OpenShift Container Platform 4.12

Service Mesh

Installation, utilisation et notes de mise à jour de Service Mesh

Last Updated: 2023-05-04

OpenShift Container Platform 4.12 Service Mesh

Installation, utilisation et notes de mise à jour de Service Mesh

Notice légale

Résumé

Ce document fournit des informations sur l'utilisation de Service Mesh dans OpenShift Container Platform.

Table des matières

CHAPITRE 1. SERVICE MESH 2.X

1.1. À PROPOS D'OPENSHIFT SERVICE MESH

NOTE

Étant donné que Red Hat OpenShift Service Mesh est publié à une cadence différente de celle d'OpenShift Container Platform et que Red Hat OpenShift Service Mesh Operator prend en charge le déploiement de plusieurs versions du site **ServiceMeshControlPlane**, la documentation de Service Mesh ne maintient pas de jeux de documentation distincts pour les versions mineures du produit. Le jeu de documentation actuel s'applique à la version la plus récente de Service Mesh, à moins que des limitations spécifiques à la version ne soient indiquées dans un sujet particulier ou pour une fonctionnalité particulière.

Pour plus d'informations sur le cycle de vie de Red Hat OpenShift Service Mesh et les plateformes prises en charge, reportez-vous à la Politique relative au cycle de vie des plateformes.

1.1.1. Introduction à Red Hat OpenShift Service Mesh

Red Hat OpenShift Service Mesh répond à une variété de problèmes dans une architecture de microservices en créant un point de contrôle centralisé dans une application. Il ajoute une couche transparente sur les applications distribuées existantes sans nécessiter de modifications du code de l'application.

Les architectures de microservices divisent le travail des applications d'entreprise en services modulaires, ce qui peut faciliter la mise à l'échelle et la maintenance. Cependant, lorsqu'une application d'entreprise construite sur une architecture de microservices augmente en taille et en complexité, elle devient difficile à comprendre et à gérer. Le Service Mesh peut résoudre ces problèmes d'architecture en capturant ou en interceptant le trafic entre les services et peut modifier, rediriger ou créer de nouvelles requêtes vers d'autres services.

Service Mesh, qui est basé sur le projet open source Istio, permet de créer facilement un réseau de services déployés qui assure la découverte, l'équilibrage de charge, l'authentification de service à service, la reprise sur panne, les mesures et la surveillance. Un maillage de services offre également des fonctionnalités opérationnelles plus complexes, notamment les tests A/B, les versions canary, le contrôle d'accès et l'authentification de bout en bout.

1.1.2. Caractéristiques principales

Red Hat OpenShift Service Mesh fournit un certain nombre de capacités clés de manière uniforme à travers un réseau de services :

- **Traffic Management** - Contrôler le flux de trafic et les appels API entre les services, rendre les appels plus fiables et rendre le réseau plus robuste face à des conditions défavorables.
- **Service Identity and Security**- Fournir aux services du réseau maillé une identité vérifiable et offrir la possibilité de protéger le trafic des services lorsqu'il circule sur des réseaux plus ou moins fiables.
- **Policy Enforcement** - Appliquer la politique de l'organisation à l'interaction entre les services, veiller à ce que les politiques d'accès soient appliquées et que les ressources soient équitablement réparties entre les consommateurs. Les changements de politique se font en

configurant le maillage, et non en modifiant le code de l'application.

- **Telemetry** - Comprendre les dépendances entre les services ainsi que la nature et le flux du trafic entre eux, ce qui permet d'identifier rapidement les problèmes.

1.2. SERVICE MESH - NOTES DE MISE À JOUR

1.2.1. Rendre l'open source plus inclusif

Red Hat s'engage à remplacer les termes problématiques dans son code, sa documentation et ses propriétés Web. Nous commençons par ces quatre termes : master, slave, blacklist et whitelist. En raison de l'ampleur de cette entreprise, ces changements seront mis en œuvre progressivement au cours de plusieurs versions à venir. Pour plus de détails, voir le message de notre directeur technique Chris Wright.

1.2.2. Nouvelles fonctionnalités et améliorations

Cette version apporte des améliorations concernant les composants et concepts suivants.

1.2.2.1. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh version 2.3.3

Cette version de Red Hat OpenShift Service Mesh corrige les vulnérabilités et expositions communes (CVE), contient des corrections de bugs et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.1.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.3.3

Composant	Version
Istio	1.14.5
Envoy Proxy	1.22.9
Jaeger	1.42.0
Kiali	1.57.7

1.2.2.2. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh version 2.3.2

Cette version de Red Hat OpenShift Service Mesh corrige les vulnérabilités et expositions communes (CVE), contient des corrections de bugs et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.2.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.3.2

Composant	Version
Istio	1.14.5

Composant	Version
Envoy Proxy	1.22.7
Jaeger	1.39
Kiali	1.57.6

1.2.2.3. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh version 2.3.1

Cette version de Red Hat OpenShift Service Mesh introduit de nouvelles fonctionnalités, traite les vulnérabilités et expositions communes (CVE), contient des corrections de bugs et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.3.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.3.1

Composant	Version
Istio	1.14.5
Envoy Proxy	1.22.4
Jaeger	1.39
Kiali	1.57.5

1.2.2.4. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh version 2.3

Cette version de Red Hat OpenShift Service Mesh introduit de nouvelles fonctionnalités, traite les vulnérabilités et expositions communes (CVE), contient des corrections de bugs, et est prise en charge sur OpenShift Container Platform 4.9, 4.10, et 4.11.

1.2.2.4.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.3

Composant	Version
Istio	1.14.3
Envoy Proxy	1.22.4
Jaeger	1.38
Kiali	1.57.3

1.2.2.4.2. Nouveau conteneur DaemonSet et ConfigMap de l'interface réseau de conteneurs (CNI)

L'espace de noms **openshift-operators** comprend un nouveau DaemonSet istio CNI **istio-cni-node-v2-3** et une nouvelle ressource **ConfigMap**, **istio-cni-config-v2-3**.

Lors de la mise à niveau vers le Service Mesh Control Plane 2.3, le DaemonSet **istio-cni-node** existant n'est pas modifié et un nouveau DaemonSet **istio-cni-node-v2-3** est créé.

Ce changement de nom n'affecte pas les versions précédentes ou tout **istio-cni-node** CNI DaemonSet associé à un plan de contrôle Service Mesh déployé à l'aide d'une version précédente.

1.2.2.4.3. Soutien à l'injection de la passerelle

Cette version introduit la prise en charge généralement disponible de l'injection de passerelle. Les configurations de passerelle sont appliquées aux proxys Envoy autonomes qui s'exécutent à la périphérie du maillage, plutôt qu'aux proxys Envoy sidecar qui s'exécutent aux côtés de vos charges de travail de service. Cela permet de personnaliser les options de passerelle. Lorsque vous utilisez l'injection de passerelle, vous devez créer les ressources suivantes dans l'espace de noms où vous souhaitez exécuter votre proxy de passerelle : **Service**, **Deployment**, **Role**, et **RoleBinding**.

1.2.2.4.4. Prise en charge d'Istio 1.14

Service Mesh 2.3 est basé sur Istio 1.14, qui apporte de nouvelles fonctionnalités et des améliorations au produit. Bien que de nombreuses fonctionnalités d'Istio 1.14 soient prises en charge, il convient de noter les exceptions suivantes :

- L'API ProxyConfig est prise en charge à l'exception du champ image.
- L'API de télémétrie est une fonctionnalité de l'aperçu technologique.
- La durée d'exécution de SPIRE n'est pas une fonctionnalité prise en charge.

1.2.2.4.5. Console OpenShift Service Mesh

IMPORTANT

OpenShift Service Mesh Console est une fonctionnalité d'aperçu technologique uniquement. Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas de les utiliser en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, ce qui permet aux clients de tester les fonctionnalités et de fournir des commentaires pendant le processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

Cette version introduit une version Technology Preview de la console OpenShift Container Platform Service Mesh, qui intègre l'interface Kiali directement dans la console web OpenShift. Pour plus d'informations, voir Introducing the OpenShift Service Mesh Console (A Technology Preview)

1.2.2.4.6. Déploiement à l'échelle du cluster

Cette version introduit le déploiement à l'échelle du cluster en tant que fonctionnalité de l'aperçu technologique. Un déploiement à l'échelle d'un cluster contient un plan de contrôle Service Mesh qui surveille les ressources pour un cluster entier. Le plan de contrôle utilise une seule requête dans tous les

espaces de noms pour surveiller chaque type de ressource Istio ou Kubernetes qui affecte la configuration du maillage. En revanche, l'approche multitenant utilise une requête par espace de noms pour chaque type de ressource. La réduction du nombre de requêtes effectuées par le plan de contrôle dans le cadre d'un déploiement à l'échelle du cluster améliore les performances.

1.2.2.4.6.1. Configuration du déploiement à l'échelle d'un cluster

L'exemple suivant d'objet **ServiceMeshControlPlane** configure un déploiement à l'échelle d'un cluster.

Pour créer un SMCP en vue d'un déploiement à l'échelle du cluster, un utilisateur doit appartenir au ClusterRole **cluster-admin**. Si le SMCP est configuré pour un déploiement à l'échelle du cluster, il doit être le seul SMCP du cluster. Vous ne pouvez pas modifier le mode du plan de contrôle de multitenant à cluster-wide (ou de cluster-wide à multitenant). Si un plan de contrôle multitenant existe déjà, supprimez-le et créez-en un nouveau.

Cet exemple configure le SMCP pour un déploiement à l'échelle du cluster.

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: cluster-wide
  namespace: istio-system
spec:
  version: v2.3
  techPreview:
    controlPlaneMode: ClusterScoped (1)
```

(1) Permet à Istiod de surveiller les ressources au niveau du cluster plutôt que de surveiller chaque espace de noms individuel.

En outre, le SMMR doit également être configuré pour un déploiement à l'échelle du cluster. Cet exemple configure le SMMR pour un déploiement à l'échelle du cluster.

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

(1) Ajoute tous les espaces de noms au maillage, y compris les espaces de noms que vous créez par la suite. Les espaces de noms suivants ne font pas partie du maillage : kube, openshift, kube-* et openshift-*.

1.2.2.5. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh version 2.2.6

Cette version de Red Hat OpenShift Service Mesh corrige les vulnérabilités et expositions communes (CVE), contient des corrections de bugs et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.5.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.2.6

Composant	Version
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.39
Kiali	1.48.5

1.2.2.6. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh version 2.2.5

Cette version de Red Hat OpenShift Service Mesh corrige les vulnérabilités et expositions communes (CVE), contient des corrections de bugs et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.6.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.2.5

Composant	Version
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.39
Kiali	1.48.3

1.2.2.7. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh version 2.2.4

Cette version de Red Hat OpenShift Service Mesh corrige les vulnérabilités et expositions communes (CVE), contient des corrections de bugs et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.7.1. Versions des composants inclus dans Red Hat OpenShift Service Mesh version 2.2.4

Composant	Version
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.36.14
Kiali	1.48.3

1.2.2.8. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh version 2.2.3

Cette version de Red Hat OpenShift Service Mesh traite les vulnérabilités et expositions communes (CVE), les corrections de bogues, et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.8.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.2.3

Composant	Version
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.36
Kiali	1.48.3

1.2.2.9. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh version 2.2.2

Cette version de Red Hat OpenShift Service Mesh traite les vulnérabilités et expositions communes (CVE), les corrections de bogues, et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.9.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.2.2

Composant	Version
Istio	1.12.7
Envoy Proxy	1.20.6
Jaeger	1.36
Kiali	1.48.2-1

1.2.2.9.2. Copier les étiquettes d'itinéraire

Avec cette amélioration, en plus de copier les annotations, vous pouvez copier des étiquettes spécifiques pour une route OpenShift. Red Hat OpenShift Service Mesh copie toutes les étiquettes et annotations présentes dans la ressource Istio Gateway (à l'exception des annotations commençant par kubectl.kubernetes.io) dans la ressource OpenShift Route gérée.

1.2.2.10. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh version 2.2.1

Cette version de Red Hat OpenShift Service Mesh traite les vulnérabilités et expositions communes (CVE), les corrections de bogues, et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.10.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.2.1

Composant	Version
Istio	1.12.7
Envoy Proxy	1.20.6
Jaeger	1.34.1
Kiali	1.48.2-1

1.2.2.11. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.2

Cette version de Red Hat OpenShift Service Mesh ajoute de nouvelles fonctionnalités et améliorations, et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.11.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.2

Composant	Version
Istio	1.12.7
Envoy Proxy	1.20.4
Jaeger	1.34.1
Kiali	1.48.0.16

1.2.2.11.2. **WasmPlugin** API

Cette version ajoute la prise en charge de l'API **WasmPlugin** et supprime l'API **ServiceMeshExtension**.

1.2.2.11.3. Soutien de ROSA

Cette version introduit la prise en charge du maillage de services pour Red Hat OpenShift sur AWS (ROSA), y compris la fédération multi-cluster.

1.2.2.11.4. **istio-node** DaemonSet renommé

Dans cette version, le DaemonSet **istio-node** est renommé **istio-cni-node** pour correspondre au nom dans Istio en amont.

1.2.2.11.5. Changements dans la mise en réseau du sidecar Envoy

Istio 1.10 a mis à jour Envoy pour envoyer le trafic au conteneur d'application en utilisant **eth0** au lieu de **lo** par défaut.

1.2.2.11.6. Service Mesh Control Plane 1.1

Cette version marque la fin de la prise en charge des plans de contrôle Service Mesh basés sur Service Mesh 1.1 pour toutes les plateformes.

1.2.2.11.7. Prise en charge d'Istio 1.12

Service Mesh 2.2 est basé sur Istio 1.12, qui apporte de nouvelles fonctionnalités et des améliorations au produit. Bien que de nombreuses fonctionnalités d'Istio 1.12 soient prises en charge, il convient de noter les fonctionnalités suivantes qui ne sont pas prises en charge :

- AuthPolicy Dry Run est une fonctionnalité de prévisualisation technique.
- gRPC Proxyless Service Mesh est une fonction de prévisualisation technique.
- L'API de télémétrie est une fonction de prévisualisation technique.
- Les sélecteurs de découverte ne sont pas une fonctionnalité prise en charge.
- Le plan de contrôle externe n'est pas une fonctionnalité prise en charge.
- L'injection de passerelle n'est pas une fonctionnalité prise en charge.

1.2.2.11.8. API de la passerelle Kubernetes

IMPORTANT

Kubernetes Gateway API est une fonctionnalité d'aperçu technologique uniquement. Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas de les utiliser en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, permettant aux clients de tester les fonctionnalités et de fournir un retour d'information pendant le processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

Kubernetes Gateway API est une fonctionnalité d'aperçu technologique qui est désactivée par défaut. Si le contrôleur de déploiement de l'API Kubernetes est désactivé, vous devez déployer et lier manuellement une passerelle d'entrée à l'objet Gateway créé.

Si le contrôleur de déploiement de l'API Kubernetes est activé, une passerelle d'entrée est automatiquement déployée lorsqu'un objet Gateway est créé.

1.2.2.11.8.1. Installation des CRD de l'API de la passerelle

Les CRDs de l'API Gateway ne sont pas pré-installés par défaut sur les clusters OpenShift. Installez les CRD avant d'activer la prise en charge de l'API Gateway dans le SMCP.

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

1.2.2.11.8.2. Activation de l'API de la passerelle Kubernetes

Pour activer cette fonctionnalité, définissez les variables d'environnement suivantes pour le conteneur **Istiod** dans **ServiceMeshControlPlane**:

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

Il est possible de restreindre l'attachement des routes sur les auditeurs de l'API Gateway en utilisant les paramètres **SameNamespace** ou **All**. Istio ignore l'utilisation des sélecteurs d'étiquettes dans **listeners.allowedRoutes.namespaces** et revient au comportement par défaut (**SameNamespace**).

1.2.2.11.8.3. Relier manuellement une passerelle existante à une ressource Gateway

Si le contrôleur de déploiement de l'API Kubernetes est désactivé, vous devez déployer manuellement puis lier une passerelle d'entrée à la ressource Gateway créée.

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

1.2.2.12. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.1.6

Cette version de Red Hat OpenShift Service Mesh corrige les vulnérabilités et expositions communes (CVE), contient des corrections de bugs et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.12.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.1.6

Composant	Version
Istio	1.9.9
Envoy Proxy	1.17.5
Jaeger	1.36
Kiali	1.36.16

1.2.2.13. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.1.5.2

Cette version de Red Hat OpenShift Service Mesh corrige les vulnérabilités et expositions communes (CVE), contient des corrections de bugs et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.13.1. Versions des composants inclus dans Red Hat OpenShift Service Mesh version 2.1.5.2

Composant	Version
Istio	1.9.9
Envoy Proxy	1.17.5
Jaeger	1.36
Kiali	1.24.17

1.2.2.14. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.1.5.1

Cette version de Red Hat OpenShift Service Mesh traite les vulnérabilités et expositions communes (CVE), les corrections de bogues, et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.14.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.1.5.1

Composant	Version
Istio	1.9.9
Envoy Proxy	1.17.5
Jaeger	1.36
Kiali	1.36.13

1.2.2.15. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.1.5

Cette version de Red Hat OpenShift Service Mesh traite les vulnérabilités et expositions communes (CVE), les corrections de bogues, et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.15.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.1.5

Composant	Version
Istio	1.9.9
Envoy Proxy	1.17.1

Composant	Version
Jaeger	1.36
Kiali	1.36.12-1

1.2.2.16. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.1.4

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

1.2.2.16.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.1.4

Composant	Version
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.30.2
Kiali	1.36.12-1

1.2.2.17. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.1.3

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

1.2.2.17.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.1.3

Composant	Version
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.30.2
Kiali	1.36.10-2

1.2.2.18. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.1.2.1

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

1.2.2.18.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.1.2.1

Composant	Version
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.30.2
Kiali	1.36.9

1.2.2.19. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.1.2

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

Avec cette version, la plateforme de traçage distribuée Red Hat OpenShift Operator est désormais installée par défaut dans l'espace de noms **openshift-distributed-tracing**. Auparavant, l'installation par défaut se faisait dans l'espace de noms **openshift-operator**.

1.2.2.19.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.1.2

Composant	Version
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.30.1
Kiali	1.36.8

1.2.2.20. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.1.1

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

Cette version ajoute également la possibilité de désactiver la création automatique de politiques de réseau.

1.2.2.20.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.1.1

Composant	Version
Istio	1.9.9
Envoy Proxy	1.17.1

Composant	Version
Jaeger	1.24.1
Kiali	1.36.7

1.2.2.20.2. Désactivation des politiques de réseau

Red Hat OpenShift Service Mesh crée et gère automatiquement un certain nombre de ressources **NetworkPolicies** dans le plan de contrôle Service Mesh et les espaces de noms des applications. Cela permet de s'assurer que les applications et le plan de contrôle peuvent communiquer entre eux.

Si vous souhaitez désactiver la création et la gestion automatiques des ressources **NetworkPolicies**, par exemple pour appliquer les politiques de sécurité de votre entreprise, vous pouvez le faire. Vous pouvez modifier le site **ServiceMeshControlPlane** pour définir le paramètre **spec.security.manageNetworkPolicy** sur **false**

NOTE

Lorsque vous désactivez **spec.security.manageNetworkPolicy**, Red Hat OpenShift Service Mesh ne créera pas d'objets **any NetworkPolicy** . L'administrateur système est responsable de la gestion du réseau et de la résolution des problèmes que cela pourrait causer.

Procédure

1. Dans la console web d'OpenShift Container Platform, cliquez sur **Operators → Installed Operators**.
2. Sélectionnez le projet dans lequel vous avez installé le plan de contrôle Service Mesh, par exemple **istio-system**, dans le menu Projet.
3. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh. Dans la colonne **Istio Service Mesh Control Plane**, cliquez sur le nom de votre **ServiceMeshControlPlane**, par exemple **basic-install**.
4. Sur la page **Create ServiceMeshControlPlane Details**, cliquez sur **YAML** pour modifier votre configuration.
5. Définissez le champ **ServiceMeshControlPlane spec.security.manageNetworkPolicy** sur **false**, comme indiqué dans cet exemple.

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

6. Cliquez sur **Save**.

1.2.2.21. Nouvelles fonctionnalités et améliorations Red Hat OpenShift Service Mesh 2.1

Cette version de Red Hat OpenShift Service Mesh ajoute la prise en charge d'Istio 1.9.8, Envoy Proxy 1.17.1, Jaeger 1.24.1, et Kiali 1.36.5 sur OpenShift Container Platform 4.6 EUS, 4.7, 4.8, 4.9, ainsi que de nouvelles fonctionnalités et améliorations.

1.2.2.21.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.1

Composant	Version
Istio	1.9.6
Envoy Proxy	1.17.1
Jaeger	1.24.1
Kiali	1.36.5

1.2.2.21.2. Fédération de maillage de services

De nouvelles définitions de ressources personnalisées (CRD) ont été ajoutées pour prendre en charge la fédération des maillages de services. Les maillages de services peuvent être fédérés à la fois au sein d'un même cluster ou entre différents clusters OpenShift. Ces nouvelles ressources incluent :

- **ServiceMeshPeer** - Définit une fédération avec une maille de service distincte, y compris la configuration de la passerelle, la configuration du certificat de confiance racine et les champs d'état. Dans une paire de mailles fédérées, chaque maille définira sa propre ressource **ServiceMeshPeer**.
- **ExportedServiceMeshSet** - Définit les services d'un site **ServiceMeshPeer** donné que le réseau homologue peut importer.
- **ImportedServiceSet** - Définit les services d'un site **ServiceMeshPeer** donné qui sont importés du réseau homologue. Ces services doivent également être mis à disposition par la ressource **ExportedServiceMeshSet** de l'homologue.

Service Mesh Federation n'est pas pris en charge entre les clusters sur Red Hat OpenShift Service on AWS (ROSA), Azure Red Hat OpenShift (ARO), ou OpenShift Dedicated (OSD).

1.2.2.21.3. Interface de réseau de conteneurs OVN-Kubernetes (CNI) généralement disponible

L'interface OVN-Kubernetes Container Network Interface (CNI) a été précédemment introduite en tant que fonctionnalité d'aperçu technologique dans Red Hat OpenShift Service Mesh 2.0.1 et est maintenant généralement disponible dans Red Hat OpenShift Service Mesh 2.1 et 2.0.x pour une utilisation sur OpenShift Container Platform 4.7.32, OpenShift Container Platform 4.8.12, et OpenShift Container Platform 4.9.

1.2.2.21.4. Service Mesh WebAssembly (WASM) Extensions

Le site **ServiceMeshExtensions** Custom Resource Definition (CRD), introduit pour la première fois dans la version 2.0 en tant qu'aperçu technologique, est désormais disponible de manière générale. Vous pouvez utiliser la CRD pour créer vos propres plugins, mais Red Hat ne fournit pas de support pour les plugins que vous créez.

Mixer a été complètement supprimé dans Service Mesh 2.1. La mise à jour d'une version 2.0.x de Service Mesh vers la version 2.1 sera bloquée si Mixer est activé. Les plugins Mixer devront être portés vers WebAssembly Extensions.

1.2.2.21.5. adaptateur WebAssembly 3scale (WASM)

Avec le retrait officiel de Mixer, OpenShift Service Mesh 2.1 ne supporte pas l'adaptateur de mélangeur 3scale. Avant de passer à Service Mesh 2.1, supprimez l'adaptateur 3scale basé sur Mixer et tous les plugins Mixer supplémentaires. Ensuite, installez et configurez manuellement le nouvel adaptateur 3scale WebAssembly avec Service Mesh 2.1 en utilisant une ressource **ServiceMeshExtension**.

3scale 2.11 introduit une nouvelle intégration de Service Mesh basée sur **WebAssembly**.

1.2.2.21.6. Prise en charge d'Istio 1.9

Service Mesh 2.1 est basé sur Istio 1.9, qui apporte un grand nombre de nouvelles fonctionnalités et d'améliorations du produit. Bien que la majorité des fonctionnalités d'Istio 1.9 soient prises en charge, il convient de noter les exceptions suivantes :

- L'intégration des machines virtuelles n'est pas encore prise en charge
- L'API Kubernetes Gateway n'est pas encore prise en charge
- La récupération et le chargement à distance des filtres HTTP WebAssembly ne sont pas encore pris en charge
- L'intégration d'une autorité de certification personnalisée à l'aide de l'API CSR de Kubernetes n'est pas encore prise en charge
- La classification des demandes pour la surveillance du trafic est une fonctionnalité de l'avant-première technique
- L'intégration avec des systèmes d'autorisation externes via l'action CUSTOM de la politique d'autorisation est une fonctionnalité en avant-première technique

1.2.2.21.7. Amélioration des performances de l'opérateur Service Mesh

Le temps que Red Hat OpenShift Service Mesh utilise pour élaguer les anciennes ressources à la fin de chaque rapprochement **ServiceMeshControlPlane** a été réduit. Cela se traduit par des déploiements **ServiceMeshControlPlane** plus rapides, et permet aux changements appliqués aux SMCP existants de prendre effet plus rapidement.

1.2.2.21.8. Mises à jour de Kiali

Kiali 1.36 comprend les fonctionnalités et améliorations suivantes :

- Fonctionnalité de dépannage du maillage de services
  - Surveillance du plan de contrôle et de la passerelle
  - État de la synchronisation du proxy
  - Vues de la configuration d'Envoy
  - Vue unifiée montrant le proxy Envoy et les journaux d'application entrelacés
- Espace de noms et cluster boxing pour supporter les vues de maillage de services fédérés

- Nouvelles validations, assistants et améliorations du traçage distribué

1.2.2.22. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0.11.1

Cette version de Red Hat OpenShift Service Mesh traite les vulnérabilités et expositions communes (CVE), les corrections de bogues, et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.22.1. Versions de composants incluses dans la version 2.0.11.1 de Red Hat OpenShift Service Mesh

Composant	Version
Istio	1.6.14
Envoy Proxy	1.14.5
Jaeger	1.36
Kiali	1.24.17

1.2.2.23. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0.11

Cette version de Red Hat OpenShift Service Mesh traite les vulnérabilités et expositions communes (CVE), les corrections de bogues, et est prise en charge sur OpenShift Container Platform 4.9 ou une version ultérieure.

1.2.2.23.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.0.11

Composant	Version
Istio	1.6.14
Envoy Proxy	1.14.5
Jaeger	1.36
Kiali	1.24.16-1

1.2.2.24. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0.10

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

1.2.2.24.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.0.10

Composant	Version
Istio	1.6.14
Envoy Proxy	1.14.5
Jaeger	1.28.0
Kiali	1.24.16-1

1.2.2.25. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0.9

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

1.2.2.25.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 2.0.9

Composant	Version
Istio	1.6.14
Envoy Proxy	1.14.5
Jaeger	1.24.1
Kiali	1.24.11

1.2.2.26. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0.8

Cette version de Red Hat OpenShift Service Mesh traite des corrections de bogues.

1.2.2.27. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0.7.1

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE).

1.2.2.27.1. Changement dans la façon dont Red Hat OpenShift Service Mesh gère les fragments d'URI

Red Hat OpenShift Service Mesh contient une vulnérabilité exploitable à distance, CVE-2021-39156, où une requête HTTP avec un fragment (une section à la fin d'un URI qui commence par un caractère #) dans le chemin URI pourrait contourner les politiques d'autorisation basées sur le chemin URI d'Istio. Par exemple, une politique d'autorisation d'Istio refuse les requêtes envoyées au chemin URI **/user/profile**. Dans les versions vulnérables, une requête avec le chemin URI **/user/profile#section1** contourne la politique de refus et se dirige vers le backend (avec l'URI normalisé **path /user/profile#section1**), ce qui peut conduire à un incident de sécurité.

Vous êtes concerné par cette vulnérabilité si vous utilisez des politiques d'autorisation avec des actions DENY et **operation.paths**, ou des actions ALLOW et **operation.notPaths**.

Avec l'atténuation, la partie fragmentaire de l'URI de la demande est supprimée avant l'autorisation et l'acheminement. Cela empêche une demande contenant un fragment dans son URI de contourner les politiques d'autorisation qui sont basées sur l'URI sans la partie fragment.

Pour ne pas être concerné par le nouveau comportement dans l'atténuation, la section du fragment dans l'URI sera conservée. Vous pouvez configurer votre site **ServiceMeshControlPlane** pour qu'il conserve les fragments de l'URI.

AVERTISSEMENT

La désactivation de ce nouveau comportement normalisera vos chemins d'accès comme décrit ci-dessus et est considérée comme dangereuse. Assurez-vous d'avoir pris en compte ce problème dans vos politiques de sécurité avant de choisir de conserver les fragments d'URI.

Exemple ServiceMeshControlPlane modification

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  techPreview:
    meshConfig:
      defaultConfig:
        proxyMetadata: HTTP_STRIP_FRAGMENT_FROM_PATH_UNSAFE_IF_DISABLED: "false"
```

1.2.2.27.2. Mise à jour requise pour les politiques d'autorisation

Istio génère des noms d'hôtes à la fois pour le nom d'hôte lui-même et pour tous les ports correspondants. Par exemple, un service virtuel ou une passerelle pour un hôte de "httpbin.foo" génère une configuration correspondant à "httpbin.foo et httpbin.foo:*". Cependant, les politiques d'autorisation de correspondance exacte ne correspondent qu'à la chaîne exacte donnée pour les champs **hosts** ou **notHosts**.

Votre cluster est affecté si vous avez des ressources **AuthorizationPolicy** utilisant la comparaison de chaînes exactes pour la règle de détermination des hosts ou notHosts.

Vous devez mettre à jour les règles de votre politique d'autorisation pour utiliser la correspondance des préfixes au lieu de la correspondance exacte. Par exemple, en remplaçant **hosts: ["httpbin.com"]** par **hosts: ["httpbin.com:*"]** dans le premier exemple **AuthorizationPolicy**.

Premier exemple de politique d'autorisation utilisant la correspondance des préfixes

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: httpbin
  namespace: foo
spec:
```

```
  action: DENY
  rules:
  - from:
    - source:
        namespaces: ["dev"]
    to:
    - operation:
        hosts: [“httpbin.com”,"httpbin.com:*"]
```

Deuxième exemple de politique d'autorisation utilisant la correspondance des préfixes

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

1.2.2.28. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0.7

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

1.2.2.29. Red Hat OpenShift Service Mesh sur Red Hat OpenShift Dedicated et Microsoft Azure Red Hat OpenShift

Red Hat OpenShift Service Mesh est désormais pris en charge par Red Hat OpenShift Dedicated et Microsoft Azure Red Hat OpenShift.

1.2.2.30. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0.6

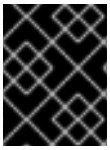
Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

1.2.2.31. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0.5

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

1.2.2.32. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0.4

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.



IMPORTANT

Des étapes manuelles doivent être effectuées pour résoudre les problèmes CVE-2021-29492 et CVE-2021-31920.

1.2.2.32.1. Mises à jour manuelles requises par CVE-2021-29492 et CVE-2021-31920

Istio contient une vulnérabilité exploitable à distance où un chemin de requête HTTP avec plusieurs barres obliques ou des caractères obliques échappés (/ ou \) pourrait potentiellement contourner une politique d'autorisation Istio lorsque des règles d'autorisation basées sur le chemin sont utilisées.

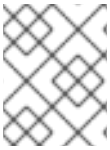
Par exemple, supposons qu'un administrateur de cluster Istio définisse une politique d'autorisation DENY pour rejeter la demande au chemin **/admin**. Une demande envoyée au chemin URL **//admin** ne sera PAS rejetée par la politique d'autorisation.

Selon la RFC 3986, le chemin **//admin** avec plusieurs barres obliques devrait techniquement être traité comme un chemin différent du chemin **/admin**. Cependant, certains services backend choisissent de normaliser les chemins URL en fusionnant les barres obliques multiples en une seule barre oblique. Cela peut entraîner un contournement de la politique d'autorisation (**//admin** ne correspond pas à **/admin**), et un utilisateur peut accéder à la ressource au chemin **/admin** dans le backend ; cela représenterait un incident de sécurité.

Votre cluster est concerné par cette vulnérabilité si vous avez des politiques d'autorisation qui utilisent les modèles **ALLOW action notPaths** field ou **DENY action paths field**. Ces modèles sont vulnérables à des contournements inattendus de la politique.

Votre cluster n'est PAS affecté par cette vulnérabilité si :

- Vous n'avez pas de politique d'autorisation.
- Vos politiques d'autorisation ne définissent pas les champs **paths** ou **notPaths**.
- Vos politiques d'autorisation utilisent des modèles de champ **ALLOW action paths** ou **DENY action notPaths**. Ces motifs pourraient uniquement entraîner un rejet inattendu au lieu d'un contournement de la politique. La mise à niveau est facultative dans ces cas.



NOTE

L'emplacement de configuration de Red Hat OpenShift Service Mesh pour la normalisation des chemins est différent de la configuration d'Istio.

1.2.2.32.2. Mise à jour de la configuration de la normalisation des chemins

Les politiques d'autorisation d'Istio peuvent être basées sur les chemins d'URL dans la requête HTTP. La normalisation des chemins, également connue sous le nom de normalisation des URI, modifie et normalise les chemins des requêtes entrantes afin que les chemins normalisés puissent être traités de manière standard. Des chemins syntaxiquement différents peuvent être équivalents après la normalisation du chemin.

Istio prend en charge les schémas de normalisation suivants sur les chemins de requête avant de les évaluer par rapport aux politiques d'autorisation et d'acheminer les requêtes :

Tableau 1.1. Schémas de normalisation

Option	Description	Exemple :	Notes

Option	Description	Exemple :	Notes
NONE	Aucune normalisation n'est effectuée. Tout ce qui est reçu par Envoy est transmis tel quel à un service de backend.	**..//a../b** est évaluée par les politiques d'autorisation et envoyée à votre service.	Ce paramètre est vulnérable à la CVE-2021-31920.
BASE	C'est actuellement l'option utilisée dans l'installation d'Istio sur **default**. Elle applique l'option **normalize_path** sur les proxies Envoy, qui suit la RFC 3986 avec une normalisation supplémentaire pour convertir les barres obliques inverses en barres obliques inverses.	**/a/../b** est normalisé à **/b**. **da** est normalisé à **/da**.	Ce paramètre est vulnérable à la CVE-2021-31920.
MERGE_SLASHES	Les barres obliques sont fusionnées après la normalisation *BASE*.	**/a//b** est normalisé à **/a/b**.	La mise à jour de ce paramètre permet d'atténuer la CVE-2021-31920.
DECODE_AND_MERGE_SLASHES	Le paramètre le plus strict lorsque vous autorisez tout le trafic par défaut. Ce paramètre est recommandé, à condition que vous testiez minutieusement les itinéraires de vos politiques d'autorisation. Les caractères barre oblique et barre oblique inverse codés en pourcentage (/, /, \ et \) sont décodés en / ou \, avant la normalisation **MERGE_SLASHES**.	**/a/b** est normalisé à **/a/b**.	Mettez à jour ce paramètre pour atténuer la CVE-2021-31920. Ce paramètre est plus sûr, mais il est également susceptible d'endommager les applications. Testez vos applications avant de les déployer en production.

Les algorithmes de normalisation sont exécutés dans l'ordre suivant :

1. Décodez en pourcentage /, /, \ et \.
2. La normalisation RFC 3986 et d'autres normalisations implémentées par l'option **normalize_path** dans Envoy.
3. Fusionner les barres obliques.

AVERTISSEMENT

Bien que ces options de normalisation représentent des recommandations issues des normes HTTP et des pratiques industrielles courantes, les applications peuvent interpréter une URL comme elles l'entendent. Lorsque vous utilisez des politiques de déni, assurez-vous de bien comprendre le comportement de votre application.

1.2.2.32.3. Exemples de configuration de la normalisation des chemins

S'assurer qu'Envoy normalise les chemins d'accès aux requêtes pour qu'ils correspondent aux attentes de vos services d'arrière-plan est essentiel pour la sécurité de votre système. Les exemples suivants peuvent vous servir de référence pour configurer votre système. Les chemins d'URL normalisés, ou les chemins d'URL originaux si **NONE** est sélectionné, seront les suivants :

1. Utilisé pour vérifier les politiques d'autorisation.
2. Transmis à l'application dorsale.

Tableau 1.2. Exemples de configuration

Si votre candidature..	Choisissez..
S'appuie sur le proxy pour effectuer la normalisation	**BASE**, **MERGE_SLASHES** ou **DECODE_AND_MERGE_SLASHES**
Normalise les chemins d'accès aux requêtes sur la base de la RFC 3986 et ne fusionne pas les barres obliques.	**BASE**
Normalise les chemins d'accès aux requêtes sur la base de la RFC 3986 et fusionne les barres obliques, mais ne décode pas les barres obliques codées en pourcentage.	**MERGE_SLASHES**
Normalise les chemins de requête sur la base de la RFC 3986, décode les barres obliquescodées en pourcentage et fusionne les barres obliques.	**DECODE_AND_MERGE_SLASHES**
Traite les chemins de requête d'une manière incompatible avec la RFC 3986.	**NONE**

1.2.2.32.4. Configuration de votre SMCP pour la normalisation des chemins d'accès

Pour configurer la normalisation des chemins pour Red Hat OpenShift Service Mesh, spécifiez ce qui suit dans votre **ServiceMeshControlPlane**. Utilisez les exemples de configuration pour vous aider à déterminer les paramètres de votre système.

SMCP v2 pathNormalisation

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

1.2.2.32.5. Configuration pour la normalisation des cas

Dans certains environnements, il peut être utile de comparer les chemins d'accès dans les politiques d'autorisation sans tenir compte de la casse. Par exemple, traiter **https://myurl/get** et **https://myurl/GeT** comme équivalents. Dans ce cas, vous pouvez utiliser le filtre **EnvoyFilter** présenté ci-dessous. Ce filtre modifie à la fois le chemin utilisé pour la comparaison et le chemin présenté à l'application. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

Enregistrez le site **EnvoyFilter** dans un fichier et exécutez la commande suivante :

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

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: ingress-case-insensitive
  namespace: istio-system
spec:
  configPatches:
  - applyTo: HTTP_FILTER
    match:
      context: GATEWAY
      listener:
        filterChain:
          filter:
            name: "envoy.filters.network.http_connection_manager"
            subFilter:
              name: "envoy.filters.http.router"
    patch:
      operation: INSERT_BEFORE
      value:
        name: envoy.lua
        typed_config:
          "@type": "type.googleapis.com/envoy.extensions.filters.http.lua.v3.Lua"
          inlineCode: |
            function envoy_on_request(request_handle)
              local path = request_handle:headers():get(":path")
              request_handle:headers():replace(":path", string.lower(path))
            end
```

1.2.2.33. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0.3

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

En outre, cette version comporte les nouvelles fonctionnalités suivantes :

- Ajout d'une option à l'outil de collecte de données **must-gather** qui recueille des informations à partir d'un espace de noms de plan de contrôle Service Mesh spécifié. Pour plus d'informations, voir OSSM-351.
- Amélioration des performances pour les plans de contrôle Service Mesh comportant des centaines d'espaces de noms

1.2.2.34. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0.2

Cette version de Red Hat OpenShift Service Mesh ajoute la prise en charge des systèmes IBM Z et IBM Power. Elle aborde également les vulnérabilités et expositions communes (CVE) et les corrections de bogues.

1.2.2.35. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0.1

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

1.2.2.36. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 2.0

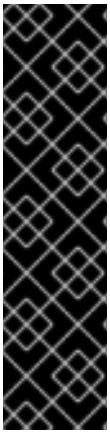
Cette version de Red Hat OpenShift Service Mesh ajoute la prise en charge de Istio 1.6.5, Jaeger 1.20.0, Kiali 1.24.2, et 3scale Istio Adapter 2.0 et OpenShift Container Platform 4.6.

En outre, cette version comporte les nouvelles fonctionnalités suivantes :

- Simplifie l'installation, les mises à jour et la gestion du plan de contrôle Service Mesh.
- Réduit l'utilisation des ressources et le temps de démarrage du plan de contrôle du Service Mesh.
- Améliore les performances en réduisant la communication entre les plans de contrôle sur le réseau.
    - Ajout de la prise en charge du Secret Discovery Service (SDS) d'Envoy. Le SDS est un mécanisme plus sûr et plus efficace pour délivrer des secrets aux mandataires Envoy side car.
- Supprime la nécessité d'utiliser les Secrets Kubernetes, qui présentent des risques de sécurité bien connus.
- Amélioration des performances lors de la rotation des certificats, les proxys n'ayant plus besoin d'être redémarrés pour reconnaître les nouveaux certificats.
    - Ajout de la prise en charge de l'architecture Telemetry v2 d'Istio, qui est construite à l'aide d'extensions WebAssembly. Cette nouvelle architecture apporte des améliorations significatives en termes de performances.
    - Mise à jour de la ressource ServiceMeshControlPlane en v2 avec une configuration simplifiée pour faciliter la gestion du Service Mesh Control Plane.
    - Introduction des extensions WebAssembly en tant que fonctionnalité de l'aperçu technologique.

1.2.3. Avant-première technologique

Certaines fonctionnalités de cette version sont actuellement en avant-première technologique. Ces fonctionnalités expérimentales ne sont pas destinées à être utilisées en production.



IMPORTANT

Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas de les utiliser en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, ce qui permet aux clients de tester les fonctionnalités et de fournir un retour d'information pendant le processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

1.2.4. Fonctionnalités obsolètes et supprimées

Certaines fonctionnalités disponibles dans les versions précédentes sont devenues obsolètes ou ont été supprimées.

Les fonctionnalités dépréciées sont toujours incluses dans OpenShift Container Platform et continuent d'être prises en charge ; cependant, elles seront supprimées dans une prochaine version de ce produit et ne sont pas recommandées pour les nouveaux déploiements.

La fonctionnalité supprimée n'existe plus dans le produit.

1.2.4.1. Fonctionnalités obsolètes et supprimées Red Hat OpenShift Service Mesh 2.3

La prise en charge des suites de chiffrement suivantes a été supprimée. Dans une prochaine version, elles seront supprimées de la liste par défaut des algorithmes de chiffrement utilisés dans les négociations TLS, tant du côté du client que du côté du serveur.

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

L'API **ServiceMeshExtension**, qui était obsolète dans Red Hat OpenShift Service Mesh version 2.2, a été supprimée dans Red Hat OpenShift Service Mesh version 2.3. Si vous utilisez l'API **ServiceMeshExtension**, vous devez migrer vers l'API **WasmPlugin** pour continuer à utiliser vos extensions WebAssembly.

1.2.4.2. Fonctionnalités obsolètes Red Hat OpenShift Service Mesh 2.2

L'API **ServiceMeshExtension** est obsolète à partir de la version 2.2 et sera supprimée dans une prochaine version. Bien que l'API **ServiceMeshExtension** soit toujours prise en charge dans la version 2.2, les clients doivent commencer à passer à la nouvelle API **WasmPlugin**.

1.2.4.3. Fonctionnalités supprimées Red Hat OpenShift Service Mesh 2.2

Cette version marque la fin de la prise en charge des plans de contrôle Service Mesh basés sur Service Mesh 1.1 pour toutes les plateformes.

1.2.4.4. Fonctionnalités supprimées Red Hat OpenShift Service Mesh 2.1

Dans Service Mesh 2.1, le composant Mixer est supprimé. Les corrections de bugs et l'assistance sont assurées jusqu'à la fin du cycle de vie de Service Mesh 2.0.

La mise à jour d'une version 2.0.x de Service Mesh vers la version 2.1 ne se fera pas si les plugins Mixer sont activés. Les plugins Mixer doivent être portés vers WebAssembly Extensions.

1.2.4.5. Fonctionnalités obsolètes Red Hat OpenShift Service Mesh 2.0

Le composant Mixer a été déprécié dans la version 2.0 et sera supprimé dans la version 2.1. Bien que l'utilisation de Mixer pour l'implémentation d'extensions soit toujours possible dans la version 2.0, les extensions auraient dû être migrées vers le nouveau mécanisme WebAssembly.

Les types de ressources suivants ne sont plus pris en charge dans Red Hat OpenShift Service Mesh 2.0 :

- **Policy** (authentication.istio.io/v1alpha1) n'est plus pris en charge. En fonction de la configuration spécifique de votre ressource Policy, vous devrez peut-être configurer plusieurs ressources pour obtenir le même effet.
  - Utiliser **RequestAuthentication** (security.istio.io/v1beta1)
  - Utiliser **PeerAuthentication** (security.istio.io/v1beta1)
- **ServiceMeshPolicy** (maistra.io/v1) n'est plus supporté.
  - Utilisez **RequestAuthentication** ou **PeerAuthentication**, comme indiqué ci-dessus, mais placez-les dans l'espace de noms du plan de contrôle Service Mesh.
- **RbacConfig** (rbac.istio.io/v1alpha1) n'est plus supporté.
  - Remplacé par **AuthorizationPolicy** (security.istio.io/v1beta1), qui englobe les comportements de **RbacConfig**, **ServiceRole**, et **ServiceRoleBinding**.
- **ServiceMeshRbacConfig** (maistra.io/v1) n'est plus supporté.
  - Utiliser **AuthorizationPolicy** comme ci-dessus, mais le placer dans l'espace de noms du plan de contrôle Service Mesh.
- **ServiceRole** (rbac.istio.io/v1alpha1) n'est plus supporté.
- **ServiceRoleBinding** (rbac.istio.io/v1alpha1) n'est plus supporté.
- Dans Kiali, les stratégies **login** et **LDAP** sont obsolètes. Une prochaine version introduira l'authentification à l'aide des fournisseurs OpenID.

1.2.5. Problèmes connus

Ces limitations existent dans Red Hat OpenShift Service Mesh :

- Red Hat OpenShift Service Mesh ne prend pas encore en charge IPv6, car il n'est pas encore entièrement pris en charge par le projet Istio en amont. Par conséquent, Red Hat OpenShift Service Mesh ne prend pas en charge les clusters à double pile.

- Disposition du graphe - La disposition du graphe Kiali peut être différente, selon l'architecture de votre application et les données à afficher (nombre de nœuds du graphe et leurs interactions). Parce qu'il est difficile, voire impossible, de créer une présentation unique qui rende bien dans toutes les situations, Kiali offre un choix de plusieurs présentations différentes. Pour choisir une autre présentation, vous pouvez choisir une autre **Layout Schema** dans le menu **Graph Settings**.

- La première fois que vous accédez à des services connexes tels que la plateforme de traçage distribuée et Grafana, à partir de la console Kiali, vous devez accepter le certificat et vous authentifier à nouveau en utilisant vos identifiants de connexion à OpenShift Container Platform. Cela se produit en raison d'un problème avec la façon dont le framework affiche les pages intégrées dans la console.

- L'application d'exemple Bookinfo ne peut pas être installée sur IBM Z et IBM Power.

- Les extensions WebAssembly ne sont pas prises en charge sur IBM Z et IBM Power.

- LuaJIT n'est pas pris en charge sur IBM Power.

1.2.5.1. Problèmes connus du Service Mesh

Il s'agit des problèmes connus dans Red Hat OpenShift Service Mesh :

- OSSM-2221 L'injection de passerelle ne fonctionne pas dans l'espace de noms du plan de contrôle. Si vous utilisez la fonctionnalité d'injection de passerelle pour créer une passerelle au même endroit que le plan de contrôle, l'injection échoue et OpenShift génère ce message :
  Warning Failed 10s kubelet, ocp-wide-vh8fd-worker-vhqm9 Failed to pull image "auto": rpc error: code = Unknown desc = reading manifest latest in docker.io/library/auto: errors

  Pour créer une passerelle dans l'espace de noms du plan de contrôle, utilisez le paramètre **gateways** dans la spécification SMCP afin de configurer les passerelles d'entrée et de sortie pour le maillage.

- OSSM-2042 Le déploiement du SMCP nommé **default** échoue. Si vous créez un objet SMCP et que vous définissez son champ de version sur v2.3, le nom de l'objet ne peut pas être **default**. Si le nom est **default**, alors le plan de contrôle échoue à se déployer, et OpenShift génère un événement **Warning** avec le message suivant :
  Error processing component mesh-config: error: [mesh-config/templates/telemetryv2_1.6.yaml: Internal error occurred: failed calling webhook "rev.validation.istio.io": Post "https://istiod-default.istio-system.svc:443/validate?timeout=10s": x509: certificate is valid for istiod.istio-system.svc, istiod-remote.istio-system.svc, istio-pilot.istio-system.svc, not istiod-default.istio-system.svc, mesh-config/templates/enable-mesh-permissive.yaml

- OSSM-1655 Le tableau de bord de Kiali affiche une erreur après l'activation de mTLS dans **SMCP**.
  Après avoir activé le paramètre **spec.security.controlPlane.mtls** dans le SMCP, la console Kiali affiche le message d'erreur suivant : **No subsets defined**.

- OSSM-1505 Ce problème se produit uniquement lors de l'utilisation de la ressource

ServiceMeshExtension sur OpenShift Container Platform 4.11. Lorsque vous utilisez **ServiceMeshExtension** sur OpenShift Container Platform 4.11, la ressource n'est jamais prête. Si vous inspectez le problème en utilisant **oc describe ServiceMeshExtension**, vous verrez l'erreur suivante : **stderr: Error creating mount namespace before pivot: function not implemented**.
Solution : **ServiceMeshExtension** a été supprimé dans Service Mesh 2.2. Migrez de **ServiceMeshExtension** vers la ressource **WasmPlugin**. Pour plus d'informations, voir Migration des ressources **ServiceMeshExtension** vers **WasmPlugin**.

- OSSM-1396 Si une ressource passerelle contient le paramètre **spec.externalIPs**, au lieu d'être recréée lors de la mise à jour de **ServiceMeshControlPlane**, la passerelle est supprimée et n'est jamais recréée.

- OSSM-1168 Lorsque les ressources de maillage de services sont créées en tant que fichier YAML unique, le sidecar de proxy Envoy n'est pas injecté de manière fiable dans les pods. Lorsque les ressources SMCP, SMMR et Deployment sont créées individuellement, le déploiement fonctionne comme prévu.

- OSSM-1115 Le champ **concurrency** de l'API **spec.proxy** ne s'est pas propagé à l'istio-proxy. Le champ **concurrency** fonctionne lorsqu'il est défini avec **ProxyConfig**. Le champ **concurrency** spécifie le nombre de threads de travailleur à exécuter. Si le champ est défini sur **0**, le nombre de threads de travailleur disponibles est égal au nombre de cœurs de l'unité centrale. Si le champ n'est pas défini, le nombre de threads travailleurs disponibles est par défaut de **2**.
Dans l'exemple suivant, le champ **concurrency** est défini comme **0**.

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

- OSSM-1052 Lors de la configuration d'un service **ExternalIP** pour l'ingressgateway dans le plan de contrôle Service Mesh, le service n'est pas créé. Le schéma du SMCP ne contient pas le paramètre du service.
Solution : Désactiver la création de la passerelle dans la spécification SMCP et gérer le déploiement de la passerelle entièrement manuellement (y compris le service, le rôle et le RoleBinding).

- OSSM-882 Ceci s'applique à Service Mesh 2.1 et aux versions antérieures. L'espace de noms est dans la liste accessible_namespace mais n'apparaît pas dans l'interface utilisateur de Kiali. Par défaut, Kiali n'affiche pas les espaces de noms commençant par "kube" car ces espaces de noms sont généralement à usage interne et ne font pas partie d'un maillage.
Par exemple, si vous créez un espace de noms appelé "akube-a" et que vous l'ajoutez à la liste des membres du Service Mesh, l'interface utilisateur de Kiali n'affiche pas l'espace de noms. Pour les motifs d'exclusion définis, le logiciel exclut les espaces de noms qui commencent par le motif ou qui le contiennent.

  Solution : Modifiez le paramètre de la ressource personnalisée de Kiali de manière à ce que le paramètre soit précédé d'un carat (^). Par exemple :

```
api:
  namespaces:
    exclude:
```

```
    - "^istio-operator"
    - "^kube-.*"
    - "^openshift.*"
    - "^ibm.*"
    - "^kiali-operator"
```

- MAISTRA-2692 Avec la suppression de Mixer, les métriques personnalisées qui ont été définies dans Service Mesh 2.0.x ne peuvent pas être utilisées dans 2.1. Les métriques personnalisées peuvent être configurées en utilisant **EnvoyFilter**. Red Hat n'est pas en mesure de prendre en charge la configuration de **EnvoyFilter**, sauf lorsqu'elle est explicitement documentée. Ceci est dû à un couplage étroit avec les API sous-jacentes d'Envoy, ce qui signifie que la compatibilité ascendante ne peut pas être maintenue.

- MAISTRA-2648 **ServiceMeshExtensions** n'est actuellement pas compatible avec les mailles déployées sur les systèmes IBM Z.

- MAISTRA-1959 *Migration to 2.0* Le grattage Prometheus (**spec.addons.prometheus.scrape** défini sur **true**) ne fonctionne pas lorsque mTLS est activé. De plus, Kiali affiche des données graphiques superflues lorsque mTLS est désactivé.
  Ce problème peut être résolu en excluant le port 15020 de la configuration du proxy, par exemple,

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

- MAISTRA-1314 Red Hat OpenShift Service Mesh ne prend pas encore en charge IPv6.

- MAISTRA-453 Si vous créez un nouveau projet et déployez des modules immédiatement, l'injection de sidecar ne se produit pas. L'opérateur ne parvient pas à ajouter le site **maistra.io/member-of** avant la création des nacelles. Les nacelles doivent donc être supprimées et recréées pour que l'injection de sidecar se produise.

- MAISTRA-158 L'application de plusieurs passerelles référençant le même nom d'hôte entraîne l'arrêt du fonctionnement de toutes les passerelles.

1.2.5.2. Problèmes connus de Kiali

NOTE

Les nouveaux problèmes pour Kiali doivent être créés dans le projet OpenShift Service Mesh avec l'adresse **Component** fixée à **Kiali**.

Voici les problèmes connus de Kiali :

- KIALI-2206 Lorsque vous accédez à la console Kiali pour la première fois, et qu'il n'y a pas de données de navigation mises en cache pour Kiali, le lien "View in Grafana" sur l'onglet Metrics de la page Kiali Service Details redirige vers le mauvais emplacement. La seule façon de rencontrer ce problème est d'accéder à Kiali pour la première fois.

- KIALI-507 Kiali ne supporte pas Internet Explorer 11. Ceci est dû au fait que les frameworks sous-jacents ne supportent pas Internet Explorer. Pour accéder à la console Kiali, utilisez l'une des deux versions les plus récentes des navigateurs Chrome, Edge, Firefox ou Safari.

1.2.5.3. Red Hat OpenShift distributed tracing known issues (problèmes connus)

Ces limitations existent dans le traçage distribué de Red Hat OpenShift :

- Apache Spark n'est pas pris en charge.
- Le déploiement en continu via AMQ/Kafka n'est pas pris en charge sur les systèmes IBM Z et IBM Power.

Ce sont les problèmes connus pour Red Hat OpenShift distributed tracing :

- OBSDA-220 Dans certains cas, si vous essayez d'extraire une image à l'aide de la collecte de données de traçage distribuées, l'extraction de l'image échoue et un message d'erreur **Failed to pull image** s'affiche. Il n'y a pas de solution à ce problème.
- TRACING-2057 L'API Kafka a été mise à jour sur **v1beta2** pour prendre en charge le Strimzi Kafka Operator 0.23.0. Cependant, cette version de l'API n'est pas prise en charge par AMQ Streams 1.6.3. Si vous avez l'environnement suivant, vos services Jaeger ne seront pas mis à niveau et vous ne pourrez pas créer de nouveaux services Jaeger ou modifier des services Jaeger existants :
  - Canal de l'opérateur Jaeger : **1.17.x stable** ou **1.20.x stable**
  - Canal de l'opérateur AMQ Streams : **amq-streams-1.6.x**
    Pour résoudre ce problème, changez le canal d'abonnement de votre opérateur AMQ Streams sur **amq-streams-1.7.x** ou **stable**.

1.2.6. Problèmes corrigés

Les problèmes suivants ont été résolus dans la version actuelle :

1.2.6.1. Le maillage de service a permis de résoudre des problèmes

- OSSM-3644 Auparavant, la passerelle egress de la fédération recevait la mauvaise mise à jour des points d'extrémité de la passerelle réseau, ce qui entraînait des entrées de points d'extrémité supplémentaires. Maintenant, la passerelle federation-egress a été mise à jour du côté du serveur afin qu'elle reçoive les bons points de terminaison de la passerelle réseau.
- OSSM-3595 Auparavant, le plugin **istio-cni** échouait parfois sur RHEL parce que SELinux n'autorisait pas l'utilitaire **iptables-restore** à ouvrir des fichiers dans le répertoire **/tmp**. Désormais, SELinux transmet **iptables-restore** via le flux d'entrée **stdin** au lieu d'un fichier.
- OSSM-3025 Istiod ne parvient parfois pas à se préparer. Parfois, lorsqu'un maillage contenait de nombreux espaces de noms membres, le module Istiod n'était pas prêt en raison d'un blocage au sein d'Istiod. Le blocage est maintenant résolu et le pod démarre comme prévu.
- OSSM-2493 Les **nodeSelector** et **tolerations** par défaut dans SMCP ne sont pas transmis à Kiali. Les **nodeSelector** et **tolerations** que vous ajoutez à **SMCP.spec.runtime.defaults** sont maintenant transmis à la ressource Kiali.

- OSSM-2492 Les tolérances par défaut dans le SMCP ne sont pas transmises à Jaeger. Les **nodeSelector** et **tolerations** que vous ajoutez à **SMCP.spec.runtime.defaults** sont maintenant transmis à la ressource Jaeger.

- OSSM-2374 Si vous avez supprimé l'une des ressources **ServiceMeshMember**, l'opérateur du Service Mesh a supprimé la ressource **ServiceMeshMemberRoll**. Bien que ce comportement soit attendu lorsque vous supprimez la dernière ressource **ServiceMeshMember**, l'opérateur ne doit pas supprimer la ressource **ServiceMeshMemberRoll** si elle contient des membres en plus de celui qui a été supprimé. Ce problème est maintenant résolu et l'opérateur ne supprime le ServiceMeshMemberRoll que lorsque la dernière ressource **ServiceMeshMember** est supprimée.

- OSSM-2373 Error trying to get OAuth metadata when logging in. Pour récupérer la version du cluster, le compte **system:anonymous** est utilisé. Avec les ClusterRoles et ClusterRoleBinding intégrés par défaut au cluster, le compte anonyme peut récupérer la version correctement. Si le compte **system:anonymous** perd ses privilèges pour récupérer la version du cluster, l'authentification OpenShift devient inutilisable.
  Ce problème est résolu par l'utilisation de Kiali SA pour récupérer la version du cluster. Cela permet également d'améliorer la sécurité sur le cluster.

- OSSM-2371 Bien que Kiali soit configuré en "vue seule", un utilisateur peut modifier le niveau de journalisation du proxy via le menu kebab de l'onglet Logs des détails de la charge de travail. Ce problème a été corrigé afin que les options sous "Set Proxy Log Level" soient désactivées lorsque Kiali est configuré en "view-only"

- OSSM-2344 Le redémarrage d'Istiod amène Kiali à inonder CRI-O de demandes de transfert de port. Ce problème se produisait lorsque Kiali ne pouvait pas se connecter à Istiod et que Kiali envoyait simultanément un grand nombre de requêtes à istiod. Kiali limite désormais le nombre de requêtes qu'il envoie à istiod.

- OSSM-2335 En faisant glisser le pointeur de la souris sur le diagramme de dispersion Traces, la console Kiali cessait parfois de répondre en raison des demandes simultanées du backend.

- OSSM-2053 En utilisant Red Hat OpenShift Service Mesh Operator 2.2 ou 2.3, pendant le rapprochement SMCP, le contrôleur SMMR a supprimé les espaces de noms membres de **SMMR.status.configuredMembers**. Cela a entraîné l'indisponibilité des services dans les espaces de noms membres pendant quelques instants.
  En utilisant Red Hat OpenShift Service Mesh Operator 2.2 ou 2.3, le contrôleur SMMR ne supprime plus les espaces de noms de **SMMR.status.configuredMembers**. Au lieu de cela, le contrôleur ajoute les espaces de noms à **SMMR.status.pendingMembers** pour indiquer qu'ils ne sont pas à jour. Pendant la réconciliation, lorsque chaque espace de noms se synchronise avec le SMCP, l'espace de noms est automatiquement supprimé de **SMMR.status.pendingMembers**.

- OSSM-1962 Utiliser **EndpointSlices** dans le contrôleur de fédération. Le contrôleur de fédération utilise désormais **EndpointSlices**, ce qui améliore l'évolutivité et les performances dans les grands déploiements. L'indicateur PILOT_USE_ENDPOINT_SLICE est activé par défaut. La désactivation de cet indicateur empêche l'utilisation des déploiements de fédération.

- OSSM-1668 Un nouveau champ **spec.security.jwksResolverCA** a été ajouté à la version 2.1 **SMCP** mais était absent des versions 2.2.0 et 2.2.1. Lors de la mise à niveau d'une version de l'opérateur où ce champ était présent vers une version de l'opérateur où ce champ manquait, le champ **.spec.security.jwksResolverCA** n'était pas disponible dans la base de données **SMCP**.

- OSSM-1325 Le pod istiod se bloque et affiche le message d'erreur suivant : **fatal error: concurrent map iteration and map write**.

- OSSM-1211 La configuration des mailles de services fédérés pour le basculement ne fonctionne pas comme prévu.
  Le journal du pilote Istiod affiche l'erreur suivante : **envoy connection [C289] TLS error: 337047686:SSL routines:tls_process_server_certificate:certificate verify failed**

- OSSM-1099 La console Kiali affiche le message suivant **Sorry, there was a problem. Try a refresh or navigate to a different page.**

- OSSM-1074 Les annotations de pods définies dans SMCP ne sont pas injectées dans les pods.

- OSSM-999 Kiali retention n'a pas fonctionné comme prévu. Les heures du calendrier étaient grisées dans le graphique du tableau de bord.

- OSSM-797 Kiali Operator pod génère **CreateContainerConfigError** lors de l'installation ou de la mise à jour de l'opérateur.

- OSSM-722 Les espaces de noms commençant par **kube** sont cachés à Kiali.

- OSSM-569 Il n'y a pas de limite de mémoire CPU pour le conteneur Prometheus **istio-proxy**. Le sidecar Prometheus **istio-proxy** utilise désormais les limites de ressources définies dans **spec.proxy.runtime.container**.

- OSSM-535 Prise en charge des validationMessages dans SMCP. Le champ **ValidationMessages** dans le plan de contrôle de la maille de service peut maintenant être défini sur **True**. Cela écrit un journal pour l'état des ressources, ce qui peut être utile lors de la résolution des problèmes.

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

- OSSM-419 Les espaces de noms avec des noms similaires apparaîtront tous dans la liste des espaces de noms de Kiali, même si les espaces de noms ne sont pas définis dans le rôle de membre du maillage de services.

- OSSM-296 Lors de l'ajout d'une configuration de santé à la ressource personnalisée (CR) de Kiali, celle-ci n'est pas répliquée dans la carte de configuration de Kiali.

- OSSM-291 Dans la console Kiali, sur les pages Applications, Services et Workloads, la fonction "Remove Label from Filters" ne fonctionne pas.

- OSSM-289 Dans la console Kiali, sur les pages de détails des services "istio-ingressgateway" et "jaeger-query", aucune trace n'est affichée. Les traces existent dans Jaeger.

- OSSM-287 Dans la console Kiali, aucune trace n'est affichée sur le service graphique.

- OSSM-285 En essayant d'accéder à la console Kiali, on reçoit le message d'erreur suivant : "Error trying to get OAuth Metadata".
  Solution : Redémarrer le pod Kiali.

- MAISTRA-2735 Les ressources que l'opérateur de Service Mesh supprime lors de la réconciliation du SMCP ont changé dans Red Hat OpenShift Service Mesh version 2.1. Auparavant, l'Opérateur supprimait une ressource avec les étiquettes suivantes :

  - **maistra.io/owner**

  - **app.kubernetes.io/version**

Désormais, l'opérateur ignore les ressources qui n'incluent pas le label **app.kubernetes.io/managed-by=maistra-istio-operator**. Si vous créez vos propres ressources, vous ne devez pas y ajouter le label **app.kubernetes.io/managed-by=maistra-istio-operator**.

- MAISTRA-2687 La passerelle de fédération Red Hat OpenShift Service Mesh 2.1 n'envoie pas la chaîne de certificats complète lors de l'utilisation de certificats externes. La passerelle de sortie de la fédération Service Mesh n'envoie que le certificat du client. Étant donné que la passerelle d'entrée de la fédération ne connaît que le certificat racine, elle ne peut pas vérifier le certificat client à moins que vous n'ajoutiez le certificat racine à l'importation de la fédération **ConfigMap**.

- MAISTRA-2635 Remplacer l'API Kubernetes obsolète. Pour rester compatible avec OpenShift Container Platform 4.8, l'API **apiextensions.k8s.io/v1beta1** est obsolète depuis Red Hat OpenShift Service Mesh 2.0.8.

- MAISTRA-2631 La fonctionnalité WASM ne fonctionne pas car podman échoue en raison de l'absence du binaire nsenter. Red Hat OpenShift Service Mesh génère le message d'erreur suivant : **Error: error configuring CNI network plugin exec: "nsenter": executable file not found in $PATH**. L'image du conteneur contient maintenant nsenter et WASM fonctionne comme prévu.

- MAISTRA-2534 Lorsque istiod tente de récupérer le JWKS pour un émetteur spécifié dans une règle JWT, le service de l'émetteur répond avec un 502. Cela empêchait le conteneur proxy d'être prêt et provoquait l'arrêt des déploiements. La correction du bogue de la communauté a été incluse dans la version 2.0.7 de Service Mesh.

- MAISTRA-2411 Lorsque l'opérateur crée une nouvelle passerelle d'entrée en utilisant **spec.gateways.additionalIngress** dans **ServiceMeshControlPlane**, il ne crée pas de **NetworkPolicy** pour la passerelle d'entrée supplémentaire comme il le fait pour la passerelle d'entrée par défaut. Cela provoque une réponse 503 de la route de la nouvelle passerelle. Solution de contournement : Créez manuellement le site **NetworkPolicy** dans l'espace de noms <istio-system>.

- MAISTRA-2401 CVE-2021-3586 servicemesh-operator : Les ressources NetworkPolicy spécifient incorrectement les ports pour les ressources entrantes. Les ressources NetworkPolicy installées pour Red Hat OpenShift Service Mesh ne spécifiaient pas correctement les ports auxquels il était possible d'accéder. Cela permettait d'accéder à tous les ports de ces ressources à partir de n'importe quel pod. Les politiques de réseau appliquées aux ressources suivantes sont affectées :

  - Cuisine
  - Grafana
  - Istiod
  - Jaeger
  - Kiali
  - Prometheus
  - Injecteur Sidecar

- MAISTRA-2378 Lorsque le cluster est configuré pour utiliser OpenShift SDN avec **ovs-multitenant** et que le maillage contient un grand nombre d'espaces de noms (200), le plugin de mise en réseau d'OpenShift Container Platform est incapable de configurer les espaces de

noms rapidement. Service Mesh s'interrompt, ce qui fait que les espaces de noms sont continuellement supprimés du maillage de services, puis réinscrits.

- MAISTRA-2370 Gestion des pierres tombales dans listerInformer. Le code de base du cache mis à jour ne gérait pas les pierres tombales lors de la traduction des événements des caches d'espace de noms vers le cache agrégé, ce qui entraînait une panique dans la routine go.

- MAISTRA-2117 Ajout du montage optionnel **ConfigMap** à l'opérateur. Le CSV contient maintenant un montage optionnel du volume **ConfigMap**, qui monte le répertoire **smcp-templates ConfigMap** s'il existe. Si **smcp-templates ConfigMap** n'existe pas, le répertoire monté est vide. Lorsque vous créez **ConfigMap**, le répertoire est rempli avec les entrées de **ConfigMap** et peut être référencé dans **SMCP.spec.profiles**. Aucun redémarrage de l'opérateur Service Mesh n'est nécessaire.
  Les clients qui utilisent l'opérateur 2.0 avec un CSV modifié pour monter le ConfigMap smcp-templates peuvent mettre à niveau vers Red Hat OpenShift Service Mesh 2.1. Après la mise à niveau, vous pouvez continuer à utiliser un ConfigMap existant, et les profils qu'il contient, sans modifier le CSV. Les clients qui utilisaient précédemment ConfigMap avec un nom différent devront soit renommer le ConfigMap, soit mettre à jour le CSV après la mise à niveau.

- MAISTRA-2010 AuthorizationPolicy ne prend pas en charge le champ **request.regex.headers**. Le site **validatingwebhook** rejette toute AuthorizationPolicy contenant ce champ, et même si vous le désactivez, Pilot essaie de le valider en utilisant le même code, et cela ne fonctionne pas.

- MAISTRA-1979 *Migration to 2.0* Le webhook de conversion supprime les champs importants suivants lors de la conversion de **SMCP.status** de v2 à v1 :

  - conditions
  - composants
  - observéGénération
  - annotations
    La mise à jour de l'opérateur vers la version 2.0 pourrait interrompre les outils clients qui lisent l'état du SMCP en utilisant la version maistra.io/v1 de la ressource.

    Cela signifie également que les colonnes READY et STATUS sont vides lorsque vous exécutez **oc get servicemeshcontrolplanes.v1.maistra.io**.

- MAISTRA-1947 *Technology Preview* Les mises à jour de ServiceMeshExtensions ne sont pas appliquées.
  Solution de contournement : Supprimez et recréez le site **ServiceMeshExtensions**.

- MAISTRA-1983 *Migration to 2.0* La mise à jour vers la version 2.0.0 avec un site **ServiceMeshControlPlane** invalide ne peut pas être facilement réparée. Les éléments non valides de la ressource **ServiceMeshControlPlane** ont provoqué une erreur irrécupérable. Le correctif permet de récupérer les erreurs. Vous pouvez supprimer la ressource non valide et la remplacer par une nouvelle ou modifier la ressource pour corriger les erreurs. Pour plus d'informations sur l'édition de votre ressource, consultez [Configuration de l'installation de Red Hat OpenShift Service Mesh].

- MAISTRA-1502 Suite à la correction des CVEs dans la version 1.0.10, les tableaux de bord Istio ne sont pas disponibles à partir du menu **Home Dashboard** dans Grafana. Pour accéder aux tableaux de bord Istio, cliquez sur le menu **Dashboard** dans le panneau de navigation et sélectionnez l'onglet **Manage**.

- MAISTRA-1399 Red Hat OpenShift Service Mesh ne vous empêche plus d'installer des protocoles CNI non pris en charge. Les configurations réseau prises en charge n'ont pas changé.

- MAISTRA-1089 *Migration to 2.0* Les passerelles créées dans un espace de noms qui n'est pas un plan de contrôle sont automatiquement supprimées. Après avoir supprimé la définition de la passerelle de la spécification SMCP, vous devez supprimer manuellement ces ressources.

- MAISTRA-858 Les messages de journal Envoy suivants décrivant les options et configurations obsolètes associées à Istio 1.1.x sont attendus :

  - [2019-06-03 07:03:28.943][19][warning][misc] [external/envoy/source/common/protobuf/utility.cc:129] Utilisation de l'option obsolète 'envoy.api.v2.listener.Filter.config'. Cette configuration sera bientôt supprimée d'Envoy.

  - [2019-08-12 22:12:59.001][13][warning][misc] [external/envoy/source/common/protobuf/utility.cc:174] Utilisation de l'option obsolète 'envoy.api.v2.Listener.use_original_dst' du fichier lds.proto. Cette configuration sera bientôt supprimée d'Envoy.

- MAISTRA-806 L'éviction du pod opérateur Istio entraîne le non-déploiement du maillage et de la CNI.
  Solution de contournement : Si le pod **istio-operator** est expulsé lors du déploiement du volet de contrôle, supprimez le pod **istio-operator** expulsé.

- MAISTRA-681 Lorsque le plan de contrôle du Service Mesh comporte de nombreux espaces de noms, il peut en résulter des problèmes de performance.

- MAISTRA-193 Des messages d'information inattendus de la console sont visibles lorsque la vérification de l'état de santé est activée pour citadel.

- Bugzilla 1821432 Les contrôles à bascule dans la page de détails des ressources personnalisées d'OpenShift Container Platform ne mettent pas à jour le CR correctement. Les contrôles UI Toggle dans la page Service Mesh Control Plane (SMCP) Overview de la console web d'OpenShift Container Platform mettent parfois à jour le mauvais champ de la ressource. Pour mettre à jour un SMCP, modifiez le contenu YAML directement ou mettez à jour la ressource à partir de la ligne de commande au lieu de cliquer sur les contrôles à bascule.

1.2.6.2. Red Hat OpenShift a corrigé des problèmes de traçage distribué

- OSSM-1910 En raison d'un problème introduit dans la version 2.6, les connexions TLS ne pouvaient pas être établies avec OpenShift Container Platform Service Mesh. Cette mise à jour résout le problème en changeant les noms des ports de service pour qu'ils correspondent aux conventions utilisées par OpenShift Container Platform Service Mesh et Istio.

- OBSDA-208 Avant cette mise à jour, les limites de ressources par défaut de 200 m de CPU et 256 m de mémoire pouvaient entraîner un redémarrage continu de la collecte des données de traçage distribué sur les grands clusters. Cette mise à jour résout le problème en supprimant ces limites de ressources.

- OBSDA-222 Avant cette mise à jour, des spans pouvaient être abandonnés dans la plateforme de traçage distribuée OpenShift Container Platform. Pour aider à prévenir ce problème, cette version met à jour les dépendances de version.

- TRACING-2337 Jaeger enregistre un message d'avertissement répétitif dans les journaux de Jaeger, similaire à ce qui suit :

  ```
  {"level":"warn","ts":1642438880.918793,"caller":"channelz/logging.go:62","msg":"[core]grpc:
  ```

> Server.Serve failed to create ServerTransport: connection error: desc = \"transport: http2Server.HandleStreams received bogus greeting from client: \\\"\\\\x16\\\\x03\\\\x01\\\\x02\\\\x00\\\\x01\\\\x00\\\\x01\\\\xfc\\\\x03\\\\x03vw\\\\x1a\\\\xc9T\\\\xe7\\\\xdaCj\\\\xb7\\\\x8dK\\\\xa6\\\"\"","system":"grpc","grpc_log":true}

Ce problème a été résolu en exposant uniquement le port HTTP(S) du service de requête, et non le port gRPC.

- TRACING-2009 L'opérateur Jaeger a été mis à jour pour inclure le support de l'opérateur Strimzi Kafka 0.23.0.
- TRACING-1907 L'injection du sidecar de l'agent Jaeger échouait en raison de l'absence de cartes de configuration dans l'espace de noms de l'application. Les cartes de configuration étaient automatiquement supprimées en raison d'un paramètre de champ **OwnerReference** incorrect et, par conséquent, les pods d'application ne dépassaient pas l'étape "ContainerCreating". Les paramètres incorrects ont été supprimés.
- TRACING-1725 Suivi de TRACING-1631. Correction supplémentaire pour s'assurer que les certificats Elasticsearch sont correctement réconciliés lorsqu'il y a plusieurs instances de production Jaeger, utilisant le même nom mais dans des espaces de noms différents. Voir aussi BZ-1918920.
- TRACING-1631 Plusieurs instances de production Jaeger, utilisant le même nom mais dans des espaces de noms différents, causant un problème de certificat Elasticsearch. Lorsque plusieurs maillages de services étaient installés, toutes les instances Elasticsearch de Jaeger avaient le même secret Elasticsearch au lieu de secrets individuels, ce qui empêchait l'opérateur Elasticsearch d'OpenShift de communiquer avec tous les clusters Elasticsearch.
- TRACING-1300 Échec de la connexion entre l'agent et le collecteur lors de l'utilisation d'Istio sidecar. Une mise à jour de l'opérateur Jaeger a activé la communication TLS par défaut entre un agent Jaeger sidecar et le collecteur Jaeger.
- TRACING-1208 Authentification "500 Internal Error" lors de l'accès à l'interface utilisateur de Jaeger. Lorsque j'essaie de m'authentifier à l'interface utilisateur en utilisant OAuth, j'obtiens une erreur 500 parce que le sidecar oauth-proxy ne fait pas confiance à l'ensemble d'autorités de certification personnalisées défini lors de l'installation avec l'adresse **additionalTrustBundle**.
- TRACING-1166 Il n'est actuellement pas possible d'utiliser la stratégie de streaming Jaeger dans un environnement déconnecté. Lorsqu'un cluster Kafka est en cours de provisionnement, il en résulte une erreur : **Failed to pull image registry.redhat.io/amq7/amq-streams-kafka-24-rhel7@sha256:f9ceca004f1b7dccb3b82d9a8027961f9fe4104e0ed69752c0bdd8078b4a1076**.
- TRACING-809 Jaeger Ingester est incompatible avec Kafka 2.3. Lorsqu'il y a deux ou plusieurs instances de Jaeger Ingester et un trafic suffisant, il génère continuellement des messages de rééquilibrage dans les journaux. Ceci est dû à une régression dans Kafka 2.3 qui a été corrigée dans Kafka 2.3.1. Pour plus d'informations, voir Jaegertracing-1819.
- BZ-1918920/LOG-1619 Les pods Elasticsearch ne sont pas redémarrés automatiquement après une mise à jour.
  Solution : Redémarrer les pods manuellement.

1.3. COMPRENDRE LE MAILLAGE DE SERVICES

Red Hat OpenShift Service Mesh fournit une plateforme pour une vision comportementale et un contrôle opérationnel sur vos microservices en réseau dans un maillage de services. Avec Red Hat OpenShift Service Mesh, vous pouvez connecter, sécuriser et surveiller les microservices dans votre

environnement OpenShift Container Platform.

1.3.1. Comprendre le maillage des services

Un *service mesh* est le réseau de microservices qui composent les applications dans une architecture distribuée de microservices et les interactions entre ces microservices. Lorsqu'un Service Mesh augmente en taille et en complexité, il peut devenir plus difficile à comprendre et à gérer.

Basé sur le projet open source Istio, Red Hat OpenShift Service Mesh ajoute une couche transparente sur les applications distribuées existantes sans nécessiter de modifications au code du service. Vous ajoutez la prise en charge de Red Hat OpenShift Service Mesh aux services en déployant un proxy sidecar spécial aux services pertinents dans le maillage qui intercepte toutes les communications réseau entre les microservices. Vous configurez et gérez le Service Mesh en utilisant les fonctionnalités du plan de contrôle du Service Mesh.

Red Hat OpenShift Service Mesh vous offre un moyen simple de créer un réseau de services déployés qui fournissent :

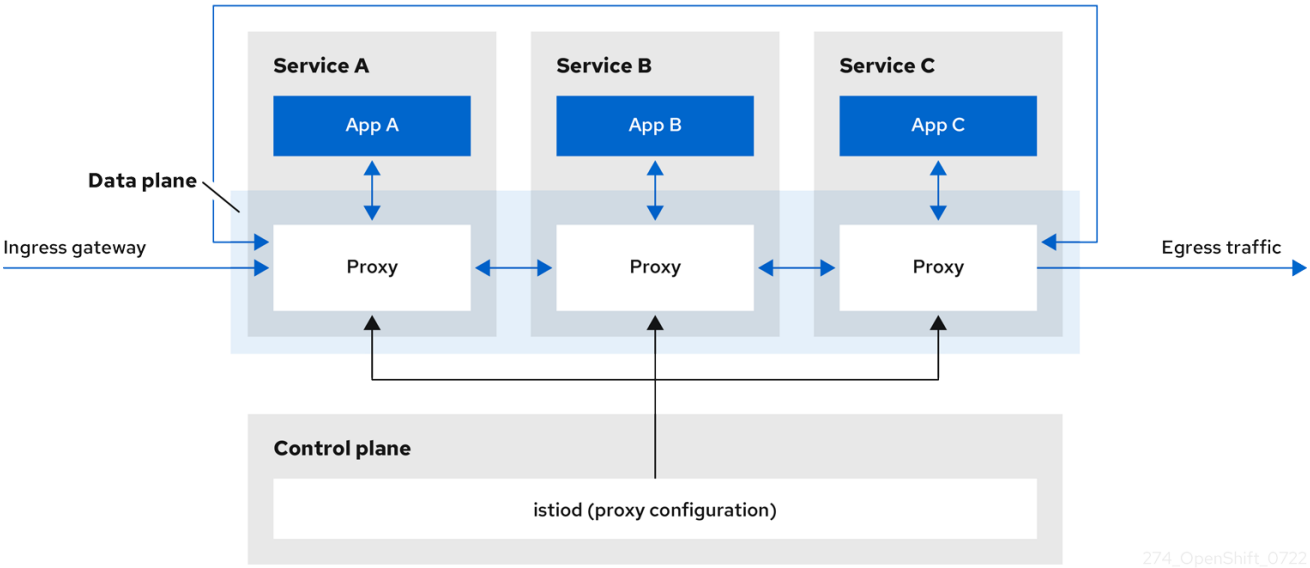
- Découverte
- Équilibrage de la charge
- Authentification de service à service
- Recouvrement des défaillances
- Metrics
- Contrôle

Red Hat OpenShift Service Mesh fournit également des fonctions opérationnelles plus complexes, notamment :

- Tests A/B
- Libération des canaris
- Contrôle d'accès
- Authentification de bout en bout

1.3.2. Architecture Service Mesh

La technologie de maillage de services fonctionne au niveau de la communication réseau. En d'autres termes, les composants du maillage de services capturent ou interceptent le trafic vers et depuis les microservices, modifiant les requêtes, les redirigeant ou créant de nouvelles requêtes vers d'autres services.



À un niveau élevé, Red Hat OpenShift Service Mesh se compose d'un plan de données et d'un plan de contrôle

Le site **data plane** est un ensemble de proxys intelligents, fonctionnant avec des conteneurs d'application dans un pod, qui interceptent et contrôlent toutes les communications réseau entrantes et sortantes entre les microservices dans le maillage de services. Le plan de données est mis en œuvre de manière à intercepter tout le trafic réseau entrant (ingress) et sortant (egress). Le plan de données Istio est composé de conteneurs Envoy fonctionnant aux côtés des conteneurs d'application dans un pod. Le conteneur Envoy agit comme un proxy, contrôlant toutes les communications réseau entrantes et sortantes du pod.

- **Envoy proxies** sont les seuls composants d'Istio qui interagissent avec le trafic du plan de données. Tout le trafic réseau entrant (ingress) et sortant (egress) entre les services passe par les proxys. Le proxy Envoy collecte également toutes les métriques liées au trafic des services au sein du maillage. Les proxys Envoy sont déployés en tant que sidecars, fonctionnant dans le même pod que les services. Les proxys Envoy sont également utilisés pour mettre en œuvre des passerelles maillées.
    - **Sidecar proxies** gérer les communications entrantes et sortantes pour leur instance de charge de travail.
    - **Gateways** sont des proxys fonctionnant comme des équilibreurs de charge recevant des connexions HTTP/TCP entrantes ou sortantes. Les configurations de passerelle sont appliquées aux proxys Envoy autonomes qui fonctionnent à la périphérie du maillage, plutôt qu'aux proxys Envoy sidecar qui fonctionnent aux côtés de vos charges de travail de service. Vous utilisez une passerelle pour gérer le trafic entrant et sortant de votre réseau maillé, en vous permettant de spécifier le trafic que vous souhaitez voir entrer ou sortir du réseau maillé.
        - **Ingress-gateway** - Également connue sous le nom de contrôleur d'entrée, la passerelle d'entrée est un proxy Envoy dédié qui reçoit et contrôle le trafic entrant dans le maillage de services. Une passerelle d'entrée permet d'appliquer des fonctions telles que la surveillance et les règles de routage au trafic entrant dans le cluster.
        - **Egress-gateway** - Également connue sous le nom de contrôleur de sortie, la passerelle de sortie est un proxy Envoy dédié qui gère le trafic quittant le maillage de service. Une passerelle de sortie permet d'appliquer des fonctions telles que la surveillance et les règles de routage au trafic sortant du maillage.

Le site **control plane** gère et configure les serveurs mandataires qui constituent le plan de données. Il est la source autorisée pour la configuration, gère le contrôle d'accès et les politiques d'utilisation, et recueille les mesures des proxies dans le maillage de service.

- Le plan de contrôle d'Istio est composé de **Istiod** qui consolide plusieurs composants précédents du plan de contrôle (Citadel, Galley, Pilot) en un seul binaire. Istiod assure la découverte des services, la configuration et la gestion des certificats. Il convertit les règles de routage de haut niveau en configurations Envoy et les propage aux sidecars au moment de l'exécution.
    - Istiod peut agir en tant qu'autorité de certification (CA), générant des certificats supportant la communication sécurisée mTLS dans le plan de données. Vous pouvez également utiliser une autorité de certification externe à cette fin.
    - Istiod est responsable de l'injection des conteneurs proxy sidecar dans les charges de travail déployées sur un cluster OpenShift.

Red Hat OpenShift Service Mesh utilise le site **istio-operator** pour gérer l'installation du plan de contrôle. Un *Operator* est un logiciel qui vous permet de mettre en œuvre et d'automatiser des activités courantes dans votre cluster OpenShift. Il agit comme un contrôleur, vous permettant de définir ou de modifier l'état souhaité des objets dans votre cluster, dans ce cas, une installation Red Hat OpenShift Service Mesh.

Red Hat OpenShift Service Mesh inclut également les modules complémentaires Istio suivants dans le cadre du produit :

- **Kiali** - Kiali est la console de gestion pour Red Hat OpenShift Service Mesh. Elle fournit des tableaux de bord, une observabilité et de solides capacités de configuration et de validation. Il montre la structure de votre maillage de services en déduisant la topologie du trafic et affiche la santé de votre maillage. Kiali fournit des métriques détaillées, une validation puissante, un accès à Grafana et une forte intégration avec la plateforme de traçage distribuée.
- **Prometheus** - Red Hat OpenShift Service Mesh utilise Prometheus pour stocker les informations de télémétrie des services. Kiali dépend de Prometheus pour obtenir des métriques, l'état de santé et la topologie du maillage.
- **Jaeger** - Red Hat OpenShift Service Mesh prend en charge la plateforme de traçage distribuée. Jaeger est un serveur de traçabilité open source qui centralise et affiche les traces associées à une requête unique entre plusieurs services. En utilisant la plateforme de traçage distribué, vous pouvez surveiller et dépanner vos systèmes distribués basés sur des microservices.
- **Elasticsearch** - Elasticsearch est un moteur de recherche et d'analyse open source, distribué et basé sur JSON. La plateforme de traçage distribuée utilise Elasticsearch pour le stockage permanent.
- **Grafana** - Grafana fournit aux administrateurs de maillage des requêtes avancées, des analyses de métriques et des tableaux de bord pour les données Istio. En option, Grafana peut être utilisé pour analyser les métriques de maillage de services.

Les intégrations Istio suivantes sont prises en charge avec Red Hat OpenShift Service Mesh :

- **3scale** - Istio fournit une intégration optionnelle avec les solutions Red Hat 3scale API Management. Pour les versions antérieures à 2.1, cette intégration était réalisée via l'adaptateur 3scale Istio. Pour les versions 2.1 et ultérieures, l'intégration de 3scale est réalisée via un module WebAssembly.

Pour plus d'informations sur l'installation de l'adaptateur 3scale, reportez-vous à la documentation de l'adaptateur Istio 3scale

1.3.3. Comprendre le Kiali

Kiali fournit une visibilité sur votre maillage de services en vous montrant les microservices dans votre maillage de services, et comment ils sont connectés.

1.3.3.1. Vue d'ensemble de Kiali

Kiali fournit une observabilité dans le Service Mesh fonctionnant sur OpenShift Container Platform. Kiali vous aide à définir, valider et observer votre maillage de services Istio. Il vous aide à comprendre la structure de votre maillage de services en déduisant la topologie, et fournit également des informations sur la santé de votre maillage de services.

Kiali fournit une vue graphique interactive de votre espace de noms en temps réel qui offre une visibilité sur des caractéristiques telles que les disjoncteurs, les taux de requête, la latence et même les graphiques des flux de trafic. Kiali offre des informations sur les composants à différents niveaux, des applications aux services et aux charges de travail, et peut afficher les interactions avec des informations contextuelles et des graphiques sur le nœud ou le bord du graphique sélectionné. Kiali offre également la possibilité de valider vos configurations Istio, telles que les passerelles, les règles de destination, les services virtuels, les politiques de maillage, etc. Kiali fournit des métriques détaillées, et une intégration Grafana de base est disponible pour les requêtes avancées. Le traçage distribué est assuré par l'intégration de Jaeger dans la console Kiali.

Kiali est installé par défaut dans le cadre du Red Hat OpenShift Service Mesh.

1.3.3.2. Architecture Kiali

Kiali est basé sur le projet open source Kiali. Kiali est composé de deux éléments : l'application Kiali et la console Kiali.

- **Kiali application** (back end) - Ce composant s'exécute dans la plateforme d'application conteneurisée et communique avec les composants du maillage de services, récupère et traite les données, et expose ces données à la console. L'application Kiali n'a pas besoin de stockage. Lors du déploiement de l'application sur un cluster, les configurations sont définies dans des ConfigMaps et des secrets.

- **Kiali console** (front end) - La console Kiali est une application web. L'application Kiali sert la console Kiali, qui interroge ensuite le back-end pour obtenir des données et les présenter à l'utilisateur.

En outre, Kiali dépend de services et de composants externes fournis par la plateforme d'application de conteneurs et Istio.

- **Red Hat Service Mesh** (Istio) - Istio est une exigence de Kiali. Istio est le composant qui fournit et contrôle le maillage de services. Bien que Kiali et Istio puissent être installés séparément, Kiali dépend d'Istio et ne fonctionnera pas s'il n'est pas présent. Kiali doit récupérer les données et les configurations d'Istio, qui sont exposées via Prometheus et l'API du cluster.

- **Prometheus** - Une instance Prometheus dédiée est incluse dans l'installation de Red Hat OpenShift Service Mesh. Lorsque la télémétrie Istio est activée, les données de métriques sont stockées dans Prometheus. Kiali utilise ces données Prometheus pour déterminer la topologie du maillage, afficher les métriques, calculer la santé, montrer les problèmes éventuels, etc. Kiali

communique directement avec Prometheus et assume le schéma de données utilisé par Istio Telemetry. Prometheus est une dépendance d'Istio et une dépendance dure pour Kiali, et de nombreuses fonctionnalités de Kiali ne fonctionneront pas sans Prometheus.

- **Cluster API** - Kiali utilise l'API de OpenShift Container Platform (cluster API) pour récupérer et résoudre les configurations de maillage de services. Kiali interroge l'API de cluster pour récupérer, par exemple, les définitions des espaces de noms, des services, des déploiements, des pods et d'autres entités. Kiali effectue également des requêtes pour résoudre les relations entre les différentes entités du cluster. L'API de cluster est également interrogée pour récupérer les configurations Istio comme les services virtuels, les règles de destination, les règles de routage, les passerelles, les quotas, etc.

- **Jaeger** - Jaeger est optionnel, mais est installé par défaut dans le cadre de l'installation de Red Hat OpenShift Service Mesh. Lorsque vous installez la plateforme de traçage distribué dans le cadre de l'installation par défaut de Red Hat OpenShift Service Mesh, la console Kiali comprend un onglet permettant d'afficher les données de traçage distribué. Notez que les données de traçage ne seront pas disponibles si vous désactivez la fonctionnalité de traçage distribué d'Istio. Notez également que l'utilisateur doit avoir accès à l'espace de noms où le plan de contrôle Service Mesh est installé pour afficher les données de traçage.

- **Grafana** - Grafana est optionnel, mais est installé par défaut dans le cadre de l'installation de Red Hat OpenShift Service Mesh. Lorsqu'elles sont disponibles, les pages de métriques de Kiali affichent des liens pour diriger l'utilisateur vers la même métrique dans Grafana. Notez que l'utilisateur doit avoir accès à l'espace de noms où le plan de contrôle Service Mesh est installé pour afficher les liens vers le tableau de bord Grafana et visualiser les données Grafana.

1.3.3.3. Caractéristiques du Kiali

La console Kiali est intégrée à Red Hat Service Mesh et offre les fonctionnalités suivantes :

- **Health** - Identifier rapidement les problèmes liés aux applications, aux services ou aux charges de travail.

- **Topology** - Visualisez la façon dont vos applications, services ou charges de travail communiquent via le graphe Kiali.

- **Metrics** - Des tableaux de bord prédéfinis vous permettent de visualiser le maillage des services et les performances des applications pour Go, Node.js, Quarkus, Spring Boot, Thorntail et Vert.x. Vous pouvez également créer vos propres tableaux de bord personnalisés. Quarkus, Spring Boot, Thorntail et Vert.x. Vous pouvez également créer vos propres tableaux de bord.

- **Tracing** - L'intégration avec Jaeger permet de suivre le parcours d'une requête à travers les différents microservices qui composent une application.

- **Validations** - Effectuer des validations avancées sur les objets Istio les plus courants (règles de destination, entrées de service, services virtuels, etc.)

- **Configuration** - Possibilité optionnelle de créer, mettre à jour et supprimer la configuration du routage Istio en utilisant des assistants ou directement dans l'éditeur YAML de la console Kiali.

1.3.4. Comprendre le traçage distribué

Chaque fois qu'un utilisateur effectue une action dans une application, une requête est exécutée par l'architecture qui peut nécessiter la participation de dizaines de services différents pour produire une réponse. Le cheminement de cette requête est une transaction distribuée. La plateforme de traçage

distribué vous permet d'effectuer un traçage distribué, qui suit le parcours d'une requête à travers les différents microservices qui composent une application.

Distributed tracing est une technique utilisée pour relier les informations relatives à différentes unités de travail - généralement exécutées dans différents processus ou hôtes - afin de comprendre l'ensemble de la chaîne d'événements d'une transaction distribuée. Le traçage distribué permet aux développeurs de visualiser les flux d'appels dans les grandes architectures orientées services. Il peut s'avérer très utile pour comprendre la sérialisation, le parallélisme et les sources de latence.

La plateforme de traçage distribuée enregistre l'exécution des demandes individuelles dans l'ensemble de la pile de microservices et les présente sous forme de traces. Un site **trace** est un chemin de données/d'exécution à travers le système. Une trace de bout en bout comprend une ou plusieurs travées.

Un site **span** représente une unité logique de travail qui comporte un nom d'opération, l'heure de début de l'opération et sa durée. Les travées peuvent être imbriquées et ordonnées pour modéliser les relations de cause à effet.

1.3.4.1. Aperçu du traçage distribué

En tant que propriétaire de service, vous pouvez utiliser le traçage distribué pour instrumenter vos services afin de recueillir des informations sur votre architecture de service. Vous pouvez utiliser le traçage distribué pour la surveillance, le profilage du réseau et le dépannage de l'interaction entre les composants dans les applications modernes, cloud-natives et basées sur les microservices.

Le traçage distribué permet d'exécuter les fonctions suivantes :

- Contrôler les transactions distribuées
- Optimiser les performances et la latence
- Effectuer une analyse des causes profondes

Le traçage distribué de Red Hat OpenShift se compose de deux éléments principaux :

- **Red Hat OpenShift distributed tracing platform** - Ce composant est basé sur le projet open source Jaeger.
- **Red Hat OpenShift distributed tracing data collection** - Ce composant est basé sur le projet open source OpenTelemetry.

Ces deux composants sont basés sur les API et l'instrumentation OpenTracing, neutres vis-à-vis des fournisseurs.

1.3.4.2. Architecture de traçage distribuée Red Hat OpenShift

Le traçage distribué de Red Hat OpenShift est constitué de plusieurs composants qui fonctionnent ensemble pour collecter, stocker et afficher les données de traçage.

- **Red Hat OpenShift distributed tracing platform** - Ce composant est basé sur le projet open source Jaeger.
  - **Client** (Client Jaeger, Tracer, Reporter, application instrumentée, bibliothèques clientes) - Les clients de la plateforme de traçage distribué sont des implémentations de l'API OpenTracing spécifiques à chaque langue. Ils peuvent être utilisés pour instrumenter des applications pour le traçage distribué, soit manuellement, soit avec une variété de

frameworks open source existants, tels que Camel (Fuse), Spring Boot (RHOAR), MicroProfile (RHOAR/Thorntail), Wildfly (EAP), et bien d'autres, qui sont déjà intégrés à OpenTracing.

- **Agent** (Agent Jaeger, file d'attente du serveur, travailleurs du processeur) - L'agent de la plate-forme de traçage distribuée est un démon de réseau qui écoute les données envoyées via le protocole UDP (User Datagram Protocol), qu'il met en lots et envoie au collecteur. L'agent est censé être placé sur le même hôte que l'application instrumentée. Cela se fait généralement en ayant un sidecar dans les environnements de conteneurs tels que Kubernetes.

- **Jaeger Collector** (Collecteur, file d'attente, travailleurs) - Comme l'agent Jaeger, le collecteur Jaeger reçoit les travées et les place dans une file d'attente interne en vue de leur traitement. Cela permet au collecteur Jaeger de retourner immédiatement au client/à l'agent au lieu d'attendre que la travée se rende au stockage.

- **Storage** (Magasin de données) - Les collecteurs ont besoin d'un backend de stockage persistant. La plateforme de traçage distribuée Red Hat OpenShift dispose d'un mécanisme enfichable pour le stockage de la portée. Notez que pour cette version, le seul stockage pris en charge est Elasticsearch.

- **Query** (Service d'interrogation) - L'interrogation est un service qui permet d'extraire des traces du stockage.

- **Ingester** (Ingester Service) - Le traçage distribué Red Hat OpenShift peut utiliser Apache Kafka comme tampon entre le collecteur et le stockage Elasticsearch. Ingester est un service qui lit les données de Kafka et les écrit dans le backend de stockage Elasticsearch.

- **Jaeger Console** - L'interface utilisateur de la plateforme de traçage distribué Red Hat OpenShift vous permet de visualiser vos données de traçage distribué. Sur la page Recherche, vous pouvez trouver des traces et explorer les détails des portées qui composent une trace individuelle.

- **Red Hat OpenShift distributed tracing data collection**- Ce composant est basé sur le projet open source OpenTelemetry.

  - **OpenTelemetry Collector** - Le collecteur OpenTelemetry permet de recevoir, de traiter et d'exporter des données de télémétrie sans tenir compte des fournisseurs. Le collecteur OpenTelemetry prend en charge les formats de données d'observabilité open-source, par exemple Jaeger et Prometheus, en les envoyant à un ou plusieurs back-ends open-source ou commerciaux. Le collecteur est l'emplacement par défaut où les bibliothèques d'instrumentation exportent leurs données de télémétrie.

1.3.4.3. Fonctionnalités de traçage distribué de Red Hat OpenShift

Le traçage distribué de Red Hat OpenShift offre les capacités suivantes :

- Intégration avec Kiali - Lorsque la configuration est correcte, vous pouvez visualiser les données de traçage distribuées à partir de la console Kiali.

- Grande évolutivité - Le back-end de traçage distribué est conçu pour ne pas avoir de point de défaillance unique et pour s'adapter aux besoins de l'entreprise.

- Propagation du contexte distribué - Permet de relier des données provenant de différents composants afin de créer une trace complète de bout en bout.

- Compatibilité ascendante avec Zipkin - Le traçage distribué Red Hat OpenShift possède des API qui lui permettent d'être utilisé comme un remplacement direct de Zipkin, mais Red Hat ne prend pas en charge la compatibilité avec Zipkin dans cette version.

1.3.5. Prochaines étapes

- Préparez-vous à installer Red Hat OpenShift Service Mesh dans votre environnement OpenShift Container Platform.

1.4. MODÈLES DE DÉPLOIEMENT DU MAILLAGE DE SERVICES

Red Hat OpenShift Service Mesh prend en charge plusieurs modèles de déploiement différents qui peuvent être combinés de différentes manières pour répondre au mieux aux exigences de votre entreprise.

1.4.1. Modèle de déploiement à une seule maille

Le modèle de déploiement Istio le plus simple est un maillage unique.

Les noms de service au sein d'un maillage doivent être uniques car Kubernetes n'autorise qu'un seul service à être nommé **myservice** dans l'espace de noms **mynamespace**. Cependant, les instances de charge de travail peuvent partager une identité commune puisque les noms de compte de service peuvent être partagés entre les charges de travail dans le même espace de noms

1.4.2. Modèle de déploiement en mode "single tenancy

Dans Istio, un locataire est un groupe d'utilisateurs qui partagent un accès et des privilèges communs pour un ensemble de charges de travail déployées. Vous pouvez utiliser les locataires pour fournir un niveau d'isolation entre différentes équipes. Vous pouvez séparer l'accès aux différents locataires en utilisant les annotations **NetworkPolicies**, **AuthorizationPolicies**, et **exportTo** sur istio.io ou les ressources de service.

Les configurations de plan de contrôle Service Mesh à un seul locataire et à l'échelle d'un cluster sont obsolètes à partir de la version 1.0 de Red Hat OpenShift Service Mesh. Red Hat OpenShift Service Mesh utilise par défaut un modèle multilocataire.

1.4.3. Modèle de déploiement multilocataire

Red Hat OpenShift Service Mesh installe un site **ServiceMeshControlPlane** qui est configuré pour le multitenant par défaut. Red Hat OpenShift Service Mesh utilise un opérateur multitenant pour gérer le cycle de vie du plan de contrôle de Service Mesh. Au sein d'un maillage, les espaces de noms sont utilisés pour la location.

Red Hat OpenShift Service Mesh utilise les ressources **ServiceMeshControlPlane** pour gérer les installations de maillage, dont la portée est limitée par défaut à l'espace de noms qui contient la ressource. Vous utilisez les ressources **ServiceMeshMemberRoll** et **ServiceMeshMember** pour inclure des espaces de noms supplémentaires dans le maillage. Un espace de noms ne peut être inclus que dans un seul maillage, et plusieurs maillages peuvent être installés dans un seul cluster OpenShift.

Les déploiements typiques de service mesh utilisent un seul plan de contrôle Service Mesh pour configurer la communication entre les services dans le maillage. Red Hat OpenShift Service Mesh prend en charge le "soft multitenancy", où il y a un plan de contrôle et un maillage par locataire, et il peut y avoir plusieurs plans de contrôle indépendants au sein du cluster. Les déploiements multitenant spécifient les projets qui peuvent accéder au Service Mesh et isolent le Service Mesh des autres instances de plan de contrôle.

L'administrateur du cluster a le contrôle et la visibilité sur tous les plans de contrôle Istio, tandis que l'administrateur du locataire n'a le contrôle que sur ses instances Service Mesh, Kiali et Jaeger.

Vous pouvez autoriser une équipe à déployer ses charges de travail uniquement dans un espace de noms donné ou dans un ensemble d'espaces de noms. Si l'administrateur du maillage de services leur accorde le rôle **mesh-user**, les utilisateurs peuvent créer une ressource **ServiceMeshMember** pour ajouter des espaces de noms à l'espace de noms **ServiceMeshMemberRoll**.

1.4.4. Modèle de déploiement multi-méthodes ou fédéré

Federation est un modèle de déploiement qui vous permet de partager des services et des charges de travail entre des mailles séparées gérées dans des domaines administratifs distincts.

Le modèle multi-cluster d'Istio nécessite un niveau élevé de confiance entre les mailles et un accès à distance à tous les serveurs API Kubernetes sur lesquels résident les mailles individuelles. La fédération Red Hat OpenShift Service Mesh adopte une approche fondée sur l'opinion pour une mise en œuvre multi-clusters de Service Mesh qui suppose *minimal* la confiance entre les mailles.

Un site *federated mesh* est un groupe de mailles se comportant comme une seule maille. Les services de chaque maille peuvent être des services uniques, par exemple une maille ajoutant des services en les important d'une autre maille, peut fournir des charges de travail supplémentaires pour les mêmes services à travers les mailles, fournissant une haute disponibilité, ou une combinaison des deux. Toutes les mailles qui sont jointes à une maille fédérée restent gérées individuellement, et vous devez configurer explicitement les services qui sont exportés et importés depuis d'autres mailles de la fédération. Les fonctions de support telles que la génération de certificats, les métriques et la collecte de traces restent locales dans leurs mailles respectives.

1.5. DIFFÉRENCES ENTRE SERVICE MESH ET ISTIO

Red Hat OpenShift Service Mesh diffère d'une installation d'Istio pour fournir des fonctionnalités supplémentaires ou pour gérer les différences lors du déploiement sur OpenShift Container Platform.

1.5.1. Différences entre Istio et Red Hat OpenShift Service Mesh

Les fonctionnalités suivantes sont différentes dans Service Mesh et Istio.

1.5.1.1. Outil de ligne de commande

L'outil de ligne de commande pour Red Hat OpenShift Service Mesh est **oc**. Red Hat OpenShift Service Mesh ne prend pas en charge **istioctl**.

1.5.1.2. Installation et mise à niveau

Red Hat OpenShift Service Mesh ne prend pas en charge les profils d'installation Istio.

Red Hat OpenShift Service Mesh ne prend pas en charge les mises à niveau canari du maillage de services.

1.5.1.3. Injection automatique

L'installation de la communauté Istio en amont injecte automatiquement le sidecar dans les pods des projets que vous avez étiquetés.

Red Hat OpenShift Service Mesh n'injecte pas automatiquement le sidecar dans les pods, mais vous

devez opter pour l'injection à l'aide d'une annotation sans étiqueter les projets. Cette méthode nécessite moins de privilèges et n'entre pas en conflit avec d'autres capacités d'OpenShift Container Platform telles que les pods de construction. Pour activer l'injection automatique, spécifiez l'étiquette **sidecar.istio.io/inject**, ou l'annotation, comme décrit dans la section *Automatic sidecar injection*.

Tableau 1.3. Paramètres de l'étiquette d'injection du sidecar et des annotations

	Amont Istio	Red Hat OpenShift Service Mesh
Étiquette de l'espace de noms	prend en charge les "personnes habilitées" et les "personnes handicapées"	soutient "disabled"
Étiquette de la cosse	prend en charge "vrai" et "faux"	prend en charge "vrai" et "faux"
Annotation du pod	ne prend en charge que les "faux"	prend en charge "vrai" et "faux"

1.5.1.4. Fonctionnalités du contrôle d'accès basé sur les rôles d'Istio

Le contrôle d'accès basé sur les rôles (RBAC) d'Istio fournit un mécanisme que vous pouvez utiliser pour contrôler l'accès à un service. Vous pouvez identifier les sujets par nom d'utilisateur ou en spécifiant un ensemble de propriétés et appliquer les contrôles d'accès en conséquence.

L'installation de la communauté Istio en amont inclut des options pour effectuer des correspondances exactes d'en-tête, des correspondances de caractères génériques dans les en-têtes, ou pour vérifier un en-tête contenant un préfixe ou un suffixe spécifique.

Red Hat OpenShift Service Mesh étend la capacité de faire correspondre les en-têtes de requête en utilisant une expression régulière. Spécifiez une clé de propriété de **request.regex.headers** avec une expression régulière.

Exemple d'en-tête de requête de correspondance de communauté Istio en amont

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

1.5.1.5. OpenSSL

Red Hat OpenShift Service Mesh remplace BoringSSL par OpenSSL. OpenSSL est une bibliothèque logicielle qui contient une implémentation open source des protocoles Secure Sockets Layer (SSL) et Transport Layer Security (TLS). Le binaire Red Hat OpenShift Service Mesh Proxy lie dynamiquement

les bibliothèques OpenSSL (libssl et libcrypto) à partir du système d'exploitation Red Hat Enterprise Linux sous-jacent.

1.5.1.6. Charges de travail externes

Red Hat OpenShift Service Mesh ne prend pas en charge les charges de travail externes, telles que les machines virtuelles exécutées en dehors d'OpenShift sur des serveurs bare metal.

1.5.1.7. Prise en charge des machines virtuelles

Vous pouvez déployer des machines virtuelles sur OpenShift en utilisant OpenShift Virtualization. Ensuite, vous pouvez appliquer une politique de maillage, telle que mTLS ou AuthorizationPolicy, à ces machines virtuelles, comme n'importe quel autre pod faisant partie d'un maillage.

1.5.1.8. Modifications des composants

- Une étiquette *maistra-version* a été ajoutée à toutes les ressources.
- Toutes les ressources Ingress ont été converties en ressources OpenShift Route.
- Grafana, le traçage distribué (Jaeger) et Kiali sont activés par défaut et exposés à travers les routes OpenShift.
- Godebug a été supprimé de tous les modèles
- Les liens **istio-multi** ServiceAccount et ClusterRoleBinding ont été supprimés, ainsi que le lien **istio-reader** ClusterRole.

1.5.1.9. Filtres Envoy

Red Hat OpenShift Service Mesh ne prend pas en charge la configuration **EnvoyFilter**, sauf lorsqu'elle est explicitement documentée. En raison du couplage étroit avec les API Envoy sous-jacentes, la compatibilité ascendante ne peut pas être maintenue. Les correctifs **EnvoyFilter** sont très sensibles au format de la configuration Envoy qui est générée par Istio. Si la configuration générée par Istio change, il est possible que l'application du correctif **EnvoyFilter** soit interrompue.

1.5.1.10. Services Envoy

Red Hat OpenShift Service Mesh ne prend pas en charge les services basés sur QUIC.

1.5.1.11. Plugin Istio Container Network Interface (CNI)

Red Hat OpenShift Service Mesh inclut le plugin CNI, qui vous offre une autre façon de configurer la mise en réseau des pods d'application. Le plugin CNI remplace la configuration du réseau **init-container**, ce qui élimine la nécessité d'accorder aux comptes de service et aux projets l'accès aux contraintes de contexte de sécurité (SCC) avec des privilèges élevés.

1.5.1.12. Paramètres mTLS globaux

Red Hat OpenShift Service Mesh crée une ressource **PeerAuthentication** qui active ou désactive l'authentification mutuelle TLS (mTLS) au sein du maillage.

1.5.1.13. Passerelles

Red Hat OpenShift Service Mesh installe des passerelles d'entrée et de sortie par défaut. Vous pouvez désactiver l'installation des passerelles dans la ressource **ServiceMeshControlPlane** (SMCP) en utilisant les paramètres suivants :

- **spec.gateways.enabled=false** pour désactiver les passerelles d'entrée et de sortie.
- **spec.gateways.ingress.enabled=false** pour désactiver les passerelles d'entrée.
- **spec.gateways.egress.enabled=false** pour désactiver les passerelles de sortie.

NOTE

L'opérateur annote les passerelles par défaut pour indiquer qu'elles sont générées et gérées par l'opérateur Red Hat OpenShift Service Mesh.

1.5.1.14. Configurations multicluster

La prise en charge de Red Hat OpenShift Service Mesh pour les configurations multicluster est limitée à la fédération de maillages de services sur plusieurs clusters.

1.5.1.15. Demandes de signature de certificat personnalisées (CSR)

Vous ne pouvez pas configurer Red Hat OpenShift Service Mesh pour traiter les CSR via l'autorité de certification (CA) de Kubernetes.

1.5.1.16. Routes pour les passerelles Istio

Les routes OpenShift pour les passerelles Istio sont automatiquement gérées dans Red Hat OpenShift Service Mesh. Chaque fois qu'une passerelle Istio est créée, mise à jour ou supprimée dans le service mesh, une route OpenShift est créée, mise à jour ou supprimée.

Un composant du plan de contrôle de Red Hat OpenShift Service Mesh appelé Istio OpenShift Routing (IOR) synchronise l'itinéraire de la passerelle. Pour plus d'informations, voir Création automatique d'itinéraires.

1.5.1.16.1. Domaines fourre-tout

Les domaines fourre-tout ("*") ne sont pas pris en charge. Si un tel domaine est trouvé dans la définition de la passerelle, Red Hat OpenShift Service Mesh *will* créera la route, mais s'appuiera sur OpenShift pour créer un nom d'hôte par défaut. Cela signifie que la route nouvellement créée *not* sera une route "catch all" ("*"), au lieu de cela, elle aura un nom d'hôte de la forme **<route-name>[-<project>].<suffix>**. Consultez la documentation d'OpenShift Container Platform pour plus d'informations sur le fonctionnement des noms d'hôtes par défaut et sur la façon dont **cluster-admin** peut les personnaliser. Si vous utilisez Red Hat OpenShift Dedicated, reportez-vous au rôle **dedicated-admin** de Red Hat OpenShift Dedicated.

1.5.1.16.2. Sous-domaines

Les sous-domaines (par exemple : "*.domain.com") sont pris en charge. Cependant, cette capacité n'est pas activée par défaut dans OpenShift Container Platform. Cela signifie que Red Hat OpenShift Service Mesh *will* crée la route avec le sous-domaine, mais qu'elle ne sera effective que si OpenShift Container Platform est configuré pour l'activer.

1.5.1.16.3. Sécurité de la couche transport

Transport Layer Security (TLS) est pris en charge. Cela signifie que, si la passerelle contient une section **tls**, la route OpenShift sera configurée pour prendre en charge TLS.

Ressources supplémentaires

- Création automatique d'itinéraires

1.5.2. Installations multi-locataires

Alors qu'Istio en amont adopte une approche à locataire unique, Red Hat OpenShift Service Mesh prend en charge plusieurs plans de contrôle indépendants au sein du cluster. Red Hat OpenShift Service Mesh utilise un opérateur multilocataire pour gérer le cycle de vie du plan de contrôle.

Red Hat OpenShift Service Mesh installe un plan de contrôle multitenant par défaut. Vous spécifiez les projets qui peuvent accéder au Service Mesh, et vous isolez le Service Mesh des autres instances du plan de contrôle.

1.5.2.1. Multitenancy ou installations en grappe

La principale différence entre une installation multitenant et une installation à l'échelle d'un cluster est l'étendue des privilèges utilisés par istod. Les composants n'utilisent plus les ressources du contrôle d'accès basé sur les rôles (RBAC) à l'échelle du cluster **ClusterRoleBinding**.

Chaque projet de la liste **ServiceMeshMemberRoll members** aura un **RoleBinding** pour chaque compte de service associé au déploiement du plan de contrôle et chaque déploiement du plan de contrôle ne surveillera que ces projets membres. Chaque projet membre se voit ajouter une étiquette **maistra.io/member-of**, où la valeur **member-of** correspond au projet contenant l'installation du plan de contrôle.

Red Hat OpenShift Service Mesh configure chaque projet membre pour assurer l'accès au réseau entre lui-même, le plan de contrôle et les autres projets membres. La configuration exacte diffère en fonction de la façon dont le réseau défini par logiciel (SDN) de OpenShift Container Platform est configuré. Voir À propos d'OpenShift SDN pour plus de détails.

Si le cluster OpenShift Container Platform est configuré pour utiliser le plugin SDN :

- **NetworkPolicy**: Red Hat OpenShift Service Mesh crée une ressource **NetworkPolicy** dans chaque projet membre permettant l'entrée de tous les pods depuis les autres membres et le plan de contrôle. Si vous supprimez un membre de Service Mesh, cette ressource **NetworkPolicy** est supprimée du projet.

  NOTE

  Cela limite également les entrées aux seuls projets membres. Si vous avez besoin de l'entrée de projets non membres, vous devez créer un site **NetworkPolicy** pour permettre le passage de ce trafic.

- **Multitenant**: Red Hat OpenShift Service Mesh joint le **NetNamespace** de chaque projet membre au **NetNamespace** du projet du plan de contrôle (l'équivalent de l'exécution de **oc adm pod-network join-projects --to control-plane-project member-project**). Si vous retirez un membre du Service Mesh, son **NetNamespace** est isolé du plan de contrôle (l'équivalent de l'exécution de **oc adm pod-network isolate-projects member-project**).

- **Subnet**: Aucune configuration supplémentaire n'est effectuée.

1.5.2.2. Ressources en grappe

Upstream Istio dispose de deux ressources de type " cluster scoped " sur lesquelles il s'appuie. Les ressources **MeshPolicy** et **ClusterRbacConfig** ne sont pas compatibles avec un cluster multitenant et ont été remplacées comme décrit ci-dessous.

- *ServiceMeshPolicy* remplace MeshPolicy pour la configuration des politiques d'authentification à l'échelle du plan de contrôle. Elle doit être créée dans le même projet que le plan de contrôle.
- *ServicemeshRbacConfig* remplace ClusterRbacConfig pour la configuration du contrôle d'accès basé sur les rôles à l'échelle du plan de contrôle. Il doit être créé dans le même projet que le plan de contrôle.

1.5.3. Kiali et maille de service

L'installation de Kiali via le Service Mesh sur OpenShift Container Platform diffère des installations communautaires de Kiali de plusieurs façons. Ces modifications sont parfois nécessaires pour résoudre des problèmes, fournir des fonctionnalités supplémentaires ou gérer les différences lors du déploiement sur OpenShift Container Platform.

- Kiali a été activé par défaut.
- L'entrée a été activée par défaut.
- Des mises à jour ont été apportées au ConfigMap de Kiali.
- Des mises à jour ont été apportées aux paramètres ClusterRole pour Kiali.
- Ne modifiez pas le ConfigMap, car vos modifications pourraient être écrasées par le Service Mesh ou les opérateurs Kiali. Les fichiers gérés par l'opérateur Kiali portent une étiquette ou une annotation **kiali.io**/. La mise à jour des fichiers de l'opérateur doit être limitée aux utilisateurs disposant des privilèges **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, la mise à jour des fichiers de l'opérateur doit être limitée aux utilisateurs disposant des privilèges **dedicated-admin**.

1.5.4. Traçage distribué et maillage des services

L'installation de la plateforme de traçage distribuée avec le Service Mesh sur OpenShift Container Platform diffère des installations communautaires de Jaeger de plusieurs façons. Ces modifications sont parfois nécessaires pour résoudre des problèmes, fournir des fonctionnalités supplémentaires ou gérer les différences lors du déploiement sur OpenShift Container Platform.

- Le traçage distribué a été activé par défaut pour Service Mesh.
- L'entrée a été activée par défaut pour le Service Mesh.
- Le nom du port Zipkin a été changé en **jaeger-collector-zipkin** (au lieu de **http**)
- Jaeger utilise Elasticsearch pour le stockage par défaut lorsque vous sélectionnez l'option de déploiement **production** ou **streaming**.
- La version communautaire d'Istio fournit une route générique "tracing". Red Hat OpenShift Service Mesh utilise une route "jaeger" qui est installée par l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift et qui est déjà protégée par OAuth.
- Red Hat OpenShift Service Mesh utilise un sidecar pour le proxy Envoy, et Jaeger utilise

également un sidecar, pour l'agent Jaeger. Ces deux sidecars sont configurés séparément et ne doivent pas être confondus l'un avec l'autre. Le sidecar du proxy crée des travées liées au trafic d'entrée et de sortie du pod. Le sidecar agent reçoit les spans émis par l'application et les envoie au collecteur Jaeger.

1.6. PRÉPARATION DE L'INSTALLATION DE SERVICE MESH

Avant de pouvoir installer Red Hat OpenShift Service Mesh, vous devez vous abonner à OpenShift Container Platform et installer OpenShift Container Platform dans une configuration prise en charge.

1.6.1. Conditions préalables

- Maintenir un abonnement OpenShift Container Platform actif sur votre compte Red Hat. Si vous n'avez pas d'abonnement, contactez votre représentant commercial pour plus d'informations.
- Consultez la présentation de OpenShift Container Platform 4.12 .
- Installez OpenShift Container Platform 4.12. Si vous installez Red Hat OpenShift Service Mesh sur un réseau restreint, suivez les instructions de l'infrastructure OpenShift Container Platform que vous avez choisie.
  - Installer OpenShift Container Platform 4.12 sur AWS
  - Installer OpenShift Container Platform 4.12 sur AWS fourni par l'utilisateur
  - Installer OpenShift Container Platform 4.12 sur du métal nu
  - Installer OpenShift Container Platform 4.12 sur vSphere
  - Installer OpenShift Container Platform 4.12 sur IBM zSystems et IBM® LinuxONE
  - Installer OpenShift Container Platform 4.12 sur IBM Power
- Installez la version de l'utilitaire de ligne de commande OpenShift Container Platform (l'outil client **oc**) qui correspond à votre version d'OpenShift Container Platform et ajoutez-la à votre chemin d'accès.
  - Si vous utilisez OpenShift Container Platform 4.12, voir À propos de l'OpenShift CLI.

Pour plus d'informations sur le cycle de vie de Red Hat OpenShift Service Mesh et les plateformes prises en charge, reportez-vous à la Politique d'assistance.

1.6.2. Configurations prises en charge

Les configurations suivantes sont prises en charge pour la version actuelle de Red Hat OpenShift Service Mesh.

1.6.2.1. Plates-formes prises en charge

L'opérateur Red Hat OpenShift Service Mesh prend en charge plusieurs versions de la ressource **ServiceMeshControlPlane**. Les plans de contrôle Service Mesh de la version 2.3 sont pris en charge sur les versions de plateforme suivantes :

- Red Hat OpenShift Container Platform version 4.9 ou ultérieure.

- Red Hat OpenShift Dedicated version 4.
- Azure Red Hat OpenShift (ARO) version 4.
- Red Hat OpenShift Service on AWS (ROSA).

1.6.2.2. Configurations non prises en charge

Les cas explicitement non pris en charge sont les suivants :

- OpenShift Online n'est pas pris en charge pour Red Hat OpenShift Service Mesh.
- Red Hat OpenShift Service Mesh ne prend pas en charge la gestion des microservices en dehors du cluster où Service Mesh est exécuté.

1.6.2.3. Configurations réseau prises en charge

Red Hat OpenShift Service Mesh prend en charge les configurations réseau suivantes.

- OpenShift-SDN
- OVN-Kubernetes est pris en charge sur OpenShift Container Platform 4.7.32 , OpenShift Container Platform 4.8.12 , et OpenShift Container Platform 4.9 .
- Plugins CNI (Container Network Interface) tiers qui ont été certifiés sur OpenShift Container Platform et qui ont passé les tests de conformité Service Mesh. Voir Plugins CNI OpenShift certifiés pour plus d'informations.

1.6.2.4. Configurations prises en charge pour Service Mesh

- Cette version de Red Hat OpenShift Service Mesh est uniquement disponible sur OpenShift Container Platform x86_64, IBM zSystems et IBM Power.
  - IBM zSystems n'est pris en charge que sur OpenShift Container Platform 4.6 et plus.
  - IBM Power n'est pris en charge que sur OpenShift Container Platform 4.6 et plus.
- Configurations dans lesquelles tous les composants Service Mesh sont contenus dans un seul cluster OpenShift Container Platform.
- Configurations qui n'intègrent pas de services externes tels que des machines virtuelles.
- Red Hat OpenShift Service Mesh ne prend pas en charge la configuration **EnvoyFilter**, sauf lorsqu'elle est explicitement documentée.

1.6.2.5. Configurations prises en charge pour Kiali

- La console Kiali n'est compatible qu'avec les deux versions les plus récentes des navigateurs Chrome, Edge, Firefox ou Safari.

1.6.2.6. Configurations prises en charge pour le traçage distribué

- L'agent Jaeger en tant que sidecar est la seule configuration supportée pour Jaeger. Jaeger en tant que daemonset n'est pas pris en charge pour les installations multitenant ou OpenShift Dedicated.

1.6.2.7. Module WebAssembly pris en charge

- 3scale WebAssembly est le seul module WebAssembly fourni. Vous pouvez créer des modules WebAssembly personnalisés.

1.6.3. Prochaines étapes

- Installez Red Hat OpenShift Service Mesh dans votre environnement OpenShift Container Platform.

1.7. INSTALLATION DES OPÉRATEURS

Pour installer Red Hat OpenShift Service Mesh, installez d'abord les opérateurs requis sur OpenShift Container Platform, puis créez une ressource **ServiceMeshControlPlane** pour déployer le plan de contrôle.

NOTE

Cette installation de base est configurée sur la base des paramètres par défaut d'OpenShift et n'est pas conçue pour une utilisation en production. Utilisez cette installation par défaut pour vérifier votre installation, puis configurez votre service mesh pour votre environnement spécifique.

Conditions préalables

- Lisez la section Préparation de l'installation de Red Hat OpenShift Service Mesh .
- Un compte avec le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

Les étapes suivantes montrent comment installer une instance de base de Red Hat OpenShift Service Mesh sur OpenShift Container Platform.

1.7.1. Vue d'ensemble de l'opérateur

Red Hat OpenShift Service Mesh nécessite les quatre opérateurs suivants :

- **OpenShift Elasticsearch** - (Facultatif) Fournit un stockage de base de données pour le traçage et la journalisation avec la plateforme de traçage distribuée. Il est basé sur le projet open source Elasticsearch.
- **Red Hat OpenShift distributed tracing platform** - Fournit un traçage distribué pour surveiller et dépanner les transactions dans les systèmes distribués complexes. Il est basé sur le projet open source Jaeger.
- **Kiali** - Fournit une observabilité pour votre maillage de services. Il vous permet de visualiser les configurations, de surveiller le trafic et d'analyser les traces dans une console unique. Il est basé sur le projet open source Kiali.
- **Red Hat OpenShift Service Mesh** - Il vous permet de connecter, de sécuriser, de contrôler et d'observer les microservices qui composent vos applications. Le Service Mesh Operator définit et surveille les ressources **ServiceMeshControlPlane** qui gèrent le déploiement, la mise à jour et la suppression des composants du Service Mesh. Il est basé sur le projet open source Istio.

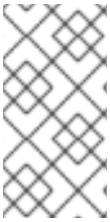
AVERTISSEMENT

N'installez pas les versions communautaires des opérateurs. Les opérateurs communautaires ne sont pas pris en charge.

1.7.2. Installation des opérateurs

Pour installer Red Hat OpenShift Service Mesh, installez les opérateurs suivants dans cet ordre. Répétez la procédure pour chaque opérateur.

- OpenShift Elasticsearch
- Plateforme de traçage distribuée Red Hat OpenShift
- Kiali
- Red Hat OpenShift Service Mesh



NOTE

Si vous avez déjà installé l'OpenShift Elasticsearch Operator dans le cadre d'OpenShift Logging, vous n'avez pas besoin d'installer à nouveau l'OpenShift Elasticsearch Operator. L'opérateur de la plateforme de traçage distribuée Red Hat OpenShift créera l'instance Elasticsearch à l'aide de l'opérateur OpenShift Elasticsearch installé.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur avec le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.
2. Dans la console Web OpenShift Container Platform, cliquez sur **Operators → OperatorHub**.
3. Tapez le nom de l'opérateur dans la boîte de filtre et sélectionnez la version Red Hat de l'opérateur. Les versions communautaires des opérateurs ne sont pas prises en charge.
4. Cliquez sur **Install**.
5. Sur la page **Install Operator** de chaque opérateur, acceptez les paramètres par défaut.
6. Cliquez sur **Install**. Attendez que l'opérateur soit installé avant de répéter les étapes pour l'opérateur suivant dans la liste.
    - OpenShift Elasticsearch Operator est installé dans l'espace de noms **openshift-operators-redhat** et est disponible pour tous les espaces de noms du cluster.
    - La plateforme de traçage distribuée Red Hat OpenShift est installée dans l'espace de noms **openshift-distributed-tracing** et est disponible pour tous les espaces de noms dans le cluster.

- Les opérateurs Kiali et Red Hat OpenShift Service Mesh sont installés dans l'espace de noms **openshift-operators** et sont disponibles pour tous les espaces de noms dans le cluster.

7. Après avoir installé les quatre opérateurs, cliquez sur **Operators → Installed Operators** pour vérifier que vos opérateurs sont installés.

1.7.3. Configuration de l'opérateur Service Mesh pour qu'il s'exécute sur les nœuds d'infrastructure

Cette tâche ne doit être effectuée que si l'opérateur Service Mesh fonctionne sur un nœud d'infrastructure.

Si l'opérateur s'exécute sur un nœud de travail, ignorez cette tâche.

Conditions préalables

- L'opérateur Service Mesh doit être installé.
- L'un des nœuds composant le déploiement doit être un nœud d'infrastructure. Pour plus d'informations, voir "Création d'ensembles de machines d'infrastructure"

Procédure

1. Modifiez la ressource Service Mesh Operator **Subscription** pour spécifier l'endroit où l'opérateur doit s'exécuter :

```
$ oc -n openshift-operators edit subscription <name> 1
```

1 **<name>** représente le nom de la ressource **Subscription**.

2. Ajoutez les adresses **nodeSelector** et **tolerations** à **spec.config** dans la ressource **Subscription**:

```
spec:
 config:
  nodeSelector: 1
    node-role.kubernetes.io/infra: ""
  tolerations: 2
  - effect: NoSchedule
    key: node-role.kubernetes.io/infra
    value: reserved
  - effect: NoExecute
    key: node-role.kubernetes.io/infra
    value: reserved
```

1 Veille à ce que le module de l'opérateur ne soit programmé que sur un nœud d'infrastructure.

2 Assure que le pod est accepté par le nœud d'infrastructure.

1.7.4. Vérification de l'exécution de Service Mesh Operator sur le nœud d'infrastructure

Procédure

- Vérifiez que le nœud associé au pod opérateur est un nœud d'infrastructure :

```
$ oc -n openshift-operators get po -l name=istio-operator -owide
```

1.7.5. Prochaines étapes

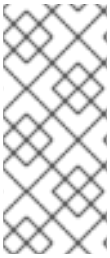
- L'opérateur Service Mesh de Red Hat OpenShift ne crée pas les définitions de ressources personnalisées (CRD) de Service Mesh tant que vous ne déployez pas un plan de contrôle Service Mesh. Vous pouvez utiliser la ressource **ServiceMeshControlPlane** pour installer et configurer les composants Service Mesh. Pour plus d'informations, voir Création du plan de contrôle ServiceMesh.

1.8. CRÉATION DU PLAN DE CONTRÔLE DU SERVICEMESH

Vous pouvez déployer une installation de base de **ServiceMeshControlPlane**(SMCP) en utilisant soit la console web d'OpenShift Container Platform, soit la ligne de commande à l'aide de l'outil client **oc**.

NOTE

Cette installation de base est configurée sur la base des paramètres par défaut d'OpenShift et n'est pas conçue pour une utilisation en production. Utilisez cette installation par défaut pour vérifier votre installation, puis configurez votre **ServiceMeshControlPlane** en fonction de votre environnement.



NOTE

Red Hat OpenShift Service on AWS (ROSA) impose des restrictions supplémentaires sur l'endroit où vous pouvez créer des ressources et, par conséquent, le déploiement par défaut ne fonctionne pas. Voir Installation de Service Mesh sur Red Hat OpenShift Service on AWS pour des exigences supplémentaires avant de déployer votre SMCP dans un environnement ROSA.

NOTE

La documentation Service Mesh utilise **istio-system** comme projet d'exemple, mais vous pouvez déployer le Service Mesh dans n'importe quel projet.

1.8.1. Déploiement du plan de contrôle Service Mesh à partir de la console web

Vous pouvez déployer une version de base de **ServiceMeshControlPlane** à l'aide de la console Web. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

Conditions préalables

- L'opérateur Red Hat OpenShift Service Mesh doit être installé.
- Un compte avec le rôle **cluster-admin**.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur avec le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

2. Créer un projet nommé **istio-system**.

   a. Naviguez jusqu'à **Home → Projects**.

   b. Cliquez sur **Create Project**.

   c. Dans le champ **Name**, entrez **istio-system**. La ressource **ServiceMeshControlPlane** doit être installée dans un projet distinct de vos microservices et de vos opérateurs.
   Ces étapes utilisent **istio-system** comme exemple, mais vous pouvez déployer votre plan de contrôle Service Mesh dans n'importe quel projet tant qu'il est séparé du projet qui contient vos services.

   d. Cliquez sur **Create**.

3. Naviguez jusqu'à **Operators → Installed Operators**.

4. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh, puis cliquez sur **Istio Service Mesh Control Plane**.

5. Dans l'onglet **Istio Service Mesh Control Plane**, cliquez sur **Create ServiceMeshControlPlane**.

6. Sur la page **Create ServiceMeshControlPlane**, acceptez la version par défaut du plan de contrôle de Service Mesh pour bénéficier des fonctionnalités disponibles dans la version la plus récente du produit. La version du plan de contrôle détermine les fonctionnalités disponibles quelle que soit la version de l'opérateur.
   Vous pouvez configurer les paramètres **ServiceMeshControlPlane** ultérieurement. Pour plus d'informations, voir Configuration de Red Hat OpenShift Service Mesh.

   a. Cliquez sur **Create**. L'opérateur crée des pods, des services et des composants du plan de contrôle Service Mesh en fonction de vos paramètres de configuration.

7. Pour vérifier que le plan de contrôle est correctement installé, cliquez sur l'onglet **Istio Service Mesh Control Plane**.

   a. Cliquez sur le nom du nouveau plan de contrôle.

   b. Cliquez sur l'onglet **Resources** pour voir les ressources du plan de contrôle de Red Hat OpenShift Service Mesh que l'opérateur a créées et configurées.

1.8.2. Déploiement du plan de contrôle Service Mesh à l'aide de la CLI

Vous pouvez déployer une version de base de **ServiceMeshControlPlane** à partir de la ligne de commande.

Conditions préalables

- L'opérateur Red Hat OpenShift Service Mesh doit être installé.
- Accès à la CLI OpenShift (**oc**).

Procédure

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. Créer un projet nommé **istio-system**.

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

3. Créez un fichier **ServiceMeshControlPlane** nommé **istio-installation.yaml** en utilisant l'exemple suivant. La version du plan de contrôle Service Mesh détermine les fonctionnalités disponibles indépendamment de la version de l'Opérateur.

Exemple version 2.3 istio-installation.yaml

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.3
  tracing:
    type: Jaeger
    sampling: 10000
  addons:
    jaeger:
      name: jaeger
      install:
        storage:
          type: Memory
    kiali:
      enabled: true
      name: kiali
    grafana:
      enabled: true
```

4. Exécutez la commande suivante pour déployer le plan de contrôle Service Mesh, où **<istio_installation.yaml>** inclut le chemin d'accès complet à votre fichier.

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

5. Pour suivre la progression du déploiement des modules, exécutez la commande suivante :

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

Vous devriez obtenir un résultat similaire à celui qui suit :

```
NAME                                  READY   STATUS    RESTARTS   AGE
grafana-b4d59bd7-mrgbr                2/2     Running   0          65m
istio-egressgateway-678dc97b4c-wrjkp  1/1     Running   0          108s
```

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

1.8.3. Validation de l'installation de SMCP avec le CLI

Vous pouvez valider la création du site **ServiceMeshControlPlane** à partir de la ligne de commande.

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

```
$ oc login https://<HOSTNAME>:6443
```

2. Exécutez la commande suivante pour vérifier l'installation du plan de contrôle Service Mesh, où **istio-system** est l'espace de noms dans lequel vous avez installé le plan de contrôle Service Mesh.

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

L'installation est terminée avec succès lorsque la colonne **STATUS** est **ComponentsReady**.

```
NAME    READY   STATUS            PROFILES      VERSION   AGE
basic   10/10   ComponentsReady   ["default"]   2.1.1     66m
```

1.8.4. Configuration de tous les composants du plan de contrôle de Service Mesh pour qu'ils s'exécutent sur des nœuds d'infrastructure

Cette tâche ne doit être effectuée que si tous les composants déployés par le plan de contrôle Service Mesh (notamment Istiod, Ingress Gateway et Egress Gateway) ainsi que les éléments facultatifs (tels que Prometheus, Grafana et Distributed Tracing) sont exécutés sur les nœuds d'infrastructure.

Si le plan de contrôle s'exécute sur un nœud de travail, ignorez cette tâche.

Procédure

1. Ouvrez la ressource **ServiceMeshControlPlane** en tant que fichier YAML :

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

1 **<name>** représente le nom de la ressource **ServiceMeshControlPlane**.

2. Pour exécuter tous les composants Service Mesh déployés par **ServiceMeshControlPlane** sur les nœuds d'infrastructure, ajoutez les champs **nodeSelector** et **tolerations** à la spécification **spec.runtime.defaults.pod** dans la ressource **ServiceMeshControlPlane**:

```
spec:
  runtime:
```

```
defaults:
  pod:
    nodeSelector: 1
      node-role.kubernetes.io/infra: ""
    tolerations: 2
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
```

1. Permet de s'assurer que les pods SMCP ne sont planifiés que sur un nœud d'infrastructure.
2. Assure que les pods sont acceptés par le nœud d'infrastructure.

1.8.5. Configuration des composants du plan de contrôle de Service Mesh pour qu'ils s'exécutent sur les nœuds d'infrastructure

Cette tâche ne doit être effectuée que si des composants individuels du plan de contrôle de Service Mesh (tels que Istiod, la passerelle d'entrée et la passerelle de sortie) sont exécutés sur des nœuds d'infrastructure.

Si le plan de contrôle est exécuté sur un nœud de travailleur, ignorez cette tâche.

Procédure

1. Ouvrez la ressource **ServiceMeshControlPlane** en tant que fichier YAML.

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

1. **<name>** représente le nom de la ressource **ServiceMeshControlPlane**.

2. Pour exécuter le composant Istiod sur un nœud d'infrastructure, ajoutez les champs **nodeSelector** et **tolerations** à la spécification **spec.runtime.components.pilot.pod** dans la ressource **ServiceMeshControlPlane**.

```
spec:
  runtime:
    components:
      pilot:
        pod:
          nodeSelector: 1
            node-role.kubernetes.io/infra: ""
          tolerations: 2
          - effect: NoSchedule
            key: node-role.kubernetes.io/infra
            value: reserved
          - effect: NoExecute
            key: node-role.kubernetes.io/infra
            value: reserved
```

1 Assure que le pod Istiod n'est programmé que sur un nœud d'infrastructure.

2 Assure que le pod est accepté par le nœud d'infrastructure.

3. Pour exécuter les passerelles d'entrée et de sortie sur les nœuds d'infrastructure, ajoutez les champs **nodeSelector** et **tolerations** à la ressource **spec.gateways.ingress.runtime.pod** et la ressource **spec.gateways.egress.runtime.pod** à la ressource **ServiceMeshControlPlane**.

```
spec:
  gateways:
    ingress:
      runtime:
        pod:
          nodeSelector: 1
            node-role.kubernetes.io/infra: ""
          tolerations: 2
          - effect: NoSchedule
            key: node-role.kubernetes.io/infra
            value: reserved
          - effect: NoExecute
            key: node-role.kubernetes.io/infra
            value: reserved
    egress:
      runtime:
        pod:
          nodeSelector: 3
            node-role.kubernetes.io/infra: ""
          tolerations: 4
          - effect: NoSchedule
            key: node-role.kubernetes.io/infra
            value: reserved
          - effect: NoExecute
            key: node-role.kubernetes.io/infra
            value: reserved
```

1 3 Veille à ce que le module de passerelle ne soit programmé que sur un nœud d'infrastructure

2 4 Assure que le pod est accepté par le nœud d'infrastructure.

1.8.6. Vérification de l'exécution du plan de contrôle Service Mesh sur les nœuds d'infrastructure

Procédure

- Confirmez que les nœuds associés aux pods Istiod, Ingress Gateway et Egress Gateway sont des nœuds d'infrastructure :

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

1.8.7. Valider votre installation SMCP avec Kiali

Vous pouvez utiliser la console Kiali pour valider votre installation Service Mesh. La console Kiali offre plusieurs moyens de valider que vos composants Service Mesh sont déployés et configurés correctement.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur disposant des droits cluster-admin. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

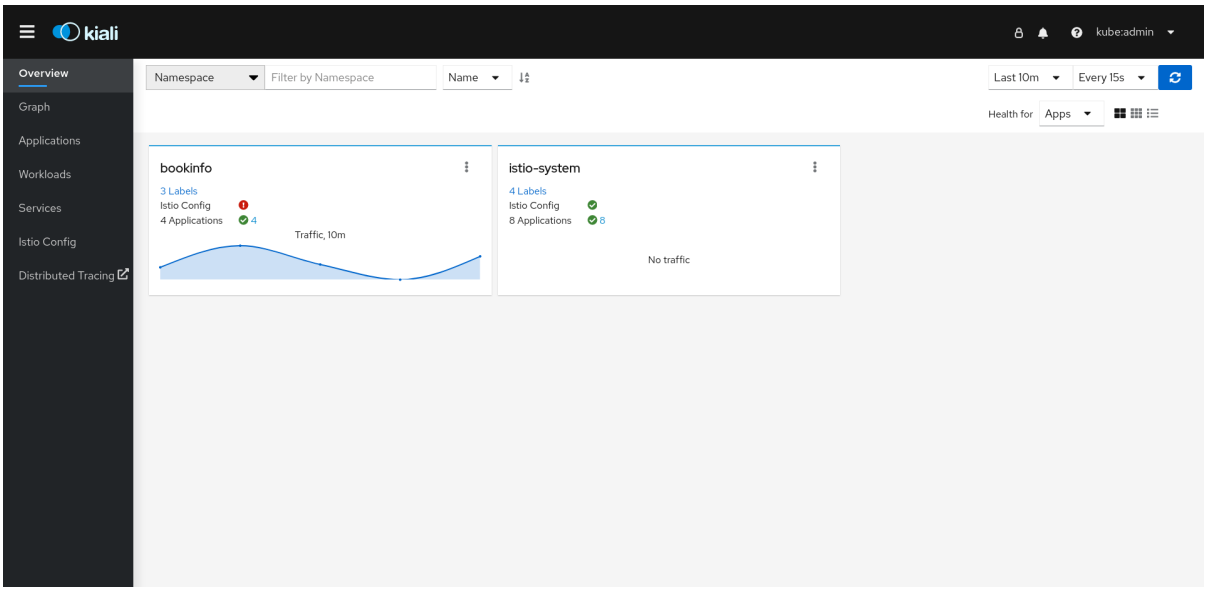
2. Naviguez jusqu'à **Networking → Routes**.

3. Sur la page **Routes**, sélectionnez le projet de plan de contrôle Service Mesh, par exemple **istio-system**, dans le menu **Namespace**.
   La colonne **Location** affiche l'adresse liée à chaque itinéraire.

4. Si nécessaire, utilisez le filtre pour trouver la route de la console Kiali. Cliquez sur l'itinéraire **Location** pour lancer la console.

5. Cliquez sur **Log In With OpenShift**
   Lorsque vous vous connectez pour la première fois à la console Kiali, vous voyez la page **Overview** qui affiche tous les espaces de noms de votre maillage de services que vous avez le droit de voir. Lorsque plusieurs espaces de noms sont affichés sur la page **Overview**, Kiali affiche en premier les espaces de noms présentant des problèmes de santé ou de validation.

   Figure 1.1. Page de présentation de Kiali



   La tuile de chaque espace de noms affiche le nombre d'étiquettes, l'état de santé de **Istio Config**, le nombre d'étiquettes, l'état de santé de **Applications** et l'état de santé de **Traffic** pour l'espace de noms. Si vous validez l'installation de la console et que les espaces de noms n'ont pas encore été ajoutés au maillage, il se peut qu'il n'y ait pas d'autres données à afficher que **istio-system**.

6. Kiali dispose de quatre tableaux de bord spécifiques à l'espace de noms dans lequel le plan de contrôle Service Mesh est installé. Pour afficher ces tableaux de bord, cliquez sur le menu **Options** sur la tuile correspondant à l'espace de noms du plan de contrôle, par exemple **istio-system**, et sélectionnez l'une des options suivantes :

- Istio Mesh Dashboard
- Istio Control Plane Dashboard
- Istio Performance Dashboard
- Istio Wasm Exetension Dashboard

Figure 1.2. Tableau de bord Grafana Istio Control Plane

Kiali installe également deux tableaux de bord Grafana supplémentaires, disponibles sur la page Grafana **Home**:

- Istio Workload Dashboard
- Istio Service Dashboard

7. Pour afficher les nœuds du plan de contrôle Service Mesh, cliquez sur la page **Graph**, sélectionnez dans le menu le site **Namespace** où vous avez installé le site **ServiceMeshControlPlane**, par exemple **istio-system**.
   a. Si nécessaire, cliquez sur **Display idle nodes**.
   b. Pour en savoir plus sur la page **Graph**, cliquez sur le lien **Graph tour**.
   c. Pour visualiser la topologie du maillage, sélectionnez un ou plusieurs espaces de noms supplémentaires dans la liste Service Mesh Member du menu **Namespace**.
8. Pour afficher la liste des applications dans l'espace de noms **istio-system**, cliquez sur la page **Applications**. Kiali affiche l'état de santé des applications.
   a. Passez votre souris sur l'icône d'information pour afficher toute information supplémentaire notée dans la colonne **Details**.
9. Pour afficher la liste des charges de travail dans l'espace de noms **istio-system**, cliquez sur la page **Workloads**. Kiali affiche l'état de santé des charges de travail.
   a. Passez votre souris sur l'icône d'information pour afficher toute information supplémentaire notée dans la colonne **Details**.
10. Pour voir la liste des services de l'espace de noms **istio-system**, cliquez sur la page **Services**. Kiali affiche l'état de santé des services et des configurations.

a. Passez votre souris sur l'icône d'information pour afficher toute information supplémentaire notée dans la colonne **Details**.

11. Pour afficher une liste des objets Istio Configuration dans l'espace de noms **istio-system**, cliquez sur la page **Istio Config**. Kiali affiche l'état de la configuration.

    a. S'il y a des erreurs de configuration, cliquez sur la ligne et Kiali ouvre le fichier de configuration avec l'erreur en surbrillance.

1.8.8. Installation sur Red Hat OpenShift Service on AWS (ROSA)

À partir de la version 2.2, Red Hat OpenShift Service Mesh prend en charge l'installation sur Red Hat OpenShift Service on AWS (ROSA). Cette section documente les exigences supplémentaires lors de l'installation de Service Mesh sur cette plateforme.

1.8.8.1. Lieu d'installation

Vous devez créer un nouvel espace de noms, par exemple **istio-system**, lors de l'installation de Red Hat OpenShift Service Mesh et de la création de l'espace de noms **ServiceMeshControlPlane**.

1.8.8.2. Configuration requise du plan de contrôle Service Mesh

La configuration par défaut dans le fichier **ServiceMeshControlPlane** ne fonctionne pas sur un cluster ROSA. Vous devez modifier le SMCP par défaut et définir **spec.security.identity.type=ThirdParty** lors de l'installation sur Red Hat OpenShift Service on AWS.

Exemple de ressource ServiceMeshControlPlane pour ROSA

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.3
  security:
    identity:
      type: ThirdParty  #required setting for ROSA
  tracing:
    type: Jaeger
    sampling: 10000
  policy:
    type: Istiod
  addons:
    grafana:
      enabled: true
    jaeger:
      install:
        storage:
          type: Memory
    kiali:
      enabled: true
    prometheus:
```

```
    enabled: true
telemetry:
  type: Istiod
```

1.8.8.3. Restrictions sur la configuration du Kiali

Red Hat OpenShift Service sur AWS impose des restrictions supplémentaires sur l'endroit où vous pouvez créer des ressources et ne vous permet pas de créer la ressource Kiali dans un espace de noms géré par Red Hat.

Cela signifie que les paramètres communs suivants pour **spec.deployment.accessible_namespaces** ne sont pas autorisés dans un cluster ROSA :

- **['**']** (tous les espaces nominatifs)
- **default**
- **codeready-***
- **openshift-***
- **redhat-***

Le message d'erreur de validation fournit une liste complète de tous les espaces de noms restreints.

Exemple de ressource Kiali pour ROSA

```
apiVersion: kiali.io/v1alpha1
kind: Kiali
metadata:
  name: kiali
  namespace: istio-system
spec:
  auth:
    strategy: openshift
  deployment:
    accessible_namespaces:   #restricted setting for ROSA
      - istio-system
    image_pull_policy: ''
    ingress_enabled: true
    namespace: istio-system
```

1.8.9. Ressources supplémentaires

Red Hat OpenShift Service Mesh prend en charge plusieurs plans de contrôle indépendants au sein du cluster. Vous pouvez créer des configurations réutilisables avec les profils **ServiceMeshControlPlane**. Pour plus d'informations, voir Créer des profils de plan de contrôle .

1.8.10. Prochaines étapes

- Créez une ressource **ServiceMeshMemberRoll** pour spécifier les espaces de noms associés au maillage de services. Pour plus d'informations, voir Ajouter des services à un maillage de services.

1.9. AJOUTER DES SERVICES À UN MAILLAGE DE SERVICES

Après avoir installé la ressource Operators and **ServiceMeshControlPlane**, ajoutez des applications, des charges de travail ou des services à votre maillage en créant une ressource **ServiceMeshMemberRoll** et en spécifiant les espaces de noms dans lesquels votre contenu est situé. Si vous disposez déjà d'une application, d'une charge de travail ou d'un service à ajouter à une ressource **ServiceMeshMemberRoll**, procédez comme suit. Ou, pour installer un exemple d'application appelé Bookinfo et l'ajouter à une ressource **ServiceMeshMemberRoll**, passez au tutoriel d'installation de l'exemple d'application Bookinfo pour voir comment une application fonctionne dans Red Hat OpenShift Service Mesh.

Les éléments énumérés dans la ressource **ServiceMeshMemberRoll** sont les applications et les flux de travail gérés par la ressource **ServiceMeshControlPlane**. Le plan de contrôle, qui comprend les opérateurs de maillage de services, Istiod et **ServiceMeshControlPlane**, et le plan de données, qui comprend les applications et le proxy Envoy, doivent se trouver dans des espaces de noms distincts.

NOTE

Après avoir ajouté l'espace de noms à **ServiceMeshMemberRoll**, l'accès aux services ou aux pods de cet espace de noms ne sera pas accessible aux appelants en dehors du maillage de services.

1.9.1. Création du rouleau de membres Red Hat OpenShift Service Mesh

Le site **ServiceMeshMemberRoll** dresse la liste des projets qui appartiennent au plan de contrôle du Service Mesh. Seuls les projets listés dans le site **ServiceMeshMemberRoll** sont affectés par le plan de contrôle. Un projet n'appartient pas à un maillage de services tant que vous ne l'avez pas ajouté au rôle de membre pour un déploiement particulier du plan de contrôle.

Vous devez créer une ressource **ServiceMeshMemberRoll** nommée **default** dans le même projet que **ServiceMeshControlPlane**, par exemple **istio-system**.

1.9.1.1. Création de la liste des membres à partir de la console web

Vous pouvez ajouter un ou plusieurs projets au rouleau de membres Service Mesh à partir de la console Web. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

Conditions préalables

- Un opérateur Red Hat OpenShift Service Mesh installé et vérifié.
- Liste des projets existants à ajouter au maillage des services.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Si vous n'avez pas encore de services pour votre maillage, ou si vous partez de zéro, créez un projet pour vos applications. Ce projet doit être différent de celui où vous avez installé le plan de contrôle Service Mesh.
   a. Naviguez jusqu'à **Home → Projects**.
   b. Saisissez un nom dans le champ **Name**.

c. Cliquez sur **Create**.

3. Naviguez jusqu'à **Operators → Installed Operators**.
4. Cliquez sur le menu **Project** et choisissez dans la liste le projet dans lequel votre ressource **ServiceMeshControlPlane** est déployée, par exemple **istio-system**.
5. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh.
6. Cliquez sur l'onglet **Istio Service Mesh Member Roll**
7. Cliquez sur **Create ServiceMeshMemberRoll**
8. Cliquez sur **Members**, puis saisissez le nom de votre projet dans le champ **Value**. Vous pouvez ajouter autant de projets que vous le souhaitez, mais un projet ne peut appartenir qu'à la ressource **one ServiceMeshMemberRoll** .
9. Cliquez sur **Create**.

1.9.1.2. Création du rouleau de membres à partir de l'interface de ligne de commande

Vous pouvez ajouter un projet au site **ServiceMeshMemberRoll** à partir de la ligne de commande.

Conditions préalables

- Un opérateur Red Hat OpenShift Service Mesh installé et vérifié.
- Liste des projets à ajouter au maillage des services.
- Accès à la CLI OpenShift (**oc**).

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform.

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. Si vous n'avez pas encore de services pour votre maillage, ou si vous partez de zéro, créez un projet pour vos applications. Ce projet doit être différent de celui où vous avez installé le plan de contrôle Service Mesh.

```
oc new-project <your-project> $ oc new-project <your-project>
```

3. Pour ajouter vos projets en tant que membres, modifiez l'exemple YAML suivant. Vous pouvez ajouter autant de projets que vous le souhaitez, mais un projet ne peut appartenir qu'à la ressource **one ServiceMeshMemberRoll** . Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

Exemple 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
```

4. Exécutez la commande suivante pour télécharger et créer la ressource **ServiceMeshMemberRoll** dans l'espace de noms **istio-system**.

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

5. Exécutez la commande suivante pour vérifier que le site **ServiceMeshMemberRoll** a été créé avec succès.

```
$ oc get smmr -n istio-system default
```

L'installation est terminée avec succès lorsque la colonne **STATUS** est **Configured**.

1.9.2. Ajouter ou supprimer des projets du maillage de services

Vous pouvez ajouter ou supprimer des projets d'une ressource Service Mesh **ServiceMeshMemberRoll** existante à l'aide de la console Web.

- Vous pouvez ajouter autant de projets que vous le souhaitez, mais un projet ne peut appartenir qu'à la ressource **one ServiceMeshMemberRoll** .
- La ressource **ServiceMeshMemberRoll** est supprimée lorsque la ressource **ServiceMeshControlPlane** correspondante est supprimée.

1.9.2.1. Ajouter ou supprimer des projets de la liste des membres à l'aide de la console web

Conditions préalables

- Un opérateur Red Hat OpenShift Service Mesh installé et vérifié.
- Une ressource existante **ServiceMeshMemberRoll**.
- Nom du projet contenant la ressource **ServiceMeshMemberRoll**.
- Noms des projets que vous souhaitez ajouter ou supprimer du maillage.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Naviguez jusqu'à **Operators → Installed Operators**.
3. Cliquez sur le menu **Project** et choisissez dans la liste le projet dans lequel votre ressource **ServiceMeshControlPlane** est déployée, par exemple **istio-system**.
4. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh.
5. Cliquez sur l'onglet **Istio Service Mesh Member Roll**
6. Cliquez sur le lien **default**.

7. Cliquez sur l'onglet YAML.
8. Modifiez le YAML pour ajouter ou supprimer des projets en tant que membres. Vous pouvez ajouter autant de projets que vous le souhaitez, mais un projet ne peut appartenir qu'à la ressource **one ServiceMeshMemberRoll** .
9. Cliquez sur **Save**.
10. Cliquez sur **Reload**.

1.9.2.2. Ajouter ou supprimer des projets de la liste des membres à l'aide de l'interface CLI

Vous pouvez modifier un rouleau de membres Service Mesh existant à l'aide de la ligne de commande.

Conditions préalables

- Un opérateur Red Hat OpenShift Service Mesh installé et vérifié.
- Une ressource existante **ServiceMeshMemberRoll**.
- Nom du projet contenant la ressource **ServiceMeshMemberRoll**.
- Noms des projets que vous souhaitez ajouter ou supprimer du maillage.
- Accès à la CLI OpenShift (**oc**).

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform.
2. Modifier la ressource **ServiceMeshMemberRoll**.

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

3. Modifiez le YAML pour ajouter ou supprimer des projets en tant que membres. Vous pouvez ajouter autant de projets que vous le souhaitez, mais un projet ne peut appartenir qu'à la ressource **one ServiceMeshMemberRoll** .

   Exemple servicemeshmemberroll-default.yaml

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

1.9.3. Exemple d'application Bookinfo

L'application d'exemple Bookinfo vous permet de tester votre installation Red Hat OpenShift Service Mesh 2.3.3 sur OpenShift Container Platform.

L'application Bookinfo affiche des informations sur un livre, à l'instar d'une entrée de catalogue d'une librairie en ligne. L'application affiche une page décrivant le livre, des détails sur le livre (ISBN, nombre de pages et autres informations) et des critiques du livre.

L'application Bookinfo se compose de ces microservices :

- Le microservice **productpage** appelle les microservices **details** et **reviews** pour remplir la page.
- Le microservice **details** contient des informations sur les livres.
- Le microservice **reviews** contient des critiques de livres. Il appelle également le microservice **ratings**.
- Le microservice **ratings** contient des informations sur le classement des livres qui accompagnent une critique de livre.

Il existe trois versions du microservice de révision :

- La version v1 ne fait pas appel au service **ratings**.
- La version v2 appelle le service **ratings** et affiche chaque évaluation sous la forme d'une à cinq étoiles noires.
- La version v3 appelle le service **ratings** et affiche chaque évaluation sous la forme d'une à cinq étoiles rouges.

1.9.3.1. Installation de l'application Bookinfo

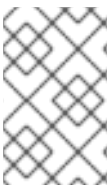
Ce tutoriel vous explique comment créer un exemple d'application en créant un projet, en déployant l'application Bookinfo dans ce projet et en visualisant l'application en cours d'exécution dans Service Mesh.

Prérequis :

- OpenShift Container Platform 4.1 ou supérieur installé.
- Red Hat OpenShift Service Mesh 2.3.3 installé.
- Accès à la CLI OpenShift (**oc**).
- Un compte avec le rôle **cluster-admin**.

NOTE

L'application d'exemple Bookinfo ne peut pas être installée sur IBM zSystems et IBM Power.



NOTE

Les commandes de cette section supposent que le projet de plan de contrôle Service Mesh est **istio-system**. Si vous avez installé le plan de contrôle dans un autre espace de noms, modifiez chaque commande avant de l'exécuter.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur disposant des droits cluster-admin. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.
2. Cliquez sur **Home → Projects**.
3. Cliquez sur **Create Project**.
4. Saisissez **info** comme **Project Name**, saisissez **Display Name**, et saisissez **Description**, puis cliquez sur **Create**.
   - Vous pouvez également exécuter cette commande à partir de l'interface de gestion pour créer le projet **info**.

     ```
     $ oc new-project info
     ```
5. Cliquez sur **Operators → Installed Operators**.
6. Cliquez sur le menu **Project** et utilisez l'espace de noms du plan de contrôle Service Mesh. Dans cet exemple, utilisez **istio-system**.
7. Cliquez sur l'opérateur **Red Hat OpenShift Service Mesh**.
8. Cliquez sur l'onglet **Istio Service Mesh Member Roll**
   a. Si vous avez déjà créé un rouleau Istio Service Mesh Member, cliquez sur son nom, puis sur l'onglet YAML pour ouvrir l'éditeur YAML.
   b. Si vous n'avez pas créé de site **ServiceMeshMemberRoll**, cliquez sur **Create ServiceMeshMemberRoll**.
9. Cliquez sur **Members**, puis saisissez le nom de votre projet dans le champ **Value**.
10. Cliquez sur **Create** pour enregistrer la liste actualisée des membres du Service Mesh.
    a. Vous pouvez également enregistrer l'exemple suivant dans un fichier YAML.

       Bookinfo ServiceMeshMemberRoll exemple servicemeshmemberroll-default.yaml

       ```
       apiVersion: maistra.io/v1
       kind: ServiceMeshMemberRoll
       metadata:
         name: default
       spec:
         members:
         - info
       ```
    b. Exécutez la commande suivante pour télécharger ce fichier et créer la ressource **ServiceMeshMemberRoll** dans l'espace de noms **istio-system**. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

       ```
       $ oc create -n istio-system -f servicemeshmemberroll-default.yaml
       ```
11. Exécutez la commande suivante pour vérifier que le site **ServiceMeshMemberRoll** a été créé avec succès.

```
$ oc get smmr -n istio-system -o wide
```

L'installation est terminée avec succès lorsque la colonne **STATUS** est **Configured**.

```
NAME      READY   STATUS       AGE   MEMBERS
default   1/1     Configured   70s   ["info"]
```

12. Depuis le CLI, déployez l'application Bookinfo dans le projet `*info*` en appliquant le fichier **bookinfo.yaml**:

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

Vous devriez obtenir un résultat similaire à celui qui suit :

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

13. Créez la passerelle d'entrée en appliquant le fichier **info-gateway.yaml**:

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

Vous devriez obtenir un résultat similaire à celui qui suit :

```
gateway.networking.istio.io/info-gateway created
virtualservice.networking.istio.io/info created
```

14. Définir la valeur du paramètre **GATEWAY_URL**:

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

1.9.3.2. Ajout de règles de destination par défaut

Avant de pouvoir utiliser l'application Bookinfo, vous devez d'abord ajouter des règles de destination par défaut. Il existe deux fichiers YAML préconfigurés, selon que vous avez activé ou non l'authentification mutuelle de la couche transport (TLS).

Procédure

1. Pour ajouter des règles de destination, exécutez l'une des commandes suivantes :

   - Si vous n'avez pas activé TLS mutuel :

     ```
     $ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.3/samples/bookinfo/networking/destination-rule-all.yaml
     ```

   - Si vous avez activé TLS mutuel :

     ```
     $ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.3/samples/bookinfo/networking/destination-rule-all-mtls.yaml
     ```

     Vous devriez obtenir un résultat similaire à celui qui suit :

     ```
     destinationrule.networking.istio.io/productpage created
     destinationrule.networking.istio.io/reviews created
     destinationrule.networking.istio.io/ratings created
     destinationrule.networking.istio.io/details created
     ```

1.9.3.3. Vérification de l'installation de Bookinfo

Pour confirmer que l'exemple d'application Bookinfo a été déployé avec succès, effectuez les étapes suivantes.

Conditions préalables

- Red Hat OpenShift Service Mesh installé.
- Suivez les étapes d'installation de l'application d'exemple Bookinfo.

Procédure à partir de l'interface de programmation

1. Connectez-vous au CLI de OpenShift Container Platform.
2. Vérifiez que tous les pods sont prêts à l'aide de cette commande :

   ```
   $ oc get pods -n info
   ```

   Tous les pods devraient avoir un statut de **Running**. Vous devriez voir une sortie similaire à la suivante :

   ```
   NAME                          READY   STATUS    RESTARTS   AGE
   details-v1-55b869668-jh7hb       2/2     Running   0          12m
   productpage-v1-6fc77ff794-nsl8r   2/2     Running   0          12m
   ratings-v1-7d7d8d8b56-55scn      2/2     Running   0          12m
   reviews-v1-868597db96-bdxgq       2/2     Running   0          12m
   reviews-v2-5b64f47978-cvssp      2/2     Running   0          12m
   reviews-v3-6dfd49b55b-vcwpf      2/2     Running   0          12m
   ```

3. Exécutez la commande suivante pour récupérer l'URL de la page produit :

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

4. Copiez et collez le résultat dans un navigateur web pour vérifier que la page du produit Bookinfo est déployée.

Procédure à partir de la console web Kiali

1. Obtenir l'adresse de la console web de Kiali.
   a. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur avec les droits **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.
   b. Naviguez jusqu'à **Networking → Routes**.
   c. Sur la page **Routes**, sélectionnez le projet de plan de contrôle Service Mesh, par exemple **istio-system**, dans le menu **Namespace**.
   La colonne **Location** affiche l'adresse liée à chaque itinéraire.
   d. Cliquez sur le lien dans la colonne **Location** pour Kiali.
   e. Cliquez sur **Log In With OpenShift** L'écran Kiali **Overview** présente des tuiles pour chaque espace de noms de projets.
2. Dans Kiali, cliquez sur **Graph**.
3. Sélectionnez info dans la liste **Namespace** et App graph dans la liste **Graph Type**.
4. Cliquez sur **Display idle nodes** dans le menu **Display**.
   Cette fonction affiche les nœuds qui sont définis mais qui n'ont pas reçu ou envoyé de demandes. Elle permet de confirmer qu'une application est correctement définie, mais qu'aucun trafic de demande n'a été signalé.

   - Utilisez le menu **Duration** pour augmenter la période de temps afin de vous assurer que le trafic plus ancien est capturé.
   - Utilisez le menu **Refresh Rate** pour actualiser le trafic plus ou moins souvent, ou pas du tout.
5. Cliquez sur **Services**, **Workloads** ou **Istio Config** pour afficher la liste des composants d'information et confirmer qu'ils sont sains.

1.9.3.4. Suppression de l'application Bookinfo

Suivez les étapes suivantes pour supprimer l'application Bookinfo.

Conditions préalables

- OpenShift Container Platform 4.1 ou supérieur installé.
- Red Hat OpenShift Service Mesh 2.3.3 installé.

- Accès à la CLI OpenShift (**oc**).

1.9.3.4.1. Supprimer le projet Bookinfo

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Cliquez sur **Home → Projects**.
3. Cliquez sur le menu **info** puis cliquez sur **Delete Project**.
4. Tapez **info** dans la boîte de dialogue de confirmation, puis cliquez sur **Delete**.
   - Vous pouvez également exécuter cette commande à l'aide du CLI pour créer le projet **info**.

     ```
     $ oc delete project info
     ```

1.9.3.4.2. Retirer le projet Bookinfo de la liste des membres du Service Mesh

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Cliquez sur **Operators → Installed Operators**.
3. Cliquez sur le menu **Project** et choisissez **istio-system** dans la liste.
4. Cliquez sur le lien **Istio Service Mesh Member Roll**sous **Provided APIS** pour l'opérateur **Red Hat OpenShift Service Mesh**.
5. Cliquez sur le menu **ServiceMeshMemberRoll** et sélectionnez **Edit Service Mesh Member Roll**.
6. Modifiez le Service Mesh Member Roll YAML par défaut et supprimez **info** de la liste **members**.
   - Vous pouvez également exécuter cette commande à l'aide du CLI pour supprimer le projet **info** du projet **ServiceMeshMemberRoll**. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

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

7. Cliquez sur **Save** pour mettre à jour la liste des membres de Service Mesh.

1.9.4. Prochaines étapes

- Pour poursuivre le processus d'installation, vous devez activer l'injection de sidecar.

1.10. ACTIVATION DE L'INJECTION DU SIDE-CAR

Après avoir ajouté les espaces de noms qui contiennent vos services à votre maillage, l'étape suivante consiste à activer l'injection automatique de sidecars dans la ressource Deployment de votre application. Vous devez activer l'injection automatique de sidecars pour chaque déploiement.

Si vous avez installé l'exemple d'application Bookinfo, l'application a été déployée et les sidecars ont été injectées dans le cadre de la procédure d'installation. Si vous utilisez votre propre projet et service, déployez vos applications sur OpenShift Container Platform.

Pour plus d'informations, voir la documentation OpenShift Container Platform, Understanding Deployment and DeploymentConfig objects.

1.10.1. Conditions préalables

- Services déployés dans le maillage , par exemple l'exemple d'application Bookinfo.
- Un fichier de ressources de déploiement.

1.10.2. Activation de l'injection automatique du side-car

Lors du déploiement d'une application, vous devez opter pour l'injection en configurant l'annotation **sidecar.istio.io/inject** dans **spec.template.metadata.annotations** vers **true** dans l'objet **deployment**. L'opt-in garantit que l'injection de sidecar n'interfère pas avec d'autres fonctionnalités d'OpenShift Container Platform telles que les builder pods utilisés par de nombreux frameworks au sein de l'écosystème d'OpenShift Container Platform.

Conditions préalables

- Identifiez les espaces de noms qui font partie de votre maillage de services et les déploiements qui ont besoin d'une injection automatique de sidecar.

Procédure

1. Pour trouver vos déploiements, utilisez la commande **oc get**.

   ```
   $ oc get deployment -n <namespace>
   ```

   Par exemple, pour afficher le fichier de déploiement du microservice "ratings-v1" dans l'espace de noms **info**, utilisez la commande suivante pour afficher la ressource au format YAML.

   ```
   oc get deployment -n info ratings-v1 -o yaml
   ```

2. Ouvrez le fichier YAML de configuration du déploiement de l'application dans un éditeur.

3. Ajoutez **spec.template.metadata.annotations.sidecar.istio/inject** à votre Deployment YAML et définissez **sidecar.istio.io/inject** à **true** comme indiqué dans l'exemple suivant.

   Exemple d'extrait de info deployment-ratings-v1.yaml

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: ratings-v1
     namespace: info
     labels:
   ```

```
      app: ratings
      version: v1
  spec:
    template:
      metadata:
        annotations:
          sidecar.istio.io/inject: 'true'
```

4. Enregistrer le fichier de configuration du déploiement.
5. Ajoutez le fichier au projet qui contient votre application.

```
oc apply -n <namespace> -f deployment.yaml
```

Dans cet exemple, **info** est le nom du projet qui contient l'application **ratings-v1** et **deployment-ratings-v1.yaml** est le fichier que vous avez modifié.

```
$ oc apply -n info -f deployment-ratings-v1.yaml
```

6. Pour vérifier que la ressource a été téléchargée avec succès, exécutez la commande suivante.

```
$ oc get deployment -n <namespace> <deploymentName> -o yaml
```

Par exemple,

```
$ oc get deployment -n info ratings-v1 -o yaml
```

1.10.3. Validation de l'injection du sidecar

La console Kiali offre plusieurs moyens de vérifier si vos applications, services et charges de travail disposent ou non d'un proxy sidecar.

Figure 1.3. Insigne de side-car manquant

La page **Graph** affiche un badge de nœud indiquant un **Missing Sidecar** sur les graphiques suivants :

- Graphique de l'application
- Graphique de l'application versionnée
- Graphique de la charge de travail

Figure 1.4. Icône de side-car manquante

La page **Applications** affiche une icône **Missing Sidecar** dans la colonne **Details** pour toutes les applications d'un espace de noms qui n'ont pas de sidecar.

La page **Workloads** affiche une icône **Missing Sidecar** dans la colonne **Details** pour toutes les applications d'un espace de noms qui n'ont pas de sidecar.

La page **Services** affiche une icône **Missing Sidecar** dans la colonne **Details** pour toutes les applications d'un espace de noms qui n'ont pas de sidecar. Lorsqu'il existe plusieurs versions d'un service, la page **Service Details** permet d'afficher les icônes **Missing Sidecar**.

La page **Workload Details** comporte un onglet spécial **Logs** unifié qui vous permet d'afficher et de corréler les journaux d'application et de proxy. Vous pouvez consulter les journaux Envoy comme un autre moyen de valider l'injection de sidecar pour les charges de travail de vos applications.

La page **Workload Details** comporte également un onglet **Envoy** pour toute charge de travail qui est un proxy Envoy ou qui a été injectée avec un proxy Envoy. Cet onglet affiche un tableau de bord Envoy intégré qui comprend des sous-onglets pour **Clusters**, **Listeners**, **Routes**, **Bootstrap**, **Config**, et **Metrics**.

Pour plus d'informations sur l'activation des journaux d'accès à Envoy, voir la section Dépannage.

Pour plus d'informations sur l'affichage des journaux Envoy, voir Affichage des journaux dans la console Kiali

1.10.4. Définition de variables d'environnement de proxy par le biais d'annotations

La configuration des mandataires Envoy sidecar est gérée par le site **ServiceMeshControlPlane**.

Vous pouvez définir des variables d'environnement pour le proxy sidecar des applications en ajoutant des annotations pod au déploiement dans le fichier **injection-template.yaml**. Les variables d'environnement sont injectées dans le sidecar.

Exemple injection-template.yaml

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

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

AVERTISSEMENT

Vous ne devez jamais inclure les étiquettes et les annotations de **maistra.io/** lorsque vous créez vos propres ressources personnalisées. Ces étiquettes et annotations indiquent que les ressources sont générées et gérées par l'opérateur. Si vous copiez le contenu d'une ressource générée par l'opérateur lorsque vous créez vos propres ressources, n'incluez pas d'étiquettes ou d'annotations commençant par **maistra.io/**. Les ressources qui incluent ces étiquettes ou annotations seront écrasées ou supprimées par l'opérateur lors de la prochaine réconciliation.

1.10.5. Mise à jour des mandataires sidecar

Afin de mettre à jour la configuration des proxies sidecar, l'administrateur de l'application doit redémarrer les pods d'application.

Si votre déploiement utilise l'injection automatique de sidecar, vous pouvez mettre à jour le modèle de pod dans le déploiement en ajoutant ou en modifiant une annotation. Exécutez la commande suivante pour redéployer les pods :

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

Si votre déploiement n'utilise pas l'injection automatique de sidecars, vous devez mettre à jour manuellement les sidecars en modifiant l'image du conteneur de sidecars spécifiée dans le déploiement ou le pod, puis redémarrer les pods.

1.10.6. Prochaines étapes

Configurez les fonctionnalités de Red Hat OpenShift Service Mesh pour votre environnement.

- Sécurité
- Gestion du trafic
- Mesures, journaux et traces

1.11. MISE À NIVEAU DU MAILLAGE DE SERVICES

Pour accéder aux fonctionnalités les plus récentes de Red Hat OpenShift Service Mesh, mettez à niveau vers la version actuelle, 2.3.3.

1.11.1. Comprendre les versions

Red Hat utilise la version sémantique pour les versions de ses produits. La version sémantique est un numéro à trois composants au format X.Y.Z, où :

- X correspond à une version majeure. Les versions majeures indiquent généralement une sorte de changement radical : changements architecturaux, changements d'API, changements de schéma et autres mises à jour majeures similaires.

- Y correspond à une version mineure. Les versions mineures contiennent de nouvelles caractéristiques et fonctionnalités tout en maintenant la compatibilité ascendante.

- Z correspond à une version de correctif (également connue sous le nom de "z-stream release"). Les versions de correctifs sont utilisées pour remédier aux vulnérabilités et expositions communes (CVE) et pour publier des corrections de bogues. Les nouvelles caractéristiques et fonctionnalités ne sont généralement pas publiées dans le cadre d'une version Patch.

1.11.1.1. Comment les versions affectent les mises à jour de Service Mesh

La procédure de mise à jour varie en fonction de la version de la mise à jour que vous effectuez.

- **Patch updates** - Les mises à jour des correctifs sont gérées par le gestionnaire du cycle de vie des opérateurs (OLM) ; elles se produisent automatiquement lorsque vous mettez à jour vos opérateurs.

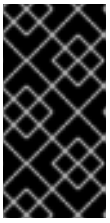
- **Minor upgrades** - Les mises à niveau mineures nécessitent à la fois une mise à jour vers la version la plus récente de Red Hat OpenShift Service Mesh Operator et une modification manuelle de la valeur **spec.version** dans vos ressources **ServiceMeshControlPlane**.

- **Major upgrades** - Les mises à niveau majeures nécessitent à la fois une mise à jour vers la version la plus récente de Red Hat OpenShift Service Mesh Operator et une modification manuelle de la valeur **spec.version** dans vos ressources **ServiceMeshControlPlane**. Étant donné que les mises à niveau majeures peuvent contenir des changements qui ne sont pas rétrocompatibles, des modifications manuelles supplémentaires peuvent être nécessaires.

1.11.1.2. Comprendre les versions de Service Mesh

Afin de comprendre quelle version de Red Hat OpenShift Service Mesh vous avez déployée sur votre système, vous devez comprendre comment chacune des versions des composants est gérée.

- **Operator** version - La version la plus récente de l'opérateur est 2.3.3. Le numéro de version de l'opérateur indique uniquement la version de l'opérateur actuellement installé. Étant donné que l'Opérateur Red Hat OpenShift Service Mesh prend en charge plusieurs versions du plan de contrôle Service Mesh, la version de l'Opérateur ne détermine pas la version de vos ressources **ServiceMeshControlPlane** déployées.



IMPORTANT

La mise à niveau vers la dernière version de l'opérateur applique automatiquement les mises à jour des correctifs, mais ne met pas automatiquement à niveau votre plan de contrôle Service Mesh vers la dernière version mineure.

- **ServiceMeshControlPlane** version - La version **ServiceMeshControlPlane** détermine la version de Red Hat OpenShift Service Mesh que vous utilisez. La valeur du champ **spec.version** dans la ressource **ServiceMeshControlPlane** contrôle l'architecture et les paramètres de configuration utilisés pour installer et déployer Red Hat OpenShift Service Mesh. Lorsque vous créez le plan de contrôle Service Mesh, vous pouvez définir la version de deux manières :

- Pour configurer la vue formulaire, sélectionnez la version dans le menu **Control Plane Version**.
- Pour configurer la vue YAML, définissez la valeur de **spec.version** dans le fichier YAML.

Operator Lifecycle Manager (OLM) ne gère pas les mises à niveau du plan de contrôle de Service Mesh, de sorte que le numéro de version de votre opérateur et de **ServiceMeshControlPlane** (SMCP) peut ne pas correspondre, à moins que vous n'ayez mis à niveau manuellement votre SMCP.

1.11.2. Considérations relatives à la mise à niveau

L'étiquette ou l'annotation **maistra.io/** ne doit pas être utilisée sur une ressource personnalisée créée par l'utilisateur, car elle indique que la ressource a été générée et doit être gérée par le Red Hat OpenShift Service Mesh Operator.

AVERTISSEMENT

Pendant la mise à niveau, l'opérateur apporte des modifications, y compris la suppression ou le remplacement de fichiers, aux ressources qui comprennent les étiquettes ou annotations suivantes indiquant que la ressource est gérée par l'opérateur.

Avant de procéder à la mise à niveau, vérifiez si des ressources personnalisées créées par l'utilisateur comportent les étiquettes ou les annotations suivantes :

- **maistra.io/** ET l'étiquette **app.kubernetes.io/managed-by** définie sur **maistra-istio-operator** (Red Hat OpenShift Service Mesh)
- **kiali.io/** (Kiali)
- **jaegertracing.io/** (Plate-forme de traçage distribuée Red Hat OpenShift)
- **logging.openshift.io/** (Red Hat Elasticsearch)

Avant la mise à niveau, vérifiez que les ressources personnalisées créées par l'utilisateur ne comportent pas d'étiquettes ou d'annotations indiquant qu'elles sont gérées par l'opérateur. Supprimez l'étiquette ou l'annotation des ressources personnalisées que vous ne souhaitez pas voir gérées par l'opérateur.

Lors du passage à la version 2.0, l'opérateur ne supprime que les ressources portant ces étiquettes dans le même espace de noms que le SMCP.

Lors du passage à la version 2.1, l'opérateur supprime les ressources portant ces étiquettes dans tous les espaces de noms.

1.11.2.1. Problèmes connus pouvant affecter la mise à jour

Les problèmes connus qui peuvent affecter votre mise à niveau sont les suivants :

- Red Hat OpenShift Service Mesh ne prend pas en charge l'utilisation de la configuration **EnvoyFilter**, sauf lorsqu'elle est explicitement documentée. Ceci est dû à un couplage étroit avec les API Envoy sous-jacentes, ce qui signifie que la compatibilité ascendante ne peut pas

être maintenue. Si vous utilisez des filtres Envoy et que la configuration générée par Istio a changé en raison de la dernière version d'Envoy introduite par la mise à niveau de votre **ServiceMeshControlPlane**, cela peut potentiellement casser tout **EnvoyFilter** que vous avez mis en œuvre.

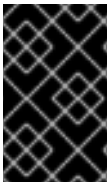
- OSSM-1505 **ServiceMeshExtension** ne fonctionne pas avec OpenShift Container Platform version 4.11. Parce que **ServiceMeshExtension** a été déprécié dans Red Hat OpenShift Service Mesh 2.2, ce problème connu ne sera pas corrigé et vous devez migrer vos extensions vers **WasmPluging**

- OSSM-1396 Si une ressource passerelle contient le paramètre **spec.externalIPs**, au lieu d'être recréée lors de la mise à jour de **ServiceMeshControlPlane**, la passerelle est supprimée et n'est jamais recréée.

- OSSM-1052 Lors de la configuration d'un service **ExternalIP** pour l'ingressgateway dans le plan de contrôle Service Mesh, le service n'est pas créé. Le schéma du SMCP ne contient pas le paramètre du service.
  Solution : Désactiver la création de la passerelle dans la spécification SMCP et gérer le déploiement de la passerelle entièrement manuellement (y compris le service, le rôle et le RoleBinding).

1.11.3. Mise à niveau des opérateurs

Pour que votre Service Mesh bénéficie des derniers correctifs de sécurité, des corrections de bogues et des mises à jour logicielles, vous devez mettre à jour vos opérateurs. Vous lancez les mises à jour des correctifs en mettant à niveau vos opérateurs.



IMPORTANT

La version de l'opérateur ne **not** détermine pas la version de votre Service Mesh. La version de votre plan de contrôle Service Mesh déployé détermine votre version de Service Mesh.

not Étant donné que Red Hat OpenShift Service Mesh Operator prend en charge plusieurs versions du plan de contrôle Service Mesh, la mise à jour de Red Hat OpenShift Service Mesh Operator ne met pas à jour la valeur **spec.version** de votre **ServiceMeshControlPlane** déployé. Notez également que la valeur **spec.version** est un nombre à deux chiffres, par exemple 2.2, et que les mises à jour de correctifs, par exemple 2.2.1, ne sont pas reflétées dans la valeur de la version SMCP.

Operator Lifecycle Manager (OLM) contrôle l'installation, la mise à niveau et le contrôle d'accès basé sur les rôles (RBAC) des opérateurs dans un cluster. L'OLM s'exécute par défaut dans OpenShift Container Platform. OLM recherche les opérateurs disponibles ainsi que les mises à niveau des opérateurs installés.

Le fait que vous deviez ou non prendre des mesures pour mettre à jour vos opérateurs dépend des paramètres que vous avez sélectionnés lors de leur installation. Lorsque vous avez installé chacun de vos opérateurs, vous avez sélectionné un **Update Channel** et un **Approval Strategy**. La combinaison de ces deux paramètres détermine quand et comment vos opérateurs sont mis à jour.

Tableau 1.4. Interaction entre le canal de mise à jour et la stratégie d'approbation

	Canal versionné	\Stabilité ou prévisibilité Chaîne

	Canal versionné	\Stabilité ou prévisibilité Chaîne
Automatic	L'opérateur est automatiquement mis à jour pour les versions mineures et les correctifs de cette version uniquement. Ne met pas automatiquement à jour la version majeure suivante (c'est-à-dire de la version 2.0 à la version 3.0). Une modification manuelle de l'abonnement à l'opérateur est nécessaire pour passer à la version majeure suivante.	Mise à jour automatique d'Operator pour toutes les versions majeures, mineures et correctives.
Manual	Mises à jour manuelles requises pour les versions mineures et les correctifs de la version spécifiée. Une modification manuelle de l'abonnement de l'opérateur est nécessaire pour passer à la version majeure suivante.	Mises à jour manuelles requises pour toutes les versions majeures, mineures et correctives.

Lorsque vous mettez à jour votre opérateur Red Hat OpenShift Service Mesh, le gestionnaire du cycle de vie de l'opérateur (OLM) supprime l'ancien pod de l'opérateur et démarre un nouveau pod. Une fois que le nouveau pod Operator démarre, le processus de réconciliation vérifie le site **ServiceMeshControlPlane** (SMCP), et s'il y a des images de conteneurs mises à jour disponibles pour l'un des composants du plan de contrôle Service Mesh, il remplace ces pods du plan de contrôle Service Mesh par ceux qui utilisent les nouvelles images de conteneurs.

Lorsque vous mettez à niveau les opérateurs de plateforme de traçage distribuée Kiali et Red Hat OpenShift, le processus de réconciliation OLM analyse le cluster et met à niveau les instances gérées vers la version du nouvel opérateur. Par exemple, si vous mettez à jour l'opérateur de plateforme de traçage distribuée Red Hat OpenShift de la version 1.30.2 à la version 1.34.1, l'opérateur recherche les instances en cours d'exécution de la plateforme de traçage distribuée et les met également à niveau vers la version 1.34.1.

Pour rester sur une version de correctif particulière de Red Hat OpenShift Service Mesh, vous devez désactiver les mises à jour automatiques et rester sur cette version spécifique de l'opérateur.

Pour plus d'informations sur la mise à niveau des opérateurs, reportez-vous à la documentation du gestionnaire du cycle de vie des opérateurs .

1.11.4. Mise à niveau du plan de contrôle

Vous devez mettre à jour manuellement le plan de contrôle pour les versions mineures et majeures. Le projet communautaire Istio recommande des mises à niveau canary, Red Hat OpenShift Service Mesh ne prend en charge que les mises à niveau in-place. Red Hat OpenShift Service Mesh exige que vous mettiez à niveau chaque version mineure vers la version mineure suivante dans l'ordre. Par exemple, vous devez mettre à niveau la version 2.0 vers la version 2.1, puis vers la version 2.2. Vous ne pouvez pas mettre à jour Red Hat OpenShift Service Mesh 2.0 vers 2.2 directement.

Lorsque vous mettez à niveau le plan de contrôle du service mesh, toutes les ressources gérées par l'opérateur, par exemple les passerelles, sont également mises à niveau.

Bien que vous puissiez déployer plusieurs versions du plan de contrôle dans le même cluster, Red Hat OpenShift Service Mesh ne prend pas en charge les mises à niveau canari du maillage de services. En d'autres termes, vous pouvez avoir différentes ressources SCMP avec différentes valeurs pour **spec.version**, mais elles ne peuvent pas gérer le même maillage.

Pour plus d'informations sur la migration de vos extensions, reportez-vous à la section Migration des ressources ServiceMeshExtension vers WasmPlugin.

1.11.4.1. Changements dans la mise à jour de la version 2.2 à la version 2.3

La mise à niveau du plan de contrôle Service Mesh de la version 2.2 à la version 2.3 introduit les changements de comportement suivants :

- Cette version nécessite l'utilisation de l'API **WasmPlugin**. La prise en charge de l'API **ServiceMeshExtension**, qui était obsolète dans la version 2.2, a été supprimée. Si vous tentez d'effectuer une mise à niveau alors que vous utilisez l'API **ServiceMeshExtension**, la mise à niveau échoue.

1.11.4.2. Changements dans la mise à jour de la version 2.1 à la version 2.2

La mise à niveau du plan de contrôle Service Mesh de la version 2.1 à la version 2.2 introduit les changements de comportement suivants :

- Le DaemonSet **istio-node** est renommé en **istio-cni-node** pour correspondre au nom dans Istio en amont.
- Istio 1.10 a mis à jour Envoy pour envoyer le trafic au conteneur d'application en utilisant **eth0** au lieu de **lo** par défaut.
- Cette version ajoute la prise en charge de l'API **WasmPlugin** et supprime l'API **ServiceMeshExtension**.

1.11.4.3. Changements dans la mise à jour de la version 2.0 à la version 2.1

La mise à niveau du plan de contrôle du Service Mesh de la version 2.0 à la version 2.1 introduit les changements architecturaux et comportementaux suivants.

Modifications de l'architecture

Mixer a été complètement supprimé dans Red Hat OpenShift Service Mesh 2.1. La mise à niveau d'une version Red Hat OpenShift Service Mesh 2.0.x vers 2.1 sera bloquée si Mixer est activé.

Si le message suivant apparaît lors de la mise à niveau de la version 2.0 à la version 2.1, mettez à jour le type **Mixer** existant en type **Istiod** dans la spécification du plan de contrôle existant avant de mettre à jour le champ **.spec.version**:

> An error occurred
> admission webhook smcp.validation.maistra.io denied the request: [support for policy.type "Mixer" and policy.Mixer options have been removed in v2.1, please use another alternative, support for telemetry.type "Mixer" and telemetry.Mixer options have been removed in v2.1, please use another alternative]"

Par exemple :

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  policy:
    type: Istiod
  telemetry:
    type: Istiod
  version: v2.3
```

Changements de comportement

- **AuthorizationPolicy** des mises à jour :
  - Avec le protocole PROXY, si vous utilisez **ipBlocks** et **notIpBlocks** pour spécifier des adresses IP distantes, mettez à jour la configuration pour utiliser **remoteIpBlocks** et **notRemoteIpBlocks** à la place.
  - Ajout de la prise en charge des réclamations JSON Web Token (JWT) imbriquées.
- **EnvoyFilter** changements en cours>
  - Doit être utilisé **typed_config**
  - xDS v2 n'est plus supporté
  - Noms de filtres obsolètes
- Les anciennes versions des serveurs mandataires peuvent signaler des codes d'état 503 lorsqu'ils reçoivent des codes d'état 1xx ou 204 de la part de serveurs mandataires plus récents.

1.11.4.4. Mise à jour du plan de contrôle du Service Mesh

Pour mettre à niveau Red Hat OpenShift Service Mesh, vous devez mettre à jour le champ de version de la ressource Red Hat OpenShift Service Mesh **ServiceMeshControlPlane** v2. Ensuite, une fois qu'elle est configurée et appliquée, redémarrez les pods d'application pour mettre à jour chaque proxy sidecar et sa configuration.

Conditions préalables

- Vous utilisez OpenShift Container Platform 4.9 ou une version ultérieure.
- Vous avez la dernière version de Red Hat OpenShift Service Mesh Operator.

Procédure

1. Passez au projet qui contient votre ressource **ServiceMeshControlPlane**. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

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

2. Vérifiez la validité de la configuration de votre ressource v2 **ServiceMeshControlPlane**.
   a. Exécutez la commande suivante pour afficher votre ressource **ServiceMeshControlPlane** en tant que ressource v2.

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

ASTUCE

Sauvegardez la configuration du plan de contrôle de Service Mesh.

3. Mettez à jour le champ **.spec.version** et appliquez la configuration.
Par exemple :

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.3
```

Au lieu d'utiliser la ligne de commande, vous pouvez également utiliser la console web pour modifier le plan de contrôle Service Mesh. Dans la console web d'OpenShift Container Platform, cliquez sur **Project** et sélectionnez le nom du projet que vous venez de saisir.

a. Cliquez sur **Operators → Installed Operators**.

b. Trouvez votre instance **ServiceMeshControlPlane**.

c. Sélectionnez **YAML view** et mettez à jour le texte du fichier YAML, comme indiqué dans l'exemple précédent.

d. Cliquez sur **Save**.

1.11.4.5. Migration de Red Hat OpenShift Service Mesh de la version 1.1 à la version 2.0

La mise à niveau de la version 1.1 à 2.0 nécessite des étapes manuelles qui migrent vos charges de travail et votre application vers une nouvelle instance de Red Hat OpenShift Service Mesh exécutant la nouvelle version.

Conditions préalables

- Vous devez mettre à niveau vers OpenShift Container Platform 4.7. avant de mettre à niveau vers Red Hat OpenShift Service Mesh 2.0.
- Vous devez avoir l'opérateur Red Hat OpenShift Service Mesh version 2.0. Si vous avez sélectionné le chemin de mise à niveau **automatic**, l'opérateur télécharge automatiquement les dernières informations. Cependant, vous devez suivre certaines étapes pour utiliser les fonctionnalités de Red Hat OpenShift Service Mesh version 2.0.

1.11.4.5.1. Mise à jour de Red Hat OpenShift Service Mesh

Pour mettre à niveau Red Hat OpenShift Service Mesh, vous devez créer une instance de la ressource Red Hat OpenShift Service Mesh **ServiceMeshControlPlane** v2 dans un nouvel espace de noms. Ensuite, une fois qu'elle est configurée, déplacez vos applications de microservices et vos charges de travail de votre ancien maillage vers le nouveau maillage de services.

Procédure

1. Vérifiez que la configuration de votre ressource v1 **ServiceMeshControlPlane** est valide.

   a. Exécutez la commande suivante pour afficher votre ressource **ServiceMeshControlPlane** en tant que ressource v2.

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

   b. Consultez le champ **spec.techPreview.errored.message** dans la sortie pour obtenir des informations sur les champs non valides.

   c. Si votre ressource v1 contient des champs non valides, la ressource n'est pas réconciliée et ne peut pas être éditée en tant que ressource v2. Toutes les mises à jour des champs de la v2 seront remplacées par les paramètres originaux de la v1. Pour corriger les champs non valides, vous pouvez remplacer, corriger ou éditer la version v1 de la ressource. Vous pouvez également supprimer la ressource sans la corriger. Une fois la ressource corrigée, elle peut être réconciliée et vous pouvez modifier ou consulter la version v2 de la ressource.

   d. Pour corriger la ressource en modifiant un fichier, utilisez **oc get** pour récupérer la ressource, modifiez le fichier texte localement et remplacez la ressource par le fichier que vous avez modifié.

   ```
   $ oc get smcp.v1.maistra.io <smcp_name> > smcp-resource.yaml
   #Edit the smcp-resource.yaml file.
   $ oc replace -f smcp-resource.yaml
   ```

   e. Pour corriger la ressource à l'aide d'un correctif, utilisez **oc patch**.

   ```
   $ oc patch smcp.v1.maistra.io <smcp_name> --type json --patch '[{"op" : \N "replace\N",
   \N "path\N":\N"/spec/path/to/bad/setting\N", \N "value\N":\N "corrected-value\N"]"
   ```

   f. Pour corriger la ressource en l'éditant à l'aide d'outils de ligne de commande, utilisez **oc edit**.

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

2. Sauvegardez votre configuration du plan de contrôle Service Mesh. Passez au projet qui contient votre ressource **ServiceMeshControlPlane**. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

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

3. Entrez la commande suivante pour récupérer la configuration actuelle. Votre <smcp_name> est spécifié dans les métadonnées de votre ressource **ServiceMeshControlPlane**, par exemple **basic-install** ou **full-install**.

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

4. Convertissez votre site **ServiceMeshControlPlane** en une version v2 du plan de contrôle qui contient des informations sur votre configuration comme point de départ.

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

5. Créez un projet. Dans le menu Projet de la console OpenShift Container Platform, cliquez sur **New Project** et entrez un nom pour votre projet, **istio-system-upgrade**, par exemple. Vous pouvez également exécuter cette commande à partir de la CLI.

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

6. Mettez à jour le champ **metadata.namespace** de votre v2 **ServiceMeshControlPlane** avec le nouveau nom de votre projet. Dans cet exemple, utilisez **istio-system-upgrade**.

7. Mettez à jour le champ **version** de 1.1 à 2.0 ou supprimez-le dans votre v2 **ServiceMeshControlPlane**.

8. Créez un **ServiceMeshControlPlane** dans le nouvel espace de noms. Sur la ligne de commande, exécutez la commande suivante pour déployer le plan de contrôle avec la version v2 du **ServiceMeshControlPlane** que vous avez récupéré. Dans cet exemple, remplacez `<smcp_name.v2> `par le chemin d'accès à votre fichier.

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

   Vous pouvez également utiliser la console pour créer le plan de contrôle Service Mesh. Dans la console web d'OpenShift Container Platform, cliquez sur **Project**. Sélectionnez ensuite le nom du projet que vous venez de saisir.

   a. Cliquez sur **Operators → Installed Operators**.

   b. Cliquez sur **Create ServiceMeshControlPlane**.

   c. Sélectionnez **YAML view** et collez le texte du fichier YAML que vous avez récupéré dans le champ. Vérifiez que le champ **apiVersion** est défini sur **maistra.io/v2** et modifiez le champ **metadata.namespace** pour utiliser le nouvel espace de noms, par exemple **istio-system-upgrade**.

   d. Cliquez sur **Create**.

1.11.4.5.2. Configuration du ServiceMeshControlPlane 2.0

La ressource **ServiceMeshControlPlane** a été modifiée pour Red Hat OpenShift Service Mesh version 2.0. Après avoir créé une version v2 de la ressource **ServiceMeshControlPlane**, modifiez-la pour profiter des nouvelles fonctionnalités et pour l'adapter à votre déploiement. Tenez compte des changements suivants dans la spécification et le comportement de Red Hat OpenShift Service Mesh 2.0 lorsque vous modifiez votre ressource **ServiceMeshControlPlane**. Vous pouvez également vous référer à la documentation du produit Red Hat OpenShift Service Mesh 2.0 pour obtenir de nouvelles informations sur les fonctionnalités que vous utilisez. La ressource v2 doit être utilisée pour les installations Red Hat OpenShift Service Mesh 2.0.

1.11.4.5.2.1. Modifications de l'architecture

Les unités architecturales utilisées par les versions précédentes ont été remplacées par Istiod. Dans la version 2.0, les composants du plan de contrôle Service Mesh - Mixer, Pilot, Citadel, Galley - et la fonctionnalité de l'injecteur sidecar ont été combinés en un seul composant, Istiod.

Bien que Mixer ne soit plus supporté en tant que composant du plan de contrôle, les plugins de politique et de télémétrie Mixer sont maintenant supportés par des extensions WASM dans Istiod. Mixer peut être activé pour la politique et la télémétrie si vous avez besoin d'intégrer des plugins Mixer hérités.

Secret Discovery Service (SDS) est utilisé pour distribuer des certificats et des clés aux sidecars directement depuis Istiod. Dans Red Hat OpenShift Service Mesh version 1.1, des secrets ont été générés par Citadel, qui ont été utilisés par les proxys pour récupérer les certificats et les clés de leurs clients.

1.11.4.5.2.2. Modifications des annotations

Les annotations suivantes ne sont plus prises en charge dans la version 2.0. Si vous utilisez l'une de ces annotations, vous devez mettre à jour votre charge de travail avant de la transférer vers un plan de contrôle Service Mesh v2.0.

- **sidecar.maistra.io/proxyCPULimit** a été remplacé par **sidecar.istio.io/proxyCPULimit**. Si vous utilisiez les annotations **sidecar.maistra.io** dans vos charges de travail, vous devez modifier ces charges de travail pour utiliser les équivalents **sidecar.istio.io** à la place.
- **sidecar.maistra.io/proxyMemoryLimit** a été remplacé par **sidecar.istio.io/proxyMemoryLimit**
- **sidecar.istio.io/discoveryAddress** n'est plus pris en charge. En outre, l'adresse de découverte par défaut est passée de **pilot.<control_plane_namespace>.svc:15010** (ou du port 15011, si mtls est activé) à **istiod-<smcp_name>.<control_plane_namespace>.svc:15012**.
- The health status port is no longer configurable and is hard-coded to 15021. * If you were defining a custom status port, for example, **status.sidecar.istio.io/port**, you must remove the override before moving the workload to a v2.0 Service Mesh control plane. Readiness checks can still be disabled by setting the status port to **0**.
- Les ressources Kubernetes Secret ne sont plus utilisées pour distribuer les certificats clients pour les sidecars. Les certificats sont désormais distribués par le service SDS d'Istiod. Si vous vous appuyiez sur des secrets montés, ils ne sont plus disponibles pour les charges de travail dans les plans de contrôle Service Mesh v2.0.

1.11.4.5.2.3. Changements de comportement

Certaines fonctionnalités de Red Hat OpenShift Service Mesh 2.0 fonctionnent différemment de celles des versions précédentes.

- Le port de préparation des passerelles est passé de **15020** à **15021**.
- La visibilité de l'hôte cible inclut les ressources VirtualService et ServiceEntry. Elle inclut toutes les restrictions appliquées par le biais des ressources Sidecar.
- La fonction TLS mutuelle automatique est activée par défaut. La communication de proxy à proxy est automatiquement configurée pour utiliser mTLS, indépendamment des politiques globales de PeerAuthentication en place.
- Les connexions sécurisées sont toujours utilisées lorsque les proxies communiquent avec le plan de contrôle du Service Mesh, quel que soit le paramètre **spec.security.controlPlane.mtls**. Le paramètre **spec.security.controlPlane.mtls** n'est utilisé que lors de la configuration des connexions pour la télémétrie ou la politique de Mixer.

1.11.4.5.2.4. Détails de la migration pour les ressources non prises en charge

Politique (authentication.istio.io/v1alpha1)

Les ressources de politique doivent être migrées vers de nouveaux types de ressources à utiliser avec les plans de contrôle Service Mesh v2.0, PeerAuthentication et RequestAuthentication. En fonction de la configuration spécifique de votre ressource de stratégie, il se peut que vous deviez configurer

plusieurs ressources pour obtenir le même effet.

TLS mutuel

L'application mutuelle de TLS est réalisée à l'aide de la ressource **security.istio.io/v1beta1** PeerAuthentication. L'ancien champ **spec.peers.mtls.mode** correspond directement au champ **spec.mtls.mode** de la nouvelle ressource. Les critères de sélection sont passés de la spécification d'un nom de service dans **spec.targets[x].name** à un sélecteur d'étiquettes dans **spec.selector.matchLabels**. Dans PeerAuthentication, les étiquettes doivent correspondre au sélecteur sur les services nommés dans la liste des cibles. Tout paramètre spécifique à un port devra être mappé dans **spec.portLevelMtls**.

Authentification

Les méthodes d'authentification supplémentaires spécifiées dans **spec.origins** doivent être mappées dans une ressource **security.istio.io/v1beta1** RequestAuthentication. **spec.selector.matchLabels** doit être configuré de la même manière que le champ PeerAuthentication. La configuration spécifique aux mandants JWT des éléments **spec.origins.jwt** correspond à des champs similaires dans les éléments **spec.rules**.

- **spec.origins[x].jwt.triggerRules** spécifié dans la politique doit être mappé dans une ou plusieurs ressources **security.istio.io/v1beta1** AuthorizationPolicy. Tout **spec.selector.labels** doit être configuré de manière similaire au même champ sur RequestAuthentication.

- **spec.origins[x].jwt.triggerRules.excludedPaths** doit être mappée dans une AuthorizationPolicy dont le spec.action est défini sur ALLOW, avec **spec.rules[x].to.operation.path** entrées correspondant aux chemins exclus.

- **spec.origins[x].jwt.triggerRules.includedPaths** doit être mappée dans une AuthorizationPolicy distincte dont **spec.action** est défini sur **ALLOW**, avec des entrées **spec.rules[x].to.operation.path** correspondant aux chemins inclus, et des entrées **spec.rules.[x].from.source.requestPrincipals** qui s'alignent sur **specified spec.origins[x].jwt.issuer** dans la ressource Policy.

ServiceMeshPolicy (maistra.io/v1)

ServiceMeshPolicy a été configuré automatiquement pour le plan de contrôle Service Mesh via le paramètre **spec.istio.global.mtls.enabled** dans la ressource v1 ou **spec.security.dataPlane.mtls** dans la ressource v2. Pour les plans de contrôle v2, une ressource PeerAuthentication fonctionnellement équivalente est créée lors de l'installation. Cette fonctionnalité est obsolète dans Red Hat OpenShift Service Mesh version 2.0

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

Ces ressources ont été remplacées par la ressource **security.istio.io/v1beta1** AuthorizationPolicy.

Pour imiter le comportement de RbacConfig, il faut écrire une AuthorizationPolicy par défaut dont les paramètres dépendent du spec.mode spécifié dans RbacConfig.

- Lorsque **spec.mode** est défini sur **OFF**, aucune ressource n'est requise car la politique par défaut est ALLOW, à moins qu'une AuthorizationPolicy ne s'applique à la demande.

- Lorsque **spec.mode** est réglé sur ON, réglez **spec: {}**. Vous devez créer des politiques AuthorizationPolicy pour tous les services du maillage.

- **spec.mode** est défini sur **ON_WITH_INCLUSION**, doit créer une AuthorizationPolicy avec **spec: {}** dans chaque espace de noms inclus. L'inclusion de services individuels n'est pas prise en charge par AuthorizationPolicy. Toutefois, dès qu'une AuthorizationPolicy s'appliquant aux

charges de travail du service est créée, toutes les autres demandes qui ne sont pas explicitement autorisées sont rejetées.

- Lorsque **spec.mode** est défini sur **ON_WITH_EXCLUSION**, il n'est pas pris en charge par AuthorizationPolicy. Une politique DENY globale peut être créée, mais une AuthorizationPolicy doit être créée pour chaque charge de travail dans le maillage car il n'existe pas de politique allow-all pouvant être appliquée à un espace de noms ou à une charge de travail.

AuthorizationPolicy comprend la configuration à la fois du sélecteur auquel la configuration s'applique, qui est similaire à la fonction ServiceRoleBinding, et des règles qui doivent être appliquées, qui sont similaires à la fonction ServiceRole.

ServiceMeshRbacConfig (maistra.io/v1)

Cette ressource est remplacée par une ressource **security.istio.io/v1beta1** AuthorizationPolicy avec un spec.selector vide dans l'espace de noms du plan de contrôle du Service Mesh. Cette politique sera la politique d'autorisation par défaut appliquée à toutes les charges de travail dans le maillage. Pour plus de détails sur la migration, voir RbacConfig ci-dessus.

1.11.4.5.2.5. Plugins de mixage

Les composants Mixer sont désactivés par défaut dans la version 2.0. Si votre charge de travail repose sur des plugins Mixer, vous devez configurer votre version 2.0 **ServiceMeshControlPlane** pour inclure les composants Mixer.

Pour activer les composants de la politique Mixer, ajoutez l'extrait suivant à votre site **ServiceMeshControlPlane**.

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

Pour activer les composants de télémétrie de Mixer, ajoutez l'extrait suivant à votre site **ServiceMeshControlPlane**.

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

Les anciens plugins de mélangeur peuvent également être migrés vers WASM et intégrés à l'aide de la nouvelle ressource personnalisée ServiceMeshExtension (maistra.io/v1alpha1).

Les filtres WASM intégrés inclus dans la distribution Istio en amont ne sont pas disponibles dans Red Hat OpenShift Service Mesh 2.0.

1.11.4.5.2.6. Changements mutuels TLS

Lors de l'utilisation de mTLS avec des stratégies PeerAuthentication spécifiques à la charge de travail, une DestinationRule correspondante est nécessaire pour autoriser le trafic si la stratégie de la charge de travail diffère de la stratégie globale/de l'espace de noms.

La fonction Auto mTLS est activée par défaut, mais peut être désactivée en attribuant la valeur false à **spec.security.dataPlane.automtls** dans la ressource **ServiceMeshControlPlane**. Lors de la désactivation de mTLS auto, les règles de destination peuvent être nécessaires pour assurer une

communication correcte entre les services. Par exemple, le fait de définir PeerAuthentication sur **STRICT** pour un espace de noms peut empêcher les services d'autres espaces de noms d'y accéder, à moins qu'une règle de destination ne configure le mode TLS pour les services de l'espace de noms.

Pour plus d'informations sur mTLS, voir Activation de la sécurité mutuelle de la couche transport (mTLS)

1.11.4.5.2.6.1. Autres exemples de mTLS

Pour désactiver le service mTLS For productpage dans l'exemple d'application info, votre ressource Policy a été configurée de la manière suivante pour Red Hat OpenShift Service Mesh v1.1.

Exemple de ressource politique

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

Pour désactiver le service mTLS For productpage dans l'application d'exemple info, utilisez l'exemple suivant pour configurer votre ressource PeerAuthentication pour Red Hat OpenShift Service Mesh v2.0.

Exemple de ressource PeerAuthentication

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

Pour activer mTLS avec authentification JWT pour le service **productpage** dans l'exemple d'application info, votre ressource Policy a été configurée de la manière suivante pour Red Hat OpenShift Service Mesh v1.1.

Exemple de ressource politique

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

```
    ports:
    - number: 9000
  peers:
  - mtls:
  origins:
  - jwt:
      issuer: "https://securetoken.google.com"
      audiences:
      - "productpage"
      jwksUri: "https://www.googleapis.com/oauth2/v1/certs"
      jwtHeaders:
      - "x-goog-iap-jwt-assertion"
      triggerRules:
      - excludedPaths:
        - exact: /health_check
  principalBinding: USE_ORIGIN
```

Pour activer l'authentification mTLS With JWT pour le service productpage dans l'application d'exemple info, utilisez l'exemple suivant pour configurer votre ressource PeerAuthentication pour Red Hat OpenShift Service Mesh v2.0.

Exemple de ressource PeerAuthentication

```
#require mtls for productpage:9000
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: productpage-mTLS-with-JWT
  namespace: <namespace>
spec:
  selector:
    matchLabels:
      # this should match the selector for the "productpage" service
      app: productpage
  portLevelMtls:
    9000:
      mode: STRICT
---
#JWT authentication for productpage
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: productpage-mTLS-with-JWT
  namespace: <namespace>
spec:
  selector:
    matchLabels:
      # this should match the selector for the "productpage" service
      app: productpage
  jwtRules:
  - issuer: "https://securetoken.google.com"
    audiences:
    - "productpage"
    jwksUri: "https://www.googleapis.com/oauth2/v1/certs"
    fromHeaders:
    - name: "x-goog-iap-jwt-assertion"
```

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

1.11.4.5.3. Recettes de configuration

Ces recettes de configuration permettent de configurer les éléments suivants.

1.11.4.5.3.1. TLS mutuel dans un plan de données

TLS mutuel pour la communication du plan de données est configuré via **spec.security.dataPlane.mtls** dans la ressource **ServiceMeshControlPlane**, qui est **false** par défaut.

1.11.4.5.3.2. Clé de signature personnalisée

Istiod gère les certificats clients et les clés privées utilisés par les proxys de services. Par défaut, Istiod utilise un certificat auto-signé pour la signature, mais vous pouvez configurer un certificat et une clé privée personnalisés. Pour plus d'informations sur la configuration des clés de signature, voir Ajout d'une clé et d'un certificat d'autorité de certification externe

1.11.4.5.3.3. Traçage

Le traçage est configuré à l'adresse **spec.tracing**. Actuellement, le seul type de traceur pris en charge est **Jaeger**. L'échantillonnage est un nombre entier échelonné représentant des incréments de 0,01 %, par exemple, 1 est 0,01 et 10000 est 100 %. La mise en œuvre du traçage et le taux d'échantillonnage peuvent être spécifiés :

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

Jaeger est configuré dans la section **addons** de la ressource **ServiceMeshControlPlane**.

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

L'installation de Jaeger peut être personnalisée à l'aide du champ **install**. La configuration des conteneurs, comme les limites des ressources, est configurée dans les champs liés à **spec.runtime.components.jaeger**. Si une ressource Jaeger correspondant à la valeur de **spec.addons.jaeger.name** existe, le plan de contrôle Service Mesh sera configuré pour utiliser l'installation existante. Utilisez une ressource Jaeger existante pour personnaliser entièrement votre installation Jaeger.

1.11.4.5.3.4. Visualisation

Kiali et Grafana sont configurés dans la section **addons** de la ressource **ServiceMeshControlPlane**.

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

-

Les installations Grafana et Kiali peuvent être personnalisées via leurs champs **install** respectifs. La personnalisation des conteneurs, comme les limites de ressources, est configurée dans **spec.runtime.components.kiali** et **spec.runtime.components.grafana**. Si une ressource Kiali existante correspondant à la valeur du nom existe, le plan de contrôle Service Mesh configure la ressource Kiali pour l'utiliser avec le plan de contrôle. Certains champs de la ressource Kiali sont remplacés, comme la liste **accessible_namespaces**, ainsi que les points de terminaison pour Grafana, Prometheus et le traçage. Utilisez une ressource existante pour personnaliser entièrement votre installation Kiali.

1.11.4.5.3.5. Utilisation des ressources et programmation

Les ressources sont configurées sous **spec.runtime.<component>**. Les noms de composants suivants sont pris en charge.

Composant	Description	Versions prises en charge
sécurité	Conteneur Citadel	v1.0/1.1
cuisine	Conteneur de cuisine	v1.0/1.1
pilote	Conteneur pilote/instiod	v1.0/1.1/2.0
mélangeur	conteneurs istio-télémétrie et istio-politique	v1.0/1.1
mixer.policy	conteneur de la politique de l'istio	v2.0
mixer.telemetry	conteneur istio-télémétrique	v2.0
global.ouathproxy	conteneur oauth-proxy utilisé avec divers addons	v1.0/1.1/2.0
sidecarInjectorWebhook	sidecar injector webhook container	v1.0/1.1
tracing.jaeger	le conteneur général de Jaeger - tous les paramètres peuvent ne pas être appliqués. La personnalisation complète de l'installation de Jaeger est possible en spécifiant une ressource Jaeger existante dans la configuration du plan de contrôle de Service Mesh.	v1.0/1.1/2.0
tracing.jaeger.agent	paramètres spécifiques à l'agent Jaeger	v1.0/1.1/2.0
tracing.jaeger.allInOne	paramètres spécifiques à Jaeger allInOne	v1.0/1.1/2.0

Composant	Description	Versions prises en charge
tracing.jaeger.collector	paramètres spécifiques au collecteur Jaeger	v1.0/1.1/2.0
tracing.jaeger.elasticsearch	paramètres spécifiques au déploiement de Jaeger elasticsearch	v1.0/1.1/2.0
tracing.jaeger.query	paramètres spécifiques à la requête Jaeger	v1.0/1.1/2.0
prometheus	conteneur prométhée	v1.0/1.1/2.0
kiali	Conteneur Kiali - la personnalisation complète de l'installation Kiali est possible en spécifiant une ressource Kiali existante dans la configuration du plan de contrôle Service Mesh.	v1.0/1.1/2.0
grafana	Conteneur Grafana	v1.0/1.1/2.0
3scale	conteneur 3scale	v1.0/1.1/2.0
wasmExtensions.cacher	Conteneur cacheur d'extensions WASM	v2.0 - aperçu technique

Certains composants prennent en charge la limitation des ressources et la planification. Pour plus d'informations, voir Performances et évolutivité.

1.11.4.5.4. Prochaines étapes de la migration de vos applications et charges de travail

Déplacez la charge de travail de l'application vers le nouveau maillage et supprimez les anciennes instances pour terminer la mise à niveau.

1.11.5. Mise à niveau du plan de données

Votre plan de données fonctionnera toujours après la mise à niveau du plan de contrôle. Mais pour appliquer les mises à jour du proxy Envoy et toute modification de la configuration du proxy, vous devez redémarrer vos pods d'application et vos charges de travail.

1.11.5.1. Mise à jour des applications et des charges de travail

Pour terminer la migration, redémarrez tous les pods d'application dans le maillage pour mettre à niveau les proxies Envoy sidecar et leur configuration.

Pour effectuer une mise à jour continue d'un déploiement, utilisez la commande suivante :

```
$ oc rollout restart <deployment>
```

Vous devez effectuer une mise à jour continue pour toutes les applications qui composent le maillage.

1.12. GESTION DES UTILISATEURS ET DES PROFILS

1.12.1. Création des membres de Red Hat OpenShift Service Mesh

ServiceMeshMember permettent aux administrateurs de Red Hat OpenShift Service Mesh de déléguer des permissions pour ajouter des projets à un service mesh, même lorsque les utilisateurs respectifs n'ont pas d'accès direct au projet de service mesh ou à la liste des membres. Bien que les administrateurs de projet reçoivent automatiquement l'autorisation de créer la ressource **ServiceMeshMember** dans leur projet, ils ne peuvent pas la diriger vers **ServiceMeshControlPlane** jusqu'à ce que l'administrateur du service mesh accorde explicitement l'accès au service mesh. Les administrateurs peuvent accorder aux utilisateurs des autorisations d'accès au maillage en leur attribuant le rôle d'utilisateur **mesh-user**. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

```
oc policy add-role-to-user -n istio-system --role-namespace istio-system mesh-user <user_name>
```

Les administrateurs peuvent modifier la liaison de rôle **mesh-user** dans le projet de plan de contrôle Service Mesh pour spécifier les utilisateurs et les groupes auxquels l'accès est accordé. Le site **ServiceMeshMember** ajoute le projet au site **ServiceMeshMemberRoll** dans le projet de plan de contrôle Service Mesh auquel il fait référence.

```
apiVersion: maistra.io/v1
kind: ServiceMeshMember
metadata:
  name: default
spec:
  controlPlaneRef:
    namespace: istio-system
    name: basic
```

La liaison de rôle **mesh-users** est créée automatiquement après que l'administrateur a créé la ressource **ServiceMeshControlPlane**. Un administrateur peut utiliser la commande suivante pour ajouter un rôle à un utilisateur.

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

L'administrateur peut également créer le lien de rôle **mesh-user** avant de créer la ressource **ServiceMeshControlPlane**. Par exemple, l'administrateur peut la créer dans la même opération **oc apply** que la ressource **ServiceMeshControlPlane**.

Cet exemple ajoute une liaison de rôle pour **alice**:

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

```
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: alice
```

1.12.2. Création de profils de plan de contrôle Service Mesh

Vous pouvez créer des configurations réutilisables avec les profils **ServiceMeshControlPlane**. Les utilisateurs individuels peuvent étendre les profils qu'ils créent avec leurs propres configurations. Les profils peuvent également hériter des informations de configuration d'autres profils. Par exemple, vous pouvez créer un plan de contrôle comptable pour l'équipe comptable et un plan de contrôle marketing pour l'équipe marketing. Si vous créez un modèle de développement et un modèle de production, les membres de l'équipe marketing et de l'équipe comptable peuvent étendre les profils de développement et de production en les personnalisant en fonction de leur équipe.

Lorsque vous configurez des profils de plan de contrôle Service Mesh, qui suivent la même syntaxe que **ServiceMeshControlPlane**, les utilisateurs héritent des paramètres de manière hiérarchique. L'Opérateur est livré avec un profil **default** avec des paramètres par défaut pour Red Hat OpenShift Service Mesh.

1.12.2.1. Création du ConfigMap

Pour ajouter des profils personnalisés, vous devez créer un **ConfigMap** nommé **smcp-templates** dans le projet **openshift-operators**. Le conteneur Operator monte automatiquement le conteneur **ConfigMap**.

Conditions préalables

- Un opérateur de maillage de services installé et vérifié.
- Un compte avec le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.
- Emplacement du déploiement de l'opérateur.
- Accès à la CLI OpenShift (**oc**).

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform en tant que **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.
2. À partir de l'interface de programmation, exécutez cette commande pour créer le ConfigMap nommé **smcp-templates** dans le projet **openshift-operators** et remplacez **<profiles-directory>** par l'emplacement des fichiers **ServiceMeshControlPlane** sur votre disque local :

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

3. Vous pouvez utiliser le paramètre **profiles** dans **ServiceMeshControlPlane** pour spécifier un ou plusieurs modèles.

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

```
spec:
  profiles:
  - default
```

1.12.2.2. Définir la bonne politique de réseau

Service Mesh crée des stratégies réseau dans le plan de contrôle Service Mesh et les espaces de noms membres pour autoriser le trafic entre eux. Avant de procéder au déploiement, tenez compte des conditions suivantes pour vous assurer que les services de votre maillage de services qui étaient auparavant exposés par l'intermédiaire d'une route OpenShift Container Platform.

- Le trafic dans le maillage de services doit toujours passer par la passerelle d'entrée (ingress-gateway) pour qu'Istio fonctionne correctement.
- Déployer des services externes au maillage de services dans des espaces de noms distincts qui ne font partie d'aucun maillage de services.
- Les services non maillés qui doivent être déployés dans un espace de noms enlisé de maillage de services doivent étiqueter leurs déploiements **maistra.io/expose-route: "true"**, ce qui garantit que les routes d'OpenShift Container Platform vers ces services fonctionnent toujours.

1.13. SÉCURITÉ

Si votre application Service Mesh est construite avec un ensemble complexe de microservices, vous pouvez utiliser Red Hat OpenShift Service Mesh pour personnaliser la sécurité de la communication entre ces services. L'infrastructure d'OpenShift Container Platform et les fonctionnalités de gestion du trafic de Service Mesh vous aident à gérer la complexité de vos applications et à sécuriser les microservices.

Avant de commencer

Si vous avez un projet, ajoutez-le à la ressource**ServiceMeshMemberRoll** .

Si vous n'avez pas de projet, installez l'exemple d'application Bookinfo et ajoutez-le à la ressource **ServiceMeshMemberRoll**. L'exemple d'application permet d'illustrer les concepts de sécurité.

1.13.1. À propos de la sécurité mutuelle de la couche transport (mTLS)

Mutual Transport Layer Security (mTLS) est un protocole qui permet à deux parties de s'authentifier mutuellement. Il s'agit du mode d'authentification par défaut dans certains protocoles (IKE, SSH) et facultatif dans d'autres (TLS). Vous pouvez utiliser mTLS sans modifier le code de l'application ou du service. Le TLS est entièrement géré par l'infrastructure du service mesh et entre les deux proxys sidecar.

Par défaut, mTLS dans Red Hat OpenShift Service Mesh est activé et configuré en mode permissif, où les sidecars dans Service Mesh acceptent à la fois le trafic en texte clair et les connexions qui sont chiffrées à l'aide de mTLS. Si un service dans votre maillage communique avec un service en dehors du maillage, un mTLS strict pourrait interrompre la communication entre ces services. Utilisez le mode permissif pendant que vous migrez vos charges de travail vers Service Mesh. Ensuite, vous pouvez activer mTLS strict dans votre maillage, votre espace de noms ou votre application.

L'activation de mTLS dans votre maillage au niveau du plan de contrôle du maillage de services sécurise l'ensemble du trafic dans votre maillage de services sans réécrire vos applications et charges de travail. Vous pouvez sécuriser les espaces de noms dans votre maillage au niveau du plan de données dans la

ressource **ServiceMeshControlPlane**. Pour personnaliser les connexions de cryptage du trafic, configurez les espaces de noms au niveau de l'application avec les ressources **PeerAuthentication** et **DestinationRule**.

1.13.1.1. Activation de mTLS strict à travers le maillage de services

Si vos charges de travail ne communiquent pas avec des services externes, vous pouvez rapidement activer mTLS à travers votre maillage sans interruption de la communication. Vous pouvez l'activer en définissant **spec.security.dataPlane.mtls** sur **true** dans la ressource **ServiceMeshControlPlane**. L'opérateur crée les ressources nécessaires.

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  version: v2.3
  security:
    dataPlane:
      mtls: true
```

Vous pouvez également activer mTLS en utilisant la console web d'OpenShift Container Platform.

Procédure

1. Connectez-vous à la console web.
2. Cliquez sur le menu **Project** et sélectionnez le projet dans lequel vous avez installé le plan de contrôle Service Mesh, par exemple **istio-system**.
3. Cliquez sur **Operators → Installed Operators**.
4. Cliquez sur **Service Mesh Control Plane** sous **Provided APIs**.
5. Cliquez sur le nom de votre ressource **ServiceMeshControlPlane**, par exemple **basic**.
6. Sur la page **Details**, cliquez sur la bascule dans la section **Security** pour **Data Plane Security**.

1.13.1.1.1. Configuration des sidecars pour les connexions entrantes pour des services spécifiques

Vous pouvez également configurer mTLS pour des services individuels en créant une politique.

Procédure

1. Créez un fichier YAML à l'aide de l'exemple suivant.

Politique d'authentification par les pairs exemple policy.yaml

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: <namespace>
spec:
  mtls:
    mode: STRICT
```

a. Remplacez **<namespace>** par l'espace de noms dans lequel se trouve le service.

2. Exécutez la commande suivante pour créer la ressource dans l'espace de noms où se trouve le service. Elle doit correspondre au champ **namespace** de la ressource Policy que vous venez de créer.

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

NOTE

Si vous n'utilisez pas mTLS automatique et que vous définissez **PeerAuthentication** sur STRICT, vous devez créer une ressource **DestinationRule** pour votre service.

1.13.1.1.2. Configuration des sidecars pour les connexions sortantes

Créez une règle de destination pour configurer Service Mesh afin qu'il utilise mTLS lors de l'envoi de requêtes à d'autres services dans le maillage.

Procédure

1. Créez un fichier YAML à l'aide de l'exemple suivant.

Exemple de règle de destination destination-rule.yaml

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

a. Remplacez **<namespace>** par l'espace de noms dans lequel se trouve le service.

2. Exécutez la commande suivante pour créer la ressource dans l'espace de noms où se trouve le service. Elle doit correspondre au champ **namespace** de la ressource **DestinationRule** que vous venez de créer.

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

1.13.1.1.3. Définition des versions minimale et maximale du protocole

Si votre environnement a des exigences spécifiques pour le trafic crypté dans votre maillage de services, vous pouvez contrôler les fonctions cryptographiques autorisées en définissant les valeurs **spec.security.controlPlane.tls.minProtocolVersion** ou **spec.security.controlPlane.tls.maxProtocolVersion** dans votre ressource **ServiceMeshControlPlane**. Ces valeurs, configurées dans votre ressource de plan de contrôle Service Mesh, définissent la version TLS minimale et maximale utilisée par les composants du maillage lorsqu'ils communiquent de manière sécurisée via TLS.

La valeur par défaut est **TLS_AUTO** et ne spécifie pas de version de TLS.

Tableau 1.5. Valeurs valables

Valeur	Description
TLS_AUTO	par défaut
TLSv1_0	TLS version 1.0
TLSv1_1	TLS version 1.1
TLSv1_2	TLS version 1.2
TLSv1_3	TLS version 1.3

Procédure

1. Connectez-vous à la console web.
2. Cliquez sur le menu **Project** et sélectionnez le projet dans lequel vous avez installé le plan de contrôle Service Mesh, par exemple **istio-system**.
3. Cliquez sur **Operators → Installed Operators**.
4. Cliquez sur **Service Mesh Control Plane** sous **Provided APIs**.
5. Cliquez sur le nom de votre ressource **ServiceMeshControlPlane**, par exemple **basic**.
6. Cliquez sur l'onglet **YAML**.
7. Insérez l'extrait de code suivant dans l'éditeur YAML. Remplacez la valeur de **minProtocolVersion** par la valeur de la version TLS. Dans cet exemple, la version TLS minimale est fixée à **TLSv1_2**.

   ServiceMeshControlPlane snippet

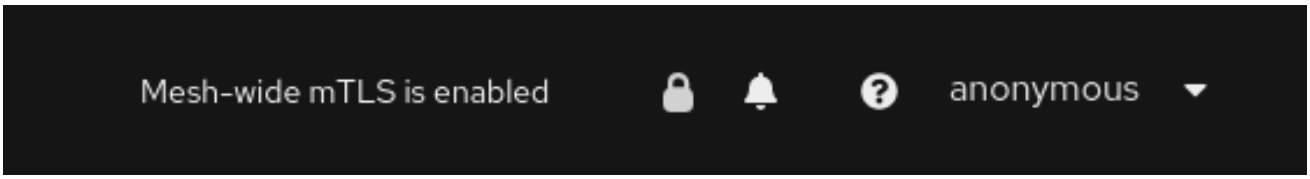
   ```
   kind: ServiceMeshControlPlane
   spec:
     security:
       controlPlane:
         tls:
           minProtocolVersion: TLSv1_2
   ```

8. Cliquez sur **Save**.
9. Cliquez sur **Refresh** pour vérifier que les modifications ont été correctement mises à jour.

1.13.1.2. Valider le chiffrement avec Kiali

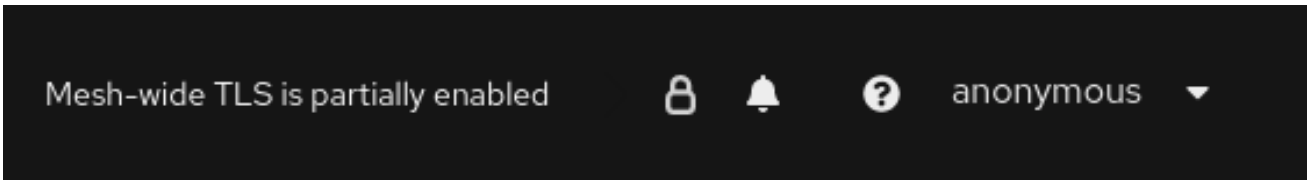
La console Kiali offre plusieurs moyens de vérifier si vos applications, services et charges de travail ont activé le chiffrement mTLS.

Figure 1.5. Icône de l'en-tête mTLS activé à l'échelle de la maille



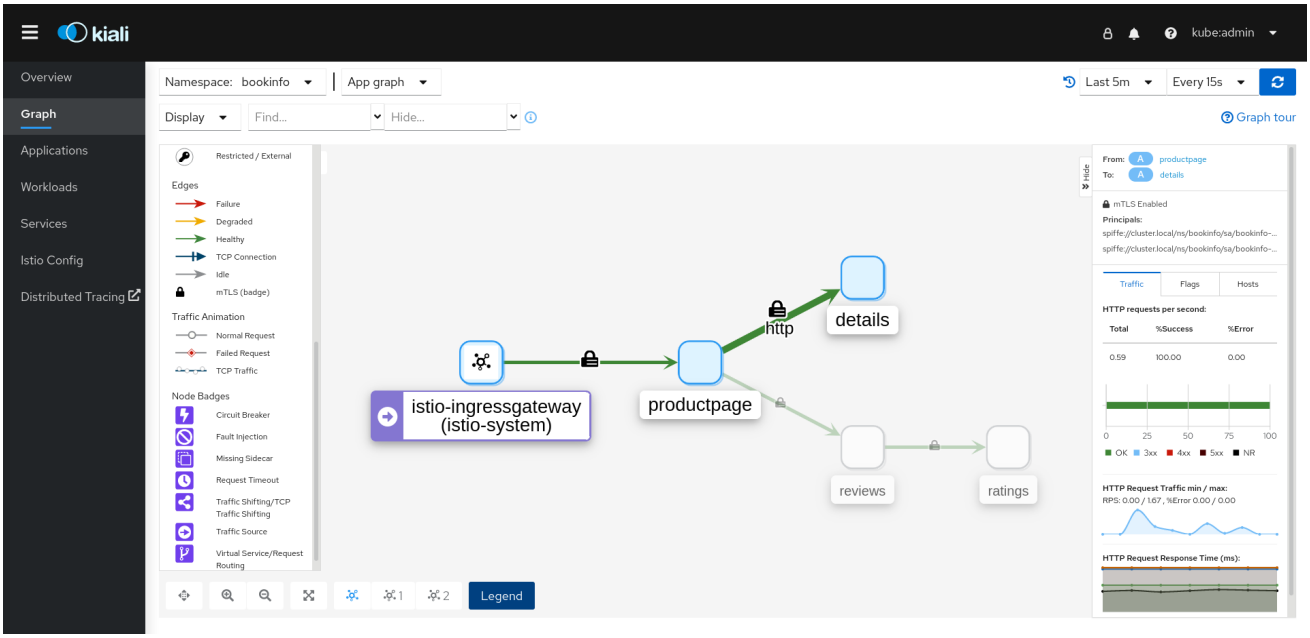
Sur le côté droit de l'en-tête, Kiali affiche une icône de cadenas lorsque le maillage a strictement activé mTLS pour l'ensemble du maillage de services. Cela signifie que toutes les communications dans le maillage utilisent mTLS.

Figure 1.6. Icône de l'en-tête maillage mTLS partiellement activé



Kiali affiche une icône de cadenas creux lorsque le maillage est configuré en mode **PERMISSIVE** ou qu'il y a une erreur dans la configuration mTLS du maillage.

Figure 1.7. Badge de sécurité



La page **Graph** offre la possibilité d'afficher un badge **Security** sur les bords du graphique pour indiquer que mTLS est activé. Pour activer les badges de sécurité sur le graphique, dans le menu **Display**, sous **Show Badges**, cochez la case **Security**. Lorsqu'une arête affiche une icône de cadenas, cela signifie qu'au moins une requête avec mTLS activé est présente. S'il y a à la fois des requêtes mTLS et des requêtes non mTLS, le panneau latéral indique le pourcentage de requêtes utilisant mTLS.

La page **Applications Detail Overview** affiche une icône **Security** sur les bords du graphique lorsqu'au moins une requête avec mTLS activé est présente.

La page **Workloads Detail Overview** affiche une icône **Security** sur les bords du graphique lorsqu'au moins une requête avec mTLS activé est présente.

La page **Services Detail Overview** affiche une icône **Security** sur les bords du graphique lorsqu'au moins une requête avec mTLS activé est présente. Notez également que Kiali affiche une icône de verrouillage dans la section **Network** à côté des ports configurés pour mTLS.

1.13.2. Configuration du contrôle d'accès basé sur les rôles (RBAC)

Les objets de contrôle d'accès basé sur les rôles (RBAC) déterminent si un utilisateur ou un service est autorisé à effectuer une action donnée au sein d'un projet. Vous pouvez définir un contrôle d'accès à l'échelle du maillage, de l'espace de noms et de la charge de travail pour vos charges de travail dans le maillage.

Pour configurer RBAC, créez une ressource **AuthorizationPolicy** dans l'espace de noms pour lequel vous configurez l'accès. Si vous configurez un accès à l'échelle de la maille, utilisez le projet dans lequel vous avez installé le plan de contrôle Service Mesh, par exemple **istio-system**.

Par exemple, avec RBAC, vous pouvez créer des politiques qui.. :

- Configurer la communication au sein du projet.
- Autoriser ou refuser l'accès complet à toutes les charges de travail dans l'espace de noms par défaut.
- Autoriser ou refuser l'accès à la passerelle d'entrée.
- Exiger un jeton pour l'accès.

Une politique d'autorisation comprend un sélecteur, une action et une liste de règles :

- Le champ **selector** indique la cible de la politique.
- Le champ **action** indique s'il faut autoriser ou refuser la demande.
- Le champ **rules** précise le moment où l'action doit être déclenchée.
  - Le champ **from** précise les contraintes relatives à l'origine de la demande.
  - Le champ **to** précise les contraintes relatives à la cible et aux paramètres de la demande.
  - Le champ **when** spécifie des conditions supplémentaires pour l'application de la règle.

Procédure

1. Créez votre ressource **AuthorizationPolicy**. L'exemple suivant montre une ressource qui met à jour la politique d'entrée **AuthorizationPolicy** pour interdire à une adresse IP d'accéder à la passerelle d'entrée.

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

```
  - from:
    - source:
      ipBlocks: ["1.2.3.4"]
```

2. Exécutez la commande suivante après avoir écrit votre ressource pour la créer dans votre espace de noms. L'espace de noms doit correspondre au champ **metadata.namespace** de la ressource **AuthorizationPolicy**.

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

Prochaines étapes

Les exemples suivants illustrent d'autres configurations courantes.

1.13.2.1. Configurer la communication au sein du projet

Vous pouvez utiliser **AuthorizationPolicy** pour configurer le plan de contrôle de votre Service Mesh afin d'autoriser ou de refuser le trafic communiquant avec votre maillage ou les services de votre maillage.

1.13.2.1.1. Restreindre l'accès aux services en dehors d'un espace de noms

Vous pouvez refuser les demandes provenant de toute source qui n'appartient pas à l'espace de noms **info** avec l'exemple de ressource **AuthorizationPolicy** suivant.

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

1.13.2.1.2. Création de politiques d'autorisation "allow-all" et "deny-all" par défaut

L'exemple suivant montre une politique d'autorisation "allow-all" qui permet un accès complet à toutes les charges de travail dans l'espace de noms **info**.

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

L'exemple suivant montre une stratégie qui refuse tout accès à toutes les charges de travail dans l'espace de noms **info**.

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

1.13.2.2. Autoriser ou refuser l'accès à la passerelle d'entrée

Vous pouvez définir une politique d'autorisation pour ajouter des listes d'autorisation ou de refus basées sur les adresses IP.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: ALLOW
  rules:
  - from:
    - source:
        ipBlocks: ["1.2.3.4", "5.6.7.0/24"]
```

1.13.2.3. Restreindre l'accès avec un jeton Web JSON

Vous pouvez restreindre l'accès à votre maillage à l'aide d'un jeton Web JSON (JWT). Après authentification, un utilisateur ou un service peut accéder aux routes et aux services associés à ce jeton.

Créez une ressource **RequestAuthentication**, qui définit les méthodes d'authentification prises en charge par une charge de travail. L'exemple suivant accepte un JWT émis par **http://localhost:8080/auth/realms/master**.

```
apiVersion: "security.istio.io/v1beta1"
kind: "RequestAuthentication"
metadata:
  name: "jwt-example"
  namespace: info
spec:
  selector:
    matchLabels:
      app: httpbin
  jwtRules:
  - issuer: "http://localhost:8080/auth/realms/master"
    jwksUri: "http://keycloak.default.svc:8080/auth/realms/master/protocol/openid-connect/certs"
```

Ensuite, créez une ressource **AuthorizationPolicy** dans le même espace de noms pour travailler avec la ressource **RequestAuthentication** que vous avez créée. L'exemple suivant exige qu'un JWT soit présent dans l'en-tête **Authorization** lors de l'envoi d'une requête aux charges de travail **httpbin**.

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

1.13.3. Configuration des suites de chiffrement et des courbes ECDH

Les suites de chiffrement et les courbes de Diffie-Hellman à courbe elliptique (courbes ECDH) peuvent vous aider à sécuriser votre maillage de services. Vous pouvez définir une liste séparée par des virgules de suites de chiffrement en utilisant **spec.security.controlplane.tls.cipherSuites** et de courbes ECDH en utilisant **spec.security.controlplane.tls.ecdhCurves** dans votre ressource **ServiceMeshControlPlane**. Si l'un de ces attributs est vide, les valeurs par défaut sont utilisées.

Le paramètre **cipherSuites** est efficace si votre maillage de services utilise TLS 1.2 ou une version antérieure. Il n'a aucun effet lors de la négociation avec TLS 1.3.

Définissez vos suites de chiffrement dans la liste séparée par des virgules, par ordre de priorité. Par exemple, **ecdhCurves: CurveP256, CurveP384** donne à **CurveP256** une priorité plus élevée que **CurveP384**.

NOTE

Vous devez inclure **TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256** ou **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256** lorsque vous configurez la suite de chiffrement. La prise en charge de HTTP/2 nécessite au moins l'une de ces suites de chiffrement.

Les suites de chiffrement prises en charge sont les suivantes

- 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

Les courbes ECDH prises en charge sont les suivantes :

- CourbeP256
- CourbeP384
- CourbeP521
- X25519

1.13.4. Ajout d'une clé et d'un certificat d'une autorité de certification externe

Par défaut, Red Hat OpenShift Service Mesh génère un certificat racine et une clé auto-signés et les utilise pour signer les certificats de charge de travail. Vous pouvez également utiliser le certificat et la clé définis par l'utilisateur pour signer les certificats de charge de travail avec le certificat racine défini par l'utilisateur. Cette tâche présente un exemple d'insertion de certificats et de clés dans Service Mesh.

Conditions préalables

- Installez Red Hat OpenShift Service Mesh avec TLS mutuel activé pour configurer les certificats.
- Cet exemple utilise les certificats du référentiel Maistra. Pour la production, utilisez vos propres certificats provenant de votre autorité de certification.
- Déployez l'application d'exemple Bookinfo pour vérifier les résultats à l'aide de ces instructions.
- OpenSSL est nécessaire pour vérifier les certificats.

1.13.4.1. Ajouter un certificat et une clé existants

Pour utiliser un certificat et une clé de signature (CA) existants, vous devez créer un fichier de chaîne de confiance qui inclut le certificat CA, la clé et le certificat racine. Vous devez utiliser les noms de fichiers exacts suivants pour chacun des certificats correspondants. Le certificat CA est nommé **ca-cert.pem**, la clé est **ca-key.pem**, et le certificat racine, qui signe **ca-cert.pem**, est nommé **root-cert.pem**. Si votre charge de travail utilise des certificats intermédiaires, vous devez les spécifier dans un fichier **cert-chain.pem**.

1. Enregistrez localement les certificats d'exemple du référentiel Maistra et remplacez **<path>** par le chemin d'accès à vos certificats.

2. Créez un secret nommé **cacert** qui comprend les fichiers d'entrée **ca-cert.pem**, **ca-key.pem**, **root-cert.pem** et **cert-chain.pem**.

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

3. Dans la ressource **ServiceMeshControlPlane**, définissez **spec.security.dataPlane.mtls true** comme **true** et configurez le champ **certificateAuthority** comme indiqué dans l'exemple suivant. La valeur par défaut de **rootCADir** est **/etc/cacerts**. Il n'est pas nécessaire de définir le champ **privateKey** si la clé et les certificats sont montés à l'emplacement par défaut. Service Mesh lit les certificats et la clé à partir des fichiers secret-mount.

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

4. Après avoir créé/modifié/supprimé le secret **cacert**, les pods de plan de contrôle Service Mesh **istiod** et **gateway** doivent être redémarrés pour que les modifications soient prises en compte. Utilisez la commande suivante pour redémarrer les modules :

```
$ oc -n istio-system delete pods -l 'app in (istiod,istio-ingressgateway, istio-egressgateway)'
```

   L'Opérateur recréera automatiquement les pods après qu'ils aient été supprimés.

5. Redémarrez les pods d'application info pour que les proxies sidecar récupèrent les changements de secret. Utilisez la commande suivante pour redémarrer les modules :

```
$ oc -n info delete pods --all
```

   Vous devriez obtenir un résultat similaire à celui qui suit :

```
pod "details-v1-6cd699df8c-j54nh" deleted
pod "productpage-v1-5ddcb4b84f-mtmf2" deleted
pod "ratings-v1-bdbcc68bc-kmng4" deleted
```

```
pod "reviews-v1-754ddd7b6f-lqhsv" deleted
pod "reviews-v2-675679877f-q67r2" deleted
pod "reviews-v3-79d7549c7-c2gjs" deleted
```

6. Vérifiez que les pods ont été créés et sont prêts avec la commande suivante :

```
$ oc get pods -n info
```

1.13.4.2. Vérification des certificats

Utilisez l'exemple d'application Bookinfo pour vérifier que les certificats de charge de travail sont signés par les certificats qui ont été insérés dans l'autorité de certification. Pour ce faire, vous devez avoir installé **openssl** sur votre machine

1. Pour extraire les certificats des charges de travail info, utilisez la commande suivante :

```
$ sleep 60
$ oc -n info exec "$(oc -n bookinfo get pod -l app=productpage -o jsonpath=
{.items..metadata.name})" -c istio-proxy -- openssl s_client -showcerts -connect details:9080
> bookinfo-proxy-cert.txt
$ sed -n '/-----BEGIN CERTIFICATE-----/{:start /-----END CERTIFICATE-----/!{N;b
start};/.*/p}' info-proxy-cert.txt > certs.pem
$ awk 'BEGIN {counter=0;} /BEGIN CERT/{counter++} { print > "proxy-cert-" counter ".pem"}'
< certs.pem
```

   Après avoir exécuté la commande, vous devriez avoir trois fichiers dans votre répertoire de travail : **proxy-cert-1.pem proxy-cert-2.pem** et **proxy-cert-3.pem**.

2. Vérifiez que le certificat racine est le même que celui spécifié par l'administrateur. Remplacez **<path>** par le chemin d'accès à vos certificats.

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

   Exécutez la syntaxe suivante dans la fenêtre du terminal.

```
$ openssl x509 -in ./proxy-cert-3.pem -text -noout > /tmp/pod-root-cert.crt.txt
```

   Comparez les certificats en exécutant la syntaxe suivante dans la fenêtre du terminal.

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

   Vous devriez obtenir le résultat suivant : **Files /tmp/root-cert.crt.txt and /tmp/pod-root-cert.crt.txt are identical**

3. Vérifiez que le certificat de l'autorité de certification est le même que celui spécifié par l'administrateur. Remplacez **<path>** par le chemin d'accès à vos certificats.

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

   Exécutez la syntaxe suivante dans la fenêtre du terminal.

```
openssl x509 -in ./proxy-cert-2.pem -text -noout > /tmp/pod-cert-chain-ca.crt.txt
```

Comparez les certificats en exécutant la syntaxe suivante dans la fenêtre du terminal.

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

Vous devriez obtenir le résultat suivant : **Files /tmp/ca-cert.crt.txt and /tmp/pod-cert-chain-ca.crt.txt are identical.**

4. Vérifiez la chaîne de certificats depuis le certificat racine jusqu'au certificat de charge de travail. Remplacez **<path>** par le chemin d'accès à vos certificats.

```
$ openssl verify -CAfile <(cat <path>/ca-cert.pem <path>/root-cert.pem) ./proxy-cert-1.pem
```

Vous devriez obtenir le résultat suivant : **./proxy-cert-1.pem: OK**

1.13.4.3. Suppression des certificats

Pour supprimer les certificats que vous avez ajoutés, procédez comme suit.

1. Supprimez le secret **cacerts**. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

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

2. Redéployer Service Mesh avec un certificat racine auto-signé dans la ressource **ServiceMeshControlPlane**.

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

1.14. GÉRER LE TRAFIC DANS VOTRE MAILLAGE DE SERVICES

En utilisant Red Hat OpenShift Service Mesh, vous pouvez contrôler le flux de trafic et les appels API entre les services. Certains services de votre maillage de services peuvent avoir besoin de communiquer au sein du maillage et d'autres peuvent avoir besoin d'être cachés. Vous pouvez gérer le trafic pour cacher des services backend spécifiques, exposer des services, créer des déploiements de test ou de version, ou ajouter une couche de sécurité sur un ensemble de services.

1.14.1. Utilisation de passerelles

Vous pouvez utiliser une passerelle pour gérer le trafic entrant et sortant de votre maillage afin de spécifier le trafic que vous souhaitez voir entrer ou sortir du maillage. Les configurations de passerelle sont appliquées aux proxys Envoy autonomes qui s'exécutent à la périphérie du maillage, plutôt qu'aux proxys Envoy sidecar qui s'exécutent aux côtés de vos charges de travail de service.

Contrairement à d'autres mécanismes de contrôle du trafic entrant dans vos systèmes, tels que les API Kubernetes Ingress, les passerelles Red Hat OpenShift Service Mesh utilisent toute la puissance et la flexibilité du routage du trafic.

La ressource de passerelle Red Hat OpenShift Service Mesh peut utiliser les propriétés d'équilibrage de charge de la couche 4-6, telles que les ports, pour exposer et configurer les paramètres TLS de Red Hat

OpenShift Service Mesh. Au lieu d'ajouter le routage du trafic de la couche application (L7) à la même ressource API, vous pouvez lier un service virtuel Red Hat OpenShift Service Mesh ordinaire à la passerelle et gérer le trafic de la passerelle comme tout autre trafic de plan de données dans un maillage de services.

Les passerelles sont principalement utilisées pour gérer le trafic entrant, mais vous pouvez également configurer des passerelles de sortie. Une passerelle de sortie vous permet de configurer un nœud de sortie dédié pour le trafic quittant le maillage. Cela vous permet de limiter les services qui ont accès aux réseaux externes, ce qui ajoute un contrôle de sécurité à votre maillage de services. Vous pouvez également utiliser une passerelle pour configurer un proxy purement interne.

Exemple de passerelle

Une ressource de passerelle décrit un équilibreur de charge fonctionnant à la périphérie du maillage et recevant des connexions HTTP/TCP entrantes ou sortantes. La spécification décrit un ensemble de ports qui doivent être exposés, le type de protocole à utiliser, la configuration SNI pour l'équilibreur de charge, etc.

L'exemple suivant montre un exemple de configuration de passerelle pour le trafic HTTPS externe entrant :

```
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
```

Cette configuration de passerelle permet au trafic HTTPS provenant de **ext-host.example.com** d'entrer dans le maillage sur le port 443, mais ne spécifie aucun routage pour le trafic.

Pour spécifier le routage et pour que la passerelle fonctionne comme prévu, vous devez également lier la passerelle à un service virtuel. Pour ce faire, vous utilisez le champ Passerelles du service virtuel, comme le montre l'exemple suivant :

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

Vous pouvez ensuite configurer le service virtuel avec des règles de routage pour le trafic externe.

1.14.1.1. Activation de l'injection de passerelle

Les configurations de passerelles s'appliquent aux proxys Envoy autonomes s'exécutant à la périphérie du maillage, plutôt qu'aux proxys Envoy sidecar s'exécutant aux côtés de vos charges de travail de service. Les passerelles étant des proxies Envoy, vous pouvez configurer Service Mesh pour injecter automatiquement des passerelles, de la même manière que vous pouvez injecter des sidecars.

En utilisant l'injection automatique pour les passerelles, vous pouvez déployer et gérer les passerelles indépendamment de la ressource **ServiceMeshControlPlane** et gérer les passerelles avec vos applications utilisateur. L'utilisation de l'injection automatique pour les déploiements de passerelles donne aux développeurs un contrôle total sur le déploiement de la passerelle tout en simplifiant les opérations. Lorsqu'une nouvelle mise à niveau est disponible ou qu'une configuration a changé, vous redémarrez les pods de passerelle pour les mettre à jour. De cette manière, le fonctionnement d'un déploiement de passerelle est identique à celui des sidecars.

NOTE

L'injection est désactivée par défaut pour l'espace de noms **ServiceMeshControlPlane**, par exemple l'espace de noms **istio-system**. La meilleure pratique en matière de sécurité consiste à déployer les passerelles dans un espace de noms différent de celui du plan de contrôle.

1.14.1.2. Déploiement de l'injection automatique de passerelle

Lors du déploiement d'une passerelle, vous devez accepter l'injection en ajoutant une étiquette d'injection ou une annotation à l'objet passerelle **deployment**. L'exemple suivant déploie une passerelle.

Conditions préalables

- L'espace de noms doit être membre du maillage en le définissant dans le site **ServiceMeshMemberRoll** ou en créant une ressource **ServiceMeshMember**.

Procédure

1. Définissez une étiquette unique pour la passerelle d'entrée Istio. Ce paramètre est nécessaire pour que la passerelle puisse sélectionner la charge de travail. Cet exemple utilise **ingressgateway** comme nom de la passerelle.

```
apiVersion: v1
kind: Service
metadata:
  name: istio-ingressgateway
  namespace: istio-ingress
spec:
  type: ClusterIP
  selector:
    istio: ingressgateway
  ports:
  - name: http
    port: 80
    targetPort: 8080
  - name: https
    port: 443
```

```
    targetPort: 8443
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: istio-ingressgateway
  namespace: istio-ingress
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  template:
    metadata:
      annotations:
        inject.istio.io/templates: gateway
      labels:
        istio: ingressgateway
        sidecar.istio.io/inject: "true" (1)
    spec:
      containers:
      - name: istio-proxy
        image: auto (2)
```

(1) Activez l'injection de la passerelle en définissant le champ **sidecar.istio.io/inject** sur **"true"**.

(2) Définissez le champ **image** sur **auto** pour que l'image soit automatiquement mise à jour à chaque démarrage du pod.

2. Configurer les rôles pour autoriser la lecture des informations d'identification pour TLS.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: istio-ingressgateway-sds
  namespace: istio-ingress
rules:
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["get", "watch", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: istio-ingressgateway-sds
  namespace: istio-ingress
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: istio-ingressgateway-sds
subjects:
- kind: ServiceAccount
  name: default
```

3. Autoriser l'accès à la nouvelle passerelle depuis l'extérieur du cluster, ce qui est nécessaire lorsque **spec.security.manageNetworkPolicy** est défini sur **true**.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: gatewayingress
  namespace: istio-ingress
spec:
  podSelector:
    matchLabels:
      istio: ingressgateway
  ingress:
    - {}
  policyTypes:
  - Ingress
```

4. Dimensionner automatiquement le pod lorsque le trafic entrant augmente. Cet exemple définit les répliques minimales à **2** et les répliques maximales à **5**. Il crée également une autre réplique lorsque l'utilisation atteint 80 %.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  labels:
    istio: ingressgateway
    release: istio
  name: ingressgatewayhpa
  namespace: istio-ingress
spec:
  maxReplicas: 5
  metrics:
  - resource:
      name: cpu
      target:
        averageUtilization: 80
        type: Utilization
    type: Resource
  minReplicas: 2
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: istio-ingressgateway
```

5. Indiquez le nombre minimum de modules qui doivent être en cours d'exécution sur le nœud. Cet exemple garantit qu'une réplique est en cours d'exécution si un module est redémarré sur un nouveau nœud.

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  labels:
    istio: ingressgateway
    release: istio
  name: ingressgatewaypdb
```

```
  namespace: istio-ingress
spec:
  minAvailable: 1
  selector:
    matchLabels:
      istio: ingressgateway
```

1.14.1.3. Gestion du trafic entrant

Dans Red Hat OpenShift Service Mesh, la passerelle d'entrée permet aux fonctionnalités telles que la surveillance, la sécurité et les règles de routage de s'appliquer au trafic qui entre dans le cluster. Utilisez une passerelle Service Mesh pour exposer un service en dehors du service mesh.

1.14.1.3.1. Détermination de l'IP et des ports d'entrée

La configuration de l'entrée diffère selon que votre environnement supporte ou non un équilibreur de charge externe. Un équilibreur de charge externe est défini dans l'IP et les ports d'entrée du cluster. Pour déterminer si l'IP et les ports de votre cluster sont configurés pour les équilibreurs de charge externes, exécutez la commande suivante. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

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

Cette commande renvoie les adresses **NAME**, **TYPE**, **CLUSTER-IP**, **EXTERNAL-IP**, **PORT(S)**, et **AGE** de chaque élément de votre espace de noms.

Si la valeur **EXTERNAL-IP** est définie, votre environnement dispose d'un équilibreur de charge externe que vous pouvez utiliser comme passerelle d'entrée.

Si la valeur **EXTERNAL-IP** est **<none>**, ou perpétuellement **<pending>**, votre environnement ne fournit pas d'équilibreur de charge externe pour la passerelle d'entrée. Vous pouvez accéder à la passerelle en utilisant le port de nœud du service.

1.14.1.3.1.1. Détermination des ports d'entrée avec un équilibreur de charge

Suivez ces instructions si votre environnement dispose d'un équilibreur de charge externe.

Procédure

1. Exécutez la commande suivante pour définir l'adresse IP et les ports d'entrée. Cette commande définit une variable dans votre terminal.

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

2. Exécutez la commande suivante pour définir le port d'entrée.

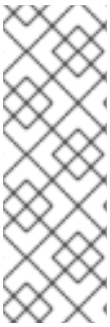
```
$ export INGRESS_PORT=$(oc -n istio-system get service istio-ingressgateway -o
jsonpath='{.spec.ports[?(@.name=="http2")].port}')
```

3. Exécutez la commande suivante pour définir le port d'entrée sécurisé.

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

4. Exécutez la commande suivante pour définir le port d'entrée TCP.

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



NOTE

Dans certains environnements, l'équilibreur de charge peut être exposé en utilisant un nom d'hôte au lieu d'une adresse IP. Dans ce cas, la valeur **EXTERNAL-IP** de la passerelle d'entrée n'est pas une adresse IP, mais un nom d'hôte. Il s'agit plutôt d'un nom d'hôte, et la commande précédente ne parvient pas à définir la variable d'environnement **INGRESS_HOST**.

Dans ce cas, utilisez la commande suivante pour corriger la valeur de **INGRESS_HOST**:

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

1.14.1.3.1.2. Détermination des ports d'entrée sans répartiteur de charge

Si votre environnement ne dispose pas d'un équilibreur de charge externe, déterminez les ports d'entrée et utilisez un port de nœud à la place.

Procédure

1. Définir les ports d'entrée.

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

2. Exécutez la commande suivante pour définir le port d'entrée sécurisé.

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

3. Exécutez la commande suivante pour définir le port d'entrée TCP.

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

1.14.1.4. Configuration d'une passerelle d'entrée

Une passerelle d'entrée est un équilibreur de charge fonctionnant à la périphérie du réseau maillé et recevant des connexions HTTP/TCP entrantes. Elle configure les ports et les protocoles exposés, mais n'inclut aucune configuration d'acheminement du trafic. Le routage du trafic entrant est plutôt configuré avec des règles de routage, de la même manière que pour les demandes de services internes.

Les étapes suivantes montrent comment créer une passerelle et configurer un site **VirtualService** pour exposer un service de l'application d'exemple Bookinfo au trafic extérieur pour les chemins /**productpage** et /**login**.

Procédure

1. Créer une passerelle pour accepter le trafic.

   a. Créez un fichier YAML et copiez-y le YAML suivant.

      Exemple de passerelle gateway.yaml

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: info-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

   b. Appliquer le fichier YAML.

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

2. Créer un objet **VirtualService** pour réécrire l'en-tête de l'hôte.

   a. Créez un fichier YAML et copiez-y le YAML suivant.

      Exemple de service virtuel

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

```
      host: productpage
      port:
        number: 9080
```

b. Appliquer le fichier YAML.

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

3. Vérifiez que la passerelle et le service virtuel ont été définis correctement.

a. Définir l'URL de la passerelle.

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

b. Définissez le numéro de port. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

```
export TARGET_PORT=$(oc -n istio-system get route istio-ingressgateway -o
jsonpath='{.spec.port.targetPort}')
```

c. Tester une page qui a été explicitement exposée.

```
curl -s -I "$GATEWAY_URL/productpage"
```

Le résultat attendu est **200**.

1.14.2. Comprendre les itinéraires automatiques

Les routes OpenShift pour les passerelles sont automatiquement gérées dans Service Mesh. Chaque fois qu'une passerelle Istio est créée, mise à jour ou supprimée dans le Service Mesh, une route OpenShift est créée, mise à jour ou supprimée.

1.14.2.1. Routes avec sous-domaines

Red Hat OpenShift Service Mesh crée la route avec le sous-domaine, mais OpenShift Container Platform doit être configuré pour l'activer. Les sous-domaines, par exemple ***.domain.com**, sont pris en charge, mais pas par défaut. Configurez une politique de wildcard OpenShift Container Platform avant de configurer une passerelle d'hôte wildcard.

Pour plus d'informations, voir Utilisation d'itinéraires avec caractères génériques .

1.14.2.2. Création d'itinéraires de sous-domaines

L'exemple suivant crée une passerelle dans l'application d'exemple Bookinfo, qui crée des itinéraires de sous-domaine.

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: gateway1
spec:
  selector:
```

```
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - www.info.com
    - info.example.com
```

La ressource **Gateway** crée les routes OpenShift suivantes. Vous pouvez vérifier que les routes sont créées en utilisant la commande suivante. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

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

Résultats attendus

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

Si vous supprimez la passerelle, Red Hat OpenShift Service Mesh supprime les itinéraires. Cependant, les routes que vous avez créées manuellement ne sont jamais modifiées par Red Hat OpenShift Service Mesh.

1.14.2.3. Etiquettes d'itinéraire et annotations

Parfois, des étiquettes ou des annotations spécifiques sont nécessaires dans une route OpenShift. Par exemple, certaines fonctionnalités avancées des routes OpenShift sont gérées à l'aide d'annotations spéciales. Voir "Annotations spécifiques aux routes" dans la section suivante "Ressources supplémentaires".

Pour ce cas et d'autres cas d'utilisation, Red Hat OpenShift Service Mesh copiera toutes les étiquettes et annotations présentes dans la ressource de passerelle Istio (à l'exception des annotations commençant par **kubectl.kubernetes.io**) dans la ressource de route OpenShift gérée.

Si vous avez besoin de labels ou d'annotations spécifiques dans les routes OpenShift créées par Service Mesh, créez-les dans la ressource Istio gateway et ils seront copiés dans les ressources OpenShift route gérées par Service Mesh.

Ressources supplémentaires

- Annotations spécifiques à l'itinéraire .

1.14.2.4. Désactivation de la création automatique d'itinéraires

Par défaut, la ressource **ServiceMeshControlPlane** synchronise automatiquement les ressources de la passerelle Istio avec les routes OpenShift. Désactiver la création automatique de routes vous donne plus de flexibilité pour contrôler les routes si vous avez un cas particulier ou si vous préférez contrôler les routes manuellement.

1.14.2.4.1. Désactivation de la création automatique d'itinéraires dans certains cas

Si vous souhaitez désactiver la gestion automatique des routes OpenShift pour une passerelle Istio spécifique, vous devez ajouter l'annotation **maistra.io/manageRoute: false** à la définition des métadonnées de la passerelle. Red Hat OpenShift Service Mesh ignorera les passerelles Istio avec cette annotation, tout en conservant la gestion automatique des autres passerelles Istio.

1.14.2.4.2. Désactivation de la création automatique d'itinéraires dans tous les cas

Vous pouvez désactiver la gestion automatique des routes OpenShift pour toutes les passerelles de votre maillage.

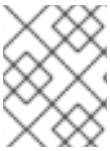
Désactivez l'intégration entre les passerelles Istio et les routes OpenShift en définissant le champ **ServiceMeshControlPlane gateways.openshiftRoute.enabled** sur **false**. Par exemple, voir l'extrait de ressource suivant.

```
apiVersion: maistra.io/v1alpha1
kind: ServiceMeshControlPlane
metadata:
  namespace: istio-system
spec:
  gateways:
    openshiftRoute:
      enabled: false
```

1.14.3. Comprendre les entrées de service

Une entrée de service ajoute une entrée au registre de service que Red Hat OpenShift Service Mesh maintient en interne. Après avoir ajouté l'entrée de service, les proxys Envoy envoient du trafic au service comme s'il s'agissait d'un service dans votre maillage. Les entrées de service vous permettent d'effectuer les opérations suivantes :

- Gérer le trafic pour les services qui s'exécutent en dehors du maillage de services.
- Redirection et transfert du trafic vers des destinations externes (telles que les API consommées sur le web) ou vers des services de l'infrastructure existante.
- Définir des politiques de relance, de temporisation et d'injection d'erreur pour les destinations externes.
- Exécutez un service de maillage dans une machine virtuelle (VM) en ajoutant des VM à votre maillage.



NOTE

Ajoutez des services d'un cluster différent au maillage pour configurer un maillage multicluster Red Hat OpenShift Service Mesh sur Kubernetes.

Exemples de saisie de services

L'exemple suivant est une entrée de service mesh-external qui ajoute la dépendance externe **ext-resource** au registre de services Red Hat OpenShift Service Mesh :

```
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
```

Spécifiez la ressource externe à l'aide du champ **hosts**. Vous pouvez le qualifier complètement ou utiliser un nom de domaine préfixé par un joker.

Vous pouvez configurer des services virtuels et des règles de destination pour contrôler le trafic vers une entrée de service de la même manière que vous configurez le trafic pour tout autre service dans le maillage. Par exemple, la règle de destination suivante configure la route du trafic pour utiliser TLS mutuel afin de sécuriser la connexion au service externe **ext-svc.example.com** qui est configuré à l'aide de l'entrée de service :

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

1.14.4. Utilisation des services virtuels

Vous pouvez acheminer des requêtes dynamiquement vers plusieurs versions d'un microservice via Red Hat OpenShift Service Mesh avec un service virtuel. Avec les services virtuels, vous pouvez :

- Adresser plusieurs services d'application par le biais d'un seul service virtuel. Si votre maillage utilise Kubernetes, par exemple, vous pouvez configurer un service virtuel pour gérer tous les services dans un espace de noms spécifique. Un service virtuel vous permet de transformer une application monolithique en un service composé de microservices distincts avec une expérience consommateur transparente.

- Configurer des règles de trafic en combinaison avec des passerelles pour contrôler le trafic entrant et sortant.

1.14.4.1. Configuration des services virtuels

Les demandes sont acheminées vers les services au sein d'un maillage de services virtuels. Chaque service virtuel consiste en un ensemble de règles de routage qui sont évaluées dans l'ordre. Red Hat OpenShift Service Mesh fait correspondre chaque demande donnée au service virtuel à une destination réelle spécifique au sein du maillage.

Sans services virtuels, Red Hat OpenShift Service Mesh distribue le trafic en utilisant l'équilibrage de charge des moindres demandes entre toutes les instances de service. Avec un service virtuel, vous

pouvez spécifier le comportement du trafic pour un ou plusieurs noms d'hôte. Les règles de routage dans le service virtuel indiquent à Red Hat OpenShift Service Mesh comment envoyer le trafic pour le service virtuel vers les destinations appropriées. Les destinations de routage peuvent être des versions du même service ou des services entièrement différents.

Procédure

1. Créez un fichier YAML à l'aide de l'exemple suivant pour acheminer les requêtes vers différentes versions de l'exemple de service d'application Bookinfo en fonction de l'utilisateur qui se connecte à l'application.

   Exemple VirtualService.yaml

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

2. Exécutez la commande suivante pour appliquer **VirtualService.yaml**, où **VirtualService.yaml** est le chemin d'accès au fichier.

   ```
   $ oc apply -f <VirtualService.yaml>
   ```

1.14.4.2. Référence de configuration du service virtuel

Paramètres	Description
`spec:` `  hosts:`	Le champ **hosts** répertorie l'adresse de destination du service virtuel à laquelle les règles de routage s'appliquent. Il s'agit de l'adresse ou des adresses utilisées pour envoyer des requêtes au service. Le nom d'hôte du service virtuel peut être une adresse IP, un nom DNS ou un nom court qui se résout en un nom de domaine pleinement qualifié.

Paramètres	Description

`spec:`
`  http:`
`  - match:`
	La section **http** contient les règles de routage du service virtuel qui décrivent les conditions de correspondance et les actions pour le routage du trafic HTTP/1.1, HTTP2 et gRPC envoyé à la destination spécifiée dans le champ hosts. Une règle de routage se compose de la destination où vous voulez que le trafic aille et de toutes les conditions de correspondance spécifiées. La première règle de routage de l'exemple comporte une condition qui commence par le champ "match". Dans cet exemple, ce routage s'applique à toutes les demandes de l'utilisateur **jason**. Ajoutez les champs **headers**, **end-user** et **exact** pour sélectionner les demandes appropriées.

`spec:`
`  http:`
`  - match:`
`  - destination:`
	Le champ **destination** dans la section route spécifie la destination réelle pour le trafic qui correspond à cette condition. Contrairement à l'hôte du service virtuel, l'hôte de la destination doit être une destination réelle qui existe dans le registre de service Red Hat OpenShift Service Mesh. Il peut s'agir d'un service mesh avec des proxies ou d'un service non mesh ajouté à l'aide d'une entrée de service. Dans cet exemple, le nom d'hôte est un nom de service Kubernetes :

1.14.5. Comprendre les règles de destination

Les règles de destination sont appliquées après l'évaluation des règles de routage des services virtuels, de sorte qu'elles s'appliquent à la destination réelle du trafic. Les services virtuels acheminent le trafic vers une destination. Les règles de destination configurent ce qu'il advient du trafic à cette destination.

Par défaut, Red Hat OpenShift Service Mesh utilise une politique d'équilibrage de charge des moindres demandes, où l'instance de service dans le pool avec le plus petit nombre de connexions actives reçoit la demande. Red Hat OpenShift Service Mesh prend également en charge les modèles suivants, que vous pouvez spécifier dans les règles de destination pour les requêtes vers un service ou un sous-ensemble de services particulier.

- Aléatoire : Les demandes sont transmises de manière aléatoire aux instances du pool.
- Pondéré : Les demandes sont transmises aux instances du pool en fonction d'un pourcentage spécifique.
- Moins de demandes : Les demandes sont transmises aux instances ayant le moins de demandes.

Exemple de règle de destination

L'exemple de règle de destination suivant configure trois sous-ensembles différents pour le service de destination **my-svc**, avec des politiques d'équilibrage de charge différentes :

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
```

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

1.14.6. Comprendre les politiques de réseau

Red Hat OpenShift Service Mesh crée et gère automatiquement un certain nombre de ressources **NetworkPolicies** dans le plan de contrôle Service Mesh et les espaces de noms des applications. Cela permet de s'assurer que les applications et le plan de contrôle peuvent communiquer entre eux.

Par exemple, si vous avez configuré votre cluster OpenShift Container Platform pour utiliser le plugin SDN, Red Hat OpenShift Service Mesh crée une ressource **NetworkPolicy** dans chaque projet membre. Cela permet l'entrée de tous les pods dans le maillage à partir des autres membres du maillage et du plan de contrôle. Cela limite également l'entrée aux seuls projets membres. Si vous avez besoin de l'entrée de projets non membres, vous devez créer une ressource **NetworkPolicy** pour autoriser ce trafic. Si vous supprimez un espace de noms de Service Mesh, cette ressource **NetworkPolicy** est supprimée du projet.

1.14.6.1. Désactivation de la création automatique de NetworkPolicy

Si vous souhaitez désactiver la création et la gestion automatiques des ressources **NetworkPolicy**, par exemple pour appliquer les politiques de sécurité de l'entreprise ou pour permettre un accès direct aux pods dans le maillage, vous pouvez le faire. Vous pouvez modifier le site **ServiceMeshControlPlane** et remplacer **spec.security.manageNetworkPolicy** par **false**.

NOTE

Lorsque vous désactivez **spec.security.manageNetworkPolicy**, Red Hat OpenShift Service Mesh ne créera pas d'objets **any NetworkPolicy** . L'administrateur système est responsable de la gestion du réseau et de la résolution des problèmes que cela pourrait causer.

Conditions préalables

- Red Hat OpenShift Service Mesh Operator version 2.1.1 ou supérieure installée.
- **ServiceMeshControlPlane** mis à jour à la version 2.1 ou supérieure.

Procédure

1. Dans la console web d'OpenShift Container Platform, cliquez sur **Operators → Installed Operators**.
2. Sélectionnez le projet dans lequel vous avez installé le plan de contrôle Service Mesh, par exemple **istio-system**, dans le menu **Project**.
3. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh. Dans la colonne **Istio Service Mesh Control Plane**, cliquez sur le nom de votre **ServiceMeshControlPlane**, par exemple **basic-install**.
4. Sur la page **Create ServiceMeshControlPlane Details**, cliquez sur **YAML** pour modifier votre configuration.
5. Définissez le champ **ServiceMeshControlPlane spec.security.manageNetworkPolicy** sur **false**, comme indiqué dans cet exemple.

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

6. Cliquez sur **Save**.

1.14.7. Configuration des side-cars pour la gestion du trafic

Par défaut, Red Hat OpenShift Service Mesh configure chaque proxy Envoy pour accepter le trafic sur tous les ports de sa charge de travail associée, et pour atteindre chaque charge de travail dans le maillage lors de la transmission du trafic. Vous pouvez utiliser une configuration sidecar pour effectuer les opérations suivantes :

- Affiner l'ensemble des ports et des protocoles acceptés par un proxy Envoy.
- Limiter l'ensemble des services que le proxy Envoy peut atteindre.

NOTE

Pour optimiser les performances de votre maillage de services, envisagez de limiter les configurations de proxy Envoy.

Dans l'application d'exemple Bookinfo, configurez un Sidecar pour que tous les services puissent atteindre d'autres services fonctionnant dans le même espace de noms et le même plan de contrôle. Cette configuration Sidecar est nécessaire pour utiliser les fonctionnalités de politique et de télémétrie de Red Hat OpenShift Service Mesh.

Procédure

1. Créez un fichier YAML à l'aide de l'exemple suivant pour spécifier que vous souhaitez qu'une configuration sidecar s'applique à toutes les charges de travail d'un espace de noms particulier. Sinon, choisissez des charges de travail spécifiques à l'aide d'une adresse **workloadSelector**.

   Exemple sidecar.yaml

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

2. Exécutez la commande suivante pour appliquer **sidecar.yaml**, où **sidecar.yaml** est le chemin d'accès au fichier.

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

3. Exécutez la commande suivante pour vérifier que le sidecar a été créé avec succès.

```
$ oc get sidecar
```

1.14.8. Tutoriel sur le routage

Ce guide fait référence à l'application Bookinfo pour fournir des exemples de routage dans une application d'exemple. Installez l'application Bookinfo pour découvrir comment fonctionnent ces exemples de routage.

1.14.8.1. Tutoriel de routage de Bookinfo

L'exemple d'application Service Mesh Bookinfo se compose de quatre microservices distincts, chacun ayant plusieurs versions. Après l'installation de l'exemple d'application Bookinfo, trois versions différentes du microservice **reviews** s'exécutent simultanément.

Lorsque vous accédez à la page de l'application Bookinfo /**product** dans un navigateur et que vous la rafraîchissez plusieurs fois, il arrive que la critique de livre contienne des étoiles et d'autres fois non. Sans version de service explicite par défaut, Service Mesh achemine les requêtes vers toutes les versions disponibles, l'une après l'autre.

Ce tutoriel vous aide à appliquer des règles qui acheminent tout le trafic vers **v1** (version 1) des microservices. Par la suite, vous pourrez appliquer une règle pour acheminer le trafic en fonction de la valeur d'un en-tête de requête HTTP.

Prérequis :

- Déployez l'application d'exemple Bookinfo pour travailler avec les exemples suivants.

1.14.8.2. Application d'un service virtuel

Dans la procédure suivante, le service virtuel achemine tout le trafic vers **v1** de chaque microservice en appliquant des services virtuels qui définissent la version par défaut des microservices.

Procédure

1. Appliquer les services virtuels.

```
$ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.3/samples/info/networking/virtual-service-all-v1.yaml
```

2. Pour vérifier que vous avez appliqué les services virtuels, affichez les itinéraires définis à l'aide de la commande suivante :

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

Cette commande renvoie une ressource de **kind: VirtualService** au format YAML.

Vous avez configuré Service Mesh pour router vers la version **v1** des microservices Bookinfo, y compris le service **reviews** version 1.

1.14.8.3. Test de la nouvelle configuration de l'itinéraire

Testez la nouvelle configuration en actualisant le site /**productpage** de l'application Bookinfo.

Procédure

1. Définissez la valeur du paramètre **GATEWAY_URL**. Vous pouvez utiliser cette variable pour trouver l'URL de votre page produit Bookinfo ultérieurement. Dans cet exemple, istio-system est le nom du projet de plan de contrôle.

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

2. Exécutez la commande suivante pour récupérer l'URL de la page produit.

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

3. Ouvrez le site Bookinfo dans votre navigateur.

La partie de la page consacrée aux avis s'affiche sans étoiles, quel que soit le nombre d'actualisations. Cela est dû au fait que vous avez configuré Service Mesh pour acheminer tout le trafic du service de commentaires vers la version **reviews:v1** et que cette version du service n'accède pas au service de classement par étoiles.

Votre maillage de services achemine désormais le trafic vers une version d'un service.

1.14.8.4. Itinéraire basé sur l'identité de l'utilisateur

Modifiez la configuration de l'itinéraire de manière à ce que tout le trafic provenant d'un utilisateur spécifique soit acheminé vers une version de service spécifique. Dans ce cas, tout le trafic provenant d'un utilisateur nommé **jason** sera acheminé vers le service **reviews:v2**.

Service Mesh n'a pas de compréhension spéciale et intégrée de l'identité de l'utilisateur. Cet exemple est rendu possible par le fait que le service **productpage** ajoute un en-tête personnalisé **end-user** à toutes les demandes HTTP sortantes adressées au service de révision.

Procédure

1. Exécutez la commande suivante pour activer le routage basé sur l'utilisateur dans l'application d'exemple Bookinfo.

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

2. Exécutez la commande suivante pour confirmer la création de la règle. Cette commande renvoie toutes les ressources de **kind: VirtualService** au format YAML.

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

3. Sur le site /**productpage** de l'application Bookinfo, connectez-vous en tant qu'utilisateur **jason** sans mot de passe.

4. Actualiser le navigateur. Les étoiles apparaissent à côté de chaque avis.

5. Connectez-vous en tant qu'autre utilisateur (choisissez le nom que vous voulez). Actualisez le navigateur. Les étoiles ont disparu. Le trafic est maintenant acheminé vers **reviews:v1** pour tous les utilisateurs sauf Jason.

Vous avez configuré avec succès l'application d'exemple Bookinfo pour acheminer le trafic en fonction de l'identité de l'utilisateur.

1.15. MESURES, JOURNAUX ET TRACES

Une fois que vous avez ajouté votre application au maillage, vous pouvez observer le flux de données à travers votre application. Si vous n'avez pas installé votre propre application, vous pouvez voir comment l'observabilité fonctionne dans Red Hat OpenShift Service Mesh en installant l'application d'exemple Bookinfo.

1.15.1. Découvrir les adresses de la console

Red Hat OpenShift Service Mesh fournit les consoles suivantes pour visualiser vos données de maillage de services :

- **Kiali console** - Kiali est la console de gestion de Red Hat OpenShift Service Mesh.
- **Jaeger console** - Jaeger est la console de gestion pour le traçage distribué de Red Hat OpenShift.
- **Grafana console** - Grafana fournit aux administrateurs de maillage des requêtes avancées, des analyses de métriques et des tableaux de bord pour les données Istio. En option, Grafana peut être utilisé pour analyser les métriques de maillage de services.
- **Prometheus console** - Red Hat OpenShift Service Mesh utilise Prometheus pour stocker les informations de télémétrie des services.

Lorsque vous installez le plan de contrôle Service Mesh, il génère automatiquement des routes pour chacun des composants installés. Une fois que vous avez l'adresse de l'itinéraire, vous pouvez accéder à la console Kiali, Jaeger, Prometheus ou Grafana pour afficher et gérer vos données de service mesh.

Prérequis

- Le composant doit être activé et installé. Par exemple, si vous n'avez pas installé le traçage distribué, vous ne pourrez pas accéder à la console Jaeger.

Procédure à partir de la console OpenShift

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur disposant des droits cluster-admin. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

2. Naviguez jusqu'à **Networking → Routes**.

3. Sur la page **Routes**, sélectionnez le projet de plan de contrôle Service Mesh, par exemple **istio-system**, dans le menu **Namespace**.
   La colonne **Location** affiche l'adresse liée à chaque itinéraire.

4. Si nécessaire, utilisez le filtre pour trouver la console du composant dont vous voulez accéder à la route. Cliquez sur l'itinéraire **Location** pour lancer la console.

5. Cliquez sur **Log In With OpenShift**

Procédure à partir du CLI

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. Passez au projet de plan de contrôle Service Mesh. Dans cet exemple, **istio-system** est le projet de plan de contrôle Service Mesh. Exécutez la commande suivante :

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

3. Pour obtenir les itinéraires des différentes consoles Red Hat OpenShift Service Mesh, exécutez la commande suivante :

```
$ oc get routes
```

   Cette commande renvoie les URL des consoles web Kiali, Jaeger, Prometheus et Grafana, ainsi que toutes les autres routes de votre maillage de services. Vous devriez obtenir une sortie similaire à la suivante :

```
NAME                   HOST/PORT                         SERVICES               PORT    TERMINATION
info-gateway           bookinfo-gateway-yourcompany.com  istio-ingressgateway   http2
grafana                grafana-yourcompany.com           grafana                <all>
reencrypt/Redirect
istio-ingressgateway   istio-ingress-yourcompany.com     istio-ingressgateway   8080
jaeger                 jaeger-yourcompany.com            jaeger-query           <all>   reencrypt
kiali                  kiali-yourcompany.com             kiali                  20001   reencrypt/Redirect
prometheus             prometheus-yourcompany.com        prometheus             <all>
reencrypt/Redirect
```

4. Copiez l'URL de la console à laquelle vous souhaitez accéder dans la colonne **HOST/PORT** dans un navigateur pour ouvrir la console.

5. Cliquez sur **Log In With OpenShift**

1.15.2. Accès à la console Kiali

Vous pouvez visualiser la topologie, la santé et les métriques de votre application dans la console Kiali. Si votre service rencontre des problèmes, la console Kiali vous permet de visualiser le flux de données à travers votre service. Vous pouvez obtenir des informations sur les composants du maillage à différents niveaux, y compris les applications abstraites, les services et les charges de travail. Kiali fournit également une vue graphique interactive de votre espace de noms en temps réel.

Pour accéder à la console Kiali, vous devez avoir Red Hat OpenShift Service Mesh installé, Kiali installé et configuré.

Le processus d'installation crée une route pour accéder à la console Kiali.

Si vous connaissez l'URL de la console Kiali, vous pouvez y accéder directement. Si vous ne connaissez pas l'URL, utilisez les instructions suivantes.

Procédure pour les administrateurs

1. Connectez-vous à la console web de OpenShift Container Platform avec un rôle d'administrateur.
2. Cliquez sur **Home → Projects**.
3. Sur la page **Projects**, si nécessaire, utilisez le filtre pour trouver le nom de votre projet.
4. Cliquez sur le nom de votre projet, par exemple **info**.
5. Sur la page **Project details**, dans la section **Launcher**, cliquez sur le lien **Kiali**.
6. Connectez-vous à la console Kiali avec le même nom d'utilisateur et le même mot de passe que ceux utilisés pour accéder à la console OpenShift Container Platform.
   Lorsque vous vous connectez pour la première fois à la console Kiali, vous voyez la page **Overview** qui affiche tous les espaces de noms de votre maillage de services que vous avez le droit de voir.

   Si vous validez l'installation de la console et que les espaces de noms n'ont pas encore été ajoutés au maillage, il se peut qu'il n'y ait pas d'autres données à afficher que **istio-system**.

Procédure pour les développeurs

1. Connectez-vous à la console web de OpenShift Container Platform avec un rôle de développeur.
2. Cliquez sur **Project**.
3. Sur la page **Project Details**, si nécessaire, utilisez le filtre pour trouver le nom de votre projet.
4. Cliquez sur le nom de votre projet, par exemple **info**.
5. Sur la page **Project**, dans la section **Launcher**, cliquez sur le lien **Kiali**.
6. Cliquez sur **Log In With OpenShift**

1.15.3. Visualisation des données de maillage des services dans la console Kiali

Le Kiali Graph offre une visualisation puissante du trafic de votre maillage. La topologie combine le trafic des requêtes en temps réel avec vos informations de configuration Istio pour présenter un aperçu immédiat du comportement de votre maillage de services, ce qui vous permet de localiser rapidement

les problèmes. Plusieurs types de graphiques vous permettent de visualiser le trafic sous forme de topologie de service de haut niveau, de topologie de charge de travail de bas niveau ou de topologie au niveau de l'application.

Plusieurs graphiques sont disponibles :

- Le site **App graph** montre une charge de travail globale pour toutes les applications qui sont étiquetées de la même manière.
- Le site **Service graph** présente un nœud pour chaque service de votre maillage, mais exclut toutes les applications et charges de travail du graphique. Il fournit une vue d'ensemble et regroupe tout le trafic pour des services définis.
- Le site **Versioned App graph** présente un nœud pour chaque version d'une application. Toutes les versions d'une application sont regroupées.
- Le site **Workload graph** présente un nœud pour chaque charge de travail dans votre maillage de services. Ce graphique ne nécessite pas l'utilisation des étiquettes d'application et de version. Si votre application n'utilise pas d'étiquettes de version, utilisez ce graphique.

Les nœuds du graphique sont agrémentés d'une variété d'informations, indiquant diverses options d'acheminement comme les services virtuels et les entrées de service, ainsi que des configurations spéciales comme l'injection de fautes et les disjoncteurs. Il peut identifier les problèmes mTLS, les problèmes de latence, le trafic d'erreur, etc. Le graphique est hautement configurable, peut afficher des animations de trafic et dispose de puissantes capacités de recherche et de masquage.

Cliquez sur le bouton **Legend** pour obtenir des informations sur les formes, les couleurs, les flèches et les badges affichés dans le graphique.

Pour afficher un résumé des métriques, sélectionnez un nœud ou une arête dans le graphique pour afficher les détails de sa métrique dans le panneau de résumé.

1.15.3.1. Modifier la disposition des graphes dans Kiali

La présentation du graphe Kiali peut varier en fonction de l'architecture de votre application et des données à afficher. Par exemple, le nombre de nœuds du graphe et leurs interactions peuvent déterminer le rendu du graphe Kiali. Parce qu'il n'est pas possible de créer une disposition unique qui rende bien dans toutes les situations, Kiali offre un choix de plusieurs dispositions différentes.

Conditions préalables

- Si vous n'avez pas installé votre propre application, installez l'application d'exemple Bookinfo. Générez ensuite du trafic pour l'application Bookinfo en entrant plusieurs fois la commande suivante.

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

  Cette commande simule un utilisateur visitant le microservice **productpage** de l'application.

Procédure

1. Lancer la console Kiali.
2. Cliquez sur **Log In With OpenShift**
3. Dans la console Kiali, cliquez sur **Graph** pour afficher un graphique de l'espace de noms.

4. Dans le menu **Namespace**, sélectionnez l'espace de noms de votre application, par exemple **info**.

5. Pour choisir une autre présentation graphique, effectuez l'une ou l'autre des opérations suivantes, ou les deux :

   - Sélectionnez différents groupes de données dans le menu situé en haut du graphique.
     - Graphique de l'application
     - Graphique des services
     - Graphique de l'application versionnée (par défaut)
     - Graphique de la charge de travail
   - Sélectionnez une autre présentation de graphique dans la légende située au bas du graphique.
     - Mise en page par défaut dagre
     - Schéma 1 cose-bilkent
     - Disposition 2 cola

1.15.3.2. Visualisation des journaux dans la console Kiali

Vous pouvez consulter les journaux de vos charges de travail dans la console Kiali. La page **Workload Detail** comprend un onglet **Logs** qui affiche une vue unifiée des journaux qui affiche à la fois les journaux d'application et de proxy. Vous pouvez sélectionner la fréquence d'actualisation de l'affichage des journaux dans Kiali.

Pour modifier le niveau de journalisation des journaux affichés dans Kiali, vous devez modifier la configuration de la journalisation pour la charge de travail ou le proxy.

Conditions préalables

- Service Mesh installé et configuré.
- Kiali installé et configuré.
- L'adresse de la console Kiali.
- Application ou exemple d'application Bookinfo ajouté au maillage.

Procédure

1. Lancer la console Kiali.
2. Cliquez sur **Log In With OpenShift**
   La page de présentation de Kiali affiche les espaces de noms qui ont été ajoutés au maillage et que vous avez le droit de visualiser.
3. Cliquez sur **Workloads**.
4. Sur la page **Workloads**, sélectionnez le projet dans le menu **Namespace**.

5. Si nécessaire, utilisez le filtre pour trouver la charge de travail dont vous souhaitez consulter les journaux. Cliquez sur la charge de travail **Name**. Par exemple, cliquez sur **ratings-v1**.

6. Sur la page **Workload Details**, cliquez sur l'onglet **Logs** pour afficher les journaux de la charge de travail.

ASTUCE

Si vous ne voyez aucune entrée de journal, il se peut que vous deviez ajuster l'intervalle de temps ou l'intervalle d'actualisation.

1.15.3.3. Visualisation des métriques dans la console Kiali

Dans la console Kiali, vous pouvez visualiser les métriques entrantes et sortantes de vos applications, charges de travail et services. Les pages de détails comprennent les onglets suivants :

- métriques des applications entrantes
- métriques de l'application sortante
- mesures de la charge de travail pour les appels entrants
- métriques de la charge de travail sortante
- métriques des services entrants

Ces onglets affichent des tableaux de bord prédéfinis, adaptés à l'application, à la charge de travail ou au niveau de service concerné. Les vues détaillées de l'application et de la charge de travail affichent des mesures de demande et de réponse telles que le volume, la durée, la taille ou le trafic TCP. La vue détaillée du service montre les mesures de demande et de réponse pour le trafic entrant uniquement.

Kiali vous permet de personnaliser les graphiques en choisissant les dimensions du graphique. Kiali peut également présenter des mesures rapportées par des mesures proxy de source ou de destination. Et pour le dépannage, Kiali peut superposer des traces sur les métriques.

Conditions préalables

- Service Mesh installé et configuré.
- Kiali installé et configuré.
- L'adresse de la console Kiali.
- (Facultatif) Le traçage distribué est installé et configuré.

Procédure

1. Lancer la console Kiali.
2. Cliquez sur **Log In With OpenShift**
   La page de présentation de Kiali affiche les espaces de noms qui ont été ajoutés au maillage et que vous avez le droit de visualiser.
3. Cliquez sur **Applications**, **Workloads** ou **Services**.

4. Sur la page **Applications**, **Workloads** ou **Services**, sélectionnez le projet dans le menu **Namespace**.

5. Si nécessaire, utilisez le filtre pour trouver l'application, la charge de travail ou le service dont vous souhaitez consulter les journaux. Cliquez sur le lien **Name**.

6. Sur la page **Application Detail**, **Workload Details** ou **Service Details**, cliquez sur l'onglet **Inbound Metrics** ou **Outbound Metrics** pour afficher les mesures.

1.15.4. Traçage distribué

Le traçage distribué est le processus de suivi de la performance des services individuels dans une application en retraçant le chemin des appels de service dans l'application. Chaque fois qu'un utilisateur effectue une action dans une application, une requête est exécutée qui peut nécessiter l'interaction de plusieurs services pour produire une réponse. Le parcours de cette requête est appelé transaction distribuée.

Red Hat OpenShift Service Mesh utilise le traçage distribué de Red Hat OpenShift pour permettre aux développeurs de visualiser les flux d'appels dans une application microservice.

1.15.4.1. Connexion d'une instance de traçage distribuée existante

Si vous disposez déjà d'une instance de plateforme de traçage distribuée Red Hat OpenShift dans OpenShift Container Platform, vous pouvez configurer votre ressource **ServiceMeshControlPlane** pour utiliser cette instance pour le traçage distribué.

Conditions préalables

- Instance de traçage distribuée Red Hat OpenShift installée et configurée.

Procédure

1. Dans la console web d'OpenShift Container Platform, cliquez sur **Operators → Installed Operators**.

2. Cliquez sur le menu **Project** et sélectionnez le projet dans lequel vous avez installé le plan de contrôle Service Mesh, par exemple **istio-system**.

3. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh. Dans la colonne **Istio Service Mesh Control Plane**, cliquez sur le nom de votre ressource **ServiceMeshControlPlane**, par exemple **basic**.

4. Ajoutez le nom de votre instance de plate-forme de traçage distribuée à l'adresse **ServiceMeshControlPlane**.

   a. Cliquez sur l'onglet **YAML**.

   b. Ajoutez le nom de votre instance de plate-forme de traçage distribuée à **spec.addons.jaeger.name** dans votre ressource **ServiceMeshControlPlane**. Dans l'exemple suivant, **distr-tracing-production** est le nom de l'instance de la plate-forme de traçage distribuée.

   Exemple de configuration de traçage distribué

```
spec:
  addons:
```

```
jaeger:
  name: distr-tracing-production
```

c. Cliquez sur **Save**.

5. Cliquez sur **Reload** pour vérifier que la ressource **ServiceMeshControlPlane** a été configurée correctement.

1.15.4.2. Réglage de la fréquence d'échantillonnage

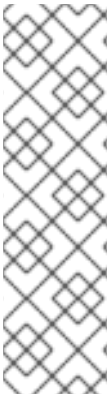
Une trace est un chemin d'exécution entre les services dans le maillage de services. Une trace est composée d'une ou plusieurs portées. Une étendue est une unité logique de travail qui a un nom, une heure de début et une durée. Le taux d'échantillonnage détermine la fréquence de persistance d'une trace.

Le taux d'échantillonnage du proxy Envoy est configuré par défaut pour échantillonner 100 % des traces dans votre maillage de services. Un taux d'échantillonnage élevé consomme les ressources et les performances du cluster, mais il est utile lors du débogage des problèmes. Avant de déployer Red Hat OpenShift Service Mesh en production, définissez la valeur à une proportion plus petite de traces. Par exemple, définissez **spec.tracing.sampling** sur **100** pour échantillonner 1 % des traces.

Configurez le taux d'échantillonnage du proxy Envoy sous la forme d'un nombre entier échelonné représentant des incréments de 0,01 %.

Dans une installation de base, **spec.tracing.sampling** est réglé sur **10000**, ce qui permet d'échantillonner 100 % des traces. Par exemple :

- En réglant la valeur sur 10, on échantillonne 0,1 % des traces.
- En réglant la valeur sur 500, on échantillonne 5 % des traces.



NOTE

Le taux d'échantillonnage du proxy Envoy s'applique aux applications qui sont disponibles pour un maillage de services et qui utilisent le proxy Envoy. Ce taux d'échantillonnage détermine la quantité de données que le proxy Envoy collecte et suit.

Le taux d'échantillonnage à distance de Jaeger s'applique aux applications qui sont externes au Service Mesh et qui n'utilisent pas le proxy Envoy, comme une base de données. Ce taux d'échantillonnage détermine la quantité de données que le système de suivi distribué collecte et stocke. Pour plus d'informations, voir Options de configuration du traçage distribué.

Procédure

1. Dans la console web d'OpenShift Container Platform, cliquez sur **Operators → Installed Operators**.
2. Cliquez sur le menu **Project** et sélectionnez le projet dans lequel vous avez installé le plan de contrôle, par exemple **istio-system**.
3. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh. Dans la colonne **Istio Service Mesh Control Plane**, cliquez sur le nom de votre ressource **ServiceMeshControlPlane**, par exemple **basic**.

4. Pour ajuster le taux d'échantillonnage, définissez une valeur différente pour **spec.tracing.sampling**.

   a. Cliquez sur l'onglet **YAML**.

   b. Définissez la valeur de **spec.tracing.sampling** dans votre ressource **ServiceMeshControlPlane**. Dans l'exemple suivant, il s'agit de **100**.

   Exemple d'échantillonnage Jaeger

   ```
   spec:
     tracing:
       sampling: 100
   ```

   c. Cliquez sur **Save**.

5. Cliquez sur **Reload** pour vérifier que la ressource **ServiceMeshControlPlane** a été configurée correctement.

1.15.5. Accéder à la console Jaeger

Pour accéder à la console Jaeger, vous devez avoir installé Red Hat OpenShift Service Mesh, Red Hat OpenShift distributed tracing platform installé et configuré.

Le processus d'installation crée une route pour accéder à la console Jaeger.

Si vous connaissez l'URL de la console Jaeger, vous pouvez y accéder directement. Si vous ne connaissez pas l'URL, suivez les instructions suivantes.

Procédure à partir de la console OpenShift

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur disposant des droits cluster-admin. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

2. Naviguez jusqu'à **Networking → Routes**.

3. Sur la page **Routes**, sélectionnez le projet de plan de contrôle Service Mesh, par exemple **istio-system**, dans le menu **Namespace**.
   La colonne **Location** affiche l'adresse liée à chaque itinéraire.

4. Si nécessaire, utilisez le filtre pour trouver la route **jaeger**. Cliquez sur la route **Location** pour lancer la console.

5. Cliquez sur **Log In With OpenShift**

Procédure à partir de la console Kiali

1. Lancer la console Kiali.

2. Cliquez sur **Distributed Tracing** dans le volet de navigation gauche.

3. Cliquez sur **Log In With OpenShift**

Procédure à partir du CLI

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. Pour demander des détails sur l'itinéraire à l'aide de la ligne de commande, entrez la commande suivante. Dans cet exemple, **istio-system** est l'espace de noms du plan de contrôle Service Mesh.

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

3. Lancez un navigateur et accédez à **https://<JAEGER_URL>**, où **<JAEGER_URL>** est l'itinéraire que vous avez découvert à l'étape précédente.

4. Connectez-vous en utilisant le même nom d'utilisateur et le même mot de passe que ceux utilisés pour accéder à la console OpenShift Container Platform.

5. Si vous avez ajouté des services au maillage de services et généré des traces, vous pouvez utiliser les filtres et le bouton **Find Traces** pour rechercher vos données de traces.
Si vous validez l'installation de la console, il n'y a pas de données de trace à afficher.

Pour plus d'informations sur la configuration de Jaeger, voir la documentation sur le traçage distribué .

1.15.6. Accéder à la console Grafana

Grafana est un outil d'analyse que vous pouvez utiliser pour afficher, interroger et analyser vos métriques de service mesh. Dans cet exemple, **istio-system** est l'espace de noms du plan de contrôle Service Mesh. Pour accéder à Grafana, procédez comme suit :

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.

2. Cliquez sur le menu **Project** et sélectionnez le projet dans lequel vous avez installé le plan de contrôle Service Mesh, par exemple **istio-system**.

3. Cliquez sur **Routes**.

4. Cliquez sur le lien dans la colonne **Location** pour la ligne **Grafana**.

5. Connectez-vous à la console Grafana avec vos identifiants OpenShift Container Platform.

1.15.7. Accès à la console Prometheus

Prometheus est un outil de surveillance et d'alerte que vous pouvez utiliser pour collecter des données multidimensionnelles sur vos microservices. Dans cet exemple, **istio-system** est l'espace de noms du plan de contrôle Service Mesh.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.

2. Cliquez sur le menu **Project** et sélectionnez le projet dans lequel vous avez installé le plan de contrôle Service Mesh, par exemple **istio-system**.

3. Cliquez sur **Routes**.
4. Cliquez sur le lien dans la colonne **Location** pour la ligne **Prometheus**.
5. Connectez-vous à la console Prometheus avec vos identifiants OpenShift Container Platform.

1.16. PERFORMANCE ET ÉVOLUTIVITÉ

Les paramètres par défaut de **ServiceMeshControlPlane** ne sont pas destinés à une utilisation en production ; ils sont conçus pour une installation réussie sur une installation par défaut d'OpenShift Container Platform, qui est un environnement aux ressources limitées. Après avoir vérifié que l'installation du SMCP a réussi, vous devez modifier les paramètres définis dans le SMCP pour les adapter à votre environnement.

1.16.1. Limiter les ressources informatiques

Par défaut, **spec.proxy** a les paramètres **cpu: 10m** et **memory: 128M**. Si vous utilisez Pilot, **spec.runtime.components.pilot** a les mêmes valeurs par défaut.

Les paramètres de l'exemple suivant sont basés sur 1 000 services et 1 000 requêtes par seconde. Vous pouvez modifier les valeurs de **cpu** et **memory** dans la page **ServiceMeshControlPlane**.

Procédure

1. Dans la console web d'OpenShift Container Platform, cliquez sur **Operators → Installed Operators**.
2. Cliquez sur le menu **Project** et sélectionnez le projet dans lequel vous avez installé le plan de contrôle Service Mesh, par exemple **istio-system**.
3. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh. Dans la colonne **Istio Service Mesh Control Plane**, cliquez sur le nom de votre **ServiceMeshControlPlane**, par exemple **basic**.
4. Ajoutez le nom de votre instance Jaeger autonome à l'adresse **ServiceMeshControlPlane**.
   a. Cliquez sur l'onglet **YAML**.
   b. Définissez les valeurs pour **spec.proxy.runtime.container.resources.requests.cpu** et **spec.proxy.runtime.container.resources.requests.memory** dans votre ressource **ServiceMeshControlPlane**.

   Exemple version 2.3 ServiceMeshControlPlane

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.3
  proxy:
    runtime:
      container:
        resources:
          requests:
```

```
          cpu: 600m
          memory: 50Mi
        limits: {}

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

c. Cliquez sur **Save**.

5. Cliquez sur **Reload** pour vérifier que la ressource **ServiceMeshControlPlane** a été configurée correctement.

1.16.2. Résultats de l'essai de charge

Le maillage des tests de charge de la communauté Istio en amont se compose de **1000** services et de **2000** sidecars avec 70 000 requêtes par seconde à l'échelle du maillage. L'exécution des tests avec Istio 1.12.3 a généré les résultats suivants :

- Le proxy Envoy utilise **0.35 vCPU** et **40 MB memory** pour 1000 requêtes par seconde passant par le proxy.
- Istiod utilise **1 vCPU** et **1.5 GB** de mémoire.
- Le proxy Envoy ajoute **2.65 ms** à la latence du 90e percentile.
- L'ancien service **istio-telemetry** (désactivé par défaut dans Service Mesh 2.0) utilise **0.6 vCPU** pour 1000 requêtes par seconde à l'échelle de la maille pour les déploiements qui utilisent Mixer. Les composants du plan de données, les proxys Envoy, gèrent les données qui circulent dans le système. Le composant du plan de contrôle du Service Mesh, Istiod, configure le plan de données. Le plan de données et le plan de contrôle ont des préoccupations distinctes en matière de performances.

1.16.2.1. Service Mesh Performance du plan de contrôle

Istiod configure les proxies sidecar en fonction des fichiers de configuration rédigés par l'utilisateur et de l'état actuel du système. Dans un environnement Kubernetes, les définitions de ressources personnalisées (CRD) et les déploiements constituent la configuration et l'état du système. Les objets de configuration Istio, tels que les passerelles et les services virtuels, fournissent la configuration créée par l'utilisateur. Pour produire la configuration des proxies, Istiod traite la configuration et l'état du système combinés de l'environnement Kubernetes et de la configuration créée par l'utilisateur.

Le plan de contrôle du Service Mesh prend en charge des milliers de services, répartis sur des milliers de pods avec un nombre similaire de services virtuels créés par l'utilisateur et d'autres objets de configuration. Les besoins en CPU et en mémoire d'Istiod augmentent avec le nombre de configurations et d'états possibles du système. La consommation de CPU varie en fonction des facteurs suivants :

- Le taux de déploiement change.

- Le taux de changement de configuration.
- Le nombre de proxies se connectant à Istiod.

Toutefois, cette partie est intrinsèquement évolutive sur le plan horizontal.

1.16.2.2. Performance du plan de données

Les performances du plan de données dépendent de nombreux facteurs, par exemple :

- Nombre de connexions de clients
- Taux de demande cible
- Taille de la demande et taille de la réponse
- Nombre de threads de l'agent proxy
- Protocol
- Cœurs de l'unité centrale
- Nombre et types de filtres proxy, en particulier les filtres liés à la télémétrie v2.

La latence, le débit et la consommation de CPU et de mémoire des proxys sont mesurés en fonction de ces facteurs.

1.16.2.2.1. Consommation de l'unité centrale et de la mémoire

Comme le proxy sidecar effectue un travail supplémentaire sur le chemin des données, il consomme de l'unité centrale et de la mémoire. Depuis Istio 1.12.3, un proxy consomme environ 0,5 vCPU pour 1000 requêtes par seconde.

La consommation de mémoire du proxy dépend de l'ensemble de l'état de configuration qu'il contient. Un grand nombre d'auditeurs, de clusters et d'itinéraires peut augmenter la consommation de mémoire.

Étant donné que le proxy ne met normalement pas en mémoire tampon les données qui transitent, le taux de requêtes n'a pas d'incidence sur la consommation de mémoire.

1.16.2.2.2. Temps de latence supplémentaire

Étant donné qu'Istio injecte un proxy sidecar sur le chemin des données, la latence est une considération importante. Istio ajoute un filtre d'authentification, un filtre de télémétrie et un filtre d'échange de métadonnées au proxy. Chaque filtre supplémentaire augmente la longueur du chemin à l'intérieur du proxy et affecte la latence.

Le proxy Envoy collecte des données télémétriques brutes après l'envoi d'une réponse au client. Le temps passé à collecter les données télémétriques brutes pour une demande ne contribue pas au temps total nécessaire pour traiter cette demande. Cependant, comme le travailleur est occupé à traiter la demande, il ne commencera pas à traiter la demande suivante immédiatement. Ce processus ajoute au temps d'attente dans la file d'attente de la demande suivante et affecte les temps de latence moyens et de queue. La latence de queue réelle dépend du modèle de trafic.

À l'intérieur du maillage, une requête traverse le proxy côté client, puis le proxy côté serveur. Dans la configuration par défaut d'Istio 1.12.3 (c'est-à-dire Istio avec télémétrie v2), les deux proxys ajoutent environ 1,7 ms et 2,7 ms à la latence du 90e et du 99e percentile, respectivement, par rapport à la

latence de base du plan de données.

1.17. CONFIGURATION DE SERVICE MESH POUR LA PRODUCTION

Lorsque vous êtes prêt à passer d'une installation de base à la production, vous devez configurer votre plan de contrôle, votre traçage et vos certificats de sécurité pour répondre aux exigences de la production.

Conditions préalables

- Installer et configurer Red Hat OpenShift Service Mesh.
- Testez votre configuration dans un environnement d'essai.

1.17.1. Configuration de la ressource ServiceMeshControlPlane pour la production

Si vous avez installé une ressource de base **ServiceMeshControlPlane** pour tester Service Mesh, vous devez la configurer selon les spécifications de production avant d'utiliser Red Hat OpenShift Service Mesh en production.

Vous ne pouvez pas modifier le champ **metadata.name** d'une ressource **ServiceMeshControlPlane** existante. Pour les déploiements en production, vous devez personnaliser le modèle par défaut.

Procédure

1. Configurer la plateforme de traçage distribuée pour la production.
   a. Modifiez la ressource **ServiceMeshControlPlane** pour utiliser la stratégie de déploiement **production**, en définissant **spec.addons.jaeger.install.storage.type** sur **Elasticsearch** et en spécifiant des options de configuration supplémentaires sous **install**. Vous pouvez créer et configurer votre instance Jaeger et définir **spec.addons.jaeger.name** comme étant le nom de l'instance Jaeger.

Paramètres par défaut de Jaeger, y compris Elasticsearch

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.3
  tracing:
    sampling: 100
    type: Jaeger
  addons:
    jaeger:
      name: MyJaeger
      install:
        storage:
          type: Elasticsearch
        ingress:
          enabled: true
  runtime:
    components:
```

```
    tracing.jaeger.elasticsearch: # only supports resources and image name
      container:
        resources: {}
```

b. Configurez le taux d'échantillonnage pour la production. Pour plus d'informations, voir la section Performances et évolutivité.

2. Assurez-vous que vos certificats de sécurité sont prêts pour la production en installant des certificats de sécurité provenant d'une autorité de certification externe. Pour plus d'informations, voir la section Sécurité.

3. Vérifiez les résultats. Entrez la commande suivante pour vérifier que la ressource **ServiceMeshControlPlane** a été correctement mise à jour. Dans cet exemple, **basic** est le nom de la ressource **ServiceMeshControlPlane**.

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

1.17.2. Ressources supplémentaires

- Pour plus d'informations sur l'optimisation des performances de Service Mesh, voir Performances et évolutivité.

1.18. CONNEXION DES MAILLES DE SERVICE

Federation est un modèle de déploiement qui vous permet de partager des services et des charges de travail entre des mailles séparées gérées dans des domaines administratifs distincts.

1.18.1. Vue d'ensemble de la Fédération

La fédération est un ensemble de fonctions qui vous permettent de connecter des services entre des mailles distinctes, ce qui permet d'utiliser les fonctions du Service Mesh telles que l'authentification, l'autorisation et la gestion du trafic dans plusieurs domaines administratifs distincts.

La mise en œuvre d'un maillage fédéré vous permet d'exécuter, de gérer et d'observer un maillage de services unique fonctionnant sur plusieurs clusters OpenShift. Red Hat OpenShift Service Mesh federation adopte une approche fondée sur l'opinion pour une mise en œuvre multi-clusters de Service Mesh qui suppose *minimal* la confiance entre les mailles.

La fédération de mailles de services suppose que chaque maille est gérée individuellement et conserve son propre administrateur. Par défaut, aucune communication n'est autorisée et aucune information n'est partagée entre les mailles. Le partage d'informations entre les mailles se fait sur la base d'un consentement explicite. Rien n'est partagé dans un maillage fédéré s'il n'a pas été configuré pour le partage. Les fonctions de support telles que la génération de certificats, les métriques et la collecte de traces restent locales dans leurs mailles respectives.

Vous configurez le site **ServiceMeshControlPlane** sur chaque maille de service pour créer des passerelles d'entrée et de sortie spécifiques à la fédération et pour spécifier le domaine de confiance pour la maille.

La fédération implique également la création de fichiers de fédération supplémentaires. Les ressources suivantes sont utilisées pour configurer la fédération entre deux ou plusieurs mailles.

- Une ressource **ServiceMeshPeer** déclare la fédération entre une paire de mailles de services.

- Une ressource **ExportedServiceSet** déclare qu'un ou plusieurs services du réseau maillé sont disponibles pour être utilisés par un réseau maillé homologue.
- Une ressource **ImportedServiceSet** indique quels services exportés par une maille homologue seront importés dans la maille.

1.18.2. Caractéristiques de la Fédération

Les caractéristiques de l'approche fédérée de Red Hat OpenShift Service Mesh pour rejoindre les maillages sont les suivantes :

- Prend en charge les certificats racine communs pour chaque maille.
- Prend en charge des certificats racine différents pour chaque maille.
- Les administrateurs de maillage doivent configurer manuellement les chaînes de certificats, les points d'extrémité de découverte de services, les domaines de confiance, etc. pour les mailles situées en dehors du maillage fédéré.
- N'exportez/importez que les services que vous souhaitez partager entre les mailles.
    - Par défaut, les informations sur les charges de travail déployées ne sont pas partagées avec les autres mailles de la fédération. Un service peut être **exported** pour le rendre visible à d'autres mailles et permettre des requêtes de charges de travail en dehors de sa propre maille.
    - Un service qui a été exporté peut être **imported** vers une autre maille, ce qui permet aux charges de travail de cette maille d'envoyer des demandes au service importé.
- Cryptage permanent des communications entre les mailles.
- Permet de configurer l'équilibrage de la charge entre les charges de travail déployées localement et les charges de travail déployées dans une autre maille de la fédération.

Lorsqu'une maille est reliée à une autre maille, elle peut effectuer les opérations suivantes :

- Fournir au réseau fédéré des informations de confiance le concernant.
- Découvrez les informations de confiance sur le maillage fédéré.
- Fournir des informations au maillage fédéré sur ses propres services exportés.
- Découvrir des informations sur les services exportés par le maillage fédéré.

1.18.3. Sécurité de la Fédération

La fédération Red Hat OpenShift Service Mesh adopte une approche fondée sur l'opinion pour une mise en œuvre multi-clusters de Service Mesh qui suppose une confiance minimale entre les mailles. La sécurité des données est intégrée dans les fonctionnalités de la fédération.

- Chaque maille est considérée comme un locataire unique, avec une administration unique.
- Vous créez un domaine de confiance unique pour chaque maille de la fédération.
- Le trafic entre les mailles fédérées est automatiquement crypté à l'aide de la sécurité mutuelle de la couche transport (mTLS).

- Le graphique de Kiali n'affiche que votre maillage et les services que vous avez importés. Vous ne pouvez pas voir les autres mailles ou services qui n'ont pas été importés dans votre maille.

1.18.4. Limites de la Fédération

L'approche fédérée de Red Hat OpenShift Service Mesh pour joindre les mailles a les limitations suivantes :

- La fédération de maillages n'est pas prise en charge sur OpenShift Dedicated.

1.18.5. Conditions préalables de la Fédération

L'approche fédérée de Red Hat OpenShift Service Mesh pour joindre les mailles a les prérequis suivants :

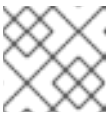
- Deux ou plusieurs clusters OpenShift Container Platform 4.6 ou supérieur.
- La fédération a été introduite dans Red Hat OpenShift Service Mesh 2.1 ou une version ultérieure. Vous devez avoir l'opérateur Red Hat OpenShift Service Mesh 2.1 ou ultérieur installé sur chaque maille que vous souhaitez fédérer.
- Vous devez avoir une version 2.1 ou plus récente de **ServiceMeshControlPlane** déployée sur chaque maille que vous voulez fédérer.
- Vous devez configurer les répartiteurs de charge prenant en charge les services associés aux passerelles de fédération pour qu'ils prennent en charge le trafic TLS brut. Le trafic de la fédération se compose de HTTPS pour la découverte et de TCP crypté brut pour le trafic de service.
- Les services que vous souhaitez exposer à un autre maillage doivent être déployés avant de pouvoir les exporter et les importer. Il ne s'agit toutefois pas d'une exigence stricte. Vous pouvez spécifier des noms de services qui n'existent pas encore pour l'exportation/importation. Lorsque vous déployez les services nommés dans les adresses **ExportedServiceSet** et **ImportedServiceSet**, ils sont automatiquement mis à disposition pour l'exportation/l'importation.

1.18.6. Planification de votre fédération de maillage

Avant de commencer à configurer votre fédération de maillage, vous devez prendre le temps de planifier votre mise en œuvre.

- Combien de mailles envisagez-vous d'intégrer dans une fédération ? Vous voudrez probablement commencer avec un nombre limité de mailles, peut-être deux ou trois.
- Quelle convention de dénomination prévoyez-vous d'utiliser pour chaque maille ? Le fait de disposer d'une convention de dénomination prédéfinie facilitera la configuration et le dépannage. Les exemples de cette documentation utilisent des couleurs différentes pour chaque maille. Vous devez décider d'une convention de dénomination qui vous aidera à déterminer qui possède et gère chaque maille, ainsi que les ressources de fédération suivantes :
  - Noms des groupes
  - Noms de réseaux de clusters
  - Noms de mailles et espaces de noms

- Passerelles d'entrée de la fédération
- Passerelles de sortie de la fédération
- Domaines de confiance en matière de sécurité



NOTE

Chaque maille de la fédération doit avoir son propre domaine de confiance.

- Quels services de chaque maillage prévoyez-vous d'exporter vers le maillage fédéré ? Chaque service peut être exporté individuellement, ou vous pouvez spécifier des étiquettes ou utiliser des caractères génériques.
  - Souhaitez-vous utiliser des alias pour les espaces de noms des services ?
  - Souhaitez-vous utiliser des alias pour les services exportés ?
- Quels sont les services exportés que chaque maille prévoit d'importer ? Chaque maille n'importe que les services dont elle a besoin.
  - Souhaitez-vous utiliser des alias pour les services importés ?

1.18.7. Fédération de maillage entre clusters

Pour connecter une instance d'OpenShift Service Mesh avec une instance fonctionnant dans un cluster différent, la procédure n'est pas très différente de celle utilisée pour connecter deux meshs déployés dans le même cluster. Cependant, la passerelle d'entrée d'un maillage doit être accessible depuis l'autre maillage. Une façon de s'en assurer est de configurer le service de passerelle comme un service **LoadBalancer** si le cluster supporte ce type de service.

Le service doit être exposé par l'intermédiaire d'un équilibreur de charge qui fonctionne à la couche 4 du modèle OSI.

1.18.7.1. Exposer l'entrée de la fédération sur des clusters fonctionnant sur du métal nu

Si le cluster fonctionne sur du métal nu et prend entièrement en charge les services **LoadBalancer**, l'adresse IP indiquée dans le champ **.status.loadBalancer.ingress.ip** de l'objet **Service** de la passerelle d'entrée doit être spécifiée comme l'une des entrées du champ **.spec.remote.addresses** de l'objet **ServiceMeshPeer**.

Si le cluster ne prend pas en charge les services **LoadBalancer**, l'utilisation d'un service **NodePort** peut être une option si les nœuds sont accessibles depuis le cluster qui exécute l'autre maille. Dans l'objet **ServiceMeshPeer**, indiquez les adresses IP des nœuds dans le champ **.spec.remote.addresses** et les ports de nœud du service dans les champs **.spec.remote.discoveryPort** et **.spec.remote.servicePort**.

1.18.7.2. Exposer l'entrée de la fédération sur des clusters fonctionnant sur IBM Power et IBM zSystems

Si le cluster fonctionne sur une infrastructure IBM Power ou IBM zSystems et prend entièrement en charge les services **LoadBalancer**, l'adresse IP figurant dans le champ **.status.loadBalancer.ingress.ip** de l'objet **Service** de la passerelle d'entrée doit être spécifiée comme l'une des entrées du champ **.spec.remote.addresses** de l'objet **ServiceMeshPeer**.

Si le cluster ne prend pas en charge les services **LoadBalancer**, l'utilisation d'un service **NodePort** peut

être une option si les nœuds sont accessibles depuis le cluster qui exécute l'autre maille. Dans l'objet **ServiceMeshPeer**, indiquez les adresses IP des nœuds dans le champ **.spec.remote.addresses** et les ports de nœud du service dans les champs **.spec.remote.discoveryPort** et **.spec.remote.servicePort**.

1.18.7.3. Exposer l'entrée de la fédération sur Amazon Web Services (AWS)

Par défaut, les services LoadBalancer dans les clusters fonctionnant sur AWS ne prennent pas en charge l'équilibrage de charge L4. Pour que la fédération Red Hat OpenShift Service Mesh fonctionne correctement, l'annotation suivante doit être ajoutée au service de passerelle d'entrée :

service.beta.kubernetes.io/aws-load-balancer-type : nlb

Le nom de domaine entièrement qualifié figurant dans le champ **.status.loadBalancer.ingress.hostname** de l'objet **Service** de la passerelle d'entrée doit être spécifié comme l'une des entrées du champ **.spec.remote.addresses** de l'objet **ServiceMeshPeer**.

1.18.7.4. Exposer l'entrée de la fédération sur Azure

Sur Microsoft Azure, il suffit de définir le type de service sur **LoadBalancer** pour que la fédération mesh fonctionne correctement.

L'adresse IP figurant dans le champ **.status.loadBalancer.ingress.ip** de l'objet "ingress gateway" (passerelle d'entrée) **Service** doit être spécifiée comme l'une des entrées du champ **.spec.remote.addresses** de l'objet **ServiceMeshPeer**.

1.18.7.5. Exposer l'entrée de la fédération sur Google Cloud Platform (GCP)

Sur Google Cloud Platform, il suffit de définir le type de service sur **LoadBalancer** pour que la fédération de maillage fonctionne correctement.

L'adresse IP figurant dans le champ **.status.loadBalancer.ingress.ip** de l'objet "ingress gateway" (passerelle d'entrée) **Service** doit être spécifiée comme l'une des entrées du champ **.spec.remote.addresses** de l'objet **ServiceMeshPeer**.

1.18.8. Liste de contrôle pour la mise en œuvre de la fédération

La fédération de maillages de services implique les activités suivantes :

- ❑ Configurez le réseau entre les clusters que vous allez fédérer.
  - ❑ Configurer les équilibreurs de charge prenant en charge les services associés aux passerelles de fédération pour prendre en charge le trafic TLS brut.
- ❑ Installation de l'opérateur Red Hat OpenShift Service Mesh version 2.1 ou ultérieure dans chacun de vos clusters.
- ❑ Déployer une version 2.1 ou ultérieure de **ServiceMeshControlPlane** sur chacun de vos clusters.
- ❑ Configurer le SMCP pour la fédération pour chaque maille que vous voulez fédérer :
  - ❑ Créez une passerelle de sortie de fédération pour chaque maille avec laquelle vous allez vous fédérer.
  - ❑ Créez une passerelle d'entrée de fédération pour chaque maille avec laquelle vous allez vous fédérer.

❑ Configurer un domaine de confiance unique.

❑ Fédérer deux ou plusieurs mailles en créant une ressource **ServiceMeshPeer** pour chaque paire de mailles.

❑ Exporter des services en créant une ressource **ExportedServiceSet** pour rendre les services disponibles d'un maillage à un maillage homologue.

❑ Importer des services en créant une ressource **ImportedServiceSet** pour importer des services partagés par un pair maillé.

1.18.9. Configuration d'un plan de contrôle Service Mesh pour la fédération

Avant de pouvoir fédérer une maille, vous devez configurer le site **ServiceMeshControlPlane** pour la fédération de mailles. Comme toutes les mailles membres de la fédération sont égales et que chaque maille est gérée indépendamment, vous devez configurer le SMCP pour la maille *each* qui participera à la fédération.

Dans l'exemple suivant, l'administrateur de **red-mesh** configure le SMCP pour la fédération avec **green-mesh** et **blue-mesh**.

Exemple de SMCP pour red-mesh

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: red-mesh
  namespace: red-mesh-system
spec:
  version: v2.3
  runtime:
    defaults:
      container:
        imagePullPolicy: Always
  gateways:
    additionalEgress:
      egress-green-mesh:
        enabled: true
        requestedNetworkView:
        - green-network
        routerMode: sni-dnat
        service:
          metadata:
            labels:
              federation.maistra.io/egress-for: egress-green-mesh
          ports:
          - port: 15443
            name: tls
          - port: 8188
            name: http-discovery  #note HTTP here
      egress-blue-mesh:
        enabled: true
        requestedNetworkView:
        - blue-network
        routerMode: sni-dnat
        service:
```

```
        metadata:
          labels:
            federation.maistra.io/egress-for: egress-blue-mesh
        ports:
        - port: 15443
          name: tls
        - port: 8188
          name: http-discovery  #note HTTP here
    additionalIngress:
      ingress-green-mesh:
        enabled: true
        routerMode: sni-dnat
        service:
          type: LoadBalancer
          metadata:
            labels:
              federation.maistra.io/ingress-for: ingress-green-mesh
          ports:
          - port: 15443
            name: tls
          - port: 8188
            name: https-discovery  #note HTTPS here
      ingress-blue-mesh:
        enabled: true
        routerMode: sni-dnat
        service:
          type: LoadBalancer
          metadata:
            labels:
              federation.maistra.io/ingress-for: ingress-blue-mesh
          ports:
          - port: 15443
            name: tls
          - port: 8188
            name: https-discovery  #note HTTPS here
    security:
      trust:
        domain: red-mesh.local
```

Tableau 1.6. Paramètres de configuration de la fédération ServiceMeshControlPlane

Paramètres	Description	Valeurs	Valeur par défaut

`spec:`
`  cluster:`
`  name:`
	Nom de la grappe. Vous n'êtes pas obligé de spécifier un nom de cluster, mais cela peut être utile pour le dépannage.	String	N/A

Paramètres	Description	Valeurs	Valeur par défaut
spec: cluster: network:	Nom du réseau du cluster. Vous n'êtes pas obligé de spécifier un nom pour le réseau, mais cela peut être utile pour la configuration et le dépannage.	String	N/A

1.18.9.1. Comprendre les passerelles de fédération

Vous utilisez un site **gateway** pour gérer le trafic entrant et sortant de votre maillage, ce qui vous permet de spécifier le trafic qui doit entrer ou sortir du maillage.

Vous utilisez des passerelles d'entrée et de sortie pour gérer le trafic entrant et sortant du maillage de services (trafic nord-sud). Lorsque vous créez un maillage fédéré, vous créez des passerelles d'entrée et de sortie supplémentaires pour faciliter la découverte des services entre les mailles fédérées, la communication entre les mailles fédérées et pour gérer le flux de trafic entre les mailles de service (trafic est-ouest).

Pour éviter les conflits de noms entre les mailles, vous devez créer des passerelles de sortie et d'entrée distinctes pour chaque maille. Par exemple, **red-mesh** aura des passerelles de sortie distinctes pour le trafic allant vers **green-mesh** et **blue-mesh**.

Tableau 1.7. Paramètres de la passerelle de la fédération

Paramètres	Description	Valeurs	Valeur par défaut
spec: gateways: additionalEgress: <egressName>:	Définir une passerelle de sortie supplémentaire pour *each* mesh peer dans la fédération.		
spec: gateways: additionalEgress: <egressName>: enabled:	Ce paramètre permet d'activer ou de désactiver la sortie de la fédération.	**true**/**false**	**true**

Paramètres	Description	Valeurs	Valeur par défaut
spec: gateways: additionalEgress: <egressName>: requestedNetwork View:	Réseaux associés aux services exportés.	Définir la valeur de **spec.cluster.network** dans le SMCP pour la maille, sinon utiliser <ServiceMeshPeer-name>-network. Par exemple, si la ressource **ServiceMeshPeer** pour cette maille est nommée **west**, le réseau sera nommé **west-network**.	
spec: gateways: additionalEgress: <egressName>: routerMode:	Le mode routeur à utiliser par la passerelle.	**sni-dnat**	
spec: gateways: additionalEgress: <egressName>: service: metadata: labels: federation.maistra.io/egress-for:	Spécifiez une étiquette unique pour la passerelle afin d'empêcher le trafic fédéré de passer par les passerelles système par défaut du cluster.		
spec: gateways: additionalEgress: <egressName>: service: ports:	Permet de spécifier les adresses **port:** et **name:** utilisées pour TLS et la découverte de services. Le trafic de la fédération consiste en un trafic de service TCP crypté brut.	Le port **15443** est nécessaire pour envoyer des requêtes de service TLS à d'autres mailles de la fédération. Le port **8188** est nécessaire pour envoyer des requêtes de découverte de service à d'autres mailles de la fédération.	

Paramètres	Description	Valeurs	Valeur par défaut
spec: gateways: additionalIngress:	Définir une passerelle d'entrée supplémentaire pour *each* mesh peer dans la fédération.		
spec: gateways: additionalIgress: <ingressName>: enabled:	Ce paramètre permet d'activer ou de désactiver l'entrée de la fédération.	**true**/**false**	**true**
spec: gateways: additionalIngress: <ingressName>: routerMode:	Le mode routeur à utiliser par la passerelle.	**sni-dnat**	
spec: gateways: additionalIngress: <ingressName>: service: type:	Le service de passerelle d'entrée doit être exposé par l'intermédiaire d'un équilibreur de charge qui fonctionne à la couche 4 du modèle OSI et qui est accessible au public.	**LoadBalancer**	
spec: gateways: additionalIngress: <ingressName>: service: type:	Si le cluster ne prend pas en charge les services **LoadBalancer**, le service de passerelle d'entrée peut être exposé par le biais d'un service **NodePort**.	**NodePort**	

Paramètres	Description	Valeurs	Valeur par défaut
spec: gateways: additionalIngress: <ingressName>: service: metadata: labels: federation.maistra.io/ingress-for:	Spécifiez une étiquette unique pour la passerelle afin d'empêcher le trafic fédéré de passer par les passerelles système par défaut du cluster.		
spec: gateways: additionalIngress: <ingressName>: service: ports:	Permet de spécifier les adresses **port:** et **name:** utilisées pour TLS et la découverte de services. Le trafic de la fédération consiste en un TCP crypté brut pour le trafic de service. Le trafic de la fédération consiste en HTTPS pour la découverte.	Le port **15443** est nécessaire pour recevoir des demandes de service TLS vers d'autres mailles de la fédération. Le port **8188** est nécessaire pour recevoir des demandes de découverte de service vers d'autres mailles de la fédération.	
spec: gateways: additionalIngress: <ingressName>: service: ports: nodePort:	Utilisé pour spécifier l'adresse **nodePort:** si le cluster ne prend pas en charge les services **LoadBalancer**.	S'il est spécifié, il est requis en plus de **port:** et **name:** pour TLS et la découverte de services. **nodePort:** doit être compris entre **30000** et**32767**.	

Dans l'exemple suivant, l'administrateur configure le SMCP pour la fédération avec le site **green-mesh** à l'aide d'un service **NodePort**.

Exemple de SMCP pour NodePort

```
gateways:
  additionalIngress:
   ingress-green-mesh:
     enabled: true
     routerMode: sni-dnat
     service:
       type: NodePort
       metadata:
```

```
    labels:
      federation.maistra.io/ingress-for: ingress-green-mesh
  ports:
  - port: 15443
    nodePort: 30510
    name: tls
  - port: 8188
    nodePort: 32359
    name: https-discovery
```

1.18.9.2. Comprendre les paramètres du domaine de confiance de la fédération

Chaque maille de la fédération doit avoir son propre domaine de confiance. Cette valeur est utilisée lors de la configuration de la fédération de mailles dans la ressource **ServiceMeshPeer**.

```
kind: ServiceMeshControlPlane
metadata:
  name: red-mesh
  namespace: red-mesh-system
spec:
  security:
    trust:
      domain: red-mesh.local
```

Tableau 1.8. Paramètres de sécurité de la fédération

Paramètres	Description	Valeurs	Valeur par défaut

`spec:`
`  security:`
`  trust:`
`  domain:`
	Utilisé pour spécifier un nom unique pour le domaine de confiance de la maille. Les domaines doivent être uniques pour chaque maille de la fédération.	**<mesh-name>.local**	N/A

Procédure à partir de la console

Suivez cette procédure pour éditer le site **ServiceMeshControlPlane** avec la console web d'OpenShift Container Platform. Cet exemple utilise le site **red-mesh** comme exemple.

1. Connectez-vous à la console web d'OpenShift Container Platform en tant qu'utilisateur ayant le rôle d'administrateur de cluster.
2. Naviguez jusqu'à **Operators → Installed Operators**.
3. Cliquez sur le menu **Project** et sélectionnez le projet dans lequel vous avez installé le plan de contrôle Service Mesh. Par exemple, **red-mesh-system**.
4. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh.
5. Dans l'onglet **Istio Service Mesh Control Plane**, cliquez sur le nom de votre **ServiceMeshControlPlane**, par exemple **red-mesh**.

6. Sur la page **Create ServiceMeshControlPlane Details**, cliquez sur **YAML** pour modifier votre configuration.

7. Modifiez votre **ServiceMeshControlPlane** pour ajouter les passerelles d'entrée et de sortie de la fédération et pour spécifier le domaine de confiance.

8. Cliquez sur **Save**.

Procédure à partir du CLI

Suivez cette procédure pour créer ou modifier le site **ServiceMeshControlPlane** à l'aide de la ligne de commande. Cet exemple utilise le site **red-mesh**.

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**. Entrez la commande suivante. Saisissez ensuite votre nom d'utilisateur et votre mot de passe lorsque vous y êtes invité.

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. Accédez au projet dans lequel vous avez installé le plan de contrôle Service Mesh, par exemple red-mesh-system.

```
$ oc project red-mesh-system
```

3. Modifiez le fichier **ServiceMeshControlPlane** pour ajouter les passerelles d'entrée et de sortie de la fédération et pour spécifier le domaine de confiance.

4. Exécutez la commande suivante pour modifier le plan de contrôle Service Mesh où **red-mesh-system** est l'espace de noms du système et **red-mesh** est le nom de l'objet **ServiceMeshControlPlane**:

```
$ oc edit -n red-mesh-system smcp red-mesh
```

5. Entrez la commande suivante, où **red-mesh-system** est l'espace de noms du système, pour voir l'état de l'installation du plan de contrôle Service Mesh.

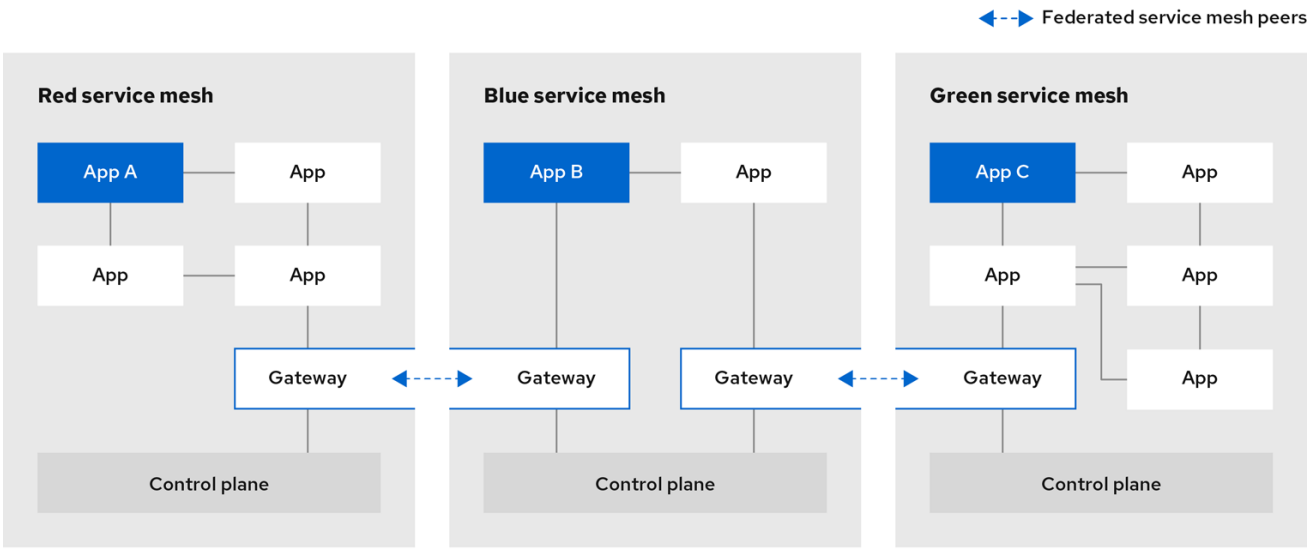
```
$ oc get smcp -n red-mesh-system
```

L'installation est terminée avec succès lorsque la colonne READY indique que tous les composants sont prêts.

```
NAME       READY   STATUS            PROFILES      VERSION   AGE
red-mesh   10/10   ComponentsReady   ["default"]   2.1.0     4m25s
```

1.18.10. Rejoindre un maillage fédéré

Vous déclarez la fédération entre deux mailles en créant une ressource **ServiceMeshPeer**. La ressource **ServiceMeshPeer** définit la fédération entre deux mailles, et vous l'utilisez pour configurer la découverte de la maille homologue, l'accès à la maille homologue, et les certificats utilisés pour valider les clients de l'autre maille.



Les mailles sont fédérées sur une base un à un, de sorte que chaque paire de pairs nécessite une paire de ressources **ServiceMeshPeer** spécifiant la connexion de fédération à l'autre maille de service. Par exemple, la fédération de deux mailles nommées **red** et **green** nécessiterait deux fichiers **ServiceMeshPeer**.

1. Sur le système de maillage rouge, créez un site **ServiceMeshPeer** pour le maillage vert.
2. Sur le système de maillage vert, créez un site **ServiceMeshPeer** pour le maillage rouge.

La fédération de trois maillages nommés **red**, **blue** et **green** nécessiterait six fichiers **ServiceMeshPeer**.

1. Sur le système de maillage rouge, créez un site **ServiceMeshPeer** pour le maillage vert.
2. Sur le système de maillage rouge, créez un site **ServiceMeshPeer** pour le maillage bleu.
3. Sur le système de maillage vert, créez un site **ServiceMeshPeer** pour le maillage rouge.
4. Sur le système de maillage vert, créez un site **ServiceMeshPeer** pour le maillage bleu.
5. Sur le système de maillage bleu, créez un site **ServiceMeshPeer** pour le maillage rouge.
6. Sur le système de maillage bleu, créez un site **ServiceMeshPeer** pour le maillage vert.

La configuration de la ressource **ServiceMeshPeer** comprend les éléments suivants :

- Adresse de la passerelle d'entrée de l'autre maille, utilisée pour la découverte et les demandes de service.
- Les noms des passerelles d'entrée et de sortie locales utilisées pour les interactions avec le réseau homologue spécifié.
- L'ID du client utilisé par l'autre maillage lorsqu'il envoie des demandes à ce maillage.
- Le domaine de confiance utilisé par l'autre maillage.
- Le nom d'un site **ConfigMap** contenant un certificat racine utilisé pour valider les certificats des clients dans le domaine de confiance utilisé par l'autre maillage.

Dans l'exemple suivant, l'administrateur de **red-mesh** configure la fédération avec **green-mesh**.

Exemple de ressource ServiceMeshPeer pour red-mesh

```
kind: ServiceMeshPeer
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  remote:
    addresses:
    - ingress-red-mesh.green-mesh-system.apps.domain.com
  gateways:
    ingress:
      name: ingress-green-mesh
    egress:
      name: egress-green-mesh
  security:
    trustDomain: green-mesh.local
    clientID: green-mesh.local/ns/green-mesh-system/sa/egress-red-mesh-service-account
    certificateChain:
      kind: ConfigMap
      name: green-mesh-ca-root-cert
```

Tableau 1.9. Paramètres de configuration de ServiceMeshPeer

Paramètres	Description	Valeurs
`metadata: name:`	Nom de la maille homologue avec laquelle cette ressource configure la fédération.	String
`metadata: namespace:`	Espace de noms du système pour ce maillage, c'est-à-dire l'endroit où le plan de contrôle du maillage de services est installé.	String
`spec: remote: addresses:`	Liste des adresses publiques des passerelles d'entrée des mailles homologues qui traitent les demandes de cette maille.	
`spec: remote: discoveryPort:`	Le port sur lequel les adresses traitent les demandes de découverte.	La valeur par défaut est 8188

Paramètres	Description	Valeurs
spec: remote: servicePort:	Le port sur lequel les adresses traitent les demandes de service.	La valeur par défaut est 15443
spec: gateways: ingress: name:	Nom de l'entrée sur ce maillage qui répond aux demandes reçues du maillage homologue. Par exemple, **ingress-green-mesh**.	
spec: gateways: egress: name:	Nom de la sortie sur ce maillage qui traite les demandes envoyées au maillage homologue. Par exemple, **egress-green-mesh**.	
spec: security: trustDomain:	Le domaine de confiance utilisé par la maille homologue.	<peerMeshName>.local
spec: security: clientID:	L'identifiant du client utilisé par la maille homologue lors d'un appel à cette maille.	<peerMeshTrustDomain>/ns/<peerMeshSystem>/sa/<peerMeshEgressGatewayName>-service-account
spec: security: certificateChain: kind: ConfigMap name:	Le type (par exemple, ConfigMap) et le nom d'une ressource contenant le certificat racine utilisé pour valider le(s) certificat(s) de client et de serveur présenté(s) à ce maillage par le maillage homologue. La clé de l'entrée de la carte de configuration contenant le certificat doit être **root-cert.pem**.	kind : ConfigMap name : <peerMesh>-ca-root-cert

1.18.10.1. Création d'une ressource ServiceMeshPeer

Conditions préalables

- Deux ou plusieurs clusters OpenShift Container Platform 4.6 ou supérieur.
- Les clusters doivent déjà être en réseau.

- Les répartiteurs de charge supportant les services associés aux passerelles de fédération doivent être configurés pour supporter le trafic TLS brut.
- Chaque cluster doit avoir une version 2.1 ou plus récente de **ServiceMeshControlPlane** configurée pour supporter la fédération déployée.
- Un compte avec le rôle **cluster-admin**.

Procédure à partir du CLI

Suivez cette procédure pour créer une ressource **ServiceMeshPeer** à partir de la ligne de commande. Cet exemple montre la création par **red-mesh** d'une ressource homologue pour **green-mesh**.

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**. Entrez la commande suivante. Saisissez ensuite votre nom d'utilisateur et votre mot de passe lorsque vous y êtes invité.

```
$ oc login --username=<NAMEOFUSER> <API token> https://<HOSTNAME>:6443
```

2. Accédez au projet dans lequel vous avez installé le plan de contrôle, par exemple, **red-mesh-system**.

```
$ oc project red-mesh-system
```

3. Créez un fichier **ServiceMeshPeer** basé sur l'exemple suivant pour les deux maillages que vous souhaitez fédérer.

Exemple de ressource ServiceMeshPeer pour le maillage rouge vers le maillage vert

```
kind: ServiceMeshPeer
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  remote:
    addresses:
    - ingress-red-mesh.green-mesh-system.apps.domain.com
  gateways:
    ingress:
      name: ingress-green-mesh
    egress:
      name: egress-green-mesh
  security:
    trustDomain: green-mesh.local
    clientID: green-mesh.local/ns/green-mesh-system/sa/egress-red-mesh-service-account
    certificateChain:
      kind: ConfigMap
      name: green-mesh-ca-root-cert
```

4. Exécutez la commande suivante pour déployer la ressource, où **red-mesh-system** est l'espace de noms du système et **servicemeshpeer.yaml** inclut un chemin complet vers le fichier que vous avez modifié :

```
$ oc create -n red-mesh-system -f servicemeshpeer.yaml
```

5. Pour confirmer que la connexion entre la maille rouge et la maille verte est établie, vérifiez l'état de la maille verte **ServiceMeshPeer** dans l'espace de noms du système de la maille rouge :

```
$ oc -n red-mesh-system get servicemeshpeer green-mesh -o yaml
```

Exemple de connexion ServiceMeshPeer entre la maille rouge et la maille verte

```
status:
  discoveryStatus:
    active:
    - pod: istiod-red-mesh-b65457658-9wq5j
      remotes:
      - connected: true
        lastConnected: "2021-10-05T13:02:25Z"
        lastFullSync: "2021-10-05T13:02:25Z"
        source: 10.128.2.149
      watch:
        connected: true
        lastConnected: "2021-10-05T13:02:55Z"
        lastDisconnectStatus: 503 Service Unavailable
        lastFullSync: "2021-10-05T13:05:43Z"
```

Le champ **status.discoveryStatus.active.remotes** indique que l'istiod de la maille homologue (dans cet exemple, la maille verte) est connecté à l'istiod de la maille courante (dans cet exemple, la maille rouge).

Le champ **status.discoveryStatus.active.watch** indique qu'istiod dans le maillage actuel est connecté à istiod dans le maillage homologue.

Si vous consultez le site **servicemeshpeer** nommé **red-mesh** dans **green-mesh-system**, vous trouverez des informations sur les deux mêmes connexions du point de vue de la maille verte.

Lorsque la connexion entre deux mailles n'est pas établie, le statut **ServiceMeshPeer** l'indique dans le champ **status.discoveryStatus.inactive**.

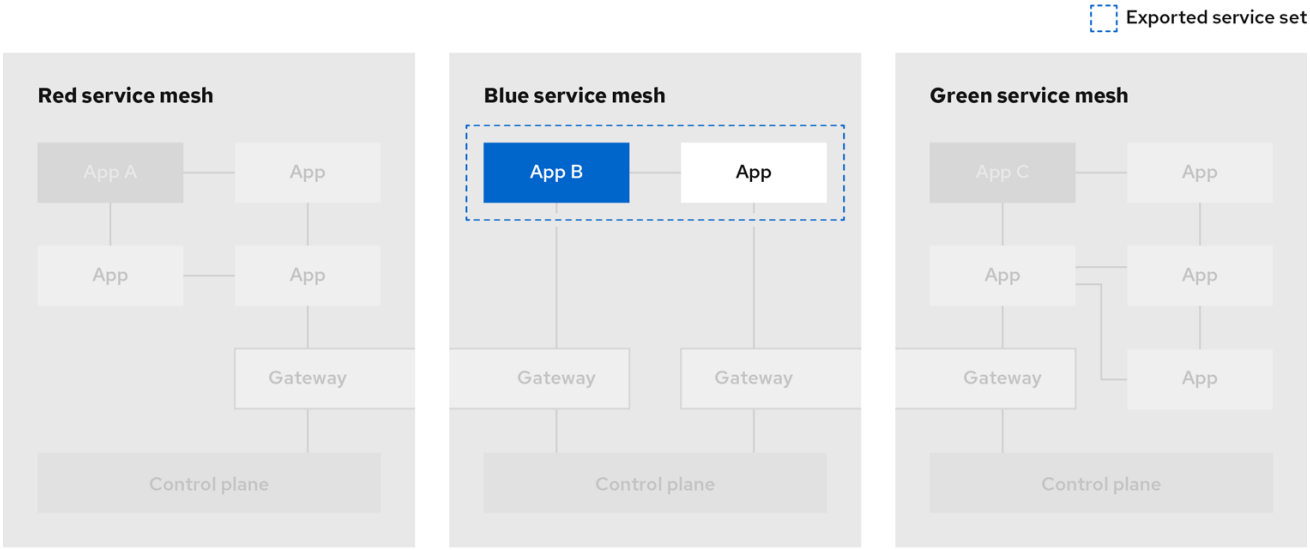
Pour plus d'informations sur la raison de l'échec d'une tentative de connexion, consultez le journal Istiod, le journal d'accès de la passerelle de sortie gérant le trafic de sortie pour le pair et la passerelle d'entrée gérant le trafic d'entrée pour le maillage actuel dans le maillage du pair.

Par exemple, si la maille rouge ne peut pas se connecter à la maille verte, vérifiez les journaux suivants :

- istiod-red-mesh dans red-mesh-system
- egress-green-mesh in red-mesh-system
- ingress-red-mesh in green-mesh-system

1.18.11. Exporter un service à partir d'un maillage fédéré

L'exportation de services permet à une maille de partager un ou plusieurs de ses services avec un autre membre de la maille fédérée.



Vous utilisez une ressource **ExportedServiceSet** pour déclarer les services d'une maille que vous mettez à la disposition d'un autre pair dans la maille fédérée. Vous devez déclarer explicitement chaque service à partager avec un pair.

- Vous pouvez sélectionner les services par espace de noms ou par nom.
- Vous pouvez utiliser des caractères génériques pour sélectionner des services, par exemple pour exporter tous les services d'un espace de noms.
- Vous pouvez exporter des services en utilisant un alias. Par exemple, vous pouvez exporter le service **foo/bar** en tant que **custom-ns/bar**.
- Vous ne pouvez exporter que des services visibles dans l'espace de noms du système de maillage. Par exemple, un service dans un autre espace de noms avec un label **networking.istio.io/exportTo** défini sur '.' ne serait pas un candidat à l'exportation.
- Pour les services exportés, les services cibles ne verront que le trafic provenant de la passerelle d'entrée, et non du demandeur initial (c'est-à-dire qu'ils ne verront pas l'identifiant du client de la passerelle de sortie de l'autre maille ou de la charge de travail à l'origine de la demande)

L'exemple suivant concerne les services que **red-mesh** exporte vers **green-mesh**.

Exemple de ressource ExportedServiceSet

```
kind: ExportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  exportRules:
  # export ratings.mesh-x-info as ratings.bookinfo
  - type: NameSelector
    nameSelector:
      namespace: red-mesh-info
      name: red-ratings
      alias:
```

```
      namespace: info
      name: ratings
  # export any service in red-mesh-info namespace with label export-service=true
  - type: LabelSelector
    labelSelector:
      namespace: red-mesh-info
      selector:
        matchLabels:
          export-service: "true"
      aliases: # export all matching services as if they were in the info namespace
      - namespace: "*"
        name: "*"
        alias:
          namespace: info
```

Tableau 1.10. Paramètres de l'ExportedServiceSet

Paramètres	Description	Valeurs
`metadata:` `  name:`	Nom du ServiceMeshPeer auquel vous exposez ce service.	Doit correspondre à la valeur **name** pour la maille dans la ressource **ServiceMeshPeer**.
`metadata:` `  namespace:`	Nom du projet/espace de noms contenant cette ressource (devrait être l'espace de noms du système pour le maillage) .	

`spec:`
`  exportRules:`
`  - type:`
	Type de règle qui régira l'exportation pour ce service. La première règle correspondante trouvée pour le service sera utilisée pour l'exportation.	**NameSelector**, **LabelSelector**

`spec:`
`  exportRules:`
`  - type: NameSelector`
`  nameSelector:`
`  namespace:`
`  name:`
	Pour créer une règle **NameSelector**, spécifiez le **namespace** du service et le **name** du service tel que défini dans la ressource **Service**.	

`spec:`
`  exportRules:`
`  - type: NameSelector`
`  nameSelector:`
`  alias:`
`  namespace:`
`  name:`
	Pour créer une règle **NameSelector** qui utilise un alias pour le service, après avoir spécifié **namespace** et **name** pour le service, spécifiez ensuite l'alias pour **namespace** et l'alias à utiliser pour **name** du service.	

Paramètres	Description	Valeurs
spec: exportRules: - type: LabelSelector labelSelector: namespace: <exportingMesh> selector: matchLabels: <labelKey>: <labelValue>	Pour créer une règle **LabelSelector**, spécifiez le **namespace** du service et spécifiez le **label** défini dans la ressource **Service**. Dans l'exemple ci-dessus, l'étiquette est **export-service**.	
spec: exportRules: - type: LabelSelector labelSelector: namespace: <exportingMesh> selector: matchLabels: <labelKey>: <labelValue> aliases: - namespace: name: alias: namespace: name:	Pour créer une règle **LabelSelector** qui utilise des alias pour les services, après avoir spécifié **selector**, spécifiez les alias à utiliser pour **name** ou **namespace** du service. Dans l'exemple ci-dessus, l'alias de l'espace de noms est **info** pour tous les services correspondants.	

Exporter les services portant le nom "ratings" de tous les espaces de noms de la maille rouge vers la maille bleue.

```
kind: ExportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: blue-mesh
  namespace: red-mesh-system
spec:
  exportRules:
  - type: NameSelector
    nameSelector:
      namespace: "*"
      name: ratings
```

Exporter tous les services de l'espace de noms west-data-center vers green-mesh

```
kind: ExportedServiceSet
apiVersion: federation.maistra.io/v1
```

```
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  exportRules:
  - type: NameSelector
    nameSelector:
      namespace: west-data-center
      name: "*"
```

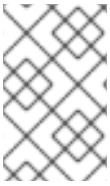
1.18.11.1. Création d'un ExportedServiceSet

Vous créez une ressource **ExportedServiceSet** pour déclarer explicitement les services que vous souhaitez mettre à la disposition d'un homologue maillé.

Les services sont exportés en tant que **<export-name>.<export-namespace>.svc.<ServiceMeshPeer.name>-exports.local** et seront automatiquement acheminés vers le service cible. Il s'agit du nom sous lequel le service exporté est connu dans le maillage d'exportation. Lorsque la passerelle d'entrée reçoit une demande destinée à ce nom, elle est acheminée vers le service exporté. Par exemple, si un service nommé **ratings.red-mesh-info** est exporté vers **green-mesh** sous le nom **ratings.bookinfo**, le service sera exporté sous le nom **ratings.bookinfo.svc.green-mesh-exports.local**, et le trafic reçu par la passerelle d'entrée pour ce nom d'hôte sera acheminé vers le service **ratings.red-mesh-bookinfo**.

Conditions préalables

- Le cluster et **ServiceMeshControlPlane** ont été configurés pour la fédération de maillage.
- Un compte avec le rôle **cluster-admin**.



NOTE

Vous pouvez configurer des services pour l'exportation même s'ils n'existent pas encore. Lorsqu'un service correspondant à la valeur spécifiée dans le ExportedServiceSet est déployé, il est automatiquement exporté.

Procédure à partir du CLI

Suivez cette procédure pour créer un site **ExportedServiceSet** à partir de la ligne de commande.

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**. Entrez la commande suivante. Saisissez ensuite votre nom d'utilisateur et votre mot de passe lorsque vous y êtes invité.

```
$ oc login --username=<NAMEOFUSER> <API token> https://<HOSTNAME>:6443
```

2. Accédez au projet dans lequel vous avez installé le plan de contrôle Service Mesh ; par exemple, **red-mesh-system**.

```
$ oc project red-mesh-system
```

3. Créez un fichier **ExportedServiceSet** basé sur l'exemple suivant où **red-mesh** exporte des services vers **green-mesh**.

Exemple de ressource ExportedServiceSet de la maille rouge à la maille verte

```
apiVersion: federation.maistra.io/v1
kind: ExportedServiceSet
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  exportRules:
  - type: NameSelector
    nameSelector:
      namespace: red-mesh-info
      name: ratings
      alias:
        namespace: info
        name: red-ratings
  - type: NameSelector
    nameSelector:
      namespace: red-mesh-info
      name: reviews
```

4. Exécutez la commande suivante pour télécharger et créer la ressource **ExportedServiceSet** dans l'espace de noms red-mesh-system.

```
$ oc create -n <ControlPlaneNamespace> -f <ExportedServiceSet.yaml>
```

Par exemple :

```
$ oc create -n red-mesh-system -f export-to-green-mesh.yaml
```

5. Créez des **ExportedServiceSets** supplémentaires si nécessaire pour chaque pair de maillage dans votre maillage fédéré.

6. Pour valider les services que vous avez exportés de **red-mesh** pour les partager avec **green-mesh**, exécutez la commande suivante :

```
$ oc get exportedserviceset <PeerMeshExportedTo> -o yaml
```

Par exemple :

```
$ oc get exportedserviceset green-mesh -o yaml
```

7. Exécutez la commande suivante pour valider les services que red-mesh exporte pour les partager avec green-mesh :

```
$ oc get exportedserviceset <PeerMeshExportedTo> -o yaml
```

Par exemple :

```
$ oc -n red-mesh-system get exportedserviceset green-mesh -o yaml
```

Exemple de validation des services exportés de la maille rouge qui sont partagés avec la maille verte.

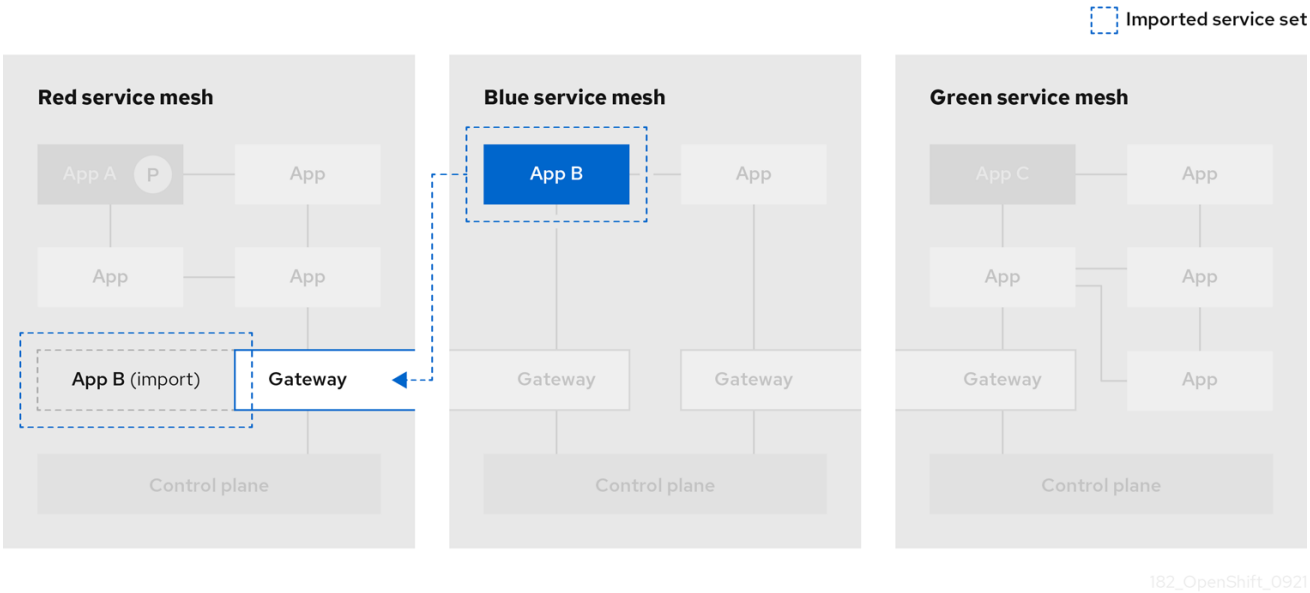
```
status:
  exportedServices:
  - exportedName: red-ratings.info.svc.green-mesh-exports.local
    localService:
      hostname: ratings.red-mesh-info.svc.cluster.local
      name: ratings
      namespace: red-mesh-info
  - exportedName: reviews.red-mesh-info.svc.green-mesh-exports.local
    localService:
      hostname: reviews.red-mesh-info.svc.cluster.local
      name: reviews
      namespace: red-mesh-info
```

Le tableau **status.exportedServices** répertorie les services qui sont actuellement exportés (ces services correspondent aux règles d'exportation de **ExportedServiceSet object**). Chaque entrée du tableau indique le nom du service exporté et des détails sur le service local exporté.

Si un service que vous vous attendiez à voir exporté est manquant, vérifiez que l'objet Service existe, que son nom ou ses étiquettes correspondent au **exportRules** défini dans l'objet **ExportedServiceSet** et que l'espace de noms de l'objet Service est configuré en tant que membre du maillage de services à l'aide de l'objet **ServiceMeshMemberRoll** ou **ServiceMeshMember**.

1.18.12. Importer un service dans un maillage fédéré

L'importation de services vous permet de spécifier explicitement quels services exportés d'un autre maillage doivent être accessibles dans votre maillage de services.



Vous utilisez une ressource **ImportedServiceSet** pour sélectionner les services à importer. Seuls les services exportés par un pair maillé et explicitement importés sont disponibles dans le maillage. Les services que vous n'importez pas explicitement ne sont pas disponibles dans le maillage.

- Vous pouvez sélectionner les services par espace de noms ou par nom.
- Vous pouvez utiliser des caractères génériques pour sélectionner des services, par exemple pour importer tous les services qui ont été exportés vers l'espace de noms.

- Vous pouvez sélectionner les services à exporter à l'aide d'un sélecteur d'étiquettes, qui peut être global pour le maillage ou limité à un espace de noms membre spécifique.
- Vous pouvez importer des services en utilisant un alias. Par exemple, vous pouvez importer le service **custom-ns/bar** en tant que **other-mesh/bar**.
- Vous pouvez spécifier un suffixe de domaine personnalisé, qui sera ajouté au **name.namespace** d'un service importé pour son nom de domaine complet ; par exemple, **bar.other-mesh.imported.local**.

L'exemple suivant concerne l'importation par **green-mesh** d'un service exporté par **red-mesh**.

Exemple ImportedServiceSet

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh #name of mesh that exported the service
  namespace: green-mesh-system #mesh namespace that service is being imported into
spec:
  importRules: # first matching rule is used
  # import ratings.info as ratings.bookinfo
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: info
      name: ratings
      alias:
        # service will be imported as ratings.info.svc.red-mesh-imports.local
        namespace: info
        name: ratings
```

Tableau 1.11. Paramètres de l'ensemble de services importé

Paramètres	Description	Valeurs
`metadata: name:`	Nom du ServiceMeshPeer qui a exporté le service vers le maillage fédéré.	
`metadata: namespace:`	Nom de l'espace de noms contenant la ressource ServiceMeshPeer (espace de noms du système maillé).	
`spec: importRules: - type:`	Type de règle qui régira l'importation du service. La première règle correspondante trouvée pour le service sera utilisée pour l'importation.	**NameSelector**

Paramètres	Description	Valeurs
spec: importRules: - type: NameSelector nameSelector: namespace: name:	Pour créer une règle **NameSelector**, spécifiez les adresses **namespace** et **name** du service exporté.	
spec: importRules: - type: NameSelector importAsLocal:	La valeur **true** permet d'agréger le point de terminaison distant avec les services locaux. Lorsque **true**, les services seront importés en tant que **<name>.<namespace>.svc.cluster.local**	**true**/**false**
spec: importRules: - type: NameSelector nameSelector: namespace: name: alias: namespace: name:	Pour créer une règle **NameSelector** qui utilise un alias pour le service, après avoir spécifié **namespace** et **name** pour le service, spécifiez ensuite l'alias pour **namespace** et l'alias à utiliser pour **name** du service.	

Importer le service "info/ratings" du maillage rouge dans le maillage bleu

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh
  namespace: blue-mesh-system
spec:
  importRules:
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: info
      name: ratings
```

Importer tous les services de l'espace de noms west-data-center du red-mesh dans le green-mesh. Ces services seront accessibles en tant que <name>.west-data-center.svc.red-mesh-imports.local

```
kind: ImportedServiceSet
```

```
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh
  namespace: green-mesh-system
spec:
  importRules:
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: west-data-center
      name: "*"
```

1.18.12.1. Création d'un ImportedServiceSet

Vous créez une ressource **ImportedServiceSet** pour déclarer explicitement les services que vous souhaitez importer dans votre maillage.

Les services sont importés avec le nom **<exported-name>.<exported-namespace>.svc.<ServiceMeshPeer.name>.remote** qui est un service "caché", visible uniquement dans l'espace de noms de la passerelle egress et associé au nom d'hôte du service exporté. Le service sera disponible localement sous le nom **<export-name>.<export-namespace>.<domainSuffix>**, où **domainSuffix** est **svc.<ServiceMeshPeer.name>-imports.local** par défaut, sauf si **importAsLocal** est défini à **true**, auquel cas **domainSuffix** est **svc.cluster.local**. Si **importAsLocal** est défini à **false**, le suffixe de domaine dans la règle d'importation sera appliqué. Vous pouvez traiter l'importation locale comme n'importe quel autre service dans le maillage. Il passe automatiquement par la passerelle de sortie, où il est redirigé vers le nom distant du service exporté.

Conditions préalables

- Le cluster et **ServiceMeshControlPlane** ont été configurés pour la fédération de maillage.
- Un compte avec le rôle **cluster-admin**.

NOTE

Vous pouvez configurer des services pour l'importation même s'ils n'ont pas encore été exportés. Lorsqu'un service correspondant à la valeur spécifiée dans ImportedServiceSet est déployé et exporté, il est automatiquement importé.

Procédure à partir du CLI

Suivez cette procédure pour créer un site **ImportedServiceSet** à partir de la ligne de commande.

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**. Entrez la commande suivante. Saisissez ensuite votre nom d'utilisateur et votre mot de passe lorsque vous y êtes invité.

```
$ oc login --username=<NAMEOFUSER> <API token> https://<HOSTNAME>:6443
```

2. Accédez au projet dans lequel vous avez installé le plan de contrôle Service Mesh ; par exemple, **green-mesh-system**.

```
$ oc project green-mesh-system
```

3. Créez un fichier **ImportedServiceSet** basé sur l'exemple suivant où **green-mesh** importe des services précédemment exportés par **red-mesh**.

   Exemple de ressource ImportedServiceSet de la maille rouge à la maille verte

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh
  namespace: green-mesh-system
spec:
  importRules:
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: info
      name: red-ratings
      alias:
        namespace: info
        name: ratings
```

4. Exécutez la commande suivante pour télécharger et créer la ressource **ImportedServiceSet** dans l'espace de noms green-mesh-system.

```
$ oc create -n <ControlPlaneNamespace> -f <ImportedServiceSet.yaml>
```

   Par exemple :

```
$ oc create -n green-mesh-system -f import-from-red-mesh.yaml
```

5. Créez des ressources **ImportedServiceSet** supplémentaires si nécessaire pour chaque pair de maillage dans votre maillage fédéré.

6. Pour valider les services que vous avez importés dans **green-mesh**, exécutez la commande suivante :

```
$ oc get importedserviceset <PeerMeshImportedInto> -o yaml
```

   Par exemple :

```
$ oc get importedserviceset green-mesh -o yaml
```

7. Exécutez la commande suivante pour valider les services importés dans un maillage.

```
$ oc get importedserviceset <PeerMeshImportedInto> -o yaml
```

   Exemple de validation de l'importation dans la maille verte des services exportés de la maille rouge à l'aide de la section status de l'espace de noms importedserviceset/red-mesh' object in the 'green-mesh-system:

```
$ oc -n green-mesh-system get importedserviceset/red-mesh -o yaml
```

```
status:
  importedServices:
  - exportedName: red-ratings.info.svc.green-mesh-exports.local
    localService:
      hostname: ratings.info.svc.red-mesh-imports.local
      name: ratings
      namespace: info
  - exportedName: reviews.red-mesh-info.svc.green-mesh-exports.local
    localService:
      hostname: ""
      name: ""
      namespace: ""
```

Dans l'exemple précédent, seul le service d'évaluation est importé, comme l'indiquent les champs remplis sous **localService**. Le service de critiques est disponible pour l'importation, mais il n'est pas importé pour l'instant parce qu'il ne correspond à aucun **importRules** dans l'objet **ImportedServiceSet**.

1.18.13. Configuration d'un maillage fédéré pour le basculement

Le basculement est la capacité de basculer automatiquement et de manière transparente vers un système de sauvegarde fiable, par exemple un autre serveur. Dans le cas d'un maillage fédéré, vous pouvez configurer un service dans un maillage pour qu'il bascule vers un service dans un autre maillage.

Vous configurez la Fédération pour le basculement en définissant les paramètres **importAsLocal** et **locality** dans une ressource **ImportedServiceSet**, puis en configurant une ressource **DestinationRule** qui configure le basculement du service vers la localité spécifiée dans la ressource **ImportedServiceSet**.

Conditions préalables

- Deux ou plusieurs clusters OpenShift Container Platform 4.6 ou plus, déjà mis en réseau et fédérés.
- **ExportedServiceSet** déjà créées pour chaque pair de maillage dans le maillage fédéré.
- **ImportedServiceSet** déjà créées pour chaque pair de maillage dans le maillage fédéré.
- Un compte avec le rôle **cluster-admin**.

1.18.13.1. Configuration d'un ImportedServiceSet pour le basculement

L'équilibrage de charge pondéré par la localité permet aux administrateurs de contrôler la distribution du trafic vers les points d'extrémité en fonction des localités d'où le trafic provient et où il aboutit. Ces localités sont spécifiées à l'aide d'étiquettes arbitraires qui désignent une hiérarchie de localités sous la forme {région}/{zone}/{sous-zone}.

Dans les exemples de cette section, le site **green-mesh** est situé dans la région **us-east** et le site **red-mesh** est situé dans la région **us-west**.

Exemple ImportedServiceSet ressource du maillage rouge au maillage vert

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh #name of mesh that exported the service
```

```
  namespace: green-mesh-system #mesh namespace that service is being imported into
spec:
  importRules: # first matching rule is used
  # import ratings.info as ratings.bookinfo
  - type: NameSelector
    importAsLocal: true
    nameSelector:
      namespace: info
      name: ratings
      alias:
        # service will be imported as ratings.info.svc.red-mesh-imports.local
        namespace: info
        name: ratings
  #Locality within which imported services should be associated.
  locality:
    region: us-west
```

Tableau 1.12. ImportedServiceLocality tableau des champs

Nom	Description	Type
région :	Région dans laquelle les services importés sont situés.	chaîne de caractères
sous-zone :	Sous-zone dans laquelle se trouvent les services importés. Si la sous-zone est spécifiée, la zone doit également l'être.	chaîne de caractères
zone :	Zone dans laquelle les services importés sont situés. Si la zone est spécifiée, la région doit également l'être.	chaîne de caractères

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin** et entrez la commande suivante :

   ```
   $ oc login --username=<NAMEOFUSER> <API token> https://<HOSTNAME>:6443
   ```

2. Passez au projet dans lequel vous avez installé le plan de contrôle Service Mesh et entrez la commande suivante :

   ```
   oc project <smcp-system>
   ```

   Par exemple, **green-mesh-system**.

   ```
   $ oc project green-mesh-system
   ```

3. Modifiez le fichier **ImportedServiceSet**, où **<ImportedServiceSet.yaml>** comprend le chemin d'accès complet au fichier que vous souhaitez modifier, en entrant la commande suivante :

   ```
   -
   ```

```
$ oc edit -n <smcp-system> -f <ImportedServiceSet.yaml>
```

Par exemple, si vous souhaitez modifier le fichier qui importe du système à mailles rouges vers le système à mailles vertes, comme indiqué dans l'exemple précédent **ImportedServiceSet**.

```
$ oc edit -n green-mesh-system -f import-from-red-mesh.yaml
```

4. Modifier le fichier :
   a. Définir **spec.importRules.importAsLocal** à **true**.
   b. Définissez **spec.locality** comme étant **region**, **zone**, ou **subzone**.
   c. Enregistrez vos modifications.

1.18.13.2. Configuration d'une règle de destination pour le basculement

Créez une ressource **DestinationRule** qui configure les éléments suivants :

- Détection des valeurs aberrantes pour le service. Cette fonction est nécessaire pour que le basculement fonctionne correctement. En particulier, elle configure les mandataires sidecar pour qu'ils sachent quand les points d'extrémité d'un service ne sont pas sains, ce qui déclenche éventuellement un basculement vers la localité suivante.
- Politique de basculement entre les régions. Cela permet de s'assurer que le basculement au-delà d'une région se déroulera de manière prévisible.

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**. Entrez la commande suivante. Saisissez ensuite votre nom d'utilisateur et votre mot de passe lorsque vous y êtes invité.

```
$ oc login --username=<NAMEOFUSER> <API token> https://<HOSTNAME>:6443
```

2. Passez au projet dans lequel vous avez installé le plan de contrôle Service Mesh.

```
oc project <smcp-system>
```

Par exemple, **green-mesh-system**.

```
$ oc project green-mesh-system
```

3. Créez un fichier **DestinationRule** basé sur l'exemple suivant où, si green-mesh est indisponible, le trafic doit être acheminé depuis green-mesh dans la région **us-east** vers red-mesh dans **us-west**.

Exemple DestinationRule

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: default-failover
  namespace: info
```

```
spec:
  host: "ratings.info.svc.cluster.local"
  trafficPolicy:
    loadBalancer:
      localityLbSetting:
        enabled: true
        failover:
          - from: us-east
            to: us-west
    outlierDetection:
      consecutive5xxErrors: 3
      interval: 10s
      baseEjectionTime: 1m
```

4. Déployez le fichier **DestinationRule**, où **<DestinationRule>** comprend le chemin complet de votre fichier, en entrant la commande suivante :

```
oc create -n <application namespace> -f <DestinationRule.yaml>
```

Par exemple :

```
$ oc create -n info -f green-mesh-us-west-DestinationRule.yaml
```

1.18.14. Supprimer un service du maillage fédéré

Si vous devez supprimer un service du maillage fédéré, par exemple s'il est devenu obsolète ou s'il a été remplacé par un autre service, vous pouvez le faire.

1.18.14.1. Pour supprimer un service d'une seule maille

Supprimer l'entrée du service de la ressource **ImportedServiceSet** pour l'homologue maillé qui ne doit plus accéder au service.

1.18.14.2. Pour supprimer un service de l'ensemble du maillage fédéré

Supprimer l'entrée du service de la ressource **ExportedServiceSet** pour la maille qui possède le service.

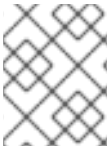
1.18.15. Supprimer une maille du maillage fédéré

Si vous devez retirer une maille de la fédération, vous pouvez le faire.

1. Modifier la ressource **ServiceMeshControlPlane** de la maille supprimée pour supprimer toutes les passerelles d'entrée de fédération pour les mailles homologues.
2. Pour chaque pair de maillage avec lequel le maillage supprimé a été fédéré :
   a. Supprimer la ressource **ServiceMeshPeer** qui relie les deux mailles.
   b. Modifiez la ressource **ServiceMeshControlPlane** de la maille homologue pour supprimer la passerelle de sortie qui dessert la maille supprimée.

1.19. EXTENSIONS

Vous pouvez utiliser des extensions WebAssembly pour ajouter de nouvelles fonctionnalités directement dans les proxies Red Hat OpenShift Service Mesh. Cela vous permet de déplacer encore plus de fonctionnalités courantes hors de vos applications, et de les mettre en œuvre dans un langage unique qui se compile en bytecode WebAssembly.



NOTE

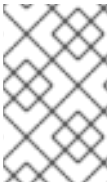
Les extensions WebAssembly ne sont pas prises en charge sur les systèmes IBM zSystems et IBM Power.

1.19.1. Vue d'ensemble des modules WebAssembly

Les modules WebAssembly peuvent être exécutés sur de nombreuses plates-formes, y compris les proxies, et disposent d'un large support linguistique, d'une exécution rapide et d'un modèle de sécurité "sandboxé" par défaut.

Les extensions Red Hat OpenShift Service Mesh sont des filtres HTTP Envoy, ce qui leur confère un large éventail de capacités :

- Manipulation du corps et des en-têtes des demandes et des réponses.
- Demandes HTTP hors bande adressées à des services qui ne se trouvent pas sur le chemin de la demande, tels que l'authentification ou la vérification de la politique.
- Stockage de données sur le canal latéral et files d'attente pour que les filtres communiquent entre eux.



NOTE

Lors de la création de nouvelles extensions WebAssembly, utilisez l'API **WasmPlugin**. L'API **ServiceMeshExtension** était obsolète dans Red Hat OpenShift Service Mesh version 2.2 et a été supprimée dans Red Hat OpenShift Service Mesh version 2.3.

L'écriture d'une extension Red Hat OpenShift Service Mesh comporte deux parties :

1. Vous devez écrire votre extension à l'aide d'un SDK qui expose l'API proxy-wasm et la compiler en un module WebAssembly.
2. Vous devez ensuite placer le module dans un conteneur.

Langues prises en charge

Vous pouvez utiliser n'importe quel langage qui compile le bytecode WebAssembly pour écrire une extension Red Hat OpenShift Service Mesh, mais les langages suivants ont des SDK existants qui exposent l'API proxy-wasm afin qu'elle puisse être consommée directement.

Tableau 1.13. Langues prises en charge

Langue	Responsable de la maintenance	Référentiel
AssemblyScript	solo.io	solo-io/proxy-runtime
C	équipe proxy-wasm (Communauté Istio)	proxy-wasm/proxy-wasm-cpp-sdk

Langue	Responsable de la maintenance	Référentiel
Aller	tetrate.io	tetratelabs/proxy-wasm-go-sdk
Rouille	équipe proxy-wasm (Communauté Istio)	proxy-wasm/proxy-wasm-rust-sdk

1.19.2. WasmPlugin format du conteneur

Istio prend en charge les images de l'Open Container Initiative (OCI) dans son mécanisme Wasm Plugin. Vous pouvez distribuer vos plugins Wasm en tant qu'image de conteneur et vous pouvez utiliser le champ **spec.url** pour faire référence à l'emplacement d'un registre de conteneurs. Par exemple, **quay.io/my-username/my-plugin:latest**.

Étant donné que chaque environnement d'exécution (runtime) d'un module WASM peut avoir des paramètres de configuration spécifiques au runtime, une image WASM peut être composée de deux couches :

- **plugin.wasm** (Obligatoire) - Couche de contenu. Cette couche consiste en un binaire **.wasm** contenant le bytecode de votre module WebAssembly, qui sera chargé par le runtime. Vous devez nommer ce fichier **plugin.wasm**.
- **runtime-config.json** (Facultatif) - Couche de configuration. Cette couche se compose d'une chaîne au format JSON qui décrit les métadonnées du module pour le système d'exécution cible. La couche de configuration peut également contenir des données supplémentaires, en fonction de l'environnement d'exécution cible. Par exemple, la configuration d'un filtre WASM Envoy contient les root_ids disponibles sur le filtre.

1.19.3. Référence API WasmPlugin

L'API WasmPlugins fournit un mécanisme pour étendre la fonctionnalité fournie par le proxy Istio à travers des filtres WebAssembly.

Vous pouvez déployer plusieurs WasmPlugins. Les paramètres **phase** et **priority** déterminent l'ordre d'exécution (dans le cadre de la chaîne de filtrage d'Envoy), ce qui permet de configurer des interactions complexes entre les WasmPlugins fournis par l'utilisateur et les filtres internes d'Istio.

Dans l'exemple suivant, un filtre d'authentification met en œuvre un flux OpenID et remplit l'en-tête Authorization avec un jeton Web JSON (JWT). L'authentification Istio consomme ce jeton et le déploie sur la passerelle d'entrée. Le fichier WasmPlugin se trouve dans le système de fichiers sidecar du proxy. Notez le champ **url**.

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: openid-connect
  namespace: istio-ingress
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  url: file:///opt/filters/openid.wasm
  sha256: 1ef0c9a92b0420cf25f7fe5d481b231464bc88f486ca3b9c83ed5cc21d2f6210
```

```
  phase: AUTHN
  pluginConfig:
    openid_server: authn
    openid_realm: ingress
```

Voici le même exemple, mais cette fois-ci, une image OCI (Open Container Initiative) est utilisée à la place d'un fichier dans le système de fichiers. Notez les champs **url**, **imagePullPolicy**, et **imagePullSecret**.

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: openid-connect
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  url: oci://private-registry:5000/openid-connect/openid:latest
  imagePullPolicy: IfNotPresent
  imagePullSecret: private-registry-pull-secret
  phase: AUTHN
  pluginConfig:
    openid_server: authn
    openid_realm: ingress
```

Tableau 1.14. Référence du champ WasmPlugin

Field	Type	Description	Exigée
spec.selector	Sélecteur de charge de travail	Critères utilisés pour sélectionner l'ensemble spécifique de pods/VMs sur lesquels cette configuration de plugin doit être appliquée. Si elle est omise, cette configuration sera appliquée à toutes les instances de charge de travail dans le même espace de noms. Si le champ **WasmPlugin** est présent dans l'espace de noms racine de la configuration, celle-ci sera appliquée à toutes les charges de travail applicables dans n'importe quel espace de noms.	Non

Field	Type	Description	Exigée
spec.url	chaîne de caractères	URL d'un module Wasm ou d'un conteneur OCI. Si aucun schéma n'est présent, la valeur par défaut est **oci://**, qui fait référence à une image OCI. Les autres schémas valables sont **file://** pour les fichiers de modules .wasm présents localement dans le conteneur proxy, et **http[s]://** pour les fichiers de modules .wasm hébergés à distance.	Non
spec.sha256	chaîne de caractères	Somme de contrôle SHA256 qui sera utilisée pour vérifier le module Wasm ou le conteneur OCI. Si le champ **url** fait déjà référence à un SHA256 (en utilisant la notation **@sha256:**), il doit correspondre à la valeur de ce champ. Si une image OCI est référencée par une balise et que ce champ est défini, sa somme de contrôle sera vérifiée par rapport au contenu de ce champ après l'extraction.	Non

Field	Type	Description	Exigée
spec.imagePullPolicy	Politique d'attraction	Le comportement d'extraction à appliquer lors de la récupération d'une image OCI. Ne s'applique que lorsque les images sont référencées par un tag au lieu d'un SHA. La valeur par défaut est **IfNotPresent**, sauf si une image OCI est référencée dans le champ **url** et que la balise **latest** est utilisée, auquel cas la valeur **Always** est la valeur par défaut, reflétant le comportement de K8. Le paramètre est ignoré si le champ **url** fait référence à un module Wasm directement en utilisant **file:**// ou **http[s]:**//.	Non
spec.imagePullSecret	chaîne de caractères	Informations d'identification à utiliser pour l'extraction d'images OCI. Le nom d'un secret dans le même espace de noms que l'objet **WasmPlugin** qui contient un secret d'extraction pour l'authentification contre le registre lors de l'extraction de l'image.	Non
spec.phase	PluginPhase	Détermine l'endroit de la chaîne de filtrage où cet objet **WasmPlugin** est injecté.	Non

Field	Type	Description	Exigée
spec.priority	**int64**	Détermine l'ordre des objets **WasmPlugins** qui ont la même valeur **phase**. Lorsque plusieurs objets **WasmPlugins** sont appliqués à la même charge de travail au cours de la même phase, ils sont appliqués par priorité et par ordre décroissant. Si le champ **priority** n'est pas défini ou si deux objets **WasmPlugins** ont la même valeur, l'ordre sera déterminé à partir du nom et de l'espace de noms des objets **WasmPlugins**. La valeur par défaut est **0**.	Non
spec.pluginName	chaîne de caractères	Le nom du plugin utilisé dans la configuration d'Envoy. Certains modules Wasm peuvent avoir besoin de cette valeur pour sélectionner le plugin Wasm à exécuter.	Non
spec.pluginConfig	Structure	La configuration qui sera transmise au plugin.	Non
spec.pluginConfig.verificationKey	chaîne de caractères	Clé publique utilisée pour vérifier les signatures des images OCI ou des modules Wasm signés. Elle doit être fournie au format PEM.	Non

L'objet **WorkloadSelector** spécifie les critères utilisés pour déterminer si un filtre peut être appliqué à un proxy. Les critères de correspondance comprennent les métadonnées associées à un proxy, les informations sur l'instance de charge de travail telles que les étiquettes attachées au pod/VM, ou toute autre information que le proxy fournit à Istio au cours de l'échange initial. Si plusieurs conditions sont spécifiées, toutes les conditions doivent correspondre pour que l'instance de charge de travail soit sélectionnée. Actuellement, seul le mécanisme de sélection basé sur les étiquettes est pris en charge.

Tableau 1.15. Sélecteur de charge de travail

Field	Type	Description	Exigée
matchLabels	map<chaîne, string>	Une ou plusieurs étiquettes qui indiquent un ensemble spécifique de pods/VMs sur lesquels une politique doit être appliquée. La portée de la recherche d'étiquettes est limitée à l'espace de noms de configuration dans lequel la ressource est présente.	Oui

L'objet **PullPolicy** spécifie le comportement de traction à appliquer lors de la récupération d'une image OCI.

Tableau 1.16. Politique d'attraction

Valeur	Description
<empty>	La valeur par défaut est **IfNotPresent**, sauf pour les images OCI avec l'étiquette latest, pour lesquelles la valeur par défaut est **Always**.
S'il n'est pas présent	Si une version existante de l'image a déjà été extraite, elle sera utilisée. Si aucune version de l'image n'est présente localement, nous utiliserons la version la plus récente.
Always	Utilisez toujours la dernière version d'une image lorsque vous utilisez ce plugin.

Struct représente une valeur de données structurée, composée de champs qui correspondent à des valeurs typées dynamiquement. Dans certains langages, les structures peuvent être représentées de manière native. Par exemple, dans les langages de script comme JavaScript, une structure est représentée comme un objet.

Tableau 1.17. Structure

Field	Type	Description
champs	map<string, Value>	Carte de valeurs typées dynamiquement.

PluginPhase spécifie la phase de la chaîne de filtrage dans laquelle le plugin sera injecté.

Tableau 1.18. PluginPhase

Field	Description
<empty>	Le plan de contrôle décide de l'endroit où insérer le plugin. Il se trouve généralement à la fin de la chaîne de filtrage, juste avant le routeur. Ne pas spécifier PluginPhase si le plugin est indépendant des autres.
AUTHN	Insérer le plugin avant les filtres d'authentification Istio.
AUTHZ	Insérer le plugin avant les filtres d'autorisation Istio et après les filtres d'authentification Istio.
STATS	Insérer le plugin avant les filtres de statistiques Istio et après les filtres d'autorisation Istio.

1.19.3.1. Déploiement des ressources WasmPlugin

Vous pouvez activer les extensions Red Hat OpenShift Service Mesh en utilisant la ressource **WasmPlugin**. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh. L'exemple suivant crée un filtre **openid-connect** qui exécute un flux OpenID Connect pour authentifier l'utilisateur.

Procédure

1. Créez l'exemple de ressource suivant :

   Exemple plugin.yaml

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: openid-connect
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  url: oci://private-registry:5000/openid-connect/openid:latest
  imagePullPolicy: IfNotPresent
  imagePullSecret: private-registry-pull-secret
  phase: AUTHN
  pluginConfig:
    openid_server: authn
    openid_realm: ingress
```

2. Appliquez votre fichier **plugin.yaml** avec la commande suivante :

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

1.19.4. ServiceMeshExtension format du conteneur

Vous devez avoir un fichier **.wasm** contenant le bytecode de votre module WebAssembly et un fichier **manifest.yaml** à la racine du système de fichiers du conteneur pour que votre image de conteneur soit une image d'extension valide.

NOTE

Lors de la création de nouvelles extensions WebAssembly, utilisez l'API **WasmPlugin**. L'API **ServiceMeshExtension** était obsolète dans Red Hat OpenShift Service Mesh version 2.2 et a été supprimée dans Red Hat OpenShift Service Mesh version 2.3.

manifest.yaml

```
schemaVersion: 1

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

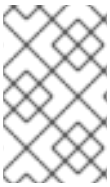
Tableau 1.19. Référence de champ pour manifest.yml

Field	Description	Exigée
schemaVersion	Utilisé pour la version du schéma du manifeste. Actuellement, la seule valeur possible est **1**.	Ce champ est obligatoire.
nom	Le nom de votre extension.	Ce champ n'est qu'une métadonnée et est actuellement inutilisé.
description	La description de votre extension.	Ce champ n'est qu'une métadonnée et est actuellement inutilisé.
version	La version de votre extension.	Ce champ n'est qu'une métadonnée et est actuellement inutilisé.
phase	La phase d'exécution par défaut de votre extension.	Ce champ est obligatoire.
priorité	La priorité par défaut de votre extension.	Ce champ est obligatoire.
module	Le chemin relatif de la racine du système de fichiers du conteneur vers votre module WebAssembly.	Ce champ est obligatoire.

1.19.5. Référence de ServiceMeshExtension

L'API ServiceMeshExtension fournit un mécanisme permettant d'étendre les fonctionnalités fournies par le proxy Istio au moyen de filtres WebAssembly. L'écriture d'une extension WebAssembly comporte deux parties :

1. Écrivez votre extension à l'aide d'un SDK qui expose l'API proxy-wasm et compilez-la dans un module WebAssembly.
2. L'emballer dans un récipient.



NOTE

Lors de la création de nouvelles extensions WebAssembly, utilisez l'API **WasmPlugin**. L'API **ServiceMeshExtension**, qui était obsolète dans Red Hat OpenShift Service Mesh version 2.2, a été supprimée dans Red Hat OpenShift Service Mesh version 2.3.

Tableau 1.20. Référence de champ ServiceMeshExtension

Field	Description
metadata.namespace	Le champ **metadata.namespace** d'une source **ServiceMeshExtension** a une sémantique particulière : s'il est égal à l'espace de nommage du plan de contrôle, l'extension sera appliquée à toutes les charges de travail du Service Mesh qui correspondent à sa valeur **workloadSelector**. Lorsqu'elle est déployée dans un autre espace de nommage du maillage, elle ne sera appliquée qu'aux charges de travail de ce même espace de nommage.
spec.workloadSelector	Le champ **spec.workloadSelector** a la même sémantique que le champ **spec.selector** de la ressource Istio Gateway. Il correspondra à une charge de travail en fonction de ses étiquettes Pod. Si aucune valeur **workloadSelector** n'est spécifiée, l'extension sera appliquée à toutes les charges de travail de l'espace de noms.
spec.config	Il s'agit d'un champ structuré qui sera transmis à l'extension, la sémantique dépendant de l'extension que vous déployez.
spec.image	Un URI d'image conteneur pointant vers l'image qui contient l'extension.

Field	Description
spec.phase	La phase détermine à quel endroit de la chaîne de filtrage l'extension est injectée, par rapport aux fonctionnalités existantes d'Istio telles que l'authentification, l'autorisation et la génération de métriques. Les valeurs valides sont : PreAuthN, PostAuthN, PreAuthZ, PostAuthZ, PreStats, PostStats. Ce champ prend par défaut la valeur définie dans le fichier **manifest.yaml** de l'extension, mais peut être remplacé par l'utilisateur.
spec.priority	Si plusieurs extensions ayant la même valeur **spec.phase** sont appliquées à la même instance de charge de travail, la valeur **spec.priority** détermine l'ordre d'exécution. Les extensions ayant une priorité plus élevée seront exécutées en premier. Cela permet de prendre en compte les extensions interdépendantes. Ce champ prend par défaut la valeur définie dans le fichier **manifest.yaml** de l'extension, mais peut être remplacé par l'utilisateur.

1.19.5.1. Déploiement des ressources ServiceMeshExtension

Vous pouvez activer les extensions Red Hat OpenShift Service Mesh en utilisant la ressource **ServiceMeshExtension**. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

NOTE

Lors de la création de nouvelles extensions WebAssembly, utilisez l'API **WasmPlugin**. L'API **ServiceMeshExtension** a été obsolète dans Red Hat OpenShift Service Mesh version 2.2 et supprimée dans Red Hat OpenShift Service Mesh version 2.3.

Pour un exemple complet qui a été construit en utilisant le SDK Rust, jetez un coup d'œil au header-append-filter. Il s'agit d'un simple filtre qui ajoute un ou plusieurs en-têtes aux réponses HTTP, avec leurs noms et valeurs extraits du champ **config** de l'extension. Vous trouverez un exemple de configuration dans l'extrait ci-dessous.

Procédure

1. Créez l'exemple de ressource suivant :

Exemple de ressource ServiceMeshExtension extension.yaml

```
apiVersion: maistra.io/v1
kind: ServiceMeshExtension
metadata:
  name: header-append
  namespace: istio-system
```

```
spec:
  workloadSelector:
    labels:
      app: httpbin
  config:
    first-header: some-value
    another-header: another-value
  image: quay.io/maistra-dev/header-append-filter:2.1
  phase: PostAuthZ
  priority: 100
```

2. Appliquez votre fichier **extension.yaml** avec la commande suivante :

```
$ oc apply -f <extension>.yaml
```

1.19.6. Migration des ressources ServiceMeshExtension vers WasmPlugin

L'API **ServiceMeshExtension**, qui était obsolète dans Red Hat OpenShift Service Mesh version 2.2, a été supprimée dans Red Hat OpenShift Service Mesh version 2.3. Si vous utilisez l'API **ServiceMeshExtension**, vous devez migrer vers l'API **WasmPlugin** pour continuer à utiliser vos extensions WebAssembly.

Les API sont très similaires. La migration se fait en deux étapes :

1. Renommer votre fichier de plugin et mettre à jour l'emballage du module.
2. Création d'une ressource **WasmPlugin** qui fait référence à l'image de conteneur mise à jour.

1.19.6.1. Modifications de l'API

La nouvelle API **WasmPlugin** est similaire à l'API **ServiceMeshExtension**, mais avec quelques différences, notamment en ce qui concerne les noms des champs :

Tableau 1.21. Changements de champ entre ServiceMeshExtensions et WasmPlugin

ServiceMeshExtension	WasmPlugin
spec.config	**spec.pluginConfig**
spec.workloadSelector	**spec.selector**
spec.image	**spec.url**
spec.phase valeurs valides : PreAuthN, PostAuthN, PreAuthZ, PostAuthZ, PreStats, PostStats	**spec.phase** valeurs valides : <empty>, AUTHN, AUTHZ, STATS

Voici un exemple de conversion d'une ressource **ServiceMeshExtension** en ressource **WasmPlugin**.

Ressource ServiceMeshExtension

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

```
metadata:
  name: header-append
  namespace: istio-system
spec:
  workloadSelector:
    labels:
      app: httpbin
  config:
    first-header: some-value
    another-header: another-value
  image: quay.io/maistra-dev/header-append-filter:2.2
  phase: PostAuthZ
  priority: 100
```

Nouvelle ressource WasmPlugin équivalente au ServiceMeshExtension ci-dessus

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: header-append
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: httpbin
  url: oci://quay.io/maistra-dev/header-append-filter:2.2
  phase: STATS
  pluginConfig:
    first-header: some-value
    another-header: another-value
```

1.19.6.2. Modifications du format de l'image du conteneur

Le nouveau format d'image du conteneur **WasmPlugin** est similaire à celui de **ServiceMeshExtensions**, avec les différences suivantes :

- Le format de conteneur **ServiceMeshExtension** nécessite un fichier de métadonnées nommé **manifest.yaml** dans le répertoire racine du système de fichiers du conteneur. Le format de conteneur **WasmPlugin** ne nécessite pas de fichier **manifest.yaml**.
- Le fichier **.wasm** (le plugin proprement dit), qui pouvait auparavant avoir n'importe quel nom de fichier, doit désormais être nommé **plugin.wasm** et se trouver dans le répertoire racine du système de fichiers du conteneur.

1.19.6.3. Migration vers les ressources WasmPlugin

Pour mettre à jour vos extensions WebAssembly de l'API **ServiceMeshExtension** à l'API **WasmPlugin**, vous devez renommer votre fichier de plugin.

Conditions préalables

- **ServiceMeshControlPlane** est mis à niveau vers la version 2.2 ou une version ultérieure.

Procédure

1. Mettez à jour l'image de votre conteneur. Si le plugin est déjà dans **/plugin.wasm** à l'intérieur du conteneur, passez à l'étape suivante. Si ce n'est pas le cas :

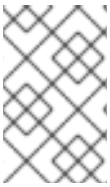
   a. Veillez à ce que le fichier du plugin soit nommé **plugin.wasm**. Vous devez nommer le fichier d'extension **plugin.wasm**.

   b. Assurez-vous que le fichier d'extension se trouve dans le répertoire racine (/). Vous devez stocker les fichiers d'extension à la racine du système de fichiers du conteneur...

   c. Reconstruisez votre image de conteneur et envoyez-la dans un registre de conteneurs.

2. Supprimez la ressource **ServiceMeshExtension** et créez une ressource **WasmPlugin** qui fait référence à la nouvelle image de conteneur que vous avez créée.

1.20. UTILISATION DU MODULE WEBASSEMBLY DE 3SCALE



NOTE

Le module **threescale-wasm-auth** fonctionne sur les intégrations de 3scale API Management 2.11 ou plus récent avec Red Hat OpenShift Service Mesh 2.1.0 ou plus récent.

Le module **threescale-wasm-auth** est un module WebAssembly qui utilise un ensemble d'interfaces, connues sous le nom d'interfaces binaires d'application (*ABI*). Cette interface est définie par la spécification *Proxy-WASM* pour piloter tout logiciel qui implémente l'ABI afin qu'il puisse autoriser les requêtes HTTP contre 3scale.

En tant que spécification ABI, Proxy-WASM définit l'interaction entre un logiciel nommé *host* et un autre nommé *module*, *program* ou *extension*. L'hôte expose un ensemble de services utilisés par le module pour effectuer une tâche et, dans le cas présent, pour traiter les demandes de proxy.

L'environnement hôte est composé d'une machine virtuelle WebAssembly qui interagit avec un logiciel, dans ce cas, un proxy HTTP.

Le module lui-même est isolé du monde extérieur, à l'exception des instructions qu'il exécute sur la machine virtuelle et de l'ABI spécifiée par Proxy-WASM. Il s'agit d'un moyen sûr de fournir des points d'extension aux logiciels : l'extension ne peut interagir que de manière bien définie avec la machine virtuelle et l'hôte. L'interaction fournit un modèle informatique et une connexion au monde extérieur que le proxy est censé avoir.

1.20.1. Compatibilité

Le module **threescale-wasm-auth** est conçu pour être entièrement compatible avec toutes les implémentations de la spécification *Proxy-WASM ABI*. Toutefois, à ce stade, il n'a été testé en profondeur que pour fonctionner avec le proxy inverse Envoy.

1.20.2. Utilisation en tant que module autonome

Grâce à sa conception autonome, il est possible de configurer ce module pour qu'il fonctionne avec des proxies Proxy-WASM indépendamment de Service Mesh, ainsi qu'avec des déploiements d'adaptateurs Istio 3scale.

1.20.3. Conditions préalables

- Le module fonctionne avec toutes les versions de 3scale prises en charge, sauf lors de la configuration d'un service pour utiliser OpenID connect (OIDC), qui nécessite 3scale 2.11 ou une version ultérieure.

1.20.4. Configuration du module threescale-wasm-auth

Les administrateurs de cluster sur OpenShift Container Platform peuvent configurer le module **threescale-wasm-auth** pour autoriser les requêtes HTTP à 3scale API Management via une interface binaire d'application (ABI). L'ABI définit l'interaction entre l'hôte et le module, exposant les services de l'hôte, et vous permet d'utiliser le module pour traiter les demandes de proxy.

1.20.4.1. L'extension WasmPlugin API

Service Mesh fournit une définition de ressource personnalisée pour spécifier et appliquer les extensions Proxy-WASM aux mandataires sidecar, connus sous le nom de **WasmPlugin**. Service Mesh applique cette ressource personnalisée à l'ensemble des charges de travail qui nécessitent une gestion de l'API HTTP avec 3scale.

Pour plus d'informations, voir la définition des ressources personnalisées.

NOTE

La configuration de l'extension WebAssembly est actuellement un processus manuel. La prise en charge de la récupération de la configuration des services du système 3scale sera disponible dans une prochaine version.

Conditions préalables

- Identifiez une charge de travail et un espace de noms Kubernetes sur votre déploiement Service Mesh que vous appliquerez ce module.
- Vous devez avoir un compte locataire 3scale. Voir SaaS ou 3scale 2.11 On-Premises avec un service correspondant et des applications et métriques pertinentes définies.
- Si vous appliquez le module au microservice **<product_page>** dans l'espace de noms **info**, voyez l'exemple d'application Bookinfo.
  - L'exemple suivant est le format YAML pour la ressource personnalisée du module **threescale-wasm-auth**. Cet exemple fait référence à la version Maistra en amont de Service Mesh, **WasmPlugin** API. Vous devez déclarer l'espace de noms dans lequel le module **threescale-wasm-auth** est déployé, ainsi que **selector** pour identifier l'ensemble des applications auxquelles le module s'appliquera :

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
  namespace: <info> 1
spec:
  selector: 2
    labels:
      app: <product_page>
  pluginConfig: <yaml_configuration>
```

```
    url: oci://registry.redhat.io/3scale-amp2/3scale-auth-wasm-rhel8:0.0.3
    phase: AUTHZ
    priority: 100
```

1. Le site **namespace**.
2. Le site **selector**.

- Le champ **spec.pluginConfig** dépend de la configuration du module et n'est pas renseigné dans l'exemple précédent. Au lieu de cela, l'exemple utilise la valeur de remplacement **<yaml_configuration>**. Vous pouvez utiliser le format de cet exemple de ressource personnalisée.
  - Le champ **spec.pluginConfig** varie en fonction de l'application. Tous les autres champs sont conservés dans plusieurs instances de cette ressource personnalisée. A titre d'exemple :
    - **url**: Ne change que lorsque des versions plus récentes du module sont déployées.
    - **phase**: Reste inchangé, puisque ce module doit être invoqué après que le proxy a effectué toute autorisation locale, telle que la validation des jetons OpenID Connect (OIDC).
- Une fois que vous avez la configuration du module dans **spec.pluginConfig** et le reste de la ressource personnalisée, appliquez-la à l'aide de la commande **oc apply**:

```
$ oc apply -f threescale-wasm-auth-info.yaml
```

Ressources supplémentaires

- Migration des ressources **ServiceMeshExtension** vers **WasmPlugin**
- Ressources personnalisées

1.20.5. Application des objets ServiceEntry externes 3scale

Pour que le module **threescale-wasm-auth** autorise les requêtes contre 3scale, le module doit avoir accès aux services 3scale. Vous pouvez le faire dans Red Hat OpenShift Service Mesh en appliquant un objet externe **ServiceEntry** et un objet correspondant **DestinationRule** pour la configuration TLS afin d'utiliser le protocole HTTPS.

Les ressources personnalisées (CR) définissent les entrées de service et les règles de destination pour un accès sécurisé depuis Service Mesh vers 3scale Hosted (SaaS) pour les composants backend et système de l'API de gestion des services et de l'API de gestion des comptes. L'API de gestion des services reçoit des requêtes sur le statut d'autorisation de chaque demande. L'API de gestion des comptes fournit les paramètres de configuration de la gestion de l'API pour vos services.

Procédure

1. Appliquez à votre cluster le CR **ServiceEntry** externe suivant et le CR **DestinationRule** connexe pour 3scale Hosted **backend**:
   a. Ajoutez le CR **ServiceEntry** à un fichier appelé **service-entry-threescale-saas-backend.yml**:

ServiceEntry CR

```
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: service-entry-threescale-saas-backend
spec:
  hosts:
  - su1.3scale.net
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
```

b. Ajoutez le CR **DestinationRule** à un fichier appelé **destination-rule-threescale-saas-backend.yml**:

DestinationRule CR

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: destination-rule-threescale-saas-backend
spec:
  host: su1.3scale.net
  trafficPolicy:
    tls:
      mode: SIMPLE
      sni: su1.3scale.net
```

c. Appliquez et enregistrez le CR **ServiceEntry** externe pour le backend 3scale Hosted à votre cluster, en exécutant la commande suivante :

```
$ oc apply -f service-entry-threescale-saas-backend.yml
```

d. Appliquez et sauvegardez le CR **DestinationRule** externe pour le backend 3scale Hosted à votre cluster, en exécutant la commande suivante :

```
$ oc apply -f destination-rule-threescale-saas-backend.yml
```

2. Appliquez à votre cluster le CR **ServiceEntry** externe suivant et le CR **DestinationRule** connexe pour 3scale Hosted **system**:

a. Ajoutez le CR **ServiceEntry** à un fichier appelé **service-entry-threescale-saas-system.yml**:

ServiceEntry CR

```
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: service-entry-threescale-saas-system
```

```
spec:
  hosts:
  - multitenant.3scale.net
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
```

b. Ajoutez le CR **DestinationRule** à un fichier appelé **destination-rule-threescale-saas-system.yml**:

DestinationRule CR

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: destination-rule-threescale-saas-system
spec:
  host: multitenant.3scale.net
  trafficPolicy:
    tls:
      mode: SIMPLE
      sni: multitenant.3scale.net
```

c. Appliquez et enregistrez le CR **ServiceEntry** externe pour le système 3scale Hosted dans votre cluster, en exécutant la commande suivante :

```
$ oc apply -f service-entry-threescale-saas-system.yml
```

d. Appliquez et enregistrez le CR **DestinationRule** externe pour le système 3scale Hosted dans votre cluster, en exécutant la commande suivante :

```
oc apply -f <destination-rule-threescale-saas-system.yml>
```

Vous pouvez également déployer un service 3scale in-mesh. Pour déployer un service 3scale in-mesh, modifiez l'emplacement des services dans le CR en déployant 3scale et en établissant un lien avec le déploiement.

Ressources supplémentaires

- Documentation sur les règles d'entrée et de destination des services

1.20.6. La configuration du module WebAssembly 3scale

La spécification de la ressource personnalisée **WasmPlugin** fournit la configuration que le module **Proxy-WASM** lit.

La spécification est intégrée dans l'hôte et lue par le module **Proxy-WASM**. Généralement, les configurations sont au format JSON pour que les modules puissent les analyser, mais la ressource **WasmPlugin** peut interpréter la valeur de la spécification comme YAML et la convertir en JSON pour qu'elle puisse être consommée par le module.

Si vous utilisez le module **Proxy-WASM** en mode autonome, vous devez écrire la configuration en utilisant le format JSON. L'utilisation du format JSON implique l'utilisation d'échappements et de guillemets lorsque cela est nécessaire dans les fichiers de configuration de **host**, par exemple **Envoy**. Lorsque vous utilisez le module WebAssembly avec la ressource **WasmPlugin**, la configuration est au format YAML. Dans ce cas, une configuration invalide oblige le module à afficher des diagnostics basés sur sa représentation JSON dans le flux de journalisation d'un sidecar.

IMPORTANT

La ressource personnalisée **EnvoyFilter** n'est pas une API prise en charge, bien qu'elle puisse être utilisée dans certaines versions de l'adaptateur 3scale Istio ou du Service Mesh. L'utilisation de la ressource personnalisée **EnvoyFilter** n'est pas recommandée. Utilisez l'API **WasmPlugin** au lieu de la ressource personnalisée **EnvoyFilter**. Si vous devez utiliser la ressource personnalisée **EnvoyFilter**, vous devez spécifier la spécification au format JSON.

1.20.6.1. Configuration du module 3scale WebAssembly

L'architecture de la configuration du module WebAssembly de 3scale dépend du service de compte et d'autorisation de 3scale et de la liste des services à gérer.

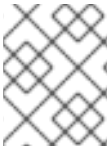
Conditions préalables

Les conditions préalables sont un ensemble de champs minimaux obligatoires dans tous les cas :

- Pour le service de compte et d'autorisation 3scale : l'URL **backend-listener**.
- Pour la liste des services à gérer : les identifiants des services et au moins une méthode de recherche d'informations d'identification et l'endroit où la trouver.
- Vous trouverez des exemples pour traiter avec **userkey**, **appid** avec **appkey**, et les modèles OpenID Connect (OIDC).
- Le module WebAssembly utilise les paramètres que vous avez spécifiés dans la configuration statique. Par exemple, si vous ajoutez une configuration de règle de mappage au module, elle s'appliquera toujours, même si le portail d'administration 3scale n'a pas de règle de mappage de ce type. Le reste de la ressource **WasmPlugin** existe autour de l'entrée YAML **spec.pluginConfig**.

1.20.6.2. L'objet api du module WebAssembly de 3scale

La chaîne de niveau supérieur **api** du module 3scale WebAssembly définit la version de la configuration que le module utilisera.



NOTE

Une version inexistante ou non prise en charge de l'objet **api** rend le module 3scale WebAssembly inopérant.

L'exemple de la chaîne de niveau supérieur api

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
```

```
  namespace: <info>
spec:
  pluginConfig:
    api: v1
...
```

L'entrée **api** définit le reste des valeurs de la configuration. La seule valeur acceptée est **v1**. Les nouveaux paramètres qui rompent la compatibilité avec la configuration actuelle ou qui nécessitent plus de logique que les modules utilisant **v1** ne peuvent pas gérer, nécessiteront des valeurs différentes.

1.20.6.3. L'objet système du module 3scale WebAssembly

L'objet de premier niveau **system** spécifie comment accéder à l'API de gestion de compte 3scale pour un compte spécifique. Le champ **upstream** est la partie la plus importante de l'objet. L'objet **system** est facultatif, mais recommandé à moins que vous ne fournissiez une configuration entièrement statique pour le module 3scale WebAssembly, ce qui est une option si vous ne voulez pas fournir de connectivité au composant *system* de 3scale.

Lorsque vous fournissez des objets de configuration statiques en plus de l'objet **system**, les objets statiques sont toujours prioritaires.

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    system:
      name: <saas_porta>
      upstream: <object>
      token: <my_account_token>
      ttl: 300
    ...
```

Tableau 1.22. system champs de l'objet

Nom	Description	Exigée
name	Un identifiant pour le service 3scale, actuellement non référencé ailleurs.	En option
upstream	Les détails d'un hôte du réseau à contacter. **upstream** fait référence à l'hôte de l'API de gestion des comptes 3scale connu sous le nom de système.	Oui
token	Un jeton d'accès personnel à 3 échelles avec des autorisations de lecture.	Oui

Nom	Description	Exigée
ttl	Le nombre minimum de secondes pendant lesquelles une configuration récupérée sur cet hôte doit être considérée comme valide avant d'essayer de récupérer de nouvelles modifications. La valeur par défaut est de 600 secondes (10 minutes). **Note:** Il n'y a pas de valeur maximale, mais le module récupérera généralement toute configuration dans un délai raisonnable après l'expiration de ce TTL.	En option

1.20.6.4. L'objet en amont du module WebAssembly 3scale

L'objet **upstream** décrit un hôte externe vers lequel le proxy peut effectuer des appels.

```
apiVersion: maistra.io/v1
upstream:
  name: outbound|443||multitenant.3scale.net
  url: "https://myaccount-admin.3scale.net/"
  timeout: 5000
...
```

Tableau 1.23. upstream champs de l'objet

Nom	Description	Exigée

Nom	Description	Exigée
name	**name** n'est pas un identifiant de forme libre. Il s'agit de l'identifiant de l'hôte externe tel qu'il est défini dans la configuration du proxy. Dans le cas des configurations **Envoy** autonomes, il correspond au nom d'un cluster, également connu sous le nom de **upstream** dans d'autres proxys. **Note:** la valeur de ce champ, car le plan de contrôle des adaptateurs Service Mesh et 3scale Istio configure le nom selon un format utilisant une barre verticale (|) comme séparateur de plusieurs champs. Pour les besoins de cette intégration, utilisez toujours le format : **outbound|<port>|| <hostname>**.	Oui
url	L'URL complète pour accéder au service décrit. Sauf si le schéma le prévoit, vous devez inclure le port TCP.	Oui
Timeout	Délai d'attente en millisecondes pour que les connexions à ce service qui mettent plus de temps à répondre soient considérées comme des erreurs. La valeur par défaut est de 1000 secondes.	En option

1.20.6.5. L'objet backend du module WebAssembly de 3scale

L'objet de premier niveau **backend** spécifie comment accéder à l'API de gestion des services 3scale pour autoriser et signaler les requêtes HTTP. Ce service est fourni par le composant *Backend* de 3scale.

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
    backend:
      name: backend
      upstream: <object>
    ...
```

Tableau 1.24. backend champs de l'objet

Nom	Description	Exigée
name	Un identifiant pour le backend 3scale, actuellement non référencé ailleurs.	En option
upstream	Les détails d'un hôte du réseau à contacter. Cela doit se référer à l'hôte 3scale Account Management API, connu, système.	Oui, c'est le champ le plus important et le plus obligatoire.

1.20.6.6. L'objet des services du module WebAssembly 3scale

L'objet de premier niveau **services** spécifie quels identificateurs de service sont traités par cette instance particulière de **module**.

Étant donné que les comptes disposent de plusieurs services, vous devez spécifier ceux qui sont gérés. Le reste de la configuration porte sur la manière de configurer les services.

Le champ **services** est obligatoire. Il s'agit d'un tableau qui doit contenir au moins un service pour être utile.

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
    services:
    - id: "2555417834789"
      token: service_token
      authorities:
        - "*.app"
        - 0.0.0.0
        - "0.0.0.0:8443"
      credentials: <object>
      mapping_rules: <object>
    ...
```

Chaque élément du tableau **services** représente un service à 3 échelles.

Tableau 1.25. services champs de l'objet

Nom	Description	Exigée
ID	Un identifiant pour ce service 3scale, actuellement non référencé ailleurs.	Oui

Nom	Description	Exigée
token	Cette adresse **token** peut être trouvée dans la configuration du proxy pour votre service dans le système ou vous pouvez l'extraire du système avec la commande suivante **curl**: **curl https://<system_host>/admin/api/services/<service_id>/proxy/configs/production/latest.json?access_token=<access_token>" | jq '.proxy_config.content.backend_authentication_value**	En option
authorities	Un tableau de chaînes, chacune représentant le *Authority* d'un *URL* à faire correspondre. Ces chaînes acceptent les motifs globaux prenant en charge l'astérisque (*), le signe plus ( ) et le point d'interrogation (?).	Oui
credentials	Un objet définissant le type d'informations à rechercher et l'endroit où elles doivent l'être.	Oui
mapping_rules	Un tableau d'objets représentant les règles de mappage et les méthodes 3scale à appliquer.	En option

1.20.6.7. L'objet d'identification du module WebAssembly 3scale

L'objet **credentials** est un composant de l'objet **service**. **credentials** spécifie le type d'informations d'identification à rechercher et les étapes à suivre pour réaliser cette action.

Tous les champs sont facultatifs, mais vous devez en spécifier au moins un, **user_key** ou **app_id**. L'ordre dans lequel vous spécifiez chaque justificatif n'a pas d'importance car il est préétabli par le module. N'indiquez qu'une seule instance de chaque justificatif.

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
    services:
```

```
- credentials:
    user_key: <array_of_lookup_queries>
    app_id: <array_of_lookup_queries>
    app_key: <array_of_lookup_queries>
...
```

Tableau 1.26. **credentials** champs de l'objet

Nom	Description	Exigée
user_key	Il s'agit d'un tableau de requêtes de recherche qui définit une clé d'utilisateur 3scale. Une clé d'utilisateur est communément appelée clé API.	En option
app_id	Il s'agit d'un tableau de requêtes de recherche qui définissent un identifiant d'application 3scale. Les identifiants d'application sont fournis par 3scale ou en utilisant un fournisseur d'identité comme Red Hat Single Sign-On (RH-SSO), ou OpenID Connect (OIDC). La résolution des requêtes de recherche spécifiées ici, chaque fois qu'elle est réussie et qu'elle se résout en deux valeurs, établit le **app_id** et le **app_key**.	En option
app_key	Il s'agit d'un tableau de requêtes de recherche qui définissent une clé d'application 3scale. Les clés d'application sans **app_id** résolu sont inutiles, c'est pourquoi ce champ ne doit être spécifié que lorsque **app_id** a été spécifié.	En option

1.20.6.8. Les requêtes de consultation du module WebAssembly 3scale

L'objet **lookup query** fait partie de n'importe quel champ de l'objet **credentials**. Il spécifie comment un champ d'identification donné doit être trouvé et traité. Lorsqu'elle est évaluée, une résolution réussie signifie qu'une ou plusieurs valeurs ont été trouvées. Une résolution échouée signifie qu'aucune valeur n'a été trouvée.

Les tableaux de **lookup queries** décrivent un court-circuit ou une relation : la résolution réussie de l'une des requêtes arrête l'évaluation de toutes les requêtes restantes et attribue la ou les valeurs au type de justificatif spécifié. Chaque requête du tableau est indépendante des autres.

Un site **lookup query** est constitué d'un seul champ, un objet source, qui peut faire partie d'un certain nombre de types de sources. Voir l'exemple suivant :

```
apiVersion: extensions.istio.io/v1alpha1
```

```
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
    services:
    - credentials:
        user_key:
          - <source_type>: <object>
          - <source_type>: <object>
          ...
        app_id:
          - <source_type>: <object>
          ...
        app_key:
          - <source_type>: <object>
          ...
    ...
```

1.20.6.9. L'objet source du module WebAssembly 3scale

Un objet **source** existe en tant que partie d'un tableau de sources dans n'importe quel champ de l'objet **credentials**. Le nom du champ de l'objet, appelé **source**-type, est l'un des suivants :

- **header**: La requête de recherche reçoit en entrée les en-têtes de requête HTTP.
- **query_string**: Le site **lookup query** reçoit en entrée les paramètres de la chaîne d'interrogation de l'URL.
- **filter**: Le site **lookup query** reçoit des métadonnées de filtrage en entrée.

Tous les objets de type **source** possèdent au moins les deux champs suivants :

Tableau 1.27. source-champs de l'objet type

Nom	Description	Exigée
keys	Un tableau de chaînes, chacune étant un **key**, faisant référence aux entrées trouvées dans les données d'entrée.	Oui

Nom	Description	Exigée
ops	Un tableau de **operations** qui effectue une correspondance d'entrée **key**. Le tableau est un pipeline dans lequel les opérations reçoivent des entrées et génèrent des sorties lors de l'opération suivante. Si **operation** ne fournit pas de sortie, **lookup query** est considéré comme ayant échoué. L'ordre des opérations dans le pipeline détermine l'ordre d'évaluation.	En option

Le nom du champ **filter** comporte une entrée obligatoire **path** pour indiquer le chemin dans les métadonnées que vous utilisez pour rechercher des données.

Lorsqu'un **key** correspond aux données d'entrée, les autres clés ne sont pas évaluées et l'algorithme de résolution des sources passe à l'exécution du **operations** (**ops**) spécifié, le cas échéant. Si aucun **ops** n'est spécifié, la valeur de résultat du **key** correspondant, s'il y en a un, est renvoyée.

Operations fournissent un moyen de spécifier certaines conditions et transformations pour les entrées dont vous disposez après que la première phase a consulté un site **key**. Utilisez **operations** lorsque vous avez besoin de transformer, décoder et affirmer des propriétés, mais ils ne fournissent pas un langage mature pour répondre à tous les besoins et manquent de *Turing-completeness*.

Une pile stocke les résultats de **operations**. Lorsqu'il est évalué, **lookup query** termine en attribuant la ou les valeurs au bas de la pile, en fonction du nombre de valeurs consommées par le titre.

1.20.6.10. L'objet des opérations du module WebAssembly 3scale

Chaque élément du tableau **ops** appartenant à un **source type** spécifique est un objet **operation** qui applique des transformations aux valeurs ou effectue des tests. Le nom du champ à utiliser pour un tel objet est le nom de l'objet **operation** lui-même, et toutes les valeurs sont les paramètres de l'objet **operation**, qui peuvent être des objets de structure, par exemple des cartes avec des champs et des valeurs, des listes ou des chaînes de caractères.

La plupart des sites **operations** traitent une ou plusieurs entrées et produisent une ou plusieurs sorties. Lorsqu'elles consomment des entrées ou produisent des sorties, elles travaillent avec une pile de valeurs : chaque valeur consommée par les opérations est extraite de la pile de valeurs et initialement remplie avec toutes les correspondances **source**. Les valeurs produites par les opérations sont poussées vers la pile. Les valeurs produites par ces opérations sont poussées vers la pile. D'autres sites **operations** ne consomment ni ne produisent de résultats autres que l'affirmation de certaines propriétés, mais ils inspectent une pile de valeurs.

NOTE

À la fin de la résolution, les valeurs reprises par l'étape suivante, comme l'attribution des valeurs à **app_id**, **app_key** ou **user_key**, proviennent des valeurs inférieures de la pile.

Il existe plusieurs catégories de **operations**:

- **decode**: Ils transforment une valeur d'entrée en la décodant pour obtenir un format différent.
- **string**: Ils prennent une chaîne de caractères en entrée et effectuent des transformations et des contrôles sur celle-ci.
- **stack**: Ils prennent un ensemble de valeurs en entrée et effectuent de multiples transformations de la pile et la sélection de positions spécifiques dans la pile.
- **check**: Ils affirment des propriétés sur des ensembles d'opérations sans effet de bord.
- **control**: Ces opérations permettent de modifier le flux d'évaluation.
- **format**: Ils analysent la structure spécifique du format des valeurs d'entrée et y recherchent des valeurs.

Toutes les opérations sont spécifiées par les identificateurs de nom sous forme de chaînes de caractères.

Ressources supplémentaires

- Opérations disponibles

1.20.6.11. L'objet mapping_rules du module WebAssembly de 3scale

L'objet **mapping_rules** fait partie de l'objet **service**. Il spécifie un ensemble de modèles de chemins REST, de métriques 3scale et d'incréments de comptage à utiliser lorsque les modèles correspondent.

Vous avez besoin de cette valeur si aucune configuration dynamique n'est fournie dans l'objet de premier niveau **system**. Si l'objet est fourni en plus de l'entrée de premier niveau **system**, l'objet **mapping_rules** est évalué en premier.

mapping_rules est un objet de type tableau. Chaque élément de ce tableau est un objet **mapping_rule**. Les règles de correspondance évaluées sur une demande entrante fournissent l'ensemble de 3scale **methods** pour l'autorisation et le rapport à *APIManager*. Lorsque plusieurs règles de correspondance se réfèrent au même **methods**, il y a une addition de **deltas** lors de l'appel à 3scale. Par exemple, si deux règles augmentent deux fois la méthode *Hits* avec **deltas** de 1 et 3, une seule entrée de méthode pour les Hits faisant rapport à 3scale a une **delta** de 4.

1.20.6.12. L'objet mapping_rule du module 3scale WebAssembly

L'objet **mapping_rule** fait partie d'un tableau dans l'objet **mapping_rules**.

Les champs de l'objet **mapping_rule** précisent les informations suivantes :

- Le site *HTTP request method* doit être assorti.
- Un modèle de correspondance avec le chemin d'accès.
- Les méthodes 3scale à déclarer ainsi que le montant à déclarer. L'ordre dans lequel vous spécifiez les zones détermine l'ordre d'analyse.

Tableau 1.28. mapping_rule champs de l'objet

Nom	Description	Exigée
method	Spécifie une chaîne représentant une méthode de requête HTTP, également connue sous le nom de verbe. Les valeurs acceptées correspondent à l'un des noms de méthode HTTP acceptés, sans tenir compte de la casse. La valeur spéciale "any" correspond à n'importe quelle méthode.	Oui
pattern	Le motif qui doit correspondre au composant du chemin URI de la demande HTTP. Ce motif suit la même syntaxe que celle documentée par 3scale. Il autorise les caractères génériques (utilisation de l'astérisque (*)) en utilisant n'importe quelle séquence de caractères entre accolades, comme **{this}**.	Oui
usages	Une liste d'objets **usage**. Lorsque la règle correspond, toutes les méthodes avec leur **deltas** sont ajoutées à la liste des méthodes envoyées à 3scale pour autorisation et rapport. Incorporer l'objet **usages** avec les champs obligatoires suivants : ● **name**: Le nom du système **method** à signaler. ● **delta**: Pour savoir de combien augmenter le site **method**.	Oui
last	Indique si la correspondance réussie de cette règle doit interrompre l'évaluation d'autres règles de mappage.	Booléen facultatif. La valeur par défaut est **false**

L'exemple suivant est indépendant des hiérarchies existantes entre les méthodes dans 3scale. En d'autres termes, tout ce qui est exécuté du côté de 3scale n'affectera pas cet exemple. Par exemple, la métrique *Hits* peut être le parent de toutes les méthodes, elle stocke donc 4 occurrences en raison de la somme de toutes les méthodes signalées dans la demande autorisée et appelle le point de terminaison de l'API 3scale **Authrep**.

L'exemple ci-dessous utilise une requête **GET** vers un chemin d'accès, **/products/1/sold**, qui répond à toutes les règles.

mapping_rules GET exemple de demande

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
    mapping_rules:
      - method: GET
        pattern: /
        usages:
          - name: hits
            delta: 1
      - method: GET
        pattern: /products/
        usages:
          - name: products
            delta: 1
      - method: ANY
        pattern: /products/{id}/sold
        usages:
          - name: sales
            delta: 1
          - name: products
            delta: 1
    ...
```

Toutes les adresses **usages** sont ajoutées à la requête que le module adresse à 3scale avec les données d'utilisation comme suit :

- Hits : 1
- produits : 2
- ventes : 1

1.20.7. Les exemples de modules 3scale WebAssembly pour les cas d'utilisation des références

Vous passerez la majeure partie de votre temps à appliquer des étapes de configuration pour obtenir des informations d'identification dans les demandes adressées à vos services.

Les exemples suivants sont tirés du site **credentials** et peuvent être modifiés pour s'adapter à des cas d'utilisation spécifiques.

Vous pouvez les combiner toutes, bien que lorsque vous spécifiez plusieurs objets sources avec leur propre **lookup queries**, ils sont évalués dans l'ordre jusqu'à ce que l'un d'entre eux soit résolu avec succès.

1.20.7.1. Clé API (user_key) dans les paramètres de la chaîne de requête

L'exemple suivant recherche une adresse **user_key** dans un paramètre de chaîne de requête ou un en-tête du même nom :

```
credentials:
  user_key:
    - query_string:
        keys:
          - user_key
    - header:
        keys:
          - user_key
```

1.20.7.2. ID et clé de l'application

L'exemple suivant permet de rechercher les informations d'identification **app_key** et **app_id** dans une requête ou des en-têtes.

```
credentials:
  app_id:
    - header:
        keys:
          - app_id
    - query_string:
        keys:
          - app_id
  app_key:
    - header:
        keys:
          - app_key
    - query_string:
        keys:
          - app_key
```

1.20.7.3. En-tête d'autorisation

Une requête comprend un en-tête **app_id** et **app_key** dans un en-tête **authorization**. S'il y a au moins une ou deux valeurs produites à la fin, alors vous pouvez assigner l'en-tête **app_key**.

La résolution ici attribue le **app_key** s'il y a un ou deux résultats à la fin.

L'en-tête **authorization** spécifie une valeur avec le type d'autorisation et sa valeur est encodée comme **Base64**. Cela signifie que vous pouvez diviser la valeur par un caractère espace, prendre la deuxième sortie et la diviser à nouveau en utilisant un deux-points (:) comme séparateur. Par exemple, si vous utilisez le format **app_id:app_key**, l'en-tête ressemble à l'exemple suivant pour **credential**:

```
aladdin:opensesame:  Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
```

Vous devez utiliser des noms de champs d'en-tête en minuscules, comme indiqué dans l'exemple suivant :

```
credentials:
  app_id:
    - header:
        keys:
```

```
      - authorization
    ops:
      - split:
          separator: " "
          max: 2
      - length:
          min: 2
      - drop:
          head: 1
      - base64_urlsafe
      - split:
          max: 2
app_key:
  - header:
      keys:
        - app_key
```

L'exemple de cas d'utilisation précédent porte sur les en-têtes d'un site **authorization**:

1. Il prend sa valeur de chaîne et la divise par un espace, en vérifiant qu'il génère au moins deux valeurs d'un type **credential** et le type **credential** lui-même, puis en abandonnant le type **credential**.

2. Il décode ensuite la deuxième valeur contenant les données dont il a besoin, et la divise en utilisant le caractère deux-points (:) pour avoir une pile d'opérations comprenant d'abord le **app_id**, puis le **app_key**, s'il existe.

    a. Si **app_key** n'existe pas dans l'en-tête d'autorisation, ses sources spécifiques sont vérifiées, par exemple l'en-tête avec la clé **app_key** dans ce cas.

3. Pour ajouter des conditions supplémentaires à **credentials**, autorisez les autorisations **Basic**, où **app_id** est soit **aladdin**, soit **admin**, ou tout **app_id** d'une longueur d'au moins 8 caractères.

4. **app_key** doit contenir une valeur et un minimum de 64 caractères, comme le montre l'exemple suivant :

```
credentials:
  app_id:
    - header:
        keys:
          - authorization
        ops:
          - split:
              separator: " "
              max: 2
          - length:
              min: 2
          - reverse
          - glob:
            - Basic
          - drop:
              tail: 1
          - base64_urlsafe
          - split:
              max: 2
          - test:
```

```
        if:
          length:
            min: 2
        then:
          - strlen:
              max: 63
          - or:
              - strlen:
                  min: 1
              - drop:
                  tail: 1
      - assert:
        - and:
          - reverse
          - or:
            - strlen:
                min: 8
            - glob:
              - aladdin
              - admin
```

5. Après avoir récupéré la valeur de l'en-tête **authorization**, vous obtenez un type **Basic credential** en inversant la pile de manière à ce que le type soit placé au-dessus.

6. Exécutez une correspondance globale. Lorsqu'elle est validée, et que l'identifiant est décodé et divisé, vous obtenez le **app_id** en bas de la pile, et potentiellement le **app_key** en haut.

7. Exécutez un **test:** s'il y a deux valeurs dans la pile, ce qui signifie qu'un **app_key** a été acquis.

    a. Assurez-vous que la longueur de la chaîne est comprise entre 1 et 63, y compris **app_id** et **app_key**. Si la longueur de la clé est nulle, abandonnez-la et continuez comme si aucune clé n'existait. S'il n'y avait qu'un **app_id** et pas de **app_key**, la branche else manquante indique que le test a réussi et l'évaluation se poursuit.

La dernière opération, **assert**, indique qu'aucun effet secondaire ne se retrouve dans la pile. Vous pouvez alors modifier la pile :

1. Inversez la pile pour que le site **app_id** soit au sommet.

    a. Qu'il y ait ou non un **app_key**, l'inversion de la pile garantit que **app_id** se trouve au sommet.

2. Utilisez **and** pour préserver le contenu de la pile entre les tests.
   Utilisez ensuite l'une des possibilités suivantes :

    - Assurez-vous que **app_id** a une longueur de chaîne d'au moins 8.
    - Assurez-vous que **app_id** correspond à **aladdin** ou **admin**.

1.20.7.4. Cas d'utilisation d'OpenID Connect (OIDC)

Pour Service Mesh et l'adaptateur Istio 3scale, vous devez déployer un site **RequestAuthentication** comme indiqué dans l'exemple suivant, en renseignant vos propres données de charge de travail et **jwtRules**:

```
apiVersion: security.istio.io/v1beta1
```

```
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: info
spec:
  selector:
    matchLabels:
      app: productpage
  jwtRules:
  - issuer: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak
    jwksUri: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak/protocol/openid-
connect/certs
```

Lorsque vous appliquez le **RequestAuthentication**, il configure **Envoy** avec un plugin natif pour valider les jetons **JWT**. Le proxy valide tout avant d'exécuter le module, de sorte que les demandes qui échouent ne parviennent pas au module 3scale WebAssembly.

Lorsqu'un jeton **JWT** est validé, le proxy stocke son contenu dans un objet de métadonnées interne, avec une entrée dont la clé dépend de la configuration spécifique du plugin. Ce cas d'utilisation vous permet de rechercher des objets de structure avec une seule entrée contenant un nom de clé inconnu.

Le 3scale **app_id** pour l'OIDC correspond à l'OAuth **client_id**. Ceci se trouve dans les champs **azp** ou **aud** des jetons **JWT**.

Pour obtenir le champ **app_id** à partir du filtre d'authentification natif d'Envoy **JWT**, voir l'exemple suivant :

```
credentials:
  app_id:
    - filter:
        path:
          - envoy.filters.http.jwt_authn
          - "0"
        keys:
          - azp
          - aud
        ops:
          - take:
              head: 1
```

L'exemple indique au module d'utiliser le type de source **filter** pour rechercher les métadonnées de filtrage d'un objet provenant du plugin natif d'authentification **JWT** spécifique à **Envoy**. Ce plugin inclut le jeton **JWT** dans un objet de structure avec une seule entrée et un nom préconfiguré. Utilisez **0** pour spécifier que vous n'accéderez qu'à l'entrée unique.

La valeur résultante est une structure pour laquelle vous résoudrez deux champs :

- **azp**: La valeur où se trouve **app_id**.
- **aud**: La valeur où cette information peut également être trouvée.

Cette opération permet de s'assurer qu'une seule valeur est conservée pour l'affectation.

1.20.7.5. Récupérer le jeton JWT dans un en-tête

Certaines configurations peuvent avoir des processus de validation pour les jetons **JWT** où le jeton validé atteindrait ce module par le biais d'un en-tête au format JSON.

Pour obtenir le site **app_id**, voir l'exemple suivant :

```
credentials:
  app_id:
    - header:
        keys:
          - x-jwt-payload
        ops:
          - base64_urlsafe
          - json:
            - keys:
              - azp
              - aud
          - take:
              head: 1
```

1.20.8. configuration minimale de travail du module WebAssembly 3scale

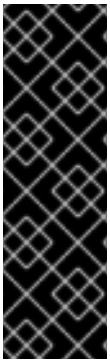
Ce qui suit est un exemple de configuration minimale de travail d'un module WebAssembly 3scale. Vous pouvez copier et coller cet exemple et le modifier pour l'adapter à votre propre configuration.

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  url: oci://registry.redhat.io/3scale-amp2/3scale-auth-wasm-rhel8:0.0.3
  imagePullSecret: <optional_pull_secret_resource>
  phase: AUTHZ
  priority: 100
  selector:
    labels:
      app: <product_page>
  pluginConfig:
    api: v1
    system:
      name: <system_name>
      upstream:
        name: outbound|443||multitenant.3scale.net
        url: https://istiodevel-admin.3scale.net/
        timeout: 5000
      token: <token>
    backend:
      name: <backend_name>
      upstream:
        name: outbound|443||su1.3scale.net
        url: https://su1.3scale.net/
        timeout: 5000
      extensions:
      - no_body
    services:
    - id: '2555417834780'
```

```
authorities:
- "*"
credentials:
  user_key:
    - query_string:
        keys:
          - <user_key>
    - header:
        keys:
          - <user_key>
  app_id:
    - query_string:
        keys:
          - <app_id>
    - header:
        keys:
          - <app_id>
  app_key:
    - query_string:
        keys:
          - <app_key>
    - header:
        keys:
          - <app_key>
```

1.21. UTILISATION DE L'ADAPTATEUR 3SCALE ISTIO

L'adaptateur 3scale Istio est un adaptateur optionnel qui vous permet d'étiqueter un service fonctionnant dans Red Hat OpenShift Service Mesh et d'intégrer ce service avec la solution 3scale API Management. Il n'est pas requis pour Red Hat OpenShift Service Mesh.



IMPORTANT

Vous ne pouvez utiliser l'adaptateur 3scale Istio qu'avec Red Hat OpenShift Service Mesh versions 2.0 et inférieures. Le composant Mixer a été déprécié dans la version 2.0 et supprimé dans la version 2.1. Pour Red Hat OpenShift Service Mesh versions 2.1.0 et ultérieures, vous devez utiliser le module 3scale WebAssembly.

Si vous souhaitez activer le cache backend 3scale avec l'adaptateur Istio 3scale, vous devez également activer la politique Mixer et la télémétrie Mixer. Voir Déployer le plan de contrôle Red Hat OpenShift Service Mesh.

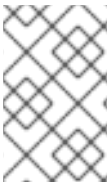
1.21.1. Intégrer l'adaptateur 3scale avec Red Hat OpenShift Service Mesh

Vous pouvez utiliser ces exemples pour configurer les requêtes vers vos services en utilisant l'adaptateur Istio 3scale.

Prérequis :

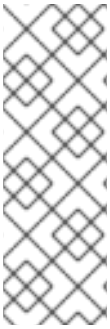
- Red Hat OpenShift Service Mesh version 2.x
- Un compte 3scale fonctionnel(SaaS ou 3scale 2.9 On-Premises)
- L'activation du backend cache nécessite 3scale 2.9 ou plus

- Prérequis de Red Hat OpenShift Service Mesh
- Assurez-vous que l'application de la politique Mixer est activée. La section Mise à jour de l'application de la politique Mixer fournit des instructions pour vérifier l'état actuel de l'application de la politique Mixer et activer l'application de la politique.
- La politique de mixage et la télémétrie doivent être activées si vous utilisez un plugin de mixage.
    - Vous devrez configurer correctement le plan de contrôle Service Mesh (SMCP) lors de la mise à niveau.



NOTE

Pour configurer l'adaptateur 3scale Istio, reportez-vous aux ressources personnalisées Red Hat OpenShift Service Mesh pour obtenir des instructions sur l'ajout de paramètres d'adaptateur au fichier de ressources personnalisées.



NOTE

Portez une attention particulière à la ressource **kind: handler**. Vous devez la mettre à jour avec les informations d'identification de votre compte 3scale. Vous pouvez optionnellement ajouter une ressource **service_id** à un handler, mais ceci n'est conservé que pour la compatibilité ascendante, puisque cela rendrait le handler utile uniquement pour un seul service dans votre compte 3scale. Si vous ajoutez **service_id** à un handler, l'activation de 3scale pour d'autres services nécessite la création d'autres handlers avec des **service_ids** différents.

Utilisez un seul gestionnaire par compte 3scale en suivant les étapes ci-dessous :

Procédure

1. Créez un gestionnaire pour votre compte 3scale et indiquez les informations d'identification de votre compte. N'indiquez pas d'identifiant de service.

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

   En option, vous pouvez fournir un champ **backend_url** dans la section *params* pour remplacer l'URL fournie par la configuration 3scale. Cela peut être utile si l'adaptateur fonctionne sur le même cluster que l'instance 3scale sur site, et que vous souhaitez utiliser le DNS interne du cluster.

2. Modifiez ou corrigez la ressource Déploiement de tous les services appartenant à votre compte 3scale comme suit :

a. Ajouter l'étiquette **"service-mesh.3scale.net/service-id"** avec une valeur correspondant à une **service_id** valide.

b. Ajoutez l'étiquette **"service-mesh.3scale.net/credentials"** dont la valeur est celle de *name of the handler resource* de l'étape 1.

3. Faites l'étape 2 pour le lier à vos identifiants de compte 3scale et à son identifiant de service, lorsque vous avez l'intention d'ajouter d'autres services.

4. Modifiez la configuration de la règle avec votre configuration 3scale pour envoyer la règle au gestionnaire 3scale.

Exemple de configuration d'une règle

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

1.21.1.1. Générer des ressources personnalisées à 3 échelles

L'adaptateur comprend un outil qui vous permet de générer les ressources personnalisées **handler**, **instance** et **rule**.

Tableau 1.29. Utilisation

Option	Description	Exigée	Valeur par défaut
-h, --help	Produit une sortie d'aide pour les options disponibles	Non	
--name	Nom unique pour cet URL, paire de jetons	Oui	
-n, --namespace	Espace de noms pour générer des modèles	Non	istio-system
-t, --token	jeton d'accès 3scale	Oui	
-u, --url	uRL du portail administratif 3scale	Oui	

Option	Description	Exigée	Valeur par défaut
--backend-url	uRL du backend 3scale. Si elle est définie, elle remplace la valeur lue dans la configuration du système	Non	
-s, --service	iD API/Service 3scale	Non	
--auth	modèle d'authentification à 3 niveaux à spécifier (1=clé API, 2=identification de l'application/clé de l'application, 3=OIDC)	Non	Hybride
-o, --output	Fichier dans lequel enregistrer les manifestes produits	Non	Sortie standard
--version	Affiche la version de l'interface de programmation et quitte immédiatement	Non	

1.21.1.1.1. Générer des modèles à partir d'exemples d'URL

NOTE

- Exécutez les commandes suivantes via **oc exec** à partir de l'image du conteneur de l'adaptateur 3scale dans Générer des manifestes à partir d'un adaptateur déployé.
- Utilisez la commande **3scale-config-gen** pour éviter les erreurs de syntaxe et d'indentation de YAML.
- Vous pouvez omettre le site **--service** si vous utilisez les annotations.
- Cette commande doit être invoquée à partir de l'image du conteneur via **oc exec**.

Procédure

- Utilisez la commande **3scale-config-gen** pour autogénérer des fichiers modèles permettant à la paire token, URL d'être partagée par plusieurs services en tant que gestionnaire unique :

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

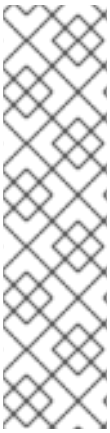
- L'exemple suivant génère les modèles avec l'identifiant du service intégré dans le gestionnaire :

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

Ressources supplémentaires

- Jetons.

1.21.1.2. Générer des manifestes à partir d'un adaptateur déployé



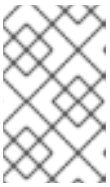
NOTE

- **NAME** est un identifiant que vous utilisez pour vous identifier auprès du service que vous gérez avec 3scale.
- La référence **CREDENTIALS_NAME** est un identifiant qui correspond à la section **match** dans la configuration de la règle. Si vous utilisez l'outil CLI, l'identifiant **NAME** est automatiquement utilisé.
- Sa valeur n'a pas besoin d'être spécifique : la valeur de l'étiquette doit simplement correspondre au contenu de la règle. Voir Routage du trafic de service à travers l'adaptateur pour plus d'informations.

1. Exécutez cette commande pour générer des manifestes à partir d'un adaptateur déployé dans l'espace de noms **istio-system**:

```
$ 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. Cela produira des exemples de sortie dans le terminal. Modifiez ces exemples si nécessaire et créez les objets à l'aide de la commande **oc create**.
3. Lorsque la requête atteint l'adaptateur, ce dernier doit savoir comment le service correspond à une API sur 3scale. Vous pouvez fournir cette information de deux façons :
   a. Étiqueter la charge de travail (recommandé)
   b. Le code dur du gestionnaire est le suivant **service_id**
4. Mettre à jour la charge de travail avec les annotations requises :



NOTE

Il suffit de mettre à jour l'identifiant de service fourni dans cet exemple s'il n'est pas déjà intégré dans le gestionnaire. **The setting in the handler takes precedence**.

```
$ export CREDENTIALS_NAME="replace-me"
export SERVICE_ID="replace-me"
export DEPLOYMENT="replace-me"
```

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

1.21.1.3. Acheminement du trafic de service via l'adaptateur

Suivez les étapes suivantes pour générer du trafic pour votre service grâce à l'adaptateur 3scale.

Conditions préalables

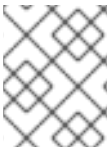
- Les informations d'identification et l'identifiant de service de votre administrateur 3scale.

Procédure

1. Correspondre à la règle **destination.labels["service-mesh.3scale.net/credentials"] == "threescale"** que vous avez précédemment créée dans la configuration, dans la ressource **kind: rule**.
2. Ajoutez l'étiquette ci-dessus à **PodTemplateSpec** sur le Déploiement de la charge de travail cible pour intégrer un service. la valeur, **threescale**, fait référence au nom du gestionnaire généré. Ce gestionnaire stocke le jeton d'accès requis pour appeler 3scale.
3. Ajoutez l'étiquette **destination.labels["service-mesh.3scale.net/service-id"] == "replace-me"** à la charge de travail pour transmettre l'identifiant du service à l'adaptateur via l'instance au moment de la demande.

1.21.2. Configurer les paramètres d'intégration dans 3scale

Suivez cette procédure pour configurer les paramètres d'intégration de 3scale.



NOTE

Pour les clients SaaS de 3scale, Red Hat OpenShift Service Mesh est activé dans le cadre du programme Early Access.

Procédure

1. Naviguez vers **[your_API_name] → Integration**
2. Cliquez sur **Settings**.
3. Sélectionnez l'option **Istio** sous *Deployment*.
   - L'option **API Key (user_key)** sous *Authentication* est sélectionnée par défaut.
4. Cliquez sur **Update Product** pour enregistrer votre sélection.
5. Cliquez sur **Configuration**.
6. Cliquez sur **Update Configuration**.

1.21.3. Comportement de mise en cache

Les réponses des API du système 3scale sont mises en cache par défaut dans l'adaptateur. Les entrées sont supprimées du cache lorsqu'elles sont plus anciennes que la valeur **cacheTTLSeconds**. Par défaut également, le rafraîchissement automatique des entrées mises en cache sera tenté quelques secondes avant leur expiration, sur la base de la valeur **cacheRefreshSeconds**. Vous pouvez désactiver le rafraîchissement automatique en définissant cette valeur à un niveau supérieur à la valeur **cacheTTLSeconds**.

La mise en cache peut être entièrement désactivée en donnant à **cacheEntriesMax** une valeur non positive.

En utilisant le processus de rafraîchissement, les valeurs mises en cache dont les hôtes deviennent inaccessibles seront réessayées avant d'être finalement purgées lorsqu'elles auront dépassé leur date d'expiration.

1.21.4. Authentification des demandes

Cette version prend en charge les méthodes d'authentification suivantes :

- **Standard API Keys**les systèmes d'authentification sont des chaînes ou des hachages aléatoires uniques qui servent d'identificateur et de jeton secret.
- **Application identifier and key pairs**es chaînes d'identification immuables et les chaînes de clés secrètes mutables.
- **OpenID authentication method**chaîne d'identification du client analysée à partir du jeton Web JSON.

1.21.4.1. Application de modèles d'authentification

Modifiez la ressource personnalisée **instance**, comme illustré dans les exemples de méthodes d'authentification suivants, pour configurer le comportement d'authentification. Vous pouvez accepter les informations d'authentification de :

- En-têtes de la demande
- Paramètres de la demande
- Les en-têtes et les paramètres de la requête

NOTE

Lorsque vous spécifiez des valeurs à partir d'en-têtes, elles doivent être en minuscules. Par exemple, si vous souhaitez envoyer un en-tête sous la forme **User-Key**, il doit être référencé dans la configuration sous la forme **request.headers["user-key"]**.

1.21.4.1.1. Méthode d'authentification de la clé API

Service Mesh recherche la clé API dans les paramètres de requête et les en-têtes de demande spécifiés dans l'option **user** du paramètre de ressource personnalisée **subject**. Il vérifie les valeurs dans l'ordre indiqué dans le fichier de ressources personnalisées. Vous pouvez limiter la recherche de la clé API aux paramètres de requête ou aux en-têtes de requête en omettant l'option indésirable.

Dans cet exemple, Service Mesh recherche la clé API dans le paramètre de requête **user_key**. Si la clé API ne figure pas dans le paramètre de la requête, Service Mesh vérifie alors l'en-tête **user-key**.

Exemple de méthode d'authentification par clé API

```
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"
```

Si vous souhaitez que l'adaptateur examine un paramètre de requête ou un en-tête de requête différent, modifiez le nom en conséquence. Par exemple, pour vérifier la clé API dans un paramètre de requête nommé "key", remplacez **request.query_params["user_key"]** par **request.query_params["key"]**.

1.21.4.1.2. Méthode d'authentification de l'ID de l'application et de la paire de clés de l'application

Service Mesh recherche l'identifiant et la clé de l'application dans les paramètres de la requête et les en-têtes de la demande, comme spécifié dans l'option **properties** du paramètre de ressource personnalisée **subject**. La clé d'application est facultative. Il vérifie les valeurs dans l'ordre indiqué dans le fichier de ressources personnalisées. Vous pouvez limiter la recherche des informations d'identification aux paramètres de requête ou aux en-têtes de requête en n'incluant pas l'option "unwanted".

Dans cet exemple, Service Mesh recherche d'abord l'identifiant et la clé de l'application dans les paramètres de la requête, puis passe aux en-têtes de la requête si nécessaire.

Exemple de méthode d'authentification par ID d'application et paire de clés d'application

```
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"
```

Si vous souhaitez que l'adaptateur examine un paramètre de requête ou un en-tête de requête différent, modifiez le nom en conséquence. Par exemple, pour vérifier l'ID de l'application dans un paramètre de requête nommé **identification**, remplacez **request.query_params["app_id"]** par **request.query_params["identification"]**.

1.21.4.1.3. Méthode d'authentification OpenID

Pour utiliser le champ *OpenID Connect (OIDC) authentication method* , utilisez la valeur **properties** sur le champ **subject** pour définir **client_id**, et éventuellement **app_key**.

Vous pouvez manipuler cet objet à l'aide des méthodes décrites précédemment. Dans l'exemple de configuration ci-dessous, l'identifiant du client (ID de l'application) est extrait du jeton Web JSON (JWT) sous l'étiquette *azp*. Vous pouvez le modifier si nécessaire.

Exemple de méthode d'authentification OpenID

```
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"] | ""
```

Pour que cette intégration fonctionne correctement, l'OIDC doit toujours être effectué dans 3scale pour que le client soit créé dans le fournisseur d'identité (IdP). Vous devez créer une autorisation de demande pour le service que vous voulez protéger dans le même espace de noms que ce service. Le JWT est transmis dans l'en-tête **Authorization** de la requête.

Dans l'exemple **RequestAuthentication** défini ci-dessous, remplacez **issuer**, **jwksUri**, et **selector** comme il convient.

Exemple de politique OpenID

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: info
spec:
  selector:
    matchLabels:
      app: productpage
  jwtRules:
  - issuer: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak
    jwksUri: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak/protocol/openid-
connect/certs
```

1.21.4.1.4. Méthode d'authentification hybride

Vous pouvez choisir de ne pas appliquer une méthode d'authentification particulière et d'accepter toutes les informations d'identification valides pour l'une ou l'autre méthode. Si une clé API et une paire ID d'application/clé d'application sont fournies, Service Mesh utilise la clé API.

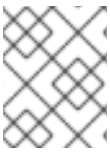
Dans cet exemple, Service Mesh vérifie la présence d'une clé API dans les paramètres de la requête, puis dans les en-têtes de la requête. S'il n'y a pas de clé API, il recherche alors un identifiant et une clé d'application dans les paramètres de la requête, puis dans les en-têtes de la requête.

Exemple de méthode d'authentification hybride

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

1.21.5. 3scale Adapter metrics

L'adaptateur, par défaut, rapporte diverses mesures Prometheus qui sont exposées sur le port **8080** au point de terminaison /**metrics**. Ces métriques donnent un aperçu de la façon dont les interactions entre l'adaptateur et 3scale se déroulent. Le service est étiqueté pour être automatiquement découvert et scrappé par Prometheus.



NOTE

Il y a des changements incompatibles dans les métriques 3scale Istio Adapter depuis les versions précédentes de Service Mesh 1.x.

Dans Prometheus, les métriques ont été renommées avec un ajout pour le cache du backend, de sorte que les métriques suivantes existent depuis Service Mesh 2.0 :

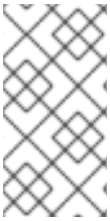
Tableau 1.30. Métriques de Prométhée

Métrique	Type	Description
threescale_latency	Histogramme	Temps de latence entre l'adaptateur et 3scale.
threescale_http_total	Compteur	Codes de réponse HTTP pour les requêtes adressées au backend 3scale.

Métrique	Type	Description
threescale_system_cache_hits	Compteur	Nombre total de demandes adressées au système 3scale à partir du cache de configuration.
threescale_backend_cache_hits	Compteur	Nombre total de requêtes adressées à 3scale backend et récupérées dans le cache du backend.

1.21.6. 3scale backend cache

Le cache backend 3scale fournit un cache d'autorisation et de rapport pour les clients de l'API de gestion des services 3scale. Ce cache est intégré dans l'adaptateur pour permettre des temps de latence plus faibles dans les réponses dans certaines situations, à condition que l'administrateur soit prêt à accepter les compromis.



NOTE

le cache backend 3scale est désactivé par défaut. la fonctionnalité 3scale backend cache troque l'imprécision dans la limitation du taux et la perte potentielle d'occurrences depuis la dernière vidange pour une faible latence et une consommation plus élevée de ressources dans le processeur et la mémoire.

1.21.6.1. Avantages de l'activation de la mémoire cache

Les avantages de l'activation de la mémoire cache sont les suivants :

- Activez le cache backend lorsque vous constatez que les latences sont élevées lors de l'accès aux services gérés par l'adaptateur Istio 3scale.
- L'activation du backend cache empêchera l'adaptateur de vérifier continuellement avec le gestionnaire d'API 3scale les autorisations de demande, ce qui réduira la latence.
    - Cela crée un cache en mémoire des autorisations 3scale pour l'adaptateur Istio 3scale à stocker et à réutiliser avant de tenter de contacter le gestionnaire d'API 3scale pour les autorisations. Les autorisations prendront alors beaucoup moins de temps pour être accordées ou refusées.
- La mise en cache du backend est utile dans les cas où vous hébergez le gestionnaire d'API 3scale dans un autre emplacement géographique que le maillage de service exécutant l'adaptateur Istio 3scale.
    - C'est généralement le cas avec la plateforme 3scale Hosted (SaaS), mais aussi si un utilisateur héberge son 3scale API manager dans un autre cluster situé dans un emplacement géographique différent, dans une zone de disponibilité différente, ou dans tous les cas où la surcharge du réseau pour atteindre le 3scale API manager est perceptible.

1.21.6.2. Compromis pour des temps de latence plus faibles

Les points suivants sont des compromis pour obtenir des temps de latence plus faibles :

- L'état d'autorisation de chaque adaptateur 3scale est mis à jour à chaque fois qu'une vidange a lieu.
    - Cela signifie que deux instances ou plus de l'adaptateur introduiront plus d'imprécision entre les périodes de rinçage.
    - Il y a plus de risques qu'un trop grand nombre de demandes soient accordées, dépassant les limites et introduisant un comportement erratique, ce qui conduit à ce que certaines demandes soient acceptées et d'autres non, en fonction de l'adaptateur qui traite chaque demande.
- Un cache d'adaptateur qui ne peut pas purger ses données et mettre à jour ses informations d'autorisation risque de s'arrêter ou de s'écraser sans signaler ses informations au gestionnaire d'API.
- Une politique d'échec ouvert ou d'échec fermé sera appliquée lorsqu'un cache d'adaptateur ne peut pas déterminer si une demande doit être accordée ou refusée, éventuellement en raison de problèmes de connectivité réseau lors de la prise de contact avec le gestionnaire d'API.
- En cas d'absence de cache, généralement juste après le démarrage de l'adaptateur ou après une longue période d'absence de connectivité, les temps de latence augmentent pour interroger le gestionnaire d'API.
- Un cache d'adaptateur doit effectuer beaucoup plus de travail pour calculer les autorisations qu'il ne le ferait en l'absence d'un cache activé, ce qui sollicite les ressources du processeur.
- Les besoins en mémoire augmenteront proportionnellement à la combinaison de la quantité de limites, d'applications et de services gérés par le cache.

1.21.6.3. Paramètres de configuration du cache du backend

Les points suivants expliquent les paramètres de configuration du cache du backend :

- Vous trouverez les paramètres pour configurer le cache backend dans les options de configuration de 3scale.
- Les trois derniers paramètres contrôlent l'activation du cache en arrière-plan :
    - **PARAM_USE_CACHE_BACKEND** - est fixé à true pour activer le backend cache.
    - **PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS** - définit le temps en secondes entre les tentatives consécutives de vidage des données du cache au gestionnaire de l'API.
    - **PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED** - définir s'il faut ou non autoriser/ouvrir ou refuser/fermer les demandes aux services lorsqu'il n'y a pas assez de données mises en cache et que le gestionnaire d'API 3scale ne peut pas être atteint.

1.21.7. 3scale Istio Adapter Émulation APIcast

L'adaptateur 3scale Istio fonctionne comme APIcast lorsque les conditions suivantes se produisent :

- Lorsqu'une requête ne peut correspondre à aucune règle de correspondance définie, le code HTTP renvoyé est 404 Not Found. Ce code était auparavant 403 Forbidden.
- Lorsqu'une demande est refusée parce qu'elle dépasse les limites fixées, le code HTTP renvoyé est 429 Too Many Requests (trop de demandes). Ce code était auparavant 403 Forbidden.

- Lors de la génération de modèles par défaut via l'interface de programmation, des traits de soulignement seront utilisés à la place des tirets pour les en-têtes, par exemple : **user_key** au lieu de **user-key**.

1.21.8. vérification de l'adaptateur Istio 3scale

Vous voudrez peut-être vérifier si l'adaptateur 3scale Istio fonctionne comme prévu. Si votre adaptateur ne fonctionne pas, suivez les étapes suivantes pour résoudre le problème.

Procédure

1. Assurez-vous que le pod *3scale-adapter* fonctionne dans l'espace de noms du plan de contrôle Service Mesh :

```
oc get pods -n <istio-system>
```

2. Vérifiez que le pod *3scale-adapter* a imprimé des informations sur son démarrage, telles que sa version :

```
oc logs <istio-system>
```

3. Lors de l'exécution de requêtes vers les services protégés par l'intégration de l'adaptateur 3scale, essayez toujours les requêtes qui n'ont pas les bonnes informations d'identification et assurez-vous qu'elles échouent. Vérifiez les journaux de l'adaptateur 3scale pour recueillir des informations supplémentaires.

Ressources supplémentaires

- Inspecter les registres des gousses et des conteneurs .

1.21.9. liste de contrôle de dépannage de l'adaptateur Istio 3scale

En tant qu'administrateur installant l'adaptateur 3scale Istio, il y a un certain nombre de scénarios qui peuvent faire que votre intégration ne fonctionne pas correctement. Utilisez la liste suivante pour dépanner votre installation :

- Indentation YAML incorrecte.
- Sections YAML manquantes.
- J'ai oublié d'appliquer les changements dans le YAML au cluster.
- Oublié d'étiqueter les charges de travail de service avec la clé **service-mesh.3scale.net/credentials**.
- Oublié d'étiqueter les charges de travail de service avec **service-mesh.3scale.net/service-id** lorsque l'on utilise des gestionnaires qui ne contiennent pas de **service_id** afin qu'ils soient réutilisables par compte.
- La ressource personnalisée *Rule* pointe vers le mauvais gestionnaire ou les mauvaises ressources personnalisées d'instance, ou les références n'ont pas le suffixe de l'espace de noms correspondant.

- La section *Rule* custom resource **match** ne peut pas correspondre au service que vous configurez, ou elle pointe vers une charge de travail de destination qui n'est pas en cours d'exécution ou qui n'existe pas.

- Mauvais jeton d'accès ou URL pour le portail d'administration 3scale dans le gestionnaire.

- La section **params/subject/properties** de la ressource personnalisée *Instance* ne répertorie pas les bons paramètres pour **app_id**, **app_key** ou **client_id**, soit parce qu'ils spécifient le mauvais emplacement, comme les paramètres de requête, les en-têtes et les demandes d'autorisation, soit parce que les noms des paramètres ne correspondent pas aux requêtes utilisées pour les tests.

- Ne pas utiliser le générateur de configuration sans se rendre compte qu'il se trouve dans l'image du conteneur de l'adaptateur et qu'il a besoin de **oc exec** pour l'invoquer.

1.22. DÉPANNAGE DE VOTRE MAILLAGE DE SERVICES

Cette section décrit comment identifier et résoudre les problèmes courants dans Red Hat OpenShift Service Mesh. Utilisez les sections suivantes pour vous aider à dépanner et à déboguer les problèmes lors du déploiement de Red Hat OpenShift Service Mesh sur OpenShift Container Platform.

1.22.1. Comprendre les versions de Service Mesh

Afin de comprendre quelle version de Red Hat OpenShift Service Mesh vous avez déployée sur votre système, vous devez comprendre comment chacune des versions des composants est gérée.

- **Operator** version - La version la plus récente de l'opérateur est 2.3.3. Le numéro de version de l'opérateur indique uniquement la version de l'opérateur actuellement installé. Étant donné que l'Opérateur Red Hat OpenShift Service Mesh prend en charge plusieurs versions du plan de contrôle Service Mesh, la version de l'Opérateur ne détermine pas la version de vos ressources **ServiceMeshControlPlane** déployées.

IMPORTANT

La mise à niveau vers la dernière version de l'opérateur applique automatiquement les mises à jour des correctifs, mais ne met pas automatiquement à niveau votre plan de contrôle Service Mesh vers la dernière version mineure.

- **ServiceMeshControlPlane** version - La version **ServiceMeshControlPlane** détermine la version de Red Hat OpenShift Service Mesh que vous utilisez. La valeur du champ **spec.version** dans la ressource **ServiceMeshControlPlane** contrôle l'architecture et les paramètres de configuration utilisés pour installer et déployer Red Hat OpenShift Service Mesh. Lorsque vous créez le plan de contrôle Service Mesh, vous pouvez définir la version de deux manières :

    - Pour configurer la vue formulaire, sélectionnez la version dans le menu **Control Plane Version**.

    - Pour configurer la vue YAML, définissez la valeur de **spec.version** dans le fichier YAML.

Operator Lifecycle Manager (OLM) ne gère pas les mises à niveau du plan de contrôle de Service Mesh, de sorte que le numéro de version de votre opérateur et de **ServiceMeshControlPlane** (SMCP) peut ne pas correspondre, à moins que vous n'ayez mis à niveau manuellement votre SMCP.

1.22.2. Dépannage Installation de l'opérateur

En plus des informations contenues dans cette section, veillez à consulter les rubriques suivantes :

- Qu'est-ce qu'un opérateur ?
- Concepts de gestion du cycle de vie des opérateurs .
- Section de dépannage d'OpenShift Operator .
- Section de dépannage de l'installation d'OpenShift .

1.22.2.1. Validation de l'installation de l'opérateur

Lorsque vous installez les opérateurs Red Hat OpenShift Service Mesh, OpenShift crée automatiquement les objets suivants dans le cadre d'une installation réussie de l'opérateur :

- cartes de configuration
- définitions de ressources personnalisées
- déploiements
- gousses
- ensembles de répliques
- rôles
- liaisons de rôles
- secrets
- comptes de service
- services

Depuis la console OpenShift Container Platform

Vous pouvez vérifier que les pods de l'opérateur sont disponibles et fonctionnent en utilisant la console OpenShift Container Platform.

1. Navigate to **Workloads → Pods**.
2. Sélectionnez l'espace de noms **openshift-operators**.
3. Vérifiez que les pods suivants existent et ont un statut de **running**:
   - **istio-operator**
   - **jaeger-operator**
   - **kiali-operator**
4. Sélectionnez l'espace de noms **openshift-operators-redhat**.
5. Vérifiez que le module **elasticsearch-operator** existe et qu'il a le statut **running**.

A partir de la ligne de commande

1. Vérifiez que les pods Operator sont disponibles et en cours d'exécution dans l'espace de noms **openshift-operators** à l'aide de la commande suivante :

```
$ oc get pods -n openshift-operators
```

Exemple de sortie

```
NAME                                READY   STATUS    RESTARTS   AGE
istio-operator-bb49787db-zgr87      1/1     Running   0          15s
jaeger-operator-7d5c4f57d8-9xphf    1/1     Running   0          2m42s
kiali-operator-f9c8d84f4-7xh2v      1/1     Running   0          64s
```

2. Vérifiez l'opérateur Elasticsearch à l'aide de la commande suivante :

```
$ oc get pods -n openshift-operators-redhat
```

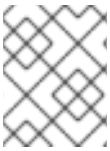
Exemple de sortie

```
NAME                                     READY   STATUS    RESTARTS   AGE
elasticsearch-operator-d4f59b968-796vq   1/1     Running   0          15s
```

1.22.2.2. Dépannage des mailles du filet Opérateurs

Si vous rencontrez des problèmes avec l'opérateur :

- Vérifiez l'état de votre abonnement à l'opérateur.
- Vérifiez que vous n'avez pas installé une version communautaire de l'opérateur au lieu de la version Red Hat prise en charge.
- Vérifiez que vous avez le rôle **cluster-admin** pour installer Red Hat OpenShift Service Mesh.
- Si le problème est lié à l'installation des opérateurs, vérifiez s'il n'y a pas d'erreurs dans les journaux des pods des opérateurs.



NOTE

Vous ne pouvez installer les opérateurs qu'à travers la console OpenShift, l'OperatorHub n'est pas accessible depuis la ligne de commande.

1.22.2.2.1. Visualisation des journaux de pods de l'opérateur

Vous pouvez afficher les journaux de l'opérateur en utilisant la commande **oc logs**. Red Hat peut demander des journaux pour aider à résoudre des cas d'assistance.

Procédure

- Pour afficher les journaux du pod de l'opérateur, entrez la commande :

```
$ oc logs -n openshift-operators <podName>
```

Par exemple,

```
$ oc logs -n openshift-operators istio-operator-bb49787db-zgr87
```

1.22.3. Dépannage du plan de contrôle

Le Service Mesh *control plane* est composé d'Istiod, qui consolide plusieurs composants de plan de contrôle précédents (Citadel, Galley, Pilot) en un seul binaire. Le déploiement de **ServiceMeshControlPlane** crée également les autres composants qui constituent Red Hat OpenShift Service Mesh, comme décrit dans la rubrique sur l'architecture.

1.22.3.1. Validation de l'installation du plan de contrôle Service Mesh

Lorsque vous créez le plan de contrôle Service Mesh, l'opérateur Service Mesh utilise les paramètres que vous avez spécifiés dans le fichier de ressources **ServiceMeshControlPlane** pour effectuer les opérations suivantes :

- Crée les composants Istio et déploie les pods suivants :
  - **istiod**
  - **istio-ingressgateway**
  - **istio-egressgateway**
  - **grafana**
  - **prometheus**
- Appelle l'opérateur Kiali pour créer un déploiement Kaili basé sur la configuration du SMCP ou de la ressource personnalisée Kiali.

NOTE

Les composants de Kiali sont affichés par l'opérateur Kiali, et non par l'opérateur Service Mesh.

- Appelle l'opérateur de plateforme de traçage distribuée Red Hat OpenShift pour créer des composants de plateforme de traçage distribuée basés sur la configuration dans la ressource personnalisée SMCP ou Jaeger.

NOTE

Vous visualisez les composants Jaeger sous l'Opérateur de plateforme de traçage distribuée Red Hat OpenShift et les composants Elasticsearch sous l'Opérateur Red Hat Elasticsearch, et non sous l'Opérateur Service Mesh.

Depuis la console OpenShift Container Platform

Vous pouvez vérifier l'installation du plan de contrôle Service Mesh dans la console web OpenShift Container Platform.

1. Naviguez jusqu'à **Operators → Installed Operators**.
2. Sélectionnez l'espace de noms **<istio-system>**.

3. Sélectionnez l'opérateur Red Hat OpenShift Service Mesh.
    a. Cliquez sur l'onglet **Istio Service Mesh Control Plane**
    b. Cliquez sur le nom de votre plan de contrôle, par exemple **basic**.
    c. Pour afficher les ressources créées par le déploiement, cliquez sur l'onglet **Resources**. Vous pouvez utiliser le filtre pour restreindre votre vue, par exemple, pour vérifier que tous les **Pods** ont un statut de **running**.
    d. Si l'état du SMCP indique un problème, consultez la sortie **status:** dans le fichier YAML pour plus d'informations.
    e. Retournez à **Operators → Installed Operators**.
4. Sélectionnez l'opérateur OpenShift Elasticsearch.
    a. Cliquez sur l'onglet **Elasticsearch**.
    b. Cliquez sur le nom du déploiement, par exemple **elasticsearch**.
    c. Pour afficher les ressources créées par le déploiement, cliquez sur l'onglet **Resources**. .
    d. Si la colonne **Status** pose des problèmes, consultez la sortie **status:** sur l'onglet **YAML** pour plus d'informations.
    e. Retournez à **Operators → Installed Operators**.
5. Sélectionnez l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift.
    a. Cliquez sur l'onglet **Jaeger**.
    b. Cliquez sur le nom de votre déploiement, par exemple **jaeger**.
    c. Pour afficher les ressources créées par le déploiement, cliquez sur l'onglet **Resources**.
    d. Si la colonne **Status** indique des problèmes, vérifiez la sortie **status:** sur l'onglet **YAML** pour plus d'informations.
    e. Naviguez jusqu'à **Operators → Installed Operators**.
6. Sélectionnez l'opérateur Kiali.
    a. Cliquez sur l'onglet **Istio Service Mesh Control Plane**
    b. Cliquez sur le nom de votre déploiement, par exemple **kiali**.
    c. Pour afficher les ressources créées par le déploiement, cliquez sur l'onglet **Resources**.
    d. Si la colonne **Status** pose des problèmes, consultez la sortie **status:** sur l'onglet **YAML** pour plus d'informations.

A partir de la ligne de commande

1. Exécutez la commande suivante pour vérifier si les pods du plan de contrôle Service Mesh sont disponibles et en cours d'exécution, où **istio-system** est l'espace de noms dans lequel vous avez installé le SMCP.

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

Exemple de sortie

```
NAME                                READY   STATUS    RESTARTS   AGE
grafana-6776785cfc-6fz7t            2/2     Running   0          102s
istio-egressgateway-5f49dd99-l9ppq  1/1     Running   0          103s
istio-ingressgateway-6dc885c48-jjd8r 1/1    Running   0          103s
istiod-basic-6c9cc55998-wg4zq       1/1     Running   0          2m14s
jaeger-6865d5d8bf-zrfss             2/2     Running   0          100s
kiali-579799fbb7-8mwc8              1/1     Running   0          46s
prometheus-5c579dfb-6qhjk           2/2     Running   0          115s
```

2. Vérifiez l'état du déploiement du plan de contrôle Service Mesh à l'aide de la commande suivante. Remplacez **istio-system** par l'espace de noms dans lequel vous avez déployé le SMCP.

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

L'installation est terminée avec succès lorsque la colonne STATUS est **ComponentsReady**.

Exemple de sortie

```
NAME    READY   STATUS            PROFILES      VERSION   AGE
basic   10/10   ComponentsReady   ["default"]   2.1.3     4m2s
```

Si vous avez modifié et redéployé votre plan de contrôle Service Mesh, le statut devrait être **UpdateSuccessful**.

Exemple de sortie

```
NAME          READY     STATUS            TEMPLATE   VERSION   AGE
basic-install   10/10     UpdateSuccessful   default     v1.1     3d16h
```

3. Si l'état du SMCP indique autre chose que **ComponentsReady**, vérifiez la sortie **status:** dans la ressource SCMP pour plus d'informations.

```
$ oc describe smcp <smcp-name> -n <controlplane-namespace>
```

Exemple de sortie

```
$ oc describe smcp basic -n istio-system
```

4. Vérifiez l'état du déploiement de Jaeger à l'aide de la commande suivante, où **istio-system** est l'espace de noms dans lequel vous avez déployé le SMCP.

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

Exemple de sortie

```
NAME    STATUS    VERSION   STRATEGY   STORAGE   AGE
jaeger   Running   1.30.0    allinone   memory    15m
```

5. Vérifiez l'état du déploiement de Kiali à l'aide de la commande suivante, où **istio-system** est l'espace de noms dans lequel vous avez déployé le SMCP.

```
$ oc get kiali -n <istio-system>
```

Exemple de sortie

```
NAME    AGE
kiali   15m
```

1.22.3.1.1. Accès à la console Kiali

Vous pouvez visualiser la topologie, la santé et les métriques de votre application dans la console Kiali. Si votre service rencontre des problèmes, la console Kiali vous permet de visualiser le flux de données à travers votre service. Vous pouvez obtenir des informations sur les composants du maillage à différents niveaux, y compris les applications abstraites, les services et les charges de travail. Kiali fournit également une vue graphique interactive de votre espace de noms en temps réel.

Pour accéder à la console Kiali, vous devez avoir Red Hat OpenShift Service Mesh installé, Kiali installé et configuré.

Le processus d'installation crée une route pour accéder à la console Kiali.

Si vous connaissez l'URL de la console Kiali, vous pouvez y accéder directement. Si vous ne connaissez pas l'URL, utilisez les instructions suivantes.

Procédure pour les administrateurs

1. Connectez-vous à la console web de OpenShift Container Platform avec un rôle d'administrateur.
2. Cliquez sur **Home → Projects**.
3. Sur la page **Projects**, si nécessaire, utilisez le filtre pour trouver le nom de votre projet.
4. Cliquez sur le nom de votre projet, par exemple **info**.
5. Sur la page **Project details**, dans la section **Launcher**, cliquez sur le lien **Kiali**.
6. Connectez-vous à la console Kiali avec le même nom d'utilisateur et le même mot de passe que ceux utilisés pour accéder à la console OpenShift Container Platform.
   Lorsque vous vous connectez pour la première fois à la console Kiali, vous voyez la page **Overview** qui affiche tous les espaces de noms de votre maillage de services que vous avez le droit de voir.

   Si vous validez l'installation de la console et que les espaces de noms n'ont pas encore été ajoutés au maillage, il se peut qu'il n'y ait pas d'autres données à afficher que **istio-system**.

Procédure pour les développeurs

1. Connectez-vous à la console web de OpenShift Container Platform avec un rôle de développeur.
2. Cliquez sur **Project**.

3. Sur la page **Project Details**, si nécessaire, utilisez le filtre pour trouver le nom de votre projet.
4. Cliquez sur le nom de votre projet, par exemple **info**.
5. Sur la page **Project**, dans la section **Launcher**, cliquez sur le lien **Kiali**.
6. Cliquez sur **Log In With OpenShift**

1.22.3.1.2. Accéder à la console Jaeger

Pour accéder à la console Jaeger, vous devez avoir installé Red Hat OpenShift Service Mesh, Red Hat OpenShift distributed tracing platform installé et configuré.

Le processus d'installation crée une route pour accéder à la console Jaeger.

Si vous connaissez l'URL de la console Jaeger, vous pouvez y accéder directement. Si vous ne connaissez pas l'URL, suivez les instructions suivantes.

Procédure à partir de la console OpenShift

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur disposant des droits cluster-admin. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.
2. Naviguez jusqu'à **Networking → Routes**.
3. Sur la page **Routes**, sélectionnez le projet de plan de contrôle Service Mesh, par exemple **istio-system**, dans le menu **Namespace**.
   La colonne **Location** affiche l'adresse liée à chaque itinéraire.
4. Si nécessaire, utilisez le filtre pour trouver la route **jaeger**. Cliquez sur la route **Location** pour lancer la console.
5. Cliquez sur **Log In With OpenShift**

Procédure à partir de la console Kiali

1. Lancer la console Kiali.
2. Cliquez sur **Distributed Tracing** dans le volet de navigation gauche.
3. Cliquez sur **Log In With OpenShift**

Procédure à partir du CLI

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

   ```
   $ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
   ```

2. Pour demander des détails sur l'itinéraire à l'aide de la ligne de commande, entrez la commande suivante. Dans cet exemple, **istio-system** est l'espace de noms du plan de contrôle Service Mesh.

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

3. Lancez un navigateur et accédez à **https://<JAEGER_URL>**, où **<JAEGER_URL>** est l'itinéraire que vous avez découvert à l'étape précédente.

4. Connectez-vous en utilisant le même nom d'utilisateur et le même mot de passe que ceux utilisés pour accéder à la console OpenShift Container Platform.

5. Si vous avez ajouté des services au maillage de services et généré des traces, vous pouvez utiliser les filtres et le bouton **Find Traces** pour rechercher vos données de traces.
Si vous validez l'installation de la console, il n'y a pas de données de trace à afficher.

1.22.3.2. Dépannage du plan de contrôle du Service Mesh

Si vous rencontrez des problèmes lors du déploiement du plan de contrôle Service Mesh,

- Assurez-vous que la ressource **ServiceMeshControlPlane** est installée dans un projet distinct de vos services et opérateurs. Cette documentation utilise le projet **istio-system** comme exemple, mais vous pouvez déployer votre plan de contrôle dans n'importe quel projet tant qu'il est séparé du projet qui contient vos opérateurs et vos services.

- Assurez-vous que les ressources personnalisées **ServiceMeshControlPlane** et **Jaeger** sont déployées dans le même projet. Par exemple, utilisez le projet **istio-system** pour les deux.

1.22.4. Dépannage du plan de données

Le site *data plane* est un ensemble de serveurs mandataires intelligents qui interceptent et contrôlent toutes les communications réseau entrantes et sortantes entre les services du maillage de services.

Red Hat OpenShift Service Mesh s'appuie sur un sidecar proxy dans le pod de l'application pour fournir des capacités de maillage de services à l'application.

1.22.4.1. Dépannage de l'injection du side-car

Red Hat OpenShift Service Mesh n'injecte pas automatiquement les sidecars de proxy aux pods. Vous devez opter pour l'injection de sidecars.

1.22.4.1.1. Dépannage de l'injection du sidecar d'Istio

Vérifiez si l'injection automatique est activée dans le déploiement de votre application. Si l'injection automatique pour le proxy Envoy est activée, il devrait y avoir une annotation **sidecar.istio.io/inject:"true"** dans la ressource **Deployment** sous **spec.template.metadata.annotations**.

1.22.4.1.2. Dépannage de l'injection du side-car de l'agent Jaeger

Vérifiez si l'injection automatique est activée dans le déploiement de votre application. Si l'injection automatique pour l'agent Jaeger est activée, il devrait y avoir une annotation **sidecar.jaegertracing.io/inject:"true"** dans la ressource **Deployment**.

Pour plus d'informations sur l'injection de sidecar, voir Activation de l'injection automatique

1.23. DÉPANNAGE DU PROXY ENVOY

Le proxy Envoy intercepte tout le trafic entrant et sortant pour tous les services du maillage de services. Envoy collecte également des données télémétriques sur le maillage de services et en rend compte. Envoy est déployé en tant que sidecar du service concerné dans le même pod.

1.23.1. Activation des journaux d'accès à Envoy

Les journaux d'accès Envoy sont utiles pour diagnostiquer les défaillances et les flux de trafic, et aident à l'analyse des flux de trafic de bout en bout.

Pour activer la journalisation des accès pour tous les conteneurs istio-proxy, modifiez l'objet **ServiceMeshControlPlane** (SMCP) afin d'ajouter un nom de fichier pour la sortie de la journalisation.

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle de cluster-admin. Entrez la commande suivante. Saisissez ensuite votre nom d'utilisateur et votre mot de passe lorsque vous y êtes invité.

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. Accédez au projet dans lequel vous avez installé le plan de contrôle Service Mesh, par exemple **istio-system**.

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

3. Modifiez le fichier **ServiceMeshControlPlane**.

```
$ oc edit smcp <nom_smcp>
```

4. Comme le montre l'exemple suivant, utilisez **name** pour spécifier le nom de fichier du journal du proxy. Si vous ne spécifiez pas de valeur pour **name**, aucune entrée de journal ne sera écrite.

```
spec:
  proxy:
    accessLogging:
      file:
        name: /dev/stdout     #file name
```

Pour plus d'informations sur la résolution des problèmes liés aux pods, voir Investigation des problèmes liés aux pods

1.23.2. Obtenir de l'aide

Si vous rencontrez des difficultés avec une procédure décrite dans cette documentation, ou avec OpenShift Container Platform en général, visitez le portail client de Red Hat . À partir du portail client, vous pouvez :

- Recherchez ou parcourez la base de connaissances de Red Hat qui contient des articles et des solutions relatifs aux produits Red Hat.
- Soumettre un cas d'assistance à Red Hat Support.
- Accéder à d'autres documents sur les produits.

Pour identifier les problèmes liés à votre cluster, vous pouvez utiliser Insights dans OpenShift Cluster Manager Hybrid Cloud Console. Insights fournit des détails sur les problèmes et, le cas échéant, des informations sur la manière de les résoudre.

Si vous avez une suggestion pour améliorer cette documentation ou si vous avez trouvé une erreur, soumettez un problème Jira pour le composant de documentation le plus pertinent. Veuillez fournir des détails spécifiques, tels que le nom de la section et la version d'OpenShift Container Platform.

1.23.2.1. À propos de la base de connaissances de Red Hat

La base de connaissances de Red Hat fournit un contenu riche destiné à vous aider à tirer le meilleur parti des produits et des technologies de Red Hat. La base de connaissances de Red Hat comprend des articles, de la documentation sur les produits et des vidéos décrivant les meilleures pratiques en matière d'installation, de configuration et d'utilisation des produits Red Hat. En outre, vous pouvez rechercher des solutions à des problèmes connus, chacune d'entre elles fournissant des descriptions concises de la cause première et des mesures correctives.

1.23.2.2. Recherche dans la base de connaissances de Red Hat

En cas de problème lié à OpenShift Container Platform, vous pouvez effectuer une recherche initiale pour déterminer si une solution existe déjà dans la base de connaissances de Red Hat.

Conditions préalables

- Vous disposez d'un compte Red Hat Customer Portal.

Procédure

1. Connectez-vous au portail client de Red Hat .
2. Dans le champ de recherche principal du portail client de Red Hat, saisissez des mots-clés et des chaînes de caractères relatifs au problème, y compris :
   - Composants de la plateforme OpenShift Container (tels que **etcd**)
   - Procédure connexe (telle que **installation**)
   - Avertissements, messages d'erreur et autres résultats liés à des défaillances explicites
3. Cliquez sur **Search**.
4. Sélectionnez le filtre de produit **OpenShift Container Platform**.
5. Sélectionnez le filtre de type de contenu **Knowledgebase**.

1.23.2.3. À propos de l'outil de collecte obligatoire

La commande CLI **oc adm must-gather** recueille les informations de votre cluster les plus susceptibles d'être nécessaires au débogage des problèmes, notamment

- Définitions des ressources
- Journaux de service

Par défaut, la commande **oc adm must-gather** utilise l'image du plugin par défaut et écrit dans **./must-gather.local**.

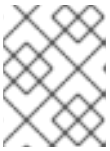
Vous pouvez également recueillir des informations spécifiques en exécutant la commande avec les arguments appropriés, comme décrit dans les sections suivantes :

- Pour collecter des données relatives à une ou plusieurs caractéristiques spécifiques, utilisez l'argument **--image** avec une image, comme indiqué dans la section suivante.
  Par exemple :

```
$ oc adm must-gather  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.12.0
```

- Pour collecter les journaux d'audit, utilisez l'argument **-- /usr/bin/gather_audit_logs**, comme décrit dans la section suivante.
  Par exemple :

```
$ oc adm must-gather -- /usr/bin/gather_audit_logs
```



NOTE

Les journaux d'audit ne sont pas collectés dans le cadre de l'ensemble d'informations par défaut afin de réduire la taille des fichiers.

Lorsque vous exécutez **oc adm must-gather**, un nouveau module portant un nom aléatoire est créé dans un nouveau projet sur le cluster. Les données sont collectées sur ce module et enregistrées dans un nouveau répertoire commençant par **must-gather.local**. Ce répertoire est créé dans le répertoire de travail actuel.

Par exemple :

```
NAMESPACE                  NAME              READY   STATUS      RESTARTS      AGE
...
openshift-must-gather-5drcj    must-gather-bklx4    2/2     Running     0          72s
openshift-must-gather-5drcj    must-gather-s8sdh    2/2     Running     0          72s
...
```

1.23.2.4. A propos de la collecte de données sur le maillage des services

Vous pouvez utiliser la commande CLI **oc adm must-gather** pour collecter des informations sur votre cluster, y compris les fonctionnalités et les objets associés à Red Hat OpenShift Service Mesh.

Conditions préalables

- Accès au cluster en tant qu'utilisateur ayant le rôle **cluster-admin**.
- L'OpenShift Container Platform CLI (**oc**) est installé.

Précédent

1. Pour collecter les données Red Hat OpenShift Service Mesh avec **must-gather**, vous devez spécifier l'image Red Hat OpenShift Service Mesh.

```
$ oc adm must-gather --image=registry.redhat.io/openshift-service-mesh/istio-must-gather-rhel8:2.3
```

2. Pour collecter les données Red Hat OpenShift Service Mesh pour un espace de noms de plan de contrôle Service Mesh spécifique avec **must-gather**, vous devez spécifier l'image Red Hat OpenShift Service Mesh et l'espace de noms. Dans cet exemple, remplacez **<namespace>** par votre espace de noms de plan de contrôle Service Mesh, tel que **istio-system**.

```
$ oc adm must-gather --image=registry.redhat.io/openshift-service-mesh/istio-must-gather-rhel8:2.3 gather <namespace>
```

Pour une assistance rapide, fournissez des informations de diagnostic pour OpenShift Container Platform et Red Hat OpenShift Service Mesh.

1.23.2.5. Soumettre un dossier d'assistance

Conditions préalables

- Vous avez accès au cluster en tant qu'utilisateur ayant le rôle **cluster-admin**.
- Vous avez installé l'OpenShift CLI (**oc**).
- Vous disposez d'un compte Red Hat Customer Portal.
- Vous avez un abonnement standard ou premium Red Hat.

Procédure

1. Connectez-vous au portail client de Red Hat et sélectionnez **SUPPORT CASES → Open a case**.
2. Sélectionnez la catégorie appropriée pour votre problème (telle que **Defect / Bug**), le produit (**OpenShift Container Platform**) et la version du produit (**4.12**, si elle n'est pas déjà remplie automatiquement).
3. Examinez la liste des solutions suggérées dans la base de connaissances de Red Hat afin de trouver une correspondance potentielle avec le problème signalé. Si les articles suggérés ne répondent pas au problème, cliquez sur **Continue**.
4. Saisissez un résumé concis mais descriptif du problème et des détails supplémentaires sur les symptômes ressentis, ainsi que vos attentes.
5. Examinez la liste mise à jour des solutions suggérées dans la base de connaissances de Red Hat afin de trouver une correspondance potentielle avec le problème signalé. La liste est affinée au fur et à mesure que vous fournissez des informations supplémentaires au cours du processus de création du cas. Si les articles suggérés ne répondent pas au problème, cliquez sur **Continue**.
6. S'assurer que les informations sur le compte présentées sont conformes aux attentes et, si ce n'est pas le cas, les modifier en conséquence.
7. Vérifiez que l'identifiant de cluster OpenShift Container Platform rempli automatiquement est correct. Si ce n'est pas le cas, obtenez manuellement votre ID de cluster.
   - Pour obtenir manuellement votre ID de cluster à l'aide de la console web d'OpenShift Container Platform :
     a. Naviguez vers **Home → Dashboards → Overview**.
     b. Trouvez la valeur dans le champ **Cluster ID** de la section **Details**.

- Il est également possible d'ouvrir un nouveau dossier de support via la console web d'OpenShift Container Platform et de faire en sorte que l'identifiant de votre cluster soit rempli automatiquement.

  a. Dans la barre d'outils, naviguez vers **(?) Help → Open Support Case**.

  b. La valeur **Cluster ID** est remplie automatiquement.

- Pour obtenir l'ID de votre cluster à l'aide de l'OpenShift CLI (**oc**), exécutez la commande suivante :

```
$ oc get clusterversion -o jsonpath='{.items[].spec.clusterID}{"\n"}'
```

8. Répondez aux questions suivantes lorsque vous y êtes invité, puis cliquez sur **Continue**:

   - Où expérimentez-vous ce comportement ? Dans quel environnement ?
   - Quand le comportement se produit-il ? A quelle fréquence ? De manière répétée ? A certains moments ?
   - Quelles informations pouvez-vous fournir sur les délais et l'impact commercial ?

9. Téléchargez les fichiers de données de diagnostic pertinents et cliquez sur **Continue**. Il est recommandé d'inclure les données recueillies à l'aide de la commande **oc adm must-gather** comme point de départ, ainsi que toutes les données spécifiques au problème qui ne sont pas recueillies par cette commande.

10. Saisissez les informations relatives à la gestion du dossier et cliquez sur **Continue**.

11. Prévisualisez les détails du cas et cliquez sur **Submit**.

1.24. RÉFÉRENCE DE CONFIGURATION DU PLAN DE CONTRÔLE SERVICE MESH

Vous pouvez personnaliser votre Red Hat OpenShift Service Mesh en modifiant la ressource par défaut **ServiceMeshControlPlane** (SMCP) ou en créant une ressource SMCP entièrement personnalisée. Cette section de référence documente les options de configuration disponibles pour la ressource SMCP.

1.24.1. Service Mesh Paramètres du plan de contrôle

Le tableau suivant répertorie les paramètres de premier niveau de la ressource **ServiceMeshControlPlane**.

Tableau 1.31. ServiceMeshControlPlane paramètres des ressources

Nom	Description	Type

Nom	Description	Type
apiVersion	APIVersion définit le schéma versionné de cette représentation d'un objet. Les serveurs convertissent les schémas reconnus à la dernière valeur interne et peuvent rejeter les valeurs non reconnues. La valeur pour la version 2.0 de **ServiceMeshControlPlane** est **maistra.io/v2**.	La valeur pour **ServiceMeshControlPlane** version 2.0 est **maistra.io/v2**.
kind	Kind est une chaîne de caractères qui représente la ressource REST que cet objet représente.	**ServiceMeshControlPlane** est la seule valeur valable pour un ServiceMeshControlPlane.
metadata	Métadonnées sur cette instance **ServiceMeshControlPlane**. Vous pouvez donner un nom à votre installation du plan de contrôle Service Mesh pour garder une trace de votre travail, par exemple **basic**.	chaîne de caractères
spec	La spécification de l'état souhaité de ce site **ServiceMeshControlPlane**. Cela inclut les options de configuration pour tous les composants qui constituent le plan de contrôle du Service Mesh.	Pour plus d'informations, voir le tableau 2.
status	L'état actuel de ce site **ServiceMeshControlPlane** et des composants qui constituent le plan de contrôle du Service Mesh.	Pour plus d'informations, voir le tableau 3.

Le tableau suivant répertorie les spécifications de la ressource **ServiceMeshControlPlane**. La modification de ces paramètres configure les composants de Red Hat OpenShift Service Mesh.

Tableau 1.32. **ServiceMeshControlPlane** spécification des ressources

Nom	Description	Paramètres configurables

Nom	Description	Paramètres configurables
addons	Le paramètre **addons** permet de configurer des fonctions supplémentaires au-delà des composants de base du plan de contrôle de Service Mesh, telles que la visualisation ou le stockage de métriques.	**3scale**, **grafana**, **jaeger**, **kiali**, et **prometheus**.
cluster	Le paramètre **cluster** définit la configuration générale du cluster (nom du cluster, nom du réseau, multi-cluster, expansion du maillage, etc.)	**meshExpansion**, **multiCluster**, **name**, et **network**
gateways	Le paramètre **gateways** permet de configurer les passerelles d'entrée et de sortie du réseau maillé.	**enabled**, **additionalEgress**, **additionalIngress**, **egress**, **ingress**, et **openshiftRoute**
general	Le paramètre **general** représente la configuration générale du plan de contrôle du Service Mesh qui n'a pas sa place ailleurs.	**logging** et **validationMessages**
policy	Le paramètre **policy** permet de configurer le contrôle des règles pour le plan de contrôle Service Mesh. Le contrôle des stratégies peut être activé en réglant **spec.policy.enabled** sur **true**.	**mixer remote** ou **type**. **type** peut être défini comme **Istiod**, **Mixer** ou **None**.
profiles	Le paramètre **profiles** permet de sélectionner le profil **ServiceMeshControlPlane** à utiliser pour les valeurs par défaut.	**default**
proxy	Le paramètre **proxy** permet de configurer le comportement par défaut des sidecars.	**accessLogging**, **adminPort**, **concurrency**, et **envoyMetricsService**
runtime	Le paramètre **runtime** permet de configurer les composants du plan de contrôle du Service Mesh.	**components** et **defaults**

Nom	Description	Paramètres configurables
security	Le paramètre **security** permet de configurer les aspects de la sécurité pour le plan de contrôle du Service Mesh.	**certificateAuthority**, **controlPlane**, **identity**, **dataPlane** et **trust**
techPreview	Le paramètre **techPreview** permet un accès anticipé aux fonctionnalités qui sont en avant-première technologique.	N/A
telemetry	Si **spec.mixer.telemetry.enabled** est réglé sur **true**, la télémétrie est activée.	**mixer remote type type** peut être réglé sur , ou . **Istiod Mixer None**
tracing	Le paramètre **tracing** permet d'activer le traçage distribué pour le maillage.	**sampling type**. peut être défini comme ou . **type Jaeger None**
version	Vous utilisez le paramètre **version** pour spécifier la version Maistra du plan de contrôle Service Mesh à installer. Lors de la création d'un **ServiceMeshControlPlane** avec une version vide, le webhook d'admission fixe la version à la version actuelle. Les nouveaux **ServiceMeshControlPlanes** dont la version est vide sont définis sur **v2.0**. Les **ServiceMeshControlPlanes** existants dont la version est vide conservent leur paramètre.	chaîne de caractères

ControlPlaneStatus représente l'état actuel de votre maillage de services.

Tableau 1.33. **ServiceMeshControlPlane** ressource **ControlPlaneStatus**

Nom	Description	Type

Nom	Description	Type
annotations	Le paramètre **annotations** stocke des informations d'état supplémentaires, généralement redondantes, telles que le nombre de composants déployés par l'outil **ServiceMeshControlPlane**. Ces états sont utilisés par l'outil de ligne de commande **oc**, qui ne permet pas encore de compter les objets dans les expressions JSONPath.	Non configurable
conditions	Représente les dernières observations disponibles sur l'état actuel de l'objet. **Reconciled** indique si l'opérateur a fini de réconcilier l'état réel des composants déployés avec la configuration de la ressource **ServiceMeshControlPlane**. **Installed** indique si le plan de contrôle Service Mesh a été installé. **Ready** indique si tous les composants du plan de contrôle Service Mesh sont prêts.	chaîne de caractères
components	Indique l'état de chaque composant du plan de contrôle Service Mesh déployé.	chaîne de caractères
appliedSpec	La spécification résultante des options de configuration après l'application de tous les profils.	**ControlPlaneSpec**
appliedValues	Le fichier values.yaml utilisé pour générer les graphiques.	**ControlPlaneSpec**
chartVersion	La version des graphiques qui a été traitée pour la dernière fois pour cette ressource.	chaîne de caractères

Nom	Description	Type
observedGeneration	Génération observée par le contrôleur lors du dernier rapprochement. Les informations contenues dans le statut se rapportent à cette génération particulière de l'objet. Le site **status.conditions** n'est pas à jour si le champ **status.observedGeneration** ne correspond pas à **metadata.generation**.	entier
operatorVersion	La version de l'opérateur qui a traité cette ressource en dernier.	chaîne de caractères
readiness	L'état de préparation des ressources appartenant aux composantes &.	chaîne de caractères

Cet exemple de définition de **ServiceMeshControlPlane** contient tous les paramètres pris en charge.

Exemple de ressource ServiceMeshControlPlane

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.3
  proxy:
    runtime:
      container:
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 500m
            memory: 128Mi
  tracing:
    type: Jaeger
  gateways:
    ingress: # istio-ingressgateway
      service:
        type: ClusterIP
        ports:
        - name: status-port
          port: 15020
        - name: http2
          port: 80
          targetPort: 8080
```

```
      - name: https
        port: 443
        targetPort: 8443
    meshExpansionPorts: []
  egress: # istio-egressgateway
    service:
      type: ClusterIP
      ports:
      - name: status-port
        port: 15020
      - name: http2
        port: 80
        targetPort: 8080
      - name: https
        port: 443
        targetPort: 8443
  additionalIngress:
    some-other-ingress-gateway: {}
  additionalEgress:
    some-other-egress-gateway: {}

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

telemetry:
  type: Istiod # or Mixer
  mixer: # only applies if telemetry.type: Mixer, for v1 telemetry
    sessionAffinity: false
    batching:
      maxEntries: 100
      maxTime: 1s
    adapters:
      kubernetesenv: true
      stdio:
        enabled: true
        outputAsJSON: true
addons:
  grafana:
    enabled: true
    install:
      config:
        env: {}
        envSecrets: {}
      persistence:
        enabled: true
        storageClassName: ""
        accessMode: ReadWriteOnce
        capacity:
          requests:
            storage: 5Gi
      service:
        ingress:
          contextPath: /grafana
```

```
        tls:
          termination: reencrypt
  kiali:
    name: kiali
    enabled: true
    install: # install kiali CR if not present
      dashboard:
        viewOnly: false
        enableGrafana: true
        enableTracing: true
        enablePrometheus: true
    service:
      ingress:
        contextPath: /kiali
  jaeger:
    name: jaeger
    install:
      storage:
        type: Elasticsearch # or Memory
        memory:
          maxTraces: 100000
        elasticsearch:
          nodeCount: 3
          storage: {}
          redundancyPolicy: SingleRedundancy
          indexCleaner: {}
      ingress: {} # jaeger ingress configuration
runtime:
  components:
    pilot:
      deployment:
        replicas: 2
      pod:
        affinity: {}
      container:
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 500m
            memory: 128Mi
    grafana:
      deployment: {}
      pod: {}
    kiali:
      deployment: {}
      pod: {}
```

1.24.2. paramètres du spec

1.24.2.1. paramètres généraux

Voici un exemple qui illustre les paramètres **spec.general** pour l'objet **ServiceMeshControlPlane** et une description des paramètres disponibles avec les valeurs appropriées.

Exemple de paramètres généraux

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  general:
    logging:
      componentLevels: {}
        # misc: error
      logAsJSON: false
    validationMessages: true
```

Tableau 1.34. Paramètres généraux d'Istio

Paramètres	Description	Valeurs	Valeur par défaut
l'exploitation forestière :	Permet de configurer la journalisation pour les composants du plan de contrôle de Service Mesh.		N/A
logging: componentLevels:	Permet de spécifier le niveau de journalisation du composant.	Valeurs possibles : **trace**, **debug**, **info**, **warning**, **error**, **fatal**, **panic**.	N/A
logging: logAsJSON:	Permet d'activer ou de désactiver la journalisation JSON.	**true**/**false**	N/A
validationMessages :	Permet d'activer ou de désactiver les messages de validation des champs d'état des ressources istio.io. Cela peut être utile pour détecter les erreurs de configuration dans les ressources.	**true**/**false**	N/A

1.24.2.2. paramètres des profils

Vous pouvez créer des configurations réutilisables avec les profils d'objets **ServiceMeshControlPlane**. Si vous ne configurez pas le paramètre **profile**, Red Hat OpenShift Service Mesh utilise le profil par défaut.

Voici un exemple qui illustre le paramètre **spec.profiles** pour l'objet **ServiceMeshControlPlane**:

Exemple de paramètres de profils

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

Pour plus d'informations sur la création de profils, voir la section Création de profils de plan de contrôle .

Pour des exemples plus détaillés de configuration de la sécurité, voir Mutual Transport Layer Security (mTLS).

1.24.2.3. paramètres de techPreview

Le paramètre **spec.techPreview** permet un accès anticipé aux fonctionnalités qui font l'objet d'un aperçu technologique.

IMPORTANT

Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas de les utiliser en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, ce qui permet aux clients de tester les fonctionnalités et de fournir un retour d'information pendant le processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

1.24.2.4. paramètres de traçage

L'exemple suivant illustre les paramètres **spec.tracing** pour l'objet **ServiceMeshControlPlane**, ainsi qu'une description des paramètres disponibles avec les valeurs appropriées.

Exemple de paramètres de traçage

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.3
  tracing:
    sampling: 100
    type: Jaeger
```

Tableau 1.35. Paramètres de traçage d'Istio

Paramètres	Description	Valeurs	Valeur par défaut
`tracing: sampling:`	Le taux d'échantillonnage détermine la fréquence à laquelle le proxy Envoy génère une trace. Vous utilisez le taux d'échantillonnage pour contrôler le pourcentage de requêtes qui sont signalées à votre système de traçage.	Valeurs entières entre 0 et 10000 représentant des incréments de 0,01% (0 à 100%). Par exemple, la valeur **10** échantillonne 0,1 % des demandes, la valeur **100** échantillonne 1 % des demandes, la valeur **500** échantillonne 5 % des demandes et la valeur **10000** échantillonne 100 % des demandes.	**10000** (100% des traces)
`tracing: type:`	Actuellement, le seul type de traçage pris en charge est **Jaeger**. Jaeger est activé par défaut. Pour désactiver le traçage, définissez le paramètre **type** sur **None**.	**None**, **Jaeger**	**Jaeger**

1.24.2.5. paramètre de version

L'Opérateur Red Hat OpenShift Service Mesh prend en charge l'installation de différentes versions de **ServiceMeshControlPlane**. Vous utilisez le paramètre de version pour spécifier la version du plan de contrôle Service Mesh à installer. Si vous ne spécifiez pas de paramètre de version lors de la création de votre SMCP, l'Opérateur définit la valeur de la dernière version : (2.3). Les objets **ServiceMeshControlPlane** existants conservent leur paramètre de version lors des mises à niveau de l'opérateur.

1.24.2.6. configuration 3scale

Le tableau suivant explique les paramètres de l'adaptateur Istio 3scale dans la ressource **ServiceMeshControlPlane**.

Exemple de paramètres 3scale

```
spec:
  addons:
    3Scale:
      enabled: false
      PARAM_THREESCALE_LISTEN_ADDR: 3333
      PARAM_THREESCALE_LOG_LEVEL: info
      PARAM_THREESCALE_LOG_JSON: true
      PARAM_THREESCALE_LOG_GRPC: false
      PARAM_THREESCALE_REPORT_METRICS: true
      PARAM_THREESCALE_METRICS_PORT: 8080
```

```
PARAM_THREESCALE_CACHE_TTL_SECONDS: 300
PARAM_THREESCALE_CACHE_REFRESH_SECONDS: 180
PARAM_THREESCALE_CACHE_ENTRIES_MAX: 1000
PARAM_THREESCALE_CACHE_REFRESH_RETRIES: 1
PARAM_THREESCALE_ALLOW_INSECURE_CONN: false
PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS: 10
PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS: 60
PARAM_USE_CACHED_BACKEND: false
PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS: 15
PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED: true
```

Tableau 1.36. paramètres 3scale

Paramètres	Description	Valeurs	Valeur par défaut
enabled	Utilisation ou non de l'adaptateur 3scale	**true**/**false**	**false**
PARAM_THREESCALE_LISTEN_ADDR	Définit l'adresse d'écoute du serveur gRPC	Numéro de port valide	**3333**
PARAM_THREESCALE_LOG_LEVEL	Définit le niveau minimum de sortie du journal.	**debug**, **info**, **warn**, **error**, ou **none**	**info**
PARAM_THREESCALE_LOG_JSON	Contrôle si le journal est formaté en JSON	**true**/**false**	**true**
PARAM_THREESCALE_LOG_GRPC	Contrôle si le journal contient des informations gRPC	**true**/**false**	**true**
PARAM_THREESCALE_REPORT_METRICS	Contrôle si les métriques du système 3scale et du backend sont collectées et rapportées à Prometheus	**true**/**false**	**true**
PARAM_THREESCALE_METRICS_PORT	Définit le port à partir duquel le point d'extrémité 3scale **/metrics** peut être mis au rebut	Numéro de port valide	**8080**
PARAM_THREESCALE_CACHE_TTL_SECONDS	Délai, en secondes, à attendre avant de purger les éléments expirés de la mémoire cache	Période de temps en secondes	**300**

Paramètres	Description	Valeurs	Valeur par défaut
PARAM_THREESCALE_CACHE_REFRESH_SECONDS	Période de temps avant l'expiration au cours de laquelle les éléments du cache sont tentés d'être rafraîchis	Période de temps en secondes	**180**
PARAM_THREESCALE_CACHE_ENTRIES_MAX	Nombre maximal d'éléments pouvant être stockés dans le cache à tout moment. La valeur **0** permet de désactiver la mise en cache	Numéro valide	**1000**
PARAM_THREESCALE_CACHE_REFRESH_RETRIES	Le nombre de fois où les hôtes inaccessibles sont réessayés pendant une boucle de mise à jour du cache	Numéro valide	**1**
PARAM_THREESCALE_ALLOW_INSECURE_CONN	Permet d'ignorer la vérification des certificats lors de l'appel des API **3scale**. Il n'est pas recommandé d'activer cette option.	**true**/**false**	**false**
PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS	Définit le nombre de secondes à attendre avant de terminer les requêtes vers le système 3scale et le backend	Période de temps en secondes	**10**
PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS	Définit le nombre maximum de secondes (/-10% de gigue) qu'une connexion peut durer avant d'être fermée	Période de temps en secondes	60
PARAM_USE_CACHE_BACKEND	Si true, tentative de création d'un cache d'apisonator en mémoire pour les demandes d'autorisation	**true**/**false**	**false**

Paramètres	Description	Valeurs	Valeur par défaut
PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS	Si le cache du backend est activé, ceci définit l'intervalle en secondes pour vider le cache par rapport à 3scale	Période de temps en secondes	15
PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED	Lorsque le cache du backend ne peut pas récupérer les données d'autorisation, il convient de refuser (fermé) ou d'autoriser (ouvert) les demandes	**true**/**false**	**true**

1.24.3. paramètre d'état

Le paramètre **status** décrit l'état actuel de votre maillage de services. Cette information est générée par l'opérateur et est en lecture seule.

Tableau 1.37. Paramètres d'état d'Istio

Nom	Description	Type
observedGeneration	Génération observée par le contrôleur lors du dernier rapprochement. Les informations contenues dans le statut se rapportent à cette génération particulière de l'objet. Le site **status.conditions** n'est pas à jour si le champ **status.observedGeneration** ne correspond pas à **metadata.generation**.	entier
annotations	Le paramètre **annotations** stocke des informations d'état supplémentaires, généralement redondantes, telles que le nombre de composants déployés par l'objet **ServiceMeshControlPlane**. Ces statuts sont utilisés par l'outil de ligne de commande **oc**, qui ne permet pas encore de compter les objets dans les expressions JSONPath.	Non configurable

Nom	Description	Type
readiness	L'état de préparation des composants et des ressources détenues.	chaîne de caractères
operatorVersion	La version de l'opérateur qui a traité cette ressource en dernier.	chaîne de caractères
components	Indique l'état de chaque composant du plan de contrôle Service Mesh déployé.	chaîne de caractères
appliedSpec	La spécification résultante des options de configuration après l'application de tous les profils.	**ControlPlaneSpec**
conditions	Représente les dernières observations disponibles sur l'état actuel de l'objet. **Reconciled** indique que l'opérateur a fini de réconcilier l'état réel des composants déployés avec la configuration de la ressource **ServiceMeshControlPlane**. **Installed** indique que le plan de contrôle du Service Mesh a été installé. **Ready** indique que tous les composants du plan de contrôle du Service Mesh sont prêts.	chaîne de caractères
chartVersion	La version des graphiques qui a été traitée pour la dernière fois pour cette ressource.	chaîne de caractères
appliedValues	Le fichier **values.yaml** qui a été utilisé pour générer les graphiques.	**ControlPlaneSpec**

1.24.4. Ressources supplémentaires

- Pour plus d'informations sur la configuration des fonctionnalités de la ressource **ServiceMeshControlPlane**, consultez les liens suivants :
  - Sécurité
  - Gestion du trafic
  - Métriques et traces

1.25. RÉFÉRENCE DE CONFIGURATION DE KIALI

Lorsque le Service Mesh Operator crée le site **ServiceMeshControlPlane**, il traite également la ressource Kiali. L'opérateur Kiali utilise ensuite cet objet lors de la création d'instances Kiali.

1.25.1. Spécifier la configuration de Kiali dans le SMCP

Vous pouvez configurer Kiali dans la section **addons** de la ressource **ServiceMeshControlPlane**. Kiali est activé par défaut. Pour désactiver Kiali, définissez **spec.addons.kiali.enabled** sur **false**.

Vous pouvez spécifier votre configuration Kiali de deux façons :

- Spécifier la configuration Kiali dans la ressource **ServiceMeshControlPlane** sous **spec.addons.kiali.install**. Cette approche présente certaines limites, car la liste complète des configurations Kiali n'est pas disponible dans le SMCP.

- Configurez et déployez une instance Kiali et spécifiez le nom de la ressource Kiali comme valeur de **spec.addons.kiali.name** dans la ressource **ServiceMeshControlPlane**. Vous devez créer le CR dans le même espace de noms que le plan de contrôle Service Mesh, par exemple, **istio-system**. Si une ressource Kiali correspondant à la valeur de **name** existe, le plan de contrôle configurera cette ressource Kiali pour l'utiliser avec le plan de contrôle. Cette approche vous permet de personnaliser entièrement votre configuration Kiali dans la ressource Kiali. Notez qu'avec cette approche, divers champs de la ressource Kiali sont remplacés par l'opérateur Service Mesh, en particulier la liste **accessible_namespaces**, ainsi que les points de terminaison pour Grafana, Prometheus et le traçage.

Exemple de paramètres SMCP pour Kiali

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  addons:
    kiali:
      name: kiali
      enabled: true
      install:
        dashboard:
          viewOnly: false
          enableGrafana: true
          enableTracing: true
          enablePrometheus: true
        service:
          ingress:
            contextPath: /kiali
```

Tableau 1.38. ServiceMeshControlPlane Paramètres de Kiali

Paramètres	Description	Valeurs	Valeur par défaut
spec: addons: kiali: name:	Nom de la ressource personnalisée Kiali. S'il existe une CR Kiali correspondant à la valeur de **name**, l'opérateur du Service Mesh utilisera cette CR pour l'installation. S'il n'existe pas de CR Kiali, l'opérateur en créera une à l'aide de cette adresse **name** et des options de configuration spécifiées dans le SMCP.	chaîne de caractères	**kiali**
kiali: enabled:	Ce paramètre permet d'activer ou de désactiver Kiali. Kiali est activé par défaut.	**true**/**false**	**true**
kiali: install:	Installer une ressource Kiali si la ressource Kiali nommée n'est pas présente. La section **install** est ignorée si **addons.kiali.enabled** est remplacé par **false**.		
kiali: install: dashboard:	Paramètres de configuration des tableaux de bord livrés avec Kiali.		
kiali: install: dashboard: viewOnly:	Ce paramètre permet d'activer ou de désactiver le mode vue seule pour la console Kiali. Lorsque ce mode est activé, les utilisateurs ne peuvent pas utiliser la console Kiali pour apporter des modifications au Service Mesh.	**true**/**false**	**false**

Paramètres	Description	Valeurs	Valeur par défaut
kiali: install: dashboard: enableGrafana:	Point d'extrémité Grafana configuré sur la base de la configuration **spec.addons.grafana** .	**true**/**false**	**true**
kiali: install: dashboard: enablePrometheus:	Point de terminaison Prometheus configuré sur la base de la configuration **spec.addons.prometheus**.	**true**/**false**	**true**
kiali: install: dashboard: enableTracing:	Point final de traçage configuré sur la base de la configuration des ressources personnalisées de Jaeger.	**true**/**false**	**true**
kiali: install: service:	Paramètres de configuration du service Kubernetes associé à l'installation de Kiali.		
kiali: install: service: metadata:	Permet de spécifier des métadonnées supplémentaires à appliquer aux ressources.	N/A	N/A
kiali: install: service: metadata: annotations:	Permet de spécifier des annotations supplémentaires à appliquer au service du composant.	chaîne de caractères	N/A
kiali: install: service: metadata: labels:	Permet de spécifier des étiquettes supplémentaires à appliquer au service du composant.	chaîne de caractères	N/A

Paramètres	Description	Valeurs	Valeur par défaut
kiali: install: service: ingress:	À utiliser pour spécifier les détails de l'accès au service du composant via une route OpenShift.	N/A	N/A
kiali: install: service: ingress: metadata: annotations:	Permet de spécifier des annotations supplémentaires à appliquer à l'entrée du service du composant.	chaîne de caractères	N/A
kiali: install: service: ingress: metadata: labels:	Permet de spécifier des étiquettes supplémentaires à appliquer à l'entrée de service du composant.	chaîne de caractères	N/A
kiali: install: service: ingress: enabled:	Permet de personnaliser une route OpenShift pour le service associé à un composant.	**true**/**false**	**true**
kiali: install: service: ingress: contextPath:	Permet de spécifier le chemin d'accès au contexte du service.	chaîne de caractères	N/A
install: service: ingress: hosts:	Permet de spécifier un seul nom d'hôte par route OpenShift. Un nom d'hôte vide implique un nom d'hôte par défaut pour la route.	chaîne de caractères	N/A

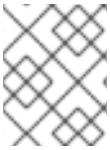
Paramètres	Description	Valeurs	Valeur par défaut

`install:`
`  service:`
`  ingress:`
`  tls:`
	Permet de configurer le TLS pour la route OpenShift.		N/A

`kiali:`
`  install:`
`  service:`
`  nodePort:`
	Permet de spécifier le site **nodePort** pour le service du composant **Values.<component>.service.nodePort.port**	entier	N/A

1.25.2. Spécifier la configuration de Kiali dans une ressource personnalisée Kiali

Vous pouvez personnaliser entièrement votre déploiement Kiali en configurant Kiali dans la ressource personnalisée Kiali (CR) plutôt que dans la ressource **ServiceMeshControlPlane** (SMCP). Cette configuration est parfois appelée "Kiali externe" car la configuration est spécifiée en dehors du SMCP.



NOTE

Vous devez déployer les ressources personnalisées **ServiceMeshControlPlane** et Kiali dans le même espace de noms. Par exemple, **istio-system**.

Vous pouvez configurer et déployer une instance Kiali, puis spécifier le site **name** de la ressource Kiali comme valeur de **spec.addons.kiali.name** dans la ressource SMCP. S'il existe un CR Kiali correspondant à la valeur de **name**, le plan de contrôle Service Mesh utilisera l'installation existante. Cette approche vous permet de personnaliser entièrement votre configuration Kiali.

1.26. RÉFÉRENCE DE LA CONFIGURATION JAEGER

Lorsque l'opérateur Service Mesh déploie la ressource **ServiceMeshControlPlane**, il peut également créer des ressources pour le traçage distribué. Service Mesh utilise Jaeger pour le traçage distribué.

1.26.1. Activation et désactivation du traçage

Vous activez le traçage distribué en spécifiant un type de traçage et un taux d'échantillonnage dans la ressource **ServiceMeshControlPlane**.

Paramètres par défaut de all-in-one Jaeger

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.3
```

```
tracing:
  sampling: 100
  type: Jaeger
```

Actuellement, le seul type de traçage pris en charge est **Jaeger**.

Jaeger est activé par défaut. Pour désactiver le traçage, définissez **type** sur **None**.

Le taux d'échantillonnage détermine la fréquence à laquelle le proxy Envoy génère une trace. Vous pouvez utiliser l'option de taux d'échantillonnage pour contrôler le pourcentage de requêtes qui sont rapportées à votre système de traçage. Vous pouvez configurer ce paramètre en fonction de votre trafic dans le maillage et de la quantité de données de traçage que vous souhaitez collecter. Vous configurez **sampling** sous la forme d'un nombre entier échelonné représentant des incréments de 0,01 %. Par exemple, la valeur **10** échantillonne 0,1 % des traces, la valeur **500** échantillonne 5 % des traces et la valeur **10000** échantillonne 100 % des traces.

NOTE

L'option de configuration de l'échantillonnage SMCP contrôle le taux d'échantillonnage Envoy. Vous configurez le taux d'échantillonnage de la trace Jaeger dans la ressource personnalisée Jaeger.

1.26.2. Spécification de la configuration de Jaeger dans le SMCP

Vous configurez Jaeger dans la section **addons** de la ressource **ServiceMeshControlPlane**. Cependant, il existe certaines limites à ce que vous pouvez configurer dans le SMCP.

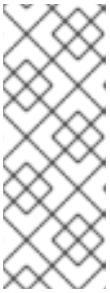
Lorsque le SMCP transmet des informations de configuration à l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift, il déclenche l'une des trois stratégies de déploiement : **allInOne**, **production**, ou **streaming**.

1.26.3. Déployer la plateforme de traçage distribuée

La plate-forme de traçage distribué dispose de stratégies de déploiement prédéfinies. Vous spécifiez une stratégie de déploiement dans le fichier de ressources personnalisées (CR) Jaeger. Lorsque vous créez une instance de la plate-forme de traçage distribuée, l'opérateur de plate-forme de traçage distribuée Red Hat OpenShift utilise ce fichier de configuration pour créer les objets nécessaires au déploiement.

L'opérateur de plateforme de traçage distribuée Red Hat OpenShift prend actuellement en charge les stratégies de déploiement suivantes :

- **allInOne** (par défaut) - Cette stratégie est destinée au développement, aux tests et aux démonstrations et ne doit pas être utilisée en production. Les principaux composants du back-end, l'agent, le collecteur et le service de requête, sont tous regroupés dans un seul exécutable, qui est configuré (par défaut) pour utiliser le stockage en mémoire. Vous pouvez configurer cette stratégie de déploiement dans le SMCP.

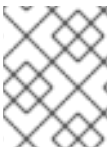


NOTE

Le stockage en mémoire n'est pas persistant, ce qui signifie que si l'instance Jaeger s'arrête, redémarre ou est remplacée, vos données de traçage seront perdues. De plus, le stockage en mémoire ne peut pas être mis à l'échelle, puisque chaque module possède sa propre mémoire. Pour le stockage persistant, vous devez utiliser les stratégies **production** ou **streaming**, qui utilisent Elasticsearch comme stockage par défaut.

- **production** - La stratégie de production est destinée aux environnements de production, où le stockage à long terme des données de traçage est important et où une architecture plus évolutive et hautement disponible est nécessaire. Chaque composant back-end est donc déployé séparément. L'agent peut être injecté en tant que sidecar dans l'application instrumentée. Les services Query et Collector sont configurés avec un type de stockage pris en charge, qui est actuellement Elasticsearch. Plusieurs instances de chacun de ces composants peuvent être provisionnées si nécessaire à des fins de performance et de résilience. Vous pouvez configurer cette stratégie de déploiement dans le SMCP, mais pour qu'elle soit entièrement personnalisée, vous devez spécifier votre configuration dans le Jaeger CR et la relier au SMCP.

- **streaming** - La stratégie de diffusion en continu est conçue pour compléter la stratégie de production en fournissant une capacité de diffusion en continu qui se situe entre le collecteur et le stockage dorsal Elasticsearch. Cela permet de réduire la pression sur le stockage dorsal dans les situations de charge élevée et permet à d'autres capacités de post-traitement des traces d'exploiter les données en temps réel directement à partir de la plateforme de diffusion en continu(AMQ Streams/Kafka). Vous ne pouvez pas configurer cette stratégie de déploiement dans le SMCP ; vous devez configurer un Jaeger CR et le relier au SMCP.



NOTE

La stratégie de streaming nécessite un abonnement Red Hat supplémentaire pour AMQ Streams.

1.26.3.1. Déploiement par défaut de la plate-forme de traçage distribuée

Si vous ne spécifiez pas d'options de configuration Jaeger, la ressource **ServiceMeshControlPlane** utilisera par défaut la stratégie de déploiement Jaeger **allInOne**. Lorsque vous utilisez la stratégie de déploiement par défaut **allInOne**, définissez **spec.addons.jaeger.install.storage.type** sur **Memory**. Vous pouvez accepter les valeurs par défaut ou spécifier des options de configuration supplémentaires sous **install**.

Paramètres Jaeger par défaut du plan de contrôle (mémoire)

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.3
  tracing:
    sampling: 10000
    type: Jaeger
  addons:
    jaeger:
      name: jaeger
```

```
    install:
      storage:
        type: Memory
```

1.26.3.2. Déploiement d'une plate-forme de traçage distribuée en production (minimale)

Pour utiliser les paramètres par défaut de la stratégie de déploiement **production**, définissez **spec.addons.jaeger.install.storage.type** sur **Elasticsearch** et spécifiez des options de configuration supplémentaires sous **install**. Notez que le SMCP ne prend en charge que la configuration des ressources Elasticsearch et du nom de l'image.

Paramètres Jaeger par défaut du plan de contrôle (Elasticsearch)

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.3
  tracing:
    sampling: 10000
    type: Jaeger
  addons:
    jaeger:
      name: jaeger  #name of Jaeger CR
      install:
        storage:
          type: Elasticsearch
        ingress:
          enabled: true
  runtime:
    components:
      tracing.jaeger.elasticsearch: # only supports resources and image name
        container:
          resources: {}
```

1.26.3.3. Déploiement d'une plateforme de traçage distribuée en production (entièrement personnalisée)

Le SMCP ne prend en charge que des paramètres Elasticsearch minimaux. Pour personnaliser entièrement votre environnement de production et accéder à tous les paramètres de configuration d'Elasticsearch, utilisez la ressource personnalisée (CR) Jaeger pour configurer Jaeger.

Créez et configurez votre instance Jaeger et donnez à **spec.addons.jaeger.name** le nom de l'instance Jaeger, dans cet exemple : **MyJaegerInstance**.

Plan de contrôle avec production Jaeger CR liée

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.3
```

```
  tracing:
    sampling: 1000
    type: Jaeger
  addons:
    jaeger:
      name: MyJaegerInstance #name of Jaeger CR
      install:
        storage:
          type: Elasticsearch
        ingress:
          enabled: true
```

1.26.3.4. Déploiement de Jaeger en streaming

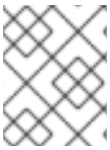
Pour utiliser la stratégie de déploiement **streaming**, vous devez d'abord créer et configurer votre instance Jaeger, puis attribuer à **spec.addons.jaeger.name** le nom de l'instance Jaeger, dans cet exemple : **MyJaegerInstance**.

Plan de contrôle avec flux Jaeger CR lié

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.3
  tracing:
    sampling: 1000
    type: Jaeger
  addons:
    jaeger:
      name: MyJaegerInstance  #name of Jaeger CR
```

1.26.4. Spécifier la configuration de Jaeger dans une ressource personnalisée Jaeger

Vous pouvez personnaliser entièrement votre déploiement Jaeger en configurant Jaeger dans la ressource personnalisée Jaeger (CR) plutôt que dans la ressource **ServiceMeshControlPlane** (SMCP). Cette configuration est parfois appelée "Jaeger externe", car la configuration est spécifiée en dehors du SMCP.



NOTE

Vous devez déployer le SMCP et Jaeger CR dans le même espace de noms. Par exemple, **istio-system**.

Vous pouvez configurer et déployer une instance Jaeger autonome, puis spécifier le site **name** de la ressource Jaeger comme valeur de **spec.addons.jaeger.name** dans la ressource SMCP. S'il existe une CR Jaeger correspondant à la valeur de **name**, le plan de contrôle Service Mesh utilisera l'installation existante. Cette approche vous permet de personnaliser entièrement votre configuration Jaeger.

1.26.4.1. Meilleures pratiques de déploiement

- Les noms des instances de traçage distribuées Red Hat OpenShift doivent être uniques. Si vous souhaitez avoir plusieurs instances de plateforme de traçage distribuée Red Hat OpenShift et que vous utilisez des agents injectés sidecar, les instances de plateforme de traçage distribuée Red Hat OpenShift doivent avoir des noms uniques, et l'annotation d'injection doit explicitement spécifier le nom de l'instance de plateforme de traçage distribuée Red Hat OpenShift vers laquelle les données de traçage doivent être rapportées.

- Si vous avez une implémentation multi-locataires et que les locataires sont séparés par des espaces de noms, déployez une instance de plateforme de traçage distribuée Red Hat OpenShift dans l'espace de noms de chaque locataire.

    - L'agent en tant que daemonset n'est pas pris en charge pour les installations multitenant ou Red Hat OpenShift Dedicated. L'agent en tant que sidecar est la seule configuration supportée pour ces cas d'utilisation.

- Si vous installez le traçage distribué dans le cadre de Red Hat OpenShift Service Mesh, les ressources de traçage distribuées doivent être installées dans le même espace de noms que la ressource **ServiceMeshControlPlane**.

Pour plus d'informations sur la configuration du stockage persistant, voir Comprendre le stockage persistant et la rubrique de configuration appropriée pour l'option de stockage choisie.

1.26.4.2. Configuration de la sécurité du traçage distribué pour le maillage de services

La plateforme de traçage distribuée utilise OAuth pour l'authentification par défaut. Cependant, Red Hat OpenShift Service Mesh utilise un secret appelé **htpasswd** pour faciliter la communication entre les services dépendants tels que Grafana, Kiali et la plateforme de traçage distribuée. Lorsque vous configurez votre plateforme de traçage distribuée dans le site **ServiceMeshControlPlane**, le Service Mesh configure automatiquement les paramètres de sécurité pour utiliser **htpasswd**.

Si vous spécifiez la configuration de votre plateforme de traçage distribuée dans une ressource personnalisée Jaeger, vous devez configurer manuellement les paramètres **htpasswd** et vous assurer que le secret **htpasswd** est monté dans votre instance Jaeger afin que Kiali puisse communiquer avec elle.

1.26.4.2.1. Configurer la sécurité du traçage distribué pour le maillage de services à partir de la console OpenShift

Vous pouvez modifier la ressource Jaeger pour configurer la sécurité de la plateforme de traçage distribuée pour une utilisation avec Service Mesh dans la console OpenShift.

Conditions préalables

- Vous avez accès au cluster en tant qu'utilisateur avec le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

- L'opérateur Red Hat OpenShift Service Mesh doit être installé.

- Le site **ServiceMeshControlPlane** déployé dans le cluster.

- Vous avez accès à la console web de OpenShift Container Platform.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**.

2. Naviguez vers Opérateurs → Opérateurs installés.
3. Cliquez sur le menu **Project** et sélectionnez dans la liste le projet dans lequel votre ressource **ServiceMeshControlPlane** est déployée, par exemple **istio-system**.
4. Cliquez sur le site **Red Hat OpenShift distributed tracing platform Operator**.
5. Sur la page **Operator Details**, cliquez sur l'onglet **Jaeger**.
6. Cliquez sur le nom de votre instance Jaeger.
7. Sur la page de détails de Jaeger, cliquez sur l'onglet **YAML** pour modifier votre configuration.
8. Modifiez le fichier de ressources personnalisées **Jaeger** pour ajouter la configuration **htpasswd**, comme indiqué dans l'exemple suivant.
   - **spec.ingress.openshift.htpasswdFile**
   - **spec.volumes**
   - **spec.volumeMounts**

   Exemple de ressource Jaeger montrant la configuration de htpasswd

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
spec:
  ingress:
    enabled: true
    openshift:
      htpasswdFile: /etc/proxy/htpasswd/auth
      sar: '{"namespace": "istio-system", "resource": "pods", "verb": "get"}'
    options: {}
    resources: {}
    security: oauth-proxy
  volumes:
    - name: secret-htpasswd
      secret:
        secretName: htpasswd
    - configMap:
        defaultMode: 420
        items:
          - key: ca-bundle.crt
            path: tls-ca-bundle.pem
        name: trusted-ca-bundle
        optional: true
      name: trusted-ca-bundle
  volumeMounts:
    - mountPath: /etc/proxy/htpasswd
      name: secret-htpasswd
    - mountPath: /etc/pki/ca-trust/extracted/pem/
      name: trusted-ca-bundle
      readOnly: true
```

9. Cliquez sur **Save**.

1.26.4.2.2. Configuration de la sécurité du traçage distribué pour le maillage de services à partir de la ligne de commande

Vous pouvez modifier la ressource Jaeger pour configurer la sécurité de la plate-forme de traçage distribuée à utiliser avec Service Mesh depuis la ligne de commande à l'aide de l'utilitaire **oc**.

Conditions préalables

- Vous avez accès au cluster en tant qu'utilisateur avec le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.
- L'opérateur Red Hat OpenShift Service Mesh doit être installé.
- Le site **ServiceMeshControlPlane** déployé dans le cluster.
- Vous avez accès à la CLI d'OpenShift (oc) qui correspond à votre version d'OpenShift Container Platform.

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

```
$ oc login https://<HOSTNAME>:6443
```

2. Passez au projet dans lequel vous avez installé le plan de contrôle, par exemple **istio-system**, en entrant la commande suivante :

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

3. Exécutez la commande suivante pour modifier le fichier de ressources personnalisées Jaeger, où **jaeger.yaml** est le nom de votre ressource personnalisée Jaeger.

```
$ oc edit -n tracing-system -f jaeger.yaml
```

4. Modifiez le fichier de ressources personnalisées **Jaeger** pour ajouter la configuration **htpasswd**, comme indiqué dans l'exemple suivant.

   - **spec.ingress.openshift.htpasswdFile**
   - **spec.volumes**
   - **spec.volumeMounts**

   Exemple de ressource Jaeger montrant la configuration de htpasswd

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
spec:
  ingress:
    enabled: true
    openshift:
      htpasswdFile: /etc/proxy/htpasswd/auth
      sar: '{"namespace": "istio-system", "resource": "pods", "verb": "get"}'
```

```
    options: {}
    resources: {}
    security: oauth-proxy
  volumes:
    - name: secret-htpasswd
      secret:
        secretName: htpasswd
    - configMap:
        defaultMode: 420
        items:
          - key: ca-bundle.crt
            path: tls-ca-bundle.pem
        name: trusted-ca-bundle
        optional: true
      name: trusted-ca-bundle
  volumeMounts:
    - mountPath: /etc/proxy/htpasswd
      name: secret-htpasswd
    - mountPath: /etc/pki/ca-trust/extracted/pem/
      name: trusted-ca-bundle
      readOnly: true
```

5. Exécutez la commande suivante pour appliquer vos modifications, où <jaeger.yaml> est le nom de votre ressource personnalisée Jaeger.

```
$ oc apply -n tracing-system -f <jaeger.yaml>
```

6. Exécutez la commande suivante pour suivre la progression du déploiement du pod :

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

1.26.4.3. Options de configuration par défaut du traçage distribué

La ressource personnalisée Jaeger (CR) définit l'architecture et les paramètres à utiliser lors de la création des ressources de la plate-forme de traçage distribuée. Vous pouvez modifier ces paramètres pour adapter la mise en œuvre de votre plate-forme de traçage distribuée aux besoins de votre entreprise.

Exemple de YAML générique Jaeger

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: name
spec:
  strategy: <deployment_strategy>
  allInOne:
    options: {}
    resources: {}
  agent:
    options: {}
    resources: {}
  collector:
    options: {}
```

```
    resources: {}
  sampling:
    options: {}
  storage:
    type:
    options: {}
  query:
    options: {}
    resources: {}
  ingester:
    options: {}
    resources: {}
  options: {}
```

Tableau 1.39. Paramètres Jaeger

Paramètres	Description	Valeurs	Valeur par défaut
apiVersion:		Version de l'API à utiliser lors de la création de l'objet.	**jaegertracing.io/v1**
jaegertracing.io/v1	**kind:**	Définit le type d'objet Kubernetes à créer.	**jaeger**
	metadata:	Données permettant d'identifier l'objet de manière unique, y compris une chaîne de caractères **name**, **UID** et, en option, **namespace**.	
OpenShift Container Platform génère automatiquement le **UID** et complète le **namespace** avec le nom du projet où l'objet est créé.	**name:**	Nom de l'objet.	Le nom de l'instance de la plate-forme de traçage distribuée.

Paramètres	Description	Valeurs	Valeur par défaut
jaeger-all-in-one-inmemory	**spec:**	Spécification de l'objet à créer.	Contient tous les paramètres de configuration de votre instance de plateforme de traçage distribuée. Lorsqu'une définition commune à tous les composants Jaeger est requise, elle est définie sous le nœud **spec**. Lorsque la définition concerne un composant individuel, elle est placée sous le nœud **spec/<component>**.
N/A	**strategy:**	Stratégie de déploiement de Jaeger	**allInOne**, **production**, ou **streaming**
allInOne	**allInOne:**	Comme l'image **allInOne** déploie l'agent, le collecteur, la requête, l'ingestion et l'interface utilisateur Jaeger dans un seul pod, la configuration de ce déploiement doit imbriquer la configuration des composants sous le paramètre **allInOne**.	
	agent:	Options de configuration qui définissent l'agent.	
	collector:	Options de configuration qui définissent le collecteur Jaeger.	
	sampling:	Options de configuration qui définissent les stratégies d'échantillonnage pour le traçage.	

Paramètres	Description	Valeurs	Valeur par défaut
	storage:	Options de configuration qui définissent le stockage. Toutes les options liées au stockage doivent être placées sous **storage**, plutôt que sous **allInOne** ou sous d'autres options de composants.	
	query:	Options de configuration qui définissent le service Query.	
	ingester:	Options de configuration qui définissent le service Ingester.	

L'exemple YAML suivant est le minimum requis pour créer un déploiement de plateforme de traçage distribuée Red Hat OpenShift en utilisant les paramètres par défaut.

Exemple minimum requis dist-tracing-all-in-one.yaml

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-all-in-one-inmemory
```

1.26.4.4. Options de configuration du collecteur Jaeger

Le collecteur Jaeger est le composant responsable de la réception des intervalles capturés par le traceur et de leur écriture dans un stockage Elasticsearch persistant lors de l'utilisation de la stratégie **production**, ou dans des flux AMQ lors de l'utilisation de la stratégie **streaming**.

Les collecteurs sont sans état et de nombreuses instances de Jaeger Collector peuvent donc être exécutées en parallèle. Les collecteurs ne nécessitent pratiquement aucune configuration, à l'exception de l'emplacement du cluster Elasticsearch.

Tableau 1.40. Paramètres utilisés par l'opérateur pour définir le collecteur Jaeger

Paramètres	Description	Valeurs
`collector:` `  replicas:`	Spécifie le nombre de répliques du collecteur à créer.	Entier, par exemple, **5**

Tableau 1.41. Paramètres de configuration transmis au collecteur

Paramètres	Description	Valeurs

`spec:`
`collector:`
`options: {}`
	Options de configuration qui définissent le collecteur Jaeger.	

`options:`
`collector:`
`num-workers:`
	Le nombre de travailleurs tirés de la file d'attente.	Entier, par exemple, **50**

`options:`
`collector:`
`queue-size:`
	Taille de la file d'attente du collecteur.	Entier, par exemple, **2000**

`options:`
`kafka:`
`producer:`
`topic: jaeger-spans`
	Le paramètre **topic** identifie la configuration Kafka utilisée par le collecteur pour produire les messages et par l'ingérateur pour consommer les messages.	Label pour le producteur.

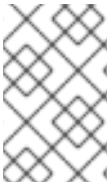
`options:`
`kafka:`
`producer:`
`brokers: my-cluster-kafka-brokers.kafka:9092`
	Identifie la configuration Kafka utilisée par le collecteur pour produire les messages. Si les courtiers ne sont pas spécifiés, et que vous avez installé AMQ Streams 1.4.0, l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift fournira lui-même Kafka.	

`options:` `log-level:`	Niveau de journalisation pour le collecteur.	Valeurs possibles : **debug**, **info**, **warn**, **error**, **fatal**, **panic**.

1.26.4.5. Options de configuration de l'échantillonnage de traçage distribué

La plateforme de traçage distribuée Red Hat OpenShift Operator peut être utilisée pour définir les stratégies d'échantillonnage qui seront fournies aux traceurs qui ont été configurés pour utiliser un échantillonneur distant.

Toutes les traces sont générées, mais seules quelques-unes sont échantillonnées. L'échantillonnage d'une trace la marque pour un traitement et un stockage ultérieurs.



NOTE

Ceci n'est pas pertinent si une trace a été lancée par le proxy Envoy, car la décision d'échantillonnage est prise à ce niveau. La décision d'échantillonnage de Jaeger n'est pertinente que lorsque la trace est lancée par une application utilisant le client.

Lorsqu'un service reçoit une requête qui ne contient pas de contexte de trace, le client lance une nouvelle trace, lui attribue un identifiant aléatoire et prend une décision d'échantillonnage basée sur la stratégie d'échantillonnage actuellement installée. La décision d'échantillonnage se propage à toutes les demandes ultérieures de la trace, de sorte que les autres services ne prennent pas à nouveau la décision d'échantillonnage.

les bibliothèques de la plate-forme de traçage distribué prennent en charge les échantillonneurs suivants :

- **Probabilistic** - L'échantillonneur prend une décision d'échantillonnage aléatoire avec une probabilité d'échantillonnage égale à la valeur de la propriété **sampling.param**. Par exemple, l'utilisation de **sampling.param=0.1** permet d'échantillonner environ 1 trace sur 10.
- **Rate Limiting** - L'échantillonneur utilise un limiteur de taux de leaky bucket pour s'assurer que les traces sont échantillonnées à un certain taux constant. Par exemple, l'utilisation de **sampling.param=2.0** permet d'échantillonner les demandes au rythme de 2 traces par seconde.

Tableau 1.42. Options d'échantillonnage Jaeger

Paramètres	Description	Valeurs	Valeur par défaut
spec: sampling: options: {} default_strategy: service_strategy:	Options de configuration qui définissent les stratégies d'échantillonnage pour le traçage.		Si vous ne fournissez pas de configuration, les collecteurs renverront la politique d'échantillonnage probabiliste par défaut avec une probabilité de 0,001 (0,1 %) pour tous les services.
default_strategy: type: service_strategy: type:	Stratégie d'échantillonnage à utiliser. Voir les descriptions ci-dessus.	Les valeurs valables sont **probabilistic**, et **ratelimiting**.	**probabilistic**
default_strategy: param: service_strategy: param:	Paramètres de la stratégie d'échantillonnage sélectionnée.	Valeurs décimales et entières (0, .1, 1, 10)	1

Cet exemple définit une stratégie d'échantillonnage par défaut qui est probabiliste, avec une probabilité de 50 % que les instances de la trace soient échantillonnées.

Exemple d'échantillonnage probabiliste

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: with-sampling
spec:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 0.5
      service_strategies:
        - service: alpha
          type: probabilistic
          param: 0.8
          operation_strategies:
            - operation: op1
              type: probabilistic
              param: 0.2
            - operation: op2
              type: probabilistic
              param: 0.4
        - service: beta
          type: ratelimiting
          param: 5
```

Si aucune configuration n'est fournie par l'utilisateur, la plate-forme de traçage distribuée utilise les paramètres suivants :

Échantillonnage par défaut

```
spec:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 1
```

1.26.4.6. Options de configuration de la mémoire de traçage distribuée

Vous configurez le stockage pour les services Collector, Ingester et Query à l'adresse **spec.storage**. Plusieurs instances de chacun de ces composants peuvent être provisionnées si nécessaire à des fins de performance et de résilience.

Tableau 1.43. Paramètres généraux de stockage utilisés par l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift pour définir le stockage du traçage distribué

Paramètres	Description	Valeurs	Valeur par défaut
spec: storage: type:	Type de stockage à utiliser pour le déploiement.	**memory** **elasticsearch** le stockage en mémoire n'est approprié que pour les environnements de développement, de test, de démonstration et de validation de principe, car les données ne persistent pas si le module est arrêté. Pour les environnements de production, la plateforme de traçage distribuée prend en charge Elasticsearch pour le stockage persistant.	**memory**
storage: secretname:	Nom du secret, par exemple **tracing-secret**.		N/A
storage: options: {}	Options de configuration qui définissent le stockage.		

Tableau 1.44. Paramètres du nettoyeur d'index Elasticsearch

Paramètres	Description	Valeurs	Valeur par défaut
storage: esIndexCleaner: enabled:	Lors de l'utilisation du stockage Elasticsearch, une tâche est créée par défaut pour nettoyer les anciennes traces de l'index. Ce paramètre permet d'activer ou de désactiver la tâche de nettoyage de l'index.	**true**/ **false**	**true**
storage: esIndexCleaner: numberOfDays:	Nombre de jours à attendre avant de supprimer un index.	Valeur entière	**7**

Paramètres	Description	Valeurs	Valeur par défaut
storage: esIndexCleaner: schedule:	Définit la fréquence de nettoyage de l'index Elasticsearch.	Cron expression	"55 23 * * *"

1.26.4.6.1. Auto-provisionnement d'une instance Elasticsearch

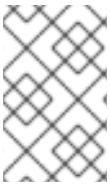
Lorsque vous déployez une ressource personnalisée Jaeger, l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift utilise l'opérateur OpenShift Elasticsearch pour créer un cluster Elasticsearch basé sur la configuration fournie dans la section **storage** du fichier de la ressource personnalisée. L'opérateur de la plateforme de traçage distribuée Red Hat OpenShift provisionnera Elasticsearch si les configurations suivantes sont définies :

- **spec.storage:type** est fixé à **elasticsearch**
- **spec.storage.elasticsearch.doNotProvision** fixé à **false**
- **spec.storage.options.es.server-urls** n'est pas définie, c'est-à-dire qu'il n'y a pas de connexion à une instance d'Elasticsearch qui n'a pas été provisionnée par l'Opérateur Red Hat Elasticsearch.

Lors du provisionnement d'Elasticsearch, l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift définit la ressource personnalisée Elasticsearch **name** à la valeur de **spec.storage.elasticsearch.name** de la ressource personnalisée Jaeger. Si vous ne spécifiez pas de valeur pour **spec.storage.elasticsearch.name**, l'opérateur utilise **elasticsearch**.

Restrictions

- Vous ne pouvez avoir qu'une seule plateforme de traçage distribuée avec instance Elasticsearch auto-provisionnée par espace de noms. Le cluster Elasticsearch doit être dédié à une seule instance de plateforme de traçage distribuée.
- Il ne peut y avoir qu'un seul Elasticsearch par espace de noms.



NOTE

Si vous avez déjà installé Elasticsearch dans le cadre d'OpenShift Logging, l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift peut utiliser l'opérateur OpenShift Elasticsearch installé pour provisionner le stockage.

Les paramètres de configuration suivants concernent une instance Elasticsearch *self-provisioned*, c'est-à-dire une instance créée par l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift à l'aide de l'opérateur OpenShift Elasticsearch. Vous spécifiez les options de configuration pour Elasticsearch auto-provisionné sous **spec:storage:elasticsearch** dans votre fichier de configuration.

Tableau 1.45. Paramètres de configuration des ressources Elasticsearch

Paramètres	Description	Valeurs	Valeur par défaut
elasticsearch: properties: doNotProvision:	À utiliser pour spécifier si une instance Elasticsearch doit être provisionnée par l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift.	**true**/**false**	**true**
elasticsearch: properties: name:	Nom de l'instance Elasticsearch. L'opérateur de la plateforme de traçage distribuée Red Hat OpenShift utilise l'instance Elasticsearch spécifiée dans ce paramètre pour se connecter à Elasticsearch.	chaîne de caractères	**elasticsearch**
elasticsearch: nodeCount:	Nombre de nœuds Elasticsearch. Pour la haute disponibilité, utilisez au moins 3 nœuds. Ne pas utiliser 2 nœuds car un problème de "split brain" peut se produire.	Valeur entière. Par exemple, Preuve de concept = 1, Déploiement minimum =3	3
elasticsearch: resources: requests: cpu:	Nombre d'unités centrales de traitement pour les requêtes, en fonction de la configuration de votre environnement.	Spécifié en cœurs ou en millicores, par exemple 200m, 0,5, 1. Par exemple, preuve de concept = 500m, déploiement minimum =1	1
elasticsearch: resources: requests: memory:	Mémoire disponible pour les requêtes, en fonction de la configuration de votre environnement.	Spécifié en octets, par exemple 200Ki, 50Mi, 5Gi. Par exemple, preuve de concept = 1Gi, déploiement minimal = 16Gi*	16Gi

Paramètres	Description	Valeurs	Valeur par défaut
elasticsearch: resources: limits: cpu:	Limitation du nombre d'unités centrales de traitement, en fonction de la configuration de votre environnement.	Spécifié en cœurs ou en millicores, par exemple 200m, 0,5, 1. Par exemple, preuve de concept = 500m, déploiement minimum =1	
elasticsearch: resources: limits: memory:	Limite de mémoire disponible en fonction de la configuration de votre environnement.	Spécifié en octets, par exemple 200Ki, 50Mi, 5Gi. Par exemple, preuve de concept = 1Gi, déploiement minimal = 16Gi*	
elasticsearch: redundancyPolicy:	La politique de réplication des données définit comment les shards Elasticsearch sont répliqués à travers les nœuds de données dans le cluster. S'il n'est pas spécifié, l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift détermine automatiquement la réplication la plus appropriée en fonction du nombre de nœuds.	**ZeroRedundancy**(pas de répliques), **SingleRedundancy**(une réplique), **MultipleRedundancy** (chaque index est réparti sur la moitié des nœuds de données), **FullRedundancy** (chaque index est entièrement répliqué sur chaque nœud de données de la grappe).	
elasticsearch: useCertManagement:	À utiliser pour spécifier si la plateforme de traçage distribuée doit ou non utiliser la fonctionnalité de gestion des certificats de Red Hat Elasticsearch Operator. Cette fonctionnalité a été ajoutée au sous-système de journalisation pour Red Hat OpenShift 5.2 dans OpenShift Container Platform 4.7 et est le paramètre préféré pour les nouveaux déploiements de Jaeger.	**true**/**false**	**true**

Paramètres	Description	Valeurs	Valeur par défaut
	*Chaque nœud Elasticsearch peut fonctionner avec un paramètre de mémoire inférieur, mais cela n'est PAS recommandé pour les déploiements en production. Pour une utilisation en production, vous ne devriez pas avoir moins de 16Gi alloués à chaque pod par défaut, mais de préférence allouez autant que possible, jusqu'à 64Gi par pod.		

Exemple de stockage de production

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 3
      resources:
        requests:
          cpu: 1
          memory: 16Gi
        limits:
          memory: 16Gi
```

Exemple de stockage avec stockage persistant :

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 1
      storage: (1)
        storageClassName: gp2
        size: 5Gi
      resources:
        requests:
          cpu: 200m
          memory: 4Gi
        limits:
          memory: 4Gi
      redundancyPolicy: ZeroRedundancy
```

(1) Configuration du stockage persistant. Dans ce cas, AWS **gp2** avec une taille de **5Gi**. Si aucune valeur n'est spécifiée, la plateforme de traçage distribuée utilise **emptyDir**. L'OpenShift Elasticsearch Operator fournit **PersistentVolumeClaim** et **PersistentVolume** qui ne sont pas

supprimés avec l'instance de plateforme de traçage distribuée. Vous pouvez monter les mêmes volumes si vous créez une instance de plateforme de traçage distribuée avec le même nom et le même espace de noms.

1.26.4.6.2. Connexion à une instance Elasticsearch existante

Vous pouvez utiliser un cluster Elasticsearch existant pour le stockage avec le traçage distribué. Un cluster Elasticsearch existant, également connu sous le nom d'instance Elasticsearch *external*, est une instance qui n'a pas été installée par l'opérateur de plateforme de traçage distribué Red Hat OpenShift ou par l'opérateur Red Hat Elasticsearch.

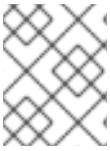
Lorsque vous déployez une ressource personnalisée Jaeger, l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift ne provisionnera pas Elasticsearch si les configurations suivantes sont définies :

- **spec.storage.elasticsearch.doNotProvision** fixé à **true**
- **spec.storage.options.es.server-urls** a une valeur
- **spec.storage.elasticsearch.name** a une valeur, ou si le nom de l'instance Elasticsearch est **elasticsearch**.

L'opérateur de la plateforme de traçage distribuée Red Hat OpenShift utilise l'instance Elasticsearch spécifiée dans **spec.storage.elasticsearch.name** pour se connecter à Elasticsearch.

Restrictions

- Vous ne pouvez pas partager ou réutiliser une instance Elasticsearch de la plateforme OpenShift Container avec une plateforme de traçage distribuée. Le cluster Elasticsearch doit être dédié à une seule instance de plateforme de traçage distribuée.



NOTE

Red Hat ne fournit pas de support pour votre instance Elasticsearch externe. Vous pouvez consulter la matrice des intégrations testées sur le portail client.

Les paramètres de configuration suivants s'appliquent à une instance Elasticsearch déjà existante, également appelée instance Elasticsearch *external*. Dans ce cas, vous spécifiez les options de configuration pour Elasticsearch sous **spec:storage:options:es** dans votre fichier de ressources personnalisé.

Tableau 1.46. Paramètres généraux de configuration de l'ES

Paramètres	Description	Valeurs	Valeur par défaut
`es:` `  server-urls:`	URL de l'instance Elasticsearch.	Le nom de domaine complet du serveur Elasticsearch.	**http://elasticsearch.<namespace>.svc:9200**

Paramètres	Description	Valeurs	Valeur par défaut
es: max-doc-count:	Nombre maximal de documents à renvoyer à partir d'une requête Elasticsearch. Cette valeur s'applique également aux agrégations. Si vous définissez à la fois **es.max-doc-count** et **es.max-num-spans**, Elasticsearch utilisera la plus petite valeur des deux.		10000
es: max-num-spans:	[**Deprecated** - Sera supprimé dans une prochaine version, utilisez **es.max-doc-count** à la place] Le nombre maximum de portées à récupérer à la fois, par requête, dans Elasticsearch. Si vous définissez à la fois **es.max-num-spans** et **es.max-doc-count**, Elasticsearch utilisera la plus petite valeur des deux.		10000
es: max-span-age:	Le délai maximal de consultation pour les périodes dans Elasticsearch.		72h0m0s
es: sniffer:	La configuration du renifleur pour Elasticsearch. Le client utilise le processus de reniflage pour trouver automatiquement tous les nœuds. Désactivé par défaut.	**true**/ **false**	**false**

Paramètres	Description	Valeurs	Valeur par défaut
es: sniffer-tls- enabled:	Option permettant d'activer TLS lors du sniffing d'un cluster Elasticsearch. Le client utilise le processus de reniflage pour trouver automatiquement tous les nœuds. Désactivé par défaut	**true**/ **false**	**false**
es: timeout:	Délai d'attente utilisé pour les requêtes. S'il est fixé à zéro, il n'y a pas de délai d'attente.		0s
es: username:	Le nom d'utilisateur requis par Elasticsearch. L'authentification de base charge également l'AC si elle est spécifiée. Voir aussi **es.password**.		
es: password:	Le mot de passe requis par Elasticsearch. Voir aussi **es.username**.		
es: version:	La version majeure d'Elasticsearch. Si elle n'est pas spécifiée, la valeur sera auto-détectée à partir d'Elasticsearch.		0

Tableau 1.47. Paramètres de réplication des données ES

Paramètres	Description	Valeurs	Valeur par défaut
es: num-replicas:	Le nombre de répliques par index dans Elasticsearch.		1
es: num-shards:	Le nombre d'unités par index dans Elasticsearch.		5

Tableau 1.48. Paramètres de configuration de l'index ES

Paramètres	Description	Valeurs	Valeur par défaut
es: create-index-templates:	Créer automatiquement des modèles d'index au démarrage de l'application lorsque le paramètre est fixé à **true**. Lorsque les modèles sont installés manuellement, le paramètre est fixé à **false**.	**true**/ **false**	**true**
es: index-prefix:	Préfixe facultatif pour les index de la plate-forme de traçage distribuée. Par exemple, la valeur "production" crée des index nommés "production-tracing-*".		

Tableau 1.49. Paramètres de configuration du processeur ES bulk

Paramètres	Description	Valeurs	Valeur par défaut
es: bulk: actions:	Nombre de demandes pouvant être ajoutées à la file d'attente avant que le processeur de masse ne décide d'enregistrer les mises à jour sur le disque.		1000
es: bulk: flush-interval:	**time.Duration** - Intervalle après lequel les demandes groupées sont validées, quels que soient les autres seuils. Pour désactiver l'intervalle de vidange du processeur de traitement en masse, fixer cette valeur à zéro.		200ms

Paramètres	Description	Valeurs	Valeur par défaut
es: bulk: size:	Nombre d'octets que les demandes groupées peuvent occuper avant que le processeur de traitement groupé ne décide d'enregistrer les mises à jour sur le disque.		5000000
es: bulk: workers:	Le nombre de travailleurs capables de recevoir et d'envoyer des requêtes en masse à Elasticsearch.		1

Tableau 1.50. Paramètres de configuration ES TLS

Paramètres	Description	Valeurs	Valeur par défaut
es: tls: ca:	Chemin d'accès à un fichier d'autorité de certification (CA) TLS utilisé pour vérifier les serveurs distants.		Utilise par défaut le truststore du système.
es: tls: cert:	Chemin d'accès à un fichier de certificat TLS, utilisé pour identifier ce processus auprès des serveurs distants.		
es: tls: enabled:	Activer la sécurité de la couche transport (TLS) lors de la communication avec les serveurs distants. Cette option est désactivée par défaut.	**true**/ **false**	**false**
es: tls: key:	Chemin d'accès à un fichier de clé privée TLS, utilisé pour identifier ce processus auprès des serveurs distants.		
es: tls: server-name:	Remplacer le nom du serveur TLS prévu dans le certificat des serveurs distants.		

Paramètres	Description	Valeurs	Valeur par défaut
es: token-file:	Chemin d'accès au fichier contenant le jeton du porteur. Cet indicateur charge également le fichier de l'autorité de certification (CA) s'il est spécifié.		

Tableau 1.51. Paramètres de configuration de l'archive ES

Paramètres	Description	Valeurs	Valeur par défaut
es-archive: bulk: actions:	Nombre de demandes pouvant être ajoutées à la file d'attente avant que le processeur de masse ne décide d'enregistrer les mises à jour sur le disque.		0
es-archive: bulk: flush-interval:	**time.Duration** - Intervalle après lequel les demandes groupées sont validées, quels que soient les autres seuils. Pour désactiver l'intervalle de vidange du processeur de traitement en masse, fixer cette valeur à zéro.		0s
es-archive: bulk: size:	Nombre d'octets que les demandes groupées peuvent occuper avant que le processeur de traitement groupé ne décide d'enregistrer les mises à jour sur le disque.		0

Paramètres	Description	Valeurs	Valeur par défaut
es-archive: bulk: workers:	Le nombre de travailleurs capables de recevoir et d'envoyer des requêtes en masse à Elasticsearch.		0
es-archive: create-index-templates:	Créer automatiquement des modèles d'index au démarrage de l'application lorsque le paramètre est fixé à **true**. Lorsque les modèles sont installés manuellement, le paramètre est fixé à **false**.	**true**/ **false**	**false**
es-archive: enabled:	Permettre un stockage supplémentaire.	**true**/ **false**	**false**
es-archive: index-prefix:	Préfixe facultatif pour les index de la plate-forme de traçage distribuée. Par exemple, la valeur "production" crée des index nommés "production-tracing-*".		
es-archive: max-doc-count:	Nombre maximal de documents à renvoyer à partir d'une requête Elasticsearch. Cette valeur s'applique également aux agrégations.		0
es-archive: max-num-spans:	[**Deprecated** - Sera supprimé dans une prochaine version, utilisez **es-archive.max-doc-count** à la place] Le nombre maximum d'étendues à récupérer à la fois, par requête, dans Elasticsearch.		0

Paramètres	Description	Valeurs	Valeur par défaut
es-archive: max-span-age:	Le délai maximal de consultation pour les périodes dans Elasticsearch.		0s
es-archive: num-replicas:	Le nombre de répliques par index dans Elasticsearch.		0
es-archive: num-shards:	Le nombre d'unités par index dans Elasticsearch.		0
es-archive: password:	Le mot de passe requis par Elasticsearch. Voir aussi **es.username**.		
es-archive: server-urls:	La liste des serveurs Elasticsearch, séparée par des virgules. Les serveurs doivent être spécifiés sous forme d'URL complètes, par exemple, **http://localhost:9200**.		
es-archive: sniffer:	La configuration du renifleur pour Elasticsearch. Le client utilise le processus de reniflage pour trouver automatiquement tous les nœuds. Désactivé par défaut.	**true**/ **false**	**false**
es-archive: sniffer-tls-enabled:	Option permettant d'activer TLS lors du sniffing d'un cluster Elasticsearch. Le client utilise le processus de reniflage pour trouver automatiquement tous les nœuds. Cette option est désactivée par défaut.	**true**/ **false**	**false**

Paramètres	Description	Valeurs	Valeur par défaut
es-archive: timeout:	Délai d'attente utilisé pour les requêtes. S'il est fixé à zéro, il n'y a pas de délai d'attente.		0s
es-archive: tls: ca:	Chemin d'accès à un fichier d'autorité de certification (CA) TLS utilisé pour vérifier les serveurs distants.		Utilise par défaut le truststore du système.
es-archive: tls: cert:	Chemin d'accès à un fichier de certificat TLS, utilisé pour identifier ce processus auprès des serveurs distants.		
es-archive: tls: enabled:	Activer la sécurité de la couche transport (TLS) lors de la communication avec les serveurs distants. Cette option est désactivée par défaut.	**true**/ **false**	**false**
es-archive: tls: key:	Chemin d'accès à un fichier de clé privée TLS, utilisé pour identifier ce processus auprès des serveurs distants.		
es-archive: tls: server-name:	Remplacer le nom du serveur TLS prévu dans le certificat des serveurs distants.		
es-archive: token-file:	Chemin d'accès au fichier contenant le jeton du porteur. Cet indicateur charge également le fichier de l'autorité de certification (CA) s'il est spécifié.		

Paramètres	Description	Valeurs	Valeur par défaut
es-archive: username:	Le nom d'utilisateur requis par Elasticsearch. L'authentification de base charge également l'AC si elle est spécifiée. Voir aussi **es-archive.password**.		
es-archive: version:	La version majeure d'Elasticsearch. Si elle n'est pas spécifiée, la valeur sera auto-détectée à partir d'Elasticsearch.		0

Exemple de stockage avec des montages de volumes

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200
        index-prefix: my-prefix
        tls:
          ca: /es/certificates/ca.crt
    secretName: tracing-secret
  volumeMounts:
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
  volumes:
    - name: certificates
      secret:
        secretName: quickstart-es-http-certs-public
```

L'exemple suivant montre un Jaeger CR utilisant un cluster Elasticsearch externe avec un certificat TLS CA monté à partir d'un volume et un utilisateur/mot de passe stocké dans un secret.

Exemple d'Elasticsearch externe :

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
```

```
strategy: production
storage:
  type: elasticsearch
  options:
    es:
      server-urls: https://quickstart-es-http.default.svc:9200 (1)
      index-prefix: my-prefix
      tls: (2)
        ca: /es/certificates/ca.crt
  secretName: tracing-secret (3)
volumeMounts: (4)
  - name: certificates
    mountPath: /es/certificates/
    readOnly: true
volumes:
  - name: certificates
    secret:
      secretName: quickstart-es-http-certs-public
```

1. URL du service Elasticsearch fonctionnant dans l'espace de noms par défaut.
2. Configuration TLS. Dans ce cas, il s'agit uniquement du certificat de l'autorité de certification, mais il peut également contenir es.tls.key et es.tls.cert lors de l'utilisation de TLS mutuel.
3. Secret qui définit les variables d'environnement ES_PASSWORD et ES_USERNAME. Créé par kubectl create secret generic tracing-secret --from-literal=ES_PASSWORD=changeme --from-literal=ES_USERNAME=elastic
4. Les montages de volumes et les volumes qui sont montés dans tous les composants de stockage.

1.26.4.7. Gestion des certificats avec Elasticsearch

Vous pouvez créer et gérer des certificats à l'aide de l'Opérateur Red Hat Elasticsearch. La gestion des certificats à l'aide de Red Hat Elasticsearch Operator vous permet également d'utiliser un seul cluster Elasticsearch avec plusieurs collecteurs Jaeger.

IMPORTANT

La gestion des certificats avec Elasticsearch est une fonctionnalité d'aperçu technologique uniquement. Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas de les utiliser en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, ce qui permet aux clients de tester les fonctionnalités et de fournir un retour d'information pendant le processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

À partir de la version 2.4, l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift délègue la création de certificats à l'opérateur Red Hat Elasticsearch en utilisant les annotations suivantes dans la ressource personnalisée Elasticsearch :

- **logging.openshift.io/elasticsearch-cert-management: "true"**
- **logging.openshift.io/elasticsearch-cert.jaeger-<shared-es-node-name>: "user.jaeger"**
- **logging.openshift.io/elasticsearch-cert.curator-<shared-es-node-name>: "system.logging.curator"**

Où **<shared-es-node-name>** est le nom du nœud Elasticsearch. Par exemple, si vous créez un nœud Elasticsearch nommé **custom-es**, votre ressource personnalisée pourrait ressembler à l'exemple suivant.

Exemple de CR Elasticsearch montrant les annotations

```
apiVersion: logging.openshift.io/v1
kind: Elasticsearch
metadata:
  annotations:
    logging.openshift.io/elasticsearch-cert-management: "true"
    logging.openshift.io/elasticsearch-cert.jaeger-custom-es: "user.jaeger"
    logging.openshift.io/elasticsearch-cert.curator-custom-es: "system.logging.curator"
  name: custom-es
spec:
  managementState: Managed
  nodeSpec:
    resources:
      limits:
        memory: 16Gi
      requests:
        cpu: 1
        memory: 16Gi
  nodes:
    - nodeCount: 3
      proxyResources: {}
      resources: {}
      roles:
        - master
        - client
        - data
      storage: {}
  redundancyPolicy: ZeroRedundancy
```

Conditions préalables

- OpenShift Container Platform 4.7
- sous-système de journalisation pour Red Hat OpenShift 5.2
- Le nœud Elasticsearch et les instances Jaeger doivent être déployés dans le même espace de noms. Par exemple, **tracing-system**.

Vous activez la gestion des certificats en définissant **spec.storage.elasticsearch.useCertManagement** sur **true** dans la ressource personnalisée Jaeger.

Exemple montrant **useCertManagement**

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      name: custom-es
      doNotProvision: true
      useCertManagement: true
```

L'opérateur de la plateforme de traçage distribuée Red Hat OpenShift définit la ressource personnalisée Elasticsearch **name** à la valeur de **spec.storage.elasticsearch.name** de la ressource personnalisée Jaeger lors du provisionnement d'Elasticsearch.

Les certificats sont fournis par l'opérateur Red Hat Elasticsearch et l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift injecte les certificats.

Pour plus d'informations sur la configuration d'Elasticsearch avec OpenShift Container Platform, voir Configuration du magasin de logs ou Configuration et déploiement du traçage distribué .

1.26.4.8. Options de configuration des requêtes

Query est un service qui récupère les traces du stockage et héberge l'interface utilisateur pour les afficher.

Tableau 1.52. Paramètres utilisés par l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift pour définir la requête

Paramètres	Description	Valeurs	Valeur par défaut

`spec:`
`  query:`
`  replicas:`
	Spécifie le nombre de répliques de requêtes à créer.	Entier, par exemple, **2**	

Tableau 1.53. Paramètres de configuration transmis à la requête

Paramètres	Description	Valeurs	Valeur par défaut

`spec:`
`  query:`
`  options: {}`
	Options de configuration qui définissent le service Query.		

`options:` `  log-level:`	Niveau de journalisation pour Query.	Valeurs possibles : **debug**, **info**, **warn**, **error**, **fatal**, **panic**.	

Paramètres	Description	Valeurs	Valeur par défaut

`options:`
`  query:`
`  base-path:`
	Le chemin de base de toutes les routes HTTP de jaeger-query peut être défini à une valeur non racine, par exemple, **/jaeger** fera commencer toutes les URL de l'interface utilisateur par **/jaeger**. Cela peut s'avérer utile lorsque jaeger-query est exécuté derrière un proxy inverse.	/<path>	

Exemple de configuration d'une requête

```
apiVersion: jaegertracing.io/v1
kind: "Jaeger"
metadata:
  name: "my-jaeger"
spec:
  strategy: allInOne
  allInOne:
    options:
      log-level: debug
      query:
        base-path: /jaeger
```

1.26.4.9. Options de configuration de l'ingestionur

Ingester est un service qui lit à partir d'un sujet Kafka et écrit dans le backend de stockage Elasticsearch. Si vous utilisez les stratégies de déploiement **allInOne** ou **production**, vous n'avez pas besoin de configurer le service Ingester.

Tableau 1.54. Paramètres du Jaeger transmis à l'injecteur

Paramètres	Description	Valeurs

`spec:`
`  ingester:`
`  options: {}`
	Options de configuration qui définissent le service Ingester.	

Paramètres	Description	Valeurs
`options:` `  deadlockInterval:`	Spécifie l'intervalle, en secondes ou en minutes, pendant lequel l'intégrateur doit attendre un message avant de s'arrêter. L'intervalle d'impasse est désactivé par défaut (défini sur **0**), afin d'éviter que l'ingester ne se termine lorsqu'aucun message n'arrive lors de l'initialisation du système.	Minutes et secondes, par exemple **1m0s**. La valeur par défaut est**0**.

`options:`
`  kafka:`
`  consumer:`
`  topic:`
	Le paramètre **topic** identifie la configuration Kafka utilisée par le collecteur pour produire les messages et par l'ingérant pour consommer les messages.	Étiquette destinée au consommateur. Par exemple, **jaeger-spans**.

`options:`
`  kafka:`
`  consumer:`
`  brokers:`
	Identifie la configuration Kafka utilisée par l'ingester pour consommer les messages.	Label pour le courtier, par exemple, **my-cluster-kafka-brokers.kafka:9092**.

`options:` `  log-level:`	Niveau de journalisation pour l'ingérant.	Valeurs possibles : **debug**, **info**, **warn**, **error**, **fatal**, **dpanic**, **panic**.

Exemple de collecteur et d'ingérateur de flux

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-streaming
spec:
  strategy: streaming
  collector:
    options:
      kafka:
        producer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
  ingester:
    options:
      kafka:
        consumer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
      ingester:
        deadlockInterval: 5
```

```
storage:
  type: elasticsearch
  options:
    es:
      server-urls: http://elasticsearch:9200
```

1.27. DÉSINSTALLATION DE SERVICE MESH

Pour désinstaller Red Hat OpenShift Service Mesh d'une instance OpenShift Container Platform existante et supprimer ses ressources, vous devez supprimer le plan de contrôle, supprimer les opérateurs et exécuter des commandes pour supprimer manuellement certaines ressources.

1.27.1. Suppression du plan de contrôle Red Hat OpenShift Service Mesh

Pour désinstaller Service Mesh d'une instance OpenShift Container Platform existante, vous devez d'abord supprimer le plan de contrôle Service Mesh et les opérateurs. Ensuite, vous exécutez des commandes pour supprimer les ressources résiduelles.

1.27.1.1. Suppression du plan de contrôle Service Mesh à l'aide de la console web

Vous pouvez supprimer le plan de contrôle Red Hat OpenShift Service Mesh en utilisant la console web.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Cliquez sur le menu **Project** et sélectionnez le projet dans lequel vous avez installé le plan de contrôle Service Mesh, par exemple **istio-system**.
3. Naviguez jusqu'à **Operators → Installed Operators**.
4. Cliquez sur **Service Mesh Control Plane** sous **Provided APIs**.
5. Cliquez sur le menu **ServiceMeshControlPlane**  .
6. Cliquez sur **Delete Service Mesh Control Plane**
7. Cliquez sur **Delete** dans la fenêtre de dialogue de confirmation pour supprimer **ServiceMeshControlPlane**.

1.27.1.2. Suppression du plan de contrôle Service Mesh à l'aide de la CLI

Vous pouvez supprimer le plan de contrôle Red Hat OpenShift Service Mesh en utilisant le CLI. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle.

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform.
2. Exécutez la commande suivante pour supprimer la ressource **ServiceMeshMemberRoll**.

```
$ oc delete smmr -n istio-system default
```

3. Exécutez cette commande pour récupérer le nom de la version installée de **ServiceMeshControlPlane**:

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

4. Remplacez **<name_of_custom_resource>** par le résultat de la commande précédente et exécutez cette commande pour supprimer la ressource personnalisée :

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

1.27.2. Retrait des opérateurs installés

Vous devez supprimer les opérateurs pour réussir à supprimer Red Hat OpenShift Service Mesh. Après avoir supprimé l'opérateur Red Hat OpenShift Service Mesh, vous devez supprimer l'opérateur Kiali, l'opérateur Red Hat OpenShift distributed tracing platform et l'opérateur OpenShift Elasticsearch.

1.27.2.1. Retrait des opérateurs

Suivez cette procédure pour supprimer les opérateurs qui composent Red Hat OpenShift Service Mesh. Répétez les étapes pour chacun des opérateurs suivants.

- Red Hat OpenShift Service Mesh
- Kiali
- Plateforme de traçage distribuée Red Hat OpenShift
- OpenShift Elasticsearch

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. À partir de la page **Operators → Installed Operators**, faites défiler ou tapez un mot-clé dans la page **Filter by name** pour trouver chaque opérateur. Cliquez ensuite sur le nom de l'opérateur.
3. Sur la page **Operator Details**, sélectionnez **Uninstall Operator** dans le menu **Actions**. Suivez les instructions pour désinstaller chaque opérateur.

1.27.3. Nettoyer les ressources de l'opérateur

Vous pouvez supprimer manuellement les ressources restantes après avoir supprimé l'opérateur Red Hat OpenShift Service Mesh à l'aide de la console web OpenShift Container Platform.

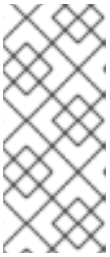
Conditions préalables

- Un compte avec un accès à l'administration du cluster. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.
- Accès à la CLI OpenShift (**oc**).

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'administrateur de cluster.

2. Exécutez les commandes suivantes pour nettoyer les ressources après avoir désinstallé les opérateurs. Si vous avez l'intention de continuer à utiliser la plate-forme de traçage distribuée en tant que service autonome sans service mesh, ne supprimez pas les ressources Jaeger.



NOTE

L'opérateur OpenShift Elasticsearch est installé par défaut dans **openshift-operators-redhat**. Les autres opérateurs sont installés par défaut dans l'espace de noms **openshift-operators**. Si vous avez installé les opérateurs dans un autre espace de noms, remplacez **openshift-operators** par le nom du projet dans lequel l'opérateur Red Hat OpenShift Service Mesh a été installé.

```
$ oc delete validatingwebhookconfiguration/openshift-operators.servicemesh-resources.maistra.io
```

```
$ oc delete mutatingwebhookconfiguration/openshift-operators.servicemesh-resources.maistra.io
```

```
$ oc delete svc maistra-admission-controller -n openshift-operators
```

```
$ oc -n openshift-operators delete ds -lmaistra-version
```

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

```
$ oc delete clusterrole istio-view istio-edit
```

```
$ oc delete clusterrole jaegers.jaegertracing.io-v1-admin jaegers.jaegertracing.io-v1-crdview jaegers.jaegertracing.io-v1-edit jaegers.jaegertracing.io-v1-view
```

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

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

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

```
$ oc delete crds jaegers.jaegertracing.io
```

```
$ oc delete cm -n openshift-operators maistra-operator-cabundle
```

```
$ oc delete cm -n openshift-operators istio-cni-config istio-cni-config-v2-3
```

```
$ oc delete sa -n openshift-operators istio-cni
```

CHAPITRE 2. SERVICE MESH 1.X

2.1. SERVICE MESH - NOTES DE MISE À JOUR

AVERTISSEMENT

You are viewing documentation for a Red Hat OpenShift Service Mesh release that is no longer supported.

Les plans de contrôle Service Mesh version 1.0 et 1.1 ne sont plus pris en charge. Pour plus d'informations sur la mise à niveau de votre plan de contrôle Service Mesh, voir Mise à niveau de Service Mesh .

Pour plus d'informations sur l'état de l'assistance d'une version particulière de Red Hat OpenShift Service Mesh, consultez la page Cycle de vie du produit .

2.1.1. Rendre l'open source plus inclusif

Red Hat s'engage à remplacer les termes problématiques dans son code, sa documentation et ses propriétés Web. Nous commençons par ces quatre termes : master, slave, blacklist et whitelist. En raison de l'ampleur de cette entreprise, ces changements seront mis en œuvre progressivement au cours de plusieurs versions à venir. Pour plus de détails, voir le message de notre directeur technique Chris Wright.

2.1.2. Introduction à Red Hat OpenShift Service Mesh

Red Hat OpenShift Service Mesh répond à une variété de problèmes dans une architecture de microservices en créant un point de contrôle centralisé dans une application. Il ajoute une couche transparente sur les applications distribuées existantes sans nécessiter de modifications du code de l'application.

Les architectures de microservices divisent le travail des applications d'entreprise en services modulaires, ce qui peut faciliter la mise à l'échelle et la maintenance. Cependant, lorsqu'une application d'entreprise construite sur une architecture de microservices augmente en taille et en complexité, elle devient difficile à comprendre et à gérer. Le Service Mesh peut résoudre ces problèmes d'architecture en capturant ou en interceptant le trafic entre les services et peut modifier, rediriger ou créer de nouvelles requêtes vers d'autres services.

Service Mesh, qui est basé sur le projet open source Istio, permet de créer facilement un réseau de services déployés qui assure la découverte, l'équilibrage de charge, l'authentification de service à service, la reprise sur panne, les mesures et la surveillance. Un maillage de services offre également des fonctionnalités opérationnelles plus complexes, notamment les tests A/B, les versions canary, le contrôle d'accès et l'authentification de bout en bout.

2.1.3. Obtenir de l'aide

Si vous rencontrez des difficultés avec une procédure décrite dans cette documentation, ou avec OpenShift Container Platform en général, visitez le portail client de Red Hat . À partir du portail client, vous pouvez :

- Recherchez ou parcourez la base de connaissances de Red Hat qui contient des articles et des solutions relatifs aux produits Red Hat.
- Soumettre un cas d'assistance à Red Hat Support.
- Accéder à d'autres documents sur les produits.

Pour identifier les problèmes liés à votre cluster, vous pouvez utiliser Insights dans OpenShift Cluster Manager Hybrid Cloud Console. Insights fournit des détails sur les problèmes et, le cas échéant, des informations sur la manière de les résoudre.

Si vous avez une suggestion pour améliorer cette documentation ou si vous avez trouvé une erreur, soumettez un problème Jira pour le composant de documentation le plus pertinent. Veuillez fournir des détails spécifiques, tels que le nom de la section et la version d'OpenShift Container Platform.

Lorsque vous ouvrez un dossier d'assistance, il est utile de fournir des informations de débogage sur votre cluster à l'équipe d'assistance de Red Hat.

L'outil **must-gather** vous permet de collecter des informations de diagnostic sur votre cluster OpenShift Container Platform, y compris les machines virtuelles et d'autres données liées à Red Hat OpenShift Service Mesh.

Pour une assistance rapide, fournissez des informations de diagnostic pour OpenShift Container Platform et Red Hat OpenShift Service Mesh.

2.1.3.1. À propos de l'outil de collecte obligatoire

La commande CLI **oc adm must-gather** recueille les informations de votre cluster les plus susceptibles d'être nécessaires au débogage des problèmes, notamment

- Définitions des ressources
- Journaux de service

Par défaut, la commande **oc adm must-gather** utilise l'image du plugin par défaut et écrit dans **./must-gather.local**.

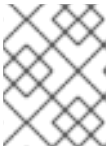
Vous pouvez également recueillir des informations spécifiques en exécutant la commande avec les arguments appropriés, comme décrit dans les sections suivantes :

- Pour collecter des données relatives à une ou plusieurs caractéristiques spécifiques, utilisez l'argument **--image** avec une image, comme indiqué dans la section suivante.
  Par exemple :

```
$ oc adm must-gather  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.12.0
```

- Pour collecter les journaux d'audit, utilisez l'argument **-- /usr/bin/gather_audit_logs**, comme décrit dans la section suivante.
  Par exemple :

```
$ oc adm must-gather -- /usr/bin/gather_audit_logs
```



NOTE

Les journaux d'audit ne sont pas collectés dans le cadre de l'ensemble d'informations par défaut afin de réduire la taille des fichiers.

Lorsque vous exécutez **oc adm must-gather**, un nouveau module portant un nom aléatoire est créé dans un nouveau projet sur le cluster. Les données sont collectées sur ce module et enregistrées dans un nouveau répertoire commençant par **must-gather.local**. Ce répertoire est créé dans le répertoire de travail actuel.

Par exemple :

```
NAMESPACE                  NAME             READY   STATUS      RESTARTS      AGE
...
openshift-must-gather-5drcj    must-gather-bklx4    2/2     Running     0          72s
openshift-must-gather-5drcj    must-gather-s8sdh    2/2     Running     0          72s
...
```

2.1.3.2. Conditions préalables

- Accès au cluster en tant qu'utilisateur ayant le rôle **cluster-admin**.
- L'OpenShift Container Platform CLI (**oc**) est installé.

2.1.3.3. A propos de la collecte de données sur le maillage des services

Vous pouvez utiliser la commande CLI **oc adm must-gather** pour collecter des informations sur votre cluster, y compris les fonctionnalités et les objets associés à Red Hat OpenShift Service Mesh.

Conditions préalables

- Accès au cluster en tant qu'utilisateur ayant le rôle **cluster-admin**.
- L'OpenShift Container Platform CLI (**oc**) est installé.

Précédent

1. Pour collecter les données Red Hat OpenShift Service Mesh avec **must-gather**, vous devez spécifier l'image Red Hat OpenShift Service Mesh.

```
$ oc adm must-gather --image=registry.redhat.io/openshift-service-mesh/istio-must-gather-rhel8:2.3
```

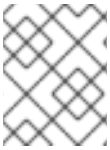
2. Pour collecter les données Red Hat OpenShift Service Mesh pour un espace de noms de plan de contrôle Service Mesh spécifique avec **must-gather**, vous devez spécifier l'image Red Hat OpenShift Service Mesh et l'espace de noms. Dans cet exemple, remplacez **<namespace>** par votre espace de noms de plan de contrôle Service Mesh, tel que **istio-system**.

```
$ oc adm must-gather --image=registry.redhat.io/openshift-service-mesh/istio-must-gather-rhel8:2.3 gather <namespace>
```

2.1.4. Configurations prises en charge par Red Hat OpenShift Service Mesh

Les configurations suivantes sont les seules prises en charge pour le Red Hat OpenShift Service Mesh :

- OpenShift Container Platform version 4.6 ou ultérieure.



NOTE

OpenShift Online et Red Hat OpenShift Dedicated ne sont pas pris en charge pour Red Hat OpenShift Service Mesh.

- Le déploiement doit être contenu dans un seul cluster OpenShift Container Platform qui n'est pas fédéré.
- Cette version de Red Hat OpenShift Service Mesh est uniquement disponible sur OpenShift Container Platform x86_64.
- Cette version ne prend en charge que les configurations dans lesquelles tous les composants Service Mesh sont contenus dans le cluster OpenShift Container Platform dans lequel il fonctionne. Elle ne prend pas en charge la gestion des microservices qui résident en dehors du cluster, ou dans un scénario multi-clusters.
- Cette version ne prend en charge que les configurations qui n'intègrent pas de services externes tels que des machines virtuelles.

Pour plus d'informations sur le cycle de vie de Red Hat OpenShift Service Mesh et les configurations prises en charge, reportez-vous à la Politique d'assistance.

2.1.4.1. Configurations prises en charge pour Kiali sur Red Hat OpenShift Service Mesh

- La console d'observabilité Kiali n'est compatible qu'avec les deux versions les plus récentes des navigateurs Chrome, Edge, Firefox ou Safari.

2.1.4.2. Adaptateurs de mélangeur pris en charge

- Cette version ne prend en charge que l'adaptateur Mixer suivant :
  - adaptateur 3scale Istio

2.1.5. Nouvelles fonctionnalités

Red Hat OpenShift Service Mesh fournit un certain nombre de capacités clés de manière uniforme à travers un réseau de services :

- **Traffic Management** - Contrôler le flux de trafic et les appels API entre les services, rendre les appels plus fiables et rendre le réseau plus robuste face à des conditions défavorables.
- **Service Identity and Security**- Fournir aux services du réseau maillé une identité vérifiable et offrir la possibilité de protéger le trafic des services lorsqu'il circule sur des réseaux plus ou moins fiables.
- **Policy Enforcement** - Appliquer la politique de l'organisation à l'interaction entre les services, veiller à ce que les politiques d'accès soient appliquées et que les ressources soient équitablement réparties entre les consommateurs. Les changements de politique se font en configurant le maillage, et non en modifiant le code de l'application.

- **Telemetry** - Comprendre les dépendances entre les services ainsi que la nature et le flux du trafic entre eux, ce qui permet d'identifier rapidement les problèmes.

2.1.5.1. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.18.2

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE).

2.1.5.1.1. Versions de composants incluses dans la version 1.1.18.2 de Red Hat OpenShift Service Mesh

Composant	Version
Istio	1.4.10
Jaeger	1.30.2
Kiali	1.12.21.1
adaptateur 3scale Istio	1.0.0

2.1.5.2. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.18.1

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE).

2.1.5.2.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 1.1.18.1

Composant	Version
Istio	1.4.10
Jaeger	1.30.2
Kiali	1.12.20.1
adaptateur 3scale Istio	1.0.0

2.1.5.3. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.18

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE).

2.1.5.3.1. Versions de composants incluses dans Red Hat OpenShift Service Mesh version 1.1.18

Composant	Version
Istio	1.4.10

Composant	Version
Jaeger	1.24.0
Kiali	1.12.18
adaptateur 3scale Istio	1.0.0

2.1.5.4. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.17.1

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE).

2.1.5.4.1. Changement dans la façon dont Red Hat OpenShift Service Mesh gère les fragments d'URI

Red Hat OpenShift Service Mesh contient une vulnérabilité exploitable à distance, CVE-2021-39156, où une requête HTTP avec un fragment (une section à la fin d'un URI qui commence par un caractère #) dans le chemin URI pourrait contourner les politiques d'autorisation basées sur le chemin URI d'Istio. Par exemple, une politique d'autorisation d'Istio refuse les requêtes envoyées au chemin URI **/user/profile**. Dans les versions vulnérables, une requête avec le chemin URI **/user/profile#section1** contourne la politique de refus et se dirige vers le backend (avec l'URI normalisé **path /user/profile#section1**), ce qui peut conduire à un incident de sécurité.

Vous êtes concerné par cette vulnérabilité si vous utilisez des politiques d'autorisation avec des actions DENY et **operation.paths**, ou des actions ALLOW et **operation.notPaths**.

Avec l'atténuation, la partie fragmentaire de l'URI de la demande est supprimée avant l'autorisation et l'acheminement. Cela empêche une demande contenant un fragment dans son URI de contourner les politiques d'autorisation qui sont basées sur l'URI sans la partie fragment.

2.1.5.4.2. Mise à jour requise pour les politiques d'autorisation

Istio génère des noms d'hôtes à la fois pour le nom d'hôte lui-même et pour tous les ports correspondants. Par exemple, un service virtuel ou une passerelle pour un hôte de "httpbin.foo" génère une configuration correspondant à "httpbin.foo et httpbin.foo:*". Cependant, les politiques d'autorisation de correspondance exacte ne correspondent qu'à la chaîne exacte donnée pour les champs **hosts** ou **notHosts**.

Votre cluster est affecté si vous avez des ressources **AuthorizationPolicy** utilisant la comparaison de chaînes exactes pour la règle de détermination des hosts ou notHosts.

Vous devez mettre à jour les règles de votre politique d'autorisation pour utiliser la correspondance des préfixes au lieu de la correspondance exacte. Par exemple, en remplaçant **hosts: ["httpbin.com"]** par **hosts: ["httpbin.com:*"]** dans le premier exemple **AuthorizationPolicy**.

Premier exemple de politique d'autorisation utilisant la correspondance des préfixes

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: httpbin
  namespace: foo
spec:
```

```
  action: DENY
  rules:
  - from:
    - source:
        namespaces: ["dev"]
    to:
    - operation:
        hosts: [“httpbin.com”,"httpbin.com:*"]
```

Deuxième exemple de politique d'autorisation utilisant la correspondance des préfixes

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

2.1.5.5. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.17

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

2.1.5.6. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.16

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

2.1.5.7. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.15

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

2.1.5.8. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.14

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

IMPORTANT

Des étapes manuelles doivent être effectuées pour résoudre les problèmes CVE-2021-29492 et CVE-2021-31920.

2.1.5.8.1. Mises à jour manuelles requises par CVE-2021-29492 et CVE-2021-31920

Istio contient une vulnérabilité exploitable à distance où un chemin de requête HTTP avec plusieurs barres obliques ou des caractères obliques échappés (/` **or** \`) pourrait potentiellement contourner une politique d'autorisation Istio lorsque des règles d'autorisation basées sur le chemin sont utilisées.

Par exemple, supposons qu'un administrateur de cluster Istio définisse une politique d'autorisation DENY pour rejeter la demande au chemin **/admin**. Une demande envoyée au chemin URL **//admin** ne sera PAS rejetée par la politique d'autorisation.

Selon la RFC 3986, le chemin **//admin** avec plusieurs barres obliques devrait techniquement être traité comme un chemin différent du chemin **/admin**. Cependant, certains services backend choisissent de normaliser les chemins URL en fusionnant les barres obliques multiples en une seule barre oblique. Cela peut entraîner un contournement de la politique d'autorisation (**//admin** ne correspond pas à **/admin**), et un utilisateur peut accéder à la ressource au chemin **/admin** dans le backend ; cela représenterait un incident de sécurité.

Votre cluster est concerné par cette vulnérabilité si vous avez des politiques d'autorisation qui utilisent les modèles **ALLOW action notPaths** field ou **DENY action paths field**. Ces modèles sont vulnérables à des contournements inattendus de la politique.

Votre cluster n'est PAS affecté par cette vulnérabilité si :

- Vous n'avez pas de politique d'autorisation.
- Vos politiques d'autorisation ne définissent pas les champs **paths** ou **notPaths**.
- Vos politiques d'autorisation utilisent des modèles de champ **ALLOW action paths** ou **DENY action notPaths**. Ces motifs pourraient uniquement entraîner un rejet inattendu au lieu d'un contournement de la politique. La mise à niveau est facultative dans ces cas.

NOTE

L'emplacement de configuration de Red Hat OpenShift Service Mesh pour la normalisation des chemins est différent de la configuration d'Istio.

2.1.5.8.2. Mise à jour de la configuration de la normalisation des chemins

Les politiques d'autorisation d'Istio peuvent être basées sur les chemins d'URL dans la requête HTTP. La normalisation des chemins, également connue sous le nom de normalisation des URI, modifie et normalise les chemins des requêtes entrantes afin que les chemins normalisés puissent être traités de manière standard. Des chemins syntaxiquement différents peuvent être équivalents après la normalisation du chemin.

Istio prend en charge les schémas de normalisation suivants sur les chemins de requête avant de les évaluer par rapport aux politiques d'autorisation et d'acheminer les requêtes :

Tableau 2.1. Schémas de normalisation

Option	Description	Exemple :	Notes
NONE	Aucune normalisation n'est effectuée. Tout ce qui est reçu par Envoy est transmis tel quel à un service de backend.	**..//a../b** est évaluée par les politiques d'autorisation et envoyée à votre service.	Ce paramètre est vulnérable à la CVE-2021-31920.

Option	Description	Exemple :	Notes
BASE	C'est actuellement l'option utilisée dans l'installation d'Istio sur **default**. Elle applique l'option **normalize_path** sur les proxies Envoy, qui suit la RFC 3986 avec une normalisation supplémentaire pour convertir les barres obliques inverses en barres obliques inverses.	**/a/../b** est normalisé à **/b**. **\da** est normalisé à **/da**.	Ce paramètre est vulnérable à la CVE-2021-31920.
MERGE_SLASHES	Les barres obliques sont fusionnées après la normalisation *BASE*.	**/a//b** est normalisé à **/a/b**.	La mise à jour de ce paramètre permet d'atténuer la CVE-2021-31920.
DECODE_AND_MERGE_SLASHES	Le paramètre le plus strict lorsque vous autorisez tout le trafic par défaut. Ce paramètre est recommandé, à condition que vous testiez minutieusement les itinéraires de vos politiques d'autorisation. Les caractères barre oblique et barre oblique inverse codés en pourcentage (/, /, \ et \) sont décodés en / ou \, avant la normalisation **MERGE_SLASHES**.	**/a/b** est normalisé à **/a/b**.	Mettez à jour ce paramètre pour atténuer la CVE-2021-31920. Ce paramètre est plus sûr, mais il est également susceptible d'endommager les applications. Testez vos applications avant de les déployer en production.

Les algorithmes de normalisation sont exécutés dans l'ordre suivant :

1. Décodez en pourcentage /, /, \ et \.
2. La normalisation RFC 3986 et d'autres normalisations implémentées par l'option **normalize_path** dans Envoy.
3. Fusionner les barres obliques.

AVERTISSEMENT

Bien que ces options de normalisation représentent des recommandations issues des normes HTTP et des pratiques industrielles courantes, les applications peuvent interpréter une URL comme elles l'entendent. Lorsque vous utilisez des politiques de déni, assurez-vous de bien comprendre le comportement de votre application.

2.1.5.8.3. Exemples de configuration de la normalisation des chemins

S'assurer qu'Envoy normalise les chemins d'accès aux requêtes pour qu'ils correspondent aux attentes de vos services d'arrière-plan est essentiel pour la sécurité de votre système. Les exemples suivants peuvent vous servir de référence pour configurer votre système. Les chemins d'URL normalisés, ou les chemins d'URL originaux si **NONE** est sélectionné, seront les suivants :

1. Utilisé pour vérifier les politiques d'autorisation.
2. Transmis à l'application dorsale.

Tableau 2.2. Exemples de configuration

Si votre candidature..	Choisissez..
S'appuie sur le proxy pour effectuer la normalisation	**BASE**, **MERGE_SLASHES** ou **DECODE_AND_MERGE_SLASHES**
Normalise les chemins d'accès aux requêtes sur la base de la RFC 3986 et ne fusionne pas les barres obliques.	**BASE**
Normalise les chemins d'accès aux requêtes sur la base de la RFC 3986 et fusionne les barres obliques, mais ne décode pas les barres obliques codées en pourcentage.	**MERGE_SLASHES**
Normalise les chemins de requête sur la base de la RFC 3986, décode les barres obliquescodées en pourcentage et fusionne les barres obliques.	**DECODE_AND_MERGE_SLASHES**
Traite les chemins de requête d'une manière incompatible avec la RFC 3986.	**NONE**

2.1.5.8.4. Configuration de votre SMCP pour la normalisation des chemins d'accès

Pour configurer la normalisation des chemins pour Red Hat OpenShift Service Mesh, spécifiez ce qui suit dans votre **ServiceMeshControlPlane**. Utilisez les exemples de configuration pour vous aider à déterminer les paramètres de votre système.

SMCP v1 pathNormalization

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

2.1.5.9. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.13

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

2.1.5.10. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.12

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

2.1.5.11. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.11

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

2.1.5.12. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.10

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

2.1.5.13. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.9

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

2.1.5.14. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.8

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

2.1.5.15. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.7

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

2.1.5.16. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.6

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

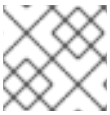
2.1.5.17. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.5

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

Cette version a également ajouté la prise en charge de la configuration des suites de chiffrement.

2.1.5.18. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.4

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

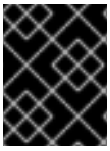


NOTE

Il y a des étapes manuelles qui doivent être complétées pour traiter CVE-2020-8663.

2.1.5.18.1. Mises à jour manuelles requises par CVE-2020-8663

Le correctif pour CVE-2020-8663**: envoy: Resource exhaustion when accepting too many connections** a ajouté une limite configurable pour les connexions en aval. L'option de configuration de cette limite doit être configurée pour atténuer cette vulnérabilité.



IMPORTANT

Ces étapes manuelles sont nécessaires pour atténuer cette CVE, que vous utilisiez la version 1.1 ou la version 1.0 de Red Hat OpenShift Service Mesh.

Cette nouvelle option de configuration s'appelle **overload.global_downstream_max_connections** et peut être configurée en tant que paramètre de proxy **runtime**. Effectuez les étapes suivantes pour configurer les limites au niveau de la passerelle d'entrée.

Procédure

1. Créez un fichier nommé **bootstrap-override.json** avec le texte suivant pour forcer le proxy à remplacer le modèle de démarrage et à charger la configuration d'exécution à partir du disque :

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

2. Créez un secret à partir du fichier **bootstrap-override.json**, en remplaçant <SMCPnamespace> par l'espace de noms dans lequel vous avez créé le plan de contrôle du maillage de services (SMCP) :

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

3. Mettez à jour la configuration du SMCP pour activer la dérogation.

Mise à jour de l'exemple de configuration SMCP n°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. Pour définir la nouvelle option de configuration, créez un secret ayant la valeur souhaitée pour le paramètre **overload.global_downstream_max_connections**. L'exemple suivant utilise la valeur **10000**:

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

5. Mettez à nouveau à jour le SMCP pour monter le secret à l'endroit où Envoy recherche la configuration d'exécution :

Mise à jour de l'exemple de configuration SMCP n°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
```

2.1.5.18.2. Mise à jour d'Elasticsearch 5 vers Elasticsearch 6

Lors de la mise à jour d'Elasticsearch 5 vers Elasticsearch 6, vous devez supprimer votre instance Jaeger, puis recréer l'instance Jaeger en raison d'un problème avec les certificats. La recréation de l'instance Jaeger entraîne la création d'un nouveau jeu de certificats. Si vous utilisez un stockage persistant, les mêmes volumes peuvent être montés pour la nouvelle instance Jaeger tant que le nom Jaeger et l'espace de noms de la nouvelle instance Jaeger sont les mêmes que ceux de l'instance Jaeger supprimée.

Procédure si Jaeger est installé dans le cadre de Red Hat Service Mesh

1. Déterminez le nom de votre fichier de ressources personnalisées Jaeger :

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

Vous devriez voir quelque chose comme ce qui suit :

```
NAME     AGE
jaeger   3d21h
```

2. Copiez le fichier de ressources personnalisées généré dans un répertoire temporaire :

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

3. Supprimer l'instance de Jaeger :

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

4. Recréez l'instance de Jaeger à partir de votre copie du fichier de ressources personnalisé :

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

5. Supprimer la copie du fichier de ressources personnalisées généré :

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

Procédure si Jaeger n'est pas installé dans le cadre du Service Mesh de Red Hat

Avant de commencer, créez une copie de votre fichier de ressources personnalisées Jaeger.

1. Supprimez l'instance de Jaeger en supprimant le fichier de ressources personnalisé :

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

Par exemple :

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

2. Recréez votre instance Jaeger à partir de la copie de sauvegarde de votre fichier de ressources personnalisé :

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

3. Validez que vos Pods ont redémarré :

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

2.1.5.19. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.3

Cette version de Red Hat OpenShift Service Mesh traite des vulnérabilités et expositions communes (CVE) et des corrections de bogues.

2.1.5.20. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.2

Cette version de Red Hat OpenShift Service Mesh corrige une vulnérabilité de sécurité.

2.1.5.21. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.1

Cette version de Red Hat OpenShift Service Mesh ajoute la prise en charge d'une installation déconnectée.

2.1.5.22. Nouvelles fonctionnalités Red Hat OpenShift Service Mesh 1.1.0

Cette version de Red Hat OpenShift Service Mesh ajoute la prise en charge d'Istio 1.4.6 et de Jaeger 1.17.1.

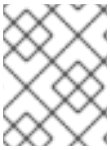
2.1.5.22.1. Mises à jour manuelles de 1.0 à 1.1

Si vous mettez à jour Red Hat OpenShift Service Mesh 1.0 vers 1.1, vous devez mettre à jour la ressource **ServiceMeshControlPlane** pour mettre à jour les composants du plan de contrôle vers la nouvelle version.

1. Dans la console web, cliquez sur l'opérateur Red Hat OpenShift Service Mesh.
2. Cliquez sur le menu **Project** et choisissez dans la liste le projet dans lequel votre **ServiceMeshControlPlane** est déployé, par exemple **istio-system**.
3. Cliquez sur le nom de votre plan de contrôle, par exemple **basic-install**.
4. Cliquez sur YAML et ajoutez un champ de version à l'adresse **spec:** de votre ressource **ServiceMeshControlPlane**. Par exemple, pour mettre à jour vers Red Hat OpenShift Service Mesh 1.1.0, ajoutez **version: v1.1**.

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

Le champ "version" indique la version de Service Mesh à installer et prend par défaut la dernière version disponible.



NOTE

Notez que la prise en charge de Red Hat OpenShift Service Mesh v1.0 a pris fin en octobre 2020. Vous devez effectuer une mise à niveau vers v1.1 ou v2.0.

2.1.6. Fonctionnalités obsolètes

Certaines fonctionnalités disponibles dans les versions précédentes sont devenues obsolètes ou ont été supprimées.

Les fonctionnalités dépréciées sont toujours incluses dans OpenShift Container Platform et continuent d'être prises en charge ; cependant, elles seront supprimées dans une prochaine version de ce produit et ne sont pas recommandées pour les nouveaux déploiements.

2.1.6.1. Fonctionnalités obsolètes Red Hat OpenShift Service Mesh 1.1.5

Les ressources personnalisées suivantes étaient obsolètes dans la version 1.1.5 et ont été supprimées dans la version 1.1.12

- **Policy** - La ressource **Policy** est obsolète et sera remplacée par la ressource **PeerAuthentication** dans une prochaine version.

- **MeshPolicy** - La ressource **MeshPolicy** est obsolète et sera remplacée par la ressource **PeerAuthentication** dans une prochaine version.
- **v1alpha1** API RBAC - La politique RBAC v1alpha1 est dépassée par la politique v1beta1 **AuthorizationPolicy**. RBAC (Role Based Access Control) définit les objets **ServiceRole** et **ServiceRoleBinding**.
  - **ServiceRole**
  - **ServiceRoleBinding**
- **RbacConfig** - **RbacConfig** met en œuvre la définition de ressource personnalisée pour contrôler le comportement RBAC d'Istio.
  - **ClusterRbacConfig**(versions antérieures à Red Hat OpenShift Service Mesh 1.0)
  - **ServiceMeshRbacConfig** (Red Hat OpenShift Service Mesh version 1.0 et ultérieure)
- Dans Kiali, les stratégies **login** et **LDAP** sont obsolètes. Une prochaine version introduira l'authentification à l'aide des fournisseurs OpenID.

Les composants suivants sont également obsolètes dans cette version et seront remplacés par le composant **Istiod** dans une version ultérieure.

- **Mixer** - le contrôle d'accès et les politiques d'utilisation
- **Pilot** - découverte des services et configuration du proxy
- **Citadel** - génération de certificats
- **Galley** - validation et distribution de la configuration

2.1.7. Problèmes connus

Ces limitations existent dans Red Hat OpenShift Service Mesh :

- Red Hat OpenShift Service Mesh ne prend pas en charge IPv6 , car il n'est pas pris en charge par le projet Istio en amont, ni entièrement pris en charge par OpenShift Container Platform.
- Disposition du graphe - La disposition du graphe Kiali peut être différente, selon l'architecture de votre application et les données à afficher (nombre de nœuds du graphe et leurs interactions). Parce qu'il est difficile, voire impossible, de créer une présentation unique qui rende bien dans toutes les situations, Kiali offre un choix de plusieurs présentations différentes. Pour choisir une autre présentation, vous pouvez choisir une autre **Layout Schema** dans le menu **Graph Settings**.
- La première fois que vous accédez à des services connexes tels que Jaeger et Grafana, à partir de la console Kiali, vous devez accepter le certificat et vous authentifier à nouveau en utilisant vos identifiants de connexion à OpenShift Container Platform. Ceci est dû à un problème avec la façon dont le framework affiche les pages intégrées dans la console.

2.1.7.1. Problèmes connus du Service Mesh

Il s'agit des problèmes connus dans Red Hat OpenShift Service Mesh :

- Mise à niveau de l'opérateur Jaeger/Kiali bloquée avec un opérateur en attente Lors de la mise à niveau des opérateurs Jaeger ou Kiali avec le Service Mesh 1.0.x installé, le statut de l'opérateur est indiqué comme étant en attente.
  Solution de contournement : Voir l'article de la base de connaissances pour plus d'informations.

- Istio-14743 En raison de limitations dans la version d'Istio sur laquelle cette version de Red Hat OpenShift Service Mesh est basée, il y a plusieurs applications qui sont actuellement incompatibles avec Service Mesh. Voir le problème communautaire lié pour plus de détails.

- MAISTRA-858 Les messages de journal Envoy suivants décrivant les options et configurations obsolètes associées à Istio 1.1.x sont attendus :

  - [2019-06-03 07:03:28.943][19][warning][misc] [external/envoy/source/common/protobuf/utility.cc:129] Utilisation de l'option obsolète 'envoy.api.v2.listener.Filter.config'. Cette configuration sera bientôt supprimée d'Envoy.

  - [2019-08-12 22:12:59.001][13][warning][misc] [external/envoy/source/common/protobuf/utility.cc:174] Utilisation de l'option obsolète 'envoy.api.v2.Listener.use_original_dst' du fichier lds.proto. Cette configuration sera bientôt supprimée d'Envoy.

- MAISTRA-806 L'éviction du pod opérateur Istio entraîne le non-déploiement du maillage et de la CNI.
  Solution de contournement : Si le pod **istio-operator** est expulsé lors du déploiement du volet de contrôle, supprimez le pod **istio-operator** expulsé.

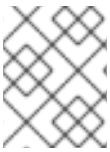
- MAISTRA-681 Lorsque le plan de contrôle comporte de nombreux espaces de noms, il peut en résulter des problèmes de performance.

- MAISTRA-465 L'opérateur Maistra ne parvient pas à créer un service pour les métriques de l'opérateur.

- MAISTRA-453 Si vous créez un nouveau projet et déployez des modules immédiatement, l'injection de sidecar ne se produit pas. L'opérateur ne parvient pas à ajouter le site **maistra.io/member-of** avant la création des nacelles. Les nacelles doivent donc être supprimées et recréées pour que l'injection de sidecar se produise.

- MAISTRA-158 L'application de plusieurs passerelles référençant le même nom d'hôte entraîne l'arrêt du fonctionnement de toutes les passerelles.

2.1.7.2. Problèmes connus de Kiali



NOTE

Les nouveaux problèmes pour Kiali doivent être créés dans le projet OpenShift Service Mesh avec l'adresse **Component** fixée à **Kiali**.

Voici les problèmes connus de Kiali :

- KIALI-2206 Lorsque vous accédez à la console Kiali pour la première fois, et qu'il n'y a pas de données de navigation mises en cache pour Kiali, le lien "View in Grafana" sur l'onglet Metrics de la page Kiali Service Details redirige vers le mauvais emplacement. La seule façon de rencontrer ce problème est d'accéder à Kiali pour la première fois.

- KIALI-507 Kiali ne supporte pas Internet Explorer 11. Ceci est dû au fait que les frameworks sous-jacents ne supportent pas Internet Explorer. Pour accéder à la console Kiali, utilisez l'une des deux versions les plus récentes des navigateurs Chrome, Edge, Firefox ou Safari.

2.1.7.3. Red Hat OpenShift distributed tracing known issues (problèmes connus)

Ces limitations existent dans le traçage distribué de Red Hat OpenShift :

- Apache Spark n'est pas pris en charge.
- Le déploiement en continu via AMQ/Kafka n'est pas pris en charge sur les systèmes IBM Z et IBM Power.

Ce sont les problèmes connus pour Red Hat OpenShift distributed tracing :

- OBSDA-220 Dans certains cas, si vous essayez d'extraire une image à l'aide de la collecte de données de traçage distribuées, l'extraction de l'image échoue et un message d'erreur **Failed to pull image** s'affiche. Il n'y a pas de solution à ce problème.
- TRACING-2057 L'API Kafka a été mise à jour sur **v1beta2** pour prendre en charge le Strimzi Kafka Operator 0.23.0. Cependant, cette version de l'API n'est pas prise en charge par AMQ Streams 1.6.3. Si vous avez l'environnement suivant, vos services Jaeger ne seront pas mis à niveau et vous ne pourrez pas créer de nouveaux services Jaeger ou modifier des services Jaeger existants :
  - Canal de l'opérateur Jaeger : **1.17.x stable** ou **1.20.x stable**
  - Canal de l'opérateur AMQ Streams : **amq-streams-1.6.x**
    Pour résoudre ce problème, changez le canal d'abonnement de votre opérateur AMQ Streams sur **amq-streams-1.7.x** ou **stable**.

2.1.8. Problèmes corrigés

Les problèmes suivants ont été résolus dans la version actuelle :

2.1.8.1. Le maillage de service a permis de résoudre des problèmes

- MAISTRA-2371 Gestion des pierres tombales dans listerInformer. Le code de base du cache mis à jour ne gérait pas les pierres tombales lors de la traduction des événements des caches d'espace de noms vers le cache agrégé, ce qui entraînait une panique dans la routine go.
- OSSM-542 Galley n'utilise pas le nouveau certificat après la rotation.
- OSSM-99 Les charges de travail générées à partir d'un pod direct sans étiquette peuvent faire planter Kiali.
- OSSM-93 IstioConfigList ne peut pas filtrer par deux noms ou plus.
- OSSM-92 L'annulation de modifications non enregistrées sur la page d'édition YAML VS/DR n'annule pas les modifications.
- OSSM-90 Traces non disponibles sur la page de détails du service.
- MAISTRA-1649 Conflit entre les services sans tête dans différents espaces de noms. Lors du déploiement de services sans tête dans différents espaces de noms, la configuration des points d'extrémité est fusionnée et des configurations Envoy invalides sont poussées vers les sidecars.

- MAISTRA-1541 Panique dans kubernetesenv lorsque le contrôleur n'est pas défini sur la référence propriétaire. Si un pod a une ownerReference qui ne spécifie pas le contrôleur, cela provoque une panique dans le code **kubernetesenv cache.go**.

- MAISTRA-1352 Les définitions de ressources personnalisées (CRD) de Cert-manager de l'installation du plan de contrôle ont été supprimées pour cette version et les versions futures. Si vous avez déjà installé Red Hat OpenShift Service Mesh, les CRD doivent être supprimés manuellement si cert-manager n'est pas utilisé.

- MAISTRA-1001 La fermeture de connexions HTTP/2 peut entraîner des erreurs de segmentation dans **istio-proxy**.

- MAISTRA-932 Ajout des métadonnées **requires** pour ajouter une relation de dépendance entre Jaeger Operator et OpenShift Elasticsearch Operator. Assure que lorsque l'opérateur Jaeger est installé, il déploie automatiquement l'opérateur OpenShift Elasticsearch s'il n'est pas disponible.

- MAISTRA-862 Galley a abandonné les montres et a cessé de fournir des configurations aux autres composants après de nombreuses suppressions et recréations d'espaces de noms.

- MAISTRA-833 Pilot a cessé de fournir la configuration après de nombreuses suppressions et recréations d'espaces de noms.

- MAISTRA-684 La version par défaut de Jaeger dans le site **istio-operator** est 1.12.0, ce qui ne correspond pas à la version 1.13.1 de Jaeger livrée dans Red Hat OpenShift Service Mesh 0.12.TechPreview.

- MAISTRA-622 Dans Maistra 0.12.0/TP12, le mode permissif ne fonctionne pas. L'utilisateur a la possibilité d'utiliser le mode texte simple ou le mode TLS mutuel, mais pas le mode permissif.

- MAISTRA-572 Jaeger ne peut pas être utilisé avec Kiali. Dans cette version, Jaeger est configuré pour utiliser le proxy OAuth, mais il est également configuré pour fonctionner uniquement via un navigateur et n'autorise pas l'accès aux services. Kiali ne peut pas communiquer correctement avec le point de terminaison Jaeger et considère que Jaeger est désactivé. Voir aussi TRACING-591.

- MAISTRA-357 Dans OpenShift 4 Beta sur AWS, il n'est pas possible, par défaut, d'accéder à un service TCP ou HTTPS via la passerelle d'entrée sur un port autre que le port 80. L'équilibreur de charge AWS dispose d'un contrôle de santé qui vérifie si le port 80 sur le point de terminaison du service est actif. Si aucun service n'est exécuté sur le port 80, le contrôle de santé de l'équilibreur de charge échoue.

- MAISTRA-348 OpenShift 4 Beta sur AWS ne prend pas en charge le trafic de la passerelle d'entrée sur des ports autres que 80 ou 443. Si vous configurez votre passerelle d'entrée pour gérer le trafic TCP avec un numéro de port autre que 80 ou 443, vous devez utiliser le nom d'hôte du service fourni par l'équilibreur de charge AWS plutôt que le routeur OpenShift comme solution de contournement.

- MAISTRA-193 Des messages d'information inattendus de la console sont visibles lorsque la vérification de l'état de santé est activée pour citadel.

- Bug 1821432 Les contrôles à bascule dans la page de détails des ressources de contrôle d'OpenShift Container Platform ne mettent pas à jour le CR correctement. Les contrôles à bascule de l'interface utilisateur dans la page de présentation du Service Mesh Control Plane (SMCP) dans la console web d'OpenShift Container Platform mettent parfois à jour le mauvais

champ de la ressource. Pour mettre à jour une ressource ServiceMeshControlPlane, modifiez le contenu YAML directement ou mettez à jour la ressource à partir de la ligne de commande au lieu de cliquer sur les contrôles à bascule.

2.1.8.2. Kiali a corrigé des problèmes

- KIALI-3239 Si un pod Kiali Operator a échoué avec un statut "Evicted", cela bloque le déploiement de l'opérateur Kiali. La solution consiste à supprimer le pod évincé et à redéployer l'opérateur Kiali.
- KIALI-3118 Après avoir modifié le ServiceMeshMemberRoll, par exemple en ajoutant ou en supprimant des projets, le pod Kiali redémarre et affiche des erreurs sur la page Graph pendant le redémarrage du pod Kiali.
- KIALI-3096 Les métriques d'exécution échouent dans le Service Mesh. Il existe un filtre OAuth entre le Service Mesh et Prometheus, qui exige qu'un jeton de porteur soit transmis à Prometheus avant que l'accès ne soit accordé. Kiali a été mis à jour pour utiliser ce jeton lors de la communication avec le serveur Prometheus, mais les métriques de l'application échouent actuellement avec des erreurs 403.
- KIALI-3070 Ce bogue n'affecte que les tableaux de bord personnalisés, et non les tableaux de bord par défaut. Lorsque vous sélectionnez des étiquettes dans les paramètres de mesure et que vous actualisez la page, vos sélections sont conservées dans le menu, mais elles ne s'affichent pas dans les graphiques.
- KIALI-2686 Lorsque le plan de contrôle a de nombreux espaces de noms, cela peut entraîner des problèmes de performance.

2.1.8.3. Red Hat OpenShift a corrigé des problèmes de traçage distribué

- OSSM-1910 En raison d'un problème introduit dans la version 2.6, les connexions TLS ne pouvaient pas être établies avec OpenShift Container Platform Service Mesh. Cette mise à jour résout le problème en changeant les noms des ports de service pour qu'ils correspondent aux conventions utilisées par OpenShift Container Platform Service Mesh et Istio.
- OBSDA-208 Avant cette mise à jour, les limites de ressources par défaut de 200 m de CPU et 256 m de mémoire pouvaient entraîner un redémarrage continu de la collecte des données de traçage distribué sur les grands clusters. Cette mise à jour résout le problème en supprimant ces limites de ressources.
- OBSDA-222 Avant cette mise à jour, des spans pouvaient être abandonnés dans la plateforme de traçage distribuée OpenShift Container Platform. Pour aider à prévenir ce problème, cette version met à jour les dépendances de version.
- TRACING-2337 Jaeger enregistre un message d'avertissement répétitif dans les journaux de Jaeger, similaire à ce qui suit :

```
{"level":"warn","ts":1642438880.918793,"caller":"channelz/logging.go:62","msg":"[core]grpc: Server.Serve failed to create ServerTransport: connection error: desc = \"transport: http2Server.HandleStreams received bogus greeting from client: \\\"\\\\x16\\\\x03\\\\x01\\\\x02\\\\x00\\\\x01\\\\x00\\\\x01\\\\xfc\\\\x03\\\\x03vw\\\\x1a\\\\xc9T\\\\xe7\\\\xdaCj\\\\xb7\\\\x8dK\\\\xa6\\\"\"","system":"grpc","grpc_log":true}
```

Ce problème a été résolu en exposant uniquement le port HTTP(S) du service de requête, et non le port gRPC.

- TRACING-2009 L'opérateur Jaeger a été mis à jour pour inclure le support de l'opérateur Strimzi Kafka 0.23.0.

- TRACING-1907 L'injection du sidecar de l'agent Jaeger échouait en raison de l'absence de cartes de configuration dans l'espace de noms de l'application. Les cartes de configuration étaient automatiquement supprimées en raison d'un paramètre de champ **OwnerReference** incorrect et, par conséquent, les pods d'application ne dépassaient pas l'étape "ContainerCreating". Les paramètres incorrects ont été supprimés.

- TRACING-1725 Suivi de TRACING-1631. Correction supplémentaire pour s'assurer que les certificats Elasticsearch sont correctement réconciliés lorsqu'il y a plusieurs instances de production Jaeger, utilisant le même nom mais dans des espaces de noms différents. Voir aussi BZ-1918920.

- TRACING-1631 Plusieurs instances de production Jaeger, utilisant le même nom mais dans des espaces de noms différents, causant un problème de certificat Elasticsearch. Lorsque plusieurs maillages de services étaient installés, toutes les instances Elasticsearch de Jaeger avaient le même secret Elasticsearch au lieu de secrets individuels, ce qui empêchait l'opérateur Elasticsearch d'OpenShift de communiquer avec tous les clusters Elasticsearch.

- TRACING-1300 Échec de la connexion entre l'agent et le collecteur lors de l'utilisation d'Istio sidecar. Une mise à jour de l'opérateur Jaeger a activé la communication TLS par défaut entre un agent Jaeger sidecar et le collecteur Jaeger.

- TRACING-1208 Authentification "500 Internal Error" lors de l'accès à l'interface utilisateur de Jaeger. Lorsque j'essaie de m'authentifier à l'interface utilisateur en utilisant OAuth, j'obtiens une erreur 500 parce que le sidecar oauth-proxy ne fait pas confiance à l'ensemble d'autorités de certification personnalisées défini lors de l'installation avec l'adresse **additionalTrustBundle**.

- TRACING-1166 Il n'est actuellement pas possible d'utiliser la stratégie de streaming Jaeger dans un environnement déconnecté. Lorsqu'un cluster Kafka est en cours de provisionnement, il en résulte une erreur : **Failed to pull image registry.redhat.io/amq7/amq-streams-kafka-24-rhel7@sha256:f9ceca004f1b7dccb3b82d9a8027961f9fe4104e0ed69752c0bdd8078b4a1076**.

- TRACING-809 Jaeger Ingester est incompatible avec Kafka 2.3. Lorsqu'il y a deux ou plusieurs instances de Jaeger Ingester et un trafic suffisant, il génère continuellement des messages de rééquilibrage dans les journaux. Ceci est dû à une régression dans Kafka 2.3 qui a été corrigée dans Kafka 2.3.1. Pour plus d'informations, voir Jaegertracing-1819.

- BZ-1918920/LOG-1619 Les pods Elasticsearch ne sont pas redémarrés automatiquement après une mise à jour.
  Solution : Redémarrer les pods manuellement.

2.2. COMPRENDRE LE MAILLAGE DE SERVICES

AVERTISSEMENT

You are viewing documentation for a Red Hat OpenShift Service Mesh release that is no longer supported.

Les plans de contrôle Service Mesh version 1.0 et 1.1 ne sont plus pris en charge. Pour plus d'informations sur la mise à niveau de votre plan de contrôle Service Mesh, voir Mise à niveau de Service Mesh .

Pour plus d'informations sur l'état de l'assistance d'une version particulière de Red Hat OpenShift Service Mesh, consultez la page Cycle de vie du produit .

Red Hat OpenShift Service Mesh fournit une plateforme pour une vision comportementale et un contrôle opérationnel sur vos microservices en réseau dans un maillage de services. Avec Red Hat OpenShift Service Mesh, vous pouvez connecter, sécuriser et surveiller les microservices dans votre environnement OpenShift Container Platform.

2.2.1. Comprendre le maillage des services

Un *service mesh* est le réseau de microservices qui composent les applications dans une architecture distribuée de microservices et les interactions entre ces microservices. Lorsqu'un Service Mesh augmente en taille et en complexité, il peut devenir plus difficile à comprendre et à gérer.

Basé sur le projet open source Istio, Red Hat OpenShift Service Mesh ajoute une couche transparente sur les applications distribuées existantes sans nécessiter de modifications au code du service. Vous ajoutez la prise en charge de Red Hat OpenShift Service Mesh aux services en déployant un proxy sidecar spécial aux services pertinents dans le maillage qui intercepte toutes les communications réseau entre les microservices. Vous configurez et gérez le Service Mesh en utilisant les fonctionnalités du plan de contrôle du Service Mesh.

Red Hat OpenShift Service Mesh vous offre un moyen simple de créer un réseau de services déployés qui fournissent :

- Découverte
- Équilibrage de la charge
- Authentification de service à service
- Recouvrement des défaillances
- Metrics
- Contrôle

Red Hat OpenShift Service Mesh fournit également des fonctions opérationnelles plus complexes, notamment :

- Tests A/B
- Libération des canaris

- Contrôle d'accès
- Authentification de bout en bout

2.2.2. Architecture Red Hat OpenShift Service Mesh

Red Hat OpenShift Service Mesh est logiquement divisé en un plan de données et un plan de contrôle :

Le site **data plane** est un ensemble de mandataires intelligents déployés en tant que sidecars. Ces mandataires interceptent et contrôlent toutes les communications réseau entrantes et sortantes entre les microservices dans le maillage de services. Les mandataires sidecar communiquent également avec Mixer, le centre de politique et de télémétrie à usage général.

- **Envoy proxy** intercepte tout le trafic entrant et sortant pour tous les services dans le maillage de services. Envoy est déployé en tant que sidecar du service concerné dans le même pod.

Le site **control plane** gère et configure les serveurs mandataires pour acheminer le trafic, et configure les mélangeurs pour appliquer les politiques et collecter les données télémétriques.

- **Mixer** applique des politiques de contrôle d'accès et d'utilisation (telles que l'autorisation, les limites de débit, les quotas, l'authentification et le traçage des demandes) et collecte des données télémétriques à partir du proxy Envoy et d'autres services.
- **Pilot** configure les proxies au moment de l'exécution. Pilot assure la découverte des services pour les sidecars Envoy, les capacités de gestion du trafic pour un routage intelligent (par exemple, les tests A/B ou les déploiements de canaris) et la résilience (délais d'attente, nouvelles tentatives et disjoncteurs).
- **Citadel** émet des certificats et en assure la rotation. Citadel fournit une authentification forte entre les services et les utilisateurs finaux grâce à une gestion intégrée des identités et des références. Vous pouvez utiliser Citadel pour mettre à niveau le trafic non chiffré dans le maillage de services. Citadel permet aux opérateurs d'appliquer des politiques basées sur l'identité des services plutôt que sur les contrôles du réseau.
- **Galley** ingère la configuration du maillage de services, puis la valide, la traite et la distribue. Galley protège les autres composants du maillage de services contre l'obtention des détails de la configuration de l'utilisateur à partir d'OpenShift Container Platform.

Red Hat OpenShift Service Mesh utilise également le site **istio-operator** pour gérer l'installation du plan de contrôle. Un *Operator* est un logiciel qui vous permet de mettre en œuvre et d'automatiser des activités courantes dans votre cluster OpenShift Container Platform. Il agit comme un contrôleur, vous permettant de définir ou de modifier l'état souhaité des objets dans votre cluster.

2.2.3. Comprendre le Kiali

Kiali fournit une visibilité sur votre maillage de services en vous montrant les microservices dans votre maillage de services, et comment ils sont connectés.

2.2.3.1. Vue d'ensemble de Kiali

Kiali fournit une observabilité dans le Service Mesh fonctionnant sur OpenShift Container Platform. Kiali vous aide à définir, valider et observer votre maillage de services Istio. Il vous aide à comprendre la structure de votre maillage de services en déduisant la topologie, et fournit également des informations sur la santé de votre maillage de services.

Kiali fournit une vue graphique interactive de votre espace de noms en temps réel qui offre une visibilité

sur des caractéristiques telles que les disjoncteurs, les taux de requête, la latence et même les graphiques des flux de trafic. Kiali offre des informations sur les composants à différents niveaux, des applications aux services et aux charges de travail, et peut afficher les interactions avec des informations contextuelles et des graphiques sur le nœud ou le bord du graphique sélectionné. Kiali offre également la possibilité de valider vos configurations Istio, telles que les passerelles, les règles de destination, les services virtuels, les politiques de maillage, etc. Kiali fournit des métriques détaillées, et une intégration Grafana de base est disponible pour les requêtes avancées. Le traçage distribué est assuré par l'intégration de Jaeger dans la console Kiali.

Kiali est installé par défaut dans le cadre du Red Hat OpenShift Service Mesh.

2.2.3.2. Architecture Kiali

Kiali est basé sur le projet open source Kiali. Kiali est composé de deux éléments : l'application Kiali et la console Kiali.

- **Kiali application** (back end) - Ce composant s'exécute dans la plateforme d'application conteneurisée et communique avec les composants du maillage de services, récupère et traite les données, et expose ces données à la console. L'application Kiali n'a pas besoin de stockage. Lors du déploiement de l'application sur un cluster, les configurations sont définies dans des ConfigMaps et des secrets.

- **Kiali console** (front end) - La console Kiali est une application web. L'application Kiali sert la console Kiali, qui interroge ensuite le back-end pour obtenir des données et les présenter à l'utilisateur.

En outre, Kiali dépend de services et de composants externes fournis par la plateforme d'application de conteneurs et Istio.

- **Red Hat Service Mesh** (Istio) - Istio est une exigence de Kiali. Istio est le composant qui fournit et contrôle le maillage de services. Bien que Kiali et Istio puissent être installés séparément, Kiali dépend d'Istio et ne fonctionnera pas s'il n'est pas présent. Kiali doit récupérer les données et les configurations d'Istio, qui sont exposées via Prometheus et l'API du cluster.

- **Prometheus** - Une instance Prometheus dédiée est incluse dans l'installation de Red Hat OpenShift Service Mesh. Lorsque la télémétrie Istio est activée, les données de métriques sont stockées dans Prometheus. Kiali utilise ces données Prometheus pour déterminer la topologie du maillage, afficher les métriques, calculer la santé, montrer les problèmes éventuels, etc. Kiali communique directement avec Prometheus et assume le schéma de données utilisé par Istio Telemetry. Prometheus est une dépendance d'Istio et une dépendance dure pour Kiali, et de nombreuses fonctionnalités de Kiali ne fonctionneront pas sans Prometheus.

- **Cluster API** - Kiali utilise l'API de OpenShift Container Platform (cluster API) pour récupérer et résoudre les configurations de maillage de services. Kiali interroge l'API de cluster pour récupérer, par exemple, les définitions des espaces de noms, des services, des déploiements, des pods et d'autres entités. Kiali effectue également des requêtes pour résoudre les relations entre les différentes entités du cluster. L'API de cluster est également interrogée pour récupérer les configurations Istio comme les services virtuels, les règles de destination, les règles de routage, les passerelles, les quotas, etc.

- **Jaeger** - Jaeger est optionnel, mais est installé par défaut dans le cadre de l'installation de Red Hat OpenShift Service Mesh. Lorsque vous installez la plateforme de traçage distribué dans le cadre de l'installation par défaut de Red Hat OpenShift Service Mesh, la console Kiali comprend un onglet permettant d'afficher les données de traçage distribué. Notez que les données de

traçage ne seront pas disponibles si vous désactivez la fonctionnalité de traçage distribué d'Istio. Notez également que l'utilisateur doit avoir accès à l'espace de noms où le plan de contrôle Service Mesh est installé pour afficher les données de traçage.

- **Grafana** - Grafana est optionnel, mais est installé par défaut dans le cadre de l'installation de Red Hat OpenShift Service Mesh. Lorsqu'elles sont disponibles, les pages de métriques de Kiali affichent des liens pour diriger l'utilisateur vers la même métrique dans Grafana. Notez que l'utilisateur doit avoir accès à l'espace de noms où le plan de contrôle Service Mesh est installé pour afficher les liens vers le tableau de bord Grafana et visualiser les données Grafana.

2.2.3.3. Caractéristiques du Kiali

La console Kiali est intégrée à Red Hat Service Mesh et offre les fonctionnalités suivantes :

- **Health** - Identifier rapidement les problèmes liés aux applications, aux services ou aux charges de travail.
- **Topology** - Visualisez la façon dont vos applications, services ou charges de travail communiquent via le graphe Kiali.
- **Metrics** - Des tableaux de bord prédéfinis vous permettent de visualiser le maillage des services et les performances des applications pour Go, Node.js, Quarkus, Spring Boot, Thorntail et Vert.x. Vous pouvez également créer vos propres tableaux de bord personnalisés. Quarkus, Spring Boot, Thorntail et Vert.x. Vous pouvez également créer vos propres tableaux de bord.
- **Tracing** - L'intégration avec Jaeger permet de suivre le parcours d'une requête à travers les différents microservices qui composent une application.
- **Validations** - Effectuer des validations avancées sur les objets Istio les plus courants (règles de destination, entrées de service, services virtuels, etc.)
- **Configuration** - Possibilité optionnelle de créer, mettre à jour et supprimer la configuration du routage Istio en utilisant des assistants ou directement dans l'éditeur YAML de la console Kiali.

2.2.4. Comprendre Jaeger

Chaque fois qu'un utilisateur effectue une action dans une application, une requête est exécutée par l'architecture qui peut nécessiter la participation de dizaines de services différents pour produire une réponse. Le chemin de cette requête est une transaction distribuée. Jaeger vous permet d'effectuer un traçage distribué, qui suit le chemin d'une requête à travers les différents microservices qui composent une application.

Distributed tracing est une technique utilisée pour relier les informations relatives à différentes unités de travail - généralement exécutées dans différents processus ou hôtes - afin de comprendre l'ensemble de la chaîne d'événements d'une transaction distribuée. Le traçage distribué permet aux développeurs de visualiser les flux d'appels dans les grandes architectures orientées services. Il peut s'avérer très utile pour comprendre la sérialisation, le parallélisme et les sources de latence.

Jaeger enregistre l'exécution des demandes individuelles dans l'ensemble de la pile de microservices et les présente sous forme de traces. Un **trace** est un chemin de données/d'exécution à travers le système. Une trace de bout en bout est composée d'une ou plusieurs travées.

Un site **span** représente une unité logique de travail dans Jaeger, avec un nom d'opération, l'heure de début de l'opération et sa durée. Les travées peuvent être imbriquées et ordonnées pour modéliser les relations de cause à effet.

2.2.4.1. Aperçu du traçage distribué

En tant que propriétaire de service, vous pouvez utiliser le traçage distribué pour instrumenter vos services afin de recueillir des informations sur votre architecture de service. Vous pouvez utiliser le traçage distribué pour la surveillance, le profilage du réseau et le dépannage de l'interaction entre les composants dans les applications modernes, cloud-natives et basées sur les microservices.

Le traçage distribué permet d'exécuter les fonctions suivantes :

- Contrôler les transactions distribuées
- Optimiser les performances et la latence
- Effectuer une analyse des causes profondes

Le traçage distribué de Red Hat OpenShift se compose de deux éléments principaux :

- **Red Hat OpenShift distributed tracing platform**- Ce composant est basé sur le projet open source Jaeger.
- **Red Hat OpenShift distributed tracing data collection**- Ce composant est basé sur le projet open source OpenTelemetry.

Ces deux composants sont basés sur les API et l'instrumentation OpenTracing, neutres vis-à-vis des fournisseurs.

2.2.4.2. Architecture de traçage distribuée

La plateforme de traçage distribuée est basée sur le projet open source Jaeger. La plateforme de traçage distribuée est constituée de plusieurs composants qui fonctionnent ensemble pour collecter, stocker et afficher les données de traçage.

- **Jaeger Client** (Tracer, Reporter, application instrumentée, bibliothèques clients) - Les clients Jaeger sont des implémentations spécifiques au langage de l'API OpenTracing. Ils peuvent être utilisés pour instrumenter des applications pour le traçage distribué, soit manuellement, soit avec une variété de frameworks open source existants, tels que Camel (Fuse), Spring Boot (RHOAR), MicroProfile (RHOAR/Thorntail), Wildfly (EAP), et bien d'autres, qui sont déjà intégrés à OpenTracing.
- **Jaeger Agent** (Server Queue, Processor Workers) - L'agent Jaeger est un démon de réseau qui écoute les données envoyées via le protocole UDP (User Datagram Protocol), qu'il met en lots et envoie au collecteur. L'agent est censé être placé sur le même hôte que l'application instrumentée. Ceci est typiquement accompli en ayant un sidecar dans les environnements de conteneurs comme Kubernetes.
- **Jaeger Collector** (File d'attente, travailleurs) - Comme l'agent, le collecteur peut recevoir des travées et les placer dans une file d'attente interne en vue de leur traitement. Cela permet au collecteur de retourner immédiatement au client ou à l'agent au lieu d'attendre que la travée soit acheminée vers le stockage.
- **Storage** (magasin de données) - Les collecteurs ont besoin d'un backend de stockage persistant. Jaeger dispose d'un mécanisme enfichable pour le stockage des données. Notez que pour cette version, le seul stockage pris en charge est Elasticsearch.
- **Query** (Service d'interrogation) - L'interrogation est un service qui permet d'extraire des traces du stockage.

- **Ingester** (Ingester Service) - Jaeger peut utiliser Apache Kafka comme tampon entre le collecteur et le stockage réel (Elasticsearch). Ingester est un service qui lit les données de Kafka et les écrit dans un autre backend de stockage (Elasticsearch).

- **Jaeger Console** - Jaeger fournit une interface utilisateur qui vous permet de visualiser vos données de traçage distribuées. Sur la page Recherche, vous pouvez trouver des traces et explorer les détails des portées qui composent une trace individuelle.

2.2.4.3. Fonctionnalités de traçage distribué de Red Hat OpenShift

Le traçage distribué de Red Hat OpenShift offre les capacités suivantes :

- Intégration avec Kiali - Lorsque la configuration est correcte, vous pouvez visualiser les données de traçage distribuées à partir de la console Kiali.

- Grande évolutivité - Le back-end de traçage distribué est conçu pour ne pas avoir de point de défaillance unique et pour s'adapter aux besoins de l'entreprise.

- Propagation du contexte distribué - Permet de relier des données provenant de différents composants afin de créer une trace complète de bout en bout.

- Compatibilité ascendante avec Zipkin - Le traçage distribué Red Hat OpenShift possède des API qui lui permettent d'être utilisé comme un remplacement direct de Zipkin, mais Red Hat ne prend pas en charge la compatibilité avec Zipkin dans cette version.

2.2.5. Prochaines étapes

- Préparez-vous à installer Red Hat OpenShift Service Mesh dans votre environnement OpenShift Container Platform.

2.3. DIFFÉRENCES ENTRE SERVICE MESH ET ISTIO

AVERTISSEMENT

You are viewing documentation for a Red Hat OpenShift Service Mesh release that is no longer supported.

Les plans de contrôle Service Mesh version 1.0 et 1.1 ne sont plus pris en charge. Pour plus d'informations sur la mise à niveau de votre plan de contrôle Service Mesh, voir Mise à niveau de Service Mesh .

Pour plus d'informations sur l'état de l'assistance d'une version particulière de Red Hat OpenShift Service Mesh, consultez la page Cycle de vie du produit .

Une installation de Red Hat OpenShift Service Mesh diffère des installations de la communauté Istio en amont de plusieurs façons. Les modifications apportées à Red Hat OpenShift Service Mesh sont parfois nécessaires pour résoudre des problèmes, fournir des fonctionnalités supplémentaires ou gérer les différences lors du déploiement sur OpenShift Container Platform.

La version actuelle de Red Hat OpenShift Service Mesh diffère de la version actuelle de la communauté Istio en amont de la manière suivante :

2.3.1. Installations multi-locataires

Alors qu'Istio en amont adopte une approche à locataire unique, Red Hat OpenShift Service Mesh prend en charge plusieurs plans de contrôle indépendants au sein du cluster. Red Hat OpenShift Service Mesh utilise un opérateur multilocataire pour gérer le cycle de vie du plan de contrôle.

Red Hat OpenShift Service Mesh installe un plan de contrôle multitenant par défaut. Vous spécifiez les projets qui peuvent accéder au Service Mesh, et vous isolez le Service Mesh des autres instances du plan de contrôle.

2.3.1.1. Multitenancy ou installations en grappe

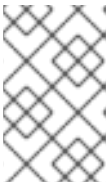
La principale différence entre une installation multitenant et une installation à l'échelle d'un cluster est l'étendue des privilèges utilisés par istod. Les composants n'utilisent plus les ressources du contrôle d'accès basé sur les rôles (RBAC) à l'échelle du cluster **ClusterRoleBinding**.

Chaque projet de la liste **ServiceMeshMemberRoll members** aura un **RoleBinding** pour chaque compte de service associé au déploiement du plan de contrôle et chaque déploiement du plan de contrôle ne surveillera que ces projets membres. Chaque projet membre se voit ajouter une étiquette **maistra.io/member-of**, où la valeur **member-of** correspond au projet contenant l'installation du plan de contrôle.

Red Hat OpenShift Service Mesh configure chaque projet membre pour assurer l'accès au réseau entre lui-même, le plan de contrôle et les autres projets membres. La configuration exacte diffère en fonction de la façon dont le réseau défini par logiciel (SDN) de OpenShift Container Platform est configuré. Voir À propos d'OpenShift SDN pour plus de détails.

Si le cluster OpenShift Container Platform est configuré pour utiliser le plugin SDN :

- **NetworkPolicy**: Red Hat OpenShift Service Mesh crée une ressource **NetworkPolicy** dans chaque projet membre permettant l'entrée de tous les pods depuis les autres membres et le plan de contrôle. Si vous supprimez un membre de Service Mesh, cette ressource **NetworkPolicy** est supprimée du projet.

  NOTE

  Cela limite également les entrées aux seuls projets membres. Si vous avez besoin de l'entrée de projets non membres, vous devez créer un site **NetworkPolicy** pour permettre le passage de ce trafic.

- **Multitenant**: Red Hat OpenShift Service Mesh joint le **NetNamespace** de chaque projet membre au **NetNamespace** du projet du plan de contrôle (l'équivalent de l'exécution de **oc adm pod-network join-projects --to control-plane-project member-project**). Si vous retirez un membre du Service Mesh, son **NetNamespace** est isolé du plan de contrôle (l'équivalent de l'exécution de **oc adm pod-network isolate-projects member-project**).

- **Subnet**: Aucune configuration supplémentaire n'est effectuée.

2.3.1.2. Ressources en grappe

Upstream Istio dispose de deux ressources de type " cluster scoped " sur lesquelles il s'appuie. Les ressources **MeshPolicy** et **ClusterRbacConfig** ne sont pas compatibles avec un cluster multitenant et ont été remplacées comme décrit ci-dessous.

- *ServiceMeshPolicy* remplace MeshPolicy pour la configuration des politiques d'authentification à l'échelle du plan de contrôle. Elle doit être créée dans le même projet que le plan de contrôle.
- *ServicemeshRbacConfig* remplace ClusterRbacConfig pour la configuration du contrôle d'accès basé sur les rôles à l'échelle du plan de contrôle. Il doit être créé dans le même projet que le plan de contrôle.

2.3.2. Différences entre Istio et Red Hat OpenShift Service Mesh

Une installation de Red Hat OpenShift Service Mesh diffère d'une installation d'Istio de plusieurs façons. Les modifications apportées à Red Hat OpenShift Service Mesh sont parfois nécessaires pour résoudre des problèmes, fournir des fonctionnalités supplémentaires ou gérer les différences lors du déploiement sur OpenShift Container Platform.

2.3.2.1. Outil de ligne de commande

L'outil de ligne de commande pour Red Hat OpenShift Service Mesh est oc. Red Hat OpenShift Service Mesh ne prend pas en charge istioctl.

2.3.2.2. Injection automatique

L'installation de la communauté Istio en amont injecte automatiquement le sidecar dans les pods des projets que vous avez étiquetés.

Red Hat OpenShift Service Mesh n'injecte pas automatiquement le sidecar dans les pods, mais vous demande d'opter pour l'injection à l'aide d'une annotation sans étiqueter les projets. Cette méthode nécessite moins de privilèges et n'entre pas en conflit avec d'autres capacités d'OpenShift telles que les pods de construction. Pour activer l'injection automatique, vous devez spécifier l'annotation **sidecar.istio.io/inject** comme décrit dans la section Injection automatique du sidecar.

2.3.2.3. Fonctionnalités du contrôle d'accès basé sur les rôles d'Istio

Le contrôle d'accès basé sur les rôles (RBAC) d'Istio fournit un mécanisme que vous pouvez utiliser pour contrôler l'accès à un service. Vous pouvez identifier les sujets par nom d'utilisateur ou en spécifiant un ensemble de propriétés et appliquer les contrôles d'accès en conséquence.

L'installation de la communauté Istio en amont inclut des options pour effectuer des correspondances exactes d'en-tête, des correspondances de caractères génériques dans les en-têtes, ou pour vérifier un en-tête contenant un préfixe ou un suffixe spécifique.

Red Hat OpenShift Service Mesh étend la capacité de faire correspondre les en-têtes de requête en utilisant une expression régulière. Spécifiez une clé de propriété de **request.regex.headers** avec une expression régulière.

Exemple d'en-tête de requête de correspondance de communauté Istio en amont

```
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 : correspondance des en-têtes de requête à l'aide d'expressions régulières

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

2.3.2.4. OpenSSL

Red Hat OpenShift Service Mesh remplace BoringSSL par OpenSSL. OpenSSL est une bibliothèque logicielle qui contient une implémentation open source des protocoles Secure Sockets Layer (SSL) et Transport Layer Security (TLS). Le binaire Red Hat OpenShift Service Mesh Proxy lie dynamiquement les bibliothèques OpenSSL (libssl et libcrypto) à partir du système d'exploitation Red Hat Enterprise Linux sous-jacent.

2.3.2.5. Modifications des composants

- Une étiquette *maistra-version* a été ajoutée à toutes les ressources.
- Toutes les ressources Ingress ont été converties en ressources OpenShift Route.
- Grafana, Tracing (Jaeger), et Kiali sont activés par défaut et exposés à travers les routes OpenShift.
- Godebug a été supprimé de tous les modèles
- Les liens **istio-multi** ServiceAccount et ClusterRoleBinding ont été supprimés, ainsi que le lien **istio-reader** ClusterRole.

2.3.2.6. Envoyé, Service secret de découverte, et certificats

- Red Hat OpenShift Service Mesh ne prend pas en charge les services basés sur QUIC.
- Le déploiement de certificats TLS à l'aide de la fonctionnalité Secret Discovery Service (SDS) d'Istio n'est pas actuellement pris en charge dans Red Hat OpenShift Service Mesh. La mise en œuvre d'Istio dépend d'un conteneur nodeagent qui utilise des montages hostPath.

2.3.2.7. Plugin Istio Container Network Interface (CNI)

Red Hat OpenShift Service Mesh inclut le plugin CNI, qui vous offre une autre façon de configurer la mise en réseau des pods d'application. Le plugin CNI remplace la configuration du réseau **init-container**,

ce qui élimine la nécessité d'accorder aux comptes de service et aux projets l'accès aux contraintes de contexte de sécurité (SCC) avec des privilèges élevés.

2.3.2.8. Routes pour les passerelles Istio

Les routes OpenShift pour les passerelles Istio sont automatiquement gérées dans Red Hat OpenShift Service Mesh. Chaque fois qu'une passerelle Istio est créée, mise à jour ou supprimée dans le service mesh, une route OpenShift est créée, mise à jour ou supprimée.

Un composant du plan de contrôle de Red Hat OpenShift Service Mesh appelé Istio OpenShift Routing (IOR) synchronise l'itinéraire de la passerelle. Pour plus d'informations, voir Création automatique d'itinéraires.

2.3.2.8.1. Domaines fourre-tout

Les domaines fourre-tout ("*") ne sont pas pris en charge. Si un tel domaine est trouvé dans la définition de la passerelle, Red Hat OpenShift Service Mesh *will* créera la route, mais s'appuiera sur OpenShift pour créer un nom d'hôte par défaut. Cela signifie que la route nouvellement créée *not* sera une route "catch all" ("*"), au lieu de cela, elle aura un nom d'hôte de la forme **<route-name>[-<project>].<suffix>**. Voir la documentation d'OpenShift pour plus d'informations sur le fonctionnement des noms d'hôtes par défaut et sur la façon dont un administrateur de cluster peut les personnaliser.

2.3.2.8.2. Sous-domaines

Les sous-domaines (par exemple : "*.domain.com") sont pris en charge. Cependant, cette capacité n'est pas activée par défaut dans OpenShift Container Platform. Cela signifie que Red Hat OpenShift Service Mesh *will* crée la route avec le sous-domaine, mais qu'elle ne sera effective que si OpenShift Container Platform est configuré pour l'activer.

2.3.2.8.3. Sécurité de la couche transport

Transport Layer Security (TLS) est pris en charge. Cela signifie que, si la passerelle contient une section **tls**, la route OpenShift sera configurée pour prendre en charge TLS.

Ressources supplémentaires

- Création automatique d'itinéraires

2.3.3. Kiali et maille de service

L'installation de Kiali via le Service Mesh sur OpenShift Container Platform diffère des installations communautaires de Kiali de plusieurs façons. Ces modifications sont parfois nécessaires pour résoudre des problèmes, fournir des fonctionnalités supplémentaires ou gérer les différences lors du déploiement sur OpenShift Container Platform.

- Kiali a été activé par défaut.
- L'entrée a été activée par défaut.
- Des mises à jour ont été apportées au ConfigMap de Kiali.
- Des mises à jour ont été apportées aux paramètres ClusterRole pour Kiali.
- Ne modifiez pas le ConfigMap, car vos modifications pourraient être écrasées par le Service Mesh ou les opérateurs Kiali. Les fichiers gérés par l'opérateur Kiali portent une étiquette ou une annotation **kiali.io**/. La mise à jour des fichiers de l'opérateur doit être limitée aux utilisateurs

disposant des privilèges **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, la mise à jour des fichiers de l'opérateur doit être limitée aux utilisateurs disposant des privilèges **dedicated-admin**.

2.3.4. Traçage distribué et maillage des services

L'installation de la plateforme de traçage distribuée avec le Service Mesh sur OpenShift Container Platform diffère des installations communautaires de Jaeger de plusieurs façons. Ces modifications sont parfois nécessaires pour résoudre des problèmes, fournir des fonctionnalités supplémentaires ou gérer les différences lors du déploiement sur OpenShift Container Platform.

- Le traçage distribué a été activé par défaut pour Service Mesh.
- L'entrée a été activée par défaut pour le Service Mesh.
- Le nom du port Zipkin a été changé en **jaeger-collector-zipkin** (au lieu de **http**)
- Jaeger utilise Elasticsearch pour le stockage par défaut lorsque vous sélectionnez l'option de déploiement **production** ou **streaming**.
- La version communautaire d'Istio fournit une route générique "tracing". Red Hat OpenShift Service Mesh utilise une route "jaeger" qui est installée par l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift et qui est déjà protégée par OAuth.
- Red Hat OpenShift Service Mesh utilise un sidecar pour le proxy Envoy, et Jaeger utilise également un sidecar, pour l'agent Jaeger. Ces deux sidecars sont configurés séparément et ne doivent pas être confondus l'un avec l'autre. Le sidecar du proxy crée des travées liées au trafic d'entrée et de sortie du pod. Le sidecar agent reçoit les spans émis par l'application et les envoie au collecteur Jaeger.

2.4. PRÉPARATION DE L'INSTALLATION DE SERVICE MESH

AVERTISSEMENT

You are viewing documentation for a Red Hat OpenShift Service Mesh release that is no longer supported.

Les plans de contrôle Service Mesh version 1.0 et 1.1 ne sont plus pris en charge. Pour plus d'informations sur la mise à niveau de votre plan de contrôle Service Mesh, voir Mise à niveau de Service Mesh .

Pour plus d'informations sur l'état de l'assistance d'une version particulière de Red Hat OpenShift Service Mesh, consultez la page Cycle de vie du produit .

Avant d'installer Red Hat OpenShift Service Mesh, passez en revue les activités d'installation et assurez-vous que vous remplissez les conditions préalables :

2.4.1. Conditions préalables

- Posséder un abonnement OpenShift Container Platform actif sur votre compte Red Hat. Si vous n'avez pas d'abonnement, contactez votre représentant commercial pour plus d'informations.
- Consultez la présentation de OpenShift Container Platform 4.12 .
- Installez OpenShift Container Platform 4.12.
  - Installer OpenShift Container Platform 4.12 sur AWS
  - Installer OpenShift Container Platform 4.12 sur AWS fourni par l'utilisateur
  - Installer OpenShift Container Platform 4.12 sur du métal nu
  - Installer OpenShift Container Platform 4.12 sur vSphere

NOTE

Si vous installez Red Hat OpenShift Service Mesh sur un réseau restreint, suivez les instructions de l'infrastructure OpenShift Container Platform que vous avez choisie.

- Installez la version de l'utilitaire de ligne de commande OpenShift Container Platform (l'outil client **oc**) qui correspond à votre version d'OpenShift Container Platform et ajoutez-la à votre chemin d'accès.
  - Si vous utilisez OpenShift Container Platform 4.12, voir À propos de l'OpenShift CLI.

2.4.2. Configurations prises en charge par Red Hat OpenShift Service Mesh

Les configurations suivantes sont les seules prises en charge pour le Red Hat OpenShift Service Mesh :

- OpenShift Container Platform version 4.6 ou ultérieure.

NOTE

OpenShift Online et Red Hat OpenShift Dedicated ne sont pas pris en charge pour Red Hat OpenShift Service Mesh.

- Le déploiement doit être contenu dans un seul cluster OpenShift Container Platform qui n'est pas fédéré.
- Cette version de Red Hat OpenShift Service Mesh est uniquement disponible sur OpenShift Container Platform x86_64.
- Cette version ne prend en charge que les configurations dans lesquelles tous les composants Service Mesh sont contenus dans le cluster OpenShift Container Platform dans lequel il fonctionne. Elle ne prend pas en charge la gestion des microservices qui résident en dehors du cluster, ou dans un scénario multi-clusters.
- Cette version ne prend en charge que les configurations qui n'intègrent pas de services externes tels que des machines virtuelles.

Pour plus d'informations sur le cycle de vie de Red Hat OpenShift Service Mesh et les configurations prises en charge, reportez-vous à la Politique d'assistance.

2.4.2.1. Configurations prises en charge pour Kiali sur Red Hat OpenShift Service Mesh

- La console d'observabilité Kiali n'est compatible qu'avec les deux versions les plus récentes des navigateurs Chrome, Edge, Firefox ou Safari.

2.4.2.2. Adaptateurs de mélangeur pris en charge

- Cette version ne prend en charge que l'adaptateur Mixer suivant :
  - adaptateur 3scale Istio

2.4.3. Vue d'ensemble de l'opérateur

Red Hat OpenShift Service Mesh nécessite les quatre opérateurs suivants :

- **OpenShift Elasticsearch** - (Facultatif) Fournit un stockage de base de données pour le traçage et la journalisation avec la plateforme de traçage distribuée. Il est basé sur le projet open source Elasticsearch.
- **Red Hat OpenShift distributed tracing platform** - Fournit un traçage distribué pour surveiller et dépanner les transactions dans les systèmes distribués complexes. Il est basé sur le projet open source Jaeger.
- **Kiali** - Fournit une observabilité pour votre maillage de services. Il vous permet de visualiser les configurations, de surveiller le trafic et d'analyser les traces dans une console unique. Il est basé sur le projet open source Kiali.
- **Red Hat OpenShift Service Mesh** - Il vous permet de connecter, de sécuriser, de contrôler et d'observer les microservices qui composent vos applications. Le Service Mesh Operator définit et surveille les ressources **ServiceMeshControlPlane** qui gèrent le déploiement, la mise à jour et la suppression des composants du Service Mesh. Il est basé sur le projet open source Istio.

AVERTISSEMENT

Voir Configuration du magasin de journaux pour plus de détails sur la configuration des paramètres Jaeger par défaut pour Elasticsearch dans un environnement de production.

2.4.4. Prochaines étapes

- Installez Red Hat OpenShift Service Mesh dans votre environnement OpenShift Container Platform.

2.5. INSTALLATION DE SERVICE MESH

AVERTISSEMENT

You are viewing documentation for a Red Hat OpenShift Service Mesh release that is no longer supported.

Les plans de contrôle Service Mesh version 1.0 et 1.1 ne sont plus pris en charge. Pour plus d'informations sur la mise à niveau de votre plan de contrôle Service Mesh, voir Mise à niveau de Service Mesh .

Pour plus d'informations sur l'état de l'assistance d'une version particulière de Red Hat OpenShift Service Mesh, consultez la page Cycle de vie du produit .

L'installation du Service Mesh implique l'installation des opérateurs OpenShift Elasticsearch, Jaeger, Kiali et Service Mesh, la création et la gestion d'une ressource **ServiceMeshControlPlane** pour déployer le plan de contrôle, et la création d'une ressource **ServiceMeshMemberRoll** pour spécifier les espaces de noms associés au Service Mesh.

NOTE

L'application de la politique de Mixer est désactivée par défaut. Vous devez l'activer pour exécuter les tâches liées aux politiques. Voir Mise à jour de l'application de la politique Mixer pour savoir comment activer l'application de la politique Mixer.

NOTE

Les installations de plan de contrôle multi-locataires constituent la configuration par défaut.

NOTE

La documentation Service Mesh utilise **istio-system** comme projet d'exemple, mais vous pouvez déployer le Service Mesh dans n'importe quel projet.

2.5.1. Conditions préalables

- Suivez le processus de préparation à l'installation de Red Hat OpenShift Service Mesh .
- Un compte avec le rôle **cluster-admin**.

Le processus d'installation de Service Mesh utilise OperatorHub pour installer la définition de ressource personnalisée **ServiceMeshControlPlane** dans le projet **openshift-operators**. Red Hat OpenShift Service Mesh définit et surveille le site **ServiceMeshControlPlane** lié au déploiement, à la mise à jour et à la suppression du plan de contrôle.

À partir de Red Hat OpenShift Service Mesh 1.1.18.2, vous devez installer l'opérateur OpenShift Elasticsearch, l'opérateur Jaeger et l'opérateur Kiali avant que l'opérateur Red Hat OpenShift Service Mesh puisse installer le plan de contrôle.

2.5.2. Installation de l'opérateur OpenShift Elasticsearch

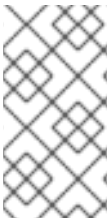
Le déploiement par défaut de la plateforme de traçage distribué Red Hat OpenShift utilise le stockage en mémoire car il est conçu pour être installé rapidement pour ceux qui évaluent le traçage distribué Red Hat OpenShift, font des démonstrations ou utilisent la plateforme de traçage distribué Red Hat OpenShift dans un environnement de test. Si vous prévoyez d'utiliser la plateforme de traçage distribué Red Hat OpenShift en production, vous devez installer et configurer une option de stockage persistant, dans ce cas, Elasticsearch.

Conditions préalables

- Vous avez accès à la console web de OpenShift Container Platform.
- Vous avez accès au cluster en tant qu'utilisateur avec le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

AVERTISSEMENT

N'installez pas les versions communautaires des opérateurs. Les opérateurs communautaires ne sont pas pris en charge.



NOTE

Si vous avez déjà installé l'OpenShift Elasticsearch Operator dans le cadre d'OpenShift Logging, vous n'avez pas besoin d'installer à nouveau l'OpenShift Elasticsearch Operator. L'opérateur de la plateforme de traçage distribuée Red Hat OpenShift crée l'instance Elasticsearch à l'aide de l'opérateur OpenShift Elasticsearch installé.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur avec le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.
2. Naviguez jusqu'à **Operators → OperatorHub**.
3. Tapez **Elasticsearch** dans la boîte de filtre pour trouver l'OpenShift Elasticsearch Operator.
4. Cliquez sur le site **OpenShift Elasticsearch Operator** fourni par Red Hat pour afficher des informations sur l'opérateur.
5. Cliquez sur **Install**.
6. Sur la page **Install Operator**, sélectionnez le canal de mise à jour **stable**. Ce canal met automatiquement à jour votre opérateur au fur et à mesure que de nouvelles versions sont publiées.
7. Accepter le projet par défaut **All namespaces on the cluster (default)** L'opérateur est ainsi installé dans le projet par défaut **openshift-operators-redhat** et mis à la disposition de tous les projets du cluster.

NOTE

L'installation d'Elasticsearch nécessite l'espace de noms **openshift-operators-redhat** pour l'opérateur OpenShift Elasticsearch. Les autres opérateurs de traçage distribués de Red Hat OpenShift sont installés dans l'espace de noms **openshift-operators**.

- Accepter la stratégie d'approbation par défaut **Automatic**. En acceptant la stratégie par défaut, lorsqu'une nouvelle version de cet opérateur est disponible, Operator Lifecycle Manager (OLM) met automatiquement à niveau l'instance en cours d'exécution de votre opérateur sans intervention humaine. Si vous sélectionnez **Manual** updates, lorsqu'une nouvelle version d'un opérateur est disponible, OLM crée une demande de mise à jour. En tant qu'administrateur de cluster, vous devez alors approuver manuellement cette demande de mise à jour pour que l'opérateur soit mis à jour avec la nouvelle version.

NOTE

La stratégie d'approbation **Manual** exige qu'un utilisateur disposant des informations d'identification appropriées approuve le processus d'installation et d'abonnement de l'opérateur.

8. Cliquez sur **Install**.

9. Sur la page **Installed Operators**, sélectionnez le projet **openshift-operators-redhat**. Attendez de voir que l'OpenShift Elasticsearch Operator affiche le statut "InstallSucceeded" avant de continuer.

2.5.3. Installation de la plateforme de traçage distribuée Red Hat OpenShift Opérateur

Pour installer Red Hat OpenShift distributed tracing platform, vous utilisez OperatorHub pour installer l'opérateur Red Hat OpenShift distributed tracing platform.

Par défaut, l'opérateur est installé dans le projet **openshift-operators**.

Conditions préalables

- Vous avez accès à la console web de OpenShift Container Platform.
- Vous avez accès au cluster en tant qu'utilisateur avec le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.
- Si vous avez besoin d'un stockage persistant, vous devez également installer l'Opérateur OpenShift Elasticsearch avant d'installer l'Opérateur de la plateforme de traçage distribuée Red Hat OpenShift.

AVERTISSEMENT

N'installez pas les versions communautaires des opérateurs. Les opérateurs communautaires ne sont pas pris en charge.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur avec le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

2. Naviguez jusqu'à **Operators → OperatorHub**.

3. Tapez **distributed tracing platform** dans le filtre pour localiser l'opérateur de la plateforme de traçage distribuée Red Hat OpenShift.

4. Cliquez sur le site **Red Hat OpenShift distributed tracing platform Operator** fourni par Red Hat pour afficher des informations sur l'opérateur.

5. Cliquez sur **Install**.

6. Sur la page **Install Operator**, sélectionnez le canal de mise à jour **stable**. Ce canal met automatiquement à jour votre opérateur au fur et à mesure que de nouvelles versions sont publiées.

7. Accepter le projet par défaut **All namespaces on the cluster (default)** L'opérateur est ainsi installé dans le projet par défaut **openshift-operators** et mis à la disposition de tous les projets du cluster.

   - Accepter la stratégie d'approbation par défaut **Automatic**. En acceptant la stratégie par défaut, lorsqu'une nouvelle version de cet opérateur est disponible, Operator Lifecycle Manager (OLM) met automatiquement à niveau l'instance en cours d'exécution de votre opérateur sans intervention humaine. Si vous sélectionnez **Manual** updates, lorsqu'une nouvelle version d'un opérateur est disponible, OLM crée une demande de mise à jour. En tant qu'administrateur de cluster, vous devez alors approuver manuellement cette demande de mise à jour pour que l'opérateur soit mis à jour avec la nouvelle version.

NOTE

La stratégie d'approbation **Manual** exige qu'un utilisateur disposant des informations d'identification appropriées approuve le processus d'installation et d'abonnement de l'opérateur.

8. Cliquez sur **Install**.

9. Naviguez jusqu'à **Operators → Installed Operators**.

10. Sur la page **Installed Operators**, sélectionnez le projet **openshift-operators**. Attendez de voir que l'opérateur de la plate-forme de traçage distribuée Red Hat OpenShift affiche un état de "Réussi" avant de continuer.

2.5.4. Installation de l'opérateur Kiali

Vous devez installer l'opérateur Kiali pour l'opérateur Red Hat OpenShift Service Mesh afin d'installer le plan de contrôle Service Mesh.

AVERTISSEMENT

N'installez pas les versions communautaires des opérateurs. Les opérateurs communautaires ne sont pas pris en charge.

Conditions préalables

- Accès à la console web d'OpenShift Container Platform.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Naviguez jusqu'à **Operators → OperatorHub**.
3. Tapez **Kiali** dans le champ de filtre pour trouver l'opérateur Kiali.
4. Cliquez sur le site **Kiali Operator** fourni par Red Hat pour afficher des informations sur l'opérateur.
5. Cliquez sur **Install**.
6. Sur la page **Operator Installation**, sélectionnez le canal de mise à jour **stable**.
7. Sélectionnez **All namespaces on the cluster (default)** L'opérateur est ainsi installé dans le projet par défaut **openshift-operators** et mis à la disposition de tous les projets du cluster.
8. Sélectionnez la stratégie d'approbation **Automatic**.

NOTE

La stratégie d'approbation manuelle exige qu'un utilisateur disposant des informations d'identification appropriées approuve le processus d'installation et d'abonnement de l'opérateur.

9. Cliquez sur **Install**.
10. La page **Installed Operators** affiche la progression de l'installation de l'opérateur Kiali.

2.5.5. Installation des opérateurs

Pour installer Red Hat OpenShift Service Mesh, installez les opérateurs suivants dans cet ordre. Répétez la procédure pour chaque opérateur.

- OpenShift Elasticsearch
- Plateforme de traçage distribuée Red Hat OpenShift
- Kiali
- Red Hat OpenShift Service Mesh

NOTE

Si vous avez déjà installé l'OpenShift Elasticsearch Operator dans le cadre d'OpenShift Logging, vous n'avez pas besoin d'installer à nouveau l'OpenShift Elasticsearch Operator. L'opérateur de la plateforme de traçage distribuée Red Hat OpenShift créera l'instance Elasticsearch à l'aide de l'opérateur OpenShift Elasticsearch installé.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur avec le rôle **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.
2. Dans la console Web OpenShift Container Platform, cliquez sur **Operators → OperatorHub**.
3. Tapez le nom de l'opérateur dans la boîte de filtre et sélectionnez la version Red Hat de l'opérateur. Les versions communautaires des opérateurs ne sont pas prises en charge.
4. Cliquez sur **Install**.
5. Sur la page **Install Operator** de chaque opérateur, acceptez les paramètres par défaut.
6. Cliquez sur **Install**. Attendez que l'opérateur soit installé avant de répéter les étapes pour l'opérateur suivant dans la liste.
   - OpenShift Elasticsearch Operator est installé dans l'espace de noms **openshift-operators-redhat** et est disponible pour tous les espaces de noms du cluster.
   - La plateforme de traçage distribuée Red Hat OpenShift est installée dans l'espace de noms **openshift-distributed-tracing** et est disponible pour tous les espaces de noms dans le cluster.
   - Les opérateurs Kiali et Red Hat OpenShift Service Mesh sont installés dans l'espace de noms **openshift-operators** et sont disponibles pour tous les espaces de noms dans le cluster.
7. Après avoir installé les quatre opérateurs, cliquez sur **Operators → Installed Operators** pour vérifier que vos opérateurs sont installés.

2.5.6. Déploiement du plan de contrôle Red Hat OpenShift Service Mesh

La ressource **ServiceMeshControlPlane** définit la configuration à utiliser lors de l'installation. Vous pouvez déployer la configuration par défaut fournie par Red Hat ou personnaliser le fichier **ServiceMeshControlPlane** en fonction de vos besoins.

Vous pouvez déployer le plan de contrôle Service Mesh en utilisant la console web OpenShift Container Platform ou à partir de la ligne de commande en utilisant l'outil client **oc**.

2.5.6.1. Déploiement du plan de contrôle à partir de la console web

Suivez cette procédure pour déployer le plan de contrôle Red Hat OpenShift Service Mesh à l'aide de la console Web. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle.

Conditions préalables

- L'opérateur Red Hat OpenShift Service Mesh doit être installé.

- Consultez les instructions pour savoir comment personnaliser l'installation de Red Hat OpenShift Service Mesh.
- Un compte avec le rôle **cluster-admin**.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**.
2. Créer un projet nommé **istio-system**.
   a. Naviguez jusqu'à **Home → Projects**.
   b. Cliquez sur **Create Project**.
   c. Saisissez **istio-system** dans le champ **Name**.
   d. Cliquez sur **Create**.
3. Naviguez jusqu'à **Operators → Installed Operators**.
4. Si nécessaire, sélectionnez **istio-system** dans le menu Projet. Il se peut que vous deviez attendre quelques instants pour que les opérateurs soient copiés dans le nouveau projet.
5. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh. Sous **Provided APIs**, l'opérateur fournit des liens pour créer deux types de ressources :
   - Une ressource **ServiceMeshControlPlane**
   - Une ressource **ServiceMeshMemberRoll**
6. Sous **Istio Service Mesh Control Plane**, cliquez sur **Create ServiceMeshControlPlane**.
7. Sur la page **Create Service Mesh Control Plane**, modifiez le YAML du modèle par défaut **ServiceMeshControlPlane** si nécessaire.

NOTE

Pour plus d'informations sur la personnalisation du plan de contrôle, voir Personnaliser l'installation de Red Hat OpenShift Service Mesh. Pour la production, vous *must* modifiez le modèle Jaeger par défaut.

8. Cliquez sur **Create** pour créer le plan de contrôle. L'opérateur crée des pods, des services et des composants du plan de contrôle Service Mesh en fonction de vos paramètres de configuration.
9. Cliquez sur l'onglet **Istio Service Mesh Control Plane**
10. Cliquez sur le nom du nouveau plan de contrôle.
11. Cliquez sur l'onglet **Resources** pour voir les ressources du plan de contrôle de Red Hat OpenShift Service Mesh que l'opérateur a créées et configurées.

2.5.6.2. Déploiement du plan de contrôle à partir de l'interface de ligne de commande

Suivez cette procédure pour déployer le plan de contrôle Red Hat OpenShift Service Mesh en ligne de commande.

Conditions préalables

- L'opérateur Red Hat OpenShift Service Mesh doit être installé.
- Consultez les instructions pour savoir comment personnaliser l'installation de Red Hat OpenShift Service Mesh.
- Un compte avec le rôle **cluster-admin**.
- Accès à la CLI OpenShift (**oc**).

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**.

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. Créer un projet nommé **istio-system**.

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

3. Créez un fichier **ServiceMeshControlPlane** nommé **istio-installation.yaml** en utilisant l'exemple trouvé dans "Customize the Red Hat OpenShift Service Mesh installation". Vous pouvez personnaliser les valeurs en fonction de votre cas d'utilisation. Pour les déploiements de production, vous pouvez *must* modifier le modèle Jaeger par défaut.

4. Exécutez la commande suivante pour déployer le plan de contrôle :

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

5. Exécutez la commande suivante pour connaître l'état de l'installation du plan de contrôle.

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

   L'installation est terminée avec succès lorsque la colonne STATUS est **ComponentsReady**.

```
NAME           READY   STATUS           PROFILES      VERSION   AGE
basic-install   11/11   ComponentsReady   ["default"]   v1.1.18   4m25s
```

6. Exécutez la commande suivante pour observer la progression des modules pendant le processus d'installation :

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

   Vous devriez obtenir un résultat similaire à celui qui suit :

Exemple de sortie

```
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
```

Pour une installation multitenant, Red Hat OpenShift Service Mesh prend en charge plusieurs plans de contrôle indépendants au sein du cluster. Vous pouvez créer des configurations réutilisables avec les modèles **ServiceMeshControlPlane**. Pour plus d'informations, voir Création de modèles de plan de contrôle.

2.5.7. Création du rouleau de membres Red Hat OpenShift Service Mesh

Le site **ServiceMeshMemberRoll** dresse la liste des projets qui appartiennent au plan de contrôle du Service Mesh. Seuls les projets listés dans le site **ServiceMeshMemberRoll** sont affectés par le plan de contrôle. Un projet n'appartient pas à un maillage de services tant que vous ne l'avez pas ajouté au rôle de membre pour un déploiement particulier du plan de contrôle.

Vous devez créer une ressource **ServiceMeshMemberRoll** nommée **default** dans le même projet que **ServiceMeshControlPlane**, par exemple **istio-system**.

2.5.7.1. Création de la liste des membres à partir de la console web

Vous pouvez ajouter un ou plusieurs projets au rouleau de membres Service Mesh à partir de la console Web. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

Conditions préalables

- Un opérateur Red Hat OpenShift Service Mesh installé et vérifié.
- Liste des projets existants à ajouter au maillage des services.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Si vous n'avez pas encore de services pour votre maillage, ou si vous partez de zéro, créez un projet pour vos applications. Ce projet doit être différent de celui où vous avez installé le plan de contrôle Service Mesh.
   a. Naviguez jusqu'à **Home → Projects**.
   b. Saisissez un nom dans le champ **Name**.
   c. Cliquez sur **Create**.
3. Naviguez jusqu'à **Operators → Installed Operators**.

4. Cliquez sur le menu **Project** et choisissez dans la liste le projet dans lequel votre ressource **ServiceMeshControlPlane** est déployée, par exemple **istio-system**.

5. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh.

6. Cliquez sur l'onglet **Istio Service Mesh Member Roll**

7. Cliquez sur **Create ServiceMeshMemberRoll**

8. Cliquez sur **Members**, puis saisissez le nom de votre projet dans le champ **Value**. Vous pouvez ajouter autant de projets que vous le souhaitez, mais un projet ne peut appartenir qu'à la ressource **one ServiceMeshMemberRoll** .

9. Cliquez sur **Create**.

2.5.7.2. Création du rouleau de membres à partir de l'interface de ligne de commande

Vous pouvez ajouter un projet au site **ServiceMeshMemberRoll** à partir de la ligne de commande.

Conditions préalables

- Un opérateur Red Hat OpenShift Service Mesh installé et vérifié.
- Liste des projets à ajouter au maillage des services.
- Accès à la CLI OpenShift (**oc**).

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform.

```
$ oc login --username=<NAMEOFUSER> https://<HOSTNAME>:6443
```

2. Si vous n'avez pas encore de services pour votre maillage, ou si vous partez de zéro, créez un projet pour vos applications. Ce projet doit être différent de celui où vous avez installé le plan de contrôle Service Mesh.

```
oc new-project <your-project> $ oc new-project <your-project>
```

3. Pour ajouter vos projets en tant que membres, modifiez l'exemple YAML suivant. Vous pouvez ajouter autant de projets que vous le souhaitez, mais un projet ne peut appartenir qu'à la ressource **one ServiceMeshMemberRoll** . Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

Exemple 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
```

4. Exécutez la commande suivante pour télécharger et créer la ressource **ServiceMeshMemberRoll** dans l'espace de noms **istio-system**.

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

5. Exécutez la commande suivante pour vérifier que le site **ServiceMeshMemberRoll** a été créé avec succès.

```
$ oc get smmr -n istio-system default
```

L'installation est terminée avec succès lorsque la colonne **STATUS** est **Configured**.

2.5.8. Ajouter ou supprimer des projets du maillage de services

Vous pouvez ajouter ou supprimer des projets d'une ressource Service Mesh **ServiceMeshMemberRoll** existante à l'aide de la console Web.

- Vous pouvez ajouter autant de projets que vous le souhaitez, mais un projet ne peut appartenir qu'à la ressource **one ServiceMeshMemberRoll** .
- La ressource **ServiceMeshMemberRoll** est supprimée lorsque la ressource **ServiceMeshControlPlane** correspondante est supprimée.

2.5.8.1. Ajouter ou supprimer des projets de la liste des membres à l'aide de la console web

Conditions préalables

- Un opérateur Red Hat OpenShift Service Mesh installé et vérifié.
- Une ressource existante **ServiceMeshMemberRoll**.
- Nom du projet contenant la ressource **ServiceMeshMemberRoll**.
- Noms des projets que vous souhaitez ajouter ou supprimer du maillage.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Naviguez jusqu'à **Operators → Installed Operators**.
3. Cliquez sur le menu **Project** et choisissez dans la liste le projet dans lequel votre ressource **ServiceMeshControlPlane** est déployée, par exemple **istio-system**.
4. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh.
5. Cliquez sur l'onglet **Istio Service Mesh Member Roll**
6. Cliquez sur le lien **default**.
7. Cliquez sur l'onglet YAML.

8. Modifiez le YAML pour ajouter ou supprimer des projets en tant que membres. Vous pouvez ajouter autant de projets que vous le souhaitez, mais un projet ne peut appartenir qu'à la ressource **one ServiceMeshMemberRoll** .

9. Cliquez sur **Save**.

10. Cliquez sur **Reload**.

2.5.8.2. Ajouter ou supprimer des projets de la liste des membres à l'aide de l'interface CLI

Vous pouvez modifier un rouleau de membres Service Mesh existant à l'aide de la ligne de commande.

Conditions préalables

- Un opérateur Red Hat OpenShift Service Mesh installé et vérifié.
- Une ressource existante **ServiceMeshMemberRoll**.
- Nom du projet contenant la ressource **ServiceMeshMemberRoll**.
- Noms des projets que vous souhaitez ajouter ou supprimer du maillage.
- Accès à la CLI OpenShift (**oc**).

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform.

2. Modifier la ressource **ServiceMeshMemberRoll**.

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

3. Modifiez le YAML pour ajouter ou supprimer des projets en tant que membres. Vous pouvez ajouter autant de projets que vous le souhaitez, mais un projet ne peut appartenir qu'à la ressource **one ServiceMeshMemberRoll** .

   Exemple servicemeshmemberroll-default.yaml

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

2.5.9. Mises à jour manuelles

Si vous choisissez de mettre à jour manuellement, le gestionnaire du cycle de vie des opérateurs (OLM) contrôle l'installation, la mise à niveau et le contrôle d'accès basé sur les rôles (RBAC) des opérateurs dans un cluster. OLM s'exécute par défaut dans OpenShift Container Platform. OLM utilise des

CatalogSources, qui utilisent l'API Operator Registry, pour rechercher les opérateurs disponibles ainsi que les mises à niveau des opérateurs installés.

- Pour plus d'informations sur la façon dont OpenShift Container Platform gère les mises à niveau, consultez la documentation Operator Lifecycle Manager.

2.5.9.1. Mise à jour des mandataires sidecar

Afin de mettre à jour la configuration des proxies sidecar, l'administrateur de l'application doit redémarrer les pods d'application.

Si votre déploiement utilise l'injection automatique de sidecar, vous pouvez mettre à jour le modèle de pod dans le déploiement en ajoutant ou en modifiant une annotation. Exécutez la commande suivante pour redéployer les pods :

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

Si votre déploiement n'utilise pas l'injection automatique de sidecars, vous devez mettre à jour manuellement les sidecars en modifiant l'image du conteneur de sidecars spécifiée dans le déploiement ou le pod, puis redémarrer les pods.

2.5.10. Prochaines étapes

- Se préparer à déployer des applications sur Red Hat OpenShift Service Mesh.

2.6. PERSONNALISATION DE LA SÉCURITÉ DANS UN MAILLAGE DE SERVICES

AVERTISSEMENT

You are viewing documentation for a Red Hat OpenShift Service Mesh release that is no longer supported.

Les plans de contrôle Service Mesh version 1.0 et 1.1 ne sont plus pris en charge. Pour plus d'informations sur la mise à niveau de votre plan de contrôle Service Mesh, voir Mise à niveau de Service Mesh .

Pour plus d'informations sur l'état de l'assistance d'une version particulière de Red Hat OpenShift Service Mesh, consultez la page Cycle de vie du produit .

Si votre application Service Mesh est construite avec un ensemble complexe de microservices, vous pouvez utiliser Red Hat OpenShift Service Mesh pour personnaliser la sécurité de la communication entre ces services. L'infrastructure d'OpenShift Container Platform et les fonctionnalités de gestion du trafic de Service Mesh peuvent vous aider à gérer la complexité de vos applications et à assurer la sécurité des services et des identités pour les microservices.

2.6.1. Activation de la sécurité mutuelle de la couche transport (mTLS)

Mutual Transport Layer Security (mTLS) est un protocole dans lequel deux parties s'authentifient mutuellement. C'est le mode d'authentification par défaut dans certains protocoles (IKE, SSH) et facultatif dans d'autres (TLS).

mTLS peut être utilisé sans modification du code de l'application ou du service. Le TLS est entièrement géré par l'infrastructure de maillage de services et entre les deux mandataires sidecar.

Par défaut, Red Hat OpenShift Service Mesh est configuré en mode permissif, où les sidecars dans Service Mesh acceptent à la fois le trafic en texte clair et les connexions qui sont chiffrées à l'aide de mTLS. Si un service dans votre maillage communique avec un service en dehors du maillage, un mTLS strict pourrait interrompre la communication entre ces services. Utilisez le mode permissif pendant que vous migrez vos charges de travail vers Service Mesh.

2.6.1.1. Activation de mTLS strict à travers le maillage

Si vos charges de travail ne communiquent pas avec des services en dehors de votre maillage et que la communication ne sera pas interrompue en n'acceptant que des connexions cryptées, vous pouvez activer mTLS dans votre maillage rapidement. Définissez **spec.istio.global.mtls.enabled** sur **true** dans votre ressource **ServiceMeshControlPlane**. L'opérateur crée les ressources nécessaires.

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

2.6.1.1.1. Configuration des sidecars pour les connexions entrantes pour des services spécifiques

Vous pouvez également configurer mTLS pour des services ou des espaces de noms individuels en créant une stratégie.

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

2.6.1.2. Configuration des sidecars pour les connexions sortantes

Créez une règle de destination pour configurer Service Mesh afin qu'il utilise mTLS lors de l'envoi de requêtes à d'autres services dans le maillage.

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

```
trafficPolicy:
  tls:
    mode: ISTIO_MUTUAL
```

2.6.1.3. Définition des versions minimale et maximale du protocole

Si votre environnement a des exigences spécifiques pour le trafic crypté dans votre maillage de services, vous pouvez contrôler les fonctions cryptographiques autorisées en définissant les valeurs **spec.security.controlPlane.tls.minProtocolVersion** ou **spec.security.controlPlane.tls.maxProtocolVersion** dans votre ressource **ServiceMeshControlPlane**. Ces valeurs, configurées dans votre ressource de plan de contrôle, définissent la version TLS minimale et maximale utilisée par les composants du maillage lorsqu'ils communiquent de manière sécurisée via TLS.

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    global:
      tls:
        minProtocolVersion: TLSv1_2
        maxProtocolVersion: TLSv1_3
```

La valeur par défaut est **TLS_AUTO** et ne spécifie pas de version de TLS.

Tableau 2.3. Valeurs valables

Valeur	Description
TLS_AUTO	par défaut
TLSv1_0	TLS version 1.0
TLSv1_1	TLS version 1.1
TLSv1_2	TLS version 1.2
TLSv1_3	TLS version 1.3

2.6.2. Configuration des suites de chiffrement et des courbes ECDH

Les suites de chiffrement et les courbes de Diffie-Hellman à courbe elliptique (courbes ECDH) peuvent vous aider à sécuriser votre maillage de services. Vous pouvez définir une liste séparée par des virgules de suites de chiffrement en utilisant **spec.istio.global.tls.cipherSuites** et de courbes ECDH en utilisant **spec.istio.global.tls.ecdhCurves** dans votre ressource **ServiceMeshControlPlane**. Si l'un de ces attributs est vide, les valeurs par défaut sont utilisées.

Le paramètre **cipherSuites** est efficace si votre maillage de services utilise TLS 1.2 ou une version antérieure. Il n'a aucun effet lors de la négociation avec TLS 1.3.

Définissez vos suites de chiffrement dans la liste séparée par des virgules, par ordre de priorité. Par exemple, **ecdhCurves: CurveP256, CurveP384** donne à **CurveP256** une priorité plus élevée que **CurveP384**.

NOTE

Vous devez inclure **TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256** ou **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256** lorsque vous configurez la suite de chiffrement. La prise en charge de HTTP/2 nécessite au moins l'une de ces suites de chiffrement.

Les suites de chiffrement prises en charge sont les suivantes

- 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

Les courbes ECDH prises en charge sont les suivantes :

- CourbeP256
- CourbeP384

- CourbeP521
- X25519

2.6.3. Ajout d'une clé et d'un certificat d'une autorité de certification externe

Par défaut, Red Hat OpenShift Service Mesh génère un certificat racine et une clé auto-signés, et les utilise pour signer les certificats de charge de travail. Vous pouvez également utiliser le certificat et la clé définis par l'utilisateur pour signer les certificats de charge de travail, avec le certificat racine défini par l'utilisateur. Cette tâche présente un exemple d'insertion de certificats et de clés dans Service Mesh.

Conditions préalables

- Vous devez avoir installé Red Hat OpenShift Service Mesh avec TLS mutuel activé pour configurer les certificats.
- Cet exemple utilise les certificats du référentiel Maistra. Pour la production, utilisez vos propres certificats provenant de votre autorité de certification.
- Vous devez déployer l'application d'exemple Bookinfo pour vérifier les résultats avec ces instructions.

2.6.3.1. Ajouter un certificat et une clé existants

Pour utiliser un certificat et une clé de signature (CA) existants, vous devez créer un fichier de chaîne de confiance qui inclut le certificat CA, la clé et le certificat racine. Vous devez utiliser les noms de fichiers exacts suivants pour chacun des certificats correspondants. Le certificat de l'autorité de certification s'appelle **ca-cert.pem**, la clé **ca-key.pem**, et le certificat racine, qui signe **ca-cert.pem**, s'appelle **root-cert.pem**. Si votre charge de travail utilise des certificats intermédiaires, vous devez les spécifier dans un fichier **cert-chain.pem**.

Ajoutez les certificats à Service Mesh en suivant les étapes suivantes. Sauvegardez localement les certificats d'exemple du repo Maistra et remplacez **<path>** par le chemin d'accès à vos certificats.

1. Créez un secret **cacert** qui comprend les fichiers d'entrée **ca-cert.pem**, **ca-key.pem**, **root-cert.pem** et **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. Dans la ressource **ServiceMeshControlPlane**, l'ensemble **global.mtls.enabled** devient **true** et l'ensemble **security.selfSigned** devient **false**. Service Mesh lit les certificats et la clé à partir des fichiers secret-mount.

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

3. Pour s'assurer que les charges de travail ajoutent rapidement les nouveaux certificats, supprimez les secrets générés par Service Mesh, nommés **istio.*** . Dans cet exemple, **istio.default**. Service Mesh émet de nouveaux certificats pour les charges de travail.

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

2.6.3.2. Vérification des certificats

Utilisez l'exemple d'application Bookinfo pour vérifier que vos certificats sont montés correctement. Tout d'abord, récupérez les certificats montés. Ensuite, vérifiez les certificats montés sur le pod.

1. Enregistrez le nom du pod dans la variable **RATINGSPOD**.

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

2. Exécutez les commandes suivantes pour récupérer les certificats montés sur le proxy.

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

Le fichier **/tmp/pod-root-cert.pem** contient le certificat racine propagé au pod.

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

Le fichier **/tmp/pod-cert-chain.pem** contient le certificat de la charge de travail et le certificat de l'autorité de certification propagé au pod.

3. Vérifiez que le certificat racine est le même que celui spécifié par l'opérateur. Remplacez **<path>** par le chemin d'accès à vos certificats.

```
$ 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
```

S'attendre à ce que la sortie soit vide.

4. Vérifiez que le certificat CA est le même que celui spécifié par Operator. Remplacez **<path>** par le chemin d'accès à vos certificats.

```
$ 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
```

S'attendre à ce que la sortie soit vide.

5. Vérifiez la chaîne de certificats depuis le certificat racine jusqu'au certificat de charge de travail. Remplacez **<path>** par le chemin d'accès à vos certificats.

```
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
```

Exemple de sortie

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

2.6.3.3. Suppression des certificats

Pour supprimer les certificats que vous avez ajoutés, procédez comme suit.

1. Retirer le secret **cacerts**.

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

2. Redéployer Service Mesh avec un certificat racine auto-signé dans la ressource **ServiceMeshControlPlane**.

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

2.7. GESTION DU TRAFIC

AVERTISSEMENT

You are viewing documentation for a Red Hat OpenShift Service Mesh release that is no longer supported.

Les plans de contrôle Service Mesh version 1.0 et 1.1 ne sont plus pris en charge. Pour plus d'informations sur la mise à niveau de votre plan de contrôle Service Mesh, voir Mise à niveau de Service Mesh .

Pour plus d'informations sur l'état de l'assistance d'une version particulière de Red Hat OpenShift Service Mesh, consultez la page Cycle de vie du produit .

Vous pouvez contrôler le flux de trafic et les appels API entre les services dans Red Hat OpenShift Service Mesh. Par exemple, certains services de votre maillage de services peuvent avoir besoin de communiquer au sein du maillage et d'autres peuvent avoir besoin d'être cachés. Gérez le trafic pour cacher des services backend spécifiques, exposer des services, créer des déploiements de test ou de version, ou ajouter une couche de sécurité sur un ensemble de services.

2.7.1. Utilisation de passerelles

Vous pouvez utiliser une passerelle pour gérer le trafic entrant et sortant de votre maillage afin de spécifier le trafic que vous souhaitez voir entrer ou sortir du maillage. Les configurations de passerelle sont appliquées aux proxys Envoy autonomes qui s'exécutent à la périphérie du maillage, plutôt qu'aux proxys Envoy sidecar qui s'exécutent aux côtés de vos charges de travail de service.

Contrairement à d'autres mécanismes de contrôle du trafic entrant dans vos systèmes, tels que les API Kubernetes Ingress, les passerelles Red Hat OpenShift Service Mesh utilisent toute la puissance et la flexibilité du routage du trafic.

La ressource de passerelle Red Hat OpenShift Service Mesh peut utiliser les propriétés d'équilibrage de charge de la couche 4-6, telles que les ports, pour exposer et configurer les paramètres TLS de Red Hat OpenShift Service Mesh. Au lieu d'ajouter le routage du trafic de la couche application (L7) à la même ressource API, vous pouvez lier un service virtuel Red Hat OpenShift Service Mesh ordinaire à la passerelle et gérer le trafic de la passerelle comme tout autre trafic de plan de données dans un maillage de services.

Les passerelles sont principalement utilisées pour gérer le trafic entrant, mais vous pouvez également configurer des passerelles de sortie. Une passerelle de sortie vous permet de configurer un nœud de sortie dédié pour le trafic quittant le maillage. Cela vous permet de limiter les services qui ont accès aux réseaux externes, ce qui ajoute un contrôle de sécurité à votre maillage de services. Vous pouvez également utiliser une passerelle pour configurer un proxy purement interne.

Exemple de passerelle

Une ressource de passerelle décrit un équilibreur de charge fonctionnant à la périphérie du maillage et recevant des connexions HTTP/TCP entrantes ou sortantes. La spécification décrit un ensemble de ports qui doivent être exposés, le type de protocole à utiliser, la configuration SNI pour l'équilibreur de charge, etc.

L'exemple suivant montre un exemple de configuration de passerelle pour le trafic HTTPS externe entrant :

```
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
```

Cette configuration de passerelle permet au trafic HTTPS provenant de **ext-host.example.com** d'entrer dans le maillage sur le port 443, mais ne spécifie aucun routage pour le trafic.

Pour spécifier le routage et pour que la passerelle fonctionne comme prévu, vous devez également lier la passerelle à un service virtuel. Pour ce faire, vous utilisez le champ Passerelles du service virtuel, comme le montre l'exemple suivant :

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

Vous pouvez ensuite configurer le service virtuel avec des règles de routage pour le trafic externe.

2.7.2. Configuration d'une passerelle d'entrée

Une passerelle d'entrée est un équilibreur de charge fonctionnant à la périphérie du réseau maillé et recevant des connexions HTTP/TCP entrantes. Elle configure les ports et les protocoles exposés, mais n'inclut aucune configuration d'acheminement du trafic. Le routage du trafic entrant est plutôt configuré avec des règles de routage, de la même manière que pour les demandes de services internes.

Les étapes suivantes montrent comment créer une passerelle et configurer un site **VirtualService** pour exposer un service de l'application d'exemple Bookinfo au trafic extérieur pour les chemins /**productpage** et /**login**.

Procédure

1. Créer une passerelle pour accepter le trafic.

    a. Créez un fichier YAML et copiez-y le YAML suivant.

    Exemple de passerelle gateway.yaml

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: info-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
```

```
    protocol: HTTP
  hosts:
  - "*"
```

b. Appliquer le fichier YAML.

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

2. Créer un objet **VirtualService** pour réécrire l'en-tête de l'hôte.

a. Créez un fichier YAML et copiez-y le YAML suivant.

Exemple de service virtuel

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

b. Appliquer le fichier YAML.

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

3. Vérifiez que la passerelle et le service virtuel ont été définis correctement.

a. Définir l'URL de la passerelle.

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

b. Définissez le numéro de port. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

```
export TARGET_PORT=$(oc -n istio-system get route istio-ingressgateway -o
jsonpath='{.spec.port.targetPort}')
```

c. Tester une page qui a été explicitement exposée.

```
curl -s -I "$GATEWAY_URL/productpage"
```

Le résultat attendu est **200**.

2.7.3. Gestion du trafic entrant

Dans Red Hat OpenShift Service Mesh, la passerelle d'entrée permet aux fonctionnalités telles que la surveillance, la sécurité et les règles de routage de s'appliquer au trafic qui entre dans le cluster. Utilisez une passerelle Service Mesh pour exposer un service en dehors du service mesh.

2.7.3.1. Détermination de l'IP et des ports d'entrée

La configuration de l'entrée diffère selon que votre environnement supporte ou non un équilibreur de charge externe. Un équilibreur de charge externe est défini dans l'IP et les ports d'entrée du cluster. Pour déterminer si l'IP et les ports de votre cluster sont configurés pour les équilibreurs de charge externes, exécutez la commande suivante. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

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

Cette commande renvoie les adresses **NAME**, **TYPE**, **CLUSTER-IP**, **EXTERNAL-IP**, **PORT(S)**, et **AGE** de chaque élément de votre espace de noms.

Si la valeur **EXTERNAL-IP** est définie, votre environnement dispose d'un équilibreur de charge externe que vous pouvez utiliser comme passerelle d'entrée.

Si la valeur **EXTERNAL-IP** est **<none>**, ou perpétuellement **<pending>**, votre environnement ne fournit pas d'équilibreur de charge externe pour la passerelle d'entrée. Vous pouvez accéder à la passerelle en utilisant le port de nœud du service.

2.7.3.1.1. Détermination des ports d'entrée avec un équilibreur de charge

Suivez ces instructions si votre environnement dispose d'un équilibreur de charge externe.

Procédure

1. Exécutez la commande suivante pour définir l'adresse IP et les ports d'entrée. Cette commande définit une variable dans votre terminal.

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

2. Exécutez la commande suivante pour définir le port d'entrée.

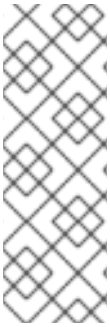
```
$ export INGRESS_PORT=$(oc -n istio-system get service istio-ingressgateway -o
jsonpath='{.spec.ports[?(@.name=="http2")].port}')
```

3. Exécutez la commande suivante pour définir le port d'entrée sécurisé.

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

4. Exécutez la commande suivante pour définir le port d'entrée TCP.

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



NOTE

Dans certains environnements, l'équilibreur de charge peut être exposé en utilisant un nom d'hôte au lieu d'une adresse IP. Dans ce cas, la valeur **EXTERNAL-IP** de la passerelle d'entrée n'est pas une adresse IP, mais un nom d'hôte. Il s'agit plutôt d'un nom d'hôte, et la commande précédente ne parvient pas à définir la variable d'environnement **INGRESS_HOST**.

Dans ce cas, utilisez la commande suivante pour corriger la valeur de **INGRESS_HOST**:

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

2.7.3.1.2. Détermination des ports d'entrée sans répartiteur de charge

Si votre environnement ne dispose pas d'un équilibreur de charge externe, déterminez les ports d'entrée et utilisez un port de nœud à la place.

Procédure

1. Définir les ports d'entrée.

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

2. Exécutez la commande suivante pour définir le port d'entrée sécurisé.

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

3. Exécutez la commande suivante pour définir le port d'entrée TCP.

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

2.7.4. Création automatique d'itinéraires

Les routes OpenShift pour les passerelles Istio sont automatiquement gérées dans Red Hat OpenShift Service Mesh. Chaque fois qu'une passerelle Istio est créée, mise à jour ou supprimée dans le service mesh, une route OpenShift est créée, mise à jour ou supprimée.

2.7.4.1. Activation de la création automatique d'itinéraires

Un composant du plan de contrôle de Red Hat OpenShift Service Mesh appelé Istio OpenShift Routing (IOR) synchronise la route de la passerelle. Activez IOR dans le cadre du déploiement du plan de contrôle.

Si la passerelle contient une section TLS, la route OpenShift sera configurée pour prendre en charge TLS.

1. Dans la ressource **ServiceMeshControlPlane**, ajoutez le paramètre **ior_enabled** et définissez-le à **true**. Par exemple, voir l'extrait de ressource suivant :

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

2.7.4.2. Sous-domaines

Red Hat OpenShift Service Mesh crée la route avec le sous-domaine, mais OpenShift Container Platform doit être configuré pour l'activer. Les sous-domaines, par exemple ***.domain.com**, sont pris en charge mais pas par défaut. Configurez une politique de wildcard OpenShift Container Platform avant de configurer un wildcard host Gateway. Pour plus d'informations, voir la section "Liens".

Si la passerelle suivante est créée :

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

Ensuite, les routes OpenShift suivantes sont créées automatiquement. Vous pouvez vérifier que les routes sont créées avec la commande suivante.

```
oc -n <control_plane_namespace> get routes
```

Résultats attendus

-

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

Si la passerelle est supprimée, Red Hat OpenShift Service Mesh supprime les routes. Cependant, les routes créées manuellement ne sont jamais modifiées par Red Hat OpenShift Service Mesh.

2.7.5. Comprendre les entrées de service

Une entrée de service ajoute une entrée au registre de service que Red Hat OpenShift Service Mesh maintient en interne. Après avoir ajouté l'entrée de service, les proxys Envoy envoient du trafic au service comme s'il s'agissait d'un service dans votre maillage. Les entrées de service vous permettent d'effectuer les opérations suivantes :

- Gérer le trafic pour les services qui s'exécutent en dehors du maillage de services.
- Redirection et transfert du trafic vers des destinations externes (telles que les API consommées sur le web) ou vers des services de l'infrastructure existante.
- Définir des politiques de relance, de temporisation et d'injection d'erreur pour les destinations externes.
- Exécutez un service de maillage dans une machine virtuelle (VM) en ajoutant des VM à votre maillage.

NOTE

Ajoutez des services d'un cluster différent au maillage pour configurer un maillage multicluster Red Hat OpenShift Service Mesh sur Kubernetes.

Exemples de saisie de services

L'exemple suivant est une entrée de service mesh-external qui ajoute la dépendance externe **ext-resource** au registre de services Red Hat OpenShift Service Mesh :

```
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
```

Spécifiez la ressource externe à l'aide du champ **hosts**. Vous pouvez le qualifier complètement ou utiliser un nom de domaine préfixé par un joker.

Vous pouvez configurer des services virtuels et des règles de destination pour contrôler le trafic vers une entrée de service de la même manière que vous configurez le trafic pour tout autre service dans le maillage. Par exemple, la règle de destination suivante configure la route du trafic pour utiliser TLS

mutuel afin de sécuriser la connexion au service externe **ext-svc.example.com** qui est configuré à l'aide de l'entrée de service :

```
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
```

2.7.6. Utilisation des services virtuels

Vous pouvez acheminer des requêtes dynamiquement vers plusieurs versions d'un microservice via Red Hat OpenShift Service Mesh avec un service virtuel. Avec les services virtuels, vous pouvez :

- Adresser plusieurs services d'application par le biais d'un seul service virtuel. Si votre maillage utilise Kubernetes, par exemple, vous pouvez configurer un service virtuel pour gérer tous les services dans un espace de noms spécifique. Un service virtuel vous permet de transformer une application monolithique en un service composé de microservices distincts avec une expérience consommateur transparente.

- Configurer des règles de trafic en combinaison avec des passerelles pour contrôler le trafic entrant et sortant.

2.7.6.1. Configuration des services virtuels

Les demandes sont acheminées vers les services au sein d'un maillage de services virtuels. Chaque service virtuel consiste en un ensemble de règles de routage qui sont évaluées dans l'ordre. Red Hat OpenShift Service Mesh fait correspondre chaque demande donnée au service virtuel à une destination réelle spécifique au sein du maillage.

Sans services virtuels, Red Hat OpenShift Service Mesh distribue le trafic en utilisant l'équilibrage de charge des moindres demandes entre toutes les instances de service. Avec un service virtuel, vous pouvez spécifier le comportement du trafic pour un ou plusieurs noms d'hôte. Les règles de routage dans le service virtuel indiquent à Red Hat OpenShift Service Mesh comment envoyer le trafic pour le service virtuel vers les destinations appropriées. Les destinations de routage peuvent être des versions du même service ou des services entièrement différents.

Procédure

1. Créez un fichier YAML à l'aide de l'exemple suivant pour acheminer les requêtes vers différentes versions de l'exemple de service d'application Bookinfo en fonction de l'utilisateur qui se connecte à l'application.

   Exemple VirtualService.yaml

   ```
   apiVersion: networking.istio.io/v1alpha3
   kind: VirtualService
   metadata:
   ```

```
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v3
```

2. Exécutez la commande suivante pour appliquer **VirtualService.yaml**, où **VirtualService.yaml** est le chemin d'accès au fichier.

```
$ oc apply -f <VirtualService.yaml>
```

2.7.6.2. Référence de configuration du service virtuel

Paramètres	Description
`spec:` `  hosts:`	Le champ **hosts** répertorie l'adresse de destination du service virtuel à laquelle les règles de routage s'appliquent. Il s'agit de l'adresse ou des adresses utilisées pour envoyer des requêtes au service. Le nom d'hôte du service virtuel peut être une adresse IP, un nom DNS ou un nom court qui se résout en un nom de domaine pleinement qualifié.

`spec:`
`  http:`
`  - match:`
	La section **http** contient les règles de routage du service virtuel qui décrivent les conditions de correspondance et les actions pour le routage du trafic HTTP/1.1, HTTP2 et gRPC envoyé à la destination spécifiée dans le champ hosts. Une règle de routage se compose de la destination où vous voulez que le trafic aille et de toutes les conditions de correspondance spécifiées. La première règle de routage de l'exemple comporte une condition qui commence par le champ "match". Dans cet exemple, ce routage s'applique à toutes les demandes de l'utilisateur **jason**. Ajoutez les champs **headers**, **end-user** et **exact** pour sélectionner les demandes appropriées.

Paramètres	Description

spec:
  http:
  - match:
  - destination:
	Le champ **destination** dans la section route spécifie la destination réelle pour le trafic qui correspond à cette condition. Contrairement à l'hôte du service virtuel, l'hôte de la destination doit être une destination réelle qui existe dans le registre de service Red Hat OpenShift Service Mesh. Il peut s'agir d'un service mesh avec des proxies ou d'un service non mesh ajouté à l'aide d'une entrée de service. Dans cet exemple, le nom d'hôte est un nom de service Kubernetes :

2.7.7. Comprendre les règles de destination

Les règles de destination sont appliquées après l'évaluation des règles de routage des services virtuels, de sorte qu'elles s'appliquent à la destination réelle du trafic. Les services virtuels acheminent le trafic vers une destination. Les règles de destination configurent ce qu'il advient du trafic à cette destination.

Par défaut, Red Hat OpenShift Service Mesh utilise une politique d'équilibrage de charge des moindres demandes, où l'instance de service dans le pool avec le plus petit nombre de connexions actives reçoit la demande. Red Hat OpenShift Service Mesh prend également en charge les modèles suivants, que vous pouvez spécifier dans les règles de destination pour les requêtes vers un service ou un sous-ensemble de services particulier.

- Aléatoire : Les demandes sont transmises de manière aléatoire aux instances du pool.
- Pondéré : Les demandes sont transmises aux instances du pool en fonction d'un pourcentage spécifique.
- Moins de demandes : Les demandes sont transmises aux instances ayant le moins de demandes.

Exemple de règle de destination

L'exemple de règle de destination suivant configure trois sous-ensembles différents pour le service de destination **my-svc**, avec des politiques d'équilibrage de charge différentes :

```
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
```

Ce guide fait référence à l'application Bookinfo pour fournir des exemples de routage dans une application d'exemple. Installez l'application Bookinfo pour découvrir comment fonctionnent ces exemples de routage.

2.7.8. Tutoriel de routage de Bookinfo

L'exemple d'application Service Mesh Bookinfo se compose de quatre microservices distincts, chacun ayant plusieurs versions. Après l'installation de l'exemple d'application Bookinfo, trois versions différentes du microservice **reviews** s'exécutent simultanément.

Lorsque vous accédez à la page de l'application Bookinfo /**product** dans un navigateur et que vous la rafraîchissez plusieurs fois, il arrive que la critique de livre contienne des étoiles et d'autres fois non. Sans version de service explicite par défaut, Service Mesh achemine les requêtes vers toutes les versions disponibles, l'une après l'autre.

Ce tutoriel vous aide à appliquer des règles qui acheminent tout le trafic vers **v1** (version 1) des microservices. Par la suite, vous pourrez appliquer une règle pour acheminer le trafic en fonction de la valeur d'un en-tête de requête HTTP.

Prérequis :

- Déployez l'application d'exemple Bookinfo pour travailler avec les exemples suivants.

2.7.8.1. Application d'un service virtuel

Dans la procédure suivante, le service virtuel achemine tout le trafic vers **v1** de chaque microservice en appliquant des services virtuels qui définissent la version par défaut des microservices.

Procédure

1. Appliquer les services virtuels.

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

2. Pour vérifier que vous avez appliqué les services virtuels, affichez les itinéraires définis à l'aide de la commande suivante :

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

   Cette commande renvoie une ressource de **kind: VirtualService** au format YAML.

Vous avez configuré Service Mesh pour router vers la version **v1** des microservices Bookinfo, y compris le service **reviews** version 1.

2.7.8.2. Test de la nouvelle configuration de l'itinéraire

Testez la nouvelle configuration en actualisant le site /**productpage** de l'application Bookinfo.

Procédure

1. Définissez la valeur du paramètre **GATEWAY_URL**. Vous pouvez utiliser cette variable pour trouver l'URL de votre page produit Bookinfo ultérieurement. Dans cet exemple, istio-system est le nom du projet de plan de contrôle.

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

2. Exécutez la commande suivante pour récupérer l'URL de la page produit.

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

3. Ouvrez le site Bookinfo dans votre navigateur.

La partie de la page consacrée aux avis s'affiche sans étoiles, quel que soit le nombre d'actualisations. Cela est dû au fait que vous avez configuré Service Mesh pour acheminer tout le trafic du service de commentaires vers la version **reviews:v1** et que cette version du service n'accède pas au service de classement par étoiles.

Votre maillage de services achemine désormais le trafic vers une version d'un service.

2.7.8.3. Itinéraire basé sur l'identité de l'utilisateur

Modifiez la configuration de l'itinéraire de manière à ce que tout le trafic provenant d'un utilisateur spécifique soit acheminé vers une version de service spécifique. Dans ce cas, tout le trafic provenant d'un utilisateur nommé **jason** sera acheminé vers le service **reviews:v2**.

Service Mesh n'a pas de compréhension spéciale et intégrée de l'identité de l'utilisateur. Cet exemple est rendu possible par le fait que le service **productpage** ajoute un en-tête personnalisé **end-user** à toutes les demandes HTTP sortantes adressées au service de révision.

Procédure

1. Exécutez la commande suivante pour activer le routage basé sur l'utilisateur dans l'application d'exemple Bookinfo.

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

2. Exécutez la commande suivante pour confirmer la création de la règle. Cette commande renvoie toutes les ressources de **kind: VirtualService** au format YAML.

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

3. Sur le site /**productpage** de l'application Bookinfo, connectez-vous en tant qu'utilisateur **jason** sans mot de passe.

4. Actualiser le navigateur. Les étoiles apparaissent à côté de chaque avis.

5. Connectez-vous en tant qu'autre utilisateur (choisissez le nom que vous voulez). Actualisez le navigateur. Les étoiles ont disparu. Le trafic est maintenant acheminé vers **reviews:v1** pour tous les utilisateurs sauf Jason.

Vous avez configuré avec succès l'application d'exemple Bookinfo pour acheminer le trafic en fonction de l'identité de l'utilisateur.

2.7.9. Ressources supplémentaires

Pour plus d'informations sur la configuration d'une politique de wildcard OpenShift Container Platform, voir Utiliser des routes wildcard.

2.8. DÉPLOIEMENT D'APPLICATIONS SUR SERVICE MESH

AVERTISSEMENT

You are viewing documentation for a Red Hat OpenShift Service Mesh release that is no longer supported.

Les plans de contrôle Service Mesh version 1.0 et 1.1 ne sont plus pris en charge. Pour plus d'informations sur la mise à niveau de votre plan de contrôle Service Mesh, voir Mise à niveau de Service Mesh .

Pour plus d'informations sur l'état de l'assistance d'une version particulière de Red Hat OpenShift Service Mesh, consultez la page Cycle de vie du produit .

Lorsque vous déployez une application dans le Service Mesh, il existe plusieurs différences entre le comportement des applications dans la version communautaire en amont d'Istio et le comportement des applications dans une installation Red Hat OpenShift Service Mesh.

2.8.1. Conditions préalables

- Review Comparing Red Hat OpenShift Service Mesh and upstream Istio community installations (en anglais)
- Revue de l'installation de Red Hat OpenShift Service Mesh

2.8.2. Création de modèles de plan de contrôle

Vous pouvez créer des configurations réutilisables avec les modèles **ServiceMeshControlPlane**. Les utilisateurs individuels peuvent étendre les modèles qu'ils créent avec leurs propres configurations. Les modèles peuvent également hériter des informations de configuration d'autres modèles. Par exemple, vous pouvez créer un plan de contrôle comptable pour l'équipe comptable et un plan de contrôle marketing pour l'équipe marketing. Si vous créez un modèle de développement et un modèle de production, les membres de l'équipe marketing et de l'équipe comptable peuvent étendre les modèles de développement et de production en les personnalisant en fonction de leur équipe.

Lorsque vous configurez des modèles de plan de contrôle, qui suivent la même syntaxe que **ServiceMeshControlPlane**, les utilisateurs héritent des paramètres de manière hiérarchique. L'Opérateur est livré avec un modèle **default** avec des paramètres par défaut pour Red Hat OpenShift Service Mesh. Pour ajouter des modèles personnalisés, vous devez créer un ConfigMap nommé **smcp-templates** dans le projet **openshift-operators** et monter le ConfigMap dans le conteneur Operator à **/usr/local/share/istio-operator/templates**.

2.8.2.1. Création du ConfigMap

Suivez cette procédure pour créer le ConfigMap.

Conditions préalables

- Un opérateur de maillage de services installé et vérifié.
- Un compte avec le rôle **cluster-admin**.
- Emplacement du déploiement de l'opérateur.
- Accès à la CLI OpenShift (**oc**).

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'administrateur de cluster.
2. À partir de l'interface de programmation, exécutez cette commande pour créer le ConfigMap nommé **smcp-templates** dans le projet **openshift-operators** et remplacez **<templates-directory>** par l'emplacement des fichiers **ServiceMeshControlPlane** sur votre disque local :

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

3. Localisez le nom de l'opérateur ClusterServiceVersion.

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

Exemple de sortie

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

4. Modifiez la version du service de cluster de l'opérateur pour indiquer à l'opérateur d'utiliser le ConfigMap **smcp-templates**.

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

5. Ajoutez un montage de volume et un volume au déploiement de l'opérateur.

```
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. Enregistrez vos modifications et quittez l'éditeur.

7. Vous pouvez désormais utiliser le paramètre **template** dans le formulaire **ServiceMeshControlPlane** pour spécifier un modèle.

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

2.8.3. Activation de l'injection automatique du side-car

Lors du déploiement d'une application, vous devez opter pour l'injection en configurant l'annotation **sidecar.istio.io/inject** dans **spec.template.metadata.annotations** vers **true** dans l'objet **deployment**. L'opt-in garantit que l'injection de sidecar n'interfère pas avec d'autres fonctionnalités d'OpenShift Container Platform telles que les builder pods utilisés par de nombreux frameworks au sein de l'écosystème d'OpenShift Container Platform.

Conditions préalables

- Identifiez les espaces de noms qui font partie de votre maillage de services et les déploiements qui ont besoin d'une injection automatique de sidecar.

Procédure

1. Pour trouver vos déploiements, utilisez la commande **oc get**.

```
$ oc get deployment -n <namespace>
```

   Par exemple, pour afficher le fichier de déploiement du microservice "ratings-v1" dans l'espace de noms **info**, utilisez la commande suivante pour afficher la ressource au format YAML.

```
oc get deployment -n info ratings-v1 -o yaml
```

2. Ouvrez le fichier YAML de configuration du déploiement de l'application dans un éditeur.

3. Ajoutez **spec.template.metadata.annotations.sidecar.istio/inject** à votre Deployment YAML et définissez **sidecar.istio.io/inject** à **true** comme indiqué dans l'exemple suivant.

   Exemple d'extrait de info deployment-ratings-v1.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ratings-v1
  namespace: info
```

```
    labels:
      app: ratings
      version: v1
  spec:
    template:
      metadata:
        annotations:
          sidecar.istio.io/inject: 'true'
```

4. Enregistrer le fichier de configuration du déploiement.

5. Ajoutez le fichier au projet qui contient votre application.

```
oc apply -n <namespace> -f deployment.yaml
```

Dans cet exemple, **info** est le nom du projet qui contient l'application **ratings-v1** et **deployment-ratings-v1.yaml** est le fichier que vous avez modifié.

```
$ oc apply -n info -f deployment-ratings-v1.yaml
```

6. Pour vérifier que la ressource a été téléchargée avec succès, exécutez la commande suivante.

```
$ oc get deployment -n <namespace> <deploymentName> -o yaml
```

Par exemple,

```
$ oc get deployment -n info ratings-v1 -o yaml
```

2.8.4. Définition de variables d'environnement de proxy par le biais d'annotations

La configuration des mandataires Envoy sidecar est gérée par le site **ServiceMeshControlPlane**.

Vous pouvez définir des variables d'environnement pour le proxy sidecar des applications en ajoutant des annotations pod au déploiement dans le fichier **injection-template.yaml**. Les variables d'environnement sont injectées dans le sidecar.

Exemple injection-template.yaml

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

AVERTISSEMENT

Vous ne devez jamais inclure les étiquettes et les annotations de **maistra.io/** lorsque vous créez vos propres ressources personnalisées. Ces étiquettes et annotations indiquent que les ressources sont générées et gérées par l'opérateur. Si vous copiez le contenu d'une ressource générée par l'opérateur lorsque vous créez vos propres ressources, n'incluez pas d'étiquettes ou d'annotations commençant par **maistra.io/**. Les ressources qui incluent ces étiquettes ou annotations seront écrasées ou supprimées par l'opérateur lors de la prochaine réconciliation.

2.8.5. Mise à jour de l'application de la politique Mixer

Dans les versions précédentes de Red Hat OpenShift Service Mesh, l'application des politiques de Mixer était activée par défaut. L'application des politiques de Mixer est désormais désactivée par défaut. Vous devez l'activer avant d'exécuter des tâches de politique.

Conditions préalables

- Accès à la CLI OpenShift (**oc**).

NOTE

Les exemples utilisent **<istio-system>** comme espace de noms du plan de contrôle. Remplacez cette valeur par l'espace de noms dans lequel vous avez déployé le plan de contrôle Service Mesh (SMCP).

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform.
2. Exécutez cette commande pour vérifier l'état actuel de l'application de la politique de mixage :

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

3. Si **disablePolicyChecks: true**, modifiez le ConfigMap du Service Mesh :

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

4. Localisez **disablePolicyChecks: true** dans le ConfigMap et changez la valeur en **false**.
5. Enregistrez la configuration et quittez l'éditeur.
6. Vérifiez à nouveau l'état d'application de la politique de mixage pour vous assurer qu'il est défini sur **false**.

2.8.5.1. Définir la bonne politique de réseau

Service Mesh crée des stratégies réseau dans le plan de contrôle Service Mesh et les espaces de noms membres pour autoriser le trafic entre eux. Avant de procéder au déploiement, tenez compte des conditions suivantes pour vous assurer que les services de votre maillage de services qui étaient

auparavant exposés par l'intermédiaire d'une route OpenShift Container Platform.

- Le trafic dans le maillage de services doit toujours passer par la passerelle d'entrée (ingress-gateway) pour qu'Istio fonctionne correctement.
- Déployer des services externes au maillage de services dans des espaces de noms distincts qui ne font partie d'aucun maillage de services.
- Les services non maillés qui doivent être déployés dans un espace de noms enlisé de maillage de services doivent étiqueter leurs déploiements **maistra.io/expose-route: "true"**, ce qui garantit que les routes d'OpenShift Container Platform vers ces services fonctionnent toujours.

2.8.6. Exemple d'application Bookinfo

L'application d'exemple Bookinfo vous permet de tester votre installation Red Hat OpenShift Service Mesh 2.3.3 sur OpenShift Container Platform.

L'application Bookinfo affiche des informations sur un livre, à l'instar d'une entrée de catalogue d'une librairie en ligne. L'application affiche une page décrivant le livre, des détails sur le livre (ISBN, nombre de pages et autres informations) et des critiques du livre.

L'application Bookinfo se compose de ces microservices :

- Le microservice **productpage** appelle les microservices **details** et **reviews** pour remplir la page.
- Le microservice **details** contient des informations sur les livres.
- Le microservice **reviews** contient des critiques de livres. Il appelle également le microservice **ratings**.
- Le microservice **ratings** contient des informations sur le classement des livres qui accompagnent une critique de livre.

Il existe trois versions du microservice de révision :

- La version v1 ne fait pas appel au service **ratings**.
- La version v2 appelle le service **ratings** et affiche chaque évaluation sous la forme d'une à cinq étoiles noires.
- La version v3 appelle le service **ratings** et affiche chaque évaluation sous la forme d'une à cinq étoiles rouges.

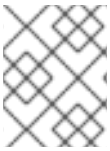
2.8.6.1. Installation de l'application Bookinfo

Ce tutoriel vous explique comment créer un exemple d'application en créant un projet, en déployant l'application Bookinfo dans ce projet et en visualisant l'application en cours d'exécution dans Service Mesh.

Prérequis :

- OpenShift Container Platform 4.1 ou supérieur installé.
- Red Hat OpenShift Service Mesh 2.3.3 installé.
- Accès à la CLI OpenShift (**oc**).

- Un compte avec le rôle **cluster-admin**.



NOTE

L'application d'exemple Bookinfo ne peut pas être installée sur IBM zSystems et IBM Power.

NOTE

Les commandes de cette section supposent que le projet de plan de contrôle Service Mesh est **istio-system**. Si vous avez installé le plan de contrôle dans un autre espace de noms, modifiez chaque commande avant de l'exécuter.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur disposant des droits cluster-admin. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.
2. Cliquez sur **Home → Projects**.
3. Cliquez sur **Create Project**.
4. Saisissez **info** comme **Project Name**, saisissez **Display Name**, et saisissez **Description**, puis cliquez sur **Create**.
   - Vous pouvez également exécuter cette commande à partir de l'interface de gestion pour créer le projet **info**.

     ```
     $ oc new-project info
     ```
5. Cliquez sur **Operators → Installed Operators**.
6. Cliquez sur le menu **Project** et utilisez l'espace de noms du plan de contrôle Service Mesh. Dans cet exemple, utilisez **istio-system**.
7. Cliquez sur l'opérateur **Red Hat OpenShift Service Mesh**.
8. Cliquez sur l'onglet **Istio Service Mesh Member Roll**
   a. Si vous avez déjà créé un rouleau Istio Service Mesh Member, cliquez sur son nom, puis sur l'onglet YAML pour ouvrir l'éditeur YAML.
   b. Si vous n'avez pas créé de site **ServiceMeshMemberRoll**, cliquez sur **Create ServiceMeshMemberRoll**.
9. Cliquez sur **Members**, puis saisissez le nom de votre projet dans le champ **Value**.
10. Cliquez sur **Create** pour enregistrer la liste actualisée des membres du Service Mesh.
    a. Vous pouvez également enregistrer l'exemple suivant dans un fichier YAML.

       Bookinfo ServiceMeshMemberRoll exemple servicemeshmemberroll-default.yaml

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

```
kind: ServiceMeshMemberRoll
metadata:
  name: default
spec:
  members:
  - info
```

b. Exécutez la commande suivante pour télécharger ce fichier et créer la ressource **ServiceMeshMemberRoll** dans l'espace de noms **istio-system**. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

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

11. Exécutez la commande suivante pour vérifier que le site **ServiceMeshMemberRoll** a été créé avec succès.

```
$ oc get smmr -n istio-system -o wide
```

L'installation est terminée avec succès lorsque la colonne **STATUS** est **Configured**.

```
NAME      READY   STATUS       AGE   MEMBERS
default   1/1     Configured   70s   ["info"]
```

12. Depuis le CLI, déployez l'application Bookinfo dans le projet `*info*` en appliquant le fichier **bookinfo.yaml**:

```
$ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.3/samples/bookinfo/platform/kube/bookinfo.yaml
```

Vous devriez obtenir un résultat similaire à celui qui suit :

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

13. Créez la passerelle d'entrée en appliquant le fichier **info-gateway.yaml**:

```
$ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.3/samples/bookinfo/networking/bookinfo-gateway.yaml
```

Vous devriez obtenir un résultat similaire à celui qui suit :

```
gateway.networking.istio.io/info-gateway created
virtualservice.networking.istio.io/info created
```

14. Définir la valeur du paramètre **GATEWAY_URL**:

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

2.8.6.2. Ajout de règles de destination par défaut

Avant de pouvoir utiliser l'application Bookinfo, vous devez d'abord ajouter des règles de destination par défaut. Il existe deux fichiers YAML préconfigurés, selon que vous avez activé ou non l'authentification mutuelle de la couche transport (TLS).

Procédure

1. Pour ajouter des règles de destination, exécutez l'une des commandes suivantes :
   - Si vous n'avez pas activé TLS mutuel :

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

   - Si vous avez activé TLS mutuel :

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

     Vous devriez obtenir un résultat similaire à celui qui suit :

     ```
     destinationrule.networking.istio.io/productpage created
     destinationrule.networking.istio.io/reviews created
     destinationrule.networking.istio.io/ratings created
     destinationrule.networking.istio.io/details created
     ```

2.8.6.3. Vérification de l'installation de Bookinfo

Pour confirmer que l'exemple d'application Bookinfo a été déployé avec succès, effectuez les étapes suivantes.

Conditions préalables

- Red Hat OpenShift Service Mesh installé.
- Suivez les étapes d'installation de l'application d'exemple Bookinfo.

Procédure à partir de l'interface de programmation

1. Connectez-vous au CLI de OpenShift Container Platform.
2. Vérifiez que tous les pods sont prêts à l'aide de cette commande :

   ```
   $ oc get pods -n info
   ```

-

Tous les pods devraient avoir un statut de **Running**. Vous devriez voir une sortie similaire à la suivante :

```
NAME                          READY   STATUS    RESTARTS   AGE
details-v1-55b869668-jh7hb       2/2     Running   0          12m
productpage-v1-6fc77ff794-nsl8r   2/2     Running   0          12m
ratings-v1-7d7d8d8b56-55scn       2/2     Running   0          12m
reviews-v1-868597db96-bdxgq       2/2     Running   0          12m
reviews-v2-5b64f47978-cvssp       2/2     Running   0          12m
reviews-v3-6dfd49b55b-vcwpf       2/2     Running   0          12m
```

3. Exécutez la commande suivante pour récupérer l'URL de la page produit :

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

4. Copiez et collez le résultat dans un navigateur web pour vérifier que la page du produit Bookinfo est déployée.

Procédure à partir de la console web Kiali

1. Obtenir l'adresse de la console web de Kiali.

   a. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur avec les droits **cluster-admin**. Si vous utilisez Red Hat OpenShift Dedicated, vous devez avoir un compte avec le rôle **dedicated-admin**.

   b. Naviguez jusqu'à **Networking** → **Routes**.

   c. Sur la page **Routes**, sélectionnez le projet de plan de contrôle Service Mesh, par exemple **istio-system**, dans le menu **Namespace**.
   La colonne **Location** affiche l'adresse liée à chaque itinéraire.

   d. Cliquez sur le lien dans la colonne **Location** pour Kiali.

   e. Cliquez sur **Log In With OpenShift** L'écran Kiali **Overview** présente des tuiles pour chaque espace de noms de projets.

2. Dans Kiali, cliquez sur **Graph**.

3. Sélectionnez info dans la liste **Namespace** et App graph dans la liste **Graph Type**.

4. Cliquez sur **Display idle nodes** dans le menu **Display**.
   Cette fonction affiche les nœuds qui sont définis mais qui n'ont pas reçu ou envoyé de demandes. Elle permet de confirmer qu'une application est correctement définie, mais qu'aucun trafic de demande n'a été signalé.

- Utilisez le menu **Duration** pour augmenter la période de temps afin de vous assurer que le trafic plus ancien est capturé.
- Utilisez le menu **Refresh Rate** pour actualiser le trafic plus ou moins souvent, ou pas du tout.

5. Cliquez sur **Services**, **Workloads** ou **Istio Config** pour afficher la liste des composants d'information et confirmer qu'ils sont sains.

2.8.6.4. Suppression de l'application Bookinfo

Suivez les étapes suivantes pour supprimer l'application Bookinfo.

Conditions préalables

- OpenShift Container Platform 4.1 ou supérieur installé.
- Red Hat OpenShift Service Mesh 2.3.3 installé.
- Accès à la CLI OpenShift (**oc**).

2.8.6.4.1. Supprimer le projet Bookinfo

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Cliquez sur **Home → Projects**.
3. Cliquez sur le menu **info** puis cliquez sur **Delete Project**.
4. Tapez **info** dans la boîte de dialogue de confirmation, puis cliquez sur **Delete**.
   - Vous pouvez également exécuter cette commande à l'aide du CLI pour créer le projet **info**.

```
$ oc delete project info
```

2.8.6.4.2. Retirer le projet Bookinfo de la liste des membres du Service Mesh

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Cliquez sur **Operators → Installed Operators**.
3. Cliquez sur le menu **Project** et choisissez **istio-system** dans la liste.
4. Cliquez sur le lien **Istio Service Mesh Member Roll**sous **Provided APIS** pour l'opérateur **Red Hat OpenShift Service Mesh**.
5. Cliquez sur le menu **ServiceMeshMemberRoll** et sélectionnez **Edit Service Mesh Member Roll**.

6. Modifiez le Service Mesh Member Roll YAML par défaut et supprimez **info** de la liste **members**.

   - Vous pouvez également exécuter cette commande à l'aide du CLI pour supprimer le projet **info** du projet **ServiceMeshMemberRoll**. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle Service Mesh.

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

7. Cliquez sur **Save** pour mettre à jour la liste des membres de Service Mesh.

2.8.7. Génération d'exemples de traces et analyse des données de traces

Jaeger est un système de traçage distribué open source. Avec Jaeger, vous pouvez effectuer une trace qui suit le chemin d'une requête à travers les différents microservices qui composent une application. Jaeger est installé par défaut dans le cadre du Service Mesh.

Ce tutoriel utilise Service Mesh et l'exemple d'application Bookinfo pour démontrer comment vous pouvez utiliser Jaeger pour effectuer un traçage distribué.

Prérequis :

- OpenShift Container Platform 4.1 ou supérieur installé.
- Red Hat OpenShift Service Mesh 2.3.3 installé.
- Jaeger activé lors de l'installation.
- L'application d'exemple Bookinfo est installée.

Procédure

1. Après avoir installé l'application d'exemple Bookinfo, envoyez du trafic au réseau. Saisissez plusieurs fois la commande suivante.

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

   Cette commande simule un utilisateur visitant le microservice **productpage** de l'application.

2. Dans la console OpenShift Container Platform, naviguez vers **Networking → Routes** et recherchez la route Jaeger, qui est l'URL répertoriée sous **Location**.

   - Il est également possible d'utiliser l'interface de commande pour obtenir des détails sur l'itinéraire. Dans cet exemple, **istio-system** est l'espace de noms du plan de contrôle Service Mesh :

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

     a. Entrez la commande suivante pour afficher l'URL de la console Jaeger. Collez le résultat dans un navigateur et naviguez jusqu'à cette URL.

        ```
        echo $JAEGER_URL
        ```

3. Connectez-vous en utilisant le même nom d'utilisateur et le même mot de passe que ceux que vous utilisez pour accéder à la console OpenShift Container Platform.

4. Dans le volet gauche du tableau de bord Jaeger, dans le menu **Service**, sélectionnez **productpage.info** et cliquez sur **Find Traces** au bas du volet. Une liste de traces s'affiche.

5. Cliquez sur l'une des traces de la liste pour ouvrir une vue détaillée de cette trace. Si vous cliquez sur la première trace de la liste, qui est la plus récente, les détails correspondant à la dernière actualisation du site /**productpage** s'affichent.

2.9. VISUALISATION DES DONNÉES ET OBSERVABILITÉ

AVERTISSEMENT

You are viewing documentation for a Red Hat OpenShift Service Mesh release that is no longer supported.

Les plans de contrôle Service Mesh version 1.0 et 1.1 ne sont plus pris en charge. Pour plus d'informations sur la mise à niveau de votre plan de contrôle Service Mesh, voir Mise à niveau de Service Mesh .

Pour plus d'informations sur l'état de l'assistance d'une version particulière de Red Hat OpenShift Service Mesh, consultez la page Cycle de vie du produit .

Vous pouvez visualiser la topologie, la santé et les métriques de votre application dans la console Kiali. Si votre service a des problèmes, la console Kiali offre des moyens de visualiser le flux de données à travers votre service. Vous pouvez visualiser des informations sur les composants du maillage à différents niveaux, y compris les applications abstraites, les services et les charges de travail. Elle fournit également une vue graphique interactive de votre espace de noms en temps réel.

Avant de commencer

Vous pouvez observer le flux de données à travers votre application si vous avez une application installée. Si vous n'avez pas installé votre propre application, vous pouvez voir comment l'observabilité fonctionne dans Red Hat OpenShift Service Mesh en installant l'application d'exemple Bookinfo.

2.9.1. Visualisation des données de maillage des services

L'opérateur Kiali travaille avec les données de télémétrie recueillies dans Red Hat OpenShift Service Mesh pour fournir des graphiques et des diagrammes de réseau en temps réel des applications, des services et des charges de travail dans votre espace de noms.

Pour accéder à la console Kiali, vous devez avoir installé Red Hat OpenShift Service Mesh et configuré des projets pour le service mesh.

Procédure

1. Utilisez le sélecteur de perspective pour passer à la perspective **Administrator**.

2. Cliquez sur **Home → Projects**.

3. Cliquez sur le nom de votre projet. Par exemple, cliquez sur **info**.

4. Dans la section **Launcher**, cliquez sur **Kiali**.

5. Connectez-vous à la console Kiali avec le même nom d'utilisateur et le même mot de passe que ceux utilisés pour accéder à la console OpenShift Container Platform.

Lorsque vous vous connectez pour la première fois à la console Kiali, vous voyez la page **Overview** qui affiche tous les espaces de noms de votre maillage de services que vous avez le droit de voir.

Si vous validez l'installation de la console, il se peut qu'il n'y ait pas de données à afficher.

2.9.2. Visualisation des données de maillage des services dans la console Kiali

Le Kiali Graph offre une visualisation puissante du trafic de votre maillage. La topologie combine le trafic des requêtes en temps réel avec vos informations de configuration Istio pour présenter un aperçu immédiat du comportement de votre maillage de services, ce qui vous permet de localiser rapidement les problèmes. Plusieurs types de graphiques vous permettent de visualiser le trafic sous forme de topologie de service de haut niveau, de topologie de charge de travail de bas niveau ou de topologie au niveau de l'application.

Plusieurs graphiques sont disponibles :

- Le site **App graph** montre une charge de travail globale pour toutes les applications qui sont étiquetées de la même manière.

- Le site **Service graph** présente un nœud pour chaque service de votre maillage, mais exclut toutes les applications et charges de travail du graphique. Il fournit une vue d'ensemble et regroupe tout le trafic pour des services définis.

- Le site **Versioned App graph** présente un nœud pour chaque version d'une application. Toutes les versions d'une application sont regroupées.

- Le site **Workload graph** présente un nœud pour chaque charge de travail dans votre maillage de services. Ce graphique ne nécessite pas l'utilisation des étiquettes d'application et de version. Si votre application n'utilise pas d'étiquettes de version, utilisez ce graphique.

Les nœuds du graphique sont agrémentés d'une variété d'informations, indiquant diverses options d'acheminement comme les services virtuels et les entrées de service, ainsi que des configurations spéciales comme l'injection de fautes et les disjoncteurs. Il peut identifier les problèmes mTLS, les problèmes de latence, le trafic d'erreur, etc. Le graphique est hautement configurable, peut afficher des animations de trafic et dispose de puissantes capacités de recherche et de masquage.

Cliquez sur le bouton **Legend** pour obtenir des informations sur les formes, les couleurs, les flèches et les badges affichés dans le graphique.

Pour afficher un résumé des métriques, sélectionnez un nœud ou une arête dans le graphique pour afficher les détails de sa métrique dans le panneau de résumé.

2.9.2.1. Modifier la disposition des graphes dans Kiali

La présentation du graphe Kiali peut varier en fonction de l'architecture de votre application et des données à afficher. Par exemple, le nombre de nœuds du graphe et leurs interactions peuvent déterminer le rendu du graphe Kiali. Parce qu'il n'est pas possible de créer une disposition unique qui rende bien dans toutes les situations, Kiali offre un choix de plusieurs dispositions différentes.

Conditions préalables

- Si vous n'avez pas installé votre propre application, installez l'application d'exemple Bookinfo. Générez ensuite du trafic pour l'application Bookinfo en entrant plusieurs fois la commande suivante.

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

  Cette commande simule un utilisateur visitant le microservice **productpage** de l'application.

Procédure

1. Lancer la console Kiali.
2. Cliquez sur **Log In With OpenShift**
3. Dans la console Kiali, cliquez sur **Graph** pour afficher un graphique de l'espace de noms.
4. Dans le menu **Namespace**, sélectionnez l'espace de noms de votre application, par exemple **info**.
5. Pour choisir une autre présentation graphique, effectuez l'une ou l'autre des opérations suivantes, ou les deux :
   - Sélectionnez différents groupes de données dans le menu situé en haut du graphique.
     - Graphique de l'application
     - Graphique des services
     - Graphique de l'application versionnée (par défaut)
     - Graphique de la charge de travail
   - Sélectionnez une autre présentation de graphique dans la légende située au bas du graphique.
     - Mise en page par défaut dagre
     - Schéma 1 cose-bilkent
     - Disposition 2 cola

2.10. RESSOURCES PERSONNALISÉES

AVERTISSEMENT

You are viewing documentation for a Red Hat OpenShift Service Mesh release that is no longer supported.

Les plans de contrôle Service Mesh version 1.0 et 1.1 ne sont plus pris en charge. Pour plus d'informations sur la mise à niveau de votre plan de contrôle Service Mesh, voir Mise à niveau de Service Mesh .

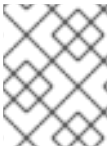
Pour plus d'informations sur l'état de l'assistance d'une version particulière de Red Hat OpenShift Service Mesh, consultez la page Cycle de vie du produit .

Vous pouvez personnaliser votre Red Hat OpenShift Service Mesh en modifiant la ressource personnalisée Service Mesh par défaut ou en créant une nouvelle ressource personnalisée.

2.10.1. Conditions préalables

- Un compte avec le rôle **cluster-admin**.
- Vous avez terminé le processus de préparation à l'installation de Red Hat OpenShift Service Mesh.
- Les opérateurs ont été installés.

2.10.2. Ressources personnalisées Red Hat OpenShift Service Mesh



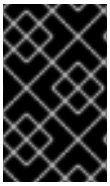
NOTE

Le projet **istio-system** est utilisé comme exemple dans toute la documentation Service Mesh, mais vous pouvez utiliser d'autres projets si nécessaire.

Un *custom resource* vous permet d'étendre l'API dans un projet ou un cluster Red Hat OpenShift Service Mesh. Lorsque vous déployez Service Mesh, il crée un site **ServiceMeshControlPlane** par défaut que vous pouvez modifier pour changer les paramètres du projet.

L'opérateur Service Mesh étend l'API en ajoutant le type de ressource **ServiceMeshControlPlane**, qui vous permet de créer des objets **ServiceMeshControlPlane** dans les projets. En créant un objet **ServiceMeshControlPlane**, vous demandez à l'opérateur d'installer un plan de contrôle Service Mesh dans le projet, configuré avec les paramètres que vous avez définis dans l'objet **ServiceMeshControlPlane**.

Cet exemple de définition **ServiceMeshControlPlane** contient tous les paramètres pris en charge et déploie les images Red Hat OpenShift Service Mesh 1.1.18.2 basées sur Red Hat Enterprise Linux (RHEL).



IMPORTANT

L'adaptateur 3scale Istio est déployé et configuré dans le fichier de ressources personnalisé. Il nécessite également un compte 3scale fonctionnel(SaaS ou On-Premises).

Exemple istio-installation.yaml

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

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

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

    mixer:
      policy:
        autoscaleEnabled: false

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

    pilot:
      autoscaleEnabled: false
      traceSampling: 100

    kiali:
      enabled: true

    grafana:
      enabled: true

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

2.10.3. Paramètres du plan de contrôle ServiceMesh

Les exemples suivants illustrent l'utilisation des paramètres **ServiceMeshControlPlane** et les tableaux fournissent des informations supplémentaires sur les paramètres pris en charge.

IMPORTANT

Les ressources que vous configurez pour Red Hat OpenShift Service Mesh avec ces paramètres, y compris les CPU, la mémoire et le nombre de pods, sont basées sur la configuration de votre cluster OpenShift Container Platform. Configurez ces paramètres en fonction des ressources disponibles dans la configuration actuelle de votre cluster.

2.10.3.1. Exemple global d'Istio

Voici un exemple qui illustre les paramètres globaux d'Istio pour **ServiceMeshControlPlane** et une description des paramètres disponibles avec les valeurs appropriées.

NOTE

Pour que l'adaptateur 3scale Istio fonctionne, **disablePolicyChecks** doit être **false**.

Exemple de paramètres globaux

```
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
```

Tableau 2.4. Paramètres globaux

Paramètres	Description	Valeurs	Valeur par défaut
disablePolicyChecks	Ce paramètre permet d'activer ou de désactiver les contrôles de politique.	**true**/**false**	**true**

Paramètres	Description	Valeurs	Valeur par défaut
policyCheckFailOpen	Ce paramètre indique si le trafic est autorisé à passer par le sidecar Envoy lorsque le service de stratégie Mixer ne peut être atteint.	**true**/**false**	**false**
tag	La balise que l'opérateur utilise pour extraire les images Istio.	Une balise d'image conteneur valide.	**1.1.0**
hub	Le hub que l'opérateur utilise pour extraire les images Istio.	Un référentiel d'images valide.	**maistra**/ ou **registry.redhat.io/openshift-service-mesh**/
mtls	Ce paramètre permet d'activer/désactiver la sécurité mutuelle de la couche transport (mTLS) entre les services par défaut.	**true**/**false**	**false**
imagePullSecrets	Si l'accès au registre fournissant les images Istio est sécurisé, listez une imagePullSecret ici.	redhat-registry-pullsecret OR quay-pullsecret	Aucun

Ces paramètres sont spécifiques au sous-ensemble de paramètres globaux du proxy.

Tableau 2.5. Paramètres du proxy

Type	Paramètres	Description	Valeurs	Valeur par défaut
requests	**cpu**	La quantité de ressources CPU demandées pour le proxy Envoy.	Ressources CPU, spécifiées en cœurs ou en millicores (par exemple, 200m, 0,5, 1) en fonction de la configuration de votre environnement.	**10m**

Type	Paramètres	Description	Valeurs	Valeur par défaut
	memory	La quantité de mémoire demandée pour le proxy Envoy	Mémoire disponible en octets (par exemple, 200Ki, 50Mi, 5Gi) en fonction de la configuration de votre environnement.	**128Mi**
limits	**cpu**	Quantité maximale de ressources CPU demandée pour le proxy Envoy.	Ressources CPU, spécifiées en cœurs ou en millicores (par exemple, 200m, 0,5, 1) en fonction de la configuration de votre environnement.	**2000m**
	memory	Quantité maximale de mémoire que le proxy Envoy est autorisé à utiliser.	Mémoire disponible en octets (par exemple, 200Ki, 50Mi, 5Gi) en fonction de la configuration de votre environnement.	**1024Mi**

2.10.3.2. Configuration de la passerelle Istio

Voici un exemple qui illustre les paramètres de la passerelle Istio pour le site **ServiceMeshControlPlane** et une description des paramètres disponibles avec les valeurs appropriées.

Exemple de paramètres de passerelle

```
gateways:
  egress:
    enabled: true
    runtime:
      deployment:
        autoScaling:
          enabled: true
          maxReplicas: 5
          minReplicas: 1
  enabled: true
  ingress:
    enabled: true
    runtime:
```

```
    deployment:
      autoScaling:
        enabled: true
        maxReplicas: 5
        minReplicas: 1
```

Tableau 2.6. Paramètres de la passerelle Istio

Paramètres	Description	Valeurs	Valeur par défaut
gateways.egress.runtime.deployment.autoScaling.enabled	Ce paramètre active/désactive la mise à l'échelle automatique.	**true**/**false**	**true**
gateways.egress.runtime.deployment.autoScaling.minReplicas	Nombre minimum de modules à déployer pour la passerelle de sortie en fonction du paramètre **autoscaleEnabled**.	Un nombre valide de pods allouables en fonction de la configuration de votre environnement.	**1**
gateways.egress.runtime.deployment.autoScaling.maxReplicas	Nombre maximal de modules à déployer pour la passerelle de sortie en fonction du paramètre **autoscaleEnabled**.	Un nombre valide de pods allouables en fonction de la configuration de votre environnement.	**5**
gateways.ingress.runtime.deployment.autoScaling.enabled	Ce paramètre active/désactive la mise à l'échelle automatique.	**true**/**false**	**true**
gateways.ingress.runtime.deployment.autoScaling.minReplicas	Le nombre minimum de pods à déployer pour la passerelle d'entrée en fonction du paramètre **autoscaleEnabled**.	Un nombre valide de pods allouables en fonction de la configuration de votre environnement.	**1**
gateways.ingress.runtime.deployment.autoScaling.maxReplicas	Nombre maximal de modules à déployer pour la passerelle d'entrée en fonction du paramètre **autoscaleEnabled**.	Un nombre valide de pods allouables en fonction de la configuration de votre environnement.	**5**

Les administrateurs de clusters peuvent se référer à la section Utilisation d'itinéraires génériques pour obtenir des instructions sur la manière d'activer les sous-domaines.

2.10.3.3. Configuration d'Istio Mixer

Voici un exemple qui illustre les paramètres du mixeur pour le site **ServiceMeshControlPlane** et une description des paramètres disponibles avec les valeurs appropriées.

Exemple de paramètres du mélangeur

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

Tableau 2.7. Paramètres de la politique Istio Mixer

Paramètres	Description	Valeurs	Valeur par défaut
enabled	Ce paramètre permet d'activer/désactiver le mixeur.	**true**/**false**	**true**
autoscaleEnabled	Ce paramètre active/désactive la mise à l'échelle automatique. Désactivez ce paramètre pour les environnements de petite taille.	**true**/**false**	**true**
autoscaleMin	Le nombre minimum de pods à déployer en fonction du paramètre **autoscaleEnabled**.	Un nombre valide de pods allouables en fonction de la configuration de votre environnement.	**1**
autoscaleMax	Le nombre maximum de pods à déployer en fonction du paramètre **autoscaleEnabled**.	Un nombre valide de pods allouables en fonction de la configuration de votre environnement.	**5**

Tableau 2.8. Paramètres de télémétrie d'Istio Mixer

Type	Paramètres	Description	Valeurs	Défaut
requests	**cpu**	Le pourcentage de ressources CPU demandées pour la télémétrie Mixer.	Ressources CPU en millicores en fonction de la configuration de votre environnement.	**10m**

Type	Paramètres	Description	Valeurs	Défaut
	memory	La quantité de mémoire requise pour la télémétrie du mélangeur.	Mémoire disponible en octets (par exemple, 200Ki, 50Mi, 5Gi) en fonction de la configuration de votre environnement.	**128Mi**
limits	**cpu**	Pourcentage maximum de ressources CPU que la télémétrie Mixer est autorisée à utiliser.	Ressources CPU en millicores en fonction de la configuration de votre environnement.	**4800m**
	memory	Quantité maximale de mémoire que la télémétrie du mélangeur est autorisée à utiliser.	Mémoire disponible en octets (par exemple, 200Ki, 50Mi, 5Gi) en fonction de la configuration de votre environnement.	**4G**

2.10.3.4. Configuration d'Istio Pilot

Vous pouvez configurer Pilot pour planifier ou fixer des limites à l'allocation des ressources. L'exemple suivant illustre les paramètres de Pilot pour le site **ServiceMeshControlPlane** et une description des paramètres disponibles avec les valeurs appropriées.

Exemple de paramètres de pilotage

```
spec:
  runtime:
    components:
      pilot:
        deployment:
          autoScaling:
            enabled: true
            minReplicas: 1
            maxReplicas: 5
            targetCPUUtilizationPercentage: 85
        pod:
          tolerations:
          - key: node.kubernetes.io/unreachable
            operator: Exists
            effect: NoExecute
            tolerationSeconds: 60
```

```
      affinity:
        podAntiAffinity:
          requiredDuringScheduling:
          - key: istio
            topologyKey: kubernetes.io/hostname
            operator: In
            values:
            - pilot
      container:
        resources:
          limits:
            cpu: 100m
            memory: 128M
```

Tableau 2.9. Paramètres d'Istio Pilot

Paramètres	Description	Valeurs	Valeur par défaut
cpu	Le pourcentage de ressources CPU demandées pour Pilot.	Ressources CPU en millicores en fonction de la configuration de votre environnement.	**10m**
memory	La quantité de mémoire demandée pour Pilot.	Mémoire disponible en octets (par exemple, 200Ki, 50Mi, 5Gi) en fonction de la configuration de votre environnement.	**128Mi**
autoscaleEnabled	Ce paramètre active/désactive la mise à l'échelle automatique. Désactivez ce paramètre pour les environnements de petite taille.	**true**/**false**	**true**
traceSampling	Cette valeur détermine la fréquence de l'échantillonnage aléatoire. **Note:** Augmenter pour le développement ou les tests.	Un pourcentage valable.	**1.0**

2.10.4. Configuration de Kiali

Lorsque le Service Mesh Operator crée le site **ServiceMeshControlPlane**, il traite également la ressource Kiali. L'opérateur Kiali utilise ensuite cet objet lors de la création d'instances Kiali.

Les paramètres par défaut de Kiali spécifiés dans le site **ServiceMeshControlPlane** sont les suivants :

Exemple de paramètres Kiali

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

Tableau 2.10. Paramètres de Kiali

Paramètres	Description	Valeurs	Valeur par défaut
activée	Ce paramètre permet d'activer ou de désactiver Kiali. Kiali est activé par défaut.	**true**/**false**	**true**
dashboard viewOnlyMode	Ce paramètre active/désactive le mode vue seule pour la console Kiali. Lorsque ce mode est activé, les utilisateurs ne peuvent pas utiliser la console pour apporter des modifications au Service Mesh.	**true**/**false**	**false**
ingress enabled	Ce paramètre permet d'activer ou de désactiver l'entrée pour Kiali.	**true**/**false**	**true**

2.10.4.1. Configuration de Kiali pour Grafana

Lorsque vous installez Kiali et Grafana dans le cadre de Red Hat OpenShift Service Mesh, l'opérateur configure les éléments suivants par défaut :

- Grafana est activé en tant que service externe pour Kiali
- Autorisation de Grafana pour la console Kiali
- URL Grafana pour la console Kiali

Kiali peut détecter automatiquement l'URL de Grafana. Cependant, si vous avez une installation personnalisée de Grafana qui n'est pas facilement détectable par Kiali, vous devez mettre à jour la valeur de l'URL dans la ressource **ServiceMeshControlPlane**.

Paramètres supplémentaires de Grafana

-

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

2.10.4.2. Configuration de Kiali pour Jaeger

Lorsque vous installez Kiali et Jaeger dans le cadre de Red Hat OpenShift Service Mesh, l'opérateur configure les éléments suivants par défaut :

- Jaeger est activé en tant que service externe pour Kiali
- Autorisation Jaeger pour la console Kiali
- URL Jaeger pour la console Kiali

Kiali peut détecter automatiquement l'URL de Jaeger. Cependant, si vous avez une installation Jaeger personnalisée qui n'est pas facilement détectable par Kiali, vous devez mettre à jour la valeur de l'URL dans la ressource **ServiceMeshControlPlane**.

Paramètres supplémentaires de Jaeger

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

2.10.5. Configuration de Jaeger

Lorsque l'opérateur de Service Mesh crée la ressource **ServiceMeshControlPlane**, il peut également créer les ressources pour le traçage distribué. Service Mesh utilise Jaeger pour le traçage distribué.

Vous pouvez spécifier votre configuration Jaeger de deux façons :

- Configurer Jaeger dans la ressource **ServiceMeshControlPlane**. Cette approche présente certaines limites.
- Configurer Jaeger dans une ressource personnalisée **Jaeger**, puis référencer cette instance de Jaeger dans la ressource **ServiceMeshControlPlane**. S'il existe une ressource Jaeger correspondant à la valeur de **name**, le plan de contrôle utilisera l'installation existante. Cette approche vous permet de personnaliser entièrement votre configuration Jaeger.

Les paramètres Jaeger par défaut spécifiés dans le site **ServiceMeshControlPlane** sont les suivants :

Paramètres par défaut de all-in-one Jaeger

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  version: v1.1
  istio:
    tracing:
      enabled: true
      jaeger:
        template: all-in-one
```

Tableau 2.11. Paramètres Jaeger

Paramètres	Description	Valeurs	Valeur par défaut
`tracing: enabled:`	Ce paramètre permet d'activer/désactiver l'installation et le déploiement du traçage par l'opérateur du maillage de services. L'installation de Jaeger est activée par défaut. Pour utiliser un déploiement Jaeger existant, définissez cette valeur sur **false**.	**true**/**false**	**true**
`jaeger: template:`	Ce paramètre indique la stratégie de déploiement Jaeger à utiliser.	• **all-in-one**- Pour le développement, les essais, les démonstrations et la validation du concept. • **production-elasticsearch** - Pour la production.	**all-in-one**

NOTE

Le modèle par défaut dans la ressource **ServiceMeshControlPlane** est la stratégie de déploiement **all-in-one** qui utilise le stockage en mémoire. Pour la production, la seule option de stockage prise en charge est Elasticsearch. Vous devez donc configurer la ressource **ServiceMeshControlPlane** pour qu'elle demande le modèle **production-elasticsearch** lorsque vous déployez Service Mesh dans un environnement de production.

2.10.5.1. Configuration d'Elasticsearch

La stratégie de déploiement par défaut de Jaeger utilise le modèle **all-in-one** afin que l'installation puisse être réalisée avec un minimum de ressources. Cependant, comme le modèle **all-in-one** utilise un

stockage en mémoire, il n'est recommandé qu'à des fins de développement, de démonstration ou de test et ne doit PAS être utilisé dans des environnements de production.

Si vous déployez Service Mesh et Jaeger dans un environnement de production, vous devez remplacer le modèle par le modèle **production-elasticsearch**, qui utilise Elasticsearch pour les besoins de stockage de Jaeger.

Elasticsearch est une application gourmande en mémoire. L'ensemble initial de nœuds spécifié dans l'installation par défaut d'OpenShift Container Platform peut ne pas être suffisant pour supporter le cluster Elasticsearch. Vous devez modifier la configuration par défaut d'Elasticsearch pour qu'elle corresponde à votre cas d'utilisation et aux ressources que vous avez demandées pour votre installation d'OpenShift Container Platform. Vous pouvez ajuster les limites de CPU et de mémoire pour chaque composant en modifiant le bloc de ressources avec des valeurs de CPU et de mémoire valides. Des nœuds supplémentaires doivent être ajoutés au cluster si vous souhaitez fonctionner avec la quantité de mémoire recommandée (ou plus). Veillez à ne pas dépasser les ressources requises pour votre installation d'OpenShift Container Platform.

Paramètres de Jaeger par défaut avec Elasticsearch

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

Tableau 2.12. Paramètres d'Elasticsearch

Paramètres	Description	Valeurs	Valeur par défaut	Examples
`tracing: enabled:`	Ce paramètre permet d'activer ou de désactiver le traçage dans Service Mesh. Jaeger est installé par défaut.	**true**/**false**	**true**	

Paramètres	Description	Valeurs	Valeur par défaut	Examples
ingress: enabled:	Ce paramètre permet d'activer ou de désactiver l'entrée pour Jaeger.	**true**/**false**	**true**	
jaeger: template:	Ce paramètre indique la stratégie de déploiement Jaeger à utiliser.	**all-in-one**/**production-elasticsearch**	**all-in-one**	
elasticsearch: nodeCount:	Nombre de nœuds Elasticsearch à créer.	Valeur entière.	1	Preuve de concept = 1, Déploiement minimal =3
requests: cpu:	Nombre d'unités centrales de traitement pour les requêtes, en fonction de la configuration de votre environnement.	Spécifié en carottes ou en millicores (par exemple, 200m, 0,5, 1).	1Gi	Preuve du concept = 500m, déploiement minimum =1
requests: memory:	Mémoire disponible pour les requêtes, en fonction de la configuration de votre environnement.	Spécifié en octets (par exemple, 200Ki, 50Mi, 5Gi).	500m	Preuve de concept = 1Gi, déploiement minimum = 16Gi*
limits: cpu:	Limitation du nombre d'unités centrales de traitement, en fonction de la configuration de votre environnement.	Spécifié en carottes ou en millicores (par exemple, 200m, 0,5, 1).		Preuve du concept = 500m, déploiement minimum =1
limits: memory:	Limite de mémoire disponible en fonction de la configuration de votre environnement.	Spécifié en octets (par exemple, 200Ki, 50Mi, 5Gi).		Preuve de concept = 1Gi, déploiement minimum = 16Gi*

Paramètres	Description	Valeurs	Valeur par défaut	Examples
	* Chaque nœud Elasticsearch peut fonctionner avec un paramètre de mémoire inférieur, bien que cela soit recommandé pour les déploiements en production (**not**). Pour une utilisation en production, vous ne devriez pas avoir moins de 16Gi alloués à chaque pod par défaut, mais il est préférable d'en allouer autant que possible, jusqu'à 64Gi par pod.			

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**.
2. Naviguez jusqu'à **Operators → Installed Operators**.
3. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh.
4. Cliquez sur l'onglet **Istio Service Mesh Control Plane**
5. Cliquez sur le nom du fichier du plan de contrôle, par exemple **basic-install**.
6. Cliquez sur l'onglet **YAML**.
7. Modifiez les paramètres Jaeger, en remplaçant le modèle par défaut **all-in-one** par les paramètres du modèle **production-elasticsearch**, modifiés en fonction de votre cas d'utilisation. Assurez-vous que l'indentation est correcte.
8. Cliquez sur **Save**.
9. Cliquez sur **Reload**. OpenShift Container Platform redéploie Jaeger et crée les ressources Elasticsearch en fonction des paramètres spécifiés.

2.10.5.2. Connexion à une instance Jaeger existante

Pour que le SMCP puisse se connecter à une instance Jaeger existante, les conditions suivantes doivent être remplies :

- L'instance Jaeger est déployée dans le même espace de noms que le plan de contrôle, par exemple dans l'espace de noms **istio-system**.
- Pour permettre une communication sécurisée entre les services, vous devez activer le proxy oauth, qui sécurise la communication avec votre instance Jaeger, et vous assurer que le secret est monté dans votre instance Jaeger pour que Kiali puisse communiquer avec elle.
- Pour utiliser une instance Jaeger personnalisée ou déjà existante, définissez **spec.istio.tracing.enabled** à "false" pour désactiver le déploiement d'une instance Jaeger.
- Fournissez le bon point de terminaison jaeger-collector à Mixer en définissant **spec.istio.global.tracer.zipkin.address** avec le nom d'hôte et le port de votre service jaeger-collector. Le nom d'hôte du service est généralement **<jaeger-instance-name>-collector.<namespace>.svc.cluster.local**.
- Fournissez à Kiali le bon point de terminaison jaeger-query pour la collecte des traces en définissant **spec.istio.kiali.jaegerInClusterURL** avec le nom d'hôte de votre service jaeger-query - le port n'est normalement pas nécessaire, car il utilise 443 par défaut. Le nom d'hôte du

service est généralement **<jaeger-instance-name>-query.<namespace>.svc.cluster.local**.

- Fournissez l'URL du tableau de bord de votre instance Jaeger à Kiali pour permettre l'accès à Jaeger via la console Kiali. Vous pouvez récupérer l'URL à partir de la route OpenShift créée par l'opérateur Jaeger. Si votre ressource Jaeger s'appelle **external-jaeger** et réside dans le projet **istio-system**, vous pouvez récupérer la route à l'aide de la commande suivante :

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

Exemple de sortie

```
NAME                HOST/PORT                                  PATH   SERVICES                [...]
external-jaeger     external-jaeger-istio-system.apps.test            external-jaeger-query   [...]
```

La valeur sous **HOST/PORT** est l'URL accessible de l'extérieur du tableau de bord Jaeger.

Exemple de ressource Jaeger

```
apiVersion: jaegertracing.io/v1
kind: "Jaeger"
metadata:
  name: "external-jaeger"
  # Deploy to the Control Plane Namespace
  namespace: istio-system
spec:
  # Set Up Authentication
  ingress:
    enabled: true
    security: oauth-proxy
    openshift:
      # This limits user access to the Jaeger instance to users who have access
      # to the control plane namespace. Make sure to set the correct namespace here
      sar: '{"namespace": "istio-system", "resource": "pods", "verb": "get"}'
      htpasswdFile: /etc/proxy/htpasswd/auth

  volumeMounts:
  - name: secret-htpasswd
    mountPath: /etc/proxy/htpasswd
  volumes:
  - name: secret-htpasswd
    secret:
      secretName: htpasswd
```

L'exemple suivant **ServiceMeshControlPlane** suppose que vous avez déployé Jaeger à l'aide de l'opérateur Jaeger et de la ressource Jaeger d'exemple.

Exemple ServiceMeshControlPlane avec Jaeger externe

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
metadata:
  name: external-jaeger
  namespace: istio-system
spec:
```

```
  version: v1.1
  istio:
    tracing:
      # Disable Jaeger deployment by service mesh operator
      enabled: false
    global:
      tracer:
        zipkin:
          # Set Endpoint for Trace Collection
          address: external-jaeger-collector.istio-system.svc.cluster.local:9411
    kiali:
      # Set Jaeger dashboard URL
      dashboard:
        jaegerURL: https://external-jaeger-istio-system.apps.test
      # Set Endpoint for Trace Querying
      jaegerInClusterURL: external-jaeger-query.istio-system.svc.cluster.local
```

2.10.5.3. Configuration d'Elasticsearch

La stratégie de déploiement par défaut de Jaeger utilise le modèle **all-in-one** afin que l'installation puisse être réalisée avec un minimum de ressources. Cependant, comme le modèle **all-in-one** utilise un stockage en mémoire, il n'est recommandé qu'à des fins de développement, de démonstration ou de test et ne doit PAS être utilisé dans des environnements de production.

Si vous déployez Service Mesh et Jaeger dans un environnement de production, vous devez remplacer le modèle par le modèle **production-elasticsearch**, qui utilise Elasticsearch pour les besoins de stockage de Jaeger.

Elasticsearch est une application gourmande en mémoire. L'ensemble initial de nœuds spécifié dans l'installation par défaut d'OpenShift Container Platform peut ne pas être suffisant pour supporter le cluster Elasticsearch. Vous devez modifier la configuration par défaut d'Elasticsearch pour qu'elle corresponde à votre cas d'utilisation et aux ressources que vous avez demandées pour votre installation d'OpenShift Container Platform. Vous pouvez ajuster les limites de CPU et de mémoire pour chaque composant en modifiant le bloc de ressources avec des valeurs de CPU et de mémoire valides. Des nœuds supplémentaires doivent être ajoutés au cluster si vous souhaitez fonctionner avec la quantité de mémoire recommandée (ou plus). Veillez à ne pas dépasser les ressources requises pour votre installation d'OpenShift Container Platform.

Paramètres de Jaeger par défaut avec Elasticsearch

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    tracing:
    enabled: true
    ingress:
      enabled: true
    jaeger:
      template: production-elasticsearch
      elasticsearch:
        nodeCount: 3
        redundancyPolicy:
        resources:
          requests:
```

```
        cpu: "1"
        memory: "16Gi"
      limits:
        cpu: "1"
        memory: "16Gi"
```

Tableau 2.13. Paramètres d'Elasticsearch

Paramètres	Description	Valeurs	Valeur par défaut	Examples
tracing: enabled:	Ce paramètre permet d'activer ou de désactiver le traçage dans Service Mesh. Jaeger est installé par défaut.	**true**/**false**	**true**	
ingress: enabled:	Ce paramètre permet d'activer ou de désactiver l'entrée pour Jaeger.	**true**/**false**	**true**	
jaeger: template:	Ce paramètre indique la stratégie de déploiement Jaeger à utiliser.	**all-in-one**/**production-elasticsearch**	**all-in-one**	
elasticsearch: nodeCount:	Nombre de nœuds Elasticsearch à créer.	Valeur entière.	1	Preuve de concept = 1, Déploiement minimal =3
requests: cpu:	Nombre d'unités centrales de traitement pour les requêtes, en fonction de la configuration de votre environnement.	Spécifié en carottes ou en millicores (par exemple, 200m, 0,5, 1).	1Gi	Preuve du concept = 500m, déploiement minimum =1
requests: memory:	Mémoire disponible pour les requêtes, en fonction de la configuration de votre environnement.	Spécifié en octets (par exemple, 200Ki, 50Mi, 5Gi).	500m	Preuve de concept = 1Gi, déploiement minimum = 16Gi*

Paramètres	Description	Valeurs	Valeur par défaut	Examples
limits: cpu:	Limitation du nombre d'unités centrales de traitement, en fonction de la configuration de votre environnement.	Spécifié en carottes ou en millicores (par exemple, 200m, 0,5, 1).		Preuve du concept = 500m, déploiement minimum =1
limits: memory:	Limite de mémoire disponible en fonction de la configuration de votre environnement.	Spécifié en octets (par exemple, 200Ki, 50Mi, 5Gi).		Preuve de concept = 1Gi, déploiement minimum = 16Gi*
	* Chaque nœud Elasticsearch peut fonctionner avec un paramètre de mémoire inférieur, bien que cela soit recommandé pour les déploiements en production (**not**). Pour une utilisation en production, vous ne devriez pas avoir moins de 16Gi alloués à chaque pod par défaut, mais il est préférable d'en allouer autant que possible, jusqu'à 64Gi par pod.			

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform en tant qu'utilisateur ayant le rôle **cluster-admin**.
2. Naviguez jusqu'à **Operators → Installed Operators**.
3. Cliquez sur l'opérateur Red Hat OpenShift Service Mesh.
4. Cliquez sur l'onglet **Istio Service Mesh Control Plane**
5. Cliquez sur le nom du fichier du plan de contrôle, par exemple **basic-install**.
6. Cliquez sur l'onglet **YAML**.
7. Modifiez les paramètres Jaeger, en remplaçant le modèle par défaut **all-in-one** par les paramètres du modèle **production-elasticsearch**, modifiés en fonction de votre cas d'utilisation. Assurez-vous que l'indentation est correcte.
8. Cliquez sur **Save**.
9. Cliquez sur **Reload**. OpenShift Container Platform redéploie Jaeger et crée les ressources Elasticsearch en fonction des paramètres spécifiés.

2.10.5.4. Configuration de la tâche de nettoyage de l'index Elasticsearch

Lorsque l'opérateur du Service Mesh crée le site **ServiceMeshControlPlane**, il crée également la ressource personnalisée (CR) pour Jaeger. L'opérateur de la plateforme de traçage distribuée Red Hat OpenShift utilise ensuite cette CR lors de la création d'instances Jaeger.

Lors de l'utilisation du stockage Elasticsearch, une tâche est créée par défaut pour nettoyer les anciennes traces. Pour configurer les options de cette tâche, vous devez éditer la ressource personnalisée (CR) Jaeger, afin de l'adapter à votre cas d'utilisation. Les options pertinentes sont énumérées ci-dessous.

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

Tableau 2.14. Paramètres du nettoyeur d'index Elasticsearch

Paramètres	Valeurs	Description
a permis :	vrai/ faux	Activer ou désactiver la tâche de nettoyage de l'index.
nombre de jours :	valeur entière	Nombre de jours à attendre avant de supprimer un index.
calendrier :	"55 23 * * *"	Expression Cron pour l'exécution du travail

Pour plus d'informations sur la configuration d'Elasticsearch avec OpenShift Container Platform, voir Configuration du magasin de logs .

2.10.6. configuration 3scale

Le tableau suivant explique les paramètres de l'adaptateur Istio 3scale dans la ressource **ServiceMeshControlPlane**.

Exemple de paramètres 3scale

```
spec:
  addons:
    3Scale:
      enabled: false
      PARAM_THREESCALE_LISTEN_ADDR: 3333
      PARAM_THREESCALE_LOG_LEVEL: info
      PARAM_THREESCALE_LOG_JSON: true
      PARAM_THREESCALE_LOG_GRPC: false
      PARAM_THREESCALE_REPORT_METRICS: true
      PARAM_THREESCALE_METRICS_PORT: 8080
      PARAM_THREESCALE_CACHE_TTL_SECONDS: 300
      PARAM_THREESCALE_CACHE_REFRESH_SECONDS: 180
      PARAM_THREESCALE_CACHE_ENTRIES_MAX: 1000
      PARAM_THREESCALE_CACHE_REFRESH_RETRIES: 1
```

```
PARAM_THREESCALE_ALLOW_INSECURE_CONN: false
PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS: 10
PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS: 60
PARAM_USE_CACHED_BACKEND: false
PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS: 15
PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED: true
```

Tableau 2.15. paramètres 3scale

Paramètres	Description	Valeurs	Valeur par défaut
enabled	Utilisation ou non de l'adaptateur 3scale	**true**/**false**	**false**
PARAM_THREESCALE_LISTEN_ADDR	Définit l'adresse d'écoute du serveur gRPC	Numéro de port valide	**3333**
PARAM_THREESCALE_LOG_LEVEL	Définit le niveau minimum de sortie du journal.	**debug**, **info**, **warn**, **error**, ou **none**	**info**
PARAM_THREESCALE_LOG_JSON	Contrôle si le journal est formaté en JSON	**true**/**false**	**true**
PARAM_THREESCALE_LOG_GRPC	Contrôle si le journal contient des informations gRPC	**true**/**false**	**true**
PARAM_THREESCALE_REPORT_METRICS	Contrôle si les métriques du système 3scale et du backend sont collectées et rapportées à Prometheus	**true**/**false**	**true**
PARAM_THREESCALE_METRICS_PORT	Définit le port à partir duquel le point d'extrémité 3scale **/metrics** peut être mis au rebut	Numéro de port valide	**8080**
PARAM_THREESCALE_CACHE_TTL_SECONDS	Délai, en secondes, à attendre avant de purger les éléments expirés de la mémoire cache	Période de temps en secondes	**300**
PARAM_THREESCALE_CACHE_REFRESH_SECONDS	Période de temps avant l'expiration au cours de laquelle les éléments du cache sont tentés d'être rafraîchis	Période de temps en secondes	**180**

Paramètres	Description	Valeurs	Valeur par défaut
PARAM_THREESCALE_CACHE_ENTRIES_MAX	Nombre maximal d'éléments pouvant être stockés dans le cache à tout moment. La valeur **0** permet de désactiver la mise en cache	Numéro valide	**1000**
PARAM_THREESCALE_CACHE_REFRESH_RETRIES	Le nombre de fois où les hôtes inaccessibles sont réessayés pendant une boucle de mise à jour du cache	Numéro valide	**1**
PARAM_THREESCALE_ALLOW_INSECURE_CONN	Permet d'ignorer la vérification des certificats lors de l'appel des API **3scale**. Il n'est pas recommandé d'activer cette option.	**true**/**false**	**false**
PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS	Définit le nombre de secondes à attendre avant de terminer les requêtes vers le système 3scale et le backend	Période de temps en secondes	**10**
PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS	Définit le nombre maximum de secondes (/-10% de gigue) qu'une connexion peut durer avant d'être fermée	Période de temps en secondes	60
PARAM_USE_CACHE_BACKEND	Si true, tentative de création d'un cache d'apisonator en mémoire pour les demandes d'autorisation	**true**/**false**	**false**
PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS	Si le cache du backend est activé, ceci définit l'intervalle en secondes pour vider le cache par rapport à 3scale	Période de temps en secondes	15

Paramètres	Description	Valeurs	Valeur par défaut
PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED	Lorsque le cache du backend ne peut pas récupérer les données d'autorisation, il convient de refuser (fermé) ou d'autoriser (ouvert) les demandes	**true/false**	**true**

2.11. UTILISATION DE L'ADAPTATEUR 3SCALE ISTIO

AVERTISSEMENT

You are viewing documentation for a Red Hat OpenShift Service Mesh release that is no longer supported.

Les plans de contrôle Service Mesh version 1.0 et 1.1 ne sont plus pris en charge. Pour plus d'informations sur la mise à niveau de votre plan de contrôle Service Mesh, voir Mise à niveau de Service Mesh .

Pour plus d'informations sur l'état de l'assistance d'une version particulière de Red Hat OpenShift Service Mesh, consultez la page Cycle de vie du produit .

L'adaptateur 3scale Istio est un adaptateur optionnel qui vous permet d'étiqueter un service fonctionnant dans Red Hat OpenShift Service Mesh et d'intégrer ce service avec la solution 3scale API Management. Il n'est pas requis pour Red Hat OpenShift Service Mesh.

2.11.1. Intégrer l'adaptateur 3scale avec Red Hat OpenShift Service Mesh

Vous pouvez utiliser ces exemples pour configurer les requêtes vers vos services en utilisant l'adaptateur Istio 3scale.

Prérequis :

- Red Hat OpenShift Service Mesh version 1.x
- Un compte 3scale fonctionnel(SaaS ou 3scale 2.5 On-Premises)
- L'activation du backend cache nécessite 3scale 2.9 ou plus
- Prérequis de Red Hat OpenShift Service Mesh

NOTE

Pour configurer l'adaptateur 3scale Istio, reportez-vous aux ressources personnalisées Red Hat OpenShift Service Mesh pour obtenir des instructions sur l'ajout de paramètres d'adaptateur au fichier de ressources personnalisées.

NOTE

Portez une attention particulière à la ressource **kind: handler**. Vous devez la mettre à jour avec les informations d'identification de votre compte 3scale. Vous pouvez optionnellement ajouter une ressource **service_id** à un handler, mais ceci n'est conservé que pour la compatibilité ascendante, puisque cela rendrait le handler utile uniquement pour un seul service dans votre compte 3scale. Si vous ajoutez **service_id** à un handler, l'activation de 3scale pour d'autres services nécessite la création d'autres handlers avec des **service_ids** différents.

Utilisez un seul gestionnaire par compte 3scale en suivant les étapes ci-dessous :

Procédure

1. Créez un gestionnaire pour votre compte 3scale et indiquez les informations d'identification de votre compte. N'indiquez pas d'identifiant de service.

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

   En option, vous pouvez fournir un champ **backend_url** dans la section *params* pour remplacer l'URL fournie par la configuration 3scale. Cela peut être utile si l'adaptateur fonctionne sur le même cluster que l'instance 3scale sur site, et que vous souhaitez utiliser le DNS interne du cluster.

2. Modifiez ou corrigez la ressource Déploiement de tous les services appartenant à votre compte 3scale comme suit :

   a. Ajouter l'étiquette **"service-mesh.3scale.net/service-id"** avec une valeur correspondant à une **service_id** valide.

   b. Ajoutez l'étiquette **"service-mesh.3scale.net/credentials"** dont la valeur est celle de *name of the handler resource* de l'étape 1.

3. Faites l'étape 2 pour le lier à vos identifiants de compte 3scale et à son identifiant de service, lorsque vous avez l'intention d'ajouter d'autres services.

4. Modifiez la configuration de la règle avec votre configuration 3scale pour envoyer la règle au gestionnaire 3scale.

   Exemple de configuration d'une règle

```
apiVersion: "config.istio.io/v1alpha2"
kind: rule
metadata:
  name: threescale
```

```
spec:
  match: destination.labels["service-mesh.3scale.net"] == "true"
  actions:
    - handler: threescale.handler
      instances:
        - threescale-authorization.instance
```

2.11.1.1. Générer des ressources personnalisées à 3 échelles

L'adaptateur comprend un outil qui vous permet de générer les ressources personnalisées **handler**, **instance** et **rule**.

Tableau 2.16. Utilisation

Option	Description	Exigée	Valeur par défaut
-h, --help	Produit une sortie d'aide pour les options disponibles	Non	
--name	Nom unique pour cet URL, paire de jetons	Oui	
-n, --namespace	Espace de noms pour générer des modèles	Non	istio-system
-t, --token	jeton d'accès 3scale	Oui	
-u, --url	uRL du portail administratif 3scale	Oui	
--backend-url	uRL du backend 3scale. Si elle est définie, elle remplace la valeur lue dans la configuration du système	Non	
-s, --service	iD API/Service 3scale	Non	
--auth	modèle d'authentification à 3 niveaux à spécifier (1=clé API, 2=identification de l'application/clé de l'application, 3=OIDC)	Non	Hybride
-o, --output	Fichier dans lequel enregistrer les manifestes produits	Non	Sortie standard

Option	Description	Exigée	Valeur par défaut
--version	Affiche la version de l'interface de programmation et quitte immédiatement	Non	

2.11.1.1.1. Générer des modèles à partir d'exemples d'URL

NOTE

- Exécutez les commandes suivantes via **oc exec** à partir de l'image du conteneur de l'adaptateur 3scale dans Générer des manifestes à partir d'un adaptateur déployé.
- Utilisez la commande **3scale-config-gen** pour éviter les erreurs de syntaxe et d'indentation de YAML.
- Vous pouvez omettre le site **--service** si vous utilisez les annotations.
- Cette commande doit être invoquée à partir de l'image du conteneur via **oc exec**.

Procédure

- Utilisez la commande **3scale-config-gen** pour autogénérer des fichiers modèles permettant à la paire token, URL d'être partagée par plusieurs services en tant que gestionnaire unique :

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

- L'exemple suivant génère les modèles avec l'identifiant du service intégré dans le gestionnaire :

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

Ressources supplémentaires

- Jetons.

2.11.1.2. Générer des manifestes à partir d'un adaptateur déployé

NOTE

- **NAME** est un identifiant que vous utilisez pour vous identifier auprès du service que vous gérez avec 3scale.
- La référence **CREDENTIALS_NAME** est un identifiant qui correspond à la section **match** dans la configuration de la règle. Si vous utilisez l'outil CLI, l'identifiant **NAME** est automatiquement utilisé.
- Sa valeur n'a pas besoin d'être spécifique : la valeur de l'étiquette doit simplement correspondre au contenu de la règle. Voir Routage du trafic de service à travers l'adaptateur pour plus d'informations.

1. Exécutez cette commande pour générer des manifestes à partir d'un adaptateur déployé dans l'espace de noms **istio-system**:

```
$ 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}
```

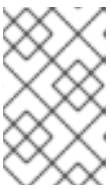
2. Cela produira des exemples de sortie dans le terminal. Modifiez ces exemples si nécessaire et créez les objets à l'aide de la commande **oc create**.

3. Lorsque la requête atteint l'adaptateur, ce dernier doit savoir comment le service correspond à une API sur 3scale. Vous pouvez fournir cette information de deux façons :

   a. Étiqueter la charge de travail (recommandé)

   b. Le code dur du gestionnaire est le suivant **service_id**

4. Mettre à jour la charge de travail avec les annotations requises :



NOTE

Il suffit de mettre à jour l'identifiant de service fourni dans cet exemple s'il n'est pas déjà intégré dans le gestionnaire. **The setting in the handler takes precedence**.

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

2.11.1.3. Acheminement du trafic de service via l'adaptateur

Suivez les étapes suivantes pour générer du trafic pour votre service grâce à l'adaptateur 3scale.

Conditions préalables

- Les informations d'identification et l'identifiant de service de votre administrateur 3scale.

Procédure

1. Correspondre à la règle **destination.labels["service-mesh.3scale.net/credentials"] == "threescale"** que vous avez précédemment créée dans la configuration, dans la ressource **kind: rule**.
2. Ajoutez l'étiquette ci-dessus à **PodTemplateSpec** sur le Déploiement de la charge de travail cible pour intégrer un service. la valeur, **threescale**, fait référence au nom du gestionnaire généré. Ce gestionnaire stocke le jeton d'accès requis pour appeler 3scale.
3. Ajoutez l'étiquette **destination.labels["service-mesh.3scale.net/service-id"] == "replace-me"** à la charge de travail pour transmettre l'identifiant du service à l'adaptateur via l'instance au moment de la demande.

2.11.2. Configurer les paramètres d'intégration dans 3scale

Suivez cette procédure pour configurer les paramètres d'intégration de 3scale.

NOTE

Pour les clients SaaS de 3scale, Red Hat OpenShift Service Mesh est activé dans le cadre du programme Early Access.

Procédure

1. Naviguez vers **[your_API_name] → Integration**
2. Cliquez sur **Settings**.
3. Sélectionnez l'option **Istio** sous *Deployment*.
   - L'option **API Key (user_key)** sous *Authentication* est sélectionnée par défaut.
4. Cliquez sur **Update Product** pour enregistrer votre sélection.
5. Cliquez sur **Configuration**.
6. Cliquez sur **Update Configuration**.

2.11.3. Comportement de mise en cache

Les réponses des API du système 3scale sont mises en cache par défaut dans l'adaptateur. Les entrées sont supprimées du cache lorsqu'elles sont plus anciennes que la valeur **cacheTTLSeconds**. Par défaut également, le rafraîchissement automatique des entrées mises en cache sera tenté quelques secondes avant leur expiration, sur la base de la valeur **cacheRefreshSeconds**. Vous pouvez désactiver le rafraîchissement automatique en définissant cette valeur à un niveau supérieur à la valeur **cacheTTLSeconds**.

La mise en cache peut être entièrement désactivée en donnant à **cacheEntriesMax** une valeur non positive.

En utilisant le processus de rafraîchissement, les valeurs mises en cache dont les hôtes deviennent inaccessibles seront réessayées avant d'être finalement purgées lorsqu'elles auront dépassé leur date d'expiration.

2.11.4. Authentification des demandes

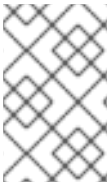
Cette version prend en charge les méthodes d'authentification suivantes :

- **Standard API Keys**les systèmes d'authentification sont des chaînes ou des hachages aléatoires uniques qui servent d'identificateur et de jeton secret.
- **Application identifier and key pairs**es chaînes d'identification immuables et les chaînes de clés secrètes mutables.
- **OpenID authentication method**chaîne d'identification du client analysée à partir du jeton Web JSON.

2.11.4.1. Application de modèles d'authentification

Modifiez la ressource personnalisée **instance**, comme illustré dans les exemples de méthodes d'authentification suivants, pour configurer le comportement d'authentification. Vous pouvez accepter les informations d'authentification de :

- En-têtes de la demande
- Paramètres de la demande
- Les en-têtes et les paramètres de la requête



NOTE

Lorsque vous spécifiez des valeurs à partir d'en-têtes, elles doivent être en minuscules. Par exemple, si vous souhaitez envoyer un en-tête sous la forme **User-Key**, il doit être référencé dans la configuration sous la forme **request.headers["user-key"]**.

2.11.4.1.1. Méthode d'authentification de la clé API

Service Mesh recherche la clé API dans les paramètres de requête et les en-têtes de demande spécifiés dans l'option **user** du paramètre de ressource personnalisée **subject**. Il vérifie les valeurs dans l'ordre indiqué dans le fichier de ressources personnalisées. Vous pouvez limiter la recherche de la clé API aux paramètres de requête ou aux en-têtes de requête en omettant l'option indésirable.

Dans cet exemple, Service Mesh recherche la clé API dans le paramètre de requête **user_key**. Si la clé API ne figure pas dans le paramètre de la requête, Service Mesh vérifie alors l'en-tête **user-key**.

Exemple de méthode d'authentification par clé API

```
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"
```

Si vous souhaitez que l'adaptateur examine un paramètre de requête ou un en-tête de requête différent, modifiez le nom en conséquence. Par exemple, pour vérifier la clé API dans un paramètre de requête nommé "key", remplacez **request.query_params["user_key"]** par **request.query_params["key"]**.

2.11.4.1.2. Méthode d'authentification de l'ID de l'application et de la paire de clés de l'application

Service Mesh recherche l'identifiant et la clé de l'application dans les paramètres de la requête et les en-têtes de la demande, comme spécifié dans l'option **properties** du paramètre de ressource personnalisée **subject**. La clé d'application est facultative. Il vérifie les valeurs dans l'ordre indiqué dans le fichier de ressources personnalisées. Vous pouvez limiter la recherche des informations d'identification aux paramètres de requête ou aux en-têtes de requête en n'incluant pas l'option "unwanted".

Dans cet exemple, Service Mesh recherche d'abord l'identifiant et la clé de l'application dans les paramètres de la requête, puis passe aux en-têtes de la requête si nécessaire.

Exemple de méthode d'authentification par ID d'application et paire de clés d'application

```
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"
```

Si vous souhaitez que l'adaptateur examine un paramètre de requête ou un en-tête de requête différent, modifiez le nom en conséquence. Par exemple, pour vérifier l'ID de l'application dans un paramètre de requête nommé **identification**, remplacez **request.query_params["app_id"]** par **request.query_params["identification"]**.

2.11.4.1.3. Méthode d'authentification OpenID

Pour utiliser le champ *OpenID Connect (OIDC) authentication method* , utilisez la valeur **properties** sur le champ **subject** pour définir **client_id**, et éventuellement **app_key**.

Vous pouvez manipuler cet objet à l'aide des méthodes décrites précédemment. Dans l'exemple de configuration ci-dessous, l'identifiant du client (ID de l'application) est extrait du jeton Web JSON (JWT) sous l'étiquette *azp*. Vous pouvez le modifier si nécessaire.

Exemple de méthode d'authentification OpenID

```
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"] | ""
```

Pour que cette intégration fonctionne correctement, l'OIDC doit toujours être effectué dans 3scale pour que le client soit créé dans le fournisseur d'identité (IdP). Vous devez créer une autorisation de demande pour le service que vous voulez protéger dans le même espace de noms que ce service. Le JWT est transmis dans l'en-tête **Authorization** de la requête.

Dans l'exemple **RequestAuthentication** défini ci-dessous, remplacez **issuer**, **jwksUri**, et **selector** comme il convient.

Exemple de politique OpenID

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: info
spec:
  selector:
    matchLabels:
      app: productpage
  jwtRules:
  - issuer: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak
    jwksUri: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak/protocol/openid-
connect/certs
```

2.11.4.1.4. Méthode d'authentification hybride

Vous pouvez choisir de ne pas appliquer une méthode d'authentification particulière et d'accepter toutes les informations d'identification valides pour l'une ou l'autre méthode. Si une clé API et une paire ID d'application/clé d'application sont fournies, Service Mesh utilise la clé API.

Dans cet exemple, Service Mesh vérifie la présence d'une clé API dans les paramètres de la requête, puis dans les en-têtes de la requête. S'il n'y a pas de clé API, il recherche alors un identifiant et une clé d'application dans les paramètres de la requête, puis dans les en-têtes de la requête.

Exemple de méthode d'authentification hybride

```
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"] | ""
```

2.11.5. 3scale Adapter metrics

L'adaptateur, par défaut, rapporte diverses mesures Prometheus qui sont exposées sur le port **8080** au point de terminaison /**metrics**. Ces métriques donnent un aperçu de la façon dont les interactions entre l'adaptateur et 3scale se déroulent. Le service est étiqueté pour être automatiquement découvert et scrappé par Prometheus.

2.11.6. vérification de l'adaptateur Istio 3scale

Vous voudrez peut-être vérifier si l'adaptateur 3scale Istio fonctionne comme prévu. Si votre adaptateur ne fonctionne pas, suivez les étapes suivantes pour résoudre le problème.

Procédure

1. Assurez-vous que le pod *3scale-adapter* fonctionne dans l'espace de noms du plan de contrôle Service Mesh :

```
oc get pods -n <istio-system>
```

2. Vérifiez que le pod *3scale-adapter* a imprimé des informations sur son démarrage, telles que sa version :

```
oc logs <istio-system>
```

3. Lors de l'exécution de requêtes vers les services protégés par l'intégration de l'adaptateur 3scale, essayez toujours les requêtes qui n'ont pas les bonnes informations d'identification et assurez-vous qu'elles échouent. Vérifiez les journaux de l'adaptateur 3scale pour recueillir des informations supplémentaires.

Ressources supplémentaires

- Inspecter les registres des gousses et des conteneurs .

2.11.7. liste de contrôle de dépannage de l'adaptateur Istio 3scale

En tant qu'administrateur installant l'adaptateur 3scale Istio, il y a un certain nombre de scénarios qui peuvent faire que votre intégration ne fonctionne pas correctement. Utilisez la liste suivante pour dépanner votre installation :

- Indentation YAML incorrecte.
- Sections YAML manquantes.
- J'ai oublié d'appliquer les changements dans le YAML au cluster.
- Oublié d'étiqueter les charges de travail de service avec la clé **service-mesh.3scale.net/credentials**.
- Oublié d'étiqueter les charges de travail de service avec **service-mesh.3scale.net/service-id** lorsque l'on utilise des gestionnaires qui ne contiennent pas de **service_id** afin qu'ils soient réutilisables par compte.
- La ressource personnalisée *Rule* pointe vers le mauvais gestionnaire ou les mauvaises ressources personnalisées d'instance, ou les références n'ont pas le suffixe de l'espace de noms correspondant.
- La section *Rule* custom resource **match** ne peut pas correspondre au service que vous configurez, ou elle pointe vers une charge de travail de destination qui n'est pas en cours d'exécution ou qui n'existe pas.
- Mauvais jeton d'accès ou URL pour le portail d'administration 3scale dans le gestionnaire.
- La section **params/subject/properties** de la ressource personnalisée *Instance* ne répertorie pas les bons paramètres pour **app_id**, **app_key** ou **client_id**, soit parce qu'ils spécifient le mauvais emplacement, comme les paramètres de requête, les en-têtes et les demandes d'autorisation, soit parce que les noms des paramètres ne correspondent pas aux requêtes utilisées pour les tests.
- Ne pas utiliser le générateur de configuration sans se rendre compte qu'il se trouve dans l'image du conteneur de l'adaptateur et qu'il a besoin de **oc exec** pour l'invoquer.

2.12. SUPPRESSION DU MAILLAGE DE SERVICES

AVERTISSEMENT

You are viewing documentation for a Red Hat OpenShift Service Mesh release that is no longer supported.

Les plans de contrôle Service Mesh version 1.0 et 1.1 ne sont plus pris en charge. Pour plus d'informations sur la mise à niveau de votre plan de contrôle Service Mesh, voir Mise à niveau de Service Mesh .

Pour plus d'informations sur l'état de l'assistance d'une version particulière de Red Hat OpenShift Service Mesh, consultez la page Cycle de vie du produit .

Pour supprimer Red Hat OpenShift Service Mesh d'une instance OpenShift Container Platform existante, supprimez le plan de contrôle avant de supprimer les opérateurs.

2.12.1. Suppression du plan de contrôle Red Hat OpenShift Service Mesh

Pour désinstaller Service Mesh d'une instance OpenShift Container Platform existante, vous devez d'abord supprimer le plan de contrôle Service Mesh et les opérateurs. Ensuite, vous exécutez des commandes pour supprimer les ressources résiduelles.

2.12.1.1. Suppression du plan de contrôle Service Mesh à l'aide de la console web

Vous pouvez supprimer le plan de contrôle Red Hat OpenShift Service Mesh en utilisant la console web.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Cliquez sur le menu **Project** et sélectionnez le projet dans lequel vous avez installé le plan de contrôle Service Mesh, par exemple **istio-system**.
3. Naviguez jusqu'à **Operators → Installed Operators**.
4. Cliquez sur **Service Mesh Control Plane** sous **Provided APIs**.
5. Cliquez sur le menu **ServiceMeshControlPlane**  .
6. Cliquez sur **Delete Service Mesh Control Plane**
7. Cliquez sur **Delete** dans la fenêtre de dialogue de confirmation pour supprimer **ServiceMeshControlPlane**.

2.12.1.2. Suppression du plan de contrôle Service Mesh à l'aide de la CLI

Vous pouvez supprimer le plan de contrôle Red Hat OpenShift Service Mesh en utilisant le CLI. Dans cet exemple, **istio-system** est le nom du projet de plan de contrôle.

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform.
2. Exécutez la commande suivante pour supprimer la ressource **ServiceMeshMemberRoll**.

```
$ oc delete smmr -n istio-system default
```

3. Exécutez cette commande pour récupérer le nom de la version installée de **ServiceMeshControlPlane**:

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

4. Remplacez **<name_of_custom_resource>** par le résultat de la commande précédente et exécutez cette commande pour supprimer la ressource personnalisée :

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

2.12.2. Retrait des opérateurs installés

Vous devez supprimer les opérateurs pour réussir à supprimer Red Hat OpenShift Service Mesh. Après avoir supprimé l'opérateur Red Hat OpenShift Service Mesh, vous devez supprimer l'opérateur Kiali, l'opérateur Red Hat OpenShift distributed tracing platform et l'opérateur OpenShift Elasticsearch.

2.12.2.1. Retrait des opérateurs

Suivez cette procédure pour supprimer les opérateurs qui composent Red Hat OpenShift Service Mesh. Répétez les étapes pour chacun des opérateurs suivants.

- Red Hat OpenShift Service Mesh
- Kiali
- Plateforme de traçage distribuée Red Hat OpenShift
- OpenShift Elasticsearch

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. À partir de la page **Operators → Installed Operators**, faites défiler ou tapez un mot-clé dans la page **Filter by name** pour trouver chaque opérateur. Cliquez ensuite sur le nom de l'opérateur.
3. Sur la page **Operator Details**, sélectionnez **Uninstall Operator** dans le menu **Actions**. Suivez les instructions pour désinstaller chaque opérateur.

2.12.2.2. Nettoyer les ressources de l'opérateur

Suivez cette procédure pour supprimer manuellement les ressources restantes après avoir supprimé l'opérateur Red Hat OpenShift Service Mesh à l'aide de la console web OpenShift Container Platform.

Conditions préalables

- Un compte avec accès à l'administration du cluster.
- Accès à la CLI OpenShift (**oc**).

Procédure

1. Connectez-vous au CLI de OpenShift Container Platform en tant qu'administrateur de cluster.
2. Exécutez les commandes suivantes pour nettoyer les ressources après avoir désinstallé les opérateurs. Si vous avez l'intention de continuer à utiliser Jaeger en tant que service autonome sans service mesh, ne supprimez pas les ressources de Jaeger.

NOTE

Les opérateurs sont installés par défaut dans l'espace de noms **openshift-operators**. Si vous avez installé les opérateurs dans un autre espace de noms, remplacez **openshift-operators** par le nom du projet dans lequel l'opérateur Red Hat OpenShift Service Mesh a été installé.

```
$ oc delete validatingwebhookconfiguration/openshift-operators.servicemesh-resources.maistra.io
```

```
$ oc delete mutatingwebhookconfiguration/openshift-operators.servicemesh-resources.maistra.io
```

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

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

```
$ oc delete clusterrole istio-view istio-edit
```

```
$ oc delete clusterrole jaegers.jaegertracing.io-v1-admin jaegers.jaegertracing.io-v1-crdview jaegers.jaegertracing.io-v1-edit jaegers.jaegertracing.io-v1-view
```

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

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

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

```
$ oc delete crds jaegers.jaegertracing.io
```

```
oc delete svc admission-controller -n <operator-project>
```

```
oc delete project <istio-system-project>
```