。

使用以下命令更正 **INGRESS_HOST** 的值：

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

在没有负载均衡器的情况下配置路由

如果您的环境没有外部负载均衡器，请按照以下步骤操作。您必须使用节点端口。

设置入口端口：

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

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

1.13.3. 使用 bookinfo 应用程序的路由示例

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。部署了三个不同的版本，一个名为 **reviews** 的微服务同时运行。

先决条件

- 部署 Bookinfo 示例应用程序以使用以下示例。

关于这个任务

要说明这个问题，请在浏览器中访问 bookinfo 应用的 **/productpage**，并多次刷新。

有时，书的评论输出中会包含星号分级，有时则没有分级信息。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。

1.13.3.1. 应用虚拟服务

要只路由到一个版本，请应用为微服务设置默认版本的虚拟服务。在以下示例中，虚拟服务将所有流量路由到每个微服务的 **v1**

1. 运行以下命令以应用虚拟服务：

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

2. 要测试该命令是否成功，请使用以下命令显示定义的路由：

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

该命令返回以下 YAML 文件。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: details
  ...
spec:
  hosts:
  - details
  http:
  - route:
    - destination:
      host: details
      subset: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
```

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

您已将 Service Mesh 配置为路由至 Bookinfo 微服务的 **v1** 版本，最重要的是 **reviews** 服务版本 1。

1.13.3.2. 测试新的路由配置

您可以通过刷新 Bookinfo 应用程序的 **/productpage** 来轻松地测试新配置。

1. 在浏览器中打开 Bookinfo 网站。URL 为 **http://$GATEWAY_URL/productpage**，其中 **$GATEWAY_URL** 是入口的外部 IP 地址。
   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。

1.13.3.3. 基于用户身份的路由

接下来，修改路由配置，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。

请注意，Service Mesh 没有任何特殊的内置用户身份了解功能。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。

1. 运行以下命令启用基于用户的路由：

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

2. 确认创建了规则：

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

该命令返回以下 YAML 文件。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
  ...
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
      end-user:
        exact: jason
    route:
    - destination:
      host: reviews
      subset: v2
  - route:
    - destination:
      host: reviews
      subset: v1
```

3. 在 Bookinfo 应用程序的 **/productpage** 中以用户 **Jason** 的身份登陆。刷新浏览器。您会看到什么？星级分级会出现在每条评论旁。

4. 以其他用户身份登录（选择任何名称）。刷新浏览器。现在就不会出现星级评分。这是因为除 Jason 外，所有用户的流量都会路由到 **review:v1**。

您已成功配置了 Service Mesh，以根据用户身份路由流量。

1.14. 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。

1.14.1. 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 1.x
- 一个有效的 3scale 账户 (SaaS 或 3scale 2.5 On-Premises)
- 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。

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 凭据和您要管理的 API 服务 ID 更新此功能。

1. 使用 3scale 配置修改处理器配置。

处**理器配置示例**

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

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。

1. 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
```

1.14.1.1. 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 1.13. 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	
--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	

1.14.1.1.1. 从 URL 示例生成模板

- 这个示例生成模板，允许多个服务作为单一处理器共享令牌， URL 对：

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

- 这个示例生成带有内嵌在处理器中的服务 ID 的模板：

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

1.14.1.2. 从部署的适配器生成清单

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}""
```

1.14.1.3. 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。

1.14.2. 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 → Configuration**。
2. 点 **Integration** 页右上角的 **edit integration settings**。
3. 在 **Service Mesh** 标题下，点 **Istio** 选项。
4. 滚动到页面底部并点击 **Update Service**。

1.14.3. 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。

1.14.4. 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。

1.14.4.1. 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。

1.14.4.1.1. 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"]**。

1.14.4.1.2. 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"]**。

1.14.4.1.3. 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

```
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。您应该为需要保护的服务在相同的命名空间内创建最终用户验证 。JWT 由请求的 **Authorization** 标头传递。

在下面定义的 **Policy** 示例中，根据情况替换 **issuer** 和 **jwksUri**。

OpenID 策略示例

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

```
    principalBinding: USE_ORIGIN
    targets:
      - name: productpage
```

1.14.4.1.4. 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"] | ""
```

1.14.5. 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。