OpenShift Container Platform 4.7

Service Mesh

Service Mesh のインストール、使用法、およびリリースノート

Last Updated: 2023-02-24

OpenShift Container Platform 4.7 Service Mesh

Service Mesh のインストール、使用法、およびリリースノート

法律上の通知

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本書では、OpenShift Container Platform で Service Mesh を使用する方法について説明します。

目次

第1章 SERVICE MESH 2.X **3**
1.1. OPENSHIFT SERVICE MESH について 3
1.2. SERVICE MESH リリースノート 4
1.3. SERVICE MESH について 30
1.4. サービスメッシュのデプロイメントモデル 37
1.5. SERVICE MESH と ISTIO の相違点 38
1.6. SERVICE MESH のインストールの準備 44
1.7. OPERATOR のインストール 46
1.8. SERVICEMESHCONTROLPLANE の作成 48
1.9. サービスメッシュへのサービスの追加 55
1.10. サイドカーコンテナーの挿入の有効化 65
1.11. SERVICE MESH のアップグレード 69
1.12. ユーザーおよびプロファイルの管理 86
1.13. セキュリティー 89
1.14. SERVICE MESH でのトラフィックの管理 101
1.15. メトリックス、ログ、およびトレース 115
1.16. パフォーマンスおよびスケーラビリティー 125
1.17. 実稼働環境の SERVICE MESH の設定 127
1.18. サービスメッシュの接続 129
1.19. 拡張 160
1.20. 3SCALE WEBASSEMBLY モジュールの使用 173
1.21. 3SCALE ISTIO アダプターの使用 192
1.22. サービスメッシュのトラブルシューティング 204
1.23. ENVOY プロキシーのトラブルシューティング 213
1.24. SERVICE MESH コントロールプレーン設定の参照 218
1.25. KIALI 設定リファレンス 231
1.26. JAEGER 設定リファレンス 236
1.27. SERVICE MESH のアンインストール 268

第2章 SERVICE MESH 1.X **271**
2.1. SERVICE MESH リリースノート 271
2.2. SERVICE MESH について 290
2.3. SERVICE MESH と ISTIO の相違点 296
2.4. SERVICE MESH のインストールの準備 300
2.5. SERVICE MESH のインストール 303
2.6. SERVICE MESH のセキュリティーのカスタマイズ 315
2.7. トラフィック管理 321
2.8. SERVICE MESH へのアプリケーションのデプロイ 333
2.9. データの可視化および可観測性 346
2.10. カスタムリソース 349
2.11. 3SCALE ISTIO アダプターの使用 369
2.12. SERVICE MESH の削除 379

第1章 SERVICE MESH 2.X

1.1. OPENSHIFT SERVICE MESH について

注記

Red Hat OpenShift Service Mesh は OpenShift Container Platform とは異なる頻度でリリースされ、Red Hat OpenShift Service Mesh Operator は **ServiceMeshControlPlane** の複数のバージョンのデプロイをサポートしているため、Service Mesh のドキュメントでは、本製品のマイナーバージョン用に個別のドキュメントセットを維持していません。現在のドキュメントセットは、特定のトピックまたは特定の機能でバージョン固有の制限がない限り、現在サポートされている Service Mesh のすべてのバージョンに適用されます。

Red Hat OpenShift Service Mesh のライフサイクルとサポートされるプラットフォームに関する追加情報については、Platform Life Cycle Policy を参照してください。

1.1.1. Red Hat OpenShift Service Mesh の概要

Red Hat OpenShift Service Mesh は、アプリケーションで一元化された制御ポイントを作成して、マイクロサービスアーキテクチャーのさまざまな問題に対応します。OpenShift Service Mesh はアプリケーションコードを変更せずに、既存の分散アプリケーションに透過的な層を追加します。

マイクロサービスアーキテクチャーは、エンタープライズアプリケーションの作業をモジュールサービスに分割するので、スケーリングとメンテナンスが容易になります。ただし、マイクロサービスアーキテクチャー上に構築されるエンタープライズアプリケーションはサイズも複雑性も増すため、マイクロサービスアーキテクチャーの理解と管理は困難です。Service Mesh は、サービス間のトラフィックをキャプチャーしたり、インターセプトしたりして、他のサービスへの新規要求を変更、リダイレクト、または作成することによってこれらのアーキテクチャーの問題に対応できます。

オープンソースの Istio project をベースにする Service Mesh では、検出、負荷分散、サービス間の認証、障害復旧、メトリクス、およびモニタリングを提供する、デプロイされたサービスのネットワークを簡単に作成できます。サービスメッシュは、A/B テスト、カナリアリリース、レート制限、アクセス制御、エンドツーエンド認証を含む、より複雑な運用機能を提供します。

1.1.2. コア機能

Red Hat OpenShift Service Mesh は、サービスのネットワーク全体で多数の主要機能を均一に提供します。

- トラフィック管理: サービス間でトラフィックおよび API 呼び出しのフローを制御し、呼び出しの安定度を高め、不利な条件下でもネットワークの堅牢性を維持します。
- サービス ID とセキュリティー: メッシュのサービスを検証可能な ID で指定でき、サービスのトラフィックがさまざまな信頼度のネットワークに送られる際にそのトラフィックを保護する機能を提供します。
- ポリシーの適用: サービス間の対話に組織のポリシーを適用し、アクセスポリシーが適用され、リソースはコンシューマー間で均等に分散されるようにします。ポリシー変更は、アプリケーションコードを変更するのではなく、メッシュを設定して行います。
- **Telemetry**: サービス間の依存関係やそれらの間のトラフィックの性質やフローを理解するのに役立ち、問題を素早く特定できます。

1.2. SERVICE MESH リリースノート

1.2.1. 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

1.2.2. 新機能および機能拡張

今回のリリースでは、以下のコンポーネントおよび概念に関連する拡張機能が追加されました。

1.2.2.1. Red Hat OpenShift Service Mesh バージョン 2.2.3 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応し、OpenShift Container Platform 4.9 以降でサポートされます。

1.2.2.1.1. Red Hat OpenShift Service Mesh バージョン 2.2.3 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.12.9
Envoy プロキシー	1.20.8
Jaeger	1.36
Kiali	1.48.3

1.2.2.2. Red Hat OpenShift Service Mesh バージョン 2.2.2 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応し、OpenShift Container Platform 4.9 以降でサポートされます。

1.2.2.2.1. Red Hat OpenShift Service Mesh バージョン 2.2.2 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.12.7
Envoy プロキシー	1.20.6
Jaeger	1.36
Kiali	1.48.2-1

1.2.2.2.2. ルートラベルのコピー

この機能強化により、アノテーションのコピーに加えて、OpenShift ルートの特定のラベルをコピーできます。Red Hat OpenShift Service Mesh は、Istio Gateway リソースに存在するすべてのラベルとアノテーション (kubectl.kubernetes.io で始まるアノテーションを除く) を管理対象の OpenShift Route リソースにコピーします。

1.2.2.3. Red Hat OpenShift Service Mesh バージョン 2.2.1 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応し、OpenShift Container Platform 4.9 以降でサポートされます。

1.2.2.3.1. Red Hat OpenShift Service Mesh バージョン 2.2.1 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.12.7
Envoy プロキシー	1.20.6
Jaeger	1.34.1
Kiali	1.48.2-1

1.2.2.4. Red Hat OpenShift Service Mesh 2.2 の新機能

このリリースの Red Hat OpenShift Service Mesh は、新しい機能と拡張機能を追加し、OpenShift Container Platform 4.9 以降でサポートされています。

1.2.2.4.1. Red Hat OpenShift Service Mesh バージョン 2.2 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.12.7
Envoy プロキシー	1.20.4
Jaeger	1.34.1
Kiali	1.48.0.16

1.2.2.4.2. WasmPlugin API

このリリースでは、**WasmPlugin** API のサポートが追加され、**ServiceMeshExtentionAPI** が廃止されました。

1.2.2.4.3. ROSA サポート

このリリースでは、マルチクラスターフェデレーションを含む Red Hat OpenShift on AWS(ROSA) のサービスメッシュサポートが導入されています。

1.2.2.4.4. istio-node DaemonSet の名前が変更されました

このリリースでは、**istio-node** DaemonSet の名前が **istio-cni-node** に変更され、アップストリーム Istio の名前と同じになりました。

1.2.2.4.5. エンボイサイドカーネットワークの変更

Istio 1.10 は、デフォルトで **lo** ではなく **eth0** を使用してトラフィックをアプリケーションコンテナーに送信するように Envoy を更新しました。

1.2.2.4.6. Service Mesh コントロールプレーン 1.1

このリリースは、すべてのプラットフォームでの Service Mesh 1.1 に基づく Service Mesh コントロールプレーンのサポートの終了を示します。

1.2.2.4.7. Istio 1.12 サポート

Service Mesh 2.2 は Istio 1.12 に基づいており、新機能と製品の機能強化をもたらします。多くの Istio 1.12 機能がサポートされていますが、サポートされていない次の機能に注意する必要があります。

- AuthPolicy ドライランはテクノロジープレビュー機能です。
- gRPC Proxyless Service Mesh は、テクノロジープレビュー機能です。
- Telemetry API は、テクノロジープレビュー機能です。
- ディスカバリーセレクターはサポート対象外の機能です。
- 外部コントロールプレーンはサポート対象外の機能です。
- ゲートウェイインジェクションはサポート対象外の機能です。

1.2.2.4.8. Kubernetes Gateway API

Kubernetes Gateway API は、デフォルトで無効になっているテクノロジープレビュー機能です。

この機能を有効にするには、**ServiceMeshControlPlane** で **Istiod** コンテナーに次の環境変数を設定します。

```
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
```

ゲートウェイ API リスナーでのルート接続を制限するには、**SameNamespace** または **All** 設定を使用します。Istio は、**listeners.allowedRoutes.namespaces** のラベルセレクターの使用を無視し、デフォルトの動作 (**SameNamespace**) に戻します。

1.2.2.5. Red Hat OpenShift Service Mesh 2.1.5.1 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応し、OpenShift Container Platform 4.9 以降でサポートされます。

1.2.2.5.1. Red Hat OpenShift Service Mesh バージョン 2.1.5.1 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.9.9
Envoy プロキシー	1.17.5
Jaeger	1.36
Kiali	1.36.13

1.2.2.6. Red Hat OpenShift Service Mesh 2.1.5 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応し、OpenShift Container Platform 4.9 以降でサポートされます。

1.2.2.6.1. Red Hat OpenShift Service Mesh バージョン 2.1.5 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.9.9
Envoy プロキシー	1.17.1
Jaeger	1.36
Kiali	1.36.12-1

1.2.2.7. Red Hat OpenShift Service Mesh 2.1.4 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.2.7.1. Red Hat OpenShift Service Mesh バージョン 2.1.4 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.9.9

コンポーネント	バージョン
Envoy プロキシー	1.17.1
Jaeger	1.30.2
Kiali	1.36.12-1

1.2.2.8. Red Hat OpenShift Service Mesh 2.1.3 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.2.8.1. Red Hat OpenShift Service Mesh バージョン 2.1.3 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.9.9
Envoy プロキシー	1.17.1
Jaeger	1.30.2
Kiali	1.36.10-2

1.2.2.9. Red Hat OpenShift Service Mesh 2.1.2.1 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.2.9.1. Red Hat OpenShift Service Mesh バージョン 2.1.2.1 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.9.9
Envoy プロキシー	1.17.1
Jaeger	1.30.2
Kiali	1.36.9

1.2.2.10. Red Hat OpenShift Service Mesh 2.1.2 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

このリリースでは、Red Hat 分散トレースプラットフォーム Operator がデフォルトで **openshift-distributed-tracing** namespace にインストールされるようになりました。以前のリリースでは、デフォルトのインストールは **openshift-operator** namespace にありました。

1.2.2.10.1. Red Hat OpenShift Service Mesh バージョン 2.1.2 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.9.9
Envoy プロキシー	1.17.1
Jaeger	1.30.1
Kiali	1.36.8

1.2.2.11. Red Hat OpenShift Service Mesh 2.1.1 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

本リリースでは、ネットワークポリシーの自動作成を無効にする機能も追加されています。

1.2.2.11.1. Red Hat OpenShift Service Mesh バージョン 2.1.1 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.9.9
Envoy プロキシー	1.17.1
Jaeger	1.24.1
Kiali	1.36.7

1.2.2.11.2. ネットワークポリシーの無効化

Red Hat OpenShift Service Mesh は、Service Mesh コントロールプレーンおよびアプリケーションネームスペースで多数の **NetworkPolicies** リソースを自動的に作成し、管理します。これは、アプリケーションとコントロールプレーンが相互に通信できるようにするために使用されます。

NetworkPolicies リソースの自動作成および管理を無効にする場合 (例: 会社のセキュリティーポリシーを適用する場合など) には、これを実行できます。**ServiceMeshControlPlane** を編集して **spec.security.manageNetworkPolicy** 設定を **false**に設定できます。

注記

spec.security.manageNetworkPolicy を無効にすると、Red Hat OpenShift Service Mesh は、**NetworkPolicy** オブジェクトをひとつも作成しません。システム管理者は、ネットワークを管理し、この原因の問題を修正します。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → Installed Operators** をクリックします。
2. Project メニューから、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
3. Red Hat OpenShift Service Mesh Operator をクリックします。**Istio Service Mesh Control Plane** 列で、**ServiceMeshControlPlane** の名前 (**basic-install** など) をクリックします。
4. **Create ServiceMeshControlPlane Details** ページで、**YAML** をクリックして設定を変更します。
5. 以下の例のように、**ServiceMeshControlPlane** フィールド **spec.security.manageNetworkPolicy** を **false** に設定します。

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

6. **Save** をクリックします。

1.2.2.12. Red Hat OpenShift Service Mesh 2.1 の新機能および機能拡張

Red Hat OpenShift Service Mesh の本リリースでは、Istio 1.9.8、Envoy Proxy 1.17.1、Jaeger 1.24.1、および Kiali 1.36.5 のサポートと新機能および機能拡張が OpenShift Container Platform 4.6 EUS、4.7、4.8、および 4.9 で追加されました。

1.2.2.12.1. Red Hat OpenShift Service Mesh バージョン 2.1 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.9.6
Envoy プロキシー	1.17.1
Jaeger	1.24.1
Kiali	1.36.5

1.2.2.12.2. Service Mesh のフェデレーション

サービスメッシュをフェデレーションできるように新規のカスタムリソース定義 (CRD) が追加されました。サービスメッシュは、同じクラスター内または異なる OpenShift クラスター間でフェデレーションすることができます。これらの新規リソースには以下が含まれます。

- **ServiceMeshPeer**: ゲートウェイ設定、ルート信頼証明書設定、ステータスフィールドなど、別のサービスメッシュでのフェデレーションを定義します。フェデレーションされたメッシュのペアでは、各メッシュは独自の **ServiceMeshPeer** リソースを個別に定義します。
- **ExportedServiceMeshSet**: ピアメッシュのインポートに利用できる特定の **ServiceMeshPeer** サービスを定義します。
- **ImportedServiceSet**: ピアメッシュからインポートする特定の **ServiceMeshPeer** のサービスを定義します。これらのサービスは、ピアの **ExportedServiceMeshSet** リソースで利用できるようにする必要もあります。

Service Mesh Federation は、Red Hat OpenShift Service on AWS (ROSA)、Azure Red Hat OpenShift (ARO)、または OpenShift Dedicated (OSD) のクラスター間ではサポートされていません。

1.2.2.12.3. OVN-Kubernetes Container Network Interface(CNI) の一般提供

OVN-Kubernetes Container Network Interface(CNI) は、以前は Red Hat OpenShift Service Mesh 2.0.1 のテクノロジープレビュー機能として導入されましたが、OpenShift Container Platform 4.7.32、OpenShift Container Platform 4.8.12、および OpenShift Container Platform 4.9 で使用できるように Red Hat OpenShift Service Mesh 2.1 および 2.0.x で一般提供されています。

1.2.2.12.4. Service Mesh WebAssembly(WASM) 拡張

ServiceMeshExtensions カスタムリソース定義 (CRD) は、最初に 2.0 でテクノロジープレビュー機能として導入され、今回のバージョンで一般公開されました。CRD を使用して独自のプラグインを構築できますが、Red Hat では独自に作成したプラグインはサポートしていません。

Mixer はサービスメッシュ 2.1 で完全に削除されました。Mixer が有効な場合は、Service Mesh 2.0.x リリースから 2.1 へのアップグレードは、ブロックされます。Mixer プラグインは WebAssembly 拡張に移植する必要があります。

1.2.2.12.5. 3scale WebAssembly Adapter(WASM)

Mixer が正式に削除されたため、OpenShift 3scale mixer アダプターは、Service Mesh 2.1 ではサポート対象外となっています。Service Mesh 2.1 にアップグレードする前に、Mixer ベースの 3scale アダプターと追加の Mixer プラグインを削除します。次に、**ServiceMeshExtension** リソースを使用して、新しい 3scale WebAssembly アダプターを Service Mesh 2.1+ で手動でインストールして設定します。

3scale 2.11 では、**WebAssembly** に基づく更新された Service Mesh の統合が導入されました。

1.2.2.12.6. Istio 1.9 サポート

Service Mesh 2.1 は Istio 1.9 をベースとしており、製品の新機能および機能拡張が数多く追加されました。Istio 1.9 の大半の機能がサポートされていますが、以下の例外に注意してください。

- 仮想マシンの統合はまだサポートされていません。
- Kubernetes Gateway API はまだサポートされていません。
- WebAssembly HTTP フィルターのリモートフェッチおよびロードはサポートされていません。
- Kubernetes CSR API を使用したカスタム CA 統合はまだサポートされていません。

- トラフィック監視要求の分類機能はテクノロジープレビュー機能です。
- Authorization ポリシーの CUSTOM アクションによる外部承認システムとの統合はテクノロジープレビュー機能です。

1.2.2.12.7. Service Mesh Operator のパフォーマンス向上

各 **ServiceMeshControlPlane** の調整の終了時に以前のリソースのプルーニングに使用する期間が短縮されました。これにより、**ServiceMeshControlPlane** のデプロイメントにかかる時間が短縮され、既存の SMCP に適用される変更がこれまでよりも早く有効になります。

1.2.2.12.8. Kiali の更新

Kiali 1.36 には、以下の機能と拡張機能が含まれています。

- Service Mesh のトラブルシューティング機能
  - コントロールプレーンおよびゲートウェイの監視
  - プロキシーの同期ステータス
  - Envoy 設定ビュー
  - Envoy プロキシーおよびアプリケーションログのインターリーブを示す統合ビュー
- フェデレーションされたサービスメッシュビューをサポートする namespace およびクラスターボックス
- 新しい検証、ウィザード、および分散トレースの機能拡張

1.2.2.13. Red Hat OpenShift Service Mesh 2.0.11.1 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応し、OpenShift Container Platform 4.9 以降でサポートされます。

1.2.2.13.1. Red Hat OpenShift Service Mesh バージョン 2.0.11.1 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.6.14
Envoy プロキシー	1.14.5
Jaeger	1.36
Kiali	1.24.17

1.2.2.14. Red Hat OpenShift Service Mesh 2.0.11 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応し、OpenShift Container Platform 4.9 以降でサポートされます。

1.2.2.14.1. Red Hat OpenShift Service Mesh バージョン 2.0.11 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.6.14
Envoy プロキシー	1.14.5
Jaeger	1.36
Kiali	1.24.16-1

1.2.2.15. Red Hat OpenShift Service Mesh 2.0.10 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.2.15.1. Red Hat OpenShift Service Mesh バージョン 2.0.10 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.6.14
Envoy プロキシー	1.14.5
Jaeger	1.28.0
Kiali	1.24.16-1

1.2.2.16. Red Hat OpenShift Service Mesh 2.0.9 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.2.16.1. Red Hat OpenShift Service Mesh バージョン 2.0.9 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.6.14
Envoy プロキシー	1.14.5
Jaeger	1.24.1
Kiali	1.24.11

1.2.2.17. Red Hat OpenShift Service Mesh 2.0.8 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、バグ修正に対応しています。

1.2.2.18. Red Hat OpenShift Service Mesh 2.0.7.1 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) に対応しています。

1.2.2.18.1. Red Hat OpenShift Service Mesh が URI フラグメントを処理する方法の変更

Red Hat OpenShift Service Mesh には、リモートで悪用可能な脆弱性 CVE-2021-39156 が含まれており、URI パスにフラグメント (URI の末尾が # 文字で始まるセクション) を含む HTTP リクエストが Istio URI パスベースの認証ポリシーを無視する可能性があります。たとえば、Istio 認証ポリシーは URI パス **/user/profile** に送信される要求を拒否します。脆弱なバージョンでは、URI パス **/user/profile#section1** のリクエストは、deny ポリシーと、(正規化された URI **path /user/profile%23section1** を使用する) バックエンドへのルートを無視するため、セキュリティーのインシデントにつながる可能性があります。

DENY アクションおよび **operation.paths**、または ALLOW アクションおよび **operation.notPaths** で認可ポリシーを使用する場合は、この脆弱性の影響を受けます。

軽減策により、リクエストの URI の断片部分が、承認とルーティングの前に削除されます。これにより、URI のフラグメントを持つ要求が、フラグメントの一部のない URI をベースとする承認ポリシーが無視できなくなります。

軽減策の新しい動作からオプトインするには、URI の fragment セクションが保持されます。**ServiceMeshControlPlane** を設定して URI フラグメントを保持することができます。

> **警告**
>
> 新しい動作を無効にすると、上記のようにパスを正規化し、安全でないと見なされます。URI フラグメントを保持することを選択する前に、セキュリティーポリシーでこれに対応していることを確認してください。

ServiceMeshControlPlane の変更例

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

1.2.2.18.2. 認証ポリシーに必要な更新

Istio はホスト名自体とすべての一致するポートの両方にホスト名を生成します。たとえば、httpbin.foo

のホストの仮想サービスまたはゲートウェイは、httpbin.foo and httpbin.foo:*に一致する設定を生成します。ただし、完全一致許可ポリシーは、**hosts** または **notHosts** フィールドに指定された完全一致文字列にのみ一致します。

ルールの正確な文字列比較を使用して hosts または notHosts を決定する **AuthorizationPolicy** リソースがある場合、クラスターは影響を受けます。

完全一致ではなく接頭辞一致を使用するように、認証ポリシー ルール を更新する必要があります。たとえば、最初の **AuthorizationPolicy** の例で **hosts: ["httpbin.com"]** を **hosts: ["httpbin.com:*"]** に置き換えます。

接頭辞一致を使用した最初の AuthorizationPolicy の例

```
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:*"]
```

接頭辞一致を使用した 2 つ目の AuthorizationPolicy の例

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

1.2.2.19. Red Hat OpenShift Service Mesh 2.0.7 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.2.20. Red Hat OpenShift Dedicated および Microsoft Azure Red Hat OpenShift 上の Red Hat OpenShift Service Mesh

Red Hat OpenShift Service Mesh は、Red Hat OpenShift Dedicated および Microsoft Azure Red Hat OpenShift 経由でサポートされるようになりました。

1.2.2.21. Red Hat OpenShift Service Mesh 2.0.6 の新機能

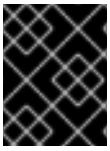
Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.2.22. Red Hat OpenShift Service Mesh 2.0.5 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.2.23. Red Hat OpenShift Service Mesh 2.0.4 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。



重要

CVE-2021-29492 および CVE-2021-31920 に対応するために、手動による手順を完了する必要があります。

1.2.2.23.1. CVE-2021-29492 および CVE-2021-31920 で必要な手動による更新

Istio にはリモートで悪用可能な脆弱性があり、複数のスラッシュまたはエスケープされたスラッシュ文字 (**%2F** または **%5C**) を持つ HTTP リクエストパスが、パスベースの認証ルールが使用される場合に Istio 認証ポリシーを無視する可能性があります。

たとえば、Istio クラスター管理者が、パス **/admin** での要求を拒否する認証 DENY ポリシーを定義すると仮定します。URL パス **//admin** に送信される要求は、認証ポリシーによって拒否されません。

RFC 3986 に応じて、複数のスラッシュを持つパス **//admin** は、**/admin** とは異なるパスとして処理される必要があります。ただし、一部のバックエンドサービスは、複数のスラッシュを単一のスラッシュにマージして URL パスを正規化することを選択します。これにより、認証ポリシーがバイパスされ (**//admin** は **/admin** に一致しません)、ユーザーはバックエンドのパス (**/admin**) でリソースにアクセスできます。これにより、セキュリティーのインシデントを示されます。

ALLOW action + notPaths フィールドまたは **DENY action + paths field** パターンを使用する認証ポリシーがある場合、クラスターはこの脆弱性の影響を受けます。これらのパターンは、予期しないポリシーのバイパスに対して脆弱です。

以下の場合、クラスターはこの脆弱性の影響を受けません。

- 認証ポリシーがありません。
- 認証ポリシーは、**paths** または **notPaths** フィールドを定義しません。
- 認証ポリシーは、**ALLOW action + paths** フィールドまたは **DENY action + notPaths** フィールドのパターンを使用します。これらのパターンは、ポリシーのバイパスではなく、予期しない拒否を生じさせる可能性があります。このような場合、アップグレードは任意です。

注記

パスの正規化向けの Red Hat OpenShift Service Mesh 設定の場所は、Istio 設定とは異なります。

1.2.2.23.2. パスの正規化設定の更新

Istio 認証ポリシーは、HTTP リクエストの URL パスをベースとする場合があります。URI の正規化として知られる パスの正規化 は、正規化されたパスを標準の方法で処理できるように、受信要求のパスを変更し、標準化します。構文の異なるパスは、パスの正規化後と同一になる場合があります。

Istio は、認証ポリシーに対して評価し、要求をルーティングする前の、要求パスでの以下の正規化スキームをサポートします。

表1.1 正規化スキーム

オプション	説明	例	注記
NONE	正規化は行われません。Envoy が受信する内容はそのまますべて、どのバックエンドサービスにも完全に転送されます。	**../%2fa../b** は認証ポリシーによって評価され、サービスに送信されます。	この設定は CVE-2021-31920 に対して脆弱です。
BASE	現時点で、これは Istio の **デフォルト** インストールで使用されるオプションです。これにより、Envoy プロキシーで **normalize_path** オプションが適用されます。これは、追加の正規化において RFC 3986 に従い、バックスラッシュをフォワードスラッシュに変換します。	**/a/../b** は **/b** に正規化されます。**\da** は **/da** に正規化されます。	この設定は CVE-2021-31920 に対して脆弱です。
MERGE_SLASHES	スラッシュは BASE の正規化後にマージされます。	**/a//b** は **/a/b** に正規化されます。	この設定に対して更新を行い、CVE-2021-31920 のリスクを軽減します。
DECODE_AND_MERGE_SLASHES	デフォルトですべてのトラフィックを許可する場合の最も厳密な設定です。この設定の場合、認証ポリシーのルートを詳細にテストする必要がある点に注意してください。パーセントでエンコードされた スラッシュおよびバックスラッシュ文字 (**%2F**、**%2f**、**%5C** および **%5c**) は、**MERGE_SLASHES** の正規化の前に / または \ にデコードされます。	**/a%2fb** は **/a/b** に正規化されます。	この設定に対して更新を行い、CVE-2021-31920 のリスクを軽減します。この設定はより安全ですが、アプリケーションが破損する可能性があります。実稼働環境にデプロイする前にアプリケーションをテストします。

正規化アルゴリズムは以下の順序で実行されます。

1. パーセントでデコードされた **%2F**、**%2f**、**%5C** および **%5c**。
2. Envoy の **normalize_path** オプションで実装された RFC 3986 およびその他の正規化。
3. スラッシュをマージします。

> **警告**
>
> これらの正規化オプションは HTTP 標準および一般的な業界プラクティスの推奨事項を表しますが、アプリケーションは独自に選択したいずれかの方法で URL を解釈する場合があります。拒否ポリシーを使用する場合には、アプリケーションの動作を把握している必要があります。

1.2.2.23.3. パスの正規化設定の例

Envoy がバックエンドサービスの期待値に一致するように要求パスを正規化することは、システムのセキュリティーを保護する上で非常に重要です。以下の例は、システムを設定するための参考として使用できます。正規化された URL パス、または **NONE** が選択されている場合は元の URL パスは以下のようになります。

1. 認証ポリシーの確認に使用されます。
2. バックエンドアプリケーションに転送されます。

表1.2 設定例

アプリケーションの条件	選択内容
プロキシーを使用して正規化を行う。	**BASE**、**MERGE_SLASHES**、または **DECODE_AND_MERGE_SLASHES**
RFC 3986 に基づいて要求パスを正規化し、スラッシュをマージしない。	**BASE**
RFC 3986 に基づいて要求パスを正規化し、スラッシュをマージするが、パーセントでエンコードされた スラッシュをデコードしない。	**MERGE_SLASHES**
RFC 3986 に基づいて要求パスを正規化し、パーセントでエンコードされた スラッシュをデコードし、スラッシュをマージする。	**DECODE_AND_MERGE_SLASHES**
RFC 3986 と互換性のない方法で要求パスを処理する。	**NONE**

1.2.2.23.4. パスを正規化するための SMCP の設定

Red Hat OpenShift Service Mesh のパスの正規化を設定するには、**ServiceMeshControlPlane** で以下を指定します。設定例を使用すると、システムの設定を判断するのに役立ちます。

SMCP v2 pathNormalization

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

1.2.2.23.5. ケース正規化 (case normalization) の設定

環境によっては、大文字と小文字を区別しない場合の比較用に 2 つのパスを認証ポリシーに用意すると便利な場合があります。たとえば、**https://myurl/get** と **https://myurl/GeT** を同等なものとして扱います。このような場合には、以下に示されている **EnvoyFilter** を使用できます。このフィルターは、比較用に使用されるパスとアプリケーションに表示されるパスの両方を変更します。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

EnvoyFilter をファイルに保存して、以下のコマンドを実行します。

```
$ 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
```

1.2.2.24. Red Hat OpenShift Service Mesh 2.0.3 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

また、本リリースには以下の新機能があります。

- 指定された Service Mesh コントロールプレーン namespace から情報を収集する **must-gather** データ収集ツールにオプションが追加されました。詳細は、OSSM-351 を参照してください。
- 数百の namespace が含まれる Service Mesh コントロールプレーンのパフォーマンスの向上

1.2.2.25. Red Hat OpenShift Service Mesh 2.0.2 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、IBM Z および IBM Power Systems のサポートが追加されました。また、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.2.26. Red Hat OpenShift Service Mesh 2.0.1 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

1.2.2.27. Red Hat OpenShift Service Mesh 2.0 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、Istio 1.6.5、Jaeger 1.20.0、Kiali 1.24.2、3scale Istio Adapter 2.0 および OpenShift Container Platform 4.6 のサポートが追加されました。

また、本リリースには以下の新機能があります。

- Service Mesh コントロールプレーンのインストール、アップグレード、および管理を単純化します。
- Service Mesh コントロールプレーンのリソース使用量と起動時間を短縮します。
- ネットワークのコントロールプレーン間の通信を削減することで、パフォーマンスが向上します。
  - Envoy の Secret Discovery Service (SDS) のサポートが追加されました。SDS は、Envoy サイドカープロキシーにシークレットを提供するためのより安全で効率的なメカニズムです。
- よく知られているセキュリティーリスクがある Kubernetes シークレットを使用する必要性がなくなります。
- プロキシーが新しい証明書を認識するのに再起動を必要としなくなったため、証明書のローテーション時にパフォーマンスが向上します。
  - WebAssembly 拡張を使用してビルドされる Istio の Telemetry v2 アーキテクチャーのサポートを追加します。この新しいアーキテクチャーにより、パフォーマンスが大幅に改善されました。
  - ServiceMeshControlPlane リソースを簡素化された設定を含む v2 に更新し、Service Mesh コントロールプレーンの管理を容易にします。
  - WebAssembly 拡張を テクノロジープレビュー として導入します。

1.2.3. テクノロジープレビュー

現在、今回のリリースに含まれる機能にはテクノロジープレビューのものがあります。これらの実験的機能は、実稼働環境での使用を目的としていません。

重要

テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

1.2.3.1. Istio の互換性およびサポート表

以下の表では、機能は以下のステータスでマークされています。

- **TP**: テクノロジープレビュー機能
- **GA**: 一般公開機能

これらの機能に関しては、Red Hat カスタマーポータルの以下のサポート範囲を参照してください。

表1.3 Istio の互換性およびサポート表

機能	Istio バージョン	サポートステータス	説明
holdApplicationUntilProxyStarts	1.7	TP	プロキシーの実行までアプリケーションコンテナーの起動をブロックします。
DNS キャプチャー	1.8	GA	デフォルトでは有効

1.2.4. 非推奨および削除された機能

以前のリリースで利用可能であった一部の機能が非推奨になるか、または削除されました。

非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

本製品では、削除機能が除去されています。

1.2.4.1. 非推奨になった Red Hat OpenShift Service Mesh 2.2 の機能

ServiceMeshExtension API は、リリース 2.2 で非推奨になり、今後のリリースで削除される予定です。**ServiceMeshExtension** API はリリース 2.2 でも引き続きサポートされていますが、お客様は新しい **WasmPluginAPI** への移行を開始する必要があります。

1.2.4.2. Red Hat OpenShift Service Mesh 2.2 から削除された機能

このリリースは、すべてのプラットフォームでの Service Mesh 1.1 に基づく Service Mesh コントロールプレーンのサポートの終了を示します。

1.2.4.3. Red Hat OpenShift Service Mesh 2.1 の新機能

Service Mesh 2.1 では、Mixer コンポーネントが削除されます。バグ修正およびサポートは、Service Mesh 2.0 の最後のライフサイクルで提供されます。

Mixer プラグインが有効な場合は、Service Mesh 2.0.x リリースから 2.1 へのアップグレードは続行されません。Mixer プラグインは、WebAssembly 拡張に移植する必要があります。

1.2.4.4. 非推奨になった Red Hat OpenShift Service Mesh 2.0 の機能

Mixer コンポーネントはリリース 2.0 で非推奨となり、リリース 2.1 で削除されます。Mixer を使用した拡張機能の実装はリリース 2.0 でも引き続きサポートされますが、拡張機能を新規の WebAssembly メカニズムに移行する必要があります。

以下のリソースタイプは Red Hat OpenShift Service Mesh 2.0 でサポートされなくなりました。

- **Policy** (authentication.istio.io/v1alpha1) はサポートされなくなりました。Policy リソースの特定の設定によっては、同じ効果を実現するために複数のリソースを設定しなければならない場合があります。
  - **RequestAuthentication** (security.istio.io/v1beta1) の使用
  - **PeerAuthentication** (security.istio.io/v1beta1) の使用
- **ServiceMeshPolicy** (maistra.io/v1) はサポートされなくなりました。
  - 上記のように **RequestAuthentication** または **PeerAuthentication** を使用しますが、Service Mesh コントロールプレーン namespace に配置します。
- **RbacConfig** (rbac.istio.io/v1alpha1) はサポートされなくなりました。
  - **AuthorizationPolicy** (security.istio.io/v1beta1) に置き換わります。これは **RbacConfig**、**ServiceRole**、および **ServiceRoleBinding** の動作を包含します。
- **ServiceMeshRbacConfig** (maistra.io/v1) がサポートされなくなりました。
  - 上記のように **AuthorizationPolicy** を使用しますが、Service Mesh コントロールプレーンの namespace に配置します。
- **ServiceRole** (rbac.istio.io/v1alpha1) がサポートされなくなりました。
- **ServiceRoleBinding** (rbac.istio.io/v1alpha1) がサポートされなくなりました。
- Kiali では、**login** および **LDAP** ストラテジーは非推奨になりました。今後のバージョンでは、OpenID プロバイダーを使用した認証が導入されます。

1.2.5. 既知の問題

Red Hat OpenShift Service Mesh には以下のような制限が存在します。

- アップストリームの Istio プロジェクトで完全にサポートされていないため、Red Hat OpenShift Service Mesh は IPv6 をサポートしていません。その結果、Red Hat OpenShiftServiceMesh はデュアルスタッククラスターはサポート対象外です。
- グラフレイアウト: Kiali グラフのレイアウトは、アプリケーションのアーキテクチャーや表示データ (グラフィックノードとその対話の数) によって異なることがあります。すべての状況に適した単一のレイアウトを作成することは不可能ではないにしても困難であるため、Kiali は複数の異なるレイアウトの選択肢を提供します。別のレイアウトを選択するには、**Graph Settings** メニューから異なる **Layout Schema** を選択します。

- Kiali コンソールから 分散トレースプラットフォーム や Grafana などの関連サービスに初めてアクセスする場合、OpenShift Container Platform のログイン認証情報を使用して証明書を受け入れ、再認証する必要があります。これは、フレームワークが組み込まれたページをコンソールで表示する方法に問題があるために生じます。
- Bookinfo サンプルアプリケーションは、IBM Z および IBM Power にインストールできません。
- WebAssembly 拡張機能は、IBM Z および IBM Power ではサポートされていません。
- LuaJIT は IBM Power ではサポートされていません。

1.2.5.1. Service Mesh の既知の問題

Red Hat OpenShift Service Mesh には次のような既知の問題が存在します。

- Istio-14743 Red Hat OpenShift Service Mesh のこのリリースがベースとしている Istio のバージョンに制限があるため、現時点で Service Mesh と互換性のないアプリケーションが存在する可能性があります。詳細は、リンク先のコミュニティーの問題を参照してください。
- OSSM-1655 SMCP で **mTLS** を有効にした後に、Kiali ダッシュボードにエラーが表示されます。
  SMCP で **spec.security.controlPlane.mtls** 設定を有効にすると、Kiali コンソールに **No subsets defined** のエラーメッセージが表示されます。
- OSSM-1505 この問題は、OpenShift Container Platform 4.11 で **ServiceMeshExtension** リソースを使用する場合にのみ発生します。OpenShift Container Platform 4.11 で **ServiceMeshExtension** を使用すると、リソースの準備が整いません。**oc describe ServiceMeshExtension** を使用して問題を調べると、**stderr: Error creating mount namespace before pivot: function not implemented** というエラーが表示されます。
  回避策: **ServiceMeshExtension** は Service Mesh 2.2 で廃止されました。**ServiceMeshExtension** から **WasmPlugin** リソースに移行します。詳細については、**ServiceMeshExtension** から **WasmPlugin** リソースへの移行を参照してください。
- OSSM-1396 ゲートウェイリソースに **spec.externalIPs** 設定が含まれている場合には、**ServiceMeshControlPlane** の更新時に再作成されず、ゲートウェイが削除されると再作成されることはありません。
- OSSM-1168 サービスメッシュリソースが単一の YAML ファイルとして作成される場合には、Envoy プロキシーサイドカーが Pod に確実に挿入されません。SMCP、SMMR、およびデプロイメントリソースを個別に作成すると、デプロイメントは想定どおりに機能します。
- OSSM-1052 Service Mesh コントロールプレーンで入力ゲートウェイのサービス **ExternalIP** を設定すると、サービスは作成されません。SMCP のスキーマには、サービスのパラメーターがありません。
  回避策: SMCP 仕様でゲートウェイの作成を無効にして、(サービス、ロール、および RoleBinding など) ゲートウェイのデプロイメントを完全に手動で管理します。
- OSSM-882 これは、Service Mesh 2.1 以前に適用されます。namespace は accessible_namespace 一覧にありますが、Kiali UI には表示されません。デフォルトでは、Kiali は kube で始まる namespace を表示しません。これらの namespace は通常内部使用のみであり、メッシュの一部ではないためです。
  たとえば、akube-a という名前の namespace を作成し、これを Service Mesh メンバーロールに追加すると、Kiali UI は namespace を表示しません。定義された除外パターンの場合、ソフトウェアは、このパターンで始まるか、そのパターンを含む namespace を除外します。

回避策: Kiali カスタムリソース設定を変更して、設定に接頭辞としてキャレット (^) を追加します。以下に例を示します。

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

- MAISTRA-2692 Mixer が削除されると、Service Mesh 2.0.x で定義されたカスタムメトリクスを 2.1 で使用できません。カスタムメトリクスは **EnvoyFilter** を使用して設定できます。明示的に文書化されている場合を除き、Red Hat は **EnvoyFilter** 設定をサポートできません。これは、下層の Envoy API と疎結合されており、後方互換性を保持することができないためです。

- MAISTRA-2648 **ServiceMeshExtensions** は現時点で IBM Z Systems にデプロイされたメッシュとは互換性がありません。

- MAISTRA-1959 2.0 への移行 Prometheus の収集 (**spec.addons.prometheus.scrape** が **true** に設定される) は mTLS が有効にされていると機能しません。また、Kiali は、mTLS が無効にされている場合に余分なグラフデータを表示します。
  この問題は、たとえば、プロキシー設定からポート 15020 を除外して対応できます。

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

- MAISTRA-1314 Red Hat OpenShift Service Mesh は IPv6 をサポートしていません。

- MAISTRA-453 新規プロジェクトを作成して Pod を即時にデプロイすると、サイドカーコンテナーの挿入は発生しません。この Operator は Pod の作成前に **maistra.io/member-of** を追加できないため、サイドカーコンテナーの挿入を発生させるには Pod を削除し、再作成する必要があります。

- MAISTRA-158 同じホスト名を参照する複数のゲートウェイを適用すると、すべてのゲートウェイが機能しなくなります。

1.2.5.2. Kiali の既知の問題

注記

Kiali についての新たな問題は、**Component** が **Kiali** に設定された状態の OpenShift Service Mesh プロジェクトに作成される必要があります。

Kiali の既知の問題は以下のとおりです。

- KIALI-2206 初回の Kiali コンソールへのアクセス時に、Kiali のキャッシュされたブラウザーデータがない場合、Kiali サービスの詳細ページの Metrics タブにある View in Grafana リンクは誤った場所にリダイレクトされます。この問題は、Kiali への初回アクセス時にのみ生じます。

- KIALI-507 Kiali は Internet Explorer 11 に対応していません。これは、基礎となるフレームワークが Internet Explorer に対応していないためです。Kiali コンソールにアクセスするには、Chrome、Edge、Firefox、または Safari ブラウザーの最新の 2 バージョンのいずれかを使用します。

1.2.5.3. Red Hat OpenShift 分散トレースの既知の問題

Red Hat OpenShift 分散トレースには、以下の制限があります。

- Apache Spark はサポートされていません。

- AMQ/Kafka 経由のストリーミングデプロイメントは、IBM Z および IBM Power Systems ではサポートされません。

これらは Red Hat OpenShift 分散トレースの既知の問題です。

- TRACING-2057 Kafka API が **v1beta2** に更新され、Strimzi Kafka Operator 0.23.0 がサポートされるようになりました。ただし、この API バージョンは AMQ Streams 1.6.3 ではサポートされません。以下の環境がある場合、Jaeger サービスはアップグレードされず、新規の Jaeger サービスを作成したり、既存の Jaeger サービスを変更したりすることはできません。

  - Jaeger Operator チャネル: **1.17.x stable** または **1.20.x stable**

  - AMQ Streams Operator チャネル: **amq-streams-1.6.x**
    この問題を解決するには、AMQ Streams Operator のサブスクリプションチャネルを **amq-streams-1.7.x** または **stable** のいずれかに切り替えます。

1.2.6. 修正された問題

次の問題は、現在のリリースで解決されています。

1.2.6.1. Service Mesh の修正された問題

- OSSM-2053 Red Hat OpenShift Service Mesh Operator 2.2 または 2.3 を使用すると、SMCP の調整中に、SMMR コントローラーがメンバーの namespace を **SMMR.status.configuredMembers** から削除しました。これにより、メンバーの namespace のサービスがしばらく利用できなくなりました。
  Red Hat OpenShift Service Mesh Operator 2.2 または 2.3 を使用すると、SMMR コントローラーは **SMMR.status.configuredMembers** から namespace を削除しなくなります。代わりに、コントローラーは namespace を **SMMR.status.pendingMembers** に追加して、それらが最新ではないことを示します。調整中に、各 namespace が SMCP と同期されると、namespace は **SMMR.status.pendingMembers** から自動的に削除されます。

- OSSM-1668 新しいフィールド **spec.security.jwksResolverCA** がバージョン 2.1 **SMCP** に追加されましたが、2.2.0 および 2.2.1 リリースにはありませんでした。このフィールドが存在する Operator バージョンから、このフィールドが存在しなかった Operator バージョンにアップグレードする場合、**SMCP** で **.spec.security.jwksResolverCA** フィールドを使用できませんでした。

- OSSM-1325 istiod Pod がクラッシュし、**fatal error: concurrent map iteration and map write** のエラーメッセージが表示されます。

- OSSM-1211 フェールオーバー用のフェデレーションサービスメッシュの設定が想定どおりに機能しません。
  Istiod パイロットログに、**envoy connection [C289] TLS error: 337047686:SSL routines:tls_process_server_certificate:certificate verify failed** のエラーが表示されます。

- OSSM-1099 Kiali コンソールに **Sorry, there was a problem** を表示していました。**Try a refresh or navigate to a different page.** メッセージが表示されました

- OSSM-1074 SMCP で定義された Pod アノテーションが Pod に注入されません。

- OSSM-999 Kiali は想定どおりに保持されませんでした。ダッシュボードグラフでは、カレンダーの時刻がグレーアウトされています。

- OSSM-797 Kiali Operator Pod は、Operator のインストールまたはアップグレード時に **CreateContainerConfigError** を生成します。

- **kube** で始まる OSSM-722 namespace は Kiali には表示されません。

- OSSM-569: Prometheus **istio-proxy** コンテナーには CPU メモリー制限がありません。
  Prometheus **istio-proxy** サイドカーは、**spec.proxy.runtime.container** で定義されたリソース制限を使用するようになりました。

- OSSM-449 VirtualService および Service により、以下のエラーが生じます。"Only unique values for domains are permitted.Duplicate entry of domain."

- 同様の名前を持つ OSSM-419 namespace は、namespace が Service Mesh Member Role で定義されていない場合でも、Kiali namespace の一覧に表示されます。

- OSSM-296 ヘルス設定を Kiali カスタムリソース (CR) に追加する場合、これは Kiali configmap にレプリケートされません。

- OSSM-291 Kiali コンソールの、Applications、Services、および Workloads ページの Remove Label from Filters が機能しません。

- OSSM-289 Kiali コンソールの istio-ingressgateway および jaeger-query サービスの Service Details ページにはトレースは表示されません。トレースは Jaeger にあります。

- OSSM-287 Kiali コンソールでは、トレースが Graph Service に表示されません。

- OSSM-285 Kiali コンソールにアクセスしようとすると、Error trying to get OAuth Metadata というエラーメッセージが表示されます。
  回避策: Kiali Pod を再起動します。

- MAISTRA-2735 Red Hat OpenShift Service Mesh バージョン 2.1 では、SMCP の調整時に Service Mesh Operator が削除するリソースが変更されました。以前のバージョンでは、Operator は以下のラベルを持つリソースを削除しました。

  - **maistra.io/owner**

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

  Operator は **app.kubernetes.io/managed-by=maistra-istio-operator** ラベルを含まないリソースを無視するようになりました。独自のリソースを作成する場合、**app.kubernetes.io/managed-by=maistra-istio-operator** ラベルをそれらに追加することはできません。

- MAISTRA-2687 外部証明書を使用する場合には、Red Hat OpenShift Service Mesh 2.1 フェデレーションゲートウェイでは、証明書チェーンが完全に送信されません。Service Mesh フェデ

レーション egress ゲートウェイはクライアント証明書のみを送信します。フェデレーション Ingress ゲートウェイはルート証明書のみを認識するため、ルート証明書をフェデレーションインポート **ConfigMap** に追加しない限り、クライアント証明書を検証できません。

- MAISTRA-2635 非推奨の Kubernetes API が置き換えられました。OpenShift Container Platform 4.8 との互換性を維持するために、**apiextensions.k8s.io/v1beta1** API は Red Hat OpenShift Service Mesh 2.0.8 で非推奨になりました。

- MAISTRA-2631 nsenter バイナリーが存在しないことが原因で Podman に問題が発生しているため、WASM 機能は使用できません。Red Hat OpenShift Service Mesh は **Error: error configuring CNI network plugin exec: "nsenter": executable file not found in $PATH** のエラーメッセージを生成します。コンテナーイメージには nsenter が含まれ、WASM が予想通りに機能するようになりました。

- MAISTRA-2534 istiod が JWT ルールで指定された発行者の JWKS の取得を試行する際に、発行者サービスは 502 で応答します。これにより、プロキシーコンテナーの準備ができなくなり、デプロイメントがハングしていました。コミュニティーバグ の修正は、Service Mesh 2.0.7 リリースに含まれています。

- MAISTRA-2411 Operator が **ServiceMeshControlPlane** で **spec.gateways.additionalIngress** を使用して新規 ingress ゲートウェイを作成する場合、Operator はデフォルトの istio-ingressgateway の場合と同様に追加の Ingress ゲートウェイの **NetworkPolicy** を作成しません。これにより、新規ゲートウェイのルートから 503 応答が生じました。
  回避策: <istio-system> namespace に **NetworkPolicy** を手動で作成します。

- MAISTRA-2401 CVE-2021-3586 servicemesh-operator: NetworkPolicy リソースが Ingress リソースのポートを誤って指定している。Red Hat OpenShift Service Mesh にインストールされた NetworkPolicy リソースでは、アクセス可能なポートが適切に指定されませんでした。これにより、すべての Pod からのこれらのリソース上のすべてのポートにアクセスできていました。以下のリソースに適用されるネットワークポリシーが影響を受けます。
  - Galley
  - Grafana
  - Istiod
  - Jaeger
  - Kiali
  - Prometheus
  - サイドカーインジェクター

- MAISTRA-2378: クラスターが **ovs-multitenant** で OpenShift SDN を使用するように設定されており、メッシュに多数の namespace (200+) が含まれる場合に、OpenShift Container Platform ネットワークプラグインは namespace を迅速に設定できません。Service Mesh がタイムアウトすると、namespace がサービスメッシュから継続的にドロップされ、再リストされます。

- MAISTRA-2370 は listerInformer で tombstones を処理します。更新されたキャッシュコードベースは、namespace キャッシュからのイベントを集約されたキャッシュに変換する際に tombstones を処理しないため、go ルーチンでパニックが生じました。

- MAISTRA-2117 オプションの **ConfigMap** マウントを Operator に追加します。CSV にはオプションの **ConfigMap** ボリュームマウントが含まれるようになり、**smcp-templates**

ConfigMap (存在する場合) をマウントします。**smcp-templates ConfigMap** が存在しない場合には、マウントされたディレクトリーは空になります。**ConfigMap** を作成すると、ディレクトリーには **ConfigMap** からのエントリーが設定され、**SMCP.spec.profiles** で参照できます。Service Mesh Operator の再起動は必要ありません。
CSV を変更して 2.0 Operator をを使用して smcp-templates ConfigMap をマウントしている場合には、Red Hat OpenShift Service Mesh 2.1 にアップグレードできます。アップグレード後に、CSV を編集せずに、既存の ConfigMap およびこれに含まれるプロファイルを引き続き使用できます。以前別の名前で ConfigMap を使用していた場合は、ConfigMap の名前を変更するか、またはアップグレード後に CSV を更新する必要があります。

- MAISTRA-2010 AuthorizationPolicy は **request.regex.headers** フィールドをサポートしません。**validatingwebhook** はこのフィールドのある AuthorizationPolicy を拒否し、これを無効にした場合でも、パイロットは同じコードを使用してこの検証を試行し、機能しません。

- MAISTRA-1979 **2.0** への移行 変換 webhook は、**SMCP.status** を v2 から v1 に変換する際に以下の重要なフィールドをドロップします。

  - conditions

  - components

  - observedGeneration

  - annotations
    Operator を 2.0 にアップグレードすると、リソースの maistra.io/v1 バージョンを使用する SMCP ステータスを読み取るクライアントツールが破損する可能性があります。

    また、**oc get servicemeshcontrolplanes.v1.maistra.io** の実行時に READY および STATUS 列が空になります。

- MAISTRA-1947 テクノロジープレビュー ServiceMeshExtensions への更新は適用されません。回避策: **ServiceMeshExtensions** を削除し、再作成します。

- MAISTRA-1983 **2.0** への移行 既存の無効な **ServiceMeshControlPlane** を使用した 2.0.0 へのアップグレードは修復できません。**ServiceMeshControlPlane** リソース内の無効な項目により、回復不可能なエラーが発生しました。修正により、エラーが回復可能になりました。無効なリソースを削除してこれを新しいリソースに置き換えるか、またはリソースを編集してエラーを修正できます。リソースの編集に関する詳細は、 [Red Hat OpenShift Service Mesh インストールの設定] を参照してください。

- MAISTRA-1502 バージョン 1.0.10 の CVE の修正により、Istio ダッシュボードは Grafana の **Home Dashboard** メニューから利用できなくなりました。Istio ダッシュボードにアクセスするには、ナビゲーションパネルの **Dashboard** メニューをクリックし、**Manage** タブを選択します。

- MAISTRA-1399 Red Hat OpenShift Service Mesh では、サポート対象外の CNI プロトコルがインストールされなくなりました。サポート対象のネットワーク設定は変更されていません。

- MAISTRA-1089 **2.0** への移行 コントロールプレーン以外の namespace で作成されたゲートウェイは自動的に削除されます。SMCP 仕様からゲートウェイ定義を削除した後にこれらのリソースを手動で削除する必要があります。

- MAISTRA-858 Istio 1.1.x に関連する非推奨のオプションと設定 について説明する以下のような Envoy ログメッセージが予想されます。

  - [2019-06-03 07:03:28.943][19][warning][misc] [external/envoy/source/common/protobuf/utility.cc:129] Using deprecated option 'envoy.api.v2.listener.Filter.config'.この設定はまもなく Envoy から削除されます。

  - [2019-08-12 22:12:59.001][13][warning][misc] [external/envoy/source/common/protobuf/utility.cc:174] Using deprecated option 'envoy.api.v2.Listener.use_original_dst' from file lds.proto.この設定はまもなく Envoy から削除されます。

- MAISTRA-806 エビクトされた Istio Operator Pod により、メッシュおよび CNI はデプロイできなくなります。
  回避策: コントロールペインのデプロイ時に **istio-operator** Pod がエビクトされる場合は、エビクトされた **istio-operator** Pod を削除します。

- MAISTRA-681 Service Mesh コントロールプレーンに多くの namespace がある場合に、パフォーマンスの問題が発生する可能性があります。

- MAISTRA-193 ヘルスチェックが citadel で有効になっていると、予期しないコンソール情報メッセージが表示されます。

- Bugzilla 1821432: OpenShift Container Platform カスタムリソースの詳細ページのトグルコントロールで CR が正しく更新されません。OpenShift Container Platform Web コンソールの Service Mesh Control Plane (SMCP) Overview ページの UI のトグルコントロールにより、リソースの誤ったフィールドが更新されることがあります。SMCP を更新するには、YAML コンテンツを直接編集するか、トグルコントロールをクリックせずにコマンドラインからリソースを更新します。

1.2.6.2. Red Hat OpenShift 分散トレースの修正された問題

- TRACING-2337 Jaeger が、Jaeger ログに以下のような警告メッセージを繰り返しロギングする。

  ```
  {"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}
  ```

  この問題は、gRPC ポートではなく、クエリーサービスの HTTP(S) ポートのみを公開することで解決されました。

- TRACING-2009 Jaeger Operator が Strimzi Kafka Operator 0.23.0 のサポートを追加するように更新されました。

- TRACING-1907: アプリケーション namespace に Config Map がないため、Jaeger エージェントサイドカーの挿入が失敗していました。Config Map は正しくない **OwnerReference** フィールドの設定により自動的に削除され、そのため、アプリケーション Pod は ContainerCreating ステージから移動しませんでした。誤った設定が削除されました。

- TRACING-1725 は TRACING-1631 に対応しています。これはもう 1 つの修正であり、同じ名前だが異なる namespace にある複数の Jaeger 実稼働インスタンスがある場合に Elasticsearch 証明書を適切に調整することができるようになりました。BZ-1918920 も参照してください。

- TRACING-1631 同じ名前を使用するが、異なる namespace 内にある複数の Jaeger 実稼働インスタンスを使用すると、Elasticsearch 証明書に問題が発生します。複数のサービスメッシュがインストールされている場合、すべての Jaeger Elasticsearch インスタンスは個別のシーク

レットではなく同じ Elasticsearch シークレットを持ち、これにより、OpenShift Elasticsearch Operator がすべての Elasticsearch クラスターと通信できなくなりました。

- TRACING-1300 Istio サイドカーを使用する場合に、Agent と Collector 間の接続が失敗します。Jaeger Operator で有効にされた TLS 通信の更新は、Jaeger サイドカーエージェントと Jaeger Collector 間でデフォルトで提供されます。

- TRACING-1208 Jaeger UI にアクセスする際に、認証の 500 Internal Error が出されます。OAuth を使用して UI に対する認証を試行すると、oauth-proxy サイドカーが **additionalTrustBundle** でインストール時に定義されたカスタム CA バンドルを信頼しないため、500 エラーが出されます。

- TRACING-1166 現時点で、Jaeger ストリーミングストラテジーを非接続環境で使用することはできません。Kafka クラスターがプロビジョニングされる際に、以下のエラーが出されます: **Failed to pull image registry.redhat.io/amq7/amq-streams-kafka-24-rhel7@sha256:f9ceca004f1b7dccb3b82d9a8027961f9fe4104e0ed69752c0bdd8078b4a1076**

- Trace-809 Jaeger Ingester には Kafka 2.3 との互換性がありません。Jaeger Ingester のインスタンスが複数あり、十分なトラフィックがある場合、リバランスメッセージがログに継続的に生成されます。これは、Kafka 2.3.1 で修正された Kafka 2.3 のリグレッションによって生じます。詳細は、Jaegertracing-1819 を参照してください。

- BZ-1918920/LOG-1619 Elasticsearch Pod は更新後に自動的に再起動しません。
  回避策: Pod を手動で再起動します。

1.3. SERVICE MESH について

Red Hat OpenShift Service Mesh は、サービスメッシュにおいてネットワーク化されたマイクロサービス全体の動作に関する洞察と運用管理のためのプラットフォームを提供します。Red Hat OpenShift Service Mesh では、OpenShift Container Platform 環境でマイクロサービスの接続、保護、監視を行うことができます。

1.3.1. サービスメッシュについて

サービスメッシュ は、分散したマイクロサービスアーキテクチャーの複数のアプリケーションを設定するマイクロサービスのネットワークであり、マイクロサービス間の対話を可能にします。Service Mesh のサイズとおよび複雑性が増大すると、これを把握し、管理することがより困難になる可能性があります。

オープンソースの Istio プロジェクトをベースとする Red Hat OpenShift Service Mesh は、サービスコードに変更を加えずに、既存の分散したアプリケーションに透過的な層を追加します。Red Hat OpenShift Service Mesh サポートは、特別なサイドカープロキシーをマイクロサービス間のネットワーク通信をすべてインターセプトするメッシュ内の関連サービスにデプロイすることで、サービスに追加できます。Service Mesh コントロールプレーンの機能を使用して Service Mesh を設定し、管理します。

Red Hat OpenShift Service Mesh により、以下を提供するデプロイされたサービスのネットワークを簡単に作成できます。

- 検出
- 負荷分散
- サービス間の認証
- 障害回復

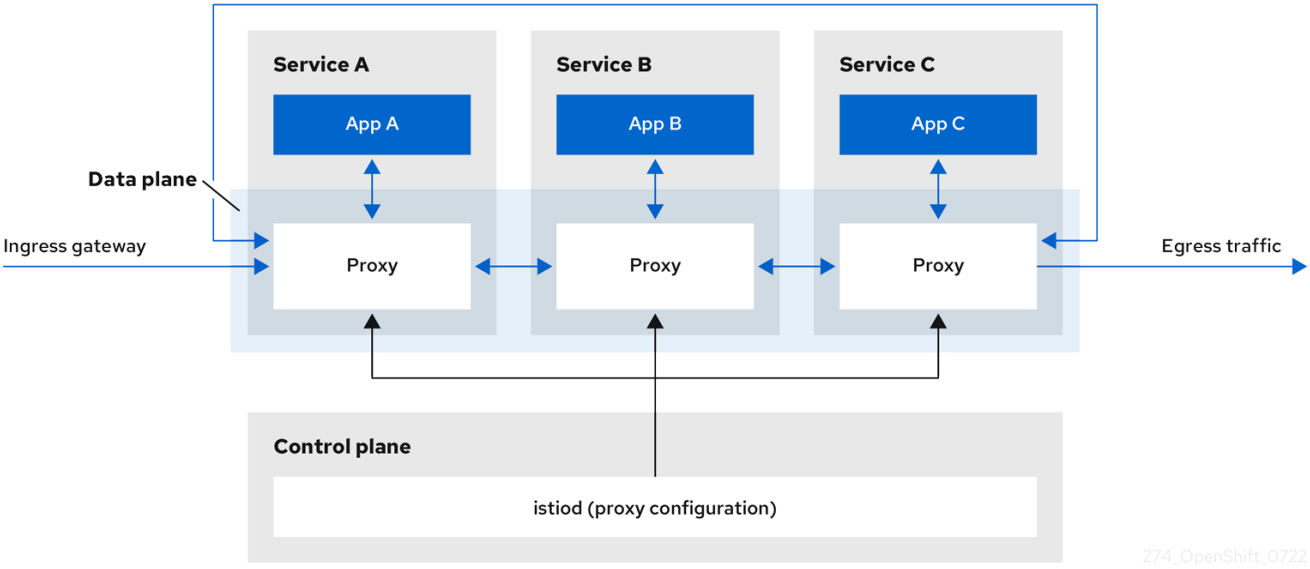
- メトリクス
- モニタリング

Red Hat OpenShift Service Mesh は、以下を含むより複雑な運用機能も提供します。

- A/B テスト
- カナリアリリース
- アクセス制御
- エンドツーエンド認証

1.3.2. Service Mesh アーキテクチャー

サービスメッシュテクノロジーはネットワーク通信レベルで動作します。つまり、サービスメッシュコンポーネントは、マイクロサービスとの間のトラフィックを取得または傍受して、リクエストを変更したり、リダイレクトしたり、他のサービスへの新しいリクエストを作成したりします。



高いレベルでは、Red Hat OpenShift Service Mesh はデータプレーンおよびコントロールプレーンで設定されます。

データプレーン は、Pod のアプリケーションコンテナーとともに実行するインテリジェントプロキシーのセットで、サービスメッシュ内のマイクロサービス間の受信および送信ネットワーク通信をすべて傍受し、制御します。データプレーンは、すべての受信 (ingress) および送信 (egress) ネットワークトラフィックを傍受するように実装されます。Istio データプレーンは、Pod のサイドアプリケーションコンテナーで実行する Envoy コンテナーで設定されます。Envoy コンテナーはプロキシーとして機能し、すべてのネットワーク通信を Pod に対して制御します。

- **Envoy** プロキシー は、データプレーントラフィックと対話する唯一の Istio コンポーネントです。プロキシー経由でサービスフロー間の受信 (ingress) および送信 (egress) ネットワークトラフィックはすべて、そのプロキシーを介して行われます。また、Envoy プロキシーは、メッシュ内のサービストラフィックに関連するすべてのメトリクスを収集します。Envoy プロキシーはサイドカーとしてデプロイされ、サービスと同じ Pod で実行されます。Envoy プロキシーは、メッシュゲートウェイの実装にも使用されます。

  - サイドカープロキシー は、割り当てられたワークロードインスタンスへのインバウンドおよびアウトバウンドの通信を管理します。

  - ゲートウェイ は、受信または送信 HTTP/TCP 接続を受信するロードバランサーとして動作するプロキシーです。ゲートウェイ設定は、サービスワークロードと共に実行されるサイドカー Envoy プロキシーではなく、メッシュのエッジで実行されているスタンドアロンの Envoy プロキシーに適用されます。ゲートウェイを使用してメッシュの受信トラフィックおよび送信トラフィックを管理することで、メッシュに入るか、またはメッシュを出るトラフィックを指定できます。

    - **Ingress-gateway** - ingress コントローラーとしても知られる、Ingress ゲートウェイはサービスメッシュに入るトラフィックを受信し、制御する専用の Envoy プロキシーです。Ingress ゲートウェイは、モニターリングおよびルーティングルールなどの機能をクラスターに入るトラフィックに適用できるようにします。

    - **Egress-gateway** - egress コントローラーとしても知られる、Egress Gateway はサービスメッシュからトラフィックを管理する専用の Envoy プロキシーです。Egress Gateway は、モニタリングおよびルートルールなどの機能をメッシュのトラフィックに適用できるようにします。

コントロールプレーン は、データプレーンを設定するプロキシーを管理し、設定します。これは、設定用の権威ソースで、アクセス制御および使用状況ポリシーを管理し、サービスメッシュのプロキシーからメトリクスを収集します。

- Istio コントロールプレーンは、以前の複数のコントロールプレーンコンポーネント (Citadel、Galley、Pilot) を単一バイナリーに統合する **Istiod** で設定されています。Istiod は、サービス検出、設定、および証明書の管理を行います。これは、高レベルのルーティングルールを Envoy 設定に変換し、それらをランタイム時にサイドカーコンテナーに伝播します。

  - Istiod は認証局 (CA) として機能し、データプレーンでセキュアな mTLS 通信に対応する証明書を生成します。この場合、外部 CA を使用することもできます。

  - Istiod は、サイドカーコンテナーを OpenShift クラスターにデプロイされたワークロードに挿入します。

Red Hat OpenShift Service Mesh は、**istio-operator** を使用してコントロールプレーンのインストールも管理します。**Operator** は、OpenShift クラスターで共通アクティビティーを実装し、自動化できるソフトウェアの設定要素です。これはコントローラーとして機能し、クラスター内の必要なオブジェクトの状態 (この場合は Red Hat OpenShift Service Mesh のインストール) を設定または変更できます。

Red Hat OpenShift Service Mesh は以下の Istio アドオンを製品の一部としてバンドルします。

- **Kiali**: Kiali は Red Hat OpenShift Service Mesh の管理コンソールです。ダッシュボード、可観測性、および堅牢な設定、ならびに検証機能を提供します。これは、トラフィックトポロジーを推測してサービスメッシュの構造を示し、メッシュの正常性を表示します。Kiali は、詳細なメトリクス、強力な検証、Grafana へのアクセス、および分散トレースプラットフォームとの強力な統合を提供します。

- **Prometheus**: Red Hat OpenShift Service Mesh は Prometheus を使用してサービスからのテレメトリー情報を保存します。Kiali は、メトリクス、ヘルスステータス、およびメッシュトポロジーを取得するために Prometheus に依存します。

- **Jaeger**: Red Hat OpenShift Service Mesh は分散トレースプラットフォームをサポートします。Jaeger はオープンソースのトレース機能で、複数のサービス間の単一要求に関連付けられたトレースを一元管理し、表示します。分散トレースプラットフォームを使用すると、マイクロサービスベースの分散システムの監視とトラブルシューティングを行うことができます。

- **Elasticsearch**: Elasticsearch は、オープンソースの分散型 JSON ベースの検索および解析エンジンです。分散トレースプラットフォームは、永続ストレージに Elasticsearch を使用します。

- **Grafana**: Grafana は、Istio データの高度なクエリーおよびメトリクス分析、ならびにダッシュボードを使用してメッシュ管理者を提供します。任意で、Grafana を使用してサービスメッシュメトリクスを分析できます。

以下の Istio 統合は Red Hat OpenShift Service Mesh でサポートされます。

- **3scale**: Istio では、オプションで Red Hat 3scale API Management ソリューションとの統合が提供されます。2.1 より前のバージョンでは、この統合は 3scale Istio アダプターを使用して実行されました。バージョン 2.1 以降では、3scale の統合は WebAssembly モジュールを介して行われます。

3scale アダプターのインストール方法に関する詳細は、3scale Istio アダプターのドキュメント を参照してください。

1.3.3. Kiali について

Kiali は、サービスメッシュのマイクロサービスとそれらの接続方法を表示してサービスメッシュを可視化します。

1.3.3.1. Kiali の概要

Kiali では、OpenShift Container Platform で実行される Service Mesh の可観測性 (Observability) を提供します。Kiali は、Istio サービスメッシュの定義、検証、および確認に役立ちます。これは、トポロジーの推測によりサービスメッシュの構造を理解しやすくし、またサービスメッシュの健全性に関する情報も提供します。

Kiali は、サーキットブレーカー、要求レート、レイテンシー、トラフィックフローのグラフなどの機能を可視化する、namespace のインタラクティブなグラフビューをリアルタイムで提供します。Kiali では、異なるレベルのコンポーネント (アプリケーションからサービスおよびワークロードまで) についての洞察を提供し、選択されたグラフノードまたはエッジに関するコンテキスト情報やチャートを含む対話を表示できます。Kiali は、ゲートウェイ、宛先ルール、仮想サービス、メッシュポリシーなど、Istio 設定を検証する機能も提供します。Kiali は詳細なメトリクスを提供し、基本的な Grafana 統合は高度なクエリーに利用できます。Jaeger を Kiali コンソールに統合することで、分散トレースを提供します。

Kiali は、デフォルトで Red Hat OpenShift Service Mesh の一部としてインストールされます。

1.3.3.2. Kiali アーキテクチャー

Kiali はオープンソースの Kiali プロジェクト に基づいています。Kiali は Kiali アプリケーションと Kiali コンソールという 2 つのコンポーネントで設定されます。

- **Kiali** アプリケーション (バックエンド): このコンポーネントはコンテナーアプリケーションプラットフォームで実行され、サービスメッシュコンポーネントと通信し、データを取得し、処理し、そのデータをコンソールに公開します。Kiali アプリケーションはストレージを必要としません。アプリケーションをクラスターにデプロイする場合、設定は ConfigMap およびシークレットに設定されます。

- **Kiali** コンソール (フロントエンド): Kiali コンソールは Web アプリケーションです。Kiali アプリケーションは Kiali コンソールを提供し、データをユーザーに表示するためにバックエンドに対してデータのクエリーを実行します。

さらに Kiali は、コンテナーアプリケーションプラットフォームと Istio が提供する外部サービスとコンポーネントに依存します。

- **Red Hat Service Mesh** (Istio): Istio は Kiali の要件です。Istio はサービスメッシュを提供し、制御するコンポーネントです。Kiali と Istio を個別にインストールすることはできますが、Kiali は Istio に依存し、Istio が存在しない場合は機能しません。Kiali は、Prometheus および Cluster API 経由で公開される Istio データおよび設定を取得する必要があります。

- **Prometheus**: 専用の Prometheus インスタンスは Red Hat OpenShift Service Mesh インストールの一部として組み込まれています。Istio Telemetry が有効にされている場合、メトリクスデータは Prometheus に保存されます。Kiali はこの Prometheus データを使用して、メッシュトポロジーの判別、メトリクスの表示、健全性の算出、可能性のある問題の表示などを行います。Kiali は Prometheus と直接通信し、Istio Telemetry で使用されるデータスキーマを想定します。Prometheus は Istio に依存しており、Kiali と明示的な依存関係があるため、Kiali の機能の多くは Prometheus なしに機能しません。

- **Cluster API**: Kiali はサービスメッシュ設定を取得し、解決するために、OpenShift Container Platform (Cluster API) の API を使用します。Kiali は Cluster API に対してクエリーを実行し、たとえば、namespace、サービス、デプロイメント、Pod、その他のエンティティーの定義を取得します。Kiali はクエリーを実行して、異なるクラスターエンティティー間の関係も解決します。Cluster API に対してもクエリーを実行し、仮想サービス、宛先ルール、ルートルール、ゲートウェイ、クォータなどの Istio 設定を取得します。

- **Jaeger**: Jaeger はオプションですが、Red Hat OpenShift Service Mesh インストールの一部としてデフォルトでインストールされます。デフォルトの Red Hat OpenShift Service Mesh インストールの一部として分散トレースプラットフォームをインストールすると、Kiali コンソールには分散トレースデータを表示するタブが含まれます。Istio の分散トレース機能を無効にした場合、トレースデータは利用できないことに注意してください。また、トレースデータを表示するには、ユーザーは Service Mesh コントロールプレーンがインストールされている namespace にアクセスできる必要があります。

- **Grafana**: Grafana はオプションですが、デフォルトでは Red Hat OpenShift Service Mesh インストールの一部としてインストールされます。使用可能な場合、Kiali のメトリクスページには Grafana 内の同じメトリクスにユーザーを移動させるリンクが表示されます。Grafana ダッシュボードへのリンクと Grafana データを表示するには、Service Mesh コントロールプレーンがインストールされている namespace にユーザーがアクセスできる必要があることに注意してください。

1.3.3.3. Kiali の機能

Kiali コンソールは Red Hat Service Mesh に統合され、以下の機能を提供します。

- 健全性: アプリケーション、サービス、またはワークロードの問題を素早く特定します。

- トポロジー: Kiali グラフを使用して、アプリケーション、サービス、またはワークロードの通信方法を可視化します。

- メトリクス: 事前定義済みのメトリクスダッシュボードを使用すると、Go、Node.js、Quarkus、Spring Boot、Thorntail、および Vert.xまた、独自のカスタムダッシュボードを作成することもできます。

- トレース: Jaeger との統合により、アプリケーションを設定するさまざまなマイクロサービスで要求のパスを追跡できます。

- 検証: 最も一般的な Istio オブジェクト (宛先ルール、サービスエントリー、仮想サービスなど) で高度な検証を実行します。

- 設定: ウィザードを使用するか、または Kiali コンソールの YAML エディターを直接使用して、Istio ルーティング設定を作成し、更新し、削除できるオプションの機能です。

1.3.4. 分散トレースについて

ユーザーがアプリケーションでアクションを実行するたびに、応答を生成するために多数の異なるサービスに参加を要求する可能性のあるアーキテクチャーによって要求が実行されます。この要求のパスは分散トランザクションです。分散トレースプラットフォームを使用すると、分散トレースを実行できます。これは、アプリケーションを設定するさまざまなマイクロサービスで要求のパスを追跡します。

分散トレースは、さまざまな作業ユニットの情報を連携させるために使用される技術です。これは、分散トランザクションでのイベントのチェーン全体を理解するために、通常さまざまなプロセスまたはホストで実行されます。分散トレースを使用すると、開発者は大規模なサービス指向アーキテクチャーで呼び出しフローを可視化できます。シリアル化、並行処理、およびレイテンシーのソースについて理解しておくことも重要です。

分散トレースプラットフォームは、マイクロサービスのスタック全体で個別のリクエストの実行を記録し、トレースとして表示します。トレース とは、システムにおけるデータ/実行パスです。エンドツーエンドトレースは、1 つ以上のスパンで設定されます。

スパンは、オペレーション名、オペレーションの開始時間および期間を持つ、作業の論理単位を表しています。スパンは因果関係をモデル化するためにネスト化され、順序付けられます。

1.3.4.1. 分散トレースの概要

サービスの所有者は、分散トレースを使用してサービスをインストルメント化し、サービスアーキテクチャーに関する洞察を得ることができます。分散トレースを使用して、現代的なクラウドネイティブのマイクロサービスベースのアプリケーションにおける、コンポーネント間の対話の監視、ネットワークプロファイリング、およびトラブルシューティングを行うことができます。

分散トレースを使用すると、以下の機能を実行できます。

- 分散トランザクションの監視
- パフォーマンスとレイテンシーの最適化
- 根本原因分析の実行

Red Hat OpenShift の分散トレースは、2 つの主要コンポーネントで設定されています。

- **Red Hat OpenShift** 分散トレースプラットフォーム: このコンポーネントは、オープンソースの Jaeger プロジェクト に基づいています。
- **Red Hat OpenShift** 分散トレースデータ収集: このコンポーネントは、オープンソースの OpenTelemetry プロジェクト に基づいています。

これらのコンポーネントは共に、特定のベンダーに依存しない OpenTracing API およびインストルメンテーションに基づいています。

1.3.4.2. Red Hat OpenShift 分散トレースのアーキテクチャー

Red Hat OpenShift 分散トレースは、複数のコンポーネントで設定されており、トレースデータを収集し、保存し、表示するためにそれらが連携します。

- **Red Hat OpenShift** 分散トレースプラットフォーム: このコンポーネントは、オープンソースの Jaeger プロジェクト に基づいています。
  - クライアント (Jaeger クライアント、Tracer、Reporter、インストルメント化されたアプリケーション、クライアントライブラリー): 分散トレースプラットフォームクライアントは、OpenTracing API の言語固有の実装です。それらは、手動または (Camel (Fuse)、

Spring Boot (RHOAR)、MicroProfile (RHOAR/Thorntail)、Wildfly (EAP)、その他 OpenTracing にすでに統合されているものを含む) 各種の既存オープンソースフレームワークを使用して、分散トレース用にアプリケーションをインストルメント化するために使用できます。

  - エージェント (Jaeger エージェント、Server Queue、Processor Worker): 分散トレースプラットフォームエージェントは、User Datagram Protocol (UDP) で送信されるスパンをリッスンするネットワークデーモンで、 Collector にバッチ処理や送信を実行します。このエージェントは、インストルメント化されたアプリケーションと同じホストに配置されることが意図されています。これは通常、Kubernetes などのコンテナー環境にサイドカーコンテナーを配置することによって実行されます。

  - **Jaeger Collector** (Collector、Queue、Worker): Jaeger エージェントと同様に、Jaeger Collector はスパンを受信し、これらを処理するために内部キューに配置します。これにより、Jaeger Collector はスパンがストレージに移動するまで待機せずに、クライアント/エージェントにすぐに戻ることができます。

  - **Storage** (Data Store): コレクターには永続ストレージのバックエンドが必要です。Red Hat OpenShift 分散トレースプラットフォームには、スパンストレージ用のプラグ可能なメカニズムがあります。本リリースでは、サポートされているストレージは Elasticsearch のみであることに注意してください。

  - **Query** (Query Service): Query は、ストレージからトレースを取得するサービスです。

  - **Ingester** (Ingester Service): Red Hat OpenShift 分散トレースは Apache Kafka を Collector と実際の Elasticsearch バッキングストレージ間のバッファーとして使用できます。Ingester は、Kafka からデータを読み取り、Elasticsearch ストレージバックエンドに書き込むサービスです。

  - **Jaeger Console**: Red Hat OpenShift 分散トレースプラットフォームユーザーインターフェイスを使用すると、分散トレースデータを可視化できます。検索ページで、トレースを検索し、個別のトレースを設定するスパンの詳細を確認することができます。

- **Red Hat OpenShift** 分散トレースデータ収集: このコンポーネントは、オープンソースの OpenTelemetry プロジェクト に基づいています。

  - **OpenTelemetry Collector**: OpenTelemetry Collector は、テレメトリーデータを受信、処理、エクスポートするためのベンダーに依存しない方法です。OpenTelemetry Collector は、Jaeger や Prometheus などのオープンソースの可観測性データ形式をサポートし、1 つ以上のオープンソースまたは商用バックエンドに送信します。Collector は、インストルメンテーションライブラリーがテレメトリーデータをエクスポートするデフォルトの場所です。

1.3.4.3. Red Hat OpenShift 分散トレースの機能

Red Hat OpenShift 分散トレースには、以下の機能が含まれます。

- Kiali との統合: 適切に設定されている場合、Kiali コンソールから分散トレースデータを表示できます。

- 高いスケーラビリティー: 分散トレースバックエンドは、単一障害点がなく、ビジネスニーズに合わせてスケーリングできるように設計されています。

- 分散コンテキストの伝播: さまざまなコンポーネントからのデータをつなぎ、完全なエンドツーエンドトレースを作成できます。

- Zipkin との後方互換性: Red Hat OpenShift 分散トレースには、Zipkin のドロップイン置き換えで使用できるようにする API がありますが、本リリースでは、Red Hat は Zipkin の互換性をサポートしていません。

1.3.5. 次のステップ

- OpenShift Container Platform 環境で Red Hat OpenShift Service Mesh をインストールする準備 をします。

1.4. サービスメッシュのデプロイメントモデル

Red Hat OpenShift Service Mesh は、さまざまなデプロイメントモデルを複数サポートし、ビジネス要件に最も適合するように、各種方法を組み合わせることができます。

1.4.1. 単一メッシュデプロイメントモデル

最も単純な Istio デプロイメントモデルは単一のメッシュです。

Kubernetes では **mynamespace** namespace で **myservice** という名前を指定できるのはサービス 1 つであるため、メッシュ内のサービス名は一意である必要があります。ただし、ワークロードインスタンスは、サービスアカウント名を同じ namespace のワークロード間で共有できるため、共通のアイデンティティーを共有できます。

1.4.2. 単一のテナンシーデプロイメントモデル

Istio では、テナントはデプロイされたワークロードで共通のアクセスおよび権限を共有するユーザーのグループです。テナントを使用して、異なるチーム間で一定レベルの分離を確保できます。Istio.io またはサービスリソースの **NetworkPolicies**、**AuthorizationPolicies**、および **exportTo** アノテーションを使用して、異なるテナントへのアクセスを分離できます。

OpenShift Service Mesh バージョン 1.0 の時点では、単一テナントの、クラスター全体での Service Mesh コントロールプレーン設定は推奨されていません。Red Hat OpenShift Service Mesh はデフォルトでマルチテナントモデルに設定されます。

1.4.3. マルチテナントデプロイメントモデル

Red Hat OpenShift Service Mesh は、デフォルトでマルチテナントとして設定される **ServiceMeshControlPlane** をインストールします。Red Hat OpenShift Service Mesh はマルチテナント Operator を使用して、Service Mesh コントロールプレーンのライフサイクルを管理します。メッシュ内では、テナントに namespace が使用されます。

Red Hat OpenShift Service Mesh は **ServiceMeshControlPlane** リソースを使用してメッシュインストールを管理します。メッシュのインストールのスコープはデフォルトでは、リソースを含む namespace に限定されます。**ServiceMeshMemberRoll** および **ServiceMeshMember** リソースを使用して、別の namespace をメッシュに追加します。Namespace は単一のメッシュにのみ組み込むことができ、複数のメッシュを単一の OpenShift クラスターにインストールできます。

通常のサービスメッシュデプロイメントでは、単一の Service Mesh コントロールプレーンを使用してメッシュ内のサービス間の通信を設定します。Red Hat OpenShift Service Mesh はテナントごとにコントロールプレーン 1 つと、メッシュが 1 つあるソフトマルチテナンシーをサポートします。クラスター内には、複数の独立したコントロールプレーンが存在させることができます。マルチテナントのデプロイメントでは、Service Mesh にアクセスできるプロジェクトを指定し、Service Mesh を他のコントロールプレーンインスタンスから分離します。

クラスター管理者はすべての Istio コントロールプレーンを制御して、表示できますが、テナント管理者は特定の Service Mesh、Kiali、および Jaeger インスタンスしか制御できません。

指定の namespace または namespace 設定だけにワークロードをデプロイするチームパーミッションを付与できます。サービスメッシュ管理者が **mesh-user** ロールを付与された場合には、ユーザーは **ServiceMeshMember** リソースを作成して namespace を **ServiceMeshMemberRoll** に追加できます。

1.4.4. マルチテーマまたはフェデレーションされたデプロイメントモデル

フェデレーション は、個別の管理ドメインで管理される個別のメッシュ間でサービスとワークロードを共有できるデプロイメントモデルです。

Istio マルチクラスターモデルでは、メッシュ間だで高いレベルの信頼が必要なだけでなく、個々のメッシュが存在するすべての Kubernetes API サーバーへのリモートアクセスも必要です。Red Hat OpenShift Service Mesh のフェデレーションは、メッシュ間の最小限の信頼を前提とする Service Mesh のマルチクラスター実装に対して独自のアプローチを採用しています。

フェデレーションされたメッシュ は、単一のメッシュとして動作させるメッシュのグループです。各メッシュのサービスは、独自のサービスにすることができます。たとえば、別のメッシュからサービスをインポートすることでサービスを追加するメッシュは、メッシュ全体で同じサービスにさらにワークロードを追加し、高可用性を提供することや、その両方を組み合わせることができます。フェデレーションされたメッシュに参加するすべてのメッシュは個別に管理されたままなので、フェデレーション内の他のメッシュとの間でエクスポートやインポートされるサービスを明示的に設定する必要があります。証明書の生成、メトリクス、トレース収集などのサポート機能は、それぞれのメッシュのローカルで機能します。

1.5. SERVICE MESH と ISTIO の相違点

Red Hat OpenShift Service Mesh は、追加機能の提供、OpenShift Container Platform へのデプロイ時の差異の処理を実行する Istio のインストールとは異なります。

1.5.1. Istio と Red Hat OpenShift Service Mesh の相違点

以下の機能は Service Mesh と Istio で異なります。

1.5.1.1. コマンドラインツール

Red Hat OpenShift Service Mesh のコマンドラインツールは **oc** です。 Red Hat OpenShift Service Mesh は、**istioctl** をサポートしません。

1.5.1.2. インストールおよびアップグレード

Red Hat OpenShift Service Mesh は、Istio インストールプロファイルをサポートしません。

Red Hat OpenShift Service Mesh はサービスメッシュのカナリアアップグレードをサポートしません。

1.5.1.3. 自動的な挿入

アップストリームの Istio コミュニティーインストールは、ラベル付けしたプロジェクト内の Pod にサイドカーコンテナーを自動的に挿入します。

Red Hat OpenShift Service Mesh は、サイドカーコンテナーをあらゆる Pod に自動的に挿入することはなく、プロジェクトにラベルを付けることなくアノテーションを使用して挿入をオプトインする必要があります。この方法で必要となる権限は少なく、ビルダー Pod などの他の OpenShift 機能と競合し

ません。自動挿入を有効にするには、サイドカーの自動挿入セクションで説明されているように **sidecar.istio.io/inject** アノテーションを指定します。

表1.4 サイドカーインジェクションのラベルとアノテーションの設定

	アップストリーム Istio	Red Hat OpenShift Service Mesh
namespace ラベル	"enabled" と "disabled"をサポート	"disabled" をサポート
Pod Label	"true" と "false"をサポート	サポート対象外
Pod のアノテーション	"false" のみをサポート	"true" と "false"をサポート

1.5.1.4. Istio ロールベースアクセス制御機能

Istio ロールベースアクセス制御機能 (RBAC) は、サービスへのアクセスを制御するために使用できるメカニズムを提供します。ユーザー名やプロパティーのセットを指定してサブジェクトを特定し、それに応じてアクセス制御を適用することができます。

アップストリームの Istio コミュニティーインストールには、ヘッダーの完全一致の実行、ヘッダーのワイルドカードの一致の実行、または特定の接頭辞または接尾辞を含むヘッダーの有無をチェックするオプションが含まれます。

Red Hat OpenShift Service Mesh は、正規表現を使用して要求ヘッダーと一致させる機能を拡張します。**request.regex.headers** のプロパティーキーを正規表現で指定します。

アップストリーム Istio コミュニティーの要求ヘッダーのマッチング例

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

1.5.1.5. OpenSSL

Red Hat OpenShift Service Mesh では、BoringSSL を OpenSSL に置き換えます。OpenSSL は、Secure Sockets Layer (SSL) プロトコルおよび Transport Layer Security (TLS) プロトコルのオープンソース実装を含むソフトウェアライブラリーです。Red Hat OpenShift Service Mesh Proxy バイナリーは、基礎となる Red Hat Enterprise Linux オペレーティングシステムから OpenSSL ライブラリー (libssl および libcrypto) を動的にリンクします。

1.5.1.6. 外部ワークロード

Red Hat OpenShift Service Mesh は、ベアメタルサーバー上で OpenShift の外部で実行される仮想マシンなどの外部ワークロードをサポートしません。

1.5.1.7. 仮想マシンのサポート

OpenShift Virtualization を使用して、仮想マシンを OpenShift にデプロイできます。次に、メッシュの一部である他の Pod と同様に、mTLS または AuthorizationPolicy などのメッシュポリシーをこれらの仮想マシンに適用できます。

1.5.1.8. コンポーネントの変更

- すべてのリソースに **maistra-version** ラベルが追加されました。
- すべての Ingress リソースが OpenShift ルートリソースに変換されました。
- Grafana、分散トレース (Jaeger)、および Kiali はデフォルトで有効にされ、OpenShift ルート経由で公開されます。
- すべてのテンプレートから Godebug が削除されました。
- **istio-multi** ServiceAccount および ClusterRoleBinding が削除されました。また、 **istio-reader** ClusterRole も削除されました。

1.5.1.9. Envoy フィルター

Red Hat OpenShift Service Mesh は、明示的に文書化されている場合を除き、**EnvoyFilter** の設定はサポートしていません。下層の EnvoyAPI と疎結合されており、後方互換性を確保できません。**EnvoyFilter** パッチは、Istio によって生成される Envoy 設定の形式に非常に敏感です。Istio で生成された設定を変更すると、**EnvoyFilter** のアプリケーションが破損する可能性があります。

1.5.1.10. Envoy サービス

Red Hat OpenShift Service Mesh は、QUIC ベースのサービスをサポートしません。

1.5.1.11. Istio Container Network Interface (CNI) プラグイン

Red Hat OpenShift Service Mesh には CNI プラグインが含まれ、アプリケーション Pod ネットワーキングを設定する代替の方法が提供されます。CNI プラグインは **init-container** ネットワーク設定を置き換えます。これにより、昇格した権限でサービスアカウントおよびプロジェクトに SCC (Security Context Constraints) へのアクセスを付与する必要がなくなります。

1.5.1.12. グローバル mTLS 設定

Red Hat OpenShift Service Mesh は、メッシュ内で相互 TLS 認証 (mTLS) を有効または無効にする **PeerAuthentication** リソースを作成します。

1.5.1.13. ゲートウェイ

Red Hat OpenShift Service Mesh は、デフォルトで受信および送信用のゲートウェイをインストールします。次の設定を使用して、**ServiceMeshControlPlane** (SMCP) リソースでゲートウェイのインストールを無効にすることができます。

- **spec.gateways.enabled=false** は、ingress ゲートウェイと egress ゲートウェイの両方を無効にします。

- **spec.gateways.ingress.enabled=false** は、ingress ゲートウェイを無効にします。
- **spec.gateways.egress.enabled=false** は、egress ゲートウェイを無効にします。

注記

Operator はデフォルトゲートウェイにアノテーションを付けて、それらが Red Hat OpenShift Service Mesh Operator によって生成および管理されていることを示します。

1.5.1.14. マルチクラスター設定

Red Hat OpenShift Service Mesh は、マルチクラスター設定をサポートしません。

1.5.1.15. カスタム証明書署名要求 (CSR)

Kubernetes 認証局 (CA) で CSR を処理するように Red Hat OpenShift Service Mesh を設定することはできません。

1.5.1.16. Istio ゲートウェイのルート

Istio ゲートウェイの OpenShift ルートは、Red Hat OpenShift Service Mesh で自動的に管理されます。Istio ゲートウェイがサービスメッシュ内で作成され、更新され、削除されるたびに、OpenShift ルートが作成され、更新され、削除されます。

Istio OpenShift Routing (IOR) と呼ばれる Red Hat OpenShift Service Mesh コントロールプレーンコンポーネントは、ゲートウェイルートを同期させます。詳細は、自動ルートの作成について参照してください。

1.5.1.16.1. catch-all ドメイン

catch-all ドメイン ("*") はサポートされません。ゲートウェイ定義で catch-all ドメインが見つかった場合、Red Hat OpenShift Service Mesh はルートを 作成します が、デフォルトのホスト名を作成するには OpenShift に依存します。つまり、新たに作成されたルートは、catch all ("*") ルート ではなく、代わりに **<route-name>[-<project>].<suffix>** 形式のホスト名を持ちます。デフォルトのホスト名の仕組みや、**cluster-admin** がカスタマイズできる仕組みについての詳細は、OpenShift Container Platform ドキュメントを参照してください。Red Hat OpenShift Dedicated を使用する場合は、Red Hat OpenShift Dedicated の **dedicated-admin** ロールを参照してください。

1.5.1.16.2. サブドメイン

サブドメイン (e.g.: "*.domain.com") はサポートされます。ただし、この機能は OpenShift Container Platform ではデフォルトで有効になっていません。つまり、Red Hat OpenShift Service Mesh はサブドメインを持つルートを 作成します が、これは OpenShift Container Platform が有効にするように設定されている場合にのみ有効になります。

1.5.1.16.3. トランスポート層セキュリティー

トランスポート層セキュリティー (TLS) がサポートされます。ゲートウェイに **tls** セクションが含まれる場合、OpenShift ルートは TLS をサポートするように設定されます。

関連情報

- 自動ルート作成

1.5.2. マルチテナントインストール

アップストリームの Istio は単一テナントのアプローチをとりますが、Red Hat OpenShift Service Mesh はクラスター内で複数の独立したコントロールプレーンをサポートします。Red Hat OpenShift Service Mesh はマルチテナント Operator を使用して、コントロールプレーンのライフサイクルを管理します。

Red Hat OpenShift Service Mesh は、デフォルトでマルチテナントコントロールプレーンをインストールします。Service Meshにアクセスできるプロジェクトを指定し、Service Meshを他のコントロールプレーンインスタンスから分離します。

1.5.2.1. マルチテナンシーとクラスター全体のインストールの比較

マルチテナントインストールとクラスター全体のインストールの主な違いは、istod で使用される権限の範囲です。コンポーネントでは、クラスタースコープのロールベースのアクセス制御 (RBAC) リソース **ClusterRoleBinding** が使用されなくなりました。

ServiceMeshMemberRoll members 一覧のすべてのプロジェクトには、コントロールプレーンのデプロイメントに関連付けられた各サービスアカウントの **RoleBinding** があり、各コントロールプレーンのデプロイメントはそれらのメンバープロジェクトのみを監視します。各メンバープロジェクトには **maistra.io/member-of** ラベルが追加されており、**member-of** の値はコントロールプレーンのインストールが含まれるプロジェクトになります。

Red Hat OpenShift Service Mesh は各メンバープロジェクトを設定し、それ自体、コントロールプレーン、および他のメンバープロジェクト間のネットワークアクセスを確保できるようにします。正確な設定は、OpenShift Container Platform のソフトウェア定義ネットワーク (SDN) の設定方法によって異なります。詳細は、OpenShift SDN についてを参照してください。

OpenShift Container Platform クラスターが SDN プラグインを使用するように設定されている場合:

- **NetworkPolicy**: Red Hat OpenShift Service Mesh は、各メンバープロジェクトで **NetworkPolicy** リソースを作成し、他のメンバーおよびコントロールプレーンからのすべての Pod に対する Ingress を許可します。Service Meshからメンバーを削除すると、この **NetworkPolicy** リソースがプロジェクトから削除されます。

  注記

  また、これにより Ingress がメンバープロジェクトのみに制限されます。メンバー以外のプロジェクトの Ingress が必要な場合は、**NetworkPolicy** を作成してそのトラフィックを許可する必要があります。

- **Multitenant**: Red Hat OpenShift Service Mesh は、各メンバープロジェクトの **NetNamespace** をコントロールプレーンプロジェクトの **NetNamespace** に追加します (**oc adm pod-network join-projects --to control-plane-project member-project** の実行と同じです)。Service Meshからメンバーを削除すると、その **NetNamespace** はコントロールプレーンから分離されます (**oc adm pod-network isolate-projects member-project** の実行と同じです)。

- **Subnet**: 追加の設定は実行されません。

1.5.2.2. クラスタースコープのリソース

アップストリーム Istio には、依存するクラスタースコープのリソースが 2 つあります。**MeshPolicy** および **ClusterRbacConfig**。これらはマルチテナントクラスターと互換性がなく、以下で説明されているように置き換えられました。

- コントロールプレーン全体の認証ポリシーを設定するために、MeshPolicy は **ServiceMeshPolicy** に置き換えられます。これは、コントロールプレーンと同じプロジェクトに作成する必要があります。
- コントロールプレーン全体のロールベースのアクセス制御を設定するために、ClusterRbacConfig は **ServicemeshRbacConfig** に置き換えられます。これは、コントロールプレーンと同じプロジェクトに作成する必要があります。

1.5.3. Kiali とサービスメッシュ

OpenShift Container Platform での Service Mesh を使用した Kiali のインストールは、複数の点でコミュニティーの Kiali インストールとは異なります。以下の変更点は、問題の解決、追加機能の提供、OpenShift Container Platform へのデプロイ時の差異の処理を実行するために必要になることがあります。

- Kiali はデフォルトで有効になっています。
- Ingress はデフォルトで有効になっている。
- Kiali ConfigMap が更新されている。
- Kiali の ClusterRole 設定が更新されている。
- 変更は Service Mesh または Kiali Operator によって上書きされる可能性があるため、ConfigMap を編集しないでください。Kiali Operator が管理するファイルには、**kiali.io**/ ラベルまたはアノテーションが付いています。Operator ファイルの更新は、**cluster-admin** 権限を持つユーザーに制限する必要があります。Red Hat OpenShift Dedicated を使用する場合に、**dedicated-admin** 権限のあるユーザーだけが Operator ファイルを更新できるようにする必要があります。

1.5.4. 分散トレースとサービスメッシュ

OpenShift Container Platform での Service Mesh を使用した分散トレースプラットフォームのインストールは、複数の点でコミュニティーの Jaeger インストールとは異なります。以下の変更点は、問題の解決、追加機能の提供、OpenShift Container Platform へのデプロイ時の差異の処理を実行するために必要になることがあります。

- 分散トレースは、Service Mesh に対してデフォルトで有効にされています。
- Ingress は、Service Mesh に対してデフォルトで有効にされています。
- Zipkin ポート名が、(**http** から) **jaeger-collector-zipkin** に変更されています。
- Jaeger は、**production** または **streaming** デプロイメントオプションのいずれかを選択する際に、デフォルトでストレージに Elasticsearch を使用します。
- Istio のコミュニティーバージョンは、一般的なトレースルートを提供します。Red Hat OpenShift Service Mesh は Red Hat OpenShift 分散トレーシプラットフォーム Operator によってインストールされ、OAuth によってすでに保護されている jaeger ルートを使用します。
- Red Hat OpenShift Service Mesh は Envoy プロキシーにサイドカーを使用し、Jaeger も Jaeger エージェントにサイドカーを使用します。両者は個別に設定し、混同しないようにしてください。プロキシーサイドカーは、Pod の Ingress および Egress トラフィックに関連するスパンを作成します。エージェントサイドカーは、アプリケーションによって出力されるスパンを受け取り、これらを Jaeger Collector に送信します。

1.6. SERVICE MESH のインストールの準備

Red Hat OpenShift Service Mesh をインストールする前に、OpenShift Container Platform にサブスクライブし、サポート対象の設定で OpenShift Container Platform をインストールする必要があります。

1.6.1. 前提条件

- お使いの Red Hat アカウントに有効な OpenShift Container Platform サブスクリプションを用意します。サブスクリプションをお持ちでない場合は、営業担当者にお問い合わせください。
- OpenShift Container Platform 4.7 の概要 を確認します。
- OpenShift Container Platform 4.7 をインストールします。制限されたネットワーク に Red Hat OpenShift Service Mesh をインストールする場合、選択した OpenShift Container Platform インフラストラクチャーの手順に従います。
  - AWS への OpenShift Container Platform 4.7 のインストール
  - ユーザーによってプロビジョニングされた AWS への OpenShift Container Platform 4.7 のインストール
  - ベアメタルへの OpenShift Container Platform 4.7 のインストール
  - vSphere への OpenShift Container Platform 4.7 のインストール
  - IBM Z および LinuxONE への OpenShift Container Platform 4.7 のインストール
  - IBM Power Systems への OpenShift Container Platform 4.7 のインストール
- OpenShift Container Platform バージョンに一致する OpenShift Container Platform コマンドラインユーティリティーのバージョン (**oc** クライアントツール) をインストールし、これをパスに追加します。
  - OpenShift Container Platform 4.7 を使用している場合は、OpenShift CLI について を参照してください。

Red Hat OpenShift Service Mesh のライフサイクルおよびサポートされるプラットフォームについての詳細は、サポートポリシーについて参照してください。

1.6.2. サポートされる構成

以下の設定は、Red Hat OpenShift Service Mesh の現行リリースでサポートされます。

1.6.2.1. サポートされるプラットフォーム

Red Hat OpenShift Service Mesh Operator は、複数のバージョンの **ServiceMeshControlPlane** リソースをサポートします。バージョン 2.2 の Service Mesh コントロールプレーンは、以下のプラットフォームバージョンでサポートされます。

- Red Hat OpenShift Container Platform バージョン 4.9 以降
- Red Hat OpenShift Dedicated バージョン 4
- Azure Red Hat OpenShift (ARO) バージョン 4
- Red Hat OpenShift Service on AWS (ROSA)

1.6.2.2. サポートされない設定

明示的にサポート対象外とされているケースには、以下が含まれます。

- OpenShift Online は Red Hat OpenShift Service Mesh に対してはサポートされていません。
- Red Hat OpenShift Service Mesh では、Service Mesh が実行されているクラスター以外にあるマイクロサービスの管理はサポートしていません。

1.6.2.3. サポートされるネットワーク設定

Red Hat OpenShift Service Mesh は以下のネットワーク設定をサポートします。

- OpenShift-SDN
- OVN-Kubernetes は OpenShift Container Platform 4.7.32+、OpenShift Container Platform 4.8.12+、および OpenShift Container Platform 4.9+ でサポートされます。
- OpenShift Container Platform で認定され、さらに Service Mesh 適合テストに合格したサードパーティーの Container Network Interface(CNI) プラグイン。詳細は、認定 OpenShift CNI プラグイン を参照してください。

1.6.2.4. Service Mesh でサポートされる設定

- Red Hat OpenShift Service Mesh の本リリースは、OpenShift Container Platform x86_64、IBM Z、および IBM Power Systems でのみ利用できます。
  - IBM Z は OpenShift Container Platform 4.6 以降でのみサポートされます。
  - IBM Power Systems は OpenShift Container Platform 4.6 以降でのみサポートされます。
- すべての Service Mesh コンポーネントが単一の OpenShift Container Platform クラスター内に含まれる設定。
- 仮想マシンなどの外部サービスを統合しない設定。
- Red Hat OpenShift Service Mesh は、明示的に文書化されている場合を除き、**EnvoyFilter** の設定はサポートしていません。

1.6.2.5. Kiali のサポートされる設定

- Kiali コンソールは Chrome、Edge、Firefox、または Safari ブラウザーの 2 つの最新リリースでのみサポートされています。

1.6.2.6. 分散トレースのサポートされる設定

- サイドカーとしての Jaeger エージェントは、Jaeger について唯一サポートされる設定です。デーモンセットとしての Jaeger はマルチテナントインストールまたは OpenShift Dedicated ではサポートされません。

1.6.2.7. サポート対象の WebAssembly モジュール

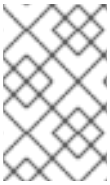
- 3scale WebAssembly は、提供されている唯一の WebAssembly モジュールです。カスタム WebAssembly モジュールを作成できます。

1.6.3. 次のステップ

- OpenShift Container Platform 環境に Red Hat OpenShift Service Mesh をインストール します。

1.7. OPERATOR のインストール

Red Hat OpenShift Service Mesh をインストールするには、まず必要な Operator を OpenShift Container Platform にインストールし、コントロールプレーンをデプロイするために **ServiceMeshControlPlane** リソースを作成します。



注記

この基本的なインストールはデフォルトの OpenShift 設定に基づいて設定され、実稼働環境での使用を目的としていません。 このデフォルトインストールを使用してインストールを確認し、お使いの環境にサービスメッシュを設定します。

前提条件

- Red Hat OpenShift Service Mesh のインストールの準備 のプロセスを確認してください。
- **cluster-admin** ロールを持つアカウントがある。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

以下の手順では、OpenShift Container Platform に Red Hat OpenShift Service Mesh の基本的なインスタンスをインストールする方法について説明します。

1.7.1. Operator の概要

Red Hat OpenShift Service Mesh には、以下の 4 つの Operator が必要です。

- **OpenShift Elasticsearch**:(オプション) 分散トレースプラットフォームでのトレースおよびロギング用にデータベースストレージを提供します。これはオープンソースの Elasticsearch プロジェクトに基づいています。
- **Red Hat OpenShift** 分散トレースプラットフォーム: 複雑な分散システムでのトランザクションを監視し、トラブルシューティングするための分散トレース機能を提供します。これはオープンソースの Jaeger プロジェクトに基づいています。
- **Kiali**: サービスメッシュの可観測性を実現します。これにより、単一のコンソールで設定を表示し、トラフィックを監視し、トレースの分析を実行できます。これはオープンソースの Kiali プロジェクトに基づいています。
- **Red Hat OpenShift Service Mesh**: アプリケーションを設定するマイクロサービスを接続し、保護し、制御し、観察することができます。Service Mesh Operator は、Service Mesh コンポーネントのデプロイメント、更新、および削除を管理する **ServiceMeshControlPlane** リソースを定義し、監視します。これはオープンソースの Istio プロジェクトに基づいています。

警告

Operator のコミュニティーバージョンはインストールしないでください。コミュニティー Operator はサポートされていません。

1.7.2. Operator のインストール

Red Hat OpenShift Service Mesh をインストールするには、以下の Operator をこの順序でインストールします。Operator ごとに手順を繰り返します。

- OpenShift Elasticsearch
- Red Hat OpenShift 分散トレースプラットフォーム
- Kiali
- Red Hat OpenShift Service Mesh

注記

OpenShift Logging の一部として OpenShift Elasticsearch Operator がすでにインストールされている場合、OpenShift Elasticsearch Operator を再びインストールする必要はありません。Red Hat OpenShift 分散トレースプラットフォーム Operator はインストールされた OpenShift Elasticsearch Operator を使用して Elasticsearch インスタンスを作成します。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
2. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** をクリックします。
3. Operator のフィルターボックスに名前を入力し、Red Hat バージョンの Operator を選択します。Operator のコミュニティーバージョンはサポートされていません。
4. **Install** をクリックします。
5. 各 Operator の **Install Operator** ページで、デフォルト設定を受け入れます。
6. **Install** をクリックします。Operator がインストールされるまで待機してから、一覧で次に来る Operator で手順を繰り返します。
   - OpenShift Elasticsearch Operator は、**openshift-operators-redhat** namespace にインストールされ、クラスター内のすべての namespace で使用できます。
   - Red Hat OpenShift 分散トレースプラットフォームは、**openshift-distributed-tracing** namespace にインストールされ、クラスター内のすべての namespace で使用できます。

- Kiali および Red Hat Service Mesh Operator は、**openshift-operators** namespace にインストールされ、クラスター内のすべての namespace で使用できます。

7. 4 つの Operator すべてをインストールしたら、**Operators → Installed Operators** をクリックし、Operator がインストールされていることを確認します。

1.7.3. 次のステップ

Red Hat OpenShift Service Mesh Operator は、Service Mesh コントロールプレーンをデプロイするまで、さまざまな Service Mesh カスタムリソース定義 (CRD) を作成しません。**ServiceMeshControlPlane** リソースを使用して、Service Mesh コンポーネントをインストールおよび設定します。詳細は、ServiceMeshControlPlane の作成 を参照してください。

1.8. SERVICEMESHCONTROLPLANE の作成

OpenShift Container Platform Web コンソールを使用するか、または **oc** クライアントツールを使用してコマンドラインから **ServiceMeshControlPlane** (SMCP) の基本的なインストールをデプロイできます。

注記

この基本的なインストールはデフォルトの OpenShift 設定に基づいて設定され、実稼働環境での使用を目的としていません。このデフォルトインストールを使用してインストールを確認し、お使いの環境に **ServiceMeshControlPlane** を設定します。

注記

Red Hat OpenShift Service on AWS (ROSA) では、リソースを作成できる場所に関して追加の制限が適用されるので、デフォルトのデプロイメントは機能しません。ROSA 環境に SMCP をデプロイする前に、追加の要件について、Red Hat OpenShift Service on AWS (ROSA) へのインストールを参照してください。

注記

Service Mesh に関するドキュメントは **istio-system** をサンプルプロジェクトとして使用しますが、サービスメッシュを任意のプロジェクトにデプロイできます。

1.8.1. Web コンソールからの Service Mesh コントロールプレーンのデプロイ

Web コンソールを使用して基本的な **ServiceMeshControlPlane** をデプロイできます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている必要がある。
- **cluster-admin** ロールを持つアカウントがある。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

2. **istio-system** という名前のプロジェクトを作成します。

   a. **Home → Projects** に移動します。

   b. **Create Project** をクリックします。

   c. **Name** フィールドに istio-system と入力します。**ServiceMeshControlPlane** リソースは、マイクロサービスおよび Operator とは異なるプロジェクトにインストールする必要があります。
   これらのステップは **istio-system** を例として使用しますが、サービスが含まれるプロジェクトから分離されない限り、Service Mesh コントロールプレーンを任意のプロジェクトにデプロイすることができます。

   d. **Create** をクリックします。

3. **Operators → Installed Operators** に移動します。

4. Red Hat OpenShift Service Mesh Operator をクリックした後に、**Istio Service Mesh Control Plane** をクリックします。

5. **Istio Service Mesh Control Plane**タブで **Create ServiceMeshControlPlane** をクリックします。

6. **Create ServiceMeshControlPlane** ページで、デフォルトの Service Mesh コントロールプレーンバージョンを受け入れて、製品の最新バージョンで利用可能な機能を活用します。コントロールプレーンのバージョンは、Operator のバージョンに関係なく利用可能な機能を判別します。
   ServiceMeshControlPlane 設定は後で設定できます。詳細は、Red Hat OpenShift Service Mesh の設定を参照してください。

   a. **Create** をクリックします。Operator は、設定パラメーターに基づいて Pod、サービス、Service Mesh コントロールプレーンのコンポーネントを作成します。

7. **Istio Service Mesh Control Plane**タブをクリックしてコントロールプレーンが正常にインストールされることを確認します。

   a. 新規コントロールプレーンの名前をクリックします。

   b. **Resources** タブをクリックして、Red Hat OpenShift Service Mesh コントロールプレーンリソース (Operator が作成し、設定したもの) を表示します。

1.8.2. CLI を使用した Service Mesh コントロールプレーンのデプロイ

コマンドラインから基本的な **ServiceMeshControlPlane** をデプロイできます。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている必要がある。
- OpenShift CLI (**oc**) へのアクセスがある。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

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

2. **istio-system** という名前のプロジェクトを作成します。

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

3. 以下の例を使用して **istio-installation.yaml** という名前の **ServiceMeshControlPlane** ファイルを作成します。Service Mesh コントロールプレーンのバージョンは、Operator のバージョンに関係なく利用可能な機能を判別します。

バージョン 2.2 istio-installation.yaml の例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.2
  tracing:
    type: Jaeger
    sampling: 10000
  addons:
    jaeger:
      name: jaeger
      install:
        storage:
          type: Memory
    kiali:
      enabled: true
      name: kiali
    grafana:
      enabled: true
```

4. 以下のコマンドを実行して Service Mesh コントロールプレーンをデプロイします。ここで、**<istio_installation.yaml>** にはファイルへの完全パスが含まれます。

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

5. Pod のデプロイメントの進行状況を監視するには、次のコマンドを実行します。

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

以下のような出力が表示されるはずです。

```
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
wasm-cacher-basic-8c986c75-vj2cd        1/1     Running   0          65m
```

-

1.8.3. CLI を使用した SMCP インストールの検証

コマンドラインから **ServiceMeshControlPlane** の作成を検証できます。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

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

2. 次のコマンドを実行して、Service Mesh コントロールプレーンのインストールを確認します。**istio-system** は、Service Mesh コントロールプレーンをインストールした namespace です。

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

STATUS 列が **ComponentsReady** の場合、インストールは正常に終了しています。

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

1.8.4. Kiali を使用した SMCP インストールの検証

Kiali コンソールを使用して、Service Mesh のインストールを検証できます。Kiali コンソールには、Service Mesh コンポーネントが適切にデプロイおよび設定されていることを検証する方法がいくつかあります。

手順

1. cluster-admin 権限を持つユーザーとして OpenShift Container Platform Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

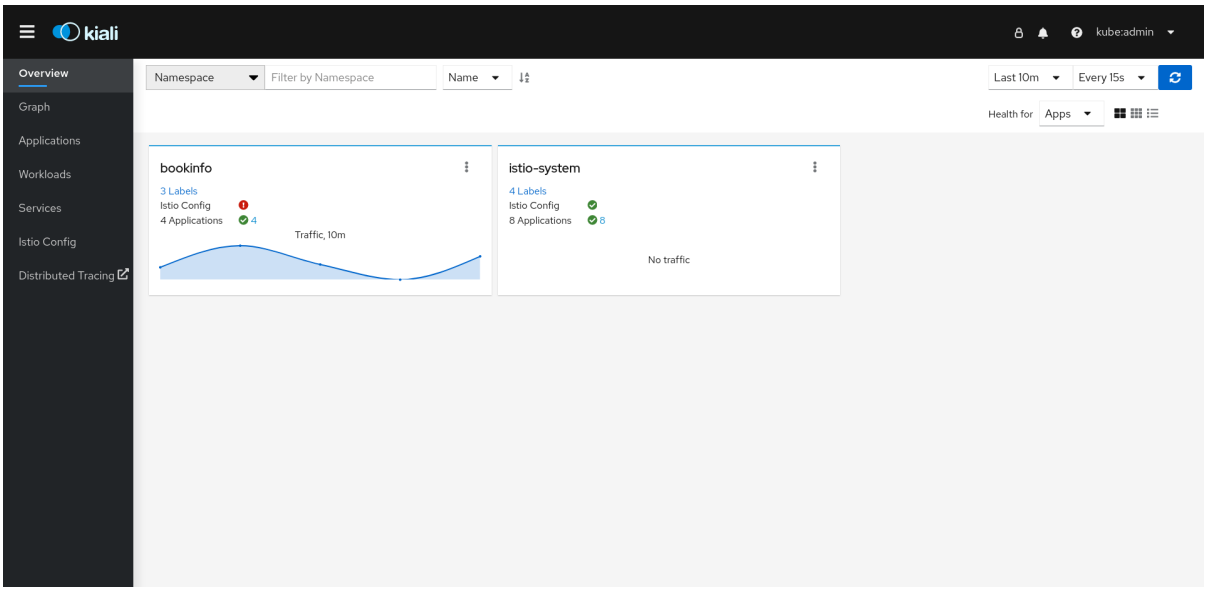
2. **Networking → Routes** に移動します。

3. **Routes** ページで、**Namespace** メニューから Service Mesh コントロールプレーンプロジェクトを選択します (例: **istio-system**)。
Location 列には、各ルートのリンク先アドレスが表示されます。

4. 必要に応じて、フィルターを使用して Kiali コンソールのルートを見つけます。ルートの **Location** をクリックしてコンソールを起動します。

5. **Log In With OpenShift** をクリックします。
初回の Kiali コンソールへのログイン時に、表示するパーミッションを持つサービスメッシュ内のすべての namespace を表示する **Overview** ページが表示されます。概要 ページに複数の namespace が表示されている場合には、Kiali は最初に正常性または検証に問題がある namespace を表示します。

図1.1 Kiali の概要ページ



各 namespace のタイルには、ラベルの数、**Istio Config** の状態、アプリケーション の状態と数、namespace の トラフィック が表示されます。コンソールのインストールを検証中で、namespace がまだメッシュに追加されていない場合、**istio-system** 以外のデータは表示されない可能性があります。

6. Kiali には、Service Mesh コントロールプレーンがインストールされている namespace 専用のダッシュボードが 4 つあります。これらのダッシュボードを表示するには、オプション メニューをクリックします ⋮ コントロールプレーン namespace のタイル (例: **istio-system**) で、次のいずれかのオプションを選択します。

- **Istio** メッシュダッシュボード
- **Istio** コントロールプレーンダッシュボード
- **Istio** パフォーマンスダッシュボード
- **Istio Wasm Exetension** ダッシュボード

図1.2 GrafanaIstio コントロールプレーンダッシュボード

Kiali は、Grafana ホームページ から入手できる追加の Grafana ダッシュボード 2 つもインストールします。

- **Istio** ワークロードダッシュボード
- **Istio** サービスダッシュボード

7. Service Mesh コントロールプレーンノードを表示するには、グラフ ページをクリックし、メニューから **ServiceMeshControlPlane** をインストールした **Namespace** を選択します (例 **:istio-system**)。
   a. 必要に応じて、**Display idle nodes** をクリックします。
   b. グラフ ページの詳細は、グラフツアー リンクをクリックしてください。
   c. メッシュトポロジーを表示するには、namespace メニューの Service Mesh メンバーロールから追加の **namespace** を 1 つまたは複数選択します。
8. **istio-system** namespace 内のアプリケーションのリストを表示するには、アプリケーション ページをクリックします。Kiali は、アプリケーションの状態を表示します。
   a. 情報アイコンの上にマウスをかざすと、詳細 列に記載されている追加情報が表示されます。
9. **istio-system** namespace のワークロードのリストを表示するには、ワークロード ページをクリックします。Kiali は、ワークロードの状態を表示します。
   a. 情報アイコンの上にマウスをかざすと、詳細 列に記載されている追加情報が表示されます。
10. **istio-system** namespace のサービスのリストを表示するには、サービス ページをクリックします。Kiali は、サービスと設定の状態を表示します。
    a. 情報アイコンの上にマウスをかざすと、詳細 列に記載されている追加情報が表示されます。
11. **istio-system** namespace の Istio 設定オブジェクトのリストを表示するには、**Istio Config** ページをクリックします。Kiali は、設定の正常性を表示します。
    a. 設定エラーがある場合は、行をクリックすると、Kiali が設定ファイルを開き、エラーが強調表示されます。

1.8.5. Red Hat OpenShift Service on AWS (ROSA) へのインストール

バージョン 2.2 以降、Red Hat OpenShift Service Mesh は Red Hat OpenShift Service on AWS (ROSA) へのインストールがサポートされます。本セクションでは、このプラットフォームに Service Mesh をインストールする際の追加の要件について説明します。

1.8.5.1. インストールの場所

Red Hat OpenShift Service Mesh をインストールし、**ServiceMeshControlPlane** を作成する際に、**istio-system** などの新規 namespace を作成する必要があります。

1.8.5.2. 必要な Service Mesh コントロールプレーンの設定

ServiceMeshControlPlane ファイルのデフォルト設定は ROSA クラスターでは機能しません。Red Hat OpenShift Service on AWS にインストールする場合は、デフォルトの SMCP を変更し、**spec.security.identity.type=ThirdParty** を設定する必要があります。

ROSA 用の ServiceMeshControlPlane リソースの例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.2
  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
```

1.8.5.3. Kiali 設定の制限

Red Hat OpenShift Service on AWS では、リソースを作成できる場所に関して追加の制限が適用され、Red Hat 管理の namespace に Kiali リソースを作成することはできません。

つまり、ROSA クラスターでは、**spec.deployment.accessible_namespaces** の以下の共通設定は許可されません。

- **['**']** (すべての namespaces)
- **default**
- **codeready-***
- **openshift-***
- **redhat-***

検証エラーメッセージでは、制限されたすべての namespace の完全なリストが提供されます。

ROSA 用の Kiali リソースの例

```
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
```

1.8.6. 関連情報

Red Hat OpenShift Service Mesh はクラスター内で複数の独立したコントロールプレーンをサポートします。**ServiceMeshControlPlane** プロファイルを使用すると、再利用可能な設定を作成することができます。詳細は、コントロールプレーンプロファイルの作成 を参照してください。

1.8.7. 次のステップ

ServiceMeshMemberRoll リソースを作成し、Service Mesh に関連付けられた namespace を指定します。詳細は、サービスメッシュへのサービスの追加 を参照してください。

1.9. サービスメッシュへのサービスの追加

Operator および **ServiceMeshControlPlane** リソースのインストール後に、**ServiceMeshMemberRoll** リソースを作成し、コンテンツが置かれている namespace を指定して、アプリケーション、ワークロード、またはサービスをメッシュに追加します。**ServiceMeshMemberRoll** リソースに追加するアプリケーション、ワークロード、またはサービスがすでにある場合には、以下の手順に従います。または、Bookinfo と呼ばれるサンプルアプリケーションをインストールして **ServiceMeshMemberRoll** リソースに追加するには、Bookinfo サンプルアプリケーション のチュートリアルまで進み、アプリケーションが Red Hat OpenShift Service Mesh でどのように機能するかを確認します。

ServiceMeshMemberRoll リソースに記載される項目は、**ServiceMeshControlPlane** リソースで管理されるアプリケーションおよびワークフローです。コントロールプレーン (Service Mesh Operator、Istiod、および **ServiceMeshControlPlane**) およびデータプレーン (アプリケーションおよび Envoy プロキシー) は別々の namespace にある必要があります。

注記

namespace を **ServiceMeshMemberRoll** に追加した後に、その namespace のサービスまたは Pod へのアクセスはサービスメッシュ外の呼び出し元からアクセスできません。

1.9.1. Red Hat OpenShift Service Mesh メンバーロールの作成

ServiceMeshMemberRoll は、Service Mesh コントロールプレーンに属するプロジェクトを一覧表示します。**ServiceMeshMemberRoll** に一覧表示されているプロジェクトのみがコントロールプレーンの影響を受けます。プロジェクトは、特定のコントロールプレーンのデプロイメント用にメンバーロールに追加するまでサービスメッシュに属しません。

istio-system など、**ServiceMeshControlPlane** と同じプロジェクトに、 **default** という名前の **ServiceMeshMemberRoll** リソースを作成する必要があります。

1.9.1.1. Web コンソールからのメンバーロールの作成

Web コンソールを使用して 1 つ以上のプロジェクトを Service Mesh メンバーロールに追加します。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされ、検証されていること。
- サービスメッシュに追加する既存プロジェクトの一覧。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. メッシュのサービスがない場合や、ゼロから作業を開始する場合は、アプリケーションのプロジェクトを作成します。これは、Service Mesh コントロールプレーンをインストールしたプロジェクトとは異なる必要があります。
   a. **Home → Projects** に移動します。
   b. **Name** フィールドに名前を入力します。
   c. **Create** をクリックします。
3. **Operators → Installed Operators** に移動します。
4. **Project** メニューをクリックし、一覧から **ServiceMeshControlPlane** リソースがデプロイされているプロジェクト (例: **istio-system**) を選択します。
5. Red Hat OpenShift Service Mesh Operator をクリックします。
6. **Istio Service Mesh Member Roll**タブをクリックします。
7. **Create ServiceMeshMemberRoll** をクリックします。
8. **Members** をクリックし、**Value** フィールドにプロジェクトの名前を入力します。任意の数のプロジェクトを追加できますが、プロジェクトは 単一 の **ServiceMeshMemberRoll** リソースしか属することができません。
9. **Create** をクリックします。

1.9.1.2. CLI からのメンバーロールの作成

コマンドラインからプロジェクトを **ServiceMeshMemberRoll** に追加します。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされ、検証されていること。
- サービスメッシュに追加するプロジェクトの一覧。
- OpenShift CLI (**oc**) へのアクセスがある。

手順

1. OpenShift Container Platform CLI にログインします。

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

2. メッシュのサービスがない場合や、ゼロから作業を開始する場合は、アプリケーションのプロジェクトを作成します。これは、Service Mesh コントロールプレーンをインストールしたプロジェクトとは異なる必要があります。

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

3. プロジェクトをメンバーとして追加するには、以下の YAML の例を変更します。任意の数のプロジェクトを追加できますが、プロジェクトは 単一 の **ServiceMeshMemberRoll** リソースしか属することができません。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

servicemeshmemberroll-default.yaml の例

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

4. 以下のコマンドを実行して、**istio-system** namespace に **ServiceMeshMemberRoll** リソースをアップロードおよび作成します。

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

5. 以下のコマンドを実行して、**ServiceMeshMemberRoll** が正常に作成されていることを確認します。

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

STATUS 列が **Configured** の場合には、インストールは正常に終了しています。

1.9.2. サービスメッシュからのプロジェクトの追加または削除

Web コンソールを使用して既存の Service Mesh **ServiceMeshMemberRoll** リソースを追加または削除できます。

- 任意の数のプロジェクトを追加できますが、プロジェクトは 単一 の **ServiceMeshMemberRoll** リソースしか属することができません。
- **ServiceMeshMemberRoll** リソースは、対応する **ServiceMeshControlPlane** リソースが削除されると削除されます。

1.9.2.1. Web コンソールを使用したメンバーロールからのプロジェクトの追加または削除

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされ、検証されていること。
- 既存の **ServiceMeshMemberRoll** リソース。
- **ServiceMeshMemberRoll** リソースを持つプロジェクトの名前。
- メッシュに/から追加または削除するプロジェクトの名前。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operators → Installed Operators** に移動します。
3. **Project** メニューをクリックし、一覧から **ServiceMeshControlPlane** リソースがデプロイされているプロジェクト (例: **istio-system**) を選択します。
4. Red Hat OpenShift Service Mesh Operator をクリックします。
5. **Istio Service Mesh Member Roll**タブをクリックします。
6. **default** リンクをクリックします。
7. YAML タブをクリックします。
8. YAML を変更して、プロジェクトをメンバーとして追加または削除します。任意の数のプロジェクトを追加できますが、プロジェクトは 単一 の **ServiceMeshMemberRoll** リソースしか属することができません。
9. **Save** をクリックします。
10. **Reload** をクリックします。

1.9.2.2. CLI を使用したメンバーロールからのプロジェクトの追加または削除

コマンドラインを使用して既存の Service Mesh Member Roll を変更できます。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされ、検証されていること。
- 既存の **ServiceMeshMemberRoll** リソース。
- **ServiceMeshMemberRoll** リソースを持つプロジェクトの名前。
- メッシュに/から追加または削除するプロジェクトの名前。
- OpenShift CLI (**oc**) へのアクセスがある。

手順

1. OpenShift Container Platform CLI にログインします。

2. **ServiceMeshMemberRoll** リソースを編集します。

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

3. YAML を変更して、プロジェクトをメンバーとして追加または削除します。任意の数のプロジェクトを追加できますが、プロジェクトは 単一 の **ServiceMeshMemberRoll** リソースしか属することができません。

servicemeshmemberroll-default.yaml の例

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

1.9.3. Bookinfo のサンプルアプリケーション

Bookinfo のサンプルアプリケーションでは、OpenShift Container Platform での Red Hat OpenShift Service Mesh 2.2.3 のインストールをテストすることができます。

Bookinfo アプリケーションは、オンラインブックストアの単一カタログエントリーのように、書籍に関する情報を表示します。このアプリケーションでは、書籍の説明、書籍の詳細 (ISBN、ページ数その他の情報)、および書評のページが表示されます。

Bookinfo アプリケーションはこれらのマイクロサービスで設定されます。

- **productpage** マイクロサービスは、**details** と **reviews** マイクロサービスを呼び出して、ページを設定します。
- **details** マイクロサービスには書籍の情報が含まれています。
- **reviews** マイクロサービスには、書評が含まれます。これは **ratings** マイクロサービスも呼び出します。
- **ratings** マイクロサービスには、書評を伴う書籍のランキング情報が含まれます。

reviews マイクロサービスには、以下の 3 つのバージョンがあります。

- バージョン v1 は、**ratings** サービスを呼び出しません。
- バージョン v2 は、**ratings** サービスを呼び出して、各評価を 1 から 5 の黒い星で表示します。
- バージョン v3 は、**ratings** サービスを呼び出して、各評価を 1 から 5 の赤い星で表示します。

1.9.3.1. Bookinfo アプリケーションのインストール

このチュートリアルでは、プロジェクトの作成、そのプロジェクトへの Bookinfo アプリケーションのデプロイ、Service Mesh での実行中のアプリケーションの表示を行い、サンプルアプリケーションを作成する方法を説明します。

前提条件:

- OpenShift Container Platform 4.1 以降がインストールされている。
- Red Hat OpenShift Service Mesh 2.2.3 がインストールされている。
- OpenShift CLI (**oc**) へのアクセスがある。
- **cluster-admin** ロールを持つアカウントがある。

注記

Bookinfo サンプルアプリケーションは、IBM Z および IBM Power Systems にインストールできません。

注記

このセクションのコマンドは、Service Mesh コントロールプレーンプロジェクトが **istio-system** であると仮定します。コントロールプレーンを別の namespace にインストールしている場合は、実行する前にそれぞれのコマンドを編集します。

手順

1. cluster-admin 権限を持つユーザーとして OpenShift Container Platform Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
2. **Home → Projects** をクリックします。
3. **Create Project** をクリックします。
4. **Project Name** として **bookinfo** を入力し、**Display Name** を入力します。その後、**Description** を入力し、**Create** をクリックします。
   - または、CLI からこのコマンドを実行して、**bookinfo** プロジェクトを作成できます。

     ```
     $ oc new-project bookinfo
     ```

5. **Operators → Installed Operators** をクリックします。
6. プロジェクト メニューをクリックし、Service Mesh コントロールプレーンの namespace を使用します。この例では **istio-system** を使用します。
7. **Red Hat OpenShift Service Mesh** Operator をクリックします。
8. **Istio Service Mesh Member Roll**タブをクリックします。
   a. Istio Service Mesh Member Roll がすでに作成されている場合には、名前をクリックしてから YAML タブをクリックし、YAML エディターを開きます。
   b. **ServiceMeshMemberRoll** を作成していない場合は、**Create ServiceMeshMemberRoll** をクリックします。
9. **Members** をクリックし、**Value** フィールドにプロジェクトの名前を入力します。
10. **Create** をクリックして、更新した Service Mesh Member Roll を保存します。

a. または、以下のサンプルを YAML ファイルに保存します。

Bookinfo ServiceMeshMemberRoll の例 (servicemeshmemberroll-default.yaml)

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

b. 以下のコマンドを実行して、そのファイルをアップロードし、**istio-system** namespace に **ServiceMeshMemberRoll** リソースを作成します。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

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

11. 以下のコマンドを実行して、**ServiceMeshMemberRoll** が正常に作成されていることを確認します。

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

STATUS 列が **Configured** の場合には、インストールは正常に終了しています。

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

12. CLI で **'bookinfo'** プロジェクトに Bookinfo アプリケーションをデプロイするには、**bookinfo.yaml** ファイルを適用します。

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

以下のような出力が表示されるはずです。

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

13. **bookinfo-gateway.yaml** ファイルを適用して Ingress ゲートウェイを作成します。

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

以下のような出力が表示されるはずです。

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

14. **GATEWAY_URL** パラメーターの値を設定します。

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

1.9.3.2. デフォルトの宛先ルールの追加

Bookinfo アプリケーションを使用するには、先にデフォルトの宛先ルールを追加する必要があります。相互トランスポート層セキュリティー (TLS) 認証を有効にしたかどうかによって、2 つの事前設定される YAML ファイルを使用できます。

手順

1. 宛先ルールを追加するには、以下のいずれかのコマンドを実行します。
   - 相互 TLS を有効にしていない場合:

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

   - 相互 TLS を有効にしている場合:

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

     以下のような出力が表示されるはずです。

     ```
     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. Bookinfo インストールの検証

Bookinfo アプリケーションのサンプルが正常にデプロイされたことを確認するには、以下の手順を実行します。

前提条件

- Red Hat OpenShift Service Mesh がインストールされている。
- Bookinfo サンプルアプリケーションのインストール手順を実行します。

CLI からの手順

1. OpenShift Container Platform CLI にログインします。

2. 以下のコマンドですべての Pod が準備状態にあることを確認します。

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

すべての Pod のステータスは **Running** である必要があります。以下のような出力が表示されるはずです。

```
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. 以下のコマンドを実行して、製品ページの URL を取得します。

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

4. Web ブラウザーで出力をコピーして貼り付けて、Bookinfo の製品ページがデプロイされていることを確認します。

Kiali Web コンソールからの手順

1. Kiali Web コンソールのアドレスを取得します。

   a. **cluster-admin** 権限を持つユーザーとして OpenShift Container Platform Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

   b. **Networking → Routes** に移動します。

   c. **Routes** ページで、 **Namespace** メニューから Service Mesh コントロールプレーンプロジェクトを選択します (例: **istio-system**)。
   Location 列には、各ルートのリンク先アドレスが表示されます。

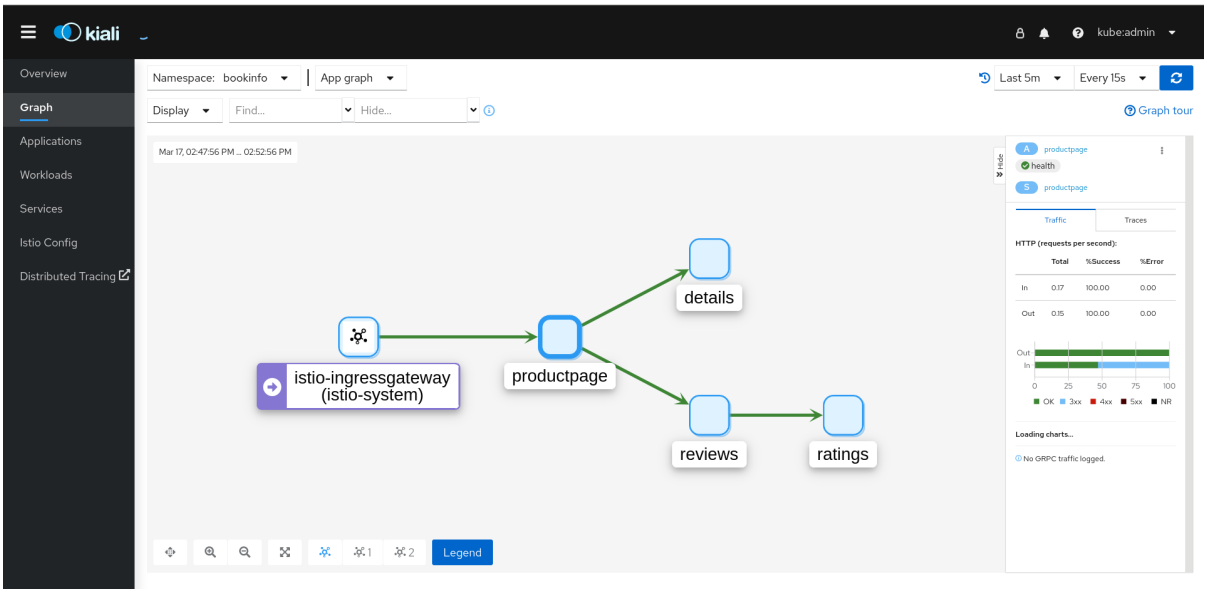
   d. Kiali の 場所 列のリンクをクリックします。

   e. **Log In With OpenShift** をクリックします。Kiali の 概要 画面には、各プロジェクトの namespace のタイルが表示されます。

2. Kiali で、グラフ をクリックします。

3. **Namespace** リストから bookinfo を選択し、 **Graph Type** リストから App graph を選択します。

4. **Display** メニューから **Display idle nodes** をクリックします。
   これにより、定義されているが要求を受信または送信していないノードが表示されます。アプリケーションが適切に定義されていることを確認できますが、要求トラフィックは報告されていません。



- 期間 メニューを使用して、期間を延ばして、古いトラフィックを取得できるようにします。
- **Refresh Rate** メニューを使用して、トラフィックを頻繁に更新するか、まったく更新しないようにします。

5. **Services**、**Workloads** または **Istio Config** をクリックして、bookinfo コンポーネントのリストビューを表示し、それらが正常であることを確認します。

1.9.3.4. Bookinfo アプリケーションの削除

以下の手順で、Bookinfo アプリケーションを削除します。

前提条件

- OpenShift Container Platform 4.1 以降がインストールされている。
- Red Hat OpenShift Service Mesh 2.2.3 がインストールされている。
- OpenShift CLI (**oc**) へのアクセスがある。

1.9.3.4.1. Bookinfo プロジェクトの削除

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Home** → **Projects** をクリックします。
3. **bookinfo** メニュー ⋮ をクリックしてから **Delete Project** をクリックします。
4. 確認ダイアログボックスに **bookinfo** と入力してから **Delete** をクリックします。
   - または、CLI を使用して以下のコマンドを実行し、**bookinfo** プロジェクトを作成できます。

```
$ oc delete project bookinfo
```

1.9.3.4.2. Service Mesh Member Roll からの Bookinfo プロジェクトの削除

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operators → Installed Operators** をクリックします。
3. **Project** メニューをクリックし、一覧から **istio-system** を選択します。
4. **Red Hat OpenShift Service Mesh** Operator の **Provided APIS** で、**Istio Service Mesh Member Roll** のリンクをクリックします。
5. **ServiceMeshMemberRoll** メニュー ⋮ をクリックし、**Edit Service Mesh Member Roll**を選択します。
6. デフォルトの Service Mesh Member Roll YAML を編集し、**members** 一覧から **bookinfo** を削除します。
   - または、CLI を使用して以下のコマンドを実行し、**ServiceMeshMemberRoll** から **bookinfo** プロジェクトを削除できます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

     ```
     $ oc -n istio-system patch --type='json' smmr default -p '[{"op": "remove", "path": "/spec/members", "value":["""bookinfo"""]}]'
     ```
7. **Save** をクリックして、Service Mesh Member Roll を更新します。

1.9.4. 次のステップ

- インストールプロセスを続行するには、サイドカーコンテナーの挿入を有効 にする必要があります。

1.10. サイドカーコンテナーの挿入の有効化

サービスが含まれる namespace をメッシュに追加したら、次の手順は、アプリケーションのデプロイメントリソースでサイドカーの自動挿入を有効にします。デプロイメントごとにサイドカーコンテナーの自動挿入を有効にする必要があります。

Bookinfo サンプルアプリケーションをインストールした場合は、アプリケーションがデプロイされ、インストール手順の一部としてサイドカーが注入されています。独自のプロジェクトおよびサービスを使用している場合は、アプリケーションを OpenShift Container Platform にデプロイします。詳細は、OpenShift Container Platform のドキュメント Understanding Deployment and DeploymentConfig objects を参照してください。

1.10.1. 前提条件

- メッシュにデプロイされたサービス(Bookinfo サンプルアプリケーションなど)。
- デプロイメントリソースファイル。

1.10.2. サイドカーコンテナーの自動挿入の有効化

アプリケーションをデプロイする場合は、**deployment** オブジェクトで **spec.template.metadata.annotations** のアノテーション **sidecar.istio.io/inject** を **true** に設定して、インジェクションをオプトインする必要があります。オプトインにより、サイドカーの挿入が OpenShift Container Platform エコシステム内の複数のフレームワークが使用する、ビルダー Pod などの他の OpenShift Container Platform 機能に干渉しないようにします。

前提条件

- サービスメッシュの一部である namespace と、サイドカーの自動注入が必要なデプロイメントを特定しておく。

手順

1. デプロイメントを見つけるには、**oc get** コマンドを使用します。

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

たとえば、**bookinfo** namespaxce の ratings-v1 マイクロサービスのデプロイメントファイルを表示するには、次のコマンドを使用して YAML 形式でリソースを表示します。

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

2. エディターでアプリケーションのデプロイメント設定の YAML ファイルを開きます。

3. 次の例に示すように、**spec.template.metadata.annotations.sidecar.istio/inject** を Deployment YAML に追加し、**sidecar.istio.io/inject** を **true** に設定します。

bookinfo deployment-ratings-v1.yaml のスニペットの例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ratings-v1
  namespace: bookinfo
  labels:
    app: ratings
    version: v1
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: 'true'
```

4. デプロイメント設定ファイルを保存します。

5. ファイルをアプリケーションが含まれるプロジェクトに追加し直します。

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

この例では、**bookinfo** は **ratings-v1** アプリを含むプロジェクトの名前であり、**deployment-ratings-v1.yaml** は編集したファイルです。

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

6. リソースが正常にアップロードされたことを確認するには、以下のコマンドを実行します。

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

以下に例を示します。

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

1.10.3. サイドカーインジェクションの検証

Kiali コンソールは、アプリケーション、サービス、ワークロードにサイドカープロキシーがあるかどうかを検証するためのいくつかの方法を提供します。

図1.3 サイドカーバッジがない

グラフ ページには、次のグラフに サイドカー がないことを示すノードバッジが表示されます。

- App graph
- Versioned app graph
- Workload graph

図1.4 サイドカーアイコンがない

アプリケーション ページでは、サイドカーがない namespace 内のアプリケーションの 詳細 列に **Missing Sidecar** アイコンが表示されます。

ワークロード ページでは、サイドカーがない namespace 内のアプリケーションの 詳細 列に **Missing Sidecar** アイコンが表示されます。

サービス ページでは、サイドカーがない namespace 内のアプリケーションの 詳細 列に **Missing Sidecar** アイコンが表示されます。サービスのバージョンが複数ある場合は、サービスの詳細 ページを使用して、**Missing Sidecar** アイコンを表示します。

ワークロードの詳細 ページには、アプリケーションログとプロキシーログを表示および相互に関連付けることができる特別な統合 **Logs** タブがあります。Envoy ログは、アプリケーションワークロードのサイドカーインジェクションを検証する別の方法として表示できます。

ワークロードの詳細 ページには、Envoy プロキシーであるか、Envoy プロキシーが注入されたワークロード用の **Envoy** タブもあります。このタブには、クラスター、リスナー、ルート、ブートストラップ、設定、および メトリック のサブタブなど、組み込みの Envoy ダッシュボードが表示されます。

Envoy アクセスログを有効にする方法については、トラブルシューティング のセクションを参照してください。

Envoy ログの表示については、Kiali コンソールでのログの表示 を参照してください。

1.10.4. アノテーションによるプロキシー環境変数の設定

Envoy サイドカープロキシーの設定は、**ServiceMeshControlPlane** によって管理されます。

デプロイメントの Pod アノテーションを **injection-template.yaml** ファイルに追加することにより、アプリケーションのサイドカープロキシーで環境変数を設定できます。環境変数がサイドカーコンテナーに挿入されます。

injection-template.yaml の例

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

警告

独自のカスタムリソースを作成するときは、**maistra.io/** ラベルとアノテーションを含めないでください。これらのラベルとアノテーションは、リソースが Operator によって生成および管理されていることを示しています。独自のリソースの作成時に Operator が生成したリソースからコンテンツをコピーする場合は、**maistra.io/** で始まるラベルやアノテーションを含めないでください。これらのラベルまたはアノテーションを含むリソースは、次回の調整時に Operator によって上書きまたは削除されます。

1.10.5. サイドカープロキシーの更新

サイドカープロキシーの設定を更新するには、アプリケーション管理者はアプリケーション Pod を再起動する必要があります。

デプロイメントで自動のサイドカーコンテナー挿入を使用する場合、アノテーションを追加または変更してデプロイメントの Pod テンプレートを更新することができます。以下のコマンドを実行して Pod を再デプロイします。

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

デプロイメントで自動のサイドカーコンテナー挿入を使用しない場合、デプロイメントまたは Pod で指定されたサイドカーコンテナーイメージを変更して Pod を再起動し、サイドカーコンテナーを手動で更新する必要があります。

1.10.6. 次のステップ

ご使用の環境用に Red Hat OpenShift Service Mesh 機能を設定します。

- セキュリティー
- トラフィック管理
- メトリックス、ログ、およびトレース

1.11. SERVICE MESH のアップグレード

Red Hat OpenShift Service Mesh の最新機能にアクセスするには、最新バージョンである 2.2.3 にアップグレードしてください。

1.11.1. バージョニングについて

Red Hat は、製品リリースにセマンティックバージョニングを使用します。セマンティックバージョニングは、X.Y.Z 形式の 3 つのコンポーネント番号になります。

- X はメジャーバージョンを表します。メジャーリリースは通常、アーキテクチャーの変更、API の変更、スキーマの変更、および同様のメジャー更新など、何らかの最新の変更を意味します。
- Y はマイナーバージョンを表します。マイナーリリースには、下位互換性を維持しながら、新しい機能が含まれています。
- Z はパッチバージョン (z-stream リリースとも呼ばれます) を表します。パッチリリースは、Common Vulnerabilities and Exposures (CVE) に対処し、バグ修正をリリースするために使用されます。通常、新機能はパッチリリースの一部としてリリースされません。

1.11.1.1. バージョニングが Service Mesh のアップグレードに与える影響

実行する更新のバージョンによって、アップグレードプロセスが異なります。

- パッチの更新: パッチのアップグレードは、Operator Lifecycle Manager (OLM) によって管理されます。Operator を更新すると自動的に発生します。
- マイナーアップグレード: マイナーアップグレードでは、最新の Red Hat OpenShift Service Mesh Operator バージョンに更新することと、**ServiceMeshControlPlane** リソースの **spec.version** 値を手動で変更することの両方が必要です。

- メジャーアップグレード: メジャーアップグレードでは、最新の Red Hat OpenShift Service Mesh Operator バージョンに更新することと、**ServiceMeshControlPlane** リソースの **spec.version** 値を手動で変更することの両方が必要です。メジャーアップグレードには後方互換性のない変更が含まれている可能性があるため、手動による追加の変更が必要になる場合があります。

1.11.1.2. Service Mesh のバージョンについて

ご使用のシステムにデプロイした Red Hat OpenShift Service Mesh のバージョンを理解するには、各コンポーネントのバージョンがどのように管理されるかを理解する必要があります。

- **Operator** バージョン: 最新の Operator バージョンは 2.2.3 です。Operator バージョン番号は、現在インストールされている Operator のバージョンのみを示します。Red Hat OpenShift Service Mesh Operator は Service Mesh コントロールプレーンの複数のバージョンをサポートするため、Operator のバージョンはデプロイされた **ServiceMeshControlPlane** リソースのバージョンを決定しません。

重要

最新の Operator バージョンにアップグレードすると、パッチの更新が自動的に適用されますが、Service Mesh コントロールプレーンは最新のマイナーバージョンに自動的にアップグレードされません。

- **ServiceMeshControlPlane** バージョン: **ServiceMeshControlPlane** バージョンは、使用している Red Hat OpenShift Service Mesh のバージョンを決定します。**ServiceMeshControlPlane** リソースの **spec.version** フィールドの値は、Red Hat OpenShift Service Mesh のインストールとデプロイに使用されるアーキテクチャーと設定を制御します。Service Mesh コントロールプレーンを作成する場合は、以下の 2 つの方法のいずれかでバージョンを設定できます。
  - Form View で設定するには、**Control Plane Version** メニューからバージョンを選択します。
  - YAML View で設定するには、YAML ファイルに **spec.version** の値を設定します。

Operator Lifecycle Manager (OLM) は Service Mesh コントロールプレーンのアップグレードをを管理しないため、SMCP を手動でアップグレードしない限り、Operator および **ServiceMeshControlPlane** (SMCP) のバージョン番号が一致しない可能性があります。

1.11.2. アップグレードに関する考慮事項

maistra.io/ ラベルまたはアノテーションは、ユーザーが作成したカスタムリソースで使用することはできません。これは、Red Hat OpenShift Service Mesh Operator によってリソースが生成され、管理される必要があることを示すためです。

警告

アップグレード時に、Operator はファイルの削除や置換などの変更を、リソースが Operator によって管理されることを示す以下のラベルまたはアノテーションを含むリソースに対して行います。

アップグレードする前に、ユーザーが作成したカスタムリソースで以下のラベルまたはアノテーションが含まれるか確認します。

- **maistra-istio-operator** (Red Hat OpenShift Service Mesh) に設定された **maistra.io/** および **app.kubernetes.io/managed-by** ラベル
- **kiali.io/** (Kiali)
- **jaegertracing.io/** (Red Hat OpenShift 分散トレースプラットフォーム)
- **logging.openshift.io/** (Red Hat Elasticsearch)

アップグレードの前に、ユーザーが作成したカスタムリソースで、それらが Operator によって管理されることを示すラベルまたはアノテーションを確認します。Operator によって管理されないカスタムリソースからラベルまたはアノテーションを削除します。

バージョン 2.0 にアップグレードする場合、Operator は SMCP と同じ namespace で、これらのラベルを持つリソースのみを削除します。

バージョン 2.1 にアップグレードする場合、Operator はすべての namespace で、これらのラベルを持つリソースを削除します。

1.11.2.1. アップグレードに影響する可能性のある既知の問題

アップグレードに影響する可能性がある既知の問題には、次のものがあります。

- Red Hat OpenShift Service Mesh は、明示的に文書化されている場合を除き、**EnvoyFilter** 設定の使用はサポートしていません。これは、下層の Envoy API と疎結合されており、後方互換性を保持することができないためです。Envoy フィルターを使用していて、**ServiceMeshControlPlane** のアップグレードによって導入された新しいバージョンの Envoy が原因で Istio によって生成された設定が変更された場合、実装した **EnvoyFilter** が壊れる可能性があります。
- OSSM-1505 **ServiceMeshExtension** は、OpenShift Container Platform バージョン 4.11 では機能しません。**ServiceMeshExtension** は Red Hat OpenShift Service Mesh 2.2 で非推奨となったため、この既知の問題は修正されず、エクステンションを **WasmPluging** に移行する必要があります。
- OSSM-1396 ゲートウェイリソースに **spec.externalIPs** 設定が含まれている場合には、ゲートウェイは、**ServiceMeshControlPlane** の更新時に再作成されずに削除され、再作成されることはありません。
- OSSM-1052 Service Mesh コントロールプレーンで入力ゲートウェイのサービス **ExternalIP** を設定すると、サービスは作成されません。SMCP のスキーマには、サービスのパラメーターがありません。
  回避策: SMCP 仕様でゲートウェイの作成を無効にして、(サービス、ロール、および RoleBinding など) ゲートウェイのデプロイメントを完全に手動で管理します。

1.11.3. Operator のアップグレード

Service Mesh に最新のセキュリティー修正、バグ修正、およびソフトウェア更新プログラムを適用し続けるには、Operator を最新の状態に保つ必要があります。パッチの更新は、Operator をアップグレードすることで開始されます。

重要

Operator のバージョンは、お使いのサービスメッシュのバージョンを 判別しません。デプロイされた Service Mesh コントロールプレーンのバージョンによって、Service Mesh のバージョンが決まります。

Red Hat OpenShift Service Mesh Operator は Service Mesh コントロールプレーンの複数のバージョンをサポートするため、Red Hat OpenShift Service Mesh Operator を更新しても、デプロイされた **ServiceMeshControlPlane** の **spec.version** 値は 更新されません。また、**spec.version** 値は 2 桁の数字 (2.2 など) であり、パッチの更新 (2.2.1 など) は SMCP バージョン値に反映されないことにも注意してください。

Operator Lifecycle Manager (OLM) は、クラスター内の Operator のインストール、アップグレード、ロールベースのアクセス制御 (RBAC) を制御します。OLM はデフォルトで OpenShift Container Platform で実行されます。OLM は利用可能な Operator のクエリーやインストールされた Operator のアップグレードを実行します。

Operator をアップグレードするためにアクションを実行する必要があるかどうかは、インストール時に選択した設定によって異なります。各 Operator をインストールしたときに、**Update Channel** および **Approval Strategy** を選択しました。これら 2 つの設定の組み合わせによって、Operator が更新されるタイミングと方法が決まります。

表1.5 更新チャネルおよび承認ストラテジーのインタラクション

	バージョン付けされたチャネル	stable または Preview チャネル
自動	そのバージョンのみのマイナーリリースおよびパッチリリースの Operator を自動的に更新します。次のメジャーバージョン (バージョン 2.0 から 3.0) には、自動的に自動的に更新されません。次のメジャーバージョンに更新するために必要な Operator サブスクリプションを手動で変更します。	すべてのメジャー、マイナーおよびパッチリリースについて、Operator を自動的に更新します。
手動	指定したバージョンのマイナーおよびパッチリリースに必要な手動更新。次のメジャーバージョンに更新するために必要な Operator サブスクリプションを手動で変更します。	すべてのメジャー、マイナー、およびパッチリリースについて、手動更新が必要になります。

Red Hat OpenShift Service Mesh Operator を更新すると、Operator Lifecycle Manager (OLM) は古い Operator Pod を削除し、新しい Pod を開始します。新しい Operator Pod が開始されると、調整プロセスは **ServiceMeshControlPlane** (SMCP) をチェックし、いずれかの Service Mesh コントロールプレーンコンポーネントの利用可能な更新されたコンテナーイメージがある場合は、それらの Service Mesh コントロールプレーン Pod を新しいコンテナーイメージを使用するものに置き換えます。

Kiali および Red Hat OpenShift 分散トレースプラットフォーム Operator をアップグレードすると、OLM 調整プロセスがクラスターをスキャンし、管理対象インスタンスを新しい Operator のバージョンにアップグレードします。たとえば、Red Hat OpenShift 分散トレースプラットフォーム Operator を

バージョン 1.30.2 からバージョン 1.34.1 に更新する場合、Operator は分散トレースプラットフォームの実行中のインスタンスをスキャンし、それらも 1.34.1 にアップグレードします。

Red Hat OpenShift Service Mesh の特定のパッチバージョンにとどまるには、自動更新を無効にして、Operator のその特定のバージョンにとどまる必要があります。

Operator のアップグレードについての詳細は、Operator Lifecycle Manager のドキュメントを参照してください。

1.11.4. コントロールプレーンのアップグレード

マイナーおよびメジャーリリースのコントロールプレーンを手動で更新する必要があります。コミュニティーの Istio プロジェクトはカナリアアップグレードを推奨していますが、Red Hat OpenShift Service Mesh はインプレースアップグレードのみをサポートしています。Red Hat OpenShift Service Mesh では、各マイナーリリースから次のマイナーリリースに順番にアップグレードする必要があります。たとえば、バージョン 2.0 からバージョン 2.1 にアップグレードしてから、バージョン 2.2 にアップグレードする必要があります。Red Hat OpenShift Service Mesh 2.0 から 2.2 に直接更新することはできません。

サービスメッシュコントロールプレーンをアップグレードすると、Operator が管理するすべてのリソース (ゲートウェイなど) もアップグレードされます。

複数のバージョンのコントロールプレーンを同じクラスターにデプロイできますが、Red Hat OpenShift Service Mesh は、サービスメッシュのカナリアアップグレードをサポートしていません。つまり、**spec.version** の値が異なるさまざまな SCMP リソースを使用できますが、同じメッシュを管理することはできません。

1.11.4.1. バージョン 2.1 から 2.2 へのアップグレードに伴う変更

Service Mesh コントロールプレーンをバージョン 2.1 から 2.2 にアップグレードすると、次の動作変更が導入されます。

- **istio-node** DaemonSet は、アップストリームの Istio の名前と一致するように **istio-cni-node** に名前が変更されました。
- Istio 1.10 は、デフォルトで **lo** ではなく **eth0** を使用してトラフィックをアプリケーションコンテナーに送信するように Envoy を更新しました。
- このリリースでは、**WasmPlugin** API のサポートが追加され、**ServiceMeshExtentionAPI** が廃止されました。

エクステンションの移行に関する詳細は、ServiceMeshExtension から WasmPlugin リソースへの移行を参照してください。

1.11.4.2. バージョン 2.0 から 2.1 へのアップグレードに伴う変更

Service Mesh コントロールプレーンをバージョン 2.0 から 2.1 にアップグレードすると、以下のアーキテクチャーおよび動作上の変更が導入されます。

アーキテクチャーの変更

Mixer は Red Hat OpenShift Service Mesh 2.1 で完全に削除されました。Red Hat OpenShift Service Mesh 2.0.x リリースから 2.1 へのアップグレードは、Mixer が有効な場合にはブロックされます。

v2.0 から v2.1 にアップグレード時に以下のメッセージが表示される場合、**.spec.version** フィールドを更新する前に、既存のコントロールプレーン仕様にすでに存在する **Mixer** タイプを **Istiod** タイプに更新します。

```
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]”
```

以下に例を示します。

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  policy:
    type: Istiod
  telemetry:
    type: Istiod
  version: v2.2
```

動作上の変更

- **AuthorizationPolicy** の更新
  - PROXY プロトコルでは、**ipBlocks** および **notIpBlocks** を使用してリモート IP アドレスを指定する場合は、代わりに **remoteIpBlocks** および **notRemoteIpBlocks** を使用するように設定を更新します。
  - ネストされた JSON Web Token(JWT) 要求のサポートが追加されました。
- **EnvoyFilter** の重大な変更
  - **typed_config** を使用する必要があります。
  - Xds v2 はサポート対象外になりました。
  - フィルター名が非推奨になりました。
- 以前のバージョンのプロキシーは、新しいプロキシーから 1xx または 204 ステータスコードを受信すると、503 ステータスコードを報告する場合があります。

1.11.4.3. Service Mesh コントロールプレーンのアップグレード

Red Hat OpenShift Service Mesh をアップグレードするには、Red Hat OpenShift Service Mesh **ServiceMeshControlPlane** v2 リソースのバージョンフィールドを更新する必要があります。次に、設定と適用が完了したら、アプリケーション Pod を再起動して各サイドカープロキシーとその設定を更新します。

前提条件

- OpenShift Container Platform 4.9 以降を実行している。
- 最新の Red Hat OpenShift Service Mesh Operator がある。

手順

1. **ServiceMeshControlPlane** リソースが含まれるプロジェクトに切り替えます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

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

2. v2 **ServiceMeshControlPlane** リソース設定をチェックし、これが有効であることを確認します。

   a. 以下のコマンドを実行して、**ServiceMeshControlPlane** リソースを v2 リソースとして表示します。

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

ヒント

Service Mesh コントロールプレーン設定をバックアップします。

3. **.spec.version** フィールドを更新し、設定を適用します。
以下に例を示します。

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.2
```

または、コマンドラインの代わりに Web コンソールを使用して Service Mesh コントロールプレーンを編集することもできます。OpenShift Container Platform Web コンソールで、**Project** をクリックし、入力したプロジェクト名を選択します。

   a. **Operators → Installed Operators** をクリックします。

   b. **ServiceMeshControlPlane** インスタンスを見つけます。

   c. **YAML view** を選択し、直前の例のように YAML ファイルのテキストを更新します。

   d. **Save** をクリックします。

1.11.4.4. Red Hat OpenShift Service Mesh のバージョン 1.1 からバージョン 2.0 への移行

バージョン 1.1 から 2.0 にアップグレードするには、ワークロードおよびアプリケーションを新規バージョンを実行する Red Hat OpenShift Service Mesh の新規インスタンスに移行する手動の手順が必要です。

前提条件

- OpenShift Container Platform 4.7 にアップグレードしてから Red Hat OpenShift Service Mesh 2.0 にアップグレードする必要があります。
- Red Hat OpenShift Service Mesh のバージョン 2.0 Operator が必要です。自動 アップグレードパスを選択した場合、Operator は最新情報を自動的にダウンロードします。ただし、Red Hat

OpenShift Service Mesh バージョン 2.0 で機能を使用するために実行する必要のある手順があります。

1.11.4.4.1. Red Hat OpenShift Service Mesh のアップグレード

Red Hat OpenShift Service Mesh をアップグレードするには、新規の namespace に Red Hat OpenShift Service Mesh **ServiceMeshControlPlane** 2.0 リソースのインスタンスを作成する必要があります。次に、設定後にマイクロサービスアプリケーションおよびワークロードを古いメッシュから新規のサービスメッシュに移動します。

手順

1. v1 **ServiceMeshControlPlane** リソース設定をチェックし、これが有効であることを確認します。

   a. 以下のコマンドを実行して、**ServiceMeshControlPlane** リソースを v2 リソースとして表示します。

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

   b. 無効なフィールドについての情報は、出力の **spec.techPreview.errored.message** フィールドを確認してください。

   c. v1 リソースに無効なフィールドがある場合、リソースは調整されず、v2 リソースとして編集することはできません。v2 フィールドへの更新はすべて、元の v1 設定で上書きされます。無効なフィールドを修正するには、リソースの v1 バージョンを置き換えるか、パッチを適用するか、または編集できます。リソースを修正せずに削除することもできます。リソースの修正後に調整でき、リソースの v2 バージョンを変更または表示できます。

   d. ファイルを編集してリソースを修正するには、**oc get** を使用してリソースを取得し、テキストファイルをローカルで編集し、リソースを編集したファイルに置き換えます。

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

   e. パッチを使用してリソースを修正するには、**oc patch** を使用します。

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

   f. コマンドラインツールで編集してリソースを修正するには、**oc edit** を使用します。

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

2. Service Mesh コントロールプレーン設定をバックアップします。**ServiceMeshControlPlane** リソースが含まれるプロジェクトに切り替えます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

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

3. 以下のコマンドを実行して、現在の設定を取得します。<smcp_name> は **ServiceMeshControlPlane** リソースのメタデータに指定されます (例: **basic-install** または **full-install**)。

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

4. **ServiceMeshControlPlane** を、開始点としての設定についての情報が含まれる v2 のコントロールプレーンバージョンに変換します。

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

5. プロジェクトを作成します。OpenShift Container Platform コンソールの Project メニューで、**New Project** をクリックし、プロジェクトの名前 (例: **istio-system-upgrade**) を入力します。または、CLI からこのコマンドを実行できます。

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

6. v2 **ServiceMeshControlPlane** の **metadata.namespace** フィールドを新規のプロジェクト名で更新します。この例では、**istio-system-upgrade** を使用します。

7. **version** フィールドを 1.1 から 2.0 に更新するか、または v2 **ServiceMeshControlPlane** でこれを削除します。

8. 新規 namespace に **ServiceMeshControlPlane** を作成します。コマンドラインで以下のコマンドを実行し、取得した **ServiceMeshControlPlane** の v2 バージョンでコントロールプレーンをデプロイします。この例では、`<smcp_name.v2>` をファイルへのパスに置き換えます。

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

   または、コンソールを使用して Service Mesh コントロールプレーンを作成することもできます。OpenShift Container Platform Web コンソールで、 **Project** をクリックします。次に、入力したプロジェクト名を選択します。

   a. **Operators → Installed Operators** をクリックします。

   b. **Create ServiceMeshControlPlane** をクリックします。

   c. **YAML view** を選択し、取得した YAML ファイルのテキストをフィールドに貼り付けます。**apiVersion** フィールドが **maistra.io/v2** に設定されていることを確認し、**metadata.namespace** フィールドを新規 namespace (例: **istio-system-upgrade**) を使用するように変更します。

   d. **Create** をクリックします。

1.11.4.4.2. 2.0 ServiceMeshControlPlane の設定

ServiceMeshControlPlane リソースは Red Hat OpenShift Service Mesh バージョン 2.0 用に変更されました。**ServiceMeshControlPlane** リソースの v2 バージョンを作成したら、新機能を利用し、デプロイメントを適合させるようにこれを変更します。**ServiceMeshControlPlane** リソースを変更するため、Red Hat OpenShift Service Mesh 2.0 の仕様および動作に以下の変更を加えることを検討してください。使用する機能の更新情報については、Red Hat OpenShift Service Mesh 2.0 の製品ドキュメントを参照してください。v2 リソースは、Red Hat OpenShift Service Mesh 2.0 インストールに使用する必要があります。

1.11.4.4.2.1. アーキテクチャーの変更

以前のバージョンで使用されるアーキテクチャーのユニットは Istiod によって置き換えられました。2.0 では、Service Mesh コントロールプレーンのコンポーネント Mixer、Pilot、Citadel、Galley、およびサイドカーインジェクター機能が単一のコンポーネントである Istiod に統合されました。

Mixer はコントロールプレーンコンポーネントとしてサポートされなくなりましたが、Mixer ポリシーおよび Telemetry プラグインは Istiod の WASM 拡張でサポートされるようになりました。レガシー Mixer プラグインを統合する必要がある場合は、ポリシーと Telemetry に対して Mixer を有効にすることができます。

シークレット検出サービス (SDS) は、証明書とキーを Istiod からサイドカーコンテナーに直接配信するために使用されます。Red Hat OpenShift Service Mesh バージョン 1.1 では、シークレットはクライアント証明書およびキーを取得するためにプロキシーによって使用される Citadel で生成されました。

1.11.4.4.2.2. アノテーションの変更

v2.0 では、以下のアノテーションに対応しなくなりました。これらのアノテーションのいずれかを使用している場合は、これを v2.0 Service Mesh コントロールプレーンに移行する前にワークロードを更新する必要があります。

- **sidecar.maistra.io/proxyCPULimit** は **sidecar.istio.io/proxyCPULimit** に置き換えられました。ワークロードで **sidecar.maistra.io** アノテーションを使用していた場合は、代わりに **sidecar.istio.io** を使用するようにこれらのワークロードを変更する必要があります。
- **sidecar.maistra.io/proxyMemoryLimit** が **sidecar.istio.io/proxyMemoryLimit** に置き換えられました。
- **sidecar.istio.io/discoveryAddress** はサポートされなくなりました。また、デフォルトの検出アドレスが **pilot.<control_plane_namespace>.svc:15010** (または mtls が有効にされている場合はポート 15011) から **istiod-<smcp_name>.<control_plane_namespace>.svc:15012** に移行しました。
- ヘルスステータスポートは設定できなくなり、15021 にハードコーディングされています。* カスタムステータスポート (例: **status.sidecar.istio.io/port**) を定義している場合、ワークロードを v2.0 Service Mesh コントロールプレーンに移行する前にオーバーライドを削除する必要があります。ステータスポートを **0** に設定すると、readiness チェックを依然として無効にできます。
- Kubernetes シークレットリソースは、サイドカーのクライアント証明書を配信するために使用されなくなりました。証明書は Istiod の SDS サービスを介して配信されるようになりました。マウントされたシークレットに依存している場合、それらは v2.0 Service Mesh コントロールプレーンのワークロードで利用不可になります。

1.11.4.4.2.3. 動作上の変更

Red Hat OpenShift Service Mesh 2.0 の機能の一部は、以前のバージョンの機能とは異なります。

- ゲートウェイの readiness ポートは **15020** から **15021** に移行しました。
- ターゲットホストの可視性には、VirtualService と ServiceEntry リソースが含まれます。これには、Sidecar リソースを介して適用される制限が含まれます。
- 自動の相互 TLS はデフォルトで有効になります。プロキシー間の通信は、実施されているグローバルの PeerAuthentication ポリシーに関係なく、mTLS を使用するように自動的に設定されます。
- セキュアな接続は、**spec.security.controlPlane.mtls** 設定に関係なく、プロキシーが Service Mesh コントロールプレーンと通信する際に常に使用されま

す。**spec.security.controlPlane.mtls** 設定は、Mixer Telemetry またはポリシーの接続を設定する場合にのみ使用されます。

1.11.4.4.2.4. サポート対象外のリソースの移行情報

Policy (authentication.istio.io/v1alpha1)

Policy リソースは、v2.0 Service Mesh コントロールプレーン、PeerAuthentication および RequestAuthentication で使用するために新規リソースタイプに移行する必要があります。Policy リソースの特定の設定によっては、同じ効果を実現するために複数のリソースを設定しなければならない場合があります。

相互 TLS

相互 TLS は、**security.istio.io/v1beta1** PeerAuthentication リソースを使用して実行されます。レガシーの **spec.peers.mtls.mode** フィールドは、新規リソースの **spec.mtls.mode** フィールドに直接マップされます。選定基準は、**spec.targets[x].name** のでのサービス名の指定から **spec.selector.matchLabels** のラベルセレクターに変更されました。PeerAuthentication では、ラベルは、ターゲット一覧で名前が指定されたサービスのセレクターと一致する必要があります。ポート固有の設定は **spec.portLevelMtls** にマップされる必要があります。

認証

spec.origins に指定される追加の認証方法は、**security.istio.io/v1beta1** RequestAuthentication リソースにマップされる必要があります。**spec.selector.matchLabels** は PeerAuthentication の同じフィールドに対して同様に設定される必要があります。**spec.origins.jwt** アイテムからの JWT プリンシパルに固有の設定は、**spec.rules** アイテムの同様のフィールドにマップされます。

- Policy で指定される **spec.origins[x].jwt.triggerRules** は 1 つ以上の **security.istio.io/v1beta1** AuthorizationPolicy リソースにマップされる必要があります。**spec.selector.labels** は、RequestAuthentication の同じフィールドに対して同様に設定される必要があります。

- **spec.origins[x].jwt.triggerRules.excludedPaths** は AuthorizationPolicy にマップされる必要があります。ここで、**spec.rules[x].to.operation.path** エントリーは除外されたパスに一致する状態で spec.action が ALLOW に設定されます。

- **spec.origins[x].jwt.triggerRules.includedPaths** は別個の AuthorizationPolicy にマップされる必要があります。ここで、**spec.rules[x].to.operation.path** エントリーは組み込まれるパスに一致し、**specified spec.origins[x].jwt.issuer** と一致する **spec.rules.[x].from.source.requestPrincipals** エントリーが Policy リソースにある状態で、 **spec.action** が **ALLOW** に設定されます。

ServiceMeshPolicy (maistra.io/v1)

ServiceMeshPolicy は、v1 リソースの **spec.istio.global.mtls.enabled** または v2 リソース設定の **spec.security.dataPlane.mtls** で Service Mesh コントロールプレーンに自動的に設定されています。v2 コントロールプレーンの場合、機能的に同等の PeerAuthentication リソースがインストール時に作成されます。この機能は、Red Hat OpenShift Service Mesh バージョン 2.0 で非推奨となりました。

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

これらのリソースは **security.istio.io/v1beta1** AuthorizationPolicy リソースに置き換えられました。

RbacConfig の動作をコピーするには、RbacConfig で指定される spec.mode に依存するデフォルトの AuthorizationPolicy を作成する必要があります。

- **spec.mode** が **OFF** に設定されている場合、AuthorizationPolicy が要求に適用されない限り、デフォルトのポリシーが ALLOW であるためリソースは必要ありません。

- **spec.mode** が ON に設定されている場合、 **spec: {}** を設定します。メッシュ内のすべてのサービスに対して AuthorizationPolicy ポリシーを作成する必要があります。
- **spec.mode** は **ON_WITH_INCLUSION** に設定されている場合、 **spec: {}** が組み込まれている各 namespace に指定された状態で AuthorizationPolicy を作成する必要があります。AuthorizationPolicy では、個別のサービスを含めることはサポートされません。ただし、サービスのワークロードに適用される AuthorizationPolicy が作成されるとすぐに、明示的に許可されない他のすべての要求は拒否されます。
- **spec.mode** が **ON_WITH_EXCLUSION** に設定されている場合、これは AuthorizationPolicy によってサポートされません。グローバル DENY ポリシーを作成できますが、namespace またはワークロードのいずれかに適用できる allow-all ポリシーがないため、メッシュ内のすべてのワークロードに対して AuthorizationPolicy を作成する必要があります。

AuthorizationPolicy には、ServiceRoleBinding が提供する機能と同様の、設定が適用されるセレクターと、ServiceRole が提供する機能と同様の、適用する必要のあるルールの両方の設定が含まれます。

ServiceMeshRbacConfig (maistra.io/v1)

このリソースは、Service Mesh コントロールプレーンの namespace で **security.istio.io/v1beta1** AuthorizationPolicy リソースを使用して置き換えられます。このポリシーは、メッシュ内のすべてのワークロードに適用されるデフォルトの認証ポリシーになります。特定の移行の詳細については、上記の RbacConfig を参照してください。

1.11.4.4.2.5. Mixer プラグイン

Mixer コンポーネントは、バージョン 2.0 ではデフォルトで無効にされます。ワークロードで Mixer プラグインを使用する場合は、Mixer コンポーネントを含めるようにバージョン 2.0 **ServiceMeshControlPlane** を設定する必要があります。

Mixer ポリシーコンポーネントを有効にするには、以下のスニペットを **ServiceMeshControlPlane** に追加します。

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

Mixer Telemetry コンポーネントを有効にするには、以下のスニペットを **ServiceMeshControlPlane** に追加します。

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

レガシーの Mixer プラグインは、新しい ServiceMeshExtension (maistra.io/v1alpha1) カスタムリソースを使用して WASM に移行し、統合することもできます。

アップストリーム Istio ディストリビューションに含まれるビルトインの WASM フィルターは Red Hat OpenShift Service Mesh 2.0 では利用できません。

1.11.4.4.2.6. 相互 TLS の変更

ワークロード固有の PeerAuthentication ポリシーで mTLS を使用する場合、ワークロードポリシーが namespace/グローバルポリシーと異なる場合にトラフィックを許可するために、対応する DestinationRule が必要になります。

自動 mTLS はデフォルトで有効にされますが、**spec.security.dataPlane.automtls** を **ServiceMeshControlPlane** リソースで false に設定して無効にできます。auto mTLS を無効にする場合、サービス間の適切な通信のために DestinationRules が必要になる場合があります。たとえば、1 つの namespace について PeerAuthentication を **STRICT** に設定すると、DestinationRule が namespace のサービスに TLS モードを設定しない限り、他の namespace のサービスがそれらにアクセスできなくなります。

mTLS について詳細は、相互トランスポート層セキュリティー (mTLS) の有効化 を参照してください。

1.11.4.4.2.6.1. その他の mTLS の例

bookinfo サンプルアプリケーションで mTLS For productpage サービスを無効にするために、Policy リソースは Red Hat OpenShift Service Mesh v1.1 に対して以下の方法で設定されました。

Policy リソースの例

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

bookinfo サンプルアプリケーションで mTLS For productpage サービスを無効にするために、以下の例を使用して Red Hat OpenShift Service Mesh v2.0 の PeerAuthentication リソースを設定します。

PeerAuthentication リソースの例

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

bookinfo サンプルアプリケーションで **productpage** サービスについて mTLS With JWT 認証を有効にするために、Policy リソースが Red Hat OpenShift Service Mesh v1.1 に対して設定されました。

Policy リソースの例

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

```
  - name: productpage
    ports:
    - number: 9000
  peers:
  - mtls:
  origins:
  - jwt:
      issuer: "https://securetoken.google.com"
      audiences:
      - "productpage"
      jwksUri: "https://www.googleapis.com/oauth2/v1/certs"
      jwtHeaders:
      - "x-goog-iap-jwt-assertion"
      triggerRules:
      - excludedPaths:
        - exact: /health_check
  principalBinding: USE_ORIGIN
```

bookinfo サンプルアプリケーションの productpage サービスの mTLS With JWT 認証を有効にするために、以下の例を使用して Red Hat OpenShift Service Mesh v2.0 の PeerAuthentication リソースを設定します。

PeerAuthentication リソースの例

```
#require mtls for productpage:9000
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: productpage-mTLS-with-JWT
  namespace: <namespace>
spec:
  selector:
    matchLabels:
      # this should match the selector for the "productpage" service
      app: productpage
  portLevelMtls:
    9000:
      mode: STRICT
---
#JWT authentication for productpage
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: productpage-mTLS-with-JWT
  namespace: <namespace>
spec:
  selector:
    matchLabels:
      # this should match the selector for the "productpage" service
      app: productpage
  jwtRules:
  - issuer: "https://securetoken.google.com"
    audiences:
    - "productpage"
    jwksUri: "https://www.googleapis.com/oauth2/v1/certs"
    fromHeaders:
```

```
    - name: "x-goog-iap-jwt-assertion"
---
#Require JWT token to access product page service from
#any client to all paths except /health_check
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: productpage-mTLS-with-JWT
  namespace: <namespace>
spec:
  action: ALLOW
  selector:
    matchLabels:
      # this should match the selector for the "productpage" service
      app: productpage
  rules:
  - to: # require JWT token to access all other paths
      - operation:
          notPaths:
          - /health_check
    from:
      - source:
          # if using principalBinding: USE_PEER in the Policy,
          # then use principals, e.g.
          # principals:
          # - “*”
          requestPrincipals:
          - “*”
  - to: # no JWT token required to access health_check
      - operation:
          paths:
          - /health_check
```

1.11.4.4.3. 設定レシピ

これらの設定レシピを使用して、以下の項目を設定することができます。

1.11.4.4.3.1. データプレーンでの相互 TLS

データプレーン通信の相互 TLS は、**ServiceMeshControlPlane** リソースの **spec.security.dataPlane.mtls** で設定されます。これはデフォルトは **false** になります。

1.11.4.4.3.2. カスタム署名キー

Istiod は、サービスプロキシーによって使用されるクライアント証明書とプライベートキーを管理します。デフォルトで、Istiod は署名用に自己署名証明書を使用しますが、カスタム証明書とシークレットキーを設定できます。署名するキーの設定方法についての詳細は、 外部認証局キーおよび証明書の追加を参照してください。

1.11.4.4.3.3. トレーシング

トレースは **spec.tracing** で設定されます。現在、サポートされるトレーサーの唯一のタイプは **Jaeger** です。サンプリングは 0.01% の増分 (例: 1 は 0.01%、10000 は 100%) を表すスケーリングされた整数です。トレースの実装およびサンプリングレートを指定できます。

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

Jaeger は、**ServiceMeshControlPlane** リソースの **addons** セクションで設定します。

```
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"
```

Jaeger インストールは **install** フィールドでカスタマイズできます。リソース制限などのコンテナー設定は、**spec.runtime.components.jaeger** の関連フィールドに設定されます。**spec.addons.jaeger.name** の値に一致する Jaeger リソースが存在する場合、Service Mesh コントロールプレーンは既存のインストールを使用するように設定されます。既存の Jaeger リソースを使用して Jaeger インストールを完全にカスタマイズします。

1.11.4.4.3.4. 可視化

Kiali および Grafana は、**ServiceMeshControlPlane** リソースの **addons** セクションで設定されます。

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

Grafana および Kiali のインストールは、それぞれの **install** フィールドでカスタマイズできます。リソース制限などのコンテナーのカスタマイズは、 **spec.runtime.components.kiali** および **spec.runtime.components.grafana** で設定されます。name の値に一致する既存の Kiali リソースが存在する場合、Service Mesh コントロールプレーンはコントロールプレーンで使用するように Kiali リソースを設定します。**accessible_namespaces** 一覧や Grafana、Prometheus、およびトレースのエンドポイントなどの Kiali リソースの一部のフィールドは上書きされます。既存のリソースを使用して Kiali インストールを完全にカスタマイズします。

1.11.4.4.3.5. リソース使用状況とスケジューリング

リソースは **spec.runtime.<component>** で設定されます。以下のコンポーネント名がサポートされます。

コンポーネント	説明	サポート対象バージョン
セキュリティー	Citadel コンテナー	v1.0/1.1
galley	Galley コンテナー	v1.0/1.1
pilot	Pilot/Istiod コンテナー	v1.0/1.1/2.0
mixer	istio-telemetry および istio-policy コンテナー	v1.0/1.1
mixer.policy	istio-policy コンテナー	v2.0
mixer.telemetry	istio-telemetry コンテナー	v2.0
global.ouathproxy	各種アドオンと共に使用する oauth-proxy コンテナー	v1.0/1.1/2.0
sidecarInjectorWebhook	サイドカーインジェクター Webhook コンテナー	v1.0/1.1
tracing.jaeger	一般的な Jaeger コンテナー: すべての設定が適用されない可能性があります。Jaeger インストールの完全なカスタマイズは、既存の Jaeger リソースを Service Mesh コントロールプレーンの設定に指定することでサポートされます。	v1.0/1.1/2.0
tracing.jaeger.agent	Jaeger エージェントに固有の設定	v1.0/1.1/2.0
tracing.jaeger.allInOne	Jaeger allInOne に固有の設定	v1.0/1.1/2.0
tracing.jaeger.collector	Jaeger コレクターに固有の設定	v1.0/1.1/2.0

コンポーネント	説明	サポート対象バージョン
tracing.jaeger.elasticsearch	Jaeger elasticsearch デプロイメントに固有の設定	v1.0/1.1/2.0
tracing.jaeger.query	Jaeger クエリーに固有の設定	v1.0/1.1/2.0
prometheus	prometheus コンテナー	v1.0/1.1/2.0
kiali	Kiali コンテナー: Kiali インストールの完全なカスタマイズは、既存の Kiali リソースを Service Mesh コントロールプレーン設定に指定してサポートされます。	v1.0/1.1/2.0
grafana	Grafana コンテナー	v1.0/1.1/2.0
3scale	3scale コンテナー	v1.0/1.1/2.0
wasmExtensions.cacher	WASM 拡張キャッシュコンテナー	V2.0: テクノロジープレビュー

コンポーネントによっては、リソースの制限およびスケジューリングをサポートするものもあります。詳細は、パフォーマンスおよびスケーラビリティー を参照してください。

1.11.4.4.4. アプリケーションとワークロードを移行するための次の手順

アプリケーションのワークロードを新規のメッシュに移動し、古いインスタンスを削除してアップグレードを完了します。

1.11.5. データプレーンのアップグレード

コントロールプレーンをアップグレードした後も、データプレーンは引き続き機能します。ただし、Envoy プロキシーに更新を適用し、プロキシー設定に変更を適用するには、アプリケーション Pod とワークロードを再起動する必要があります。

1.11.5.1. アプリケーションおよびワークロードの更新

移行を完了するには、メッシュ内のすべてのアプリケーション Pod を再起動して、Envoy サイドカープロキシーおよびそれらの設定をアップグレードします。

デプロイメントのローリング更新を実行するには、以下のコマンドを使用します。

```
$ oc rollout restart <deployment>
```

メッシュを設定するすべてのアプリケーションに対してローリング更新を実行する必要があります。

1.12. ユーザーおよびプロファイルの管理

1.12.1. Red Hat OpenShift Service Mesh メンバーの作成

ServiceMeshMember リソースは、各ユーザーがサービスメッシュプロジェクトまたはメンバーロールに直接アクセスできない場合でも、Red Hat OpenShift Service Mesh の管理者がプロジェクトをサービスメッシュに追加するパーミッションを委譲する方法を提供します。プロジェクト管理者にはプロジェクトで **ServiceMeshMember** リソースを作成するためのパーミッションが自動的に付与されますが、サービスメッシュ管理者がサービスメッシュへのアクセスを明示的に付与するまで、これらのプロジェクト管理者はこれを **ServiceMeshControlPlane** にポイントすることはできません。管理者は、ユーザーに **mesh-user** ユーザーロールを付与してメッシュにアクセスするパーミッションをユーザーに付与できます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

```
$ oc policy add-role-to-user -n istio-system --role-namespace istio-system mesh-user <user_name>
```

管理者は Service Mesh コントロールプレーンプロジェクトで **mesh user** ロールバインディングを変更し、アクセスが付与されたユーザーおよびグループを指定できます。**ServiceMeshMember** は、プロジェクトを、参照する Service Mesh コントロールプレーンプロジェクト内の **ServiceMeshMemberRoll** に追加します。

```
apiVersion: maistra.io/v1
kind: ServiceMeshMember
metadata:
  name: default
spec:
  controlPlaneRef:
    namespace: istio-system
    name: basic
```

mesh-users ロールバインディングは、管理者が **ServiceMeshControlPlane** リソースを作成した後に自動的に作成されます。管理者は以下のコマンドを使用してロールをユーザーに追加できます。

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

管理者は、**ServiceMeshControlPlane** リソースを作成する前に、**mesh-user** ロールバインディングを作成することもできます。たとえば、管理者は **ServiceMeshControlPlane** リソースと同じ **oc apply** 操作でこれを作成できます。

この例では、**alice** のロールバインディングを追加します。

```
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. Service Mesh コントロールプレーンプロファイルの作成

ServiceMeshControlPlane プロファイルを使用すると、再利用可能な設定を作成することができま

す。各ユーザーは、作成するプロファイルを独自の設定で拡張することができます。プロファイルは、他のプロファイルから設定情報を継承することもできます。たとえば、会計チーム用の会計コントロールプレーンとマーケティングチーム用のマーケティングコントロールプレーンを作成できます。開発プロファイルと実稼働テンプレートを作成する場合、マーケティングチームおよび会計チームのメンバーは、チーム固有のカスタマイズで開発および実稼働プロファイルを拡張することができます。

ServiceMeshControlPlane と同じ構文に従う Service Mesh コントロールプレーンのプロファイルを設定する場合、ユーザーは階層的に設定を継承します。Operator は、Red Hat OpenShift Service Mesh のデフォルト設定を使用する **default** プロファイルと共に提供されます。

1.12.2.1. ConfigMap の作成

カスタムプロファイルを追加するには、**openshift-operators** プロジェクトで **smcp-templates** という名前の **ConfigMap** を作成する必要があります。Operator コンテナーは **ConfigMap** を自動的にマウントします。

前提条件

- Service Mesh Operator がインストールされ、検証されていること。
- **cluster-admin** ロールを持つアカウントがある。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
- Operator デプロイメントの場所。
- **oc** として知られる OpenShift Container Platform コマンドラインインターフェイス (CLI) へのアクセス。

手順

1. **cluster-admin** として OpenShift Container Platform CLI にログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
2. CLI で以下のコマンドを実行し、 **openshift-operators** プロジェクトに **smcp-templates** という名前の ConfigMap を作成し、**<profiles-directory>** をローカルディスクの **ServiceMeshControlPlane** ファイルの場所に置き換えます。

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

3. **ServiceMeshControlPlane** で **template** パラメーターを使用して 1 つ以上のテンプレートを指定できます。

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

1.12.2.2. 適切なネットワークポリシーの設定

Service Mesh は Service Mesh コントロールプレーンおよびメンバー namespace にネットワークポリシーを作成し、それらの間のトラフィックを許可します。デプロイする前に、以下の条件を考慮し、OpenShift Container Platform ルートで以前に公開されたサービスメッシュのサービスを確認します。

- Istio が適切に機能するには、サービスメッシュへのトラフィックが常に ingress-gateway を経由する必要があります。
- サービスメッシュ外のサービスは、サービスメッシュにない個別の namespace にデプロイします。
- サービスメッシュでリストされた namespace 内にデプロイする必要のあるメッシュ以外のサービスでは、それらのデプロイメント **maistra.io/expose-route: "true"** にラベルを付けます。これにより、これらのサービスへの OpenShift Container Platform ルートは依然として機能します。

1.13. セキュリティー

サービスメッシュアプリケーションが複雑な配列のマイクロサービスで構築されている場合、Red Hat OpenShift Service Mesh を使用してそれらのサービス間の通信のセキュリティーをカスタマイズできます。Service Mesh のトラフィック管理機能と共に OpenShift Container Platform のインフラストラクチャーを使用すると、アプリケーションの複雑性を管理し、マイクロサービスのセキュリティーを確保できるようにします。

作業を開始する前に

プロジェクトがある場合は、プロジェクトを **ServiceMeshMemberRoll** リソース に追加します。

プロジェクトがない場合は、Bookinfo サンプルアプリケーション をインストールし、これを **ServiceMeshMemberRoll** リソースに追加します。サンプルアプリケーションは、セキュリティーの概念を説明するのに役立ちます。

1.13.1. Mutual Transport Layer Security (mTLS) について

Mutual Transport Layer Security (mTLS) は、二者が相互認証できるようにするプロトコルです。これは、一部のプロトコル (IKE、SSH) での認証のデフォルトモードであり、他のプロトコル (TLS) ではオプションになります。mTLS は、アプリケーションやサービスコードを変更せずに使用できます。TLS は、サービスメッシュインフラストラクチャーおよび 2 つのサイドカープロキシー間で完全に処理されます。

デフォルトで、Red Hat OpenShift Service Mesh の mTLS は有効にされ、Permissive モードに設定されます。この場合、Service Mesh のサイドカーは、プレーンテキストのトラフィックと mTLS を使用して暗号化される接続の両方を受け入れます。メッシュのサービスがメッシュ外のサービスと通信している場合、厳密な mTLS によりサービス間の通信に障害が発生する可能性があります。ワークロードを Service Mesh に移行する間に Permissive モードを使用します。次に、メッシュ、namespace、またはアプリケーションで厳密な mTLS を有効にできます。

Service Mesh コントロールプレーンのレベルでメッシュ全体で mTLS を有効にすると、アプリケーションとワークロードを書き換えずにサービスメッシュ内のすべてのトラフィックのセキュリティーが保護されます。メッシュの namespace のセキュリティーは、**ServiceMeshControlPlane** リソースのデータプレーンレベルで保護できます。トラフィックの暗号化接続をカスタマイズするには、アプリケーションレベルで namespace を **PeerAuthentication** および **DestinationRule** リソースで設定します。

1.13.1.1. サービスメッシュ全体での厳密な mTLS の有効化

ワークロードがメッシュ外のサービスと通信しない場合には、通信を中断せずに mTLS をメッシュ全体ですぐに有効化できます。これを有効にするには、**ServiceMeshControlPlane** リソースで **spec.security.dataPlane.mtls** を **true** に設定します。Operator は必要なリソースを作成します。

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  version: v2.2
  security:
    dataPlane:
      mtls: true
```

OpenShift Container Platform Web コンソールを使用して mTLS を有効にすることもできます。

手順

1. Web コンソールにログインします。
2. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
3. **Operators → Installed Operators** をクリックします。
4. **Provided APIs** の **Service Mesh Control Plane** をクリックします。
5. **ServiceMeshControlPlane** リソースの名前 (例: **basic**) をクリックします。
6. **Details** ページで、**Data Plane Security** の **Security** セクションでトグルをクリックします。

1.13.1.1.1. 特定のサービスの受信接続用のサイドカーの設定

ポリシーを作成して、個別のサービスに mTLS を設定することもできます。

手順

1. 以下のサンプルを使用して YAML ファイルを作成します。

   PeerAuthentication ポリシーの例 (policy.yaml)

   ```
   apiVersion: security.istio.io/v1beta1
   kind: PeerAuthentication
   metadata:
     name: default
     namespace: <namespace>
   spec:
     mtls:
       mode: STRICT
   ```

   a. **<namespace>** は、サービスが置かれている namespace に置き換えます。

2. 以下のコマンドを実行して、サービスが置かれている namespace にリソースを作成します。先ほど作成した Policy リソースの **namespace** フィールドと一致させる必要があります。

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

注記

自動 mTLS を使用しておらず、**PeerAuthentication** を STRICT に設定する場合は、サービスの **DestinationRule** リソースを作成する必要があります。

1.13.1.1.2. 送信接続用のサイドカーの設定

宛先ルールを作成し、Service Mesh がメッシュ内の他のサービスに要求を送信する際に mTLS を使用するように設定します。

手順

1. 以下のサンプルを使用して YAML ファイルを作成します。

   destinationRule の例 (destination-rule.yaml)

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

   a. **<namespace>** は、サービスが置かれている namespace に置き換えます。

2. 以下のコマンドを実行して、サービスが置かれている namespace にリソースを作成します。先ほど作成した **DestinationRule** リソースの **namespace** フィールドと一致させる必要があります。

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

1.13.1.1.3. 最小および最大のプロトコルバージョンの設定

ご使用の環境のサービスメッシュに暗号化されたトラフィックの特定の要件がある場合、許可される暗号化機能を制御できます。これは、**ServiceMeshControlPlane** リソースに **spec.security.controlPlane.tls.minProtocolVersion** または **spec.security.controlPlane.tls.maxProtocolVersion** を設定して許可できます。Service Mesh コントロールプレーンリソースで設定されるこれらの値は、TLS 経由でセキュアに通信する場合にメッシュコンポーネントによって使用される最小および最大の TLS バージョンを定義します。

デフォルトは **TLS_AUTO** であり、TLS のバージョンは指定しません。

表1.6 有効な値

値	説明
TLS_AUTO	default
TLSv1_0	TLS バージョン 1.0

値	説明
TLSv1_1	TLS バージョン 1.1
TLSv1_2	TLS バージョン 1.2
TLSv1_3	TLS バージョン 1.3

手順

1. Web コンソールにログインします。
2. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
3. **Operators → Installed Operators** をクリックします。
4. **Provided APIs** の **Service Mesh Control Plane** をクリックします。
5. **ServiceMeshControlPlane** リソースの名前 (例: **basic**) をクリックします。
6. **YAML** タブをクリックします。
7. 以下のコードスニペットを YAML エディターに挿入します。**minProtocolVersion** の値は、TLS バージョンの値に置き換えます。この例では、最小の TLS バージョンは **TLSv1_2** に設定されます。

   ServiceMeshControlPlane スニペット

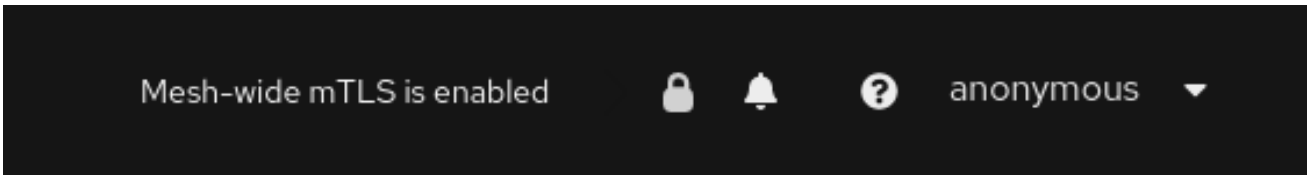
   ```
   kind: ServiceMeshControlPlane
   spec:
     security:
       controlPlane:
         tls:
           minProtocolVersion: TLSv1_2
   ```

8. **Save** をクリックします。
9. **Refresh** をクリックし、変更が正しく更新されたことを確認します。

1.13.1.2. Kiali による暗号化の検証

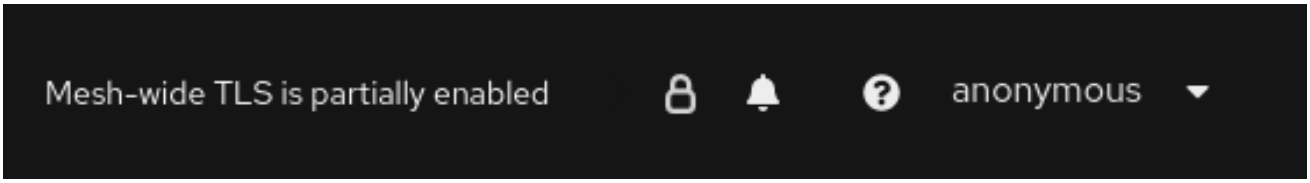
Kiali コンソールは、アプリケーション、サービス、ワークロードが mTLS 暗号化を有効にしているかどうかを検証するためのいくつかの方法を提供します。

図1.5 マストヘッドアイコン メッシュワイド **mTLS** が有効



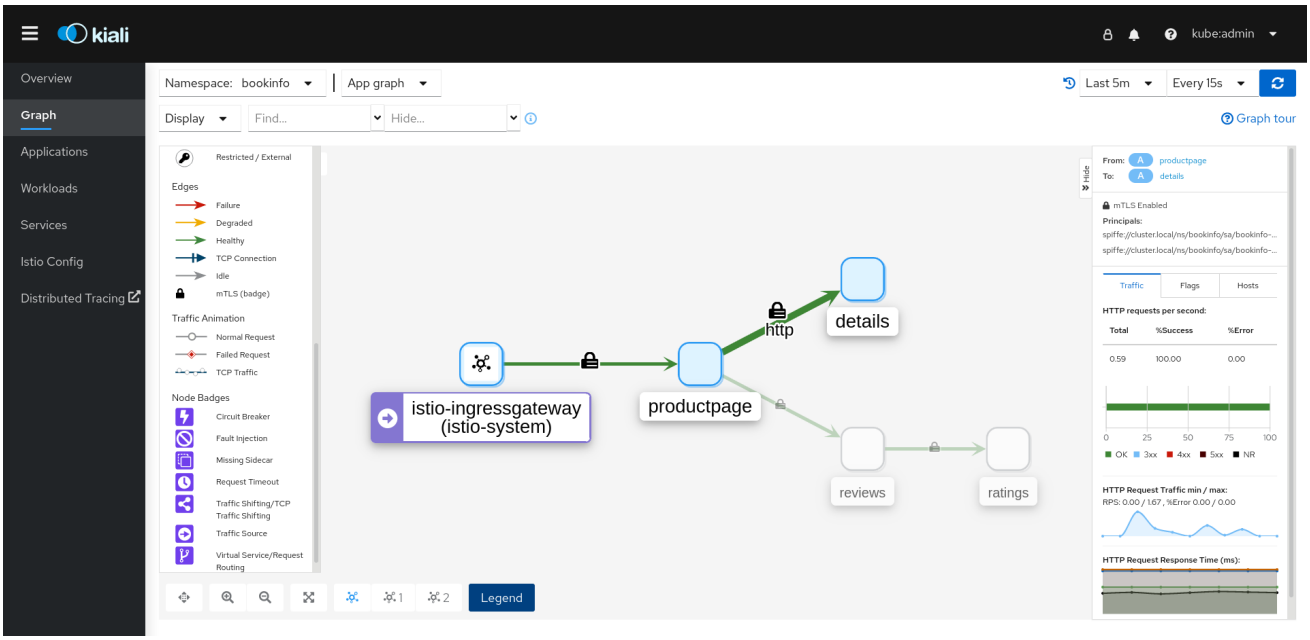
サービスメッシュ全体で厳密に mTLS が有効化されている場合、Kiali はマストヘッドの右側にロックアイコンを表示します。これは、メッシュ内のすべての通信に mTLS が使用されていることを意味します。

図1.6 マストヘッドアイコン メッシュワイド **mTLS** が一部有効



メッシュが **PERMISSIVE** モードに設定されているか、メッシュ全体の mTLS 設定にエラーが発生している場合、Kiali は中空ロックアイコンを表示します。

図1.7 セキュリティーバッジ



Graph ページには、mTLS が有効であることを示すために、グラフの端に **Security** バッジを表示するオプションがあります。グラフにセキュリティーバッジを表示するには、**Display** メニューの **Show Badges** で **Security** チェックボックスをオンにします。エッジにロックアイコンが表示されている場合、mTLS が有効なリクエストが少なくとも 1 つ存在することを意味します。mTLS と non-TLS の両方のリクエストがある場合、サイドパネルには mTLS を使用するリクエストのパーセンテージが表示されます。

Applications Detail Overview ページでは、mTLS が有効なリクエストが 1 つ以上存在するグラフの端に **Security** アイコンが表示されます。

Workloads Detail Overview ページでは、mTLS が有効なリクエストが 1 つ以上存在するグラフの端に **Security** アイコンが表示されます。

Services Detail Overview ページでは、mTLS が有効なリクエストが 1 つ以上存在するグラフの端に **Security** アイコンが表示されます。また、Kiali では、mTLS を設定したポートの横の **Network** セクションにロックアイコンが表示されることに注意してください。

1.13.2. ロールベースアクセス制御 (RBAC) の設定

Role-based Access Control (RBAC: ロールベースアクセス制御) オブジェクトは、ユーザーまたはサービスがプロジェクト内で所定のアクションを実行することが許可されるかどうかを決定します。メッ

シュでワークロードのメッシュ全体、namespace 全体、およびワークロード全体のアクセス制御を定義できます。

RBAC を設定するには、アクセスを設定する namespace で **AuthorizationPolicy** リソースを作成します。メッシュ全体のアクセスを設定する場合は、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を使用します。

たとえば、RBAC を使用して以下を実行するポリシーを作成できます。

- プロジェクト内通信を設定します。
- デフォルト namespace のすべてのワークロードへの完全アクセスを許可または拒否します。
- ingress ゲートウェイアクセスを許可または拒否します。
- アクセスにはトークンが必要です。

認証ポリシーには、セレクター、アクション、およびルールの一覧が含まれます。

- **selector** フィールドは、ポリシーのターゲットを指定します。
- **action** フィールドは、要求を許可または拒否するかどうかを指定します。
- **rules** フィールドは、アクションをトリガーするタイミングを指定します。
  - **from** フィールドは、要求元についての制約を指定します。
  - **to** フィールドは、要求のターゲットおよびパラメーターの制約を指定します。
  - **when** フィールドは、ルールを適用する追加の条件を指定します。

手順

1. **AuthorizationPolicy** リソースを作成します。以下の例は、ingress-policy **AuthorizationPolicy** を更新して、IP アドレスが Ingress ゲートウェイにアクセスすることを拒否するリソースを示しています。

```
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. リソースを作成した後に以下のコマンドを実行して、namespace にリソースを作成します。namespace は、**AuthorizationPolicy** リソースの **metadata.namespace** フィールドと一致する必要があります。

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

次のステップ

その他の一般的な設定については、以下の例を参照してください。

1.13.2.1. プロジェクト内通信の設定

AuthorizationPolicy を使用して Service Mesh コントロールプレーンを設定し、メッシュまたはメッシュ内のサービスとの通信トラフィックを許可したり、拒否したりすることができます。

1.13.2.1.1. namespace 外のサービスへのアクセス制限

以下の **AuthorizationPolicy** リソースの例を使用して、**bookinfo** namespace にないソースからの要求を拒否することができます。

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

1.13.2.1.2. allow-all およびデフォルトの deny-all 認証ポリシーの作成

以下の例は、**bookinfo** namespace のすべてのワークロードへの完全なアクセスを許可する allow-all 認証ポリシーを示しています。

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

以下の例は、**bookinfo** namespace のすべてのワークロードへのアクセスを拒否するポリシーを示しています。

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
```

```
  namespace: bookinfo
spec:
  {}
```

1.13.2.2. ingress ゲートウェイへのアクセスの許可または拒否

認証ポリシーを設定し、IP アドレスに基づいて許可または拒否リストを追加できます。

```
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. JSON Web トークンを使用したアクセスの制限

JSON Web Token (JWT) を使用してメッシュにアクセスできる内容を制限できます。認証後に、ユーザーまたはサービスはそのトークンに関連付けられたルート、サービスにアクセスできます。

ワークロードでサポートされる認証方法を定義する **RequestAuthentication** リソースを作成します。以下の例では、**http://localhost:8080/auth/realms/master** で実行される JWT を受け入れます。

```
apiVersion: "security.istio.io/v1beta1"
kind: "RequestAuthentication"
metadata:
  name: "jwt-example"
  namespace: bookinfo
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"
```

次に、同じ namespace に **AuthorizationPolicy** リソースを作成し、作成した **RequestAuthentication** リソースと連携させます。以下の例では、要求を **httpbin** ワークロードに送信する際に、JWT は **Authorization** ヘッダーになければなりません。

```
apiVersion: "security.istio.io/v1beta1"
kind: "AuthorizationPolicy"
metadata:
  name: "frontend-ingress"
  namespace: bookinfo
spec:
```

```
selector:
  matchLabels:
    app: httpbin
action: DENY
rules:
- from:
  - source:
      notRequestPrincipals: ["*"]
```

1.13.3. 暗号化スイートおよび ECDH 曲線の設定

暗号化スイートおよび ECDH 曲線 (Elliptic-curve Diffie-Hellman) は、サービスメッシュのセキュリティーを保護するのに役立ちます。暗号化スイートのコンマ区切りの一覧を **spec.security.controlplane.tls.cipherSuites** を使用して定義し、 ECDH 曲線を **ServiceMeshControlPlane** リソースの **spec.security.controlplane.tls.ecdhCurves** を使用して定義できます。これらの属性のいずれかが空の場合、デフォルト値が使用されます。

サービスメッシュが TLS 1.2 以前のバージョンを使用する場合、**cipherSuites** 設定が有効になります。この設定は、TLS 1.3 でネゴシエートする場合は影響を与えません。

コンマ区切りの一覧に暗号化スイートを優先度順に設定します。たとえば、**ecdhCurves: CurveP256, CurveP384** は、**CurveP256** を **CurveP384** よりも高い優先順位として設定します。

注記

暗号化スイートを設定する場合は、**TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256** または **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256** のいずれかを追加する必要があります。HTTP/2 のサポートには、1 つ以上の以下の暗号スイートが必要です。

サポートされている暗号化スイートは以下になります。

- 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

サポートされる ECDH 曲線は以下のとおりです。

- CurveP256
- CurveP384
- CurveP521
- X25519

1.13.4. 外部認証局キーおよび証明書の追加

デフォルトで、Red Hat OpenShift Service Mesh は自己署名ルート証明書およびキーを生成し、それらを使用してワークロード証明書に署名します。ユーザー定義の証明書およびキーを使用して、ユーザー定義のルート証明書を使用してワークロード証明書に署名することもできます。このタスクは、証明書およびキーを Service Mesh にプラグインするサンプルを示しています。

前提条件

- 相互 TLS を有効にして Red Hat OpenShift Service Mesh をインストールし、証明書を設定する。
- この例では、Maistra リポジトリー からの証明書を使用します。実稼働環境の場合は、認証局から独自の証明書を使用します。
- Bookinfo サンプルアプリケーションをデプロイして以下の手順で結果を確認しておく。
- OpenSSL は、証明書を検証するために必要です。

1.13.4.1. 既存の証明書およびキーの追加

既存の署名 (CA) 証明書およびキーを使用するには、CA 証明書、キー、ルート証明書が含まれる信頼ファイルのチェーンを作成する必要があります。それぞれの対応する証明書について以下のファイル名をそのまま使用する必要があります。CA 証明書は **ca-cert.pem** と呼ばれ、キーは **ca-key.pem** であり、**ca-cert.pem** を署名するルート証明書は **root-cert.pem** と呼ばれます。ワークロードで中間証明書を使用する場合は、**cert-chain.pem** ファイルでそれらを指定する必要があります。

1. Maistra リポジトリー からサンプル証明書をローカルに保存し、**<path>** を証明書へのパスに置き換えます。
2. **cacert** という名前のシークレットを作成します。これには、入力ファイルの **ca-cert.pem**、**ca-key.pem**、**root-cert.pem** および **cert-chain.pem** が含まれます。

```
$ 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. **ServiceMeshControlPlane** リソースで、**spec.security.dataPlane.mtls true** を **true** に設定し、以下の例のように **certificateAuthority** フィールドを設定します。デフォルトの **rootCADir** は **/etc/cacerts** です。キーおよび証明書がデフォルトの場所にマウントされている場合は、**privateKey** を設定する必要はありません。Service Mesh は、secret-mount ファイルから証明書およびキーを読み取ります。

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

4. **cacert** シークレットを作成/変更/削除した後に、変更を有効にするために、Service Mesh コントロールプレーンの **istiod** と **gateway** Pod を再起動する必要があります。以下のコマンドで Pod を再起動します。

```
$ oc -n istio-system delete pods -l 'app in (istiod,istio-ingressgateway, istio-egressgateway)'
```

Operator は、Pod を削除した後、自動的に再作成します。

5. bookinfo アプリケーションの Pod を再起動し、sidecar プロキシーがシークレットの変更を取り込むようにします。以下のコマンドで Pod を再起動します。

```
$ oc -n bookinfo delete pods --all
```

以下のような出力が表示されるはずです。

```
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. 以下のコマンドで、Pod が作成され、準備ができたことを確認します。

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

1.13.4.2. 証明書の確認

Bookinfo サンプルアプリケーションを使用して、ワークロード証明書が CA に差し込まれた証明書によって署名されていることを確認します。そのためには、マシンに **openssl** がインストールされている必要があります。

1. bookinfo ワークロードから証明書を抽出するには、以下のコマンドを使用します。

```
$ sleep 60
$ oc -n bookinfo 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}' bookinfo-proxy-cert.txt > certs.pem
$ awk 'BEGIN {counter=0;} /BEGIN CERT/{counter++} { print > "proxy-cert-" counter ".pem"}'
< certs.pem
```

   コマンドを実行すると、作業ディレクトリーに **proxy-cert-1.pem**、**proxy-cert-2.pem**、**proxy-cert-3.pem** の 3 つのファイルが作成されるはずです。

2. ルート証明書が管理者が指定したものと同じであることを確認します。**<path>** を証明書へのパスに置き換えます。

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

   ターミナルウィンドウで次の構文を実行します。

```
$ openssl x509 -in ./proxy-cert-3.pem -text -noout > /tmp/pod-root-cert.crt.txt
```

   ターミナルウィンドウで以下の構文を実行して、証明書を比較します。

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

   以下のような結果が表示されるはずです: **Files /tmp/root-cert.crt.txt and /tmp/pod-root-cert.crt.txt are identical**

3. CA 証明書が管理者が指定したものと同じであることを確認します。**<path>** を証明書へのパスに置き換えます。

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

   ターミナルウィンドウで次の構文を実行します。

```
$ openssl x509 -in ./proxy-cert-2.pem -text -noout > /tmp/pod-cert-chain-ca.crt.txt
```

   ターミナルウィンドウで以下の構文を実行して、証明書を比較します。

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

   以下のような結果が表示されるはずです: **Files /tmp/ca-cert.crt.txt and /tmp/pod-cert-chain-ca.crt.txt are identical.**

4. ルート証明書からワークロード証明書への証明書チェーンを確認します。**<path>** を証明書へのパスに置き換えます。

```
$ openssl verify -CAfile <(cat <path>/ca-cert.pem <path>/root-cert.pem) ./proxy-cert-1.pem
```

以下のような出力が表示されるはずです: **./proxy-cert-1.pem: OK**

1.13.4.3. 証明書の削除

追加した証明書を削除するには、以下の手順に従います。

1. シークレット **cacerts** を削除します。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

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

2. **ServiceMeshControlPlane** リソースで自己署名ルート証明書を使用して Service Mesh を再デプロイします。

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

1.14. SERVICE MESH でのトラフィックの管理

Red Hat OpenShift Service Mesh を使用すると、サービス間のトラフィックのフローおよび API 呼び出しを制御できます。サービスメッシュの一部のサービスはメッシュ内で通信する必要があり、他のサービスは非表示にする必要がある場合があります。トラフィックを管理して、特定のバックエンドサービスを非表示にし、サービスを公開し、テストまたはバージョン管理デプロイメントを作成し、または一連のサービスのセキュリティーの層を追加できます。

1.14.1. ゲートウェイの使用

ゲートウェイを使用してメッシュの受信トラフィックおよび送信トラフィックを管理することで、メッシュに入るか、またはメッシュを出るトラフィックを指定できます。ゲートウェイ設定は、サービスワークロードと共に実行されるサイドカー Envoy プロキシーではなく、メッシュのエッジで実行されているスタンドアロンの Envoy プロキシーに適用されます。

Kubernetes Ingress API などのシステムに入るトラフィックを制御する他のメカニズムとは異なり、Red Hat OpenShift Service Mesh ゲートウェイではトラフィックのルーティングの機能および柔軟性を最大限に利用できます。Red Hat OpenShift Service Mesh ゲートウェイリソースは、Red Hat OpenShift Service Mesh TLS 設定を公開して設定するポートなど、4-6 の負荷分散プロパティーを階層化できます。アプリケーション層のトラフィックルーティング (L7) を同じ API リソースに追加する代わりに、通常の Red Hat OpenShift Service Mesh 仮想サービスをゲートウェイにバインドし、サービスメッシュ内の他のデータプレーントラフィックのようにゲートウェイトラフィックを管理することができます。

ゲートウェイは ingress トラフィックの管理に主に使用されますが、egress ゲートウェイを設定することもできます。egress ゲートウェイを使用すると、メッシュからのトラフィック専用の終了ノードを設定できます。これにより、サービスメッシュにセキュリティー制御を追加することで、外部ネットワークにアクセスできるサービスを制限できます。また、ゲートウェイを使用して完全に内部のプロキシーを設定することもできます。

ゲートウェイの例

ゲートウェイリソースは、着信または発信 HTTP/TCP 接続を受信するメッシュのエッジで動作するロードバランサーを表します。この仕様は、公開する必要のあるポートのセット、使用するプロトコルのタイプ、ロードバランサー用の SNI 設定などについて記述します。

以下の例は、外部 HTTPS Ingress トラフィックのゲートウェイ設定を示しています。

```
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
```

このゲートウェイ設定により、ポート 443 での **ext-host.example.com** からメッシュへの HTTPS トラフィックが可能になりますが、トラフィックのルーティングは指定されません。

ルーティングを指定し、ゲートウェイが意図される通りに機能するには、ゲートウェイを仮想サービスにバインドする必要もあります。これは、以下の例のように、仮想サービスのゲートウェイフィールドを使用して実行します。

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

次に、仮想サービスを外部トラフィックのルーティングルールを使用して設定できます。

1.14.1.1. Ingress トラフィックの管理

Red Hat OpenShift Service Mesh では、Ingress Gateway は、モニタリング、セキュリティー、ルートルールなどの機能をクラスターに入るトラフィックに適用できるようにします。Service Meshゲートウェイを使用してサービスメッシュ外のサービスを公開します。

1.14.1.1.1. Ingress IP およびポートの判別

Ingress 設定は、環境が外部ロードバランサーをサポートするかどうかによって異なります。外部ロードバランサーはクラスターの Ingress IP およびポートに設定されます。クラスターの IP およびポートが外部ロードバランサーに設定されているかどうかを判別するには、以下のコマンドを実行します。こ

の例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

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

このコマンドは、namespace のそれぞれの項目の **NAME**、**TYPE**、**CLUSTER-IP**、**EXTERNAL-IP**、**PORT(S)**、および **AGE** を返します。

EXTERNAL-IP 値が設定されている場合には、環境には Ingress ゲートウェイに使用できる外部ロードバランサーがあります。

EXTERNAL-IP の値が **<none>** または永続的に **<pending>** の場合、環境は Ingress ゲートウェイの外部ロードバランサーを提供しません。サービスの ノードポート を使用してゲートウェイにアクセスできます。

1.14.1.1.1.1. ロードバランサーを使用した Ingress ポートの判別

お使いの環境に外部ロードバランサーがある場合には、以下の手順に従います。

手順

1. 以下のコマンドを実行して Ingress IP およびポートを設定します。このコマンドは、ターミナルに変数を設定します。

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

2. 以下のコマンドを実行して Ingress ポートを設定します。

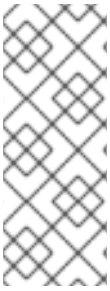
```
$ export INGRESS_PORT=$(oc -n istio-system get service istio-ingressgateway -o
jsonpath='{.spec.ports[?(@.name=="http2")].port}')
```

3. 以下のコマンドを実行してセキュアな Ingress ポートを設定します。

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

4. 以下のコマンドを実行して TCP Ingress ポートを設定します。

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



注記

一部の環境では、ロードバランサーは IP アドレスの代わりにホスト名を使用して公開される場合があります。この場合、Ingress ゲートウェイの **EXTERNAL-IP** 値は IP アドレスではありません。これはホスト名であり、直前のコマンドは **INGRESS_HOST** 環境変数の設定に失敗します。

失敗した場合には、以下のコマンドを使用して **INGRESS_HOST** 値を修正します。

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

1.14.1.1.1.2. ロードバランサーのない Ingress ポートの判別

お使いの環境に外部ロードバランサーがない場合は、Ingress ポートを判別し、代わりにノードポートを使用します。

手順

1. Ingress ポートを設定します。

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

2. 以下のコマンドを実行してセキュアな Ingress ポートを設定します。

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

3. 以下のコマンドを実行して TCP Ingress ポートを設定します。

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

1.14.1.2. Ingress ゲートウェイの設定

Ingress ゲートウェイは、受信 HTTP/TCP 接続を受信するメッシュのエッジで稼働するロードバランサーです。このゲートウェイは、公開されるポートおよびプロトコルを設定しますが、これにはトラフィックルーティングの設定は含まれません。Ingress トラフィックに対するトラフィックルーティングは、内部サービス要求の場合と同様に、ルーティングルールで設定されます。

以下の手順では、ゲートウェイを作成し、**/productpage** と **/login** のパスの外部トラフィックに、Bookinfo サンプルアプリケーションのサービスを公開するように、**VirtualService** を設定します。

手順

1. トラフィックを受け入れるゲートウェイを作成します。

   a. YAML ファイルを作成し、以下の YAML をこれにコピーします。

ゲートウェイの例 (gateway.yaml)

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

b. YAML ファイルを適用します。

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

2. **VirtualService** オブジェクトを作成し、ホストヘッダーを再作成します。

a. YAML ファイルを作成し、以下の YAML をこれにコピーします。

仮想サービスの例

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

b. YAML ファイルを適用します。

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

3. ゲートウェイと VirtualService が正しく設定されていることを確認してください。

a. ゲートウェイ URL を設定します。

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

b. ポート番号を設定します。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

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

c. 明示的に公開されているページをテストします。

```
curl -s -I "$GATEWAY_URL/productpage"
```

想定される結果は **200** です。

1.14.2. 自動ルートについて

ゲートウェイの OpenShift ルートは Service Mesh で自動的に管理されます。Istio ゲートウェイがサービスメッシュ内で作成され、更新され、削除されるたびに、OpenShift ルートが作成され、更新され、削除されます。

1.14.2.1. サブドメインのあるルート

Red Hat OpenShift Service Mesh はサブドメインでルートを作成しますが、OpenShift Container Platform はこれを有効にするように設定される必要があります。***.domain.com** などのサブドメインはサポートされますが、デフォルトでは設定されません。ワイルドカードホストゲートウェイを設定する前に OpenShift Container Platform ワイルドカードポリシーを設定します。

詳細は、ワイルドカードルートの使用 を参照してください。

1.14.2.2. サブドメインルートの作成

以下の例では、サブドメインルートを作成する Bookinfo サンプルアプリケーションにゲートウェイを作成します。

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

Gateway リソースは、次の OpenShift ルートを作成します。ルートが以下のコマンドを使用して作成されていることを確認できます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

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

予想される出力

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

このゲートウェイが削除されると、Red Hat OpenShift Service Mesh はルートを削除します。ただし、手動で作成したルートは Red Hat OpenShift Service Mesh によって変更されることはありません。

1.14.2.3. ルートラベルとアノテーション

OpenShift ルートでは、特定のラベルまたはアノテーションが必要になる場合があります。たとえば、OpenShift ルートの高度な機能の一部は、特別なアノテーションを使用して管理されます。以下の関連情報セクションのルート固有のアノテーションを参照してください。

このユースケースおよび他のユースケースでは、Red Hat OpenShift Service Mesh は (**kubectl.kubernetes.io** で始まるものを除く) Istio Gateway リソースにあるすべてのラベルとアノテーションを管理対象の OpenShift Route リソースにコピーします。

Service Mesh によって作成される OpenShift ルートで特定のラベルまたはアノテーションが必要な場合は、それらを Istio Gateway リソースで作成すると、Service Mesh で管理される OpenShift ルートリソースにコピーされます。

1.14.2.4. 自動ルート作成の無効化

デフォルトで、**ServiceMeshControlPlane** リソースは Istio ゲートウェイリソースと OpenShift ルートを自動的に同期します。自動ルート作成を無効にすると、特殊なケースがある場合やルートを手動で制御する場合に、ルートをより柔軟に制御できます。

1.14.2.4.1. 特定のケースでの自動ルート作成の無効化

特定の Istio ゲートウェイの OpenShift ルートの自動管理を無効にする場合は、アノテーション **maistra.io/manageRoute: false** をゲートウェイのメタデータ定義に追加する必要があります。Red Hat OpenShift Service Mesh は、他の Istio ゲートウェイの自動管理を維持しつつ、このアノテーションの付いた Istio ゲートウェイを無視します。

1.14.2.4.2. すべてのケースでの自動ルート作成の無効化

メッシュ内のすべてのゲートウェイの OpenShift ルートの自動管理を無効にできます。

Istio ゲートウェイと OpenShift ルート間の統合を無効にするには、**ServiceMeshControlPlane** フィールド **gateways.openshiftRoute.enabled** を **false** に設定します。たとえば、以下のリソーススニペットを参照してください。

```
apiVersion: maistra.io/v1alpha1
kind:
metadata:
  namespace: istio-system
spec:
  gateways:
    openshiftRoute:
      enabled: false
```

1.14.3. サービスエントリーについて

サービスエントリーは、Red Hat OpenShift Service Mesh が内部で維持するサービスレジストリーにエントリーを追加します。サービスエントリーの追加後、Envoy プロキシーはメッシュ内のサービスであるかのようにトラフィックをサービスに送信できます。サービスエントリーを使用すると、以下が可能になります。

- サービスメッシュ外で実行されるサービスのトラフィックを管理します。
- Web から消費される API やレガシーインフラストラクチャーのサービスへのトラフィックなど、外部宛先のトラフィックをリダイレクトし、転送します。
- 外部宛先の再試行、タイムアウト、およびフォールトインジェクションポリシーを定義します。
- 仮想マシンをメッシュに追加して、仮想マシン (VM) でメッシュサービスを実行します。

注記

別のクラスターからメッシュにサービスを追加し、Kubernetes でマルチクラスター Red Hat OpenShift Service Mesh メッシュを設定します。

サービスエントリーの例

以下の mesh-external サービスエントリーの例では、 **ext-resource** の外部依存関係を Red Hat OpenShift Service Mesh サービスレジストリーに追加します。

```
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
```

hosts フィールドを使用して外部リソースを指定します。これを完全に修飾することも、ワイルドカードの接頭辞が付けられたドメイン名を使用することもできます。

仮想サービスおよび宛先ルールを設定して、メッシュ内の他のサービスのトラフィックを設定するのと同じように、サービスエントリーへのトラフィックを制御できます。たとえば、以下の宛先ルールでは、トラフィックルートを、サービスエントリーを使用して設定される **ext-svc.example.com** 外部サービスへの接続のセキュリティーを保護するために相互 TLS を使用するように設定します。

```
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. VirtualServices の使用

仮想サービスを使用して、Red Hat OpenShift Service Mesh で複数バージョンのマイクロサービスに要求を動的にルーティングできます。仮想サービスを使用すると、以下が可能になります。

- 単一の仮想サービスで複数のアプリケーションサービスに対応する。メッシュが Kubernetes を使用する場合などに、仮想サービスを特定の namespace のすべてのサービスを処理するように設定できます。仮想サービスを使用すると、モノリシックなアプリケーションをシームレスに、個別のマイクロサービスで設定されるサービスに変換できます。
- ingress および egress トラフィックを制御できるようにゲートウェイと組み合わせてトラフィックルールを設定する。

1.14.4.1. VirtualServices の設定

要求は、仮想サービスを使用してサービスメッシュ内のサービスにルーティングされます。それぞれの仮想サービスは、順番に評価される一連のルーティングルールで設定されます。Red Hat OpenShift Service Mesh は、仮想サービスへのそれぞれの指定された要求をメッシュ内の特定の実際の宛先に一致させます。

仮想サービスがない場合、Red Hat OpenShift Service Mesh はすべてのサービスインスタンス間のラウンドロビン負荷分散を使用してトラフィックを分散します。仮想サービスを使用すると、1 つ以上のホスト名のトラフィック動作を指定できます。仮想サービスのルーティングルールでは、仮想サービスのトラフィックを適切な宛先に送信する方法を Red Hat OpenShift Service Mesh に指示します。ルートの宛先は、同じサービスのバージョンまたは全く異なるサービスにすることができます。

手順

1. アプリケーションに接続するユーザーに基づき、異なるバージョンの Bookinfo アプリケーションサービスのサンプルに、要求をルーティングする以下の例を使用して、YAML ファイルを作成します。

   VirtualService.yaml の例

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

2. 以下のコマンドを実行して **VirtualService.yaml** を適用します。**VirtualService.yaml** はファイルへのパスです。

```
$ oc apply -f <VirtualService.yaml>
```

1.14.4.2. VirtualService 設定リファレンス

パラメーター	説明
`spec:` `  hosts:`	**hosts** フィールドには、ルーティングルールが適用される仮想サービスのユーザーの宛先アドレスが一覧表示されます。これは、サービスへの要求送信に使用するアドレスです。仮想サービスのホスト名は、IP アドレス、DNS 名または完全修飾ドメイン名に解決される短縮名になります。

`spec:`
`  http:`
`  - match:`
	http セクションには、ホストフィールドで指定された宛先に送信される HTTP/1.1、HTTP2、および gRPC トラフィックのルーティングの一致条件とアクションを記述する仮想サービスのルーティングルールが含まれます。ルーティングルールは、トラフィックの宛先と、指定の一致条件で設定されます。この例の最初のルーティングルールには条件があり、match フィールドで始まります。この例では、このルーティングはユーザー **jason** からの要求すべてに適用されます。**headers**、**end-user**、および **exact** フィールドを追加し、適切な要求を選択します。

`spec:`
`  http:`
`  - match:`
`  - destination:`
	route セクションの **destination** フィールドは、この条件に一致するトラフィックの実際の宛先を指定します。仮想サービスのホストとは異なり、宛先のホストは Red Hat OpenShift Service Mesh サービスレジストリーに存在する実際の宛先でなければなりません。これは、プロキシーが含まれるメッシュサービス、またはサービスエントリーを使用して追加されたメッシュ以外のサービスである可能性があります。この例では、ホスト名は Kubernetes サービス名です。

1.14.5. 宛先ルールについて

宛先ルールは仮想サービスのルーティングルールが評価された後に適用されるため、それらはトラフィックの実際の宛先に適用されます。仮想サービスはトラフィックを宛先にルーティングします。宛先ルールでは、その宛先のトラフィックに生じる内容を設定します。

デフォルトで、Red Hat OpenShift Service Mesh はラウンドロビンの負荷分散ポリシーを使用します。このポリシーでは、プールの各サービスインスタンスが順番に要求を取得します。Red Hat OpenShift Service Mesh は以下のモデルもサポートします。このモデルは、特定のサービスまたはサービスサブセットへの要求の宛先ルールに指定できます。

- Random: 要求はプール内のインスタンスにランダムに転送されます。

- Weighted: 要求は特定のパーセンテージに応じてプールのインスタンスに転送されます。
- Least requests: 要求は要求の数が最も少ないインスタンスに転送されます。

宛先ルールの例

以下の宛先ルールの例では、異なる負荷分散ポリシーで **my-svc** 宛先サービスに 3 つの異なるサブセットを設定します。

```
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. ネットワークポリシーについて

Red Hat OpenShift Service Mesh は、Service Mesh コントロールプレーンおよびアプリケーションネームスペースで多数の **NetworkPolicies** リソースを自動的に作成し、管理します。これは、アプリケーションとコントロールプレーンが相互に通信できるようにするために使用されます。

たとえば、OpenShift Container Platform クラスターが SDN プラグインを使用するように設定されている場合、Red Hat OpenShift Service Mesh は各メンバープロジェクトで **NetworkPolicy** リソースを作成します。これにより、他のメッシュメンバーおよびコントロールプレーンからのメッシュ内のすべての Pod に対する ingress が有効になります。また、これにより Ingress がメンバープロジェクトのみに制限されます。メンバー以外のプロジェクトの Ingress が必要な場合は、**NetworkPolicy** を作成してそのトラフィックを許可する必要があります。Service Mesh から namespace を削除する場合、この **NetworkPolicy** リソースはプロジェクトから削除されます。

1.14.6.1. NetworkPolicy 自動作成の無効化

NetworkPolicy リソースの自動作成および管理を無効にする場合 (例: 会社のセキュリティーポリシーを適用したり、メッシュ内の Pod への直接アクセスを許可する場合など) はこれを実行できます。**ServiceMeshControlPlane** を編集し、**spec.security.manageNetworkPolicy** を **false** に設定できます。

注記

spec.security.manageNetworkPolicy を無効にすると、Red Hat OpenShift Service Mesh は、**NetworkPolicy** オブジェクトをひとつも作成しません。システム管理者は、ネットワークを管理し、この原因の問題を修正します。

前提条件

- Red Hat OpenShift Service Mesh Operator バージョン 2.1.1 以降がインストールされている。
- **ServiceMeshControlPlane** リソースはバージョン 2.1 以降に更新されている。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → Installed Operators** をクリックします。
2. **Project** メニューから、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
3. Red Hat OpenShift Service Mesh Operator をクリックします。**Istio Service Mesh Control Plane** 列で、**ServiceMeshControlPlane** の名前 (**basic-install** など) をクリックします。
4. **Create ServiceMeshControlPlane Details** ページで、**YAML** をクリックして設定を変更します。
5. 以下の例のように、**ServiceMeshControlPlane** フィールド **spec.security.manageNetworkPolicy** を **false** に設定します。

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

6. **Save** をクリックします。

1.14.7. トラフィック管理のサイドカーの設定

デフォルトで、Red Hat OpenShift Service Mesh は、すべての Envoy プロキシーを、トラフィックの転送時にすべてのポートで関連付けられたワークロードについてのトラフィックを受け入れ、メッシュ内のすべてのワークロードに到達するように設定します。サイドカー設定を使用して以下を実行できます。

- Envoy プロキシーが受け入れるポートとプロトコルのセットを微調整します。
- Envoy プロキシーが到達できるサービスのセットを制限します。

注記

サービスメッシュのパフォーマンスを最適化するには、Envoy プロキシー設定の制限を検討してください。

Bookinfo サンプルアプリケーションで、同じ namespace およびコントロールプレーンで実行されている他のサービスに度のサービスからでもアクセスできるように Sidecar を設定します。この Sidecar 設定は、Red Hat OpenShift Service Mesh ポリシーおよび Telemetry 機能での使用に必要になります。

手順

1. 以下の例を使用して YAML ファイルを作成し、サイドカー設定を特定の namespace の全ワークロードに適用するように指定します。それ以外の場合は、**workloadSelector** を使用して特定のワークロードを選択します。

   sidecar.yaml の例

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

2. 以下のコマンドを実行して **sidecar.yaml** を適用します。ここでは、**sidecar.yaml** はファイルへのパスです。

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

3. サイドカーが正常に作成されたことを確認するには、以下のコマンドを実行します。

   ```
   $ oc get sidecar
   ```

1.14.8. ルーティングチュートリアル

本書では Bookinfo サンプルアプリケーションを参照して、サンプルアプリケーションでのルーティングの例を説明します。Bookinfo アプリケーション をインストールして、これらのルーティングのサンプルがどのように機能するかを確認します。

1.14.8.1. Bookinfo ルーティングチュートリアル

Service Mesh Bookinfo サンプルアプリケーションは、それぞれが複数のバージョンを持つ 4 つの別個のマイクロサービスで設定されます。Bookinfo サンプルアプリケーションをインストールした後に、**reviews** マイクロサービスの 3 つの異なるバージョンが同時に実行されます。

ブラウザーで Bookinfo アプリケーションの **/product** ページにアクセスして数回更新すると、書評の出力に星評価が含まれる場合と含まれない場合があります。ルーティング先の明示的なデフォルトサービスバージョンがない場合、Service Mesh は、利用可能なすべてのバージョンに要求をルーティングしていきます。

このチュートリアルは、すべてのトラフィックをマイクロサービスの **v1** (バージョン 1) にルーティングするルールを適用するのに役立ちます。後に、HTTP リクエストヘッダーの値に基づいてトラフィックをルーティングするためのルールを適用できます。

前提条件:

- 以下の例に合わせて Bookinfo サンプルアプリケーションをデプロイする。

1.14.8.2. 仮想サービスの適用

以下の手順では、マイクロサービスのデフォルトバージョンを設定する仮想サービスを適用して、各マイクロサービスの **v1** にすべてのトラフィックをルーティングします。

手順

1. 仮想サービスを適用します。

```
$ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.2/samples/bookinfo/networking/virtual-service-all-v1.yaml
```

2. 仮想サービスの適用を確認するには、以下のコマンドで定義されたルートを表示します。

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

このコマンドでは、YAML 形式で **kind: VirtualService** のリソースを返します。

Service Mesh を Bookinfo マイクロサービスの **v1** バージョン (例: **reviews** サービスバージョン 1) にルーティングするように設定しています。

1.14.8.3. 新規ルート設定のテスト

Bookinfo アプリケーションの /**productpage** を更新して、新しい設定をテストします。

手順

1. **GATEWAY_URL** パラメーターの値を設定します。この変数を使用して、Bookinfo 製品ページの URL を後で見つけることができます。この例では、istio-system はコントロールプレーンプロジェクトの名前です。

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

2. 以下のコマンドを実行して、製品ページの URL を取得します。

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

3. ブラウザーで Bookinfo サイトを開きます。

更新回数に関係なく、ページのレビュー部分は星評価なしに表示されます。これは、Service Mesh を、reviews サービスのすべてのトラフィックをバージョン **reviews:v1** にルーティングするように設定しているためであり、サービスのこのバージョンは星評価サービスにアクセスしません。

サービスメッシュは、トラフィックを 1 つのバージョンのサービスにルーティングするようになりました。

1.14.8.4. ユーザーアイデンティティーに基づくルート

ルート設定を変更して、特定のユーザーからのトラフィックすべてが特定のサービスバージョンにルーティングされるようにします。この場合、**jason** という名前のユーザーからのトラフィックはすべて、サービス **reviews:v2** にルーティングされます。

Service Mesh には、ユーザーアイデンティティーについての特別な組み込み情報はありません。この例は、**productpage** サービスが reviews サービスへのすべてのアウトバウンド HTTP リクエストにカスタム **end-user** ヘッダーを追加することで有効にされます。

手順

1. 以下のコマンドを実行して、Bookinfo アプリケーション例でユーザーベースのルーティングを有効にします。

```
$ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.2/samples/bookinfo/networking/virtual-service-reviews-test-v2.yaml
```

2. 以下のコマンドを実行して、ルールの作成を確認します。このコマンドは、**kind: VirtualService** のすべてのリソースを YAML 形式で返します。

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

3. Bookinfo アプリケーションの **/productpage** で、パスワードなしでユーザー **jason** としてログインします。

4. ブラウザーを更新します。各レビューの横に星評価が表示されます。

5. 別のユーザーとしてログインします (任意の名前を指定します)。ブラウザーを更新します。これで星がなくなりました。Jason 以外のすべてのユーザーのトラフィックが **reviews:v1** にルーティングされるようになりました。

ユーザーアイデンティティーに基づいてトラフィックをルーティングするように Bookinfo のアプリケーションサンプルが正常に設定されています。

1.15. メトリックス、ログ、およびトレース

アプリケーションをメッシュに追加したら、アプリケーション経由でデータフローを確認できます。独自のアプリケーションがインストールされていない場合、Bookinfo サンプルアプリケーション をインストールして、Red Hat OpenShift Service Mesh での可観測性の機能を確認できます。

1.15.1. コンソールアドレスの検出

Red Hat OpenShift Service Mesh は、サービスメッシュデータを表示する以下のコンソールを提供します。

- **Kiali** コンソール: Kiali は Red Hat OpenShift Service Mesh の管理コンソールです。
- **Jaeger** コンソール: Jaeger は Red Hat OpenShift 分散トレースの管理コンソールです。
- **Grafana** コンソール: Grafana は、Istio データの高度なクエリーとメトリックス分析およびダッシュボードをメッシュ管理者に提供します。任意で、Grafana を使用してサービスメッシュメトリクスを分析できます。
- **Prometheus** コンソール: Red Hat OpenShift Service Mesh は Prometheus を使用してサービスからのテレメトリー情報を保存します。

Service Mesh コントロールプレーンのインストール時に、インストールされた各コンポーネントのルートを自動的に生成します。ルートアドレスを作成したら、Kiali、Jaeger、Prometheus、または Grafana コンソールにアクセスして、サービスメッシュデータを表示および管理できます。

前提条件

- コンポーネントが有効で、インストールされていること。たとえば、分散トレースをインストールしていない場合、Jaeger コンソールにはアクセスできません。

OpenShift コンソールからの手順

1. cluster-admin 権限を持つユーザーとして OpenShift Container Platform Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

2. **Networking → Routes** に移動します。

3. **Routes** ページで、**Namespace** メニューから Service Mesh コントロールプレーンプロジェクトを選択します (例: **istio-system**)。
   Location 列には、各ルートのリンク先アドレスが表示されます。

4. 必要な場合は、フィルターを使用して、アクセスするルートを持つコンポーネントコンソールを検索します。ルートの **Location** をクリックしてコンソールを起動します。

5. **Log In With OpenShift** をクリックします。

CLI からの手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

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

2. Service Mesh コントロールプレーンプロジェクトに切り替えます。この例では、**istio-system** は Service Mesh コントロールプレーンプロジェクトです。以下のコマンドを実行します。

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

3. 各種 Red Hat OpenShift Service Mesh コンソールのルートを取得するには、以下のコマンドを実行します。

```
$ oc get routes
```

   このコマンドは、Kiali、Jaeger、Prometheus、および Grafana Web コンソールの URL と、サービスメッシュ内の他のルートの URL を返します。以下のような出力が表示されるはずです。

```
NAME                   HOST/PORT                        SERVICES                PORT    TERMINATION
bookinfo-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. **HOST/PORT** コラムからアクセスするコンソールの URL をブラウザーにコピーして、コンソールを開きます。

5. **Log In With OpenShift** をクリックします。

1.15.2. Kiali コンソールへのアクセス

Kiali コンソールでアプリケーションのトポロジー、健全性、およびメトリクスを表示できます。サービスで問題が発生した場合には、Kiali コンソールは、サービス経由でデータフローを表示できます。抽象アプリケーションからサービスおよびワークロードまで、さまざまなレベルでのメッシュコンポーネントに関する洞察を得ることができます。Kiali は、リアルタイムで namespace のインタラクティブなグラフビューも提供します。

Kiali コンソールにアクセスするには、Red Hat OpenShift Service Mesh がインストールされ、Kiali がインストールおよび設定されている必要があります。

インストールプロセスにより、Kiali コンソールにアクセスするためのルートが作成されます。

Kiali コンソールの URL が分かっている場合は、直接アクセスできます。URL が分からない場合は、以下の指示を使用します。

管理者の手順

1. 管理者ロールで OpenShift Container Platform Web コンソールにログインします。

2. **Home → Projects** をクリックします。

3. **Projects** ページで、必要に応じてフィルターを使用してプロジェクトの名前を検索します。

4. プロジェクトの名前をクリックします (例: **bookinfo**)。

5. **Project details** ページの **Launcher** セクションで、**Kiali** リンクをクリックします。

6. OpenShift Container Platform コンソールにアクセスするときに使用するものと同じユーザー名とパスワードを使用して Kiali コンソールにログインします。
   初回の Kiali コンソールへのログイン時に、表示するパーミッションを持つサービスメッシュ内のすべての namespace を表示する **Overview** ページが表示されます。

   コンソールのインストールを検証中で、namespace がまだメッシュに追加されていない場合、**istio-system** 以外のデータは表示されない可能性があります。

開発者の手順

1. 開発者ロールで OpenShift Container Platform Web コンソールにログインします。

2. **Project** をクリックします。

3. 必要に応じて、**Project Details** ページで、フィルターを使用してプロジェクトの名前を検索します。

4. プロジェクトの名前をクリックします (例: **bookinfo**)。

5. **Project** ページの **Launcher** セクションで、**Kiali** リンクをクリックします。

6. **Log In With OpenShift** をクリックします。

1.15.3. Kiali コンソールでのサービスメッシュデータの表示

Kiali グラフは、メッシュトラフィックの強力な視覚化を提供します。トポロジーは、リアルタイムの要求トラフィックを Istio 設定情報と組み合わせ、サービスメッシュの動作に関する洞察を即座に得て、問題を迅速に特定できるようにします。複数のグラフタイプを使用すると、トラフィックを高レベルのサービストポロジー、低レベルのワークロードトポロジー、またはアプリケーションレベルのトポロジーとして視覚化できます。

以下から選択できるグラフがいくつかあります。

- **App** グラフ は、同じラベルが付けられたすべてのアプリケーションの集約ワークロードを示します。

- **Service** グラフ は、メッシュ内の各サービスのノードを表示しますが、グラフからすべてのアプリケーションおよびワークロードを除外します。これは高レベルのビューを提供し、定義されたサービスのすべてのトラフィックを集約します。

- **Versioned App** グラフ は、アプリケーションの各バージョンのノードを表示します。アプリケーションの全バージョンがグループ化されます。

- **Workload** グラフ は、サービスメッシュの各ワークロードのノードを表示します。このグラフでは、app および version のラベルを使用する必要はありません。アプリケーションが version ラベルを使用しない場合は、このグラフを使用します。

グラフノードは、さまざまな情報で装飾され、仮想サービスやサービスエントリーなどのさまざまなルートルーティングオプションや、フォールトインジェクションやサーキットブレーカーなどの特別な設定を指定します。mTLS の問題、レイテンシーの問題、エラートラフィックなどを特定できます。グラフは高度な設定が可能で、トラフィックのアニメーションを表示でき、強力な検索機能や非表示機能があります。

Legend ボタンをクリックして、グラフに表示されるシェイプ、色、矢印、バッジに関する情報を表示します。

メトリクスの要約を表示するには、グラフ内のノードまたはエッジを選択し、そのメトリクスの詳細をサマリーの詳細パネルに表示します。

1.15.3.1. Kiali でのグラフレイアウトの変更

Kiali グラフのレイアウトは、アプリケーションのアーキテクチャーや表示データによって異なることがあります。たとえば、グラフノードの数およびそのインタラクションにより、Kiali グラフのレンダリング方法を判別できます。すべての状況に適した単一のレイアウトを作成することは不可能であるため、Kiali は複数の異なるレイアウトの選択肢を提供します。

前提条件

- 独自のアプリケーションがインストールされていない場合は、Bookinfo サンプルアプリケーションをインストールします。次に、以下のコマンドを複数回入力して Bookinfo アプリケーションのトラフィックを生成します。

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

このコマンドはアプリケーションの **productpage** マイクロサービスにアクセスするユーザーをシミュレートします。

手順

1. Kiali コンソールを起動します。
2. **Log In With OpenShift** をクリックします。
3. Kiali コンソールで、**Graph** をクリックし、namespace グラフを表示します。
4. **Namespace** メニューから、アプリケーション namespace (例: **bookinfo**) を選択します。
5. 別のグラフレイアウトを選択するには、以下のいずれか、または両方を行います。
   - グラフの上部にあるメニューから、異なるグラフデータグループを選択します。
     - App graph
     - Service graph
     - Versioned App graph (デフォルト)
     - Workload graph
   - グラフの下部にある Legend から別のグラフレイアウトを選択します。
     - Layout default dagre
     - Layout 1 cose-bilkent
     - Layout 2 cola

1.15.3.2. Kiali コンソールでのログの表示

Kiali コンソールでワークロードのログを表示することができます。**Workload Details** ページには **Logs** タブが含まれており、アプリケーションとプロキシーログの両方を表示する統一されたログビューが表示されます。Kiali でログ表示を更新する頻度を選択できます。

Kiali に表示されるログのロギングレベルを変更するには、ワークロードまたはプロキシーのロギング設定を変更します。

前提条件

- Service Mesh がインストールされ、設定されている。
- Kiali がインストールされ、設定されている。
- Kiali コンソールのアドレス。
- アプリケーションまたは Bookinfo サンプルアプリケーションがメッシュに追加されました。

手順

1. Kiali コンソールを起動します。
2. **Log In With OpenShift** をクリックします。

Kiali Overview ページには、閲覧権限を持つメッシュに追加された namespace が表示されます。

3. **Workloads** をクリックします。
4. **Workloads** ページで、**Namespace** メニューからプロジェクトを選択します。
5. 必要な場合は、フィルターを使用して、表示するログがあるワークロードを見つけます。ワークロードの名前をクリックします。たとえば、**ratings-v1** をクリックします。
6. **Workload Details** ページで、**Logs** タブをクリックしてワークロードのログを表示します。

ヒント

ログエントリーが表示されない場合は、時間範囲または更新間隔のいずれかを調整する必要がある場合があります。

1.15.3.3. Kiali コンソールでのメトリックスの表示

Kiali コンソールで、アプリケーション、ワークロード、サービスのインバウンドおよびアウトバウンドメトリックスを表示できます。詳細ページには、以下のタブが含まれます。

- inbound Application metrics
- outbound Application metrics
- inbound Workload metrics
- outbound Workload metrics
- inbound Service metrics

これらのタブには、関連するアプリケーション、ワークロード、またはサービスレベルに合わせて調整された事前定義済みのメトリックスダッシュボードが表示されます。アプリケーションおよびワークロードの詳細ビューには、ボリューム、期間、サイズ、TCP トラフィックなどの要求および応答メトリックスが表示されます。サービスの詳細ビューには、インバウンドトラフィックの要求および応答メトリックスのみ表示されます。

Kiali では、チャート化されたディメンションを選択してチャートをカスタマイズできます。Kiali は、ソースまたは宛先プロキシーメトリックスによって報告されるメトリックスを表示することもできます。また、トラブルシューティングのために、Kiali はメトリックスでトレースをオーバーレイできます。

前提条件

- Service Mesh がインストールされ、設定されている。
- Kiali がインストールされ、設定されている。
- Kiali コンソールのアドレス。
- (オプション): 分散トレースがインストールされ、設定されます。

手順

1. Kiali コンソールを起動します。

2. **Log In With OpenShift** をクリックします。
   Kiali Overview ページには、閲覧権限を持つメッシュに追加された namespace が表示されます。

3. **Applications**、**Workloads**、または **Services** のいずれかをクリックします。

4. **Applications**、**Workloads**、または **Services** ページで、**Namespace** メニューからプロジェクトを選択します。

5. 必要な場合は、フィルターを使用して、ログを表示するアプリケーション、ワークロード、またはサービスを検索します。名前をクリックします。

6. **Application Detail**、**Workload Details**、または **Service Details** ページで、**Inbound Metrics** または **Outbound Metrics** タブをクリックしてメトリックスを表示します。

1.15.4. 分散トレース

分散トレースは、アプリケーションのサービス呼び出しのパスを追跡して、アプリケーション内の個々のサービスのパフォーマンスを追跡するプロセスです。アプリケーションでユーザーがアクションを起こすたびに、要求が実行され、多くのサービスが応答を生成するために対話する必要がある場合があります。この要求のパスは、分散トランザクションと呼ばれます。

Red Hat OpenShift Service Mesh は Red Hat OpenShift 分散トレースを使用して、開発者がマイクロサービスアプリケーションで呼び出しフローを表示できるようにします。

1.15.4.1. 既存の分散トレースインスタンスの接続

OpenShift Container Platform に既存の Red Hat OpenShift 分散トレースプラットフォームインスタンスがある場合、分散トレースにそのインスタンスを使用するように **ServiceMeshControlPlane** リソースを設定できます。

前提条件

- Red Hat OpenShift 分散トレースインスタンスがインストールされ、設定されている。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → Installed Operators** をクリックします。

2. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。

3. Red Hat OpenShift Service Mesh Operator をクリックします。**Istio Service Mesh Control Plane** 列で、**ServiceMeshControlPlane** リソースの名前 (**basic** など) をクリックします。

4. 分散トレースプラットフォームインスタンスの名前を **ServiceMeshControlPlane** に追加します。

   a. **YAML** タブをクリックします。

   b. 分散トレースプラットフォームインスタンスの名前を **ServiceMeshControlPlane** リソースの **spec.addons.jaeger.name** に追加します。以下の例では、**str-tracing-production** は分散トレースプラットフォームインスタンスの名前です。

      分散トレースの設定例

```
spec:
  addons:
    jaeger:
      name: distr-tracing-production
```

c. **Save** をクリックします。

5. **Reload** をクリックして、**ServiceMeshControlPlane** リソースが正しく設定されていることを確認します。

1.15.4.2. サンプリングレートの調整

トレースは、サービスメッシュ内のサービス間の実行パスです。トレースは、1 つ以上のスパンで設定されます。スパンは、名前、開始時間、および期間を持つ作業の論理単位です。サンプリングレートは、トレースが永続化される頻度を決定します。

Envoy プロキシーのサンプリングレートは、デフォルトでサービスメッシュでトレースの 100% をサンプリングするように設定されています。サンプリングレートはクラスターリソースおよびパフォーマンスを消費しますが、問題のデバッグを行う場合に役立ちます。Red Hat OpenShift Service Mesh を実稼働環境でデプロイする前に、値を小さめのトレースサイズに設定します。たとえば、**spec.tracing.sampling** を **100** に設定し、トレースの 1% をサンプリングします。

Envoy プロキシーサンプリングレートを、0.01% の増分を表すスケーリングされた整数として設定します。

基本的なインストールでは、**spec.tracing.sampling** は **10000** に設定され、トレースの 100% をサンプリングします。以下に例を示します。

- この値を 10 サンプル (トレースの 0.1%) に設定します。
- この値を 500 サンプル (トレースの 5%) に設定します。

注記

Envoy プロキシーサンプリングレートは、Service Mesh で利用可能なアプリケーションに適用され、Envoy プロキシーを使用します。このサンプリングレートは、Envoy プロキシーが収集および追跡するデータ量を決定します。

Jaeger リモートサンプリングレートは、Service Mesh の外部にあるアプリケーションに適用され、データベースなどの Envoy プロキシーを使用しません。このサンプリングレートは、分散トレースシステムが収集および保存するデータ量を決定します。詳細は、分散トレース設定オプション を参照してください。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → Installed Operators** をクリックします。
2. **Project** メニューをクリックし、コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
3. Red Hat OpenShift Service Mesh Operator をクリックします。**Istio Service Mesh Control Plane** 列で、**ServiceMeshControlPlane** リソースの名前 (**basic** など) をクリックします。
4. サンプリングレートを調整するには、**spec.tracing.sampling** に別の値を設定します。

a. **YAML** タブをクリックします。

b. **ServiceMeshControlPlane** リソースで **spec.tracing.sampling** の値を設定します。以下の例では、**100** に設定します。

Jaeger サンプリングの例

```
spec:
  tracing:
    sampling: 100
```

c. **Save** をクリックします。

5. **Reload** をクリックして、**ServiceMeshControlPlane** リソースが正しく設定されていることを確認します。

1.15.5. Jaeger コンソールへのアクセス

Jaeger コンソールにアクセスするには、Red Hat OpenShift Service Mesh がインストールされ、Red Hat OpenShift 分散トレースプラットフォームがインストールおよび設定されている必要があります。

インストールプロセスにより、Jaeger コンソールにアクセスするためのルートが作成されます。

Jaeger コンソールの URL が分かっている場合は、これに直接アクセスできます。URL が分からない場合は、以下の指示を使用します。

OpenShift コンソールからの手順

1. cluster-admin 権限を持つユーザーとして OpenShift Container Platform Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

2. **Networking → Routes** に移動します。

3. **Routes** ページで、**Namespace** メニューから Service Mesh コントロールプレーンプロジェクトを選択します (例: **istio-system**)。
   Location 列には、各ルートのリンク先アドレスが表示されます。

4. 必要な場合は、フィルターを使用して **jaeger** ルートを検索します。ルートの **Location** をクリックしてコンソールを起動します。

5. **Log In With OpenShift** をクリックします。

Kiali コンソールからの手順

1. Kiali コンソールを起動します。

2. 左側のナビゲーションペインで **Distributed Tracing** をクリックします。

3. **Log In With OpenShift** をクリックします。

CLI からの手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

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

2. コマンドラインを使用してルートの詳細をクエリーするには、以下のコマンドを入力します。この例では、**istio-system** が Service Mesh コントロールプレーンの namespace です。

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

3. ブラウザーを起動し、**https://<JAEGER_URL>** に移動します。ここで、**<JAEGER_URL>** は直前の手順で検出されたルートです。

4. OpenShift Container Platform コンソールへアクセスするときに使用するものと同じユーザー名とパスワードを使用してログインします。

5. サービスメッシュにサービスを追加し、トレースを生成している場合は、フィルターと **Find Traces** ボタンを使用してトレースデータを検索します。
コンソールインストールを検証すると、表示するトレースデータはありません。

Jaeger の設定に関する詳細は、分散トレースのドキュメント を参照してください。

1.15.6. Grafana コンソールへのアクセス

Grafana は、サービスメッシュメトリクスの表示、クエリー、および分析に使用できる解析ツールです。この例では、**istio-system** が Service Mesh コントロールプレーンの namespace です。Grafana にアクセスするには、以下の手順を実施します。

手順

1. OpenShift Container Platform Web コンソールにログインします。

2. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。

3. **Routes** をクリックします。

4. **Grafana** 行の **Location** コラムのリンクをクリックします。

5. OpenShift Container Platform 認証情報を使用して Grafana コンソールにログインします。

1.15.7. Prometheus コンソールへのアクセス

Prometheus は、マイクロサービスに関する多次元データの収集に使用できるモニタリングおよびアラートツールです。この例では、**istio-system** が Service Mesh コントロールプレーンの namespace です。

手順

1. OpenShift Container Platform Web コンソールにログインします。

2. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。

3. **Routes** をクリックします。

4. **Prometheus** 行の **Location** コラムのリンクをクリックします。

5. OpenShift Container Platform 認証情報を使用して Prometheus コンソールにログインします。

1.16. パフォーマンスおよびスケーラビリティー

デフォルトの **ServiceMeshControlPlane** 設定は実稼働環境での使用を目的としていません。それらはリソース面で制限のあるデフォルトの OpenShift Container Platform インストールに正常にインストールされるように設計されています。SMCP インストールに成功したことを確認したら、SMCP 内で定義した設定をお使いの環境に合わせて変更する必要があります。

1.16.1. コンピュートリソースでの制限の設定

デフォルトでは、**spec.proxy** には **cpu:10m** および **memory:128M** の設定があります。Pilot を使用している場合、**spec.runtime.components.pilot** には同じデフォルト値があります。

以下の例の設定は、1 秒あたり 1,000 サービスおよび 1,000 要求をベースとしています。**ServiceMeshControlPlane** で **cpu** および **memory** の値を変更できます。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → Installed Operators** をクリックします。

2. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。

3. Red Hat OpenShift Service Mesh Operator をクリックします。**Istio Service Mesh Control Plane** 列で、**ServiceMeshControlPlane** の名前 (**basic** など) をクリックします。

4. スタンドアロンの Jaeger インスタンスの名前を **ServiceMeshControlPlane** に追加します。

   a. **YAML** タブをクリックします。

   b. **ServiceMeshControlPlane** リソースの **spec.proxy.runtime.container.resources.requests.cpu** および **spec.proxy.runtime.container.resources.requests.memory** の値を設定します。

   バージョン 2.2 ServiceMeshControlPlane の例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.2
  proxy:
    runtime:
      container:
        resources:
          requests:
```

```
            cpu: 600m
            memory: 50Mi
          limits: {}

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

c. **Save** をクリックします。

5. **Reload** をクリックして、**ServiceMeshControlPlane** リソースが正しく設定されていることを確認します。

1.16.2. テスト結果の読み込み

アップストリームの Istio コミュニティーの負荷テストのメッシュは、1 秒あたり 70,000 のメッシュ全体の要求を持つ **1000** サービスと **2000** サイドカーで設定されます。Istio 1.12.3 を使用してテストを実行後、以下の結果が生成されました。

- Envoy プロキシーは、プロキシーを通過する 1 秒あたり/要求 1000 件あたり **0.35 vCPU** および **40 MB** メモリー を使用します。
- Istiod は **1 vCPU** および **1.5 GB** のメモリーを使用します。
- Envoy プロキシーは **2.65 ms** を 90 % レイテンシーに追加します。
- レガシー **istio-telemetry** サービス (Service Mesh 2.0 ではデフォルトで無効にされます) は、Mixer を使用するデプロイメントについて、1 秒あたりの 1000 のメッシュ全体の要求ごとに **0.6 vCPU** を使用します。データプレーンのコンポーネントである Envoy プロキシーは、システムを通過するデータフローを処理します。Service Mesh コントロールプレーンコンポーネントである Istiod は、データプレーンを設定します。データプレーンおよびコントロールプレーンには、さまざまなパフォーマンスに関する懸念点があります。

1.16.2.1. Service Mesh コントロールプレーンのパフォーマンス

Istiod は、ユーザーが作成する設定ファイルおよびシステムの現在の状態に基づいてサイドカープロキシーを設定します。Kubernetes 環境では、カスタムリソース定義 (CRD) およびデプロイメントはシステムの設定および状態を設定します。ゲートウェイや仮想サービスなどの Istio 設定オブジェクトは、ユーザーが作成する設定を提供します。プロキシーの設定を生成するために、Istiod は Kubernetes 環境およびユーザー作成の設定から、組み合わせた設定およびシステムの状態を処理します。

Service Mesh コントロールプレーンは、数千のサービスをサポートし、これらは同様の数のユーザーが作成する仮想サービスおよびその他の設定オブジェクトと共に数千の Pod 全体に分散されます。Istiod の CPU およびメモリー要件は、設定数および使用可能なシステムの状態と共にスケーリングされます。CPU の消費は、以下の要素でスケーリングします。

- デプロイメントの変更レート。
- 設定の変更レート。

- Istiod へのプロキシー数。

ただし、この部分は水平的にスケーリングが可能です。

1.16.2.2. データプレーンのパフォーマンス

データプレーンのパフォーマンスは、以下を含む数多くの要因によって変わります。

- クライアント接続の数
- ターゲットの要求レート
- 要求サイズおよび応答サイズ
- プロキシーワーカーのスレッド数
- プロトコル
- CPU コア数
- プロキシーフィルターの数およびタイプ (とくに Telemetry v2 関連のフィルター)。

レイテンシー、スループット、およびプロキシーの CPU およびメモリーの消費は、これらの要素の関数として測定されます。

1.16.2.2.1. CPU およびメモリーの消費

サイドカープロキシーはデータパスで追加の作業を実行するため、CPU およびメモリーを消費します。Istio 1.12.3 の時点で、プロキシーは 1 秒あたり 1000 要求ベースで 0.5 vCPU を消費します。

プロキシーのメモリー消費は、プロキシーが保持する設定の状態の合計数によって異なります。多数のリスナー、クラスター、およびルートは、メモリーの使用量を増やす可能性があります。

通常、プロキシーは通過するデータをバッファーに入れないため、要求レートはメモリー消費には影響を及ぼしません。

1.16.2.2.2. その他のレイテンシー

Istio はデータパスにサイドカープロキシーを挿入するため、レイテンシーは重要な考慮事項になります。Istio は認証フィルター、Telemetry フィルター、およびメタデータ交換フィルターをプロキシーに追加します。すべての追加フィルターはプロキシー内のパスの長さに追加され、これはレイテンシーに影響を及ぼします。

Envoy プロキシーは、応答がクライアントに送信された後に未加工の Telemetry データを収集します。要求についての未加工の Telemetry の収集に費やされる時間は、その要求の完了にかかる合計時間には影響を与えません。ただし、ワーカーは要求処理にビジー状態になるため、ワーカーは次の要求の処理をすぐに開始しません。このプロセスは、次の要求のキューの待機時間に追加され、平均のレイテンシーおよびテイルレイテンシー (Tail Latency) に影響を及ぼします。実際のテイルレイテンシーは、トラフィックのパターンによって異なります。

メッシュ内で、要求はクライアント側のプロキシーを通過してから、サーバー側のプロキシーを通過します。Istio 1.12.3 のデフォルト設定 (Istio と Telemetry v2) では、2 つのプロキシーは、ベースラインのデータプレーンのレイテンシーに対してそれぞれ約 1.7 ms および 2.7 ms を 90 および 99 番目のパーセンタイルレイテンシーに追加します。

1.17. 実稼働環境の SERVICE MESH の設定

基本インストールから実稼働環境に移行する準備ができたら、実稼働環境の要件を満たすようにコントロールプレーン、トレーシング、およびセキュリティー証明書を設定する必要があります。

前提条件

- Red Hat OpenShift Service Mesh をインストールして設定しておく。
- ステージング環境で設定をテストしておく

1.17.1. 実稼働環境用の ServiceMeshControlPlane リソース設定

Service Mesh をテストするために基本的な **ServiceMeshControlPlane** リソースをインストールしている場合は、実稼働環境で Red Hat OpenShift Service Mesh を使用する前にこれを実稼働仕様に設定する必要があります。

既存の **ServiceMeshControlPlane** リソースの **metadata.name** フィールドを変更できません。実稼働デプロイメントの場合は、デフォルトのテンプレートをカスタマイズする必要があります。

手順

1. 実稼働環境用の分散トレースプラットフォームを設定します。

   a. **ServiceMeshControlPlane** リソースは、**production** デプロイメントストラテジーを使用するように編集するには、**spec.addons.jaeger.install.storage.type** を **Elasticsearch** に設定し、**install** で追加設定オプションを指定します。Jaeger インスタンスを作成および設定し、**spec.addons.jaeger.name** を Jaeger インスタンスの名前に設定できます。

   デフォルト Jaeger パラメーター (例: Elasticsearch)

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.2
  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. 実稼働環境のサンプリングレートを設定します。詳細は、パフォーマンスおよびスケーラビリティーセクションを参照してください。

2. 外部認証局からセキュリティー証明書をインストールして、セキュリティー証明書が実稼働可能であることを確認します。詳細は、セキュリティーのセクションを参照してください。

3. 結果を確認します。以下のコマンドを実行して、**ServiceMeshControlPlane** リソースが適切に更新されていることを確認します。この例では、**basic** は **ServiceMeshControlPlane** リソースの名前です。

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

1.17.2. 関連情報

- パフォーマンス用のサービスメッシュのチューニングについての詳細は、Perforance and Scalability を参照してください。

1.18. サービスメッシュの接続

フェデレーション は、個別の管理ドメインで管理される個別のメッシュ間でサービスとワークロードを共有できるデプロイメントモデルです。

1.18.1. フェデレーションの概要

フェデレーションは、個別のメッシュ間でサービスを接続できるようにする機能セットであり、複数の別個の管理対象ドメイン間での認証、認可、およびトラフィック管理などの Service Mesh 機能を使用できるようにします。

フェデレーションメッシュを実装すると、複数の OpenShift クラスター全体で実行されている単一のサービスメッシュを実行、管理、および監視できます。Red Hat OpenShift Service Mesh のフェデレーションは、メッシュ間の最小限の信頼を前提とする Service Mesh のマルチクラスター実装に対して独自のアプローチを採用しています。

Service Mesh フェデレーションでは、各メッシュが個別に管理され、独自の管理者を用意することが前提です。デフォルトの動作では、通信が許可されず、メッシュ間で情報を共有されません。メッシュ間の情報、オプトインを明示することで共有されます。共有設定されていない限り、フェデレーションメッシュでは共有されません。証明書の生成、メトリクス、トレース収集などのサポート機能は、それぞれのメッシュのローカルで機能します。

各サービスメッシュで **ServiceMeshControlPlane** が、フェデレーション専用の ingress および egress ゲートウェイを作成してメッシュの信頼ドメインを指定するように設定します。

フェデレーションでは、追加でフェデレーションファイルも作成されます。以下のリソースを使用して、2 つ以上のメッシュ間のフェデレーションを設定します。

- **ServiceMeshPeer** リソースは、サービスメッシュのペア間のフェデレーションを宣言します。
- **ExportedServiceSet** リソース: メッシュからの 1 つ以上のサービスがピアメッシュで使用できることを宣言します。
- **ImportedServiceSet** リソース: ピアメッシュでエクスポートされたどのサービスがメッシュにインポートされるかを宣言します。

1.18.2. フェデレーション機能

メッシュに参加するための Red Hat OpenShift Service Mesh フェデレーションアプローチの機能は以下のとおりです。

- 各メッシュの共通ルート証明書をサポートします。
- 各メッシュの異なるルート証明書をサポートします。
- メッシュ管理者は、フェデレーションメッシュの外部にあるメッシュの証明書チェーン、サービス検出エンドポイント、信頼ドメインを手動で設定する必要があります。
- メッシュ間で共有するサービスのみをエクスポート/インポートします。
  - デフォルトは、デプロイされたワークロードに関する情報は、フェデレーション内の他のメッシュとは共有されません。サービスをエクスポートして他のメッシュに公開し、独自のメッシュ外のワークロードから要求できるようにします。
  - エクスポートされたサービスは別のメッシュにインポートでき、そのメッシュのワークロードをインポートされたサービスに送信できるようにします。
- メッシュ間の通信を常時暗号化します。
- ローカルにデプロイされたワークロードおよびフェデレーション内の別のメッシュにデプロイされたワークロードの間における負荷分散の設定をサポートします。

メッシュが別のメッシュに参加すると、以下を実行できます。

- フェデレーションメッシュに対して自信の信頼情報を提供します。
- フェデレーションメッシュについての信頼情報を検出します。
- 独自にエクスポートされたサービスに関する情報をフェデレーションメッシュに提供します。
- フェデレーションメッシュでエクスポートされるサービスの情報を検出します。

1.18.3. フェデレーションセキュリティー

Red Hat OpenShift Service Mesh のフェデレーションは、メッシュ間の最小限の信頼を前提とする Service Mesh のマルチクラスター実装に対して独自のアプローチを採用しています。データセキュリティーは、フェデレーション機能の一部として組み込まれています。

- メッシュごとに、テナントや管理が一意となっています。
- フェデレーションで各メッシュに一意の信頼ドメインを作成します。
- フェデレーションメッシュ間のトラフィックは、相互トランスポート層セキュリティー (mTLS) を使用して自動的に暗号化されます。
- Kiali グラフは、インポートしたメッシュとサービスのみを表示します。メッシュにインポートされていない他のメッシュまたはサービスを確認できません。

1.18.4. フェデレーションの制限

メッシュに参加するための Red Hat OpenShift Service Mesh フェデレーションアプローチには、以下の制限があります。

- メッシュのフェデレーションは OpenShift Dedicated ではサポートされていません。
- メッシュのフェデレーションは、Microsoft Azure Red Hat OpenShift(ARO) ではサポートされません。

1.18.5. フェデレーションの前提条件

メッシュに参加するための Red Hat OpenShift Service Mesh フェデレーションアプローチには、以下の前提条件があります。

- 2 つ以上の OpenShift Container Platform 4.6 以降のクラスター。
- フェデレーション機能は Red Hat OpenShift Service Mesh 2.1 で導入されました。フェデレーションする必要のある各メッシュに Red Hat OpenShift Service Mesh 2.1 Operator がインストールされている必要があります。
- フェデレーションする必要のある各メッシュにバージョン 2.1 の **ServiceMeshControlPlane** をデプロイする必要があります。
- 生の TLS トラフィックをサポートするように、フェデレーションゲートウェイに関連付けられたサービスをサポートするロードバランサーを設定する必要があります。フェデレーショントラフィックは、検出用の HTTPS と、サービストラフィック用の生の暗号化 TCP で設定されます。
- 別のメッシュに公開するサービスは、エクスポートおよびインポートする前にデプロイする必要があります。ただし、これは厳密な要件ではありません。エクスポート/インポート用にまだ存在していないサービス名を指定できます。**ExportedServiceSet** と **ImportedServiceSet** という名前のサービスをデプロイする場合に、これらのサービスは自動的にエクスポート/インポートで利用可能になります。

1.18.6. メッシュフェデレーションのプランニング

メッシュフェデレーションの設定を開始する前に、実装の計画に時間をかけるようにしてください。

- フェデレーションに参加させる予定のメッシュは何個ありますか ?まずは、2 つから 3 つ程度の限られた数のメッシュで開始する必要があります。
- 各メッシュにどの命名規則を使用する予定ですか ?事前定義の命名規則があると、設定とトラブルシューティングに役立ちます。本書の例では、メッシュごとに異なる色を使用します。各メッシュと以下のフェデレーションリソースの所有者および管理者を判断できるように、命名規則を決定する必要があります。
  - クラスター名
  - クラスターネットワーク名
  - メッシュ名と namespace
  - フェデレーション Ingress ゲートウェイ
  - フェデレーション egress ゲートウェイ
  - セキュリティー信頼ドメイン

注記

フェデレーションの各メッシュには、一意の信頼ドメインが必要です。

- 各メッシュのどのサービスをフェデレーションメッシュにエクスポートする予定ですか ?各サービスは個別にエクスポートすることも、ラベルを指定したり、ワイルドカードを使用したりすることもできます。

  - サービスの namespace にエイリアスを使用しますか？
  - エクスポートされたサービスにエイリアスを使用しますか？
- 各メッシュでどのエクスポートサービスを、インポートする予定ですか ?各メッシュは必要なサービスのみをインポートします。
  - インポートしたサービスにエイリアスを使用しますか？

1.18.7. クラスター全体でのメッシュフェデレーション

OpenShift Service Mesh のインスタンスを別のクラスターで実行されているインスタンスに接続するには、同じクラスターにデプロイされた 2 つのメッシュに接続する場合とほぼ変わりません。ただし、メッシュの Ingress ゲートウェイに、他のメッシュから到達可できる必要があります。確実に到達できるようにするには、クラスターがこのタイプのサービスをサポートする場合には、ゲートウェイサービスを **LoadBalancer** サービスとして設定します。

サービスは、OSI モデルのレイヤー 4 で動作するロードバランサー経由で公開する必要があります。

1.18.7.1. ベアメタルで実行されるクラスターでのフェデレーション Ingress の公開

クラスターがベアメタルで実行され、**LoadBalancer** サービスを完全にサポートする場合には、ingress ゲートウェイ **Service** オブジェクトの **.status.loadBalancer.ingress.ip** フィールドにある IP アドレスを **ServiceMeshPeer** オブジェクトの **.spec.remote.addresses** フィールドにあるエントリーの 1 つとして指定する必要があります。

クラスターが **LoadBalancer** サービスをサポートしない場合には、他のメッシュを実行するクラスターからノードにアクセスできるのであれば **NodePort** サービスを使用することも可能です。**ServiceMeshPeer** オブジェクトで、**.spec.remote.addresses** フィールドのノードの IP アドレスと、**.spec.remote.discoveryPort** と **.spec.remote.servicePort** フィールドのサービスのノードポートを指定します。

1.18.7.2. IBM Power および IBM Z で実行されているクラスターでのフェデレーション入力の公開

クラスターが IBM Power または IBM Z インフラストラクチャーで実行され、**LoadBalancer**サービスを完全にサポートする場合には、ingress ゲートウェイ **Service** オブジェクトの **.status.loadBalancer.ingress.ip** フィールドにある IP アドレスを **ServiceMeshPeer** オブジェクトの **.spec.remote.addresses** フィールドにあるエントリーの 1 つとして指定する必要があります。

クラスターが **LoadBalancer** サービスをサポートしない場合には、他のメッシュを実行するクラスターからノードにアクセスできるのであれば **NodePort** サービスを使用することも可能です。**ServiceMeshPeer** オブジェクトで、**.spec.remote.addresses** フィールドのノードの IP アドレスと、**.spec.remote.discoveryPort** と **.spec.remote.servicePort** フィールドのサービスのノードポートを指定します。

1.18.7.3. Amazon Web Services(AWS) でのフェデレーション Ingress の公開

デフォルトでは、AWS で実行されるクラスターの LoadBalancer サービスで L4 負荷分散はサポートされていません。Red Hat OpenShift Service Mesh フェデレーションを正常に機能させるには、以下のアノテーションを Ingress ゲートウェイサービスに追加する必要があります。

service.beta.kubernetes.io/aws-load-balancer-type: nlb

ingress ゲートウェイ **Service** オブジェクトの **.status.loadBalancer.ingress.hostname** フィールドにある完全修飾ドメイン名は、**ServiceMeshPeer** オブジェクトの **.spec.remote.addresses** フィールドにあるエントリーの 1 つとして指定する必要があります。

1.18.7.4. Azure でのフェデレーション Ingress の公開

Microsoft Azure では、サービスタイプを **LoadBalancer** に設定するだけで、メッシュフェデレーションが正しく動作します。

ingress ゲートウェイ **Service** オブジェクトの **.status.loadBalancer.ingress.ip** フィールドにある IP アドレスは、**ServiceMeshPeer** オブジェクトの **.spec.remote.addresses** フィールドにあるエントリーの 1 つとして指定する必要があります。

1.18.7.5. Google Cloud Platform(GCP) でのフェデレーション Ingress の公開

Google Cloud Platform では、サービスタイプを **LoadBalancer** に設定するだけで、メッシュフェデレーションが正しく動作します。

ingress ゲートウェイ **Service** オブジェクトの **.status.loadBalancer.ingress.ip** フィールドにある IP アドレスは、**ServiceMeshPeer** オブジェクトの **.spec.remote.addresses** フィールドにあるエントリーの 1 つとして指定する必要があります。

1.18.8. フェデレーション実装のチェックリスト

サービスメッシュのフェデレーションには、以下のアクティビティーが含まれます。

- ❑ フェデレーションする予定のクラスター間のネットワークを設定する。
  - ❑ 生の TLS トラフィックをサポートするように、フェデレーションゲートウェイに関連付けられたサービスをサポートするロードバランサーを設定する。
- ❑ Red Hat OpenShift Service Mesh バージョン 2.1 以降の Operator を各クラスターにインストールする。
- ❑ バージョン 2.1 以降の **ServiceMeshControlPlane** を各クラスターにデプロイする。
- ❑ フェデレーションする各メッシュのフェデレーションに SMCP を設定する。
  - ❑ フェデレーションする各メッシュにフェデレーション egress ゲートウェイを作成する。
  - ❑ フェデレーションする各メッシュにフェデレーション Ingress ゲートウェイを作成する。
  - ❑ 一意の信頼ドメインを設定する。
- ❑ 各メッシュのペアの **ServiceMeshPeer** リソースを作成して、2 つ以上のメッシュをフェデレーションする。
- ❑ **ExportedServiceSet** リソースを作成してサービスをエクスポートし、1 つのメッシュからピアメッシュにサービスが利用できるようにします。
- ❑ **ImportedServiceSet** リソースを作成してサービスをインポートし、メッシュピアで共有されるサービスをインポートします。

1.18.9. フェデレーション用の Service Mesh コントロールプレーンの設定

メッシュをフェデレーションする前に、メッシュフェデレーションの **ServiceMeshControlPlane** を設定する必要があります。フェデレーションに所属する全メッシュは同等で、各メッシュは個別に管理されるため、フェデレーションに参加する 各 メッシュに SMCP を設定する必要があります。

以下の例では、**red-mesh** の管理者は **green-mesh** と **blue-mesh** の両方を使用して、フェデレーションに SMCP を設定します。

Red-mesh のサンプル SMCP

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: red-mesh
  namespace: red-mesh-system
spec:
  version: v2.2
  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
```

表1.7 ServiceMeshControlPlane フェデレーション設定パラメーター

パラメーター	説明	値	デフォルト値

`spec:`
`  cluster:`
`  name:`
	クラスターの名前。クラスター名は指定する必要はありませんが、トラブルシューティングに役立ちます。	文字列	該当なし

`spec:`
`  cluster:`
`  network:`
	クラスターネットワークの名前。ネットワークの名前は指定する必要はありませんが、設定およびトラブルシューティングに役立ちます。	文字列	該当なし

1.18.9.1. フェデレーションゲートウェイについて

ゲートウェイ を使用してメッシュの受信トラフィックおよび送信トラフィックを管理することで、メッシュに入るか、またはメッシュを出るトラフィックを指定できます。

Ingress および egress ゲートウェイを使用して、サービスメッシュ (North-South トラフィック) に入退出するトラフィックを管理します。フェデレーションメッシュの作成時に、追加の Ingress/egress ゲートウェイを作成し、フェデレーションメッシュ間のサービス検出や通信を用意にして、サービスメッシュ間のトラフィックフロー (East-West トラフィック) を管理します。

メッシュ間の命名の競合を回避するには、各メッシュに個別の egress および ingress ゲートウェイを作成する必要があります。たとえば、**red-mesh** には、**green-mesh** および **blue-mesh** に移動するトラフィックに対して個別の egress ゲートウェイがあります。

表1.8 フェデレーションゲートウェイパラメーター

パラメーター	説明	値	デフォルト値
spec: gateways: additionalEgress: <egressName>:	フェデレーションの **各** メッシュピアの egress ゲートウェイを追加で定義します。		
spec: gateways: additionalEgress: <egressName>: enabled:	このパラメーターは、フェデレーションの egress を有効または無効にします。	**true/false**	**true**
spec: gateways: additionalEgress: <egressName>: requestedNetworkView:	エクスポートされたサービスに関連付けられたネットワーク。	メッシュの SMCP で **spec.cluster.network** の値に設定します。それ以外の場合は、<ServiceMeshPeer-name>-network を使用します。たとえば、メッシュの **ServiceMeshPeer** リソースの名前が **west** の場合には、ネットワークは **west-network** になります。	
spec: gateways: additionalEgress: <egressName>: routerMode:	ゲートウェイが使用するルーターモード。	**sni-dnat**	

パラメーター	説明	値	デフォルト値
spec: gateways: additionalEgress: <egressName>: service: metadata: labels: federation.maistra.io/egress-for:	フェデレーショントラフィックがクラスターのデフォルトのシステムゲートウェイを通過しないように、ゲートウェイに一意のラベルを指定します。		
spec: gateways: additionalEgress: <egressName>: service: ports:	TLS およびサービス検出用の **port:** と **name:** を指定するのに使用します。フェデレーショントラフィックは、サービストラフィック用の生の暗号化 TCP で設定されます。	ポート **15443** は、フェデレーションで TLS サービス要求を他のメッシュに送信するために必要です。ポート **8188** は、フェデレーションでサービス検出要求を他のメッシュに送信するために必要です。	
spec: gateways: additionalIngress:	フェデレーションで、各メッシュピアの追加の Ingress ゲートウェイを定義します。		
spec: gateways: additionalIgress: <ingressName>: enabled:	このパラメーターは、フェデレーション Ingress を有効または無効にします。	**true/false**	**true**
spec: gateways: additionalIngress: <ingressName>: routerMode:	ゲートウェイが使用するルーターモード。	**sni-dnat**	

パラメーター	説明	値	デフォルト値
spec: gateways: additionalIngress: <ingressName>: service: type:	Ingress ゲートウェイサービスは、OSI モデルのレイヤー 4 で動作し、一般公開されているロードバランサー経由で公開する必要があります。	**LoadBalancer**	
spec: gateways: additionalIngress: <ingressName>: service: type:	クラスターが **LoadBalancer** サービスをサポートしていない場合には、入力ゲートウェイサービスは **NodePort** サービスを介して公開できます。	**NodePort**	
spec: gateways: additionalIngress: <ingressName>: service: metadata: labels: federation.maistra.io/ingress-for:	フェデレーショントラフィックがクラスターのデフォルトのシステムゲートウェイを通過しないように、ゲートウェイに一意のラベルを指定します。		
spec: gateways: additionalIngress: <ingressName>: service: ports:	TLS およびサービス検出用の **port:** と **name:** を指定するのに使用します。フェデレーショントラフィックは、サービストラフィック用の生の暗号化 TCP で設定されます。フェデレーショントラフィックは、検出用に HTTPS で設定されます。	ポート **15443** は、フェデレーションの他のメッシュへの TLS サービス要求を受信するために必要です。ポート **8188** は、フェデレーションの他のメッシュへのサービス検出要求を受信するために必要です。	

パラメーター	説明	値	デフォルト値
spec: gateways: additionalIngress: <ingressName>: service: ports: nodePort:	クラスターが **LoadBalancer** サービスをサポートしていない場合には、**nodePort:** を指定するために使用されます。	指定した場合には、**port:** と **name:** 以外に、TLS とサービス検出の両方でこれが必須です。**nodePort: 30000-32767** の範囲である必要があります。	

次の例では、管理者は **NodePort** サービスを使用して グリーンメッシュ とのフェデレーション用に SMCP を設定しています。

NodePort の SMCP 例

```
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. フェデレーション信頼ドメインパラメーターについて

フェデレーションの各メッシュには、一意の信頼ドメインが必要です。この値は、ServiceMesh **Peer** リソースでメッシュフェデレーションを設定する時に使用されます。

```
kind: ServiceMeshControlPlane
metadata:
  name: red-mesh
  namespace: red-mesh-system
spec:
  security:
    trust:
      domain: red-mesh.local
```

表1.9 フェデレーションセキュリティーパラメーター

パラメーター	説明	値	デフォルト値

`spec:`
`  security:`
`  trust:`
`  domain:`
	メッシュの信頼ドメインの一意の名前を指定するために使用されます。ドメインは、フェデレーション内のすべてのメッシュで一意である必要があります。	**<mesh-name>.local**	該当なし

コンソールからの手順

以下の手順に従って、OpenShift Container Platform Web コンソールで **ServiceMeshControlPlane** を編集します。この例では、**red-mesh** をサンプルとして使用しています。

1. cluster-admin ロールが割り当てられたユーザーとして OpenShift Container Platform Web コンソールにログインします。
2. **Operators → Installed Operators** に移動します。
3. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクトを選択します。例: **red-mesh-system**
4. Red Hat OpenShift Service Mesh Operator をクリックします。
5. **Istio Service Mesh Control Plane**タブで、**ServiceMeshControlPlane** の名前 (**red-mesh** など) をクリックします。
6. **Create ServiceMeshControlPlane Details**ページで、**YAML** をクリックして設定を変更します。
7. **ServiceMeshControlPlane** を変更してフェデレーション Ingress および egress ゲートウェイを追加し、信頼ドメインを指定します。
8. **Save** をクリックします。

CLI からの手順

以下の手順に従って、コマンドラインで **ServiceMeshControlPlane** を作成するか、または編集します。この例では、**red-mesh** をサンプルとして使用しています。

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。以下のコマンドを入力します。次に、プロンプトが表示されたら、ユーザー名とパスワードを入力します。

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

2. Service Mesh コントロールプレーンをインストールしたプロジェクト (例: red-mesh-system) に切り替えます。

```
$ oc project red-mesh-system
```

3. **ServiceMeshControlPlane** ファイルを編集し、フェデレーション Ingress および egress ゲートウェイを追加して信頼ドメインを指定します。

4. 以下のコマンドを実行して Service Mesh コントロールプレーンを編集します。ここで、**red-mesh-system** はシステムの namespace であり、**red-mesh** は **ServiceMeshControlPlane** オブジェクトの名前になります。

```
$ oc edit -n red-mesh-system smcp red-mesh
```

5. 以下のコマンドを実行して、Service Mesh コントロールプレーンのインストールのステータスを確認します。このコマンドでは、**red-mesh-system** がシステム namespace に置き換えます。

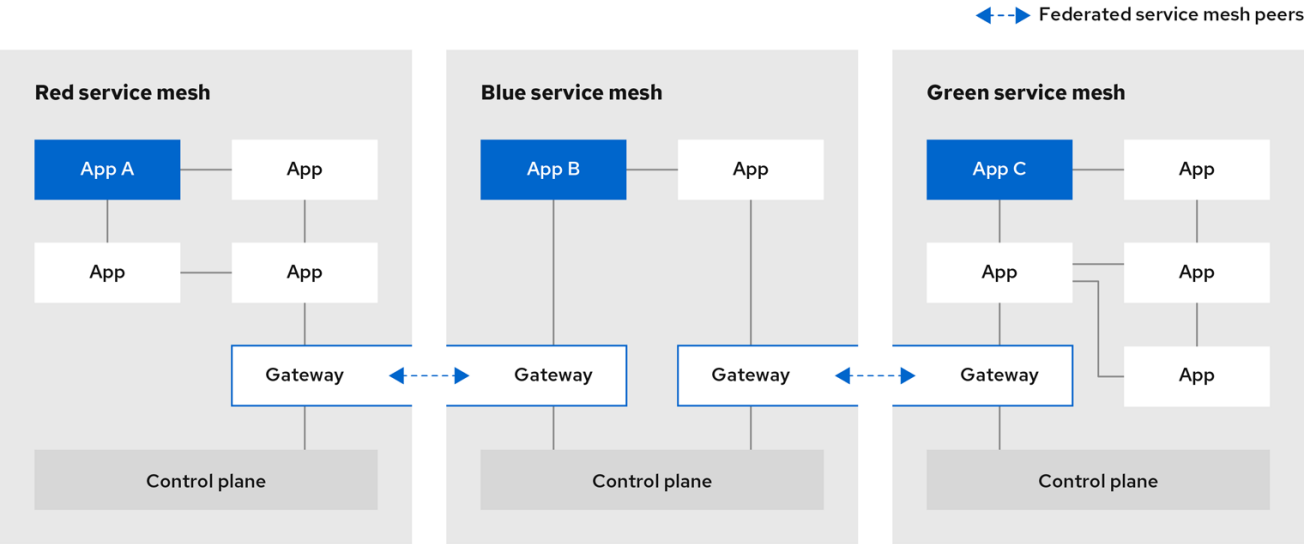
```
$ oc get smcp -n red-mesh-system
```

READY 列にすべてのコンポーネントが準備状態であることが示されると、インストールは正常に終了しています。

```
NAME      READY   STATUS            PROFILES      VERSION   AGE
red-mesh   10/10   ComponentsReady   ["default"]   2.1.0     4m25s
```

1.18.10. フェデレーションメッシュへの参加

ServiceMeshPeer リソースを作成して、2 つのメッシュ間のフェデレーションを宣言します。**ServiceMeshPeer** リソースは、2 つのメッシュ間のフェデレーションを定義し、これを使用してピアメッシュの検出設定、ピアメッシュへのアクセス、および他のメッシュのクライアントの検証に使用される証明書を定義します。



メッシュは 1 対 1 でフェデレーションされるため、ピアの各ペアでは、他のサービスメッシュへのフェデレーション接続を指定する **ServiceMeshPeer** リソースのペアが必要です。たとえば、**red** および **green** という名前の 2 つのメッシュには 2 つの **ServiceMeshPeer** ファイルが必要です。

1. Red-mesh-system で、Green メッシュの **ServiceMeshPeer** を作成します。
2. Green-mesh-system で、Red メッシュの **ServiceMeshPeer** を作成します。

red、**blue** および **green** という名前の 3 つのメッシュのフェデレーションには 6 つの **ServiceMeshPeer** ファイルが必要になります。

1. Red-mesh-system で、Green メッシュの **ServiceMeshPeer** を作成します。
2. Red-mesh-system で、Blue メッシュの **ServiceMeshPeer** を作成します。
3. Green-mesh-system で、Red メッシュの **ServiceMeshPeer** を作成します。
4. Green-mesh-system で、Blue メッシュの **ServiceMeshPeer** を作成します。
5. Blue-mesh-system で、Red メッシュの **ServiceMeshPeer** を作成します。
6. Blue-mesh-system で、Green メッシュの **ServiceMeshPeer** を作成します。

ServiceMeshPeer リソースの設定には、以下が含まれます。

- 検出およびサービス要求に使用される他のメッシュの Ingress ゲートウェイのアドレス。
- 指定のピアメッシュとの対話に使用されるローカル ingress および egress ゲートウェイの名前。
- このメッシュへの要求の送信時に他のメッシュで使用されるクライアント ID。
- 他のメッシュで使用される信頼ドメイン。
- **ConfigMap** の名前。これには、他のメッシュで使用される信頼ドメインのクライアント証明書の検証に使用するルート証明書が含まれます 。

以下の例では、**red-mesh** の管理者は **green-mesh** でフェデレーションを設定します。

Red-mesh の ServiceMeshPeer リソースの例

```
kind: ServiceMeshPeer
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  remote:
    addresses:
    - ingress-red-mesh.green-mesh-system.apps.domain.com
  gateways:
    ingress:
      name: ingress-green-mesh
    egress:
      name: egress-green-mesh
  security:
    trustDomain: green-mesh.local
    clientID: green-mesh.local/ns/green-mesh-system/sa/egress-red-mesh-service-account
    certificateChain:
      kind: ConfigMap
      name: green-mesh-ca-root-cert
```

表1.10 ServiceMeshPeer 設定パラメーター

パラメーター	説明	値
metadata: name:	このリソースがフェデレーションを設定するピアメッシュの名前。	文字列
metadata: namespace:	このメッシュのシステム namespace (Service Mesh コントロールプレーンのインストール先)。	文字列
spec: remote: addresses:	このメッシュからの要求に対応するピアメッシュの Ingress ゲートウェイのパブリックアドレス一覧。	
spec: remote: discoveryPort:	アドレスが検出要求を処理するポート。	デフォルトは 8188 です。
spec: remote: servicePort:	アドレスがサービス要求を処理するポート。	デフォルトは 15443 です。
spec: gateways: ingress: name:	ピアメッシュからの受信要求に対応するこのメッシュの Ingress の名前。例: **ingress-green-mesh**	
spec: gateways: egress: name:	ピアメッシュに送信される要求に対応するこのメッシュ上の egress の名前。例: **egress-green-mesh**	
spec: security: trustDomain:	ピアメッシュで使用される信頼ドメイン。	<peerMeshName>.local
spec: security: clientID:	このメッシュの呼び出し時にピアメッシュが使用するクライアント ID。	<peerMeshTrustDomain>/ns/<peerMeshSystem>/sa/<peerMeshEgressGatewayName>-service-account

パラメーター	説明	値
spec: security: certificateChain: kind: ConfigMap name:	ピアメッシュがこのメッシュに提示したクライアント証明書の検証に使用されるルート証明書が含まれるリソースの種類 (例: ConfigMap) と名前。証明書が含まれる Config Map エントリーの鍵は **root-cert.pem** である必要があります。	kind: ConfigMap name: <peerMesh>-ca-root-cert

1.18.10.1. ServiceMeshPeer リソースの作成

前提条件

- 2 つ以上の OpenShift Container Platform 4.6 以降のクラスター。
- クラスターのネットワーク設定が完了している。
- 生の TLS トラフィックをサポートするように、フェデレーションゲートウェイに関連付けられたサービスをサポートするロードバランサーを設定する必要があります。
- 各クラスターには、フェデレーションデプロイをサポートするようにバージョン 2.1 **ServiceMeshControlPlane** が設定されている必要があります。
- **cluster-admin** ロールを持つアカウントがある。

CLI からの手順

以下の手順に従って、コマンドラインから **ServiceMeshPeer** リソースを作成します。以下の例では、**red-mesh** が **green-mesh** のピアリソースを作成しています。

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。以下のコマンドを入力します。次に、プロンプトが表示されたら、ユーザー名とパスワードを入力します。

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

2. コントロールプレーンをインストールしたプロジェクト (例: **red-mesh-system**) に切り替えます。

```
$ oc project red-mesh-system
```

3. フェデレーションする 2 つのメッシュについて以下の例をもとに、**ServiceMeshPeer** ファイルを作成します。

Red-mesh から green-mesh への ServiceMeshPeer リソースのサンプル

```
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. 以下のコマンドを実行してリソースをデプロイします。ここで、**red-mesh-system** はシステムの namespace に置き換え、**servicemeshpeer.yaml** には編集したファイルへのフルパスが含まれます。

```
$ oc create -n red-mesh-system -f servicemeshpeer.yaml
```

5. red メッシュと green メッシュ間の接続確立を確認するには、red-mesh-system namespace の green-mesh **ServiceMeshPeer** のステータスを調べます。

```
$ oc -n red-mesh-system get servicemeshpeer green-mesh -o yaml
```

Red-mesh と green-mesh 間の ServiceMeshPeer 接続の例

```
status:
  discoveryStatus:
    active:
    - pod: istiod-red-mesh-b65457658-9wq5j
      remotes:
      - connected: true
        lastConnected: "2021-10-05T13:02:25Z"
        lastFullSync: "2021-10-05T13:02:25Z"
        source: 10.128.2.149
      watch:
        connected: true
        lastConnected: "2021-10-05T13:02:55Z"
        lastDisconnectStatus: 503 Service Unavailable
        lastFullSync: "2021-10-05T13:05:43Z"
```

status.discoveryStatus.active.remotes フィールドは、ピアメッシュ (この例では Green メッシュ) が現在のメッシュ (この例では赤のメッシュ) の istiod に接続されていることを示します。

status.discoveryStatus.active.watch フィールドは、現在のメッシュの istiod がピアメッシュで istiod に接続されていることを示します。

green-mesh-system で **red-mesh** という名前の **servicemeshpeer** を確認すると、Green メッシュの観点からの 2 つの同じ接続に関する情報が表示されます。

2 つのメッシュ間の接続が確立されていない場合には、**ServiceMeshPeer** ステータスは、**status.discoveryStatus.inactive** フィールドにこれを示します。

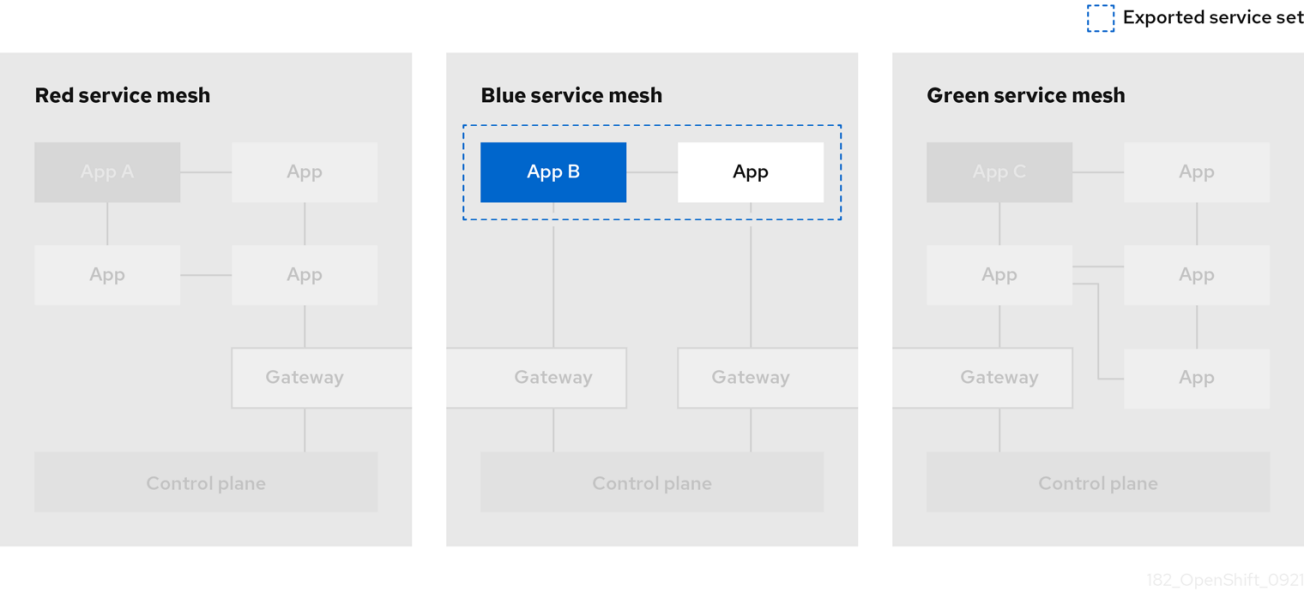
接続に失敗する理由については、Istiod ログ、ピアの egress トラフィックを処理する egress ゲートウェイのアクセスログ、およびピアメッシュの現在のメッシュの ingress トラフィックを処理する ingress ゲートウェイを調べてください。

たとえば、red メッシュが green メッシュに接続できない場合は、以下のログを確認します。

- red-mesh-system の istiod-red-mesh
- red-mesh-system の Egress-green-mesh
- green-mesh-system の ingress-red-mesh

1.18.11. フェデレーションメッシュからのサービスのエクスポート

サービスをエクスポートすると、メッシュは、フェデレーションされたメッシュの別のメンバーとサービスを共有できます。



ExportedServiceSet リソースを使用して、フェデレーションメッシュ内の別のピアから利用できるように指定したメッシュからサービスを宣言します。ピアと共有される各サービスを明示的に宣言する必要があります。

- サービスは、namespace または名前別に選択できます。
- ワイルドカードを使用してサービスを選択できます。たとえば、namespace 内のすべてのサービスをエクスポートします。
- エイリアスを使用してサービスをエクスポートできます。たとえば、**foo/bar** サービスを **custom-ns/bar** としてエクスポートできます。
- メッシュのシステム namespace に表示されるサービスのみをエクスポートできます。たとえば、**networking.istio.io/exportTo** ラベルが '.' に設定された別の namespace のサービスは、エクスポートの候補にはなりません。

- エクスポートされたサービスの場合には、それらのターゲットサービスは、元の要求元 (他のメッシュの egress ゲートウェイや、要求元のワークロードのクライアント ID は表示されない) ではなく、ingress ゲートウェイからのトラフィックのみが表示されます。

以下の例は、 **red-mesh** が **green-mesh** にエクスポートするサービス向けです。

ExportedServiceSet リソースの例

```
kind: ExportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  exportRules:
  # export ratings.mesh-x-bookinfo as ratings.bookinfo
  - type: NameSelector
    nameSelector:
      namespace: red-mesh-bookinfo
      name: red-ratings
      alias:
        namespace: bookinfo
        name: ratings
  # export any service in red-mesh-bookinfo namespace with label export-service=true
  - type: LabelSelector
    labelSelector:
      namespace: red-mesh-bookinfo
      selector:
        matchLabels:
          export-service: "true"
      aliases: # export all matching services as if they were in the bookinfo namespace
      - namespace: "*"
        name: "*"
        alias:
          namespace: bookinfo
```

表1.11 ExportedServiceSet パラメーター

パラメーター	説明	値
`metadata:` `  name:`	このサービスを公開する ServiceMeshPeer の名前。	**ServiceMeshPeer** リソースのメッシュの **name** 値と一致する必要があります。
`metadata:` `  namespace:`	このリソースを含むプロジェクト/namespace の名前 (メッシュのシステム namespace を指定する必要があります)。	

`spec:`
`  exportRules:`
`  - type:`
	このサービスのエクスポートを管理するルールのタイプ。サービスで最初に一致するルールがエクスポートに使用されます。	**NameSelector**, **LabelSelector**

パラメーター	説明	値
spec: exportRules: - type: NameSelector nameSelector: namespace: name:	**NameSelector** ルールを作成するには、サービスの **namespace** および サービスの **namespace** を **Service** リソースで定義されるとおり指定します。	
spec: exportRules: - type: NameSelector nameSelector: alias: namespace: name:	サービスの **namespace** と **name** を指定した後に、**namespace** のエイリアスと、サービスの **name** として使用するエイリアスを指定し、このサービスのエイリアスに使用する **NameSelector** ルールを作成します。	
spec: exportRules: - type: LabelSelector labelSelector: namespace: <exportingMesh> selector: matchLabels: <labelKey>: <labelValue>	**LabelSelector** ルールを作成するには、サービスの **namespace** を指定し、**Service** リソースで定義されている ラベル を指定します。上記の例では、ラベルは **export-service** です。	
spec: exportRules: - type: LabelSelector labelSelector: namespace: <exportingMesh> selector: matchLabels: <labelKey>: <labelValue> aliases: - namespace: name: alias: namespace: name:	サービスのエイリアスを使用する **LabelSelector** ルールを作成するには、**selector** を指定した後で、サービスの **name** または **namespace** に使用するエイリアスを指定します。上記の例では、一致するすべてのサービスの namespace エイリアスは **bookinfo** です。	

Red-mesh のすべての namespace から blue-mesh へ ratings という名前のサービスをエクスポートします。

```
kind: ExportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: blue-mesh
  namespace: red-mesh-system
spec:
  exportRules:
  - type: NameSelector
    nameSelector:
      namespace: "*"
      name: ratings
```

すべてのサービスを west-data-center namespace から green-mesh にエクスポートします。

```
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. ExportedServiceSet の作成

ExportedServiceSet リソースを作成し、メッシュピアで利用可能なサービスを明示的に宣言します。

サービスは **<export-name>.<export-namespace>.svc.<ServiceMeshPeer.name>-exports.local** としてエクスポートされ、ターゲットサービスに自動的にルーティングされます。これは、エクスポートメッシュでエクスポートされたサービスで認識される名前です。Ingress ゲートウェイが宛先がこの名前の要求を受信すると、エクスポートされる実際のサービスにルーティングされます。たとえば、**ratings.red-mesh-bookinfo** という名前のサービスが **ratings.bookinfo** として **green-mesh** にエクスポートされると、 サービスは **ratings.bookinfo.svc.green-mesh-exports.local** という名前でエクスポートされ、そのホスト名の ingress ゲートウェイが受信するトラフィックは **ratings.red-mesh-bookinfo** サービスにルーティングされます。

前提条件

- クラスターおよび **ServiceMeshControlPlane** がメッシュフェデレーション用に設定されている。
- **cluster-admin** ロールを持つアカウントがある。

注記

サービスがまだ存在していない場合でも、エクスポート用にサービスを設定できます。ExportedServiceSet で指定された値に一致するサービスがデプロイされ、自動的にエクスポートされます。

CLI からの手順

以下の手順に従って、コマンドラインから **ExportedServiceSet** を作成します。

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。以下のコマンドを入力します。次に、プロンプトが表示されたら、ユーザー名とパスワードを入力します。

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

2. Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **red-mesh-system**) に切り替えます。

```
$ oc project red-mesh-system
```

3. 以下の例に基づいて、**ExportedServiceSet** ファイルを作成します。ここでは、**red-mesh** がサービスを **green-mesh** にエクスポートします。

red-mesh から green-mesh への ExportedServiceSet リソースの例

```
apiVersion: federation.maistra.io/v1
kind: ExportedServiceSet
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  exportRules:
  - type: NameSelector
    nameSelector:
      namespace: red-mesh-bookinfo
      name: ratings
      alias:
        namespace: bookinfo
        name: red-ratings
  - type: NameSelector
    nameSelector:
      namespace: red-mesh-bookinfo
      name: reviews
```

4. 以下のコマンドを実行して、red-mesh-system namespace に **ExportedServiceSet** リソースをアップロードおよび作成します。

```
$ oc create -n <ControlPlaneNamespace> -f <ExportedServiceSet.yaml>
```

以下に例を示します。

```
$ oc create -n red-mesh-system -f export-to-green-mesh.yaml
```

5. フェデレーションメッシュのメッシュピアごとに、必要に応じて追加の **ExportedServiceSets** を作成します。

6. **red-mesh** からエクスポートして **green-mesh** に共有したサービスを検証するには、以下のコマンドを実行します。

```
$ oc get exportedserviceset <PeerMeshExportedTo> -o yaml
```

以下に例を示します。

```
$ oc get exportedserviceset green-mesh -o yaml
```

7. 以下のコマンドを実行して、red-mesh が green-mesh と共有するためにエクスポートしたサービスを検証します。

```
$ oc get exportedserviceset <PeerMeshExportedTo> -o yaml
```

以下に例を示します。

```
$ oc -n red-mesh-system get exportedserviceset green-mesh -o yaml
```

red メッシュからエクスポートして Green メッシュに共有したサービスの検証例。

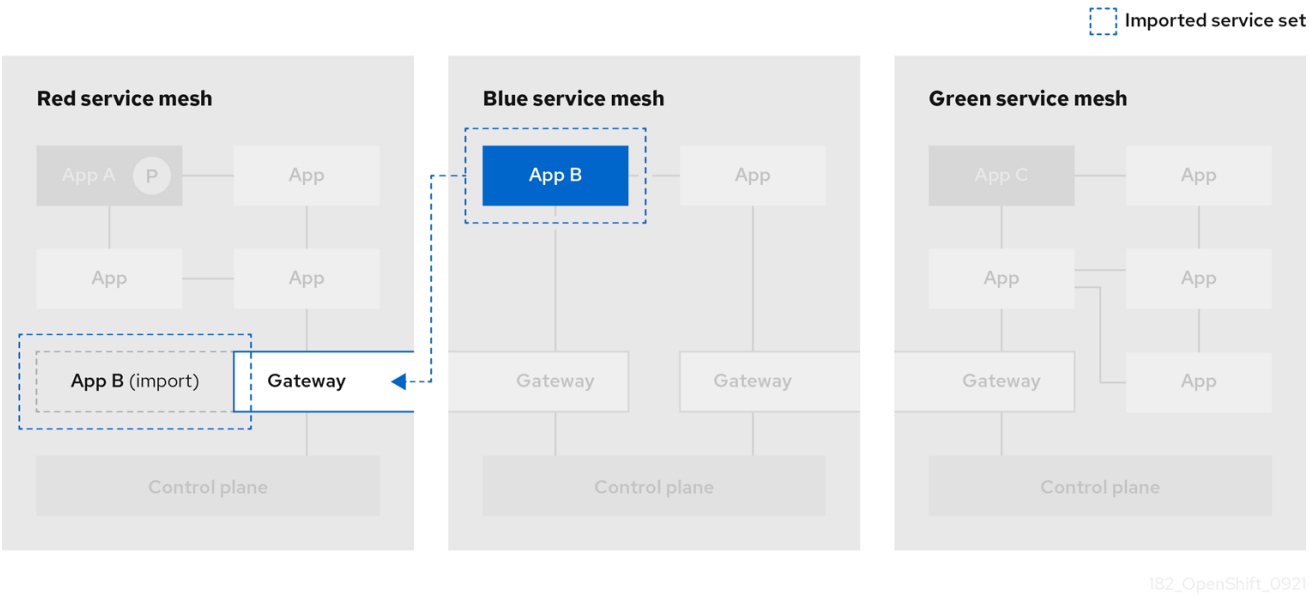
```
status:
  exportedServices:
  - exportedName: red-ratings.bookinfo.svc.green-mesh-exports.local
    localService:
      hostname: ratings.red-mesh-bookinfo.svc.cluster.local
      name: ratings
      namespace: red-mesh-bookinfo
  - exportedName: reviews.red-mesh-bookinfo.svc.green-mesh-exports.local
    localService:
      hostname: reviews.red-mesh-bookinfo.svc.cluster.local
      name: reviews
      namespace: red-mesh-bookinfo
```

status.exportedServices 配列では、現在エクスポートされているサービス (これらのサービスは **ExportedServiceSet** オブジェクト のエクスポートルールに一致) を一覧表示します。配列の各エントリーは、エクスポートされたサービスの名前と、エクスポートされたローカルサービスの詳細を示します。

エクスポート予定のサービスがない場合には、Service オブジェクトが存在すること、その名前またはラベルが **ExportedServiceSet** オブジェクトで定義される **exportRules** と一致すること、および Service オブジェクトの namespace が **ServiceMeshMemberRoll** または **ServiceMeshMember** オブジェクトを使用してサービスメッシュのメンバーとして設定されることを確認します。

1.18.12. サービスのフェデレーションメッシュへのインポート

サービスをインポートすると、別のメッシュからエクスポートされたサービスの内、サービスメッシュ内でアクセスできるものを明示的に指定できます。



ImportedServiceSet リソースを使用して、インポートするサービスを選択します。メッシュピアによってエクスポートされ、明示的にインポートされるサービスのみがメッシュで利用できます。明示的にインポートしない場合には、サービスは、メッシュ内で利用できません。

- サービスは、namespace または名前別に選択できます。
- namespace にエクスポートされたすべてのサービスをインポートするなど、ワイルドカードを使用してサービスを選択できます。
- メッシュにグローバルであるか、特定のメンバーの namespace の範囲内にあるラベルセレクターを使用してエクスポートするサービスを選択できます。
- エイリアスを使用してサービスをインポートできます。たとえば、**custom-ns/bar** サービスを **other-mesh/bar** としてインポートできます。
- カスタムドメイン接尾辞を指定できます。これは、**bar.other-mesh.imported.local** など、インポートされたサービスの **name.namespace** に、完全修飾ドメイン名として追加されます。

以下の例は、**red-mesh** でエクスポートされたサービスをインポートする **green-mesh** の例です。

ImportedServiceSet の例

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh #name of mesh that exported the service
  namespace: green-mesh-system #mesh namespace that service is being imported into
spec:
  importRules: # first matching rule is used
  # import ratings.bookinfo as ratings.bookinfo
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: bookinfo
      name: ratings
      alias:
```

```
    # service will be imported as ratings.bookinfo.svc.red-mesh-imports.local
    namespace: bookinfo
    name: ratings
```

表1.12 ImportedServiceSet パラメーター

パラメーター	説明	値
`metadata:` `  name:`	サービスをフェデレーションメッシュにエクスポートした ServiceMeshPeer の名前。	
`metadata:` `  namespace:`	ServiceMeshPeer リソース (メッシュシステム namespace) を含む namespace の名前。	

`spec:`
`  importRules:`
`  - type:`
	サービスのインポートを管理するルールのタイプ。サービスで最初に一致するルールがインポートに使用されます。	**NameSelector**

`spec:`
`  importRules:`
`  - type: NameSelector`
`  nameSelector:`
`  namespace:`
`  name:`
	NameSelector ルールを作成するには、**namespace** およびエクスポートされたサービスの **name** を指定します。	

`spec:`
`  importRules:`
`  - type: NameSelector`
`  importAsLocal:`
	リモートエンドポイントをローカルサービスで集約するには、**true** に設定します。**true** の場合は、サービスは **<name>.<namespace>.svc.cluster.local** としてインポートされます。	**true/false**

`spec:`
`  importRules:`
`  - type: NameSelector`
`  nameSelector:`
`  namespace:`
`  name:`
`  alias:`
`  namespace:`
`  name:`
	サービスの **namespace** と **name** を指定した後に、**namespace** のエイリアスと、サービスの **name** として使用するエイリアスを指定し、このサービスのエイリアスに使用する **NameSelector** ルールを作成します。	

red-mesh から blue-mesh への "bookinfo/ratings" サービスのインポート

```
kind: ImportedServiceSet
```

```
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh
  namespace: blue-mesh-system
spec:
  importRules:
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: bookinfo
      name: ratings
```

Red-mesh の west-data-center namespace からすべてのサービスを green-mesh にインポートします。これらのサービスは、<name>.west-data-center.svc.red-mesh-imports.local としてアクセスできます。

```
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. ImportedServiceSet の作成

ImportedServiceSet リソースを作成し、メッシュにインポートするサービスを明示的に宣言します。

サービスは、**<exported-name>.<exported-namespace>.svc.<ServiceMeshPeer.name>.remote** という名前でインポートされます。これは非表示のサービスで、egress ゲートウェイ namespace にのみ表示され、エクスポートされたサービスのホスト名に関連付けられます。このサービスは、ローカルから **<export-name>.<export-namespace>.<domainSuffix>** として利用可能になります。ここでは、**importAsLocal** が **true** に設定されていない限り、**domainSuffix** はデフォルトで **svc.<ServiceMeshPeer.name>-imports.local** となっています。True の場合には、**domainSuffix** は **svc.cluster.local** となります。**ImportAsLocal** が **false** に設定されている場合には、インポートルールのドメイン接尾辞が適用されます。ローカルインポートは、メッシュ内の他のサービスと同様に扱うことができます。これは egress ゲートウェイを介して自動的にルーティングされ、エクスポートされたサービスのリモート名にリダイレクトされます。

前提条件

- クラスターおよび **ServiceMeshControlPlane** がメッシュフェデレーション用に設定されている。
- **cluster-admin** ロールを持つアカウントがある。

注記

サービスがまだエクスポートされていない場合でも、インポートするように設定できます。ImportedServiceSet で指定された値に一致するサービスがデプロイされてエクスポートされると、これは自動的にインポートされます。

CLI からの手順

この手順に従って、コマンドラインから **ImportedServiceSet** を作成します。

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。以下のコマンドを入力します。次に、プロンプトが表示されたら、ユーザー名とパスワードを入力します。

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

2. Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **green-mesh-system**) に切り替えます。

```
$ oc project green-mesh-system
```

3. 以下の例に基づいて **ImportedServiceSet** ファイルを作成します。ここでは、**green-mesh** が、**red-mesh** によって以前にエクスポートされたサービスをインポートします。

red-mesh から green-mesh への ImportedServiceSet リソースの例

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh
  namespace: green-mesh-system
spec:
  importRules:
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: bookinfo
      name: red-ratings
      alias:
        namespace: bookinfo
        name: ratings
```

4. 以下のコマンドを実行して、green-mesh-system namespace に **ImportedServiceSet** リソースをアップロードおよび作成します。

```
$ oc create -n <ControlPlaneNamespace> -f <ImportedServiceSet.yaml>
```

以下に例を示します。

```
$ oc create -n green-mesh-system -f import-from-red-mesh.yaml
```

5. フェデレーションメッシュ内のメッシュピアごとに、必要に応じて追加の **ImportedServiceSet** リソースを作成します。

6. **green-mesh** にインポートしたサービスを検証するには、以下のコマンドを実行します。

```
$ oc get importedserviceset <PeerMeshImportedInto> -o yaml
```

以下に例を示します。

```
$ oc get importedserviceset green-mesh -o yaml
```

7. 以下のコマンドを実行して、メッシュにインポートされたサービスを検証します。

```
$ oc get importedserviceset <PeerMeshImportedInto> -o yaml
```

red mesh からエクスポートされたサービスが 'green-mesh-system namespace の importedserviceset/red-mesh' オブジェクト のステータスセクションを使用して、green メッシュにインポートされていることを検証する例:

```
$ oc -n green-mesh-system get importedserviceset/red-mesh -o yaml
```

```
status:
  importedServices:
  - exportedName: red-ratings.bookinfo.svc.green-mesh-exports.local
    localService:
      hostname: ratings.bookinfo.svc.red-mesh-imports.local
      name: ratings
      namespace: bookinfo
  - exportedName: reviews.red-mesh-bookinfo.svc.green-mesh-exports.local
    localService:
      hostname: ""
      name: ""
      namespace: ""
```

上記の例では、**localService** の入力済みフィールドで示されているように、ratings サービスのみがインポートされます。Reviews サービスはインポートできますが、**ImportedServiceSet** オブジェクトの **importRules** と一致しないため、現時点ではインポートされません。

1.18.13. フェイルオーバー用のフェデレーションメッシュの設定

フェイルオーバーとは、別のサーバーなどの信頼性の高いバックアップシステムに自動的かつシームレスに切り替える機能です。フェデレーションメッシュの場合、あるメッシュのサービスを設定して、別のメッシュのサービスにフェイルオーバーすることができます。

ImportedServiceSet リソースで **importAsLocal** と **locality** の設定を指定してから、**ImportedServiceSet** で指定されたローカリティーへのサービスのフェイルオーバーを設定する **DestinationRule** を指定することにより、フェイルオーバーのフェデレーションを設定します。

前提条件

- 2 つ以上の OpenShift Container Platform 4.6 以降のクラスターがすでにネットワーク化およびフェデレーションされている。
- フェデレーションメッシュのメッシュピアごとにすでに作成されている **ExportedServiceSet** リソース。

- フェデレーションメッシュのメッシュピアごとにすでに作成されている **ImportedServiceSet** リソース。
- **cluster-admin** ロールを持つアカウントがある。

1.18.13.1. フェイルオーバー用の ImportedServiceSet の設定

ローカリティ加重負荷分散を使用すると、管理者は、トラフィックの発信元と終了場所のローカリティに基づいて、エンドポイントへのトラフィックの分散を制御できます。これらのローカリティは、{region}/{zone}/{sub-zone} 形式でローカリティの階層を指定する任意のラベルを使用して指定します。

このセクションの例では、**green-mesh**は米国**us-east**地域にあり、**us-east**は**us-east**地域にあります。

red-mesh から green-mesh への ImportedServiceSet リソースの例

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh #name of mesh that exported the service
  namespace: green-mesh-system #mesh namespace that service is being imported into
spec:
  importRules: # first matching rule is used
  # import ratings.bookinfo as ratings.bookinfo
  - type: NameSelector
    importAsLocal: true
    nameSelector:
      namespace: bookinfo
      name: ratings
      alias:
        # service will be imported as ratings.bookinfo.svc.red-mesh-imports.local
        namespace: bookinfo
        name: ratings
  #Locality within which imported services should be associated.
  locality:
    region: us-west
```

表1.13 **ImportedServiceLocality** フィールドテーブル

名前	説明	タイプ
region:	インポートされたサービスが配置されている地域。	文字列
サブゾーン:	インポートされたサービスが配置されているサブゾーン。サブゾーンが指定されている場合は、ゾーンも指定する必要があります。	文字列
zone:	インポートされたサービスが配置されているゾーン。ゾーンを指定する場合は、リージョンも指定する必要があります。	文字列

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインし、次のコマンドを入力します。

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

2. Service Mesh コントロールプレーンをインストールしたプロジェクトに変更し、次のコマンドを入力します。

```
$ oc project <smcp-system>
```

たとえば、**green-mesh-system** です。

```
$ oc project green-mesh-system
```

3. **ImportedServiceSet** ファイルを編集します。ここで、**<ImportedServiceSet.yaml>** には、編集するファイルへのフルパスが含まれています。以下のコマンドを入力してください。

```
$ oc edit -n <smcp-system> -f <ImportedServiceSet.yaml>
```

たとえば、前の **ImportedServiceSet** の例で示したように、red-mesh-system から green-mesh-system にインポートするファイルを変更する場合は、以下のようになります。

```
$ oc edit -n green-mesh-system -f import-from-red-mesh.yaml
```

4. ファイルを変更します。

   a. **spec.importRules.importAsLocal**を**true**に設定します。

   b. **spec.locality** を **region** 、**zone**、または **subzone** に設定します。

   c. 変更を保存します。

1.18.13.2. フェイルオーバー用の DestinationRule の設定

以下を設定する **DestinationRule** リソースを作成します。

- サービスの外れ値の検出。これは、フェイルオーバーを正しく機能させるには必要です。特に、サービスのエンドポイントが異常である場合には把握できるようにサイドカープロキシーを設定し、最終的に次のローカリティへのフェイルオーバーをトリガーします。
- リージョン間のフェイルオーバーポリシー。これにより、リージョンの境界を超えたフェイルオーバーが予測どおりに動作することが保証されます。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。以下のコマンドを入力します。次に、プロンプトが表示されたら、ユーザー名とパスワードを入力します。

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

2. Service Mesh コントロールプレーンをインストールしたプロジェクトに変更します。

```
$ oc project <smcp-system>
```

たとえば、**green-mesh-system** です。

```
$ oc project green-mesh-system
```

3. 次の例に基づいて **DestinationRule** ファイルを作成します。ここで、green-mesh が使用できない場合、トラフィックは **us-east** リージョンの green-mesh から **us-west** の red-mesh にルーティングされます。

   DestinationRule の例

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: default-failover
  namespace: bookinfo
spec:
  host: "ratings.bookinfo.svc.cluster.local"
  trafficPolicy:
    loadBalancer:
      localityLbSetting:
        enabled: true
        failover:
          - from: us-east
            to: us-west
    outlierDetection:
      consecutive5xxErrors: 3
      interval: 10s
      baseEjectionTime: 1m
```

4. **DestinationRule** をデプロイします。次のコマンドを入力します。**<DestinationRule>** にはファイルへのフルパスを追加します。

```
$ oc create -n <application namespace> -f <DestinationRule.yaml>
```

以下に例を示します。

```
$ oc create -n bookinfo -f green-mesh-us-west-DestinationRule.yaml
```

1.18.14. フェデレーションメッシュからのサービスの削除

フェデレーションメッシュからサービスを削除する必要がある場合 (非推奨になるか、または別のサービスで置き換えられる場合など)、削除できます。

1.18.14.1. 単一のメッシュからサービスを削除するには、以下を実行します。

サービスにアクセスすべきでないメッシュピアの **ImportedServiceSet** リソースからサービスのエントリーを削除します。

1.18.14.2. フェデレーションメッシュ全体からサービスを削除するには、以下を実行します。

サービスを所有するメッシュの **ExportedServiceSet** リソースからサービスのエントリーを削除します。

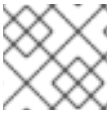
1.18.15. フェデレーションメッシュからのメッシュの削除

フェデレーションからメッシュを削除する必要がある場合は、削除できます。

1. 削除されたメッシュの **ServiceMeshControlPlane** リソースを編集して、ピアメッシュのすべてのフェデレーション Ingress ゲートウェイを削除します。
2. 削除されたメッシュがフェデレーションされているメッシュピアごとに、以下を実行します。
   a. 2 つのメッシュをリンクする **ServiceMeshPeer** リソースを削除します。
   b. ピアメッシュの **ServiceMeshControlPlane** リソースを編集して、削除されたメッシュに対応する egress ゲートウェイを削除します。

1.19. 拡張

WebAssembly 拡張機能を使用して、Red Hat Service Mesh プロキシーに直接新しい機能を追加できます。これにより、お使いのアプリケーションから、さらに一般的な機能を移動して、単一の言語で実装して、WebAssembly bytecode にコンパイルできます。



注記

WebAssembly 拡張機能は、IBM Z および IBM Power ではサポートされていません。

1.19.1. WebAssembly モジュールの概要

WebAssembly モジュールは、プロキシーなどの多くのプラットフォームで実行でき、これには、幅広い言語サポート、高速実行、および sandboxed-by-default (デフォルトでサンドボックス化される) セキュリティーモデルが含まれます。

Red Hat OpenShift Service Mesh 拡張機能として Envoy HTTP フィルター を使用でき、幅広い機能を提供します。

- 要求と応答の本体とヘッダーの操作
- 認証やポリシーのチェックなど、要求パスにないサービスへの帯域外 HTTP 要求
- 相互に通信するフィルター用のサイドチャネルデータストレージおよびキュー

注記

新しい WebAssembly 拡張機能を作成するときは、WasmPluginAPI を使用してください。ServiceMeshExtension API は、Red Hat OpenShift Service Mesh バージョン 2.2 で非推奨になり、今後のリリースで削除される予定です。

Red Hat OpenShift Service Mesh 拡張機能の作成には 2 つの部分があります。

1. proxy-wasm API を公開する SDK を使用して拡張機能を記述し、それを WebAssembly モジュールにコンパイルする必要があります。
2. 次に、モジュールをコンテナーにパッケージ化する必要があります。

サポートされる言語

WebAssembly バイトコードにコンパイルする言語を使用して Red Hat OpenShift Service Mesh 拡張を作成できますが、以下の言語には proxy-wasm API を公開する既存の SDK があるため、これを直接使用できます。

表1.14 サポートされる言語

言語	保守管理者	リポジトリー
AssemblyScript	solo.io	solo-io/proxy-runtime
C++	proxy-wasm チーム (Istio コミュニティー)	proxy-wasm/proxy-wasm-cpp-sdk
Go	tetrate.io	tetratelabs/proxy-wasm-go-sdk
Rust	proxy-wasm チーム (Istio コミュニティー)	proxy-wasm/proxy-wasm-rust-sdk

1.19.2. WasmPlugin コンテナー形式

Istio は、Wasm プラグインメカニズムで Open Container Initiative (OCI) イメージをサポートしています。Wasm プラグインをコンテナーイメージとして配布でき、**spec.url** フィールドを使用してコンテナーレジストリーの場所を参照できます。たとえば、**quay.io/my-username/my-plugin:latest** です。

WASM モジュールの各実行環境 (ランタイム) にはランタイム固有の設定パラメーターを設定できるため、WASM イメージは次の 2 つの階層で設定できます。

- **plugin.wasm** (必須) - コンテンツレイヤー。このレイヤーは、ランタイムによってロードされる WebAssembly モジュールのバイトコードを含む **.wasm** バイナリーで設定されます。このファイルには **plugin.wasm** という名前を付ける必要があります。
- **runtime-config.json** (オプション) - 設定レイヤー。このレイヤーは、ターゲットランタイムのモジュールに関するメタデータを記述する JSON 形式の文字列で設定されます。ターゲットランタイムによっては、設定レイヤーに追加のデータが含まれる場合もあります。たとえば、WASM Envoy Filter の設定には、フィルターで使用可能な root_ids が含まれています。

1.19.3. WasmPlugin API リファレンス

WasmPlugins API には、WebAssembly フィルターを介して Istio プロキシーによって提供される機能を拡張するメカニズムがあります。

複数の WasmPlugin をデプロイできます。**phase** および **priority** の設定により、実行の順序が (Envoy のフィルターチェーンの一部として) 決定され、ユーザー提供の WasmPlugins と Istio の内部フィルター間の複雑な対話設定が可能になります。

次の例では、認証フィルターが OpenID フローを実装し、Authorization ヘッダーに JSON Web Token (JWT) を入力します。Istio 認証はこのトークンを消費し、ingress ゲートウェイにデプロイします。WasmPlugin ファイルはプロキシーサイドカーファイルシステムに存在します。フィールドの **URL** に注意してください。

```
apiVersion: extensions.istio.io/v1alpha1
```

```
kind: WasmPlugin
metadata:
  name: openid-connect
  namespace: istio-ingress
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  url: file:///opt/filters/openid.wasm
  sha256: 1ef0c9a92b0420cf25f7fe5d481b231464bc88f486ca3b9c83ed5cc21d2f6210
  phase: AUTHN
  pluginConfig:
    openid_server: authn
    openid_realm: ingress
```

以下は同じ例ですが、今回はファイルシステム内のファイルの代わりに Open Container Initiative (OCI) イメージが使用されています。フィールド **url**、**imagePullPolicy**、および **imagePullSecret** に注意してください。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: openid-connect
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  url: oci://private-registry:5000/openid-connect/openid:latest
  imagePullPolicy: IfNotPresent
  imagePullSecret: private-registry-pull-secret
  phase: AUTHN
  pluginConfig:
    openid_server: authn
    openid_realm: ingress
```

表1.15 **WasmPlugin** フィールドリファレンス

フィールド	タイプ	説明	必須

フィールド	タイプ	説明	必須
spec.selector	WorkloadSelector	このプラグイン設定を適用する必要がある Pod/VM の特定のセットを選択するために使用される基準。省略した場合に、この設定は同じ namespace 内のすべてのワークロードインスタンスに適用されます。**WasmPlugin** フィールドが configroot namespace に存在する場合には、任意の namespace の該当するすべてのワークロードに適用されます。	いいえ
spec.url	文字列	Wasm モジュールまたは OCI コンテナーの URL。スキームが存在しない場合には、デフォルトで **oci:**// になり、OCI イメージを参照します。その他の有効なスキームとして、プロキシーコンテナー内にローカルに存在する.wasm モジュールファイルを参照するための **file:**// と、リモートでホストされる.wasm モジュールファイルを参照するため の **http s:**// があります。	いいえ
spec.sha256	文字列	Wasm モジュールまたは OCI コンテナーの検証に使用される SHA256 チェックサム。**url** フィールドがすでに SHA256 を参照している場合 (**@sha256:** 表記を使用) には、このフィールドの値と一致する必要があります。OCI イメージがタグによって参照され、このフィールドが設定されている場合には、プル後にそのチェックサムがこのフィールドの内容に対して検証されます。	いいえ

フィールド	タイプ	説明	必須
spec.imagePullPolicy	PullPolicy	OCI イメージをフェッチするときに適用されるプル動作。イメージが SHA ではなくタグで参照されている場合にのみ参照します。デフォルト値は **IfNotPresent** です。ただし、**URL** フィールドで OCI イメージが参照され、**latest** タグが使用されている場合は、**Always** の値がデフォルトで、K8s の動作を反映しています。**url** フィールドが **file:**// または **http s:**// を使用して Wasm モジュールを直接参照している場合には、設定が無視されます。	いいえ
spec.imagePullSecret	文字列	OCI イメージのプルに使用するクレデンシャル。イメージをプルするときにレジストリーに対して認証するためのプルシークレットなど、**WasmPlugin** オブジェクトと同じ namespace 内のシークレットの名前。	いいえ
spec.phase	PluginPhase	フィルターチェーンのどこにこの **WasmPlugin** オブジェクトを挿入するかを決定します。	いいえ

フィールド	タイプ	説明	必須
spec.priority	**int64**	**phase** の値が同じ **WasmPlugins** オブジェクトの順序を決定します。複数の **WasmPlugins** オブジェクトが同じフェーズで同じワークロードに適用される場合には、それらは優先度と降順をもとに適用されます。優先度 フィールドが設定されていない場合や値が同じ **WasmPlugins** オブジェクトが 2 つある場合には、順序は **WasmPlugins** オブジェクトの名前と namespace をもとに決定されます。デフォルト値は **0** です。	いいえ
spec.pluginName	文字列	Envoy 設定で使用されるプラグイン名。一部の Wasm モジュールでは、実行する Wasm プラグインを選択するためにこの値が必要になる場合があります。	いいえ
spec.pluginConfig	Struct	プラグインに渡される設定。	いいえ
spec.pluginConfig.verificationKey	文字列	署名された OCI イメージまたは Wasm モジュールの署名を検証するために使用される公開鍵。PEM 形式で指定する必要があります。	いいえ

WorkloadSelector オブジェクトは、フィルターをプロキシーに適用できるかどうかを判別するために使用される条件を指定します。一致の条件には、プロキシーに関連付けられたメタデータ、Pod/VM に添付されたラベルなどのワークロードインスタンス情報、またはプロキシーが最初のハンドシェイク中に Istio に提供するその他の情報が含まれます。複数の条件が指定されている場合には、ワークロードインスタンスを選択するには、すべての条件が一致する必要があります。現在、ラベルベースの選択メカニズムのみがサポートされています。

表1.16 WorkloadSelector

フィールド	タイプ	説明	必須
matchLabels	map<string, string>	ポリシーを適用する必要がある Pod/VM の特定のセットを示す 1 つ以上のラベル。ラベル検索の範囲は、リソースが存在する設定 namespace に限定されます。	はい

PullPolicy オブジェクトは、OCI イメージをフェッチするときに適用されるプル動作を指定します。

表1.17 PullPolicy

値	説明
<empty>	デフォルト値は **IfNotPresent** です。ただし、OCI イメージのタグが latest の場合は、デフォルト値は **Always** です。
IfNotPresent	イメージの既存のバージョンが以前にプルされている場合は、それが使用されます。イメージのバージョンがローカルに存在しない場合は、最新バージョンをプルします。
Always	このプラグインを適用するときは、常に最新バージョンのイメージをプルします。

Struct は、動的に型付けされた値にマップされるフィールドで設定される構造化データ値を表します。一部の言語では、Struct はネイティブ表現でサポートされている場合があります。たとえば、JavaScript のようなスクリプト言語では、構造体はオブジェクトとして表されます。

表1.18 Struct

フィールド	タイプ	説明
fields	map<string, Value>	動的に型付けされた値のマップ。

PluginPhase は、プラグインが注入されるフィルターチェーンのフェーズを指定します。

表1.19 PluginPhase

フィールド	説明
<empty>	コントロールプレーンは、プラグインを挿入する場所を決定します。これは通常、フィルターチェーンの最後かつ、ルーターの直前にあります。プラグインが他のプラグインから独立している場合は、PluginPhase を指定しないでください。

フィールド	説明
AUTHN	Istio 認証フィルターの前にプラグインを挿入します。
AUTHZ	Istio 認証フィルターの前と Istio 認証フィルターの後にプラグインを挿入します。
STATS	Istio 統計フィルターの前と Istio 認証フィルターの後にプラグインを挿入します。

1.19.3.1. WasmPlugin リソースのデプロイ

WasmPlugin リソースを使用して、Red Hat Service Mesh エクステンションを有効にできます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。次の例では、OpenID Connect フローを実行してユーザーを認証する **openid-connect** フィルターを作成します。

手順

1. 以下のリソース例を作成します。

plugin.yaml の例

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: openid-connect
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  url: oci://private-registry:5000/openid-connect/openid:latest
  imagePullPolicy: IfNotPresent
  imagePullSecret: private-registry-pull-secret
  phase: AUTHN
  pluginConfig:
    openid_server: authn
    openid_realm: ingress
```

2. 次のコマンドを使用して **plugin.yaml** ファイルを適用します。

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

1.19.4. ServiceMeshExtension コンテナー形式

コンテナーイメージを有効な拡張イメージにするために、WebAssembly モジュールのバイトコードを含む **.wasm** ファイルとコンテナーファイルシステムのルートに **manifest.yaml** ファイルが必要です。

注記

新しい WebAssembly 拡張機能を作成するときは、WasmPlugin を使用してください。ServiceMeshExtension は、Red Hat OpenShift Service Mesh バージョン 2.2 で非推奨になり、今後のリリースで削除される予定です。

manifest.yaml

```
schemaVersion: 1

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

表1.20 **manifest.yml** フィールドの参照情報

フィールド	説明	必須
schemaVersion	マニフェストスキーマのバージョン管理に使用されます。現時点で使用できる値は **1** のみです。	これは必須フィールドです。
name	拡張の名前です。	このフィールドは単なるメタデータであり、現時点では使用されていません。
description	拡張の説明。	このフィールドは単なるメタデータであり、現時点では使用されていません。
version	拡張のバージョンです。	このフィールドは単なるメタデータであり、現時点では使用されていません。
phase	拡張のデフォルトの実行フェーズです。	これは必須フィールドです。
priority	拡張のデフォルトの優先度です。	これは必須フィールドです。
module	コンテナーファイルシステムのルートから WebAssembly モジュールへの相対パスです。	これは必須フィールドです。

1.19.5. ServiceMeshExtension リファレンス

ServiceMeshExtension API には、WebAssembly フィルターを介して Istio プロキシーによって提供される機能を拡張するメカニズムがあります。WebAssembly 拡張機能の作成には 2 つの部分があります。

1. proxy-wasm API を公開する SDK を使用して拡張機能を記述し、それを WebAssembly モジュールにコンパイルします。
2. コンテナーにパッケージ化します。

注記

新しい WebAssembly 拡張機能を作成するときは、WasmPlugin を使用してください。ServiceMeshExtension は、Red Hat OpenShift Service Mesh バージョン 2.2 で非推奨になり、今後のリリースで削除される予定です。

表1.21 ServiceMeshExtension フィールドの参照情報

フィールド	説明
metadata.namespace	**ServiceMeshExtension** ソースの **metadata.namespace** フィールドには特殊なセマンティクスが含まれます。これが Control Plane Namespace と等しい場合、拡張はその **workloadSelector** の値に一致する Service Mesh のすべてのワークロードに適用されます。これは、その他の Mesh namespace にデプロイされる場合、同じ namespace のワークロードにのみ適用されます。
spec.workloadSelector	**spec.workloadSelector** フィールドには、Istio Gateway リソース の **spec.selector** フィールドと同じセマンティクスが含まれます。これは Pod ラベルに基づいてワークロードに一致します。**workloadSelector** の値が指定されていない場合、拡張は namespace のすべてのワークロードに適用されます。
spec.config	これは、拡張に転送される構造化フィールドで、セマンティクスはデプロイしている拡張に依存します。
spec.image	拡張を保持するイメージを参照するコンテナーイメージ URI です。
spec.phase	このフェーズでは、認証、認可、メトリクスの生成などの既存の Istio 機能に関連して、拡張が挿入されるフィルターチェーン内の場所を決定します。有効な値は、PreAuthN、PostAuthN、PreAuthZ、PostAuthZ、PreStats、PostStats です。このフィールドは、拡張の **manifest.yaml** ファイルで設定される値にデフォルト設定されますが、ユーザーが上書きできます。

フィールド	説明
spec.priority	同じ **spec.phase** の値を持つ複数の拡張機能が同じワークロードインスタンスに適用される場合、**spec.priority** は実行の順序を決定します。優先度が高い拡張機能が最初に実行されます。これにより、相互に依存する拡張が使用可能になります。このフィールドは、拡張の **manifest.yaml** ファイルで設定される値にデフォルト設定されますが、ユーザーが上書きできます。

1.19.5.1. ServiceMeshExtension リソースのデプロイ

Red Hat OpenShift Service Mesh 拡張機能は **ServiceMeshExtension** リソースを使用して有効にできます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

注記

新しい WebAssembly 拡張機能を作成するときは、WasmPlugin を使用してください。ServiceMeshExtension は、Red Hat OpenShift Service Mesh バージョン 2.2 で非推奨になり、今後のリリースで削除される予定です。

Rust SDK を使用してビルドされる完全なサンプルについては、header-append-filter を参照してください。これは、拡張の **config** フィールドから取られた名前および値で HTTP 応答に 1 つ以上のヘッダーを追加する単純なフィルターです。以下のスニペットの設定例を参照してください。

手順

1. 以下のリソース例を作成します。

ServiceMeshExtension リソース拡張機能の例

```
apiVersion: maistra.io/v1
kind: ServiceMeshExtension
metadata:
  name: header-append
  namespace: istio-system
spec:
  workloadSelector:
    labels:
      app: httpbin
  config:
    first-header: some-value
    another-header: another-value
  image: quay.io/maistra-dev/header-append-filter:2.1
  phase: PostAuthZ
  priority: 100
```

2. 以下のコマンドを使用して **extension.yaml** ファイルを適用します。

```
$ oc apply -f <extension>.yaml
```

1.19.6. ServiceMeshExtension リソースから WasmPlugin リソースへの移行

ServiceMeshExtension API は、Red Hat OpenShift Service Mesh バージョン 2.2 で非推奨になり、今後のリリースで削除される予定です。**ServiceMeshExtention** API を使用している場合は、WebAssembly 拡張を引き続き使用するために **WasmPlugin** API に移行する必要があります。

API は非常に似ています。移行の手順は、以下の 2 つのステップで設定されます。

1. プラグインファイルの名前を変更し、モジュールパッケージを更新する。
2. 更新されたコンテナーイメージを参照する **WasmPlugin** リソースを作成する。

1.19.6.1. API の変更

新しい **WasmPlugin** API は **ServiceMeshExtension** に似ていますが、いくつかの違いがあります (特にフィールド名)。

表1.22 **ServiceMeshExtensions** と **WasmPlugin**の間のフィールドの変更

ServiceMeshExtension	WasmPlugin
spec.config	**spec.pluginConfig**
spec.workloadSelector	**spec.selector**
spec.image	**spec.url**
spec.phase 有効な値: PreAuthN、PostAuthN、PreAuthZ、PostAuthZ、PreStats、PostStats	**spec.phase** 有効な値: <empty>、AUTHN、AUTHZ、STATS

以下は、**ServiceMeshExtension** リソースを **WasmPlugin** リソースに変換する方法の例になります。

ServiceMeshExtension リソース

```
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
```

上記の ServiceMeshExtension と等価な新しい WasmPlugin リソース

```
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. コンテナーイメージの形式の変更

新しい **WasmPlugin** コンテナーイメージの形式は **ServiceMeshExtensions** と似ていますが、以下のような違いがあります。

- **ServiceMeshExtension** コンテナー形式には、コンテナーファイルシステムのルートディレクトリーに **manifest.yaml** という名前のメタデータファイルが必要でした。**WasmPlugin** コンテナー形式には **manifest.yaml** ファイルは必要ありません。
- 任意のファイル名を付けることができた **.wasm** ファイル (実際のプラグイン) には **plugin.wasm** という名前を付け、コンテナーファイルシステムのルートディレクトリーに配置する必要があります。

1.19.6.3. WasmPlugin リソースへの移行

WebAssembly 拡張を **ServiceMeshExtension** API から **WasmPlugin** API にアップグレードするには、プラグインファイルの名前を変更します。

前提条件

- **ServiceMeshControlPlane** がバージョン 2.2 以降にアップグレードされる。

注意

どちらのプラグインも要求ごとに呼び出されるため、新しい **WasmPlugin** リソースを作成する前に、既存の **ServiceMeshExtension** リソースを削除しなければならない場合があります。2 つのプラグインが同時にアクティブな状態では、望ましくない結果が生じる可能性があります。

手順

1. コンテナーイメージを更新します。プラグインがすでにコンテナー内の **/plugin.wasm** にある場合は、次のステップに進みます。そうでない場合は、以下を行います。
   a. プラグインファイルの名前が **plugin.wasm** であることを確認します。拡張ファイルには **plugin.wasm** という名前を付ける必要があります。
   b. プラグインファイルがルート (/) ディレクトリーにあることを確認します。拡張ファイルをコンテナーファイルシステムのルートに配置する必要があります。
   c. コンテナーイメージを再ビルドして、これをコンテナーレジストリーにプッシュします。

2. **ServiceMeshExtension** リソースを削除し、ビルドした新しいコンテナーイメージを参照する **WasmPlugin** リソースを作成します。

1.20. 3SCALE WEBASSEMBLY モジュールの使用

注記

threescale-wasm-auth モジュールは、3scale API Management 2.11 以降と Red Hat OpenShift Service Mesh 2.1.0 以降のインテグレーションで実行されます。

threescale-wasm-auth モジュールは、アプリケーションバイナリーインターフェイス (**ABI**) と呼ばれるインターフェイスセットを使用する WebAssembly モジュールです。これは、ABI を実装するソフトウェアの一部を駆動する **Proxy-WASM** 仕様によって定義され、3scale に対する HTTP リクエストを承認できます。

ABI 仕様として、Proxy-WASM は、**host** という名前のソフトウェアと、モジュール、プログラム、または拡張という名前のソフトウェアの間の相互作用を定義します 。ホストは、モジュールが使用するサービスセットを公開してタスクを実行し、この場合はプロキシーリクエストを処理します。

ホスト環境は、ソフトウェアの一部 (この場合は HTTP プロキシー) と対話する WebAssembly 仮想マシンで設定されています。

モジュール自体は、仮想マシンで実行する手順と Proxy-WASM によって指定された ABI を除き、外部環境とは独立して実行されます。これは、ソフトウェアを参照する拡張を提供するのに安全な方法です。拡張は、仮想マシンおよびホストと明確に定義された方法でのみ対話できます。対話により、コンピューティングモデルと、プロキシーが持つ外部への接続が提供されます。

1.20.1. 互換性

threescale-wasm-auth モジュールは、**Proxy-WASM ABI** 仕様のすべての実装と完全に互換性を持つように設計されています。ただし、この時点で、連携することが完全にテストされているのは Envoy リバースプロキシーだけです。

1.20.2. スタンドアロンモジュールとしての使用

その自己完結型設計により、このモジュールが Service Mesh と 3scale Istio アダプターのデプロイメントとは独立して Proxy-WASM プロキシーと連携するように設定できます。

1.20.3. 前提条件

- このモジュールは、サポートされているすべての 3scale リリースで動作します。ただし、3scale 2.11 以降が必要な OpenID 接続 (OIDC) を使用するようにサービスを設定する場合を除きます。

1.20.4. threescale-wasm-auth モジュールの設定

OpenShift Container Platform のクラスター管理者は、**threescale-wasm-auth** モジュールを設定し、アプリケーションバイナリーインターフェイス (ABI) を使用して 3scale API Management への HTTP 要求を承認することができます。ABI は、ホストとモジュール間の対話を定義し、ホストサービスを公開し、モジュールを使用してプロキシー要求を処理することができます。

1.20.4.1. Service Mesh の拡張

Service Mesh は カスタムリソース定義 を提供し、**ServiceMeshExtension** と呼ばれる Proxy-WASM 拡張をサイドカープロキシーに指定および適用します。Service Mesh は、3scale での HTTP API 管理を必要とするワークロードのセットにこのカスタムリソースを適用します。

注記

WebAssembly 拡張の設定は、現在手動プロセスです。3scale システムからサービスの設定を取得するサポートは、今後のリリースでご利用いただけます。

前提条件

- このモジュールを適用する Service Mesh デプロイメントで Kubernetes ワークロードおよび namespace を特定します。
- 3scale テナントアカウントが必要です。適合するサービスならびに該当するアプリケーションおよびメトリクスが定義された SaaS または オンプレミス型 3scale 2.11 を参照してください。
- モジュールを **bookinfo** namespace の **productpage** マイクロサービスに適用する場合は、Bookinfo のサンプルアプリケーション を参照してください。
  - 以下の例は、**threescale-wasm-auth** モジュールのカスタムリソースの YAML 形式です。この例では、アップストリームの Maistra バージョンの Service Mesh である ServiceMeshExtension API を参照します。モジュールが適用されるアプリケーションのセットを特定する **WorkloadSelector** と合わせて、**threescale-wasm-auth** モジュールがデプロイされる namespace を宣言する必要があります。

```
apiVersion: maistra.io/v1
kind: ServiceMeshExtension
metadata:
  name: threescale-wasm-auth
  namespace: bookinfo (1)
spec:
  workloadSelector: (2)
    labels:
      app: productpage
  config: <yaml_configuration>
  image: registry.redhat.io/openshift-service-mesh/3scale-auth-wasm-rhel8:0.0.1
  phase: PostAuthZ
  priority: 100
```

(1) **namespace**

(2) **WorkloadSelector**

- **spec.config** フィールドはモジュール設定に依存し、直前の例では入力されません。この例では、**<yaml_configuration>** プレースホルダーの値を使用します。このカスタムリソースの例の形式を使用できます。
  - **spec.config** フィールドの値はアプリケーションによって異なります。その他のフィールドはすべて、このカスタムリソースの複数のインスタンス間で永続します。以下に例を示します。
    - **image**: 新しいバージョンのモジュールがデプロイされる場合にのみ変更されます。

    - **phase**: このモジュールは、OpenID Connect (OIDC) トークンの検証など、プロキシーがローカルの承認を行った後に呼び出す必要があるため、同じままです。

- **spec.config** および残りのカスタムリソースにモジュール設定を追加したら、**oc apply** コマンドでこれを適用します。

```
$ oc apply -f threescale-wasm-auth-bookinfo.yaml
```

関連情報

- **ServiceMeshExtension** リソースのデプロイ
- Custom Resources

1.20.5. 3scale 外部 ServiceEntry オブジェクトの適用

threescale-wasm-auth モジュールに 3scale に対するクエストを承認させるには、モジュールは 3scale サービスにアクセスできる必要があります。これは、外部 **ServiceEntry** オブジェクトを適用して Red Hat OpenShift Service Mesh および Istio 内で実行できます。

Service Management API および Account Management API のバックエンドおよびシステムコンポーネントについて、カスタムリソースは、Service Mesh 内からホスト型 3scale (SaaS) にアクセスするためのサービスエントリーを設定します。Service Management API は、各リクエストの承認ステータスのクエリーを受信します。Account Management API は、サービスの API 管理設定を提供します。

手順

- 以下の外部 **ServiceEntry** カスタムリソースをクラスターに適用します。

ホスト型 3scale バックエンド用のカスタムリソース

```
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: threescale-saas-backend
spec:
  hosts:
  - su1.3scale.net
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
```

ホスト型 3scale システムのカスタムリソース

```
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: threescale-saas-system
spec:
  hosts:
  - multitenant.3scale.net
```

```
ports:
- number: 443
  name: https
  protocol: HTTPS
location: MESH_EXTERNAL
resolution: DNS
```

以下のいずれかの方法を使用して **oc apply** コマンドを使用して、オブジェクトを適用できます。

- オブジェクトを 1 つ以上のファイルに保存し、以下の構文を使用します。

```
$ oc apply -f <filename.yml>
```

- 最初にファイルに保存せずにオブジェクトを適用するには、以下のコマンドを使用します。

```
$ echo -n "<filename.yml>" | oc apply -f -
```

または、メッシュ内の 3scale サービスをデプロイすることができます。これには、カスタムリソース内のこれらのサービスの場所を変更します。

関連情報

- **ServiceEntry** ドキュメント

1.20.6. 3scale WebAssembly モジュール設定

ServiceMeshExtension カスタムリソース仕様は、**Proxy-WASM** モジュールが読み取る設定を提供します。

仕様はホストに組み込まれ、**Proxy-WASM** モジュールによって読み取られます。通常、設定は、解析するモジュールの JSON ファイル形式ですが、**ServiceMeshExtension** リソースは仕様値を YAML として解釈し、モジュールで使用するために JSON に変換できます。

スタンドアロンモードで **Proxy-WASM** モジュールを使用する場合は、JSON 形式を使用して設定を作成する必要があります。JSON 形式を使用する場合、**host** 設定ファイル内の必要な場所で、エスケープと引用を使用することができます (例:**Envoy**)。**ServiceMeshExtension** リソースで WebAssembly モジュールを使用する場合、設定は YAML 形式になります。この場合、無効な設定により、JSON 表現に基づいて診断がモジュールによって強制的にサイドカーコンテナーのロギングストリームに表示されます。

重要

EnvoyFilter カスタムリソースは一部の 3scale Istio アダプターまたは Service Mesh リリースで使用できますが、このカスタムリソースはサポートされる API ではありません。**EnvoyFilter** カスタムリソースの使用は推奨されていません。**EnvoyFilter** カスタムリソースの代わりに **ServiceMeshExtension** API を使用します。**EnvoyFilter** カスタムリソースを使用する必要がある場合は、仕様を JSON 形式で指定する必要があります。

1.20.6.1. 3scale WebAssembly モジュールの設定

3scale の WebAssembly モジュール設定のアーキテクチャーは、3scale アカウントおよび承認サービスや処理するサービスの一覧によって異なります。

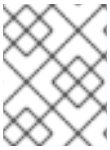
前提条件

前提条件は、すべてのケースで最小の必須フィールドのセットです。

- 3scale アカウントおよび承認サービス:**backend-listener** URL。
- 処理するサービス一覧: サービス ID と少なくとも 1 つの認証情報の検索方法、およびその検索場所。
- **userkey**、**appid**、**appkey**、および OpenID Connect (OIDC) パターンを処理する例があります。
- WebAssembly モジュールは、静的設定で指定した設定を使用します。たとえば、モジュールにマッピングルール設定を追加する場合は、3scale 管理ポータルにこのようなマッピングルールが設定されていない場合でも、常に適用されます。残りの **ServiceMeshExtension** リソースは **spec.config** YAML エントリーにあります。

1.20.6.2. 3scale WebAssembly モジュール api オブジェクト

3scale WebAssembly モジュールからの **api** 最上位文字列は、モジュールが使用する設定のバージョンを定義します。



注記

存在しないバージョンまたはサポート対象外のバージョンの **api** オブジェクトの場合、レンダリングされた 3scale WebAssembly モジュールは操作不能になります。

api 最上位文字列の例

```
apiVersion: maistra.io/v1
kind: ServiceMeshExtension
metadata:
  name: threescale-wasm-auth
  namespace: bookinfo
spec:
  config:
    api: v1
...
```

api エントリーは、設定の残りの値を定義します。許可される値は **v1** のみです。現在の設定との互換性を壊す、または **v1** を使用するモジュールが処理できないロジックを必要とする新しい設定には、異なる値が必要になります。

1.20.6.3. 3scale WebAssembly モジュール system オブジェクト

system 最上位オブジェクトは、特定のアカウントの 3scale Account Management API にアクセスする方法を指定します。**upstream** フィールドは、オブジェクトの最も重要な部分です。**system** オブジェクトはオプションですが、3scale の**system**コンポーネントへの接続を提供しない場合のオプションである 3scale WebAssembly モジュールに完全な静的設定を提供する場合を除き、推奨されます。

system オブジェクトに加えて静的設定オブジェクトを指定する場合は、静的な設定オブジェクトが優先されます。

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

```
kind: ServiceMeshExtension
metadata:
  name: threescale-wasm-auth
spec:
  ...
  config:
    system:
      name: saas_porta
      upstream: <object>
      token: myaccount_token
      ttl: 300
  ...
```

表1.23 **system** オブジェクトフィールド

名前	説明	必須
name	3scale サービスの識別子 (現在、参照されていません)。	任意
upstream	問い合わせるネットワークホストの詳細。**upstream** は、system として知られる 3scale Account Management API ホストを参照します。	はい
token	読み取り権限を持つ 3scale の個人アクセストークン。	はい
ttl	新規の変更を取得する前に、このホストから取得した設定を有効なものと見なす最小時間 (秒数)。デフォルトは 600 秒 (10 分) です。**注記:** 最大の期間はありませんが、モジュールは通常、この TTL が経過した後に妥当な時間内に設定を取得します。	任意

1.20.6.4. 3scale WebAssembly モジュール upstream オブジェクト

upstream オブジェクトは、プロキシーが呼び出しを実行できる外部ホストについて説明しています。

```
apiVersion: maistra.io/v1
upstream:
  name: outbound|443||multitenant.3scale.net
  url: "https://myaccount-admin.3scale.net/"
  timeout: 5000
...
```

表1.24 **upstream** オブジェクトフィールド

名前	説明	必須
name	**name** は自由形式の識別子ではありません。これは、プロキシー設定で定義される外部ホストの識別子です。スタンドアロン **Envoy** 設定の場合、これは他のプロキシーの **upstream** とも呼ばれる クラスター 名にマッピングします。**注記:**Service Mesh および 3scale Istio アダプターコントロールプレーンは、複数のフィールドの区切り文字として垂直バー (|) を使用する形式に従って名前を設定します。この統合の目的上、常に **outbound|<port>|| <hostname>** の形式を使用します。	はい
url	記述されたサービスにアクセスするための完全な URL。スキームによって暗示されていない限り、TCP ポートが含まれている必要があります。	はい
Timeout	応答にかかる時間がこの設定を超えたこのサービスへの接続がエラーとみなされるためのタイムアウト (ミリ秒単位)。デフォルトは 1000 秒です。	任意

1.20.6.5. 3scale WebAssembly モジュール backend オブジェクト

backend 最上位オブジェクトは、HTTP リクエストの承認および報告のために 3scale Service Management API にアクセスする方法を指定します。このサービスは、3scale の バックエンド コンポーネントによって提供されます。

```
apiVersion: maistra.io/v1
kind: ServiceMeshExtension
metadata:
  name: threescale-wasm-auth
spec:
  config:
    ...
    backend:
      name: backend
      upstream: <object>
    ...
```

表1.25 **backend** オブジェクトフィールド

名前	説明	必須
name	3scale バックエンドの識別子 (現在、参照されていません)。	任意
upstream	問い合わせるネットワークホストの詳細。これは、system として知られる 3scale Account Management API ホストを参照する必要があります。	有効。最も重要な必須フィールドです。

1.20.6.6. 3scale WebAssembly モジュール services オブジェクト

services の最上位オブジェクトは、**module** のこの特定のインスタンスで処理されるサービス識別子を指定します。

アカウントには複数のサービスがあるため、どのサービスを処理するかを指定する必要があります。残りの設定は、サービスの設定方法に関するものです。

services フィールドは必須です。有用とするサービスを少なくとも 1 つ含める必要がある配列です。

```
apiVersion: maistra.io/v1
kind: ServiceMeshExtension
metadata:
  name: threescale-wasm-auth
spec:
  config:
    ...
    services:
    - id: "2555417834789"
      token: service_token
      authorities:
        - "*.app"
        - 0.0.0.0
        - "0.0.0.0:8443"
      credentials: <object>
      mapping_rules: <object>
    ...
```

services 配列の各要素は、3scale サービスを表します。

表1.26 **services** オブジェクトフィールド

名前	説明	必須
ID	この 3scale サービスの識別子 (現在、参照されていません)。	はい

名前	説明	必須
token	この **token** は、System 内のサービスのプロキシー設定にあるか、以下の **curl** コマンドを使用して System から取得できます。 **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**	はい
authorities	文字列の配列。それぞれが一致する URL の**認証局** を表します。これらの文字列は、アスタリスク (*)、正符号 (+)、および疑問符 (?) マッチャーに対応する glob パターンを受け入れます。	はい
credentials	検索する認証情報の種類と場所を定義するオブジェクト。	はい
mapping_rules	ヒットするマッピングルールおよび 3scale メソッドを表すオブジェクトの配列。	はい

1.20.6.7. 3scale WebAssembly モジュール credentials オブジェクト

credentials オブジェクトは **service** オブジェクトのコンポーネントです。**credentials** は、検索する認証情報の種類と、このアクションを実行する手順を指定します。

すべてのフィールドはオプションですが、少なくとも 1 つの **user_key** または **app_id** を指定する必要があります。各認証情報を指定する順番は、モジュールによって事前確立されているために無関係です。各認証情報の 1 つのインスタンスのみを指定します。

```
apiVersion: maistra.io/v1
kind: ServiceMeshExtension
metadata:
  name: threescale-wasm-auth
spec:
  config:
    ...
    services:
    - credentials:
        user_key: <array_of_lookup_queries>
        app_id: <array_of_lookup_queries>
        app_key: <array_of_lookup_queries>
    ...
```

表1.27 **credentials** オブジェクトフィールド

名前	説明	必須
user_key	これは、3scale ユーザーキーを定義する検索クエリーの配列です。ユーザーキーは、一般に API キーと呼ばれます。	任意
app_id	これは、3scale のアプリケーション識別子を定義する検索クエリーの配列です。アプリケーションの識別子は、3scale または Red Hat Single Sign-On (RH-SSO) や OpenID Connect (OIDC) などのアイデンティティープロバイダーを使用して提供されます。成功して 2 つの値に解決するたびに、ここで指定された検索クエリーの解決で、**app_id** と **app_key** が設定されます。	任意
app_key	これは、3scale のアプリケーションキーを定義する検索クエリーの配列です。解決される **app_id** のないアプリケーションキーは無意味なため、**app_id** が指定されている場合のみこのフィールドを指定します。	任意

1.20.6.8. 3scale WebAssembly モジュール検索クエリー

lookup query オブジェクトは、**credentials** オブジェクトのフィールドの一部になります。特定の認証情報フィールドが検出され、処理される方法を指定します。評価されると、解決に成功すると、1 つ以上の値が見つかったことを意味します。解決に失敗したことは、値が見つからなかったことを意味します。

lookup queries の配列は、ショートサーキットまたは関係を定義しています。いずれかのクエリーの正常な解決により、残りのクエリーの評価が停止され、値を指定の credential-type に割り当てます。アレイの各クエリーは、互いに独立しています。

lookup query は、1 つのフィールド (ソースオブジェクト) で設定されています。これは、複数のソースタイプの 1 つになります。以下の例を参照してください。

```
apiVersion: maistra.io/v1
kind: ServiceMeshExtension
metadata:
  name: threescale-wasm-auth
spec:
  config:
    ...
    services:
    - credentials:
        user_key:
```

```
        - <source_type>: <object>
        - <source_type>: <object>
        ...
      app_id:
        - <source_type>: <object>
        ...
      app_key:
        - <source_type>: <object>
        ...
  ...
```

1.20.6.9. 3scale WebAssembly モジュール source オブジェクト

source オブジェクトは、任意の **credentials** オブジェクトフィールド内のソースの配列の一部として存在します。**source**-type として参照されるオブジェクトフィールド名は、以下のいずれかになります。

- **header**: 検索クエリーは、HTTP リクエストヘッダーを入力として受け取ります。
- **query_string**: **lookup query** は、URL クエリー文字列パラメーターを入力として受け取ります。
- **filter**: **lookup query** は、フィルターメタデータをインプットとして受け取ります。

すべての **source**-type オブジェクトには、少なくとも以下の 2 つのフィールドがあります。

表1.28 **source-type** オブジェクトフィールド

名前	説明	必須
keys	文字列の配列。それぞれが **key** で、入力データで検索されたエントリーを参照します。	はい
ops	**key** エントリーの照合を行う **operations** の配列。配列は、操作が入力を受け取り、次の操作の出力を生成するパイプラインです。出力に失敗した **operation** は、**lookup query** を失敗として解決します。操作のパイプラインの順序により、評価の順序が決定されます。	任意

filter フィールド名には、データの検索に使用するメタデータのパスを表示するのに必要な **path** エントリーがあります。

キーが 入力データと一致する場合は、残りの鍵は評価されず、ソース解決アルゴリズムは、指定した **operations** (**ops**) の実行にジャンプします。**ops** を指定しないと、一致する **key** の結果値 (ある場合) が返されます。

Operations は、最初のフェーズが **key** を検索した後に、入力に対する特定の条件および変換を指定する方法を提供します。プロパティーを変換、デコード、および要求する必要があるときに、**operations** を使用しますが、すべてのニーズに対応する成熟した言語は提供されず、Turing-completeness はあ

りません。

スタックは **operations** の出力を保存します。評価されると、認証情報が消費する値の数に応じて、スタックの下部に値を割り当てて、**lookup query** は終了します。

1.20.6.10. 3scale WebAssembly モジュール operations オブジェクト

特定の **source type** に属する **ops** 配列の各要素は、値に変換を適用するか、テストを実行する **operation** オブジェクトです。このようなオブジェクトに使用するフィールド名は **operation** 自体の名前で、値は **operation** に対するパラメーターです。これは、フィールドと値のマップ、リスト、または文字列など、構造化オブジェクトになります。

ほとんどの **operations** は、1 つ以上の入力を処理し、1 つ以上の出力を生成します。入力を使用したり、出力を生成したりする場合、それらは値のスタックで作業します。操作によって消費される各値は、値のスタックからポップアップされ、**source** マッチと共に初期入力されます。出力される値はスタックにプッシュされます。他の **operations** は、特定のプロパティーを要求する以外、出力を使用または生成しませんが、値のスタックを検査します。

注記

解決が完了すると、次の手順 (値を **app_id**、**app_key**、または **user_key** に割り当てるなど) でピックアップされる値はスタックの下部の値から取得されます。

operations カテゴリーはいくつかあります。

- **decode**: 別の形式を取得するために、入力値をデコードして変換します。
- **string**: 文字列値を入力として取り、変換を実行し、確認します。
- **stack**: 入力の値のセットを取得し、複数のスタック変換とスタック内の特定の位置の選択を実行します。
- **check**: 影響を及ぼさない方法で、操作セットに関するプロパティーを要求します。
- **control**: 評価フローを変更できる操作を実施します。
- **format**: 入力値の形式固有の構造を解析し、その値を検索します。

すべての操作は、name 識別子で文字列として指定されます。

関連情報

- 利用可能な 操作

1.20.6.11. 3scale WebAssembly モジュール mapping_rules オブジェクト

mapping_rules オブジェクトは **service** オブジェクトの一部です。これは、REST パスパターンのセットならびに関連する 3scale メトリクスおよびパターンが一致する時に使用するカウント増分を指定します。

system 最上位オブジェクトに動的設定が提供されていない場合、値が必要です。**system** 最上位エントリーに加えてオブジェクトが提供されると、**mapping_rules** オブジェクトが最初に評価されます。

mapping_rules は配列オブジェクトです。そのアレイの各要素は **mapping_rule** オブジェクトです。受信したリクエストの評価されたマッチするマッピングルールにより、承認および**APIManager** へのレ

ポート用の 3scale **methods** のセットが提供されます。複数のマッチングルールが同じ **methods** を参照する場合、3scale への呼び出し時に **deltas** の合算があります。たとえば、2 つのルールが、1 と 3 の **deltas** で **Hits** メソッドを 2 回増やすと、3scale にレポートする Hits の単一のメソッドエントリーの **delta** は 4 になります。

1.20.6.12. 3scale WebAssembly モジュール mapping_rule オブジェクト

mapping_rule オブジェクトは **mapping_rules** オブジェクトの配列の一部です。

mapping_rule オブジェクトフィールドは、以下の情報を指定します。

- 照合する **HTTP** 要求メソッド。
- パスに一致するパターン。
- 報告する量と共にレポートする 3scale メソッド。フィールドを指定する順番により、評価順序が決定されます。

表1.29 **mapping_rule** オブジェクトフィールド

名前	説明	必須
メソッド	HTTP リクエストメソッド (動詞) を表す文字列を指定します。許可される値は、許可される HTTP メソッド名の 1 つと一致し、大文字と小文字を区別しません。すべてのマッチのすべてのメソッドの特殊な値。	はい
pattern	HTTP リクエストの URI パスコンポーネントに一致するパターン。このパターンは、3scale で説明されている構文に従います。**{this}** などの中括弧間の文字のシーケンスを使用するワイルドカード (アスタリスク (*) 文字の使用) が許可されます。	はい
usages	**usage** オブジェクトの一覧。ルールがマッチすると、**deltas** を持つすべてのメソッドが、承認およびレポートのために 3scale に送信されるメソッドの一覧に追加されます。 以下の必須フィールドに **usages** オブジェクトを埋め込みます。 ● **name**: レポートする **method** のシステム名。 ● **delta**: その **method** の増分。	はい

名前	説明	必須
last	このルールが正常にマッチした場合に、それ以外のマッピングルールの評価を停止する必要があるかどうか。	任意のブール値。デフォルトは **false** です。

以下の例は、3scale のメソッド間の既存の階層とは独立しています。つまり、3scale 側で実行されたすべての内容はこれには影響しません。たとえば、**Hits** メトリクスは、それらすべての親となる可能性があるため、承認されたリクエストで報告されたすべてのメソッドの合計により 4 ヒットを保管し、3scale **Authrep** API エンドポイントを呼び出します。

以下の例では、すべてのルールに一致する、パス **/products/1/sold** への **GET** リクエストを使用します。

mapping_rules GET リクエストの例

```
apiVersion: maistra.io/v1
kind: ServiceMeshExtension
metadata:
  name: threescale-wasm-auth
spec:
  config:
    ...
    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
    ...
```

すべての **usages** は、モジュールが使用状況データを使用して 3scale に実施するリクエストに追加されます。

- Hits: 1
- products: 2
- sales: 1

1.20.7. credentials のユースケースについての 3scale WebAssembly モジュールの例

ほとんどの時間を費やして、設定手順を適用してサービスへのリクエストの認証情報を取得します。

以下は **credentials** の例です。これは、特定のユースケースに合わせて変更できます。

複数のソースオブジェクトと独自の **lookup queries** を指定する場合、これらはすべて組み合わせることができますが、いずれか 1 つが正しく解決されるまで、それらは順番に評価されます。

1.20.7.1. クエリー文字列パラメーターの API キー (user_key)

以下の例では、クエリー文字列パラメーターまたは同じ名前のヘッダーで **user_key** を検索します。

```
credentials:
  user_key:
    - query_string:
        keys:
          - user_key
    - header:
        keys:
          - user_key
```

1.20.7.2. アプリケーション ID およびキー

以下の例では、クエリーまたはヘッダーの **app_key** および **app_id** 認証情報を検索します。

```
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. 認証ヘッダー

リクエストには、**authorization** ヘッダーに **app_id** および **app_key** が含まれます。最後に出力される値が 1 つまたは 2 つある場合は、**app_key** を割り当てることができます。

ここでの解決は、最後に出力された 1 つまたは 2 つの出力がある場合は **app_key** を割り当てます。

authorization ヘッダーは承認の種類で値を指定し、その値は **Base64** としてエンコードされます。つまり、値を空白文字で分割し、2 番目の出力を取得して、コロン (:) をセパレーターとして使用して再度分割できます。たとえば、**app_id:app_key** という形式を使用する場合、ヘッダーは以下の **credential** の例のようになります。

```
aladdin:opensesame:  Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
```

以下の例のように、小文字のヘッダーフィールド名を使用する必要があります。

```
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
```

前述のユースケースの例は、**authorization** のヘッダーを確認します。

1. これは文字列の値を取り、スペースで分割し、**credential**-type および **credential** 自体の少なくとも 2 つの値を生成することを確認してから、**credential**-type をドロップします。

2. 次に、必要なデータが含まれる 2 番目の値をデコードし、最初の **app_id** の後にもしあれば **app_key** が含まれる操作スタックとなるように、コロン (:) 文字を使用して分割します。

   a. **app_key** が認証ヘッダーに存在しない場合は、特定のソースがチェックされます (この場合は、キー **app_key** のヘッダーなど)。

3. **credentials** に追加の条件を追加するには、**Basic** 認証を許可します。ここで、**app_id** は **aladdin** もしくは **admin**、または 長さが 8 文字以上の任意の **app_id** になります。

4. **app_key** には値が含まれ、以下の例のように最小で 64 文字を指定する必要があります。

```
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. **authorization** ヘッダーの値を選択したら、タイプが上部に配置されるようにスタックを逆にして **Basic credential**-type を取得します。

6. glob マッチを実行します。検証し、認証情報がデコードされ、分割されると、スタックの下部に **app_id** を取得し、上部に **app_key** を取得する可能性があります。

7. **test:** を実行します。スタックに 2 つの値がある場合は、 **app_key** が取得されたことになります。

   a. **app_id** および **app_key** を含め、文字列の長さが 1 から 63 文字になるようにします。キーの長さがゼロの場合は破棄し、キーが存在しないものとして続行します。**app_id** のみがあり、**app_key** がない場合は、不明なブランチは、テストに成功し、評価が続行されます。

最後の操作は **assert** で、スタックに副作用がないことを示します。その後、スタックを変更することができます。

1. **app_id** が最上部になるように、スタックを逆にします。

   a. **app_key** が存在するかどうかで、スタックを逆にすると、**app_id** が上部になります。

2. **and** を使用して、テスト間でスタックの内容を保持します。
   次に、以下のいずれかの方法を使用します。

   - **app_id** に 8 文字以上の文字列が設定されていることを確認してください。
   - **app_id** が **aladdin** または **admin** と一致していることを確認します。

1.20.7.4. OpenID Connect (OIDC) のユースケース

Service Mesh および 3scale Istio アダプターの場合、以下の例のように **RequestAuthentication** をデプロイし、独自のワークロードデータおよび **jwtRules** を入力する必要があります。

```
apiVersion: security.istio.io/v1beta1
  kind: RequestAuthentication
  metadata:
    name: jwt-example
    namespace: bookinfo
  spec:
    selector:
      matchLabels:
        app: productpage
    jwtRules:
    - issuer: >-
        http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak
      jwksUri: >-
        http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak/protocol/openid-
connect/certs
```

RequestAuthentication を適用するとき、**JWT** トークンを検証するためにネイティブプラグインで **Envoy** を設定します。プロキシーは、モジュールを実行する前にすべてを検証します。したがって、失敗したリクエストが 3scale WebAssembly モジュールに実行されません。

JWT トークンが検証されると、プロキシーはそのコンテンツを内部メタデータオブジェクトに格納します。エントリーのキーは、プラグインの特定の設定に依存します。このユースケースでは、不明なキー名が含まれる単一のエントリーを持つ構造化オブジェクトを検索することができます。

OIDC の 3scale **app_id** は、OAuth **client_id** と一致します。これは **JWT** トークンの **azp** フィールドまたは **aud** フィールドにあります。

Envoy のネイティブ **JWT** 認証フィルターから **app_id** フィールドを取得するには、以下の例を参照してください。

```
credentials:
  app_id:
    - filter:
        path:
          - envoy.filters.http.jwt_authn
          - "0"
        keys:
          - azp
          - aud
        ops:
          - take:
              head: 1
```

この例では、モジュールに対し、**filter** ソースタイプを使用して **Envoy** 固有の **JWT** 認証ネイティブプラグインからオブジェクトのフィルターメタデータを検索するよう指示します。このプラグインには、1 つのエントリーと事前に設定された名前を持つ構造化オブジェクトの一部として **JWT** トークンが含まれます。**0** を使用して、単一のエントリーのみにアクセスするように指定します。

結果の値は、以下の 2 つのフィールドを解決する構造です。

- **azp**: **app_id** が見つけられる値。
- **aud**: この情報も見つけられる値。

この操作により、割り当て用に 1 つの値のみが保持されます。

1.20.7.5. ヘッダーからの JWT トークンの取得

設定によっては、**JWT** トークンの検証プロセスがある場合があります。この場合、検証されたトークンが JSON 形式のヘッダーを介してこのモジュールに到達します。

app_id を取得するには、以下の例を参照してください。

```
credentials:
  app_id:
    - header:
        keys:
          - x-jwt-payload
        ops:
          - base64_urlsafe
          - json:
            - keys:
              - azp
              - aud
          - take:
              head: 1
```

1.20.8. 3scale WebAssembly モジュールの機能する最低限の設定

以下は、3scale WebAssembly モジュールの機能する最低限の設定の例です。これをコピーアンドペーストし、これを独自の設定で機能するように編集することができます。

```
apiVersion: maistra.io/v1
kind: ServiceMeshExtension
metadata:
  name: threescale-auth
spec:
  image: registry.redhat.io/openshift-service-mesh/3scale-auth-wasm-rhel8:0.0.1
  phase: PostAuthZ
  priority: 100
  workloadSelector:
    labels:
      app: productpage
  config:
    api: v1
    system:
      name: system-name
      upstream:
        name: outbound|443||multitenant.3scale.net
        url: https://istiodevel-admin.3scale.net/
        timeout: 5000
      token: atoken
    backend:
      name: backend-name
      upstream:
        name: outbound|443||su1.3scale.net
        url: https://su1.3scale.net/
        timeout: 5000
      extensions:
```

```
    - no_body
  services:
  - id: '2555417834780'
    token: service_token
    authorities:
    - "*"
      credentials:
        app_id:
          - header:
              keys:
                - app_id
          - query_string:
              keys:
                - app_id
                - application_id
        app_key:
          - header:
              keys:
                - app_key
          - query_string:
              keys:
                - app_key
                - application_key
        user_key:
          - query_string:
              keys:
                - user_key
          - header:
              keys:
                - user_key
    mapping_rules:
    - method: GET
      pattern: "/"
      usages:
      - name: Hits
        delta: 1
    - method: GET
      pattern: "/o{*}c"
      usages:
      - name: oidc
        delta: 1
      - name: Hits
        delta: 1
    - method: any
      pattern: "/{anything}?bigsale={*}"
      usages:
      - name: sale
        delta: 5
```

1.21. 3SCALE ISTIO アダプターの使用

3scale Istio アダプターはオプションのアダプターであり、これを使用すると、Red Hat OpenShift Service Mesh 内で実行中のサービスにラベルを付け、そのサービスを 3scale API Management ソリューションと統合できます。これは Red Hat OpenShift Service Mesh には必要ありません。

重要

3scale Istio アダプターは、Red Hat OpenShift Service Mesh バージョン 2.0 以前でのみ使用できます。Mixer コンポーネントはリリース 2.0 で非推奨となり、リリース 2.1 で削除されました。Red Hat OpenShift Service Mesh バージョン 2.1.0 以降では、3scale WebAssembly モジュール を使用する必要があります。

3scale Istio アダプターで 3scale バックエンドキャッシュを有効にする必要がある場合には、Mixer ポリシーと Mixer Telemetry も有効にする必要があります。Red Hat OpenShift Service Mesh コントロールプレーンのデプロイ を参照してください。

1.21.1. 3scale アダプターと Red Hat OpenShift Service Mesh の統合

これらの例を使用して、3scale Istio アダプターを使用してサービスに対する要求を設定できます。

前提条件:

- Red Hat OpenShift Service Mesh バージョン 2.x
- 稼働している 3scale アカウント (SaaS または 3scale 2.9 On-Premises)
- バックエンドキャッシュを有効にするには 3scale 2.9 以上が必要です。
- Red Hat OpenShift Service Mesh の前提条件
- Mixer ポリシーの適用が有効になっていることを確認します。Mixer ポリシー適用の更新についてのセクションでは、現在の Mixer ポリシーの適用ステータスをチェックし、ポリシーの適用を有効にする手順が説明されています。
- Mixer プラグインを使用している場合は、Mixer ポリシーと Telemetry は有効にされる必要があります。
  - アップグレード時に、Service Mesh コントロールプレーン (SMCP) を適切に設定する必要があります。

注記

3scale Istio アダプターを設定するために、アダプターパラメーターをカスタムリソースファイルに追加する手順については、Red Hat OpenShift Service Mesh カスタムリソースを参照してください。

注記

kind: handler リソースにとくに注意してください。これを 3scale アカウントの認証情報を使用して更新する必要があります。オプションで **service_id** をハンドラーに追加できますが、この設定は 3scale アカウントの 1 つのサービスに対してのみ有効で、後方互換性を確保するためにだけ維持されています。**service_id** をハンドラーに追加する場合は、他のサービスに対して 3scale を有効にする必要がある場合は、別の **service_ids** で追加のハンドラーを作成する必要があります。

以下の手順に従って、3scale アカウントごとに単一のハンドラーを使用します。

手順

1. 3scale アカウントのハンドラーを作成し、アカウントの認証情報を指定します。サービス識別子を省略します。

```
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"
```

   オプションで、**params** セクション内の **backend_url** フィールドを指定して、3scale 設定によって提供される URL を上書きできます。これは、アダプターが 3scale のオンプレミスインスタンスと同じクラスターで実行され、内部クラスター DNS を利用する必要がある場合に役立ちます。

2. 3scale アカウントに属するサービスの Deployment リソースを編集するか、またはパッチを適用します。

   a. 有効な **service_id** に対応する値を使用して **"service-mesh.3scale.net/service-id"** ラベルを追加します。

   b. 手順 1 の ハンドラーリソースの名前 の値を使用して **"service-mesh.3scale.net/credentials"** ラベルを追加します。

3. 他のサービスを追加する場合には、手順 2 を実行して、3scale アカウントの認証情報とそのサービス識別子にリンクします。

4. 3scale 設定でルールの設定を変更し、ルールを 3scale ハンドラーにディスパッチします。

   ルール設定の例

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

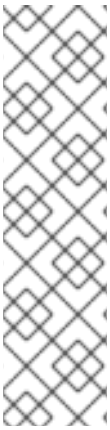
1.21.1.1. 3scale カスタムリソースの生成

アダプターには、**handler**、**instance**、および **rule** カスタムリソースの生成を可能にするツールが含まれます。

表1.30 使用法

オプション	説明	必須	デフォルト値
-h, --help	利用可能なオプションについてのヘルプ出力を生成します	いいえ	
--name	この URL の一意の名前、トークンのペア	はい	
-n, --namespace	テンプレートを生成する namespace	いいえ	istio-system
-t, --token	3scale アクセストークン	はい	
-u, --url	3scale 管理ポータル URL	はい	
--backend-url	3scale バックエンド URL。これが設定されている場合、システム設定から読み込まれる値がオーバーライドされます。	いいえ	
-s, --service	3scale API/サービス ID	いいえ	
--auth	指定する 3scale 認証パターン (1=API Key, 2=App Id/App Key, 3=OIDC)	いいえ	ハイブリッド
-o, --output	生成されたマニフェストを保存するファイル	いいえ	標準出力
--version	CLI バージョンを出力し、即座に終了する	いいえ	

1.21.1.1.1. URL サンプルからのテンプレートの生成



注記

- デプロイされたアダプターからのマニフェストの生成 で、3scale アダプターコンテナーイメージからの **oc exec** を使用して以下のコマンドを実行します。
- **3scale-config-gen** コマンドを使用すると、YAML 構文とインデントエラーを回避するのに役立ちます。
- このアノテーションを使用する場合は **--service** を省略できます。
- このコマンドは、**oc exec** を使用してコンテナーイメージ内から起動する必要があります。

手順

- **3scale-config-gen** コマンドを使用して、トークンと URL のペアを 1 つのハンドラーとして複数のサービスで共有できるようにテンプレートを自動生成します。

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

- 以下の例では、ハンドラーに埋め込まれたサービス ID を使用してテンプレートを生成します。

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

関連情報

- トークン

1.21.1.2. デプロイされたアダプターからのマニフェストの生成

注記

- **NAME** は、3scale で管理するサービスの識別に使用する識別子です。
- **CREDENTIALS_NAME** 参照は、ルール設定の **match** セクションに対応する識別子です。CLI ツールを使用している場合は、**NAME** 識別子に自動設定されます。
- この値は具体的なものでなくても構いませんが、ラベル値はルールの内容と一致させる必要があります。詳細は、アダプター経由でのサービストラフィックのルーティング を参照してください。

1. このコマンドを実行して、**istio-system** namespace でデプロイされたアダプターからマニフェストを生成します。

```
$ 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. これでターミナルにサンプル出力が生成されます。必要に応じて、これらのサンプルを編集し、**oc create** コマンドを使用してオブジェクトを作成します。
3. 要求がアダプターに到達すると、アダプターはサービスが 3scale の API にどのようにマッピングされるかを認識している必要があります。この情報は、以下のいずれかの方法で提供できます。
   a. ワークロードにラベルを付ける (推奨)
   b. ハンドラーを **service_id** としてハードコーディングする
4. 必要なアノテーションでワークロードを更新します。

注記

ハンドラーにまだ組み込まれていない場合は、このサンプルで提供されたサービス ID のみを更新する必要があります。ハンドラーの設定が優先されます。

```
$ 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. アダプター経由でのサービストラフィックのルーティング

以下の手順に従って、3scale アダプターを使用してサービスのトラフィックを処理します。

前提条件

- 3scale 管理者から受け取る認証情報とサービス ID

手順

1. **kind: rule** リソース内で、以前に設定で作成した **destination.labels["service-mesh.3scale.net/credentials"] == "threescale"** ルールと一致させます。
2. 上記のラベルを、ターゲットワークロードのデプロイメントで **PodTemplateSpec** に追加し、サービスを統合します。値 **threescale** は生成されたハンドラーの名前を参照します。このハンドラーは、3scale を呼び出すのに必要なアクセストークンを保存します。
3. **destination.labels["service-mesh.3scale.net/service-id"] == "replace-me"** ラベルをワークロードに追加し、要求時にサービス ID をインスタンス経由でアダプターに渡します。

1.21.2. 3scale での統合設定

以下の手順に従って、3scale の統合設定を行います。

注記

3scale SaaS を使用している場合、Red Hat OpenShift Service Mesh は Early Access プログラムの一部として有効にされています。

手順

1. **[your_API_name] → Integration** の順に移動します。
2. **Settings** をクリックします。
3. **Deployment** で **Istio** オプションを選択します。
   - デフォルトでは **Authentication** の **API Key (user_key)** オプションが選択されます。

4. **Update Product** をクリックして選択内容を保存します。

5. **Configuration** をクリックします。

6. 設定の更新 をクリックします。

1.21.3. キャッシング動作

3scale System API からの応答は、アダプター内でデフォルトでキャッシュされます。**cacheTTLSeconds** 値よりも古いと、エントリーはキャッシュから消去されます。また、デフォルトでキャッシュされたエントリーの自動更新は、**cacheRefreshSeconds** 値に基づいて、期限が切れる前に数秒間試行されます。**cacheTTLSeconds** 値よりも高い値を設定することで、自動更新を無効にできます。

cacheEntriesMax を正の値以外に設定すると、キャッシングを完全に無効にできます。

更新プロセスを使用すると、到達不能になるホストのキャッシュされた値が、期限が切れて最終的に消去される前に再試行されます。

1.21.4. 認証要求

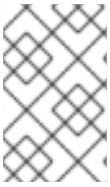
本リリースでは、以下の認証方法をサポートします。

- 標準 **API** キー: 単一のランダム文字列またはハッシュが識別子およびシークレットトークンとして機能します。
- アプリケーション **ID** とキーのペア: イミュータブルな識別子とミュータブルなシークレットキー文字列。
- **OpenID** 認証方法: JSON Web トークンから解析されるクライアント ID 文字列。

1.21.4.1. 認証パターンの適用

以下の認証方法の例に従って **instance** カスタムリソースを変更し、認証動作を設定します。認証情報は、以下から受け取ることができます。

- 要求ヘッダー
- 要求パラメーター
- 要求ヘッダーとクエリーパラメーターの両方



注記

ヘッダーの値を指定する場合、この値は小文字である必要があります。たとえば、ヘッダーを **User-Key** として送信する必要がある場合、これは設定で **request.headers["user-key"]** として参照される必要があります。

1.21.4.1.1. API キー認証方法

Service Mesh は、**subject** カスタムリソースパラメーターの **user** オプションで指定されたクエリーパラメーターと要求ヘッダーで API キーを検索します。これは、カスタムリソースファイルで指定される順序で値をチェックします。不要なオプションを省略することで、API キーの検索をクエリーパラメーターまたは要求ヘッダーに制限できます。

この例では、Service Mesh は **user_key** クエリーパラメーターの API キーを検索します。API キーがクエリーパラメーターにない場合、Service Mesh は **x-user-key** ヘッダーを確認します。

API キー認証方法の例

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

アダプターが異なるクエリーパラメーターまたは要求ヘッダーを検査するようにする場合は、名前を適宜変更します。たとえば、key というクエリーパラメーターの API キーを確認するには、**request.query_params["user_key"]** を **request.query_params["key"]** に変更します。

1.21.4.1.2. アプリケーション ID およびアプリケーションキーペアの認証方法

Service Mesh は、**subject** カスタムリソースパラメーターの **properties** オプションで指定されるように、クエリーパラメーターと要求ヘッダーでアプリケーション ID とアプリケーションキーを検索します。アプリケーションキーはオプションです。これは、カスタムリソースファイルで指定される順序で値をチェックします。不要なオプションを含めないことで、認証情報の検索をクエリーパラメーターまたは要求ヘッダーのいずれかに制限できます。

この例では、Service Mesh は最初にクエリーパラメーターのアプリケーション ID とアプリケーションキーを検索し、必要に応じて要求ヘッダーに移動します。

アプリケーション ID およびアプリケーションキーペアの認証方法の例

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

アダプターが異なるクエリーパラメーターまたは要求ヘッダーを検査するようにする場合は、名前を適宜変更します。たとえば、**identification** という名前のクエリーパラメーターのアプリケーション ID を確認するには、**request.query_params["app_id"]** を **request.query_params["identification"]** に変更します。

1.21.4.1.3. OpenID 認証方法

OpenID Connect (OIDC) 認証方法 を使用するには、**subject** フィールドで **properties** 値を使用して **client_id** および任意で **app_key** を設定します。

このオブジェクトは、前述の方法を使用して操作することができます。以下の設定例では、クライアント識別子 (アプリケーション ID) は、**azp** ラベルの JSON Web Token (JWT) から解析されます。これは必要に応じて変更できます。

OpenID 認証方法の例

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

この統合を正常に機能させるには、クライアントがアイデンティティープロバイダー (IdP) で作成されるよう OIDC を 3scale で実行する必要があります。保護するサービスと同じ namespace でサービスの に要求の認証 を作成する必要があります。JWT は要求の **Authorization** ヘッダーに渡されます。

以下に定義されるサンプル **RequestAuthentication** で、**issuer**、**jwksUri**、および **selector** を適宜置き換えます。

OpenID Policy の例

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: bookinfo
spec:
  selector:
    matchLabels:
      app: productpage
  jwtRules:
  - issuer: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak
    jwksUri: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak/protocol/openid-
connect/certs
```

1.21.4.1.4. ハイブリッド認証方法

特定の認証方法を適用せず、いずれかの方法の有効な認証情報を受け入れる方法を選択できます。API キーとアプリケーション ID/アプリケーションキーペアの両方が提供される場合、Service Mesh は API キーを使用します。

この例では、Service Mesh がクエリーパラメーターの API キーをチェックし、次に要求ヘッダーを確認します。API キーがない場合、クエリーパラメーターのアプリケーション ID とキーをチェックし、次に要求ヘッダーを確認します。

ハイブリッド認証方法の例

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

1.21.5. 3scale アダプターメトリクス

アダプターはデフォルトで、**/metrics** エンドポイントのポート **8080** で公開されるさまざまな Prometheus メトリクスを報告します。これらのメトリクスから、アダプターと 3scale 間の対話方法についての洞察が提供されます。サービスには、自動的に検出され、Prometheus によって収集されるようにラベルが付けられます。

注記

3scale Istio Adapter メトリクスには、Service Mesh 1.x の以前のリリース以降、互換性のない変更があります。

Prometheus では、以下のメトリクスが Service Mesh 2.0 の時点で使用できるように、バックエンドキャッシュの 1 つの追加と共にメトリクスの名前が変更されています。

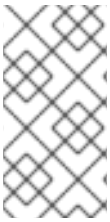
表1.31 Prometheus メトリクス

メトリクス	タイプ	説明
threescale_latency	ヒストグラム	アダプターと 3scale 間の要求レイテンシーです。
threescale_http_total	カウンター	3scale バックエンドへの要求についての HTTP ステータスの応答コード。

メトリクス	タイプ	説明
threescale_system_cache_hits	カウンター	設定キャッシュからフェッチされる 3scale システムへの要求の合計数。
threescale_backend_cache_hits	カウンター	バックエンドキャッシュからフェッチされる 3scale バックエンドへの要求の合計数。

1.21.6. 3scale バックエンドキャッシュ

3scale バックエンドキャッシュは、3scale Service Management API のクライアントの認証およびレポートキャッシュを提供します。このキャッシュはアダプターに組み込まれ、管理者がトレードオフを受け入れることが予想される特定の状況での応答の低レイテンシーが可能になります。



注記

3scale バックエンドキャッシュはデフォルトで無効にされます。3scale バックエンドキャッシュ機能では、低レイテンシーとプロセッサーおよびメモリーのリソースの高い使用率と引き換えに、速度制限における不正確な状況が生じたり、フラッシュの最後の実行からのヒットを失う可能性があります。

1.21.6.1. バックエンドキャッシュを有効にする利点

バックエンドキャッシュを有効にする利点には以下が含まれます。

- 3scale Istio Adapter が管理するサービスへのアクセス時にレイテンシーが高くなる場合にバックエンドキャッシュを有効にします。
- バックエンドキャッシュを有効にすると、3scale API マネージャーで要求の認証について継続的にチェックされなくなり、これによりレイテンシーが短縮されます。
  - これにより、3scale API マネージャーにアクセスして認証を試行する前に、3scale Istio アダプターが保存し、再利用する 3scale 認証のインメモリーキャッシュが作成されます。これにより、認証の許可または拒否にかかる時間が大幅に少なくなります。
- バックエンドキャッシュは、3scale Istio アダプターを実行するサービスメッシュとは異なる地理的な場所で 3scale API マネージャーをホストする場合に役立ちます。
  - 通常、これは 3scale のホスト型 (SaaS) プラットフォームに該当しますが、ユーザーが異なるアベイラビリティーゾーンの地理的に異なる場所にある別のクラスターで 3scale API マネージャーをホストする場合や、3scale API Manager に到達するためのネットワークのオーバーヘッドを考慮する必要がある場合にも使用できます。

1.21.6.2. 低レイテンシーを確保するためのトレードオフ

以下は、低レイテンシーを確保するためのトレードオフです。

- フラッシュが発生するたびに、3scale アダプターの承認状態が更新されます。
  - つまり、アダプターの 2 つ以上のインスタンスで、フラッシュ期間の間隔についての不正確な状態が生じます。

  - 要求が過剰になり、制限を超過し、誤った動作を生じさせ、さらには各要求を処理するアダプターによって処理される要求と処理されない要求とが発生する高い可能性があります。

- データをフラッシュできず、その認証情報を更新できないアダプターキャッシュは、その情報を API マネージャーに報告せずにシャットダウンまたはクラッシュする可能性があります。

- アダプターのキャッシュで、API マネージャーと通信する際のネットワーク接続が原因と予想される問題などにより、要求を許可/拒否する必要があるかどうかを判別できない場合に、fail open または fail closed ポリシーが適用されます。

- キャッシュミスが発生すると、通常はアダプターの起動直後、または接続なしの状態が長く続いた後に、API マネージャーのクエリーを行うためにレイテンシーが増加します。

- アダプターキャッシュでは、キャッシュを有効にしない場合よりも、認証の計算により多くの作業が必要になります。これにより、より多くのプロセッサーリソースが必要になります。

- メモリー要件は、キャッシュで管理される制限、アプリケーションおよびサービスの量の組み合わせに比例して増加します。

1.21.6.3. バックエンドキャッシュ設定

以下では、バックエンドキャッシュの設定について説明します。

- バックエンドキャッシュを設定するための設定内容については、3scale 設定オプションを参照してください。

- 最後の 3 つの設定では、バックエンドキャッシュの有効化を制御します。

  - **PARAM_USE_CACHE_BACKEND**: バックエンドキャッシュを有効にするには true に設定します。

  - **PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS**: キャッシュデータの API マネージャーへのフラッシュの試行間の時間 (秒単位) を設定します。

  - **PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED**: キャッシュされたデータが十分になく、3scale API マネージャーに到達できない場合にサービスへの要求を許可/オープン/または拒否/クローズするかどうかについて設定します。

1.21.7. 3scale Istio Adapter APIcast エミュレーション

3scale Istio アダプターは、以下の条件が満たされる場合に APIcast と同様に動作します。

- 要求が定義されるマッピングルールと一致しない場合、返される HTTP コードは 404 Not Found になります。これは、以前は 403 Forbidden でした。

- 要求が制限を超えるために拒否されると、返される HTTP コードは 429 Too Many Requests になります。これは、以前は 403 Forbidden でした。

- CLI でデフォルトのテンプレートを生成する場合、ヘッダーにはハイフンではなくアンダースコアが使用されます (例: **user-key** ではなく **user_key** が使用されます)。

1.21.8. 3scale Istio Adapter の検証

3scale Istio Adapter が予想通りに機能しているかどうかを確認します。アダプターが機能しない場合は、以下の手順に従って問題のトラブルシューティングを行うことができます。

手順

1. **3scale-adapter** Pod が Service Mesh コントロールプレーン namespace で実行されていることを確認します。

```
$ oc get pods -n <istio-system>
```

2. そのバージョンなど、**3scale-adapter** Pod が起動に関する情報を出力したことを確認します。

```
$ oc logs <istio-system>
```

3. 3scale Adapter の統合で保護されているサービスに対して要求を実行すると、正しい認証情報がかけているという要求を必ず試し、その要求が失敗することを確認します。3scale Adapter ログをチェックして、追加情報を収集します。

関連情報

- Pod およびコンテナーログの検査

1.21.9. 3scale Istio adapter のトラブルシューティングのチェックリスト

管理者が 3scale Istio adapter をインストールすると、統合が適切に機能しなくなる可能性のあるシナリオが複数あります。以下の一覧を使用して、インストールのトラブルシューティングを行います。

- YAML のインデントが間違っている。
- YAML セクションがない。
- YAML の変更をクラスターに適用するのを忘れている。
- **service-mesh.3scale.net/credentials** キーでサービスのワークロードにラベルを付けるのを忘れている。
- **service_id** が含まれないハンドラーを使用してアカウントごとに再利用できるようにする時に **service-mesh.3scale.net/service-id** サービスワークロードにラベルを付けるのを忘れている。
- **Rule** カスタムリソースが誤ったハンドラーまたはインスタンスカスタムリソースを参照しているか、対応する namespace の接尾辞がかけている参照を指定している。
- **Rule** カスタムリソースの **match** セクションは、設定中のサービスと同じでない可能性があるか、現在実行中でない、または存在しない宛先ワークロードを参照している。
- ハンドラーの 3scale 管理ポータルのアクセストークンまたは URL が正しくない。
- クエリーパラメーター、ヘッダー、承認要求などの誤った場所を指定しているか、パラメーター名がテストで使用する要求と一致しないため、インスタンス のカスタムリソースの **params/subject/properties** セクションで、 **app_id**、**app_key** または **client_id** の正しいパラメーターの表示に失敗する。
- 設定ジェネレーターがアダプターコンテナーイメージに実際に存在しており、**oc exec** で呼び出す必要があることに気づかなかったため、設定ジェネレーターの使用に失敗する。

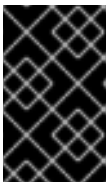
1.22. サービスメッシュのトラブルシューティング

このセクションでは、Red Hat OpenShift Service Mesh で一般的な問題を特定し、解決する方法を説明します。以下のセクションを使用して、OpenShift Container Platform に Red Hat OpenShift Service Mesh をデプロイする際の問題のトラブルシューティングおよびデバッグに役立ちます。

1.22.1. Service Mesh のバージョンについて

ご使用のシステムにデプロイした Red Hat OpenShift Service Mesh のバージョンを理解するには、各コンポーネントのバージョンがどのように管理されるかを理解する必要があります。

- **Operator** バージョン: 最新の Operator バージョンは 2.2.3 です。Operator バージョン番号は、現在インストールされている Operator のバージョンのみを示します。Red Hat OpenShift Service Mesh Operator は Service Mesh コントロールプレーンの複数のバージョンをサポートするため、Operator のバージョンはデプロイされた **ServiceMeshControlPlane** リソースのバージョンを決定しません。



重要

最新の Operator バージョンにアップグレードすると、パッチの更新が自動的に適用されますが、Service Mesh コントロールプレーンは最新のマイナーバージョンに自動的にアップグレードされません。

- **ServiceMeshControlPlane** バージョン: **ServiceMeshControlPlane** バージョンは、使用している Red Hat OpenShift Service Mesh のバージョンを決定します。**ServiceMeshControlPlane** リソースの **spec.version** フィールドの値は、Red Hat OpenShift Service Mesh のインストールとデプロイに使用されるアーキテクチャーと設定を制御します。Service Mesh コントロールプレーンを作成する場合は、以下の 2 つの方法のいずれかでバージョンを設定できます。
  - Form View で設定するには、**Control Plane Version** メニューからバージョンを選択します。
  - YAML View で設定するには、YAML ファイルに **spec.version** の値を設定します。

Operator Lifecycle Manager (OLM) は Service Mesh コントロールプレーンのアップグレードをを管理しないため、SMCP を手動でアップグレードしない限り、Operator および **ServiceMeshControlPlane** (SMCP) のバージョン番号が一致しない可能性があります。

1.22.2. Operator インストールのトラブルシューティング

このセクションの情報に加えて、次のトピックを確認してください。

- Operator について
- Operator Lifecycle Management の概念。
- OpenShift Operator のトラブルシューティングセクション。
- OpenShift インストールのトラブルシューティングセクション。

1.22.2.1. Operator インストールの検証

Red Hat OpenShift Service Mesh Operator のインストール時に、OpenShift は正常な Operator インストールの一部として以下のオブジェクトを自動的に作成します。

- Config Map

- カスタムリソース定義
- デプロイメント
- pods
- レプリカセット
- roles
- ロールバインディング
- secrets
- サービスアカウント
- services

OpenShift Container Platform コンソールからの使用

OpenShift Container Platform コンソールを使用して Operator Pod が利用可能であり、実行していることを確認できます。

1. **Workloads → Pods** に移動します。
2. **openshift-operators** namespace を選択します。
3. 以下の Pod が存在し、ステータスが **running** であることを確認します。
   - **istio-operator**
   - **jaeger-operator**
   - **kiali-operator**
4. **openshift-operators-redhat** namespace を選択します。
5. **elasticsearch-operator** Pod が存在し、ステータスが **running** であることを確認します。

コマンドラインで以下を行います。

1. 以下のコマンドを使用して、Operator Pod が利用可能で、**openshift-operators** namespace で実行していることを確認します。

```
$ oc get pods -n openshift-operators
```

出力例

```
NAME                                READY   STATUS    RESTARTS   AGE
istio-operator-bb49787db-zgr87      1/1     Running   0          15s
jaeger-operator-7d5c4f57d8-9xphf    1/1     Running   0          2m42s
kiali-operator-f9c8d84f4-7xh2v      1/1     Running   0          64s
```

2. 以下のコマンドを使用して Elasticsearch Operator を確認します。

```
$ oc get pods -n openshift-operators-redhat
```

出力例

```
NAME                                  READY   STATUS    RESTARTS   AGE
elasticsearch-operator-d4f59b968-796vq    1/1     Running   0          15s
```

1.22.2.2. サービスメッシュ Operator のトラブルシューティング

Operator に問題が発生した場合は、以下を実行します。

- Operator サブスクリプションのステータスを確認します。
- サポートされる Red Hat バージョンではなく、コミュニティーバージョンの Operator をインストールしていないことを確認します。
- Red Hat OpenShift Service Mesh をインストールするために **cluster-admin** ロールがあることを確認します。
- 問題が Operator のインストールに関連する場合は、Operator Pod ログでエラーの有無を確認します。

注記

Operator は OpenShift コンソールからのみインストールでき、OperatorHub はコマンドラインからアクセスできません。

1.22.2.2.1. Operator Pod ログの表示

oc logs コマンドを使用して、Operator ログを表示できます。Red Hat は、サポートケースの解決に役立つログをリクエストする場合があります。

手順

- Operator Pod ログを表示するには、以下のコマンドを入力します。

```
$ oc logs -n openshift-operators <podName>
```

以下に例を示します。

```
$ oc logs -n openshift-operators istio-operator-bb49787db-zgr87
```

1.22.3. コントロールプレーンのトラブルシューティング

Service Mesh コントロールプレーン は Istiod で設定されており、以前のいくつかのコントロールプレーンコンポーネント (Citadel、Galley、Pilot) を単一バイナリーに統合します。**ServiceMeshControlPlane** をデプロイすると、アーキテクチャー で説明されているように、Red Hat OpenShift Service Mesh を設定する他のコンポーネントも作成します。

1.22.3.1. Service Mesh コントロールプレーンのインストールの検証

Service Mesh コントロールプレーンの作成時に、Service Mesh Operator は **ServiceMeshControlPlane** リソースファイルに指定したパラメーターを使用して以下を実行します。

- Istio コンポーネントを作成し、以下の Pod をデプロイします。

  - **istiod**
  - **istio-ingressgateway**
  - **istio-egressgateway**
  - **grafana**
  - **prometheus**
  - **wasm-cacher**

- SMCP または Kiali カスタムリソースのいずれかの設定に基づいて Kaili デプロイメントを作成するには、Kiali Operator を呼び出します。

> **注記**
>
> Service Mesh Operator ではなく、Kiali Operator で Kiali コンポーネントを表示します。

- Red Hat OpenShift 分散トレースプラットフォーム Operator を呼び出して、SMCP または Jaeger カスタムリソースのいずれかでの設定に基づいて分散トレースプラットフォームコンポーネントを作成します。

> **注記**
>
> Jaeger コンポーネントは Red Hat OpenShift 分散トレースプラットフォーム Operator の下に表示され、Elasticsearch コンポーネントは Service Mesh Operator ではなく Red Hat Elasticsearch Operator の下に表示されます。

OpenShift Container Platform コンソールからの使用

OpenShift Container Platform Web コンソールで Service Mesh コントロールプレーンのインストールを確認できます。

1. **Operators → Installed Operators** に移動します。
2. **<istio-system>** namespace を選択します。
3. Red Hat OpenShift Service Mesh Operator を選択します。
   a. **Istio Service Mesh Control Plane**タブをクリックします。
   b. コントロールプレーンの名前 (**basic** など) をクリックします。
   c. デプロイメントによって作成されたリソースを表示するには、**Resources** タブをクリックします。フィルターを使用してビューを絞り込むことができます。たとえば、すべての **Pod** のステータスが **running** になっていることを確認できます。
   d. SMCP のステータスが問題を示す場合は、YAML ファイルの **status:** 出力で詳細を確認してください。
   e. **Operators → Installed Operators** に戻ります。
4. OpenShift Elasticsearch Operator を選択します。

a. **Elasticsearch** タブをクリックします。

b. デプロイメントの名前をクリックします (例: **elasticsearch**)。

c. デプロイメントによって作成されたリソースを表示するには、**Resources** タブをクリックします。

d. ステータス 列に問題がある場合は、**YAML** タブの ステータス**:** 出力で詳細を確認してください。

e. **Operators → Installed Operators** に戻ります。

5. Red Hat OpenShift 分散トレースプラットフォーム Operator を選択します。

a. **Jaeger** タブをクリックします。

b. デプロイメントの名前をクリックします (例: **jaeger**)。

c. デプロイメントによって作成されたリソースを表示するには、**Resources** タブをクリックします。

d. ステータス 列に問題がある場合は、**YAML** タブの **status:** 出力で詳細を確認してください。

e. **Operators → Installed Operators** に移動します。

6. Kiali Operator を選択します。

a. **Istio Service Mesh Control Plane**タブをクリックします。

b. デプロイメントの名前をクリックします (例: **kiali**)。

c. デプロイメントによって作成されたリソースを表示するには、**Resources** タブをクリックします。

d. ステータス 列に問題がある場合は、**YAML** タブの ステータス**:** 出力で詳細を確認してください。

コマンドラインで以下を行います。

1. 以下のコマンドを実行して、Service Mesh コントロールプレーン Pod が利用可能かどうか確認します。**istio-system** は SMCP をインストールした namespace になります。

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

出力例

```
NAME                                READY   STATUS    RESTARTS   AGE
grafana-6776785cfc-6fz7t            2/2     Running   0          102s
istio-egressgateway-5f49dd99-l9ppq  1/1     Running   0          103s
istio-ingressgateway-6dc885c48-jjd8r  1/1   Running   0          103s
istiod-basic-6c9cc55998-wg4zq       1/1     Running   0          2m14s
jaeger-6865d5d8bf-zrfss             2/2     Running   0          100s
kiali-579799fbb7-8mwc8              1/1     Running   0          46s
prometheus-5c579dfb-6qhjk           2/2     Running   0          115s
wasm-cacher-basic-5b99bfcddb-m775l  1/1     Running   0          86s
```

2. 次のコマンドを使用して、Service Mesh コントロールプレーンのデプロイのステータスを確認します。**istio-system** は、SMCP をデプロイした namespace に置き換えます。

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

STATUS 列が **ComponentsReady** の場合、インストールは正常に終了しています。

出力例

```
NAME    READY   STATUS           PROFILES      VERSION   AGE
basic   10/10   ComponentsReady   ["default"]   2.1.3     4m2s
```

Service Mesh コントロールプレーンを変更および再デプロイする場合、ステータスは **UpdateSuccessful** が表示されるはずです。

出力例

```
NAME           READY     STATUS            TEMPLATE   VERSION   AGE
basic-install   10/10     UpdateSuccessful   default     v1.1      3d16h
```

3. SMCP のステータスが **ComponentsReady** 以外の場合は、SCMP リソースの **status:** 出力で詳細を確認してください。

```
$ oc describe smcp <smcp-name> -n <controlplane-namespace>
```

出力例

```
$ oc describe smcp basic -n istio-system
```

4. 以下のコマンドを使用して、Jaeger デプロイメントのステータスを確認します。ここでは、**istio-system** は SMCP をデプロイした namespace に置き換えます。

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

出力例

```
NAME     STATUS    VERSION   STRATEGY   STORAGE   AGE
jaeger   Running   1.30.0    allinone   memory    15m
```

5. 以下のコマンドを使用して、Kiali デプロイメントのステータスを確認します。ここでは、**istio-system** は SMCP をデプロイした namespace に置き換えます。

```
$ oc get kiali -n <istio-system>
```

出力例

```
NAME    AGE
kiali   15m
```

1.22.3.1.1. Kiali コンソールへのアクセス

Kiali コンソールでアプリケーションのトポロジー、健全性、およびメトリクスを表示できます。サービスで問題が発生した場合には、Kiali コンソールは、サービス経由でデータフローを表示できます。抽象アプリケーションからサービスおよびワークロードまで、さまざまなレベルでのメッシュコンポーネントに関する洞察を得ることができます。Kiali は、リアルタイムで namespace のインタラクティブなグラフビューも提供します。

Kiali コンソールにアクセスするには、Red Hat OpenShift Service Mesh がインストールされ、Kiali がインストールおよび設定されている必要があります。

インストールプロセスにより、Kiali コンソールにアクセスするためのルートが作成されます。

Kiali コンソールの URL が分かっている場合は、直接アクセスできます。URL が分からない場合は、以下の指示を使用します。

管理者の手順

1. 管理者ロールで OpenShift Container Platform Web コンソールにログインします。
2. **Home → Projects** をクリックします。
3. **Projects** ページで、必要に応じてフィルターを使用してプロジェクトの名前を検索します。
4. プロジェクトの名前をクリックします (例: **bookinfo**)。
5. **Project details** ページの **Launcher** セクションで、**Kiali** リンクをクリックします。
6. OpenShift Container Platform コンソールにアクセスするときに使用するものと同じユーザー名とパスワードを使用して Kiali コンソールにログインします。
   初回の Kiali コンソールへのログイン時に、表示するパーミッションを持つサービスメッシュ内のすべての namespace を表示する **Overview** ページが表示されます。

   コンソールのインストールを検証中で、namespace がまだメッシュに追加されていない場合、**istio-system** 以外のデータは表示されない可能性があります。

開発者の手順

1. 開発者ロールで OpenShift Container Platform Web コンソールにログインします。
2. **Project** をクリックします。
3. 必要に応じて、**Project Details** ページで、フィルターを使用してプロジェクトの名前を検索します。
4. プロジェクトの名前をクリックします (例: **bookinfo**)。
5. **Project** ページの **Launcher** セクションで、**Kiali** リンクをクリックします。
6. **Log In With OpenShift** をクリックします。

1.22.3.1.2. Jaeger コンソールへのアクセス

Jaeger コンソールにアクセスするには、Red Hat OpenShift Service Mesh がインストールされ、Red Hat OpenShift 分散トレースプラットフォームがインストールおよび設定されている必要があります。

インストールプロセスにより、Jaeger コンソールにアクセスするためのルートが作成されます。

Jaeger コンソールの URL が分かっている場合は、これに直接アクセスできます。URL が分からない場合は、以下の指示を使用します。

OpenShift コンソールからの手順

1. cluster-admin 権限を持つユーザーとして OpenShift Container Platform Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
2. **Networking** → **Routes** に移動します。
3. **Routes** ページで、**Namespace** メニューから Service Mesh コントロールプレーンプロジェクトを選択します (例: **istio-system**)。
   Location 列には、各ルートのリンク先アドレスが表示されます。
4. 必要な場合は、フィルターを使用して **jaeger** ルートを検索します。ルートの **Location** をクリックしてコンソールを起動します。
5. **Log In With OpenShift** をクリックします。

Kiali コンソールからの手順

1. Kiali コンソールを起動します。
2. 左側のナビゲーションペインで **Distributed Tracing** をクリックします。
3. **Log In With OpenShift** をクリックします。

CLI からの手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

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

2. コマンドラインを使用してルートの詳細をクエリーするには、以下のコマンドを入力します。この例では、**istio-system** が Service Mesh コントロールプレーンの namespace です。

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

3. ブラウザーを起動し、**https://<JAEGER_URL>** に移動します。ここで、**<JAEGER_URL>** は直前の手順で検出されたルートです。
4. OpenShift Container Platform コンソールへアクセスするときに使用するものと同じユーザー名とパスワードを使用してログインします。
5. サービスメッシュにサービスを追加し、トレースを生成している場合は、フィルターと **Find Traces** ボタンを使用してトレースデータを検索します。
   コンソールインストールを検証すると、表示するトレースデータはありません。

1.22.3.2. Service Mesh コントロールプレーンのトラブルシューティング

Service Mesh コントロールプレーンのデプロイ時に問題が発生した場合は、

- **ServiceMeshControlPlane** リソースがサービスおよび Operator とは別のプロジェクトにインストールされていることを確認します。本書では **istio-system** プロジェクトをサンプルとして使用しますが、Operator およびサービスが含まれるプロジェクトから分離されている限り、コントロールプレーンを任意のプロジェクトにデプロイすることができます。

- **ServiceMeshControlPlane** および **Jaeger** カスタムリソースが同じプロジェクトにデプロイされていることを確認します。たとえば、両方の **istio-system** プロジェクトを使用します。

1.22.4. データプレーンのトラブルシューティング

データプレーン は、サービスメッシュ内のサービス間の受信および送信ネットワーク通信をすべて傍受し、制御するインテリジェントプロキシーのセットです。

Red Hat OpenShift Service Mesh は、アプリケーションの Pod 内のプロキシーサイドカーに依存して、アプリケーションにサービスメッシュ機能を提供します。

1.22.4.1. サイドカーインジェクションのトラブルシューティング

Red Hat OpenShift Service Mesh は、プロキシーサイドカーコンテナーを Pod に自動的に挿入しません。サイドカーインジェクションをオプトインする必要があります。

1.22.4.1.1. Istio サイドカーインジェクションのトラブルシューティング

アプリケーションのデプロイメントで自動インジェクションが有効になっているかどうかを確認します。Envoy プロキシーの自動インジェクションが有効になっている場合は、**spec.template.metadata.annotations** の下の **Deployment** リソースに **sidecar.istio.io/inject:"true"** アノテーションがなければなりません。

1.22.4.1.2. Jaeger エージェントのサイドカーインジェクションのトラブルシューティング

アプリケーションのデプロイメントで自動インジェクションが有効になっているかどうかを確認します。Jaeger エージェントの自動インジェクションが有効な場合は、**Deployment** リソースに **sidecar.jaegertracing.io/inject:"true"** アノテーションが必要です。

サイドカーインジェクションの詳細は、自動インジェクションの有効化 を参照してください。

1.23. ENVOY プロキシーのトラブルシューティング

Envoy プロキシーは、サービスメッシュ内の全サービスの受信トラフィックおよび送信トラフィックをすべてインターセプトします。Envoy はサービスメッシュでテレメトリーを収集し、報告します。Envoy は、同じ Pod の関連するサービスに対してサイドカーコンテナーとしてデプロイされます。

1.23.1. Envoy アクセスログの有効化

Envoy アクセスログは、トラフィックの障害およびフローの診断に役立ち、エンドツーエンドのトラフィックフロー分析に役立ちます。

すべての istio-proxy コンテナーのアクセスロギングを有効にするには、**ServiceMeshControlPlane** (SMCP) オブジェクトを編集してロギングの出力のファイル名を追加します。

手順

1. cluster-admin ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。以下のコマンドを入力します。次に、プロンプトが表示されたら、ユーザー名とパスワードを入力します。

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

2. Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) に切り替えます。

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

3. **ServiceMeshControlPlane** ファイルを編集します。

```
$ oc edit smcp <smcp_name>
```

4. 以下の例で示すように、**name** を使用してプロキシーログのファイル名を指定します。**name** の値を指定しないと、ログエントリーは書き込まれません。

```
spec:
  proxy:
    accessLogging:
      file:
        name: /dev/stdout     #file name
```

Pod の問題のトラブルシューティングについての詳細は、Investigating Pod issues を参照してください。

1.23.2. サポート

本書で説明されている手順、または OpenShift Container Platform で問題が発生した場合は、Red Hat カスタマーポータル にアクセスしてください。カスタマーポータルでは、次のことができます。

- Red Hat 製品に関するアーティクルおよびソリューションについての Red Hat ナレッジベースの検索またはブラウズ。
- Red Hat サポートに対するサポートケースの送信。
- その他の製品ドキュメントへのアクセス。

クラスターの問題を特定するには、OpenShift Cluster Manager で Insights を使用できます。Insights により、問題の詳細と、利用可能な場合は問題の解決方法に関する情報が提供されます。

本書の改善への提案がある場合、またはエラーを見つけた場合は、最も関連性の高いドキュメントコンポーネントの Jira Issue を送信してください。セクション名や OpenShift Container Platform バージョンなどの具体的な情報を提供してください。

1.23.2.1. Red Hat ナレッジベースについて

Red Hat ナレッジベース は、お客様が Red Hat の製品やテクノロジーを最大限に活用できるようにするための豊富なコンテンツを提供します。Red Hat ナレッジベースは、Red Hat 製品のインストール、設定、および使用に関する記事、製品ドキュメント、および動画で設定されています。さらに、簡潔な根本的な原因についての説明や修正手順を説明した既知の問題のソリューションを検索できます。

1.23.2.2. Red Hat ナレッジベースの検索

OpenShift Container Platform の問題が発生した場合には、初期検索を実行して、解決策を Red Hat ナレッジベース内ですでに見つけることができるかどうかを確認できます。

前提条件

- Red Hat カスタマーポータルのアカウントがある。

手順

1. Red Hat カスタマーポータル にログインします。
2. 主な Red Hat カスタマーポータルの検索フィールドには、問題に関連する入力キーワードおよび文字列を入力します。これらには、以下が含まれます。
   - OpenShift Container Platform コンポーネント (**etcd** など)
   - 関連する手順 (**installation** など)
   - 明示的な失敗に関連する警告、エラーメッセージ、およびその他の出力
3. **Search** をクリックします。
4. **OpenShift Container Platform** 製品フィルターを選択します。
5. ナレッジベース のコンテンツタイプフィルターを選択します。

1.23.2.3. must-gather ツールについて

oc adm must-gather CLI コマンドは、以下のような問題のデバッグに必要となる可能性のあるクラスターからの情報を収集します。

- リソース定義
- サービスログ

デフォルトで、**oc adm must-gather** コマンドはデフォルトのプラグインイメージを使用し、**./must-gather.local** に書き込みを行います。

または、以下のセクションで説明されているように、適切な引数を指定してコマンドを実行すると、特定の情報を収集できます。

- 1 つ以上の特定の機能に関連するデータを収集するには、以下のセクションに示すように、イメージと共に **--image** 引数を使用します。
  以下に例を示します。

  ```
  $ oc adm must-gather  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.9.0
  ```

- 監査ログを収集するには、以下のセクションで説明されているように **-- /usr/bin/gather_audit_logs** 引数を使用します。
  以下に例を示します。

  ```
  $ oc adm must-gather -- /usr/bin/gather_audit_logs
  ```

注記

ファイルのサイズを小さくするために、監査ログはデフォルトの情報セットの一部として収集されません。

oc adm must-gather を実行すると、ランダムな名前を持つ新規 Pod がクラスターの新規プロジェクトに作成されます。データは Pod で収集され、**must-gather.local** で始まる新規ディレクトリーに保存されます。このディレクトリーは、現行の作業ディレクトリーに作成されます。

以下に例を示します。

```
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. サービスメッシュデータの収集について

oc adm must-gather CLI コマンドを使用してクラスターについての情報を収集できます。これには、Red Hat OpenShift Service Mesh に関連する機能およびオブジェクトが含まれます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift Container Platform CLI (**oc**) がインストールされていること。

手順

1. **must-gather** で Red Hat OpenShift Service Mesh データを収集するには、Red Hat OpenShift Service Mesh イメージを指定する必要があります。

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

2. **must-gather** で特定の Service Mesh コントロールプレーン namespace の Red Hat OpenShift Service Mesh データを収集するには、Red Hat OpenShift Service Mesh イメージおよび namespace を指定する必要があります。この例では、**<namespace>** を **istio-system** などの Service Mesh コントロールプレーンの namespace に置き換えます。

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

迅速なサポートを得るには、OpenShift Container Platform と Red Hat OpenShift Service Mesh の両方の診断情報を提供してください。

1.23.2.5. サポートケースの送信

前提条件

- OpenShift CLI (**oc**) がインストールされている。

- Red Hat カスタマーポータルのアカウントがある。
- OpenShift Cluster Manager にアクセスできる。

手順

1. Red Hat カスタマーポータル にログインし、**SUPPORT CASES → Open a case** を選択します。
2. 問題の該当するカテゴリー (**Defect / Bug** など)、製品 (**OpenShift Container Platform**)、および製品バージョン (すでに自動入力されていない場合は **4.7**) を選択します。
3. 報告されている問題に対する一致に基づいて提案される Red Hat ナレッジベースソリューションの一覧を確認してください。提案されている記事が問題に対応していない場合は、**Continue** をクリックします。
4. 問題についての簡潔で説明的な概要と、確認されている現象および予想される動作についての詳細情報を入力します。
5. 報告されている問題に対する一致に基づいて提案される Red Hat ナレッジベースソリューションの更新された一覧を確認してください。ケース作成プロセスでより多くの情報を提供すると、この一覧の絞り込みが行われます。提案されている記事が問題に対応していない場合は、**Continue** をクリックします。
6. アカウント情報が予想通りに表示されていることを確認し、そうでない場合は適宜修正します。
7. 自動入力された OpenShift Container Platform クラスター ID が正しいことを確認します。正しくない場合は、クラスター ID を手動で取得します。
   - OpenShift Container Platform Web コンソールを使用してクラスター ID を手動で取得するには、以下を実行します。
     a. **Home → Dashboards → Overview** に移動します。
     b. **Details** セクションの **Cluster ID** フィールドで値を見つけます。
   - または、OpenShift Container Platform Web コンソールで新規サポートケースを作成し、クラスター ID を自動的に入力することができます。
     a. ツールバーから、**(?) Help → Open Support Case** に移動します。
     b. **Cluster ID** 値が自動的に入力されます。
   - OpenShift CLI (**oc**) を使用してクラスター ID を取得するには、以下のコマンドを実行します。

     ```
     $ oc get clusterversion -o jsonpath='{.items[].spec.clusterID}{"\n"}'
     ```

8. プロンプトが表示されたら、以下の質問に入力し、**Continue** をクリックします。
   - 動作はどこで発生しているか？どの環境を使用しているか？
   - 動作はいつ発生するか？頻度は？繰り返し発生するか？特定のタイミングで発生するか？
   - 時間枠およびビジネスへの影響について提供できるどのような情報があるか？

9. 関連する診断データファイルをアップロードし、**Continue** をクリックします。まず **oc adm must-gather** コマンドを使用して収集されるデータと、そのコマンドによって収集されない問題に固有のデータを含めることが推奨されます。

10. 関連するケース管理の詳細情報を入力し、**Continue** をクリックします。

11. ケースの詳細をプレビューし、**Submit** をクリックします。

1.24. SERVICE MESH コントロールプレーン設定の参照

デフォルトの **ServiceMeshControlPlane** (SMCP) リソースを変更するか、または完全にカスタムの SMCP リソースを作成して Red Hat OpenShift Service Mesh をカスタマイズできます。このリファレンスセクションでは、SMCP リソースで利用可能な設定オプションについて説明します。

1.24.1. Service Mesh コントロールプレーンのパラメーター

以下の表は、**ServiceMeshControlPlane** リソースのトップレベルのパラメーターを一覧表示しています。

表1.32 **ServiceMeshControlPlane** リソースパラメーター

名前	説明	タイプ
apiVersion	APIVersion はオブジェクトのこの表現のバージョンスキーマを定義します。サーバーは認識されたスキーマを最新の内部値に変換し、認識されない値は拒否することがあります。**ServiceMeshControlPlane** バージョン 2.0 の値は **maistra.io/v2** です。	**ServiceMeshControlPlane** バージョン 2.0 の値は **maistra.io/v2** です。
kind	kind はこのオブジェクトが表す REST リソースを表す文字列の値です。	ServiceMeshControlPlane で唯一有効な値は、**ServiceMeshControlPlane** です。
metadata	この **ServiceMeshControlPlane** インスタンスについてのメタデータ。Service Mesh コントロールプレーンインストールの名前を指定して作業を追跡できます (**basic** など)。	文字列
spec	この **ServiceMeshControlPlane** の必要な状態の仕様です。これには、Service Mesh コントロールプレーンを設定するすべてのコンポーネントの設定オプションが含まれます。	詳細は、表 2 を参照してください。

名前	説明	タイプ
status	この **ServiceMeshControlPlane** と Service Mesh コントロールプレーンを設定するコンポーネントの現在のステータスです。	詳細は、表 3 を参照してください。

以下の表は、**ServiceMeshControlPlane** リソースの仕様を一覧表示しています。これらのパラメーターを変更すると、Red Hat OpenShift Service Mesh コンポーネントが設定されます。

表1.33 **ServiceMeshControlPlane** リソース仕様

名前	説明	設定可能なパラメーター
addons	**addons** パラメーターを使用して、可視化やメトリクスストレージなど、コアの Service Mesh コントロールプレーンコンポーネント以外の追加機能を設定します。	**3scale**、**grafana**、**jaeger**、**kiali**、および **prometheus**
cluster	**cluster** パラメーターは、クラスターの一般的な設定 (クラスター名、ネットワーク名、マルチクラスター、メッシュ拡張など) の設定を行います。	**meshExpansion**、**multiCluster**、**name**、および **network**
gateways	**gateways** パラメーターを使用して、メッシュの ingress および egress ゲートウェイを設定します。	**enabled**、**additionalEgress**、**additionalIngress**、**egress**、**ingress**、および **openshiftRoute**
general	**general** パラメーターは、その他の場所には適合しない一般的な Service Mesh コントロールプレーンの設定を表します。	**logging** および **validationMessages**
policy	**policy** パラメーターを使用して、Service Mesh コントロールプレーンのポリシーチェックを設定します。ポリシーチェックを有効にするには、**spec.policy.enabled** を **true** に設定します。	**mixer remote**、または **type**。**type** は **Istiod**、**Mixer** または **None** に設定できます。
profiles	**profiles** パラメーターを使用して、デフォルト値に使用するために **ServiceMeshControlPlane** プロファイルを選択します。	**default**

名前	説明	設定可能なパラメーター
proxy	**proxy** パラメーターを使用してサイドカーのデフォルト動作を設定します。	**accessLogging**、**adminPort**、**concurrency**、および **envoyMetricsService**
runtime	ランタイム パラメーターを使用して、Service Mesh コントロールプレーンコンポーネントを設定します。	**components**、および **defaults**
security	**security** パラメーターを使用すると、Service Mesh コントロールプレーンのセキュリティーの各種機能を設定できます。	**certificateAuthority**、**controlPlane**、**identity**、**data Plane** および **trust**
techPreview	**techPreview** パラメーターを使用すると、テクノロジープレビュー機能への早期アクセスが可能になります。	該当なし
telemetry	**spec.mixer.telemetry.enabled** が **true** に設定されている場合、telemetry は有効にされます。	**mixer**、**remote**、および **type**。**type** は **Istiod**、**Mixer** または **None** に設定できます。
tracing	**tracing** パラメーターを使用して、メッシュの分散トレースを有効にします。	**sampling**、**type**。**type** は **Jaeger** または **None** に設定できます。
version	**version** パラメーターは、インストールする Service Mesh コントロールプレーンの Maistra バージョンを指定します。空のバージョンで **ServiceMeshControlPlane** を作成する場合、受付 Webhook はバージョンを現行バージョンに設定します。空のバージョンの新規の **ServiceMeshControlPlanes** は **v2.0** に設定されます。空のバージョンの既存の **ServiceMeshControlPlanes** はそれらの設定を保持します。	文字列

ControlPlaneStatus はサービスメッシュの現在の状態を表します。

表1.34 **ServiceMeshControlPlane** リソース **ControlPlaneStatus**

名前	説明	タイプ
annotations	**annotations** パラメーターは、通常は **ServiceMeshControlPlane** によってデプロイされるコンポーネントの数などの追加の余分なステータス情報を保存します。これらのステータスは、JSONPath 式でオブジェクトのカウントを許可しないコマンドラインツールの **oc** で使用されます。	設定不可
conditions	オブジェクトの現在の状態として観察される最新の状態を表します。**Reconcile** は、Operator がデプロイされるコンポーネントの実際の状態の調整を **ServiceMeshControlPlane** リソースの設定を使用して完了したかどうかを示します。**Installed** は、Service Mesh コントロールプレーンがインストールされているかどうかを示します。**Ready** は、すべての Service Mesh コントロールプレーンコンポーネントの準備ができているかどうかを示します。	文字列
コンポーネント	デプロイされた各 Service Mesh コントロールプレーンコンポーネントのステータスを表示します。	文字列
appliedSpec	すべてのプロファイルが適用された後に生成される設定の仕様です。	**ControlPlaneSpec**
appliedValues	チャートの生成に使用される生成される values.yaml です。	**ControlPlaneSpec**
chartVersion	このリソースに対して最後に処理されたチャートのバージョンです。	文字列

名前	説明	タイプ
observedGeneration	直近の調整時にコントローラーによって観察される生成です。ステータスの情報は、オブジェクトの特定の生成に関連するものです。**status.conditions** は、**status.observedGeneration** フィールドが **metadata.generation** に一致しない場合は最新の状態ではありません。	整数
operatorVersion	このリソースを最後に処理した Operator のバージョンです。	文字列
readiness	コンポーネントおよび所有リソースの準備状態 (readiness) のステータス	文字列

この例の **ServiceMeshControlPlane** の定義には、サポート対象のパラメーターがすべて含まれます。

ServiceMeshControlPlane リソースの例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.2
  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. 仕様パラメーター

1.24.2.1. 一般的なパラメーター

以下の例は、**ServiceMeshControlPlane** オブジェクトの **spec.general** パラメーターと適切な値を持つ利用可能なパラメーターの説明を示しています。

一般的なパラメーターの例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  general:
    logging:
      componentLevels: {}
        # misc: error
      logAsJSON: false
    validationMessages: true
```

表1.35 Istio の一般的なパラメーター

パラメーター	説明	値	デフォルト値
`logging:`	Service Mesh コントロールプレーンコンポーネントのロギングを設定するために使用します。		該当なし
`logging: componentLevels:`	コンポーネントのロギングレベルを指定するために使用します。	使用できる値は、**trace**、**debug**、**info**、**warning**、**error**、**fatal**、**panic** です。	該当なし
`logging: logLevels:`	使用できる値は、**trace**、**debug**、**info**、**warning**、**error**、**fatal**、**panic** です。		該当なし
`logging: logAsJSON:`	JSON ロギングを有効または無効にします。	**true/false**	該当なし
`validationMessages:`	istio.io リソースの status フィールドへの検証メッセージを有効または無効にするのに使用します。これは、リソースで設定エラーを検出するのに役立ちます。	**true/false**	該当なし

1.24.2.2. プロファイルパラメーター

ServiceMeshControlPlane オブジェクトプロファイルを使用すると、再利用可能な設定を作成することができます。**profile** 設定を設定しない場合、Red Hat OpenShift Service Mesh は default プロファイルを使用します。

以下の例は、**ServiceMeshControlPlane** オブジェクトの **spec.profiles** パラメーターを示しています。

プロファイルパラメーターの例

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

プロファイルの作成に関する詳細は、コントロールプレーンプロファイルの作成 を参照してください。

セキュリティー設定の詳細な例は、Mutual Transport Layer Security (mTLS) を参照してください。

1.24.2.3. techPreview パラメーター

spec.techPreview パラメーターを使用すると、テクノロジープレビュー機能への早期アクセスが可能になります。

重要

テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

1.24.2.4. トレースパラメーター

以下の例は、**ServiceMeshControlPlane** オブジェクトの **spec.tracing** パラメーターと適切な値を持つ利用可能なパラメーターの説明を示しています。

トレースパラメーターの例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.2
  tracing:
    sampling: 100
    type: Jaeger
```

表1.36 Istio トレースパラメーター

パラメーター	説明	値	デフォルト値
tracing: sampling:	サンプリングレートは、Envoy プロキシーがトレースを生成する頻度を決定します。サンプリングレートを使用して、トレースシステムに報告される要求の割合を制御します。	0 から 10000 までの整数値で、インクリメントは 0.01% (0 から 100%) になります。たとえば、値を **10** に設定するとリクエストの 0.1% がサンプリングされ、値を **100** に設定するとリクエストの 1% がサンプリングされ、値を **500** に設定するとリクエストの 5% がサンプリングされ、**10000** に設定するとリクエストの 100% がサンプリングされます。	**10000** (トレースの 100%)
tracing: type:	現在、サポートされるトレーサーの唯一のタイプは **Jaeger** です。Jaeger はデフォルトで有効にされます。トレースを無効にするには、**type** パラメーターを **None** に設定します。	**none**、**Jaeger**	**Jaeger**

1.24.2.5. バージョンパラメーター

Red Hat OpenShift Service Mesh Operator は、さまざまなバージョンの **ServiceMeshControlPlane** のインストールをサポートしています。version パラメーターは、インストールする Service Mesh コントロールプレーンのバージョンを指定します。SMCP の作成時にバージョンパラメーターを指定しない場合には、Operator は値を最新バージョン (2.2) に設定します。既存の **ServiceMeshControlPlane** オブジェクトは、Operator のアップグレード中にバージョン設定を保持します。

1.24.2.6. 3scale の設定

以下の表では、**ServiceMeshControlPlane** リソースの 3scale Istio アダプターのパラメーターについて説明しています。

3scale パラメーターの例

```
spec:
  addons:
    3Scale:
      enabled: false
      PARAM_THREESCALE_LISTEN_ADDR: 3333
      PARAM_THREESCALE_LOG_LEVEL: info
      PARAM_THREESCALE_LOG_JSON: true
      PARAM_THREESCALE_LOG_GRPC: false
      PARAM_THREESCALE_REPORT_METRICS: true
```

```
PARAM_THREESCALE_METRICS_PORT: 8080
PARAM_THREESCALE_CACHE_TTL_SECONDS: 300
PARAM_THREESCALE_CACHE_REFRESH_SECONDS: 180
PARAM_THREESCALE_CACHE_ENTRIES_MAX: 1000
PARAM_THREESCALE_CACHE_REFRESH_RETRIES: 1
PARAM_THREESCALE_ALLOW_INSECURE_CONN: false
PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS: 10
PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS: 60
PARAM_USE_CACHED_BACKEND: false
PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS: 15
PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED: true
```

表1.37 3scale パラメーター

パラメーター	説明	値	デフォルト値
enabled	3scale アダプターを使用するかどうか	**true/false**	**false**
PARAM_THREESCALE_LISTEN_ADDR	gRPC サーバーのリッスンアドレスを設定します。	有効なポート番号	**3333**
PARAM_THREESCALE_LOG_LEVEL	ログ出力の最小レベルを設定します。	**debug**、**info**、**warn**、**error**、または **none**	**info**
PARAM_THREESCALE_LOG_JSON	ログが JSON としてフォーマットされるかどうかを制御します。	**true/false**	**true**
PARAM_THREESCALE_LOG_GRPC	ログに gRPC 情報を含むかどうかを制御します。	**true/false**	**true**
PARAM_THREESCALE_REPORT_METRICS	3scale システムおよびバックエンドメトリクスが収集され、Prometheus に報告されるかどうかを制御します。	**true/false**	**true**
PARAM_THREESCALE_METRICS_PORT	3scale /**metrics** エンドポイントをスクラップできるポートを設定します。	有効なポート番号	**8080**
PARAM_THREESCALE_CACHE_TTL_SECONDS	キャッシュから期限切れのアイテムを消去するまで待機する時間 (秒単位)。	時間 (秒単位)	**300**

パラメーター	説明	値	デフォルト値
PARAM_THREESCALE_CACHE_REFRESH_SECONDS	キャッシュ要素の更新を試行する場合の期限	時間 (秒単位)	**180**
PARAM_THREESCALE_CACHE_ENTRIES_MAX	キャッシュにいつでも保存できるアイテムの最大数。キャッシュを無効にするには **0** に設定します。	有効な数字	**1000**
PARAM_THREESCALE_CACHE_REFRESH_RETRIES	キャッシュ更新ループ時に到達できないホストが再試行される回数	有効な数字	**1**
PARAM_THREESCALE_ALLOW_INSECURE_CONN	**3scale** API 呼び出し時の証明書の検証を省略できるようにします。この有効化は推奨されていません。	**true/false**	**false**
PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS	3scale システムおよびバックエンドへの要求を終了するまで待機する秒数を設定します。	時間 (秒単位)	**10**
PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS	接続を閉じるまでの最大秒数 (+/-10% のジッター) を設定します。	時間 (秒単位)	60
PARAM_USE_CACHE_BACKEND	true の場合、承認要求のインメモリー apisonator キャッシュの作成を試行します。	**true/false**	**false**
PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS	バックエンドキャッシュが有効な場合には、3scale に対してキャッシュをフラッシュする間隔を秒単位で設定します。	時間 (秒単位)	15
PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED	バックエンドキャッシュが承認データを取得できない場合は常に、要求を拒否する (クローズする) か、許可する (オープンする) かどうか。	**true/false**	**true**

1.24.3. ステータスパラメーター

status パラメーターは、サービスメッシュの現在の状態を記述します。この情報は Operator によって生成され、読み取り専用です。

表1.38 Istio ステータスパラメーター

名前	説明	タイプ
observedGeneration	直近の調整時にコントローラーによって観察される生成です。ステータスの情報は、オブジェクトの特定の生成に関連するものです。**status.conditions** は、**status.observedGeneration** フィールドが **metadata.generation** に一致しない場合は最新の状態ではありません。	整数
annotations	**annotations** パラメーターは、通常は **ServiceMeshControlPlane** オブジェクトによってデプロイされるコンポーネントの数などの追加の余分なステータス情報を保存します。これらのステータスは、JSONPath 式でオブジェクトのカウントを許可しないコマンドラインツールの **oc** で使用されます。	設定不可
readiness	コンポーネントおよび所有リソースの readiness ステータスです。	文字列
operatorVersion	このリソースを最後に処理した Operator のバージョンです。	文字列
コンポーネント	デプロイされた各 Service Mesh コントロールプレーンコンポーネントのステータスを表示します。	文字列
appliedSpec	すべてのプロファイルが適用された後に生成される設定の仕様です。	**ControlPlaneSpec**

名前	説明	タイプ
conditions	オブジェクトの現在の状態として観察される最新の状態を表します。**Reconciled** は、Operator がデプロイされるコンポーネントの実際の状態の調整を **ServiceMeshControlPlane** リソースの設定を使用して完了したかどうかを示します。インストール 済みは、Service Mesh コントロールプレーンがインストールされていることを示します。**Ready** は、すべての Service Mesh コントロールプレーンコンポーネントの準備が整っていることを示します。	文字列
chartVersion	このリソースに対して最後に処理されたチャートのバージョンです。	文字列
appliedValues	チャートの生成に使用された、生成される **values.yaml** ファイル。	**ControlPlaneSpec**

1.24.4. 関連情報

- **ServiceMeshControlPlane** リソースで機能を設定する方法についての詳細は、以下のリンクを参照してください。
  - セキュリティー
  - トラフィック管理
  - メトリクスとトレース

1.25. KIALI 設定リファレンス

Service Mesh Operator は **ServiceMeshControlPlane** を作成する際に、Kiali リソースも処理します。次に Kiali Operator は Kiali インスタンスの作成時にこのオブジェクトを使用します。

1.25.1. SMCP での Kiali 設定の指定

Kiali は、**ServiceMeshControlPlane** リソースの **addons** セクションで設定できます。Kiali はデフォルトで有効です。Kiali を無効にするには、**spec.addons.kiali.enabled** を **false** に設定します。

Kiali 設定は、以下の 2 つの方法のいずれかで指定できます。

- **spec.addons.kiali.install** の **ServiceMeshControlPlane** リソースで Kiali 設定を指定します。Kiali 設定の完全なリストが SMCP で利用できないため、このアプローチにはいくつかの制限があります。

- Kiali インスタンスを設定してデプロイし、Kiali リソースの名前を **ServiceMeshControlPlane** リソースの **spec.addons.kiali.name** の値として指定します。CR は、Service Mesh コントロールプレーンと同じ namespace (例: **istio-system**) に作成する必要があります。**name** の値に一致する Kiali リソースが存在する場合には、コントロールプレーンは、そのコントロールプレーンで使用するために対象の Kiali リソースを設定します。このアプローチにより、Kiali リソースで Kiali 設定を完全にカスタマイズできます。このアプローチでは、Kiali リソースのさまざまなフィールド (例: **accessible_namespaces** リスト)、および Grafana、Prometheus、およびトレースのエンドポイントが上書きされることに注意してください。

Kiali の SMCP パラメーターの例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  addons:
    kiali:
      name: kiali
      enabled: true
      install:
        dashboard:
          viewOnly: false
          enableGrafana: true
          enableTracing: true
          enablePrometheus: true
        service:
          ingress:
            contextPath: /kiali
```

表1.39 **ServiceMeshControlPlane Kiali** パラメーター

パラメーター	説明	値	デフォルト値

`spec:`
`  addons:`
`  kiali:`
`  name:`
	Kiali カスタムリソースの名前。**name** の値に一致する Kiali CR が存在する場合には、Service Mesh Operator はその CR をインストールに使用します。Kiali CR が存在しない場合には、Operator はこの 名前 と SMCP で指定された設定オプションを使用して Kiali CR を作成します。	文字列	**kiali**

`kiali:` `  enabled:`	このパラメーターは、Kiali を有効または無効にします。Kiali はデフォルトで有効です。	**true/false**	**true**

パラメーター	説明	値	デフォルト値
kiali: install:	指定された Kiali リソースが存在しない場合は、Kiali リソースをインストールします。**addons.kiali.enabled** が **false** に設定されている場合は、**install** セクションは無視されます。		
kiali: install: dashboard:	Kiali に付属のダッシュボードの設定パラメーター。		
kiali: install: dashboard: viewOnly:	このパラメーターは、Kiali コンソールの表示専用 (view-only) モードを有効または無効にします。表示専用モードを有効にすると、ユーザーは Kiali コンソールを使用して Service Mesh を変更できなくなります。	**true/false**	**false**
kiali: install: dashboard: enableGrafana:	**spec.addons.grafana** 設定に基づいて設定された Grafana エンドポイント。	**true/false**	**true**
kiali: install: dashboard: enablePrometheus:	**spec.addons.prometheus** 設定に基づいて設定された Prometheus エンドポイント。	**true/false**	**true**
kiali: install: dashboard: enableTracing:	Jaeger カスタムリソース設定に基づいて設定されたトレースエンドポイント。	**true/false**	**true**

パラメーター	説明	値	デフォルト値
kiali: install: service:	Kiali インストールに関連付けられた Kubernetes サービスの設定パラメーター。		
kiali: install: service: metadata:	リソースに適用する追加のメタデータを指定するために使用します。	該当なし	該当なし
kiali: install: service: metadata: annotations:	コンポーネントのサービスに適用するアノテーションを追加で指定するために使用します。	文字列	該当なし
kiali: install: service: metadata: labels:	コンポーネントのサービスに適用するラベルを追加で指定するために使用します。	文字列	該当なし
kiali: install: service: ingress:	OpenShift Route を介してコンポーネントのサービスにアクセスする詳細を指定するために使用します。	該当なし	該当なし
kiali: install: service: ingress: metadata: annotations:	コンポーネントのサービス入力に適用する注アノテーションを追加で指定するために使用します。	文字列	該当なし
kiali: install: service: ingress: metadata: labels:	コンポーネントのサービス ingress に適用するラベルを追加で指定するために使用します。	文字列	該当なし

パラメーター	説明	値	デフォルト値
kiali: install: service: ingress: enabled:	コンポーネントに関連付けられたサービスの OpenShift ルートをカスタマイズするために使用します。	**true/false**	**true**
kiali: install: service: ingress: contextPath:	サービスへのコンテキストパスを指定するために使用します。	文字列	該当なし
install: service: ingress: hosts:	OpenShift ルートごとに単一のホスト名を指定するために使用します。空のホスト名は、ルートのデフォルトのホスト名を意味します。	文字列	該当なし
install: service: ingress: tls:	OpenShift ルートの TLS を設定するために使用します。		該当なし
kiali: install: service: nodePort:	コンポーネントのサービス **Values**。**<component>.service.nodePort.port** の **nodePort** を指定するために使用します	整数	該当なし

1.25.2. Kiali カスタムリソースでの Kiali 設定の指定

ServiceMeshControlPlane (SMCP) リソースではなく、Kiali カスタムリソース (CR) で Kiali を設定することにより、Kiali デプロイメントを完全にカスタマイズできます。この設定は SMCP の外部で指定されるため、外部 Kiali と呼ばれることもあります。

注記

ServiceMeshControlPlane と Kiali カスタムリソースを同じ namespace にデプロイする必要があります。たとえば、**istio-system** となります。

Kiali インスタンスを設定してデプロイしてから、SMCP リソースの **spec.addons.kiali.name** の値として Kiali リソースの **name** を指定できます。**name** の値に一致する Kiali CR が存在する場合、Service Mesh コントロールプレーンは既存のインストールを使用します。この方法では、Kiali 設定を完全にカ

スタマイズできます。

1.26. JAEGER 設定リファレンス

Service Mesh Operator は **ServiceMeshControlPlane** リソースをデプロイする際に、分散トレースのリソースを作成することもできます。Service Mesh は分散トレースに Jaeger を使用します。

1.26.1. トレースの有効化および無効化

ServiceMeshControlPlane リソースでトレースタイプおよびサンプリングレートを指定して、分散トレースを有効にします。

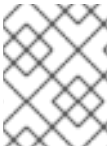
デフォルトの all-in-one Jaeger パラメーター

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.2
  tracing:
    sampling: 100
    type: Jaeger
```

現在、サポートされるトレーサーの唯一のタイプは **Jaeger** です。

Jaeger はデフォルトで有効にされます。トレースを無効にするには、**type** を **None** に設定します。

サンプリングレートは、Envoy プロキシーがトレースを生成する頻度を決定します。サンプリングレートオプションを使用して、トレースシステムに報告される要求の割合を制御できます。この設定は、メッシュ内のトラフィックおよび収集するトレースデータ量に基づいて設定できます。**sampling** は 0.01% の増分を表すスケーリングされた整数として設定します。たとえば、値を **10** サンプル (0.1% トレース)、および **500** サンプル (5% トレース)、および **10000** サンプル (100% トレース) に設定します。



注記

SMCP サンプリング設定オプションは Envoy サンプリングレートを制御します。Jaeger トレースサンプリングレートを Jaeger カスタムリソースで設定します。

1.26.2. SMCP での Jaeger 設定の指定

Jaeger は、**ServiceMeshControlPlane** リソースの **addons** セクションで設定します。ただし、SMCP で設定可能な内容にはいくつかの制限があります。

SMCP が設定情報を Red Hat OpenShift 分散トレースプラットフォーム Operator に渡すと、**allInOne**、**production**、または **streaming** の 3 つのデプロイメントストラテジーのいずれかがトリガーされます。

1.26.3. 分散トレースプラットフォームのデプロイ

分散トレースプラットフォームには、事前に定義されたデプロイメントストラテジーがあります。Jaeger カスタムリソース (CR) ファイルでデプロイメントストラテジーを指定します。分散トレースプラットフォームインスタンスの作成時に、Red Hat OpenShift 分散トレースプラットフォーム Operator

はこの設定ファイルを使用してデプロイメントに必要なオブジェクトを作成します。

Red Hat OpenShift 分散トレースプラットフォーム Operator は現時点で以下のデプロイメントストラテジーをサポートします。

- **allInOne** (デフォルト): このストラテジーは、開発、テスト、およびデモを目的としたものであり、実稼働での使用を目的としたものではありません。主なバックエンドコンポーネントである Agent、Collector、および Query サービスはすべて、インメモリーストレージを使用するように (デフォルトで) 設定された単一の実行可能ファイルにパッケージ化されます。このデプロイメントストラテジーは、SMCP で設定できます。

> **注記**
>
> インメモリーストレージには永続性がありません。つまり、Jaeger インスタンスがシャットダウンするか、再起動するか、または置き換えられると、トレースデータが失われます。各 Pod には独自のメモリーがあるため、インメモリーストレージはスケーリングできません。永続ストレージの場合、デフォルトのストレージとして Elasticsearch を使用する **production** または **streaming** ストラテジーを使用する必要があります。

- **production**: production ストラテジーは、実稼働環境向けのストラテジーであり、トレースデータの長期の保存が重要となり、より拡張性および高可用性のあるアーキテクチャーも必要になります。そのため、バックエンドの各コンポーネントは別々にデプロイされます。エージェントは、インストルメント化されたアプリケーションのサイドカーとして挿入できます。Query および Collector サービスは、サポートされているストレージタイプ (現時点では Elasticsearch) で設定されます。これらの各コンポーネントの複数のインスタンスは、パフォーマンスと回復性を確保するために、必要に応じてプロビジョニングできます。このデプロイメントストラテジーを SMCP に設定できますが、完全にカスタマイズするには、Jaeger CR で設定を指定し、SMCP にリンクする必要があります。

- **streaming**: streaming ストラテジーは、Collector と Elasticsearch バックエンドストレージ間に配置されるストリーミング機能を提供することで、production ストラテジーを増強する目的で設計されています。これにより、負荷の高い状況でバックエンドストレージに加わる圧力を軽減し、他のトレース処理後の機能がストリーミングプラットフォーム (AMQ Streams/ Kafka) から直接リアルタイムのスパンデータを利用できるようにします。このデプロイメントストラテジーを SMCP で設定することはできません。Jaeger CR を設定し、SMCP へのリンクを設定する必要があります。

> **注記**
>
> streaming ストラテジーには、AMQ Streams 用の追加の Red Hat サブスクリプションが必要です。

1.26.3.1. デフォルトの分散トレースプラットフォームのデプロイ

Jaeger 設定オプションを指定しない場合、**ServiceMeshControlPlane** リソースはデフォルトで **allInOne** Jaeger デプロイメントストラテジーを使用します。デフォルトの **allInOne** デプロイメントストラテジーを使用する場合は、**spec.addons.jaeger.install.storage.type** を **Memory** に設定します。デフォルトを使用するか、または **install** で追加設定オプションを許可できます。

コントロールプレーンのデフォルト Jaeger パラメーター (Memory)

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
```

```
  name: basic
spec:
  version: v2.2
  tracing:
    sampling: 10000
    type: Jaeger
  addons:
    jaeger:
      name: jaeger
      install:
        storage:
          type: Memory
```

1.26.3.2. 分散トレースプラットフォームの実稼働デプロイメント (最小)

production デプロイメントストラテジーのデフォルト設定を使用するには、**spec.addons.jaeger.install.storage.type** を **Elasticsearch** に設定し、**install** で追加設定オプションを指定します。SMCP は Elasticsearch リソースおよびイメージ名の設定のみをサポートすることに注意してください。

コントロールプレーンのデフォルト Jaeger パラメーター (Elasticsearch)

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.2
  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. 分散トレースプラットフォームの実稼働デプロイメント (完全にカスタマイズ)

SMCP は最小限の Elasticsearch パラメーターのみをサポートします。実稼働環境を完全にカスタマイズし、すべての Elasticsearch 設定パラメーターにアクセスするには、Jaeger カスタムリソース (CR) を使用して Jaeger を設定します。

または、Jaeger インスタンスを作成および設定し、**spec.addons.jaeger.name** を Jaeger インスタンスの名前 (この例では **MyJaegerInstance**) に設定できます。

Jaeger production CR がリンクされたコントロールプレーン

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.2
  tracing:
    sampling: 1000
    type: Jaeger
  addons:
    jaeger:
      name: MyJaegerInstance #name of Jaeger CR
      install:
        storage:
          type: Elasticsearch
        ingress:
          enabled: true
```

1.26.3.4. Jaeger デプロイメントのストリーミング

streaming デプロイメントストラテジーを使用するには、まず Jaeger インスタンスを作成および設定してから、**spec.addons.jaeger.name** を Jaeger インスタンスの名前 (この例では **MyJaegerInstance**) に設定します。

リンクされた Jaeger ストリーミング CR を使用したコントロールプレーン

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.2
  tracing:
    sampling: 1000
    type: Jaeger
  addons:
    jaeger:
      name: MyJaegerInstance  #name of Jaeger CR
```

1.26.4. Jaeger カスタムリソースでの Jaeger 設定の指定

ServiceMeshControlPlane (SMCP) リソースではなく Jaeger カスタムリソース (CR) に Jaeger を設定し、Jaeger デプロイメントを完全にカスタマイズすることができます。この設定は SMCP の外部に指定されているため、外部 Jaeger と呼ばれることもあります。

注記

SMCP と Jaeger CR を同じ namespace にデプロイする必要があります。たとえば、**istio-system** となります。

スタンドアロンの Jaeger インスタンスを設定し、デプロイしてから、Jaeger リソースの **name** を、SMCP リソースの **spec.addons.jaeger.name** の値として指定できます。**name** の値に一致する Jaeger CR が存在する場合、Service Mesh コントロールプレーンは既存のインストールを使用します。この方

法では、Jaeger 設定を完全にカスタマイズできます。

1.26.4.1. デプロイメントのベストプラクティス

- Red Hat OpenShift 分散トレースインスタンスの名前は一意でなければなりません。複数の Red Hat OpenShift 分散トレースプラットフォームインスタンスがあり、サイドカーが挿入されたエージェントを使用している場合、Red Hat OpenShift 分散トレースプラットフォームインスタンスには一意の名前が必要となり、挿入 (injection) のアノテーションはトレースデータを報告する必要のある Red Hat OpenShift 分散トレースプラットフォームインスタンスの名前を明示的に指定する必要があります。

- マルチテナントの実装があり、テナントが namespace で分離されている場合、Red Hat OpenShift 分散トレースプラットフォームインスタンスを各テナント namespace にデプロイします。

  - デーモンセットとしてのエージェントは、マルチテナントインストールまたは Red Hat OpenShift Dedicated ではサポートされません。サイドカーとしてのエージェントは、これらのユースケースでサポートされる唯一の設定です。

- Red Hat OpenShift Service Mesh の一部として分散トレースをインストールする場合、分散トレースリソースは、**ServiceMeshControlPlane** リソースと同じ namespace にインストールする必要があります。

永続ストレージの設定は、永続ストレージについて と、選択したストレージオプションに適した設定トピックを参照してください。

1.26.4.2. サービスメッシュの分散トレースセキュリティーの設定

分散トレーシングプラットフォームは、デフォルトの認証に OAuth を使用します。ただし、Red Hat OpenShift Service Mesh は **htpasswd** と呼ばれるシークレットを使用して、Grafana、Kiali、分散トレーシングプラットフォームなどの依存サービス間の通信を容易にします。**ServiceMeshControlPlane** で分散トレーシングプラットフォームを設定すると、Service Mesh は **htpasswd** を使用するようにセキュリティー設定を自動的に設定します。

Jaeger カスタムリソースで分散トレースプラットフォーム設定を指定している場合は、**htpasswd** 設定を手動で設定し、htpasswd シークレットが Jaeger インスタンスにマウントされていることを確認して、**Kiali** が Jaeger インスタンスと通信できるようにする必要があります。

1.26.4.2.1. OpenShift コンソールからのサービスメッシュの分散トレースセキュリティー設定

Jaeger リソースを変更して、OpenShift コンソールの Service Mesh で使用する分散トレースプラットフォームセキュリティーを設定できます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

- Red Hat OpenShift Service Mesh Operator がインストールされている必要がある。

- クラスターにデプロイされた **ServiceMeshControlPlane**。

- OpenShift Container Platform Web コンソールへのアクセスがある。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。
2. Operators → Installed Operators の順に移動します。
3. **Project** メニューをクリックし、一覧から **ServiceMeshControlPlane** リソースがデプロイされているプロジェクト (例: **istio-system**) を選択します。
4. **Red Hat OpenShift distributed tracing platform Operator**をクリックします。
5. **Operator** の詳細 ページで、**Jaeger** タブをクリックします。
6. Jaeger インスタンスの名前をクリックします。
7. Jaeger の詳細ページで、**YAML** タブをクリックして設定を変更します。
8. 次の例に示すように、**Jaeger** カスタムリソースファイルを編集して、**htpasswd** 設定を追加します。
   - **spec.ingress.openshift.htpasswdFile**
   - **spec.volumes**
   - **spec.volumeMounts**

   htpasswd 設定を示す Jaeger リソースの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
spec:
  ingress:
    enabled: true
    openshift:
      htpasswdFile: /etc/proxy/htpasswd/auth
      sar: '{"namespace": "istio-system", "resource": "pods", "verb": "get"}'
    options: {}
    resources: {}
    security: oauth-proxy
  volumes:
    - name: secret-htpasswd
      secret:
        secretName: htpasswd
    - configMap:
        defaultMode: 420
        items:
          - key: ca-bundle.crt
            path: tls-ca-bundle.pem
        name: trusted-ca-bundle
        optional: true
      name: trusted-ca-bundle
  volumeMounts:
    - mountPath: /etc/proxy/htpasswd
      name: secret-htpasswd
    - mountPath: /etc/pki/ca-trust/extracted/pem/
      name: trusted-ca-bundle
      readOnly: true
```

9. **Save** をクリックします。

1.26.4.2.2. コマンドラインからのサービスメッシュの分散トレースセキュリティーの設定

Jaeger リソースを変更して、**oc** ユーティリティーを使用してコマンドラインから Service Mesh で使用する分散トレースプラットフォームセキュリティーを設定できます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
- Red Hat OpenShift Service Mesh Operator がインストールされている必要がある。
- クラスターにデプロイされた **ServiceMeshControlPlane**。
- OpenShift Container Platform バージョンに一致する OpenShift CLI (oc) にアクセスできる。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

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

2. 次のコマンドを入力して、コントロールプレーンをインストールしたプロジェクト (**istio-system** など) に変更します。

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

3. 次のコマンドを実行して Jaeger カスタムリソースファイルを編集します。ここで、**jaeger.yaml** は Jaeger カスタムリソースの名前に置き換えます。

```
$ oc edit -n tracing-system -f jaeger.yaml
```

4. 次の例に示すように、**Jaeger** カスタムリソースファイルを編集して、**htpasswd** 設定を追加します。

   - **spec.ingress.openshift.htpasswdFile**
   - **spec.volumes**
   - **spec.volumeMounts**

   htpasswd 設定を示す Jaeger リソースの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
spec:
  ingress:
    enabled: true
    openshift:
      htpasswdFile: /etc/proxy/htpasswd/auth
```

```
      sar: '{"namespace": "istio-system", "resource": "pods", "verb": "get"}'
    options: {}
    resources: {}
    security: oauth-proxy
  volumes:
    - name: secret-htpasswd
      secret:
        secretName: htpasswd
    - configMap:
        defaultMode: 420
        items:
          - key: ca-bundle.crt
            path: tls-ca-bundle.pem
        name: trusted-ca-bundle
        optional: true
      name: trusted-ca-bundle
  volumeMounts:
    - mountPath: /etc/proxy/htpasswd
      name: secret-htpasswd
    - mountPath: /etc/pki/ca-trust/extracted/pem/
      name: trusted-ca-bundle
      readOnly: true
```

5. 以下のコマンドを実行して変更を適用します。ここで、<jaeger.yaml> は Jaeger カスタムリソースの名前に置き換えます。

```
$ oc apply -n tracing-system -f <jaeger.yaml>
```

6. Pod のデプロイメントの進行状況を監視するには、次のコマンドを実行します。

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

1.26.4.3. 分散トレースのデフォルト設定オプション

Jaeger カスタムリソース (CR) は、分散トレースプラットフォームリソースの作成時に使用されるアーキテクチャーおよび設定を定義します。これらのパラメーターを変更して、分散トレースプラットフォームの実装をビジネスニーズに合わせてカスタマイズできます。

Jaeger 汎用 YAML の例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: name
spec:
  strategy: <deployment_strategy>
  allInOne:
    options: {}
    resources: {}
  agent:
    options: {}
    resources: {}
  collector:
    options: {}
```

```
    resources: {}
  sampling:
    options: {}
  storage:
    type:
    options: {}
  query:
    options: {}
    resources: {}
  ingester:
    options: {}
    resources: {}
  options: {}
```

表1.40 Jaeger パラメーター

パラメーター	説明	値	デフォルト値
apiVersion:		オブジェクトの作成時に使用する API バージョン。	**jaegertracing.io/v1**
jaegertracing.io/v1	**kind:**	作成する Kubernetes オブジェクトの種類を定義します。	**jaeger**
	metadata:	**name** 文字列、**UID**、およびオプションの **namespace** などのオブジェクトを一意に特定するのに役立つデータ。	
OpenShift Container Platform は **UID** を自動的に生成し、オブジェクトが作成されるプロジェクトの名前で **namespace** を完了します。	**name:**	オブジェクトの名前。	分散トレースプラットフォームインスタンスの名前。
jaeger-all-in-one-inmemory	**spec:**	作成するオブジェクトの仕様。	分散トレースプラットフォームインスタンスのすべての設定パラメーターが含まれます。すべての Jaeger コンポーネントの共通定義が必要な場合、これは **spec** ノードで定義されます。定義が個々のコンポーネントに関連する場合は、**spec/<component>** ノードに置かれます。

パラメーター	説明	値	デフォルト値
該当なし	**strategy:**	Jaeger デプロイメントストラテジー	**allInOne**、**production**、または **streaming**
allInOne	**allInOne:**	**allInOne** イメージは Agent、Collector、Query、Ingester、および Jaeger UI を単一 Pod にデプロイするため、このデプロイメントの設定は、コンポーネント設定を **allInOne** パラメーターの下でネストする必要があります。	
	agent:	Agent を定義する設定オプション。	
	collector:	Jaeger Collector を定義する設定オプション。	
	sampling:	トレース用のサンプリングストラテジーを定義する設定オプション。	
	storage:	ストレージを定義する設定オプション。すべてのストレージ関連のオプションは、**allInOne** または他のコンポーネントオプションではなく、**storage** に配置される必要があります。	
	query:	Query サービスを定義する設定オプション。	
	ingester:	Ingester サービスを定義する設定オプション。	

以下の YAML の例は、デフォルト設定を使用して Red Hat OpenShift 分散トレースプラットフォームのデプロイメントを作成するために最低限必要なものです。

最小限必要な dist-tracing-all-in-one.yaml の例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-all-in-one-inmemory
```

1.26.4.4. Jaeger Collector 設定オプション

Jaeger Collector は、トレーサーによってキャプチャーされたスパンを受信し、**production** ストラテジーを使用する場合はそれらを永続 Elasticsearch ストレージに書き込み、**streaming** ストラテジーを使用する場合は AMQ Streams に書き込むコンポーネントです。

Collector はステートレスであるため、Jaeger Collector のインスタンスの多くは並行して実行できます。Elasticsearch クラスターの場所を除き、Collector では設定がほとんど必要ありません。

表**1.41 Operator** によって使用される **Jaeger Collector** パラメーターを定義するためのパラメーター

パラメーター	説明	値
`collector:` `  replicas:`	作成する Collector レプリカの数を指定します。	整数 (例: **5**)。

表**1.42 Collector** に渡される設定パラメーター

パラメーター	説明	値

`spec:`
`  collector:`
`  options: {}`
	Jaeger Collector を定義する設定オプション。	

`options:`
`  collector:`
`  num-workers:`
	キューからプルするワーカーの数。	整数 (例: **50**)。

`options:`
`  collector:`
`  queue-size:`
	Collector キューのサイズ。	整数 (例: **2000**)。

`options:`
`  kafka:`
`  producer:`
`  topic: jaeger-spans`
	topic パラメーターは、Collector によってメッセージを生成するために使用され、Ingester によってメッセージを消費するために使用される Kafka 設定を特定します。	プロデューサーのラベル。

パラメーター	説明	値

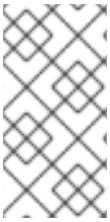
`options:`
`  kafka:`
`  producer:`
`  brokers: my-cluster-kafka-brokers.kafka:9092`
	メッセージを生成するために Collector によって使用される Kafka 設定を特定します。ブローカーが指定されていない場合で、AMQ Streams 1.4.0+ がインストールされている場合、Red Hat OpenShift 分散トレースプラットフォーム Operator は Kafka をセルフプロビジョニングします。	

`options:` `  log-level:`	Collector のロギングレベル。	使用できる値は、**debug**、**info**、**warn**、**error**、**fatal**、**panic** です。

1.26.4.5. 分散トレースのサンプリング設定オプション

この Red Hat OpenShift 分散トレースプラットフォーム Operator は、リモートサンプラーを使用するように設定されているトレーサーに提供されるサンプリングストラテジーを定義するために使用できます。

すべてのトレースが生成される間に、それらの一部のみがサンプリングされます。トレースをサンプリングすると、追加の処理や保存のためにトレースにマークが付けられます。



注記

これは、トレースがサンプリングの意思決定が行われる際に Envoy プロキシーによって開始されている場合には関連がありません。Jaeger サンプリングの意思決定は、トレースがクライアントを使用してアプリケーションによって開始される場合にのみ関連します。

サービスがトレースコンテキストが含まれていない要求を受信すると、クライアントは新しいトレースを開始し、これにランダムなトレース ID を割り当て、現在インストールされているサンプリングストラテジーに基づいてサンプリングの意思決定を行います。サンプリングの意思決定はトレース内の後続のすべての要求に伝播され、他のサービスが再度サンプリングの意思決定を行わないようにします。

分散トレースプラットフォームライブラリーは、以下のサンプラーをサポートします。

- **Probabilistic**: サンプラーは、**sampling.param** プロパティーの値と等しいサンプリングの確率で、ランダムなサンプリングの意思決定を行います。たとえば、**sampling.param=0.1** を使用した場合、約 10 のうち 1 トレースがサンプリングされます。
- **Rate Limiting**: サンプラーは、リーキーバケット (leaky bucket) レートリミッターを使用して、トレースが一定のレートでサンプリングされるようにします。たとえば、**sampling.param=2.0** を使用した場合、1 秒あたり 2 トレースの割合で要求がサンプリングされます。

表1.43 Jaeger サンプリングのオプション

パラメーター	説明	値	デフォルト値
spec: sampling: options: {} default_strategy: service_strategy:	トレース用のサンプリングストラテジーを定義する設定オプション。		設定を指定しない場合、Collector はすべてのサービスの確率 0.001 (0.1%) のデフォルトの確率的なサンプリングポリシーを返します。
default_strategy: type: service_strategy: type:	使用するサンプリングストラテジー。上記の説明を参照してください。	有効な値は **probabilistic**、および **ratelimiting** です。	**probabilistic**
default_strategy: param: service_strategy: param:	選択したサンプリングストラテジーのパラメーター	10 進値および整数値 (0、.1、1、10)	1

この例では、トレースインスタンスをサンプリングする確率が 50% の確率的なデフォルトサンプリングストラテジーを定義します。

確率的なサンプリングの例

```
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
```

ユーザーによって指定される設定がない場合、分散トレースプラットフォームは以下の設定を使用します。

デフォルトのサンプリング

```
spec:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 1
```

1.26.4.6. 分散トレースのストレージ設定オプション

spec.storage の下で Collector、Ingester、および Query サービスのストレージを設定します。これらの各コンポーネントの複数のインスタンスは、パフォーマンスと回復性を確保するために、必要に応じてプロビジョニングできます。

表1.44 分散トレースストレージを定義するために Red Hat OpenShift 分散トレースプラットフォーム Operator によって使用される一般的なストレージパラメーター

パラメーター	説明	値	デフォルト値

`spec:`
`  storage:`
`  type:`
	デプロイメントに使用するストレージのタイプ。	**memory** または **elasticsearch**メモリーストレージは、Pod がシャットダウンした場合にデータが永続化されないため、開発、テスト、デモ、および概念検証用の環境にのみに適しています。実稼働環境については、分散トレースプラットフォームは永続ストレージの Elasticsearch をサポートします。	**memory**

`storage:` `  secretname:`	シークレットの名前 (例:**tracing-secret**)。		該当なし
`storage:` `  options: {}`	ストレージを定義する設定オプション。		

表1.45 Elasticsearch インデックスクリーナーのパラメーター

パラメーター	説明	値	デフォルト値
storage: esIndexCleaner: enabled:	Elasticsearch ストレージを使用する場合、デフォルトでジョブが作成され、古いトレースをインデックスからクリーンアップします。このパラメーターは、インデックスクリーナージョブを有効または無効にします。	**true**/ **false**	**true**
storage: esIndexCleaner: numberOfDays:	インデックスの削除を待機する日数。	整数値	**7**
storage: esIndexCleaner: schedule:	Elasticsearch インデックスを消去する頻度についてのスケジュールを定義します。	cron 式	"55 23 * * *"

1.26.4.6.1. Elasticsearch インスタンスの自動プロビジョニング

Jaeger カスタムリソースをデプロイする場合には、Red Hat OpenShift 分散トレースプラットフォーム Operator は、OpenShift Elasticsearch Operator を使用して、カスタムリソースファイルの ストレージ セクションで提供される設定に基づいて Elasticsearch クラスターを作成します。以下の設定が設定されている場合には、Red Hat 分散トレースプラットフォーム Operator は Elasticsearch をプロビジョニングします。

- **spec.storage:type** は **elasticsearch** に設定されている
- **spec.storage.elasticsearch.doNotProvision** は **false** に設定されている
- **spec.storage.options.es.server-urls** が定義されていない。つまり、Red Hat Elasticsearch Operator によってプロビジョニングされていない Elasticsearch インスタンスへの接続がない。

Elasticsearch をプロビジョニングする場には、Red Hat OpenShift 分散トレースプラットフォーム Operator は、Elasticsearch カスタムリソース 名 を Jaeger カスタムリソースの **spec.storage.elasticsearch.name** の値に設定します。**spec.storage.elasticsearch.name** に値を指定しない場合には、Operator は **elasticsearch** を使用します。

制約

- namespace ごとにセルフプロビジョニングされた Elasticsearch インスタンスがある分散トレースプラットフォーム 1 つだけを使用できます。Elasticsearch クラスターは単一の 分散トレースプラットフォームインスタンスの専用のクラスターになります。
- namespace ごとに 1 つの Elasticsearch のみを使用できます。

注記

Elasticsearch を OpenShift ロギングの一部としてインストールしている場合、Red Hat OpenShift 分散トレースプラットフォーム Operator はインストールされた OpenShift Elasticsearch Operator を使用してストレージをプロビジョニングできます。

以下の設定パラメーターは、セルフプロビジョニングされた Elasticsearch インスタンスについてのものです。これは、OpenShift Elasticsearch Operator を使用して Red Hat OpenShift 分散トレースプラットフォーム Operator によって作成されるインスタンスです。セルフプロビジョニングされた Elasticsearch の設定オプションは、設定ファイルの **spec:storage:elasticsearch** の下で指定します。

表1.46 Elasticsearch リソース設定パラメーター

パラメーター	説明	値	デフォルト値
elasticsearch: properties: doNotProvision:	Elasticsearch インスタンスを Red Hat 分散トレースプラットフォーム Operator がプロビジョニングする必要があるかどうかを指定するために使用します。	**true/false**	**true**
elasticsearch: properties: name:	Elasticsearch インスタンスの名前。Red Hat OpenShift 分散トレースプラットフォーム Operator は、このパラメーターで指定された Elasticsearch インスタンスを使用して Elasticsearch に接続します。	文字列	**elasticsearch**
elasticsearch: nodeCount:	Elasticsearch ノードの数。高可用性を確保するには、少なくとも 3 つのノードを使用します。スプリットブレインの問題が生じる可能性があるため、2 つのノードを使用しないでください。	整数値。例: 概念実証用 = 1、最小デプロイメント = 3	3
elasticsearch: resources: requests: cpu:	ご使用の環境設定に基づく、要求に対する中央処理単位の数。	コアまたはミリコアで指定されます (例: 200m、0.5、1)。例: 概念実証用 = 500m、最小デプロイメント = 1	1

パラメーター	説明	値	デフォルト値
elasticsearch: resources: requests: memory:	ご使用の環境設定に基づく、要求に使用できるメモリー。	バイト単位で指定します (例:200Ki、50Mi、5Gi)。例: 概念実証用 = 1Gi、最小デプロイメント = 16Gi*	16Gi
elasticsearch: resources: limits: cpu:	ご使用の環境設定に基づく、中央処理単位数の制限。	コアまたはミリコアで指定されます (例: 200m、0.5、1)。例: 概念実証用 = 500m、最小デプロイメント = 1	
elasticsearch: resources: limits: memory:	ご使用の環境設定に基づく、利用可能なメモリー制限。	バイト単位で指定します (例:200Ki、50Mi、5Gi)。例: 概念実証用 = 1Gi、最小デプロイメント = 16Gi*	
elasticsearch: redundancyPolicy:	データレプリケーションポリシーは、Elasticsearch シャードをクラスター内のデータノードにレプリケートする方法を定義します。指定されていない場合、Red Hat OpenShift 分散トレースプラットフォーム Operator はノード数に基づいて最も適切なレプリケーションを自動的に判別します。	**ZeroRedundancy**(レプリカシャードなし)、**SingleRedundancy**(レプリカシャード 1 つ)、**MultipleRedundancy**(各インデックスはデータノードの半分に分散される)、**FullRedundancy**(各インデックスはクラスター内のすべてのデータノードに完全にレプリケートされます)	
elasticsearch: useCertManagement:	分散トレースプラットフォームが Red Hat の証明書管理機能を使用するかどうかを指定するために使用します。この機能は、OpenShift Container Platform 4.7 の Red Hat OpenShift 5.2 向けのロギングサブシステムに追加されており、新しい Jaeger デプロイメントに推奨の設定です。	**true/false**	**true**

パラメーター	説明	値	デフォルト値
	各 Elasticsearch ノードはこれより低い値のメモリー設定でも動作しますが、これは実稼働環境でのデプロイメントには推奨されません。実稼働環境で使用する場合、デフォルトで各 Pod に割り当てる設定を 16Gi 未満にすることはできず、Pod ごとに最大 64Gi を割り当てることを推奨します。		

実稼働ストレージの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 3
      resources:
        requests:
          cpu: 1
          memory: 16Gi
        limits:
          memory: 16Gi
```

永続ストレージを含むストレージの例:

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 1
      storage: (1)
        storageClassName: gp2
        size: 5Gi
      resources:
        requests:
          cpu: 200m
          memory: 4Gi
        limits:
          memory: 4Gi
      redundancyPolicy: ZeroRedundancy
```

(1) 永続ストレージの設定。この場合、AWS **gp2** のサイズは **5Gi** です。値の指定がない場合、分散トレースプラットフォームは **emptyDir** を使用します。OpenShift Elasticsearch Operator は、分散トレースプラットフォームインスタンスで削除されない**PersistentVolumeClaim** および **PersistentVolume** をプロビジョニングします。同じ名前および namespace で分散トレースプ

ラットフォームインスタンスを作成する場合は、同じボリュームをマウントできます。

1.26.4.6.2. 既存の Elasticsearch インスタンスへの接続

分散トレースを使用したストレージには、既存の Elasticsearch クラスターを使用できます。外部 Elasticsearch インスタンスとも呼ばれる既存の Elasticsearch クラスターは、Red Hat 分散トレースプラットフォーム Operator または Red Hat Operator によってインストールされなかったインスタンスです。

以下の設定が指定されている場合に、Jaeger カスタムリソースをデプロイすると、Red Hat 分散トレースプラットフォーム Operator は Elasticsearch をプロビジョニングしません。

- **spec.storage.elasticsearch.doNotProvision** が **true** に設定されている
- **spec.storage.options.es.server-urls** に値がある
- **spec.storage.elasticsearch.name** に値がある場合、または Elasticsearch インスタンス名が **elasticsearch** の場合。

Red Hat OpenShift 分散トレースプラットフォーム Operator は、**spec.storage.elasticsearch.name** で指定された Elasticsearch インスタンスを使用して Elasticsearch に接続します。

制約

- 分散トレースプラットフォームで OpenShift Container Platform ロギング Elasticsearch インスタンスを共有したり、再利用したりすることはできません。Elasticsearch クラスターは単一の分散トレースプラットフォームインスタンスの専用のクラスターになります。

注記

Red Hat は、外部 Elasticsearch インスタンスのサポートを提供しません。カスタマーポータル でテスト済み統合マトリックスを確認できます。

以下の設定パラメーターは、外部 Elasticsearch インスタンスとして知られる、既存の Elasticsearch インスタンス向けです。この場合、カスタムリソースファイルの **spec:storage:options:es** で、Elasticsearch の設定オプションを指定します。

表1.47 汎用 ES 設定パラメーター

パラメーター	説明	値	デフォルト値
`es:` `  server-urls:`	Elasticsearch インスタンスの URL。	Elasticsearch サーバーの完全修飾ドメイン名。	**http://elasticsearch.<namespace>.svc:9200**

パラメーター	説明	値	デフォルト値
es: max-doc-count:	Elasticsearch クエリーから返す最大ドキュメント数。これは集約にも適用されます。**es.max-doc-count** と **es.max-num-spans** の両方を設定する場合、Elasticsearch は 2 つの内の小さい方の値を使用します。		10000
es: max-num-spans:	[**非推奨**: 今後のリリースで削除されます。代わりに **es.max-doc-count** を使用してください。] Elasticsearch のクエリーごとに、1 度にフェッチするスパンの最大数。**es.max-num-spans** と **es.max-doc-count** の両方を設定する場合、Elasticsearch は 2 つの内の小さい方の値を使用します。		10000
es: max-span-age:	Elasticsearch のスパンについての最大ルックバック。		72h0m0s
es: sniffer:	Elasticsearch のスニファー設定。クライアントはスニッフィングプロセスを使用してすべてのノードを自動的に検索します。デフォルトでは無効にされています。	**true**/ **false**	**false**
es: sniffer-tls-enabled:	Elasticsearch クラスターに対してスニッフィングする際に TLS を有効にするためのオプション。クライアントはスニッフィングプロセスを使用してすべてのノードを自動的に検索します。デフォルトでは無効にされています。	**true**/ **false**	**false**

パラメーター	説明	値	デフォルト値
es: timeout:	クエリーに使用されるタイムアウト。ゼロに設定するとタイムアウトはありません。		0s
es: username:	Elasticsearch で必要なユーザー名。Basic 認証は、指定されている場合に CA も読み込みます。**es.password** も参照してください。		
es: password:	Elasticsearch で必要なパスワード。**es.username** も参照してください。		
es: version:	主要な Elasticsearch バージョン。指定されていない場合、値は Elasticsearch から自動検出されます。		0

表1.48 ES データレプリケーションパラメーター

パラメーター	説明	値	デフォルト値
es: num-replicas:	Elasticsearch のインデックスごとのレプリカ数。		1
es: num-shards:	Elasticsearch のインデックスごとのシャード数。		5

表1.49 ES インデックス設定パラメーター

パラメーター	説明	値	デフォルト値

パラメーター	説明	値	デフォルト値
es: create-index-templates:	**true** に設定されている場合、アプリケーションの起動時にインデックステンプレートを自動的に作成します。テンプレートが手動でインストールされる場合は、**false** に設定されます。	**true**/ **false**	**true**
es: index-prefix:	分散トレースプラットフォームインデックスのオプション接頭辞。たとえば、これを production に設定すると、production-tracing-*という名前のインデックスが作成されます。		

表1.50 ES バルクプロセッサー設定パラメーター

パラメーター	説明	値	デフォルト値
es: bulk: actions:	バルクプロセッサーがディスクへの更新のコミットを決定する前にキューに追加できる要求の数。		1000
es: bulk: flush-interval:	**time.Duration**: この後に、他のしきい値に関係なく一括要求がコミットされます。バルクプロセッサーのフラッシュ間隔を無効にするには、これをゼロに設定します。		200ms
es: bulk: size:	バルクプロセッサーがディスクへの更新をコミットするまでに一括要求が発生する可能性のあるバイト数。		5000000
es: bulk: workers:	一括要求を受信し、Elasticsearch にコミットできるワーカーの数。		1

表1.51 ES TLS 設定パラメーター

パラメーター	説明	値	デフォルト値
es: tls: ca:	リモートサーバーの検証に使用される TLS 認証局 (CA) ファイルへのパス。		デフォルトではシステムトラストストアを使用します。
es: tls: cert:	リモートサーバーに対するこのプロセスの特定に使用される TLS 証明書ファイルへのパス。		
es: tls: enabled:	リモートサーバーと通信する際に、トランスポート層セキュリティー (TLS) を有効にします。デフォルトでは無効にされています。	**true/ false**	**false**
es: tls: key:	リモートサーバーに対するこのプロセスの特定に使用される TLS 秘密鍵ファイルへのパス。		
es: tls: server-name:	リモートサーバーの証明書の予想される TLS サーバー名を上書きします。		
es: token-file:	ベアラートークンが含まれるファイルへのパス。このフラグは、指定されている場合は認証局 (CA) ファイルも読み込みます。		

表1.52 ES アーカイブ設定パラメーター

パラメーター	説明	値	デフォルト値
es-archive: bulk: actions:	バルクプロセッサーがディスクへの更新のコミットを決定する前にキューに追加できる要求の数。		0

パラメーター	説明	値	デフォルト値
es-archive: bulk: flush-interval:	**time.Duration**: この後に、他のしきい値に関係なく一括要求がコミットされます。バルクプロセッサーのフラッシュ間隔を無効にするには、これをゼロに設定します。		0s
es-archive: bulk: size:	バルクプロセッサーがディスクへの更新をコミットするまでに一括要求が発生する可能性のあるバイト数。		0
es-archive: bulk: workers:	一括要求を受信し、Elasticsearch にコミットできるワーカーの数。		0
es-archive: create-index-templates:	**true** に設定されている場合、アプリケーションの起動時にインデックステンプレートを自動的に作成します。テンプレートが手動でインストールされる場合は、**false** に設定されます。	**true**/ **false**	**false**
es-archive: enabled:	追加ストレージを有効にします。	**true**/ **false**	**false**
es-archive: index-prefix:	分散トレースプラットフォームインデックスのオプション接頭辞。たとえば、これを production に設定すると、production-tracing-*という名前のインデックスが作成されます。		
es-archive: max-doc-count:	Elasticsearch クエリーから返す最大ドキュメント数。これは集約にも適用されます。		0

パラメーター	説明	値	デフォルト値
es-archive: max-num-spans:	[**非推奨**: 今後のリリースで削除されます。代わりに **es-archive.max-doc-count** を使用してください。] Elasticsearch のクエリーごとに、1度にフェッチするスパンの最大数。		0
es-archive: max-span-age:	Elasticsearch のスパンについての最大ルックバック。		0s
es-archive: num-replicas:	Elasticsearch のインデックスごとのレプリカ数。		0
es-archive: num-shards:	Elasticsearch のインデックスごとのシャード数。		0
es-archive: password:	Elasticsearch で必要なパスワード。**es.username** も参照してください。		
es-archive: server-urls:	Elasticsearch サーバーのコンマ区切りの一覧。完全修飾 URL(例: **http://localhost:9200**) として指定される必要があります。		
es-archive: sniffer:	Elasticsearch のスニファー設定。クライアントはスニッフィングプロセスを使用してすべてのノードを自動的に検索します。デフォルトでは無効にされています。	**true**/ **false**	**false**

パラメーター	説明	値	デフォルト値
es-archive: sniffer-tls-enabled:	Elasticsearch クラスターに対してスニッフィングする際に TLS を有効にするためのオプション。クライアントはスニッフィングプロセスを使用してすべてのノードを自動的に検索します。デフォルトでは無効にされています。	**true**/ **false**	**false**
es-archive: timeout:	クエリーに使用されるタイムアウト。ゼロに設定するとタイムアウトはありません。		0s
es-archive: tls: ca:	リモートサーバーの検証に使用される TLS 認証局 (CA) ファイルへのパス。		デフォルトではシステムトラストストアを使用します。
es-archive: tls: cert:	リモートサーバーに対するこのプロセスの特定に使用される TLS 証明書ファイルへのパス。		
es-archive: tls: enabled:	リモートサーバーと通信する際に、トランスポート層セキュリティー (TLS) を有効にします。デフォルトでは無効にされています。	**true**/ **false**	**false**
es-archive: tls: key:	リモートサーバーに対するこのプロセスの特定に使用される TLS 秘密鍵ファイルへのパス。		
es-archive: tls: server-name:	リモートサーバーの証明書の予想される TLS サーバー名を上書きします。		

パラメーター	説明	値	デフォルト値
es-archive: token-file:	ベアラートークンが含まれるファイルへのパス。このフラグは、指定されている場合は認証局 (CA) ファイルも読み込みます。		
es-archive: username:	Elasticsearch で必要なユーザー名。Basic 認証は、指定されている場合に CA も読み込みます。**es-archive.password** も参照してください。		
es-archive: version:	主要な Elasticsearch バージョン。指定されていない場合、値は Elasticsearch から自動検出されます。		0

ボリュームマウントを含むストレージの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200
        index-prefix: my-prefix
        tls:
          ca: /es/certificates/ca.crt
    secretName: tracing-secret
  volumeMounts:
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
  volumes:
    - name: certificates
      secret:
        secretName: quickstart-es-http-certs-public
```

以下の例は、ボリュームからマウントされる TLS CA 証明書およびシークレットに保存されるユーザー/パスワードを使用して外部 Elasticsearch クラスターを使用する Jaeger CR を示しています。

外部 Elasticsearch の例:

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200 (1)
        index-prefix: my-prefix
        tls: (2)
          ca: /es/certificates/ca.crt
    secretName: tracing-secret (3)
  volumeMounts: (4)
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
  volumes:
    - name: certificates
      secret:
        secretName: quickstart-es-http-certs-public
```

1. デフォルト namespace で実行されている Elasticsearch サービスへの URL。
2. TLS 設定。この場合、CA 証明書のみを使用できますが、相互 TLS を使用する場合に es.tls.key および es.tls.cert を含めることもできます。
3. 環境変数 ES_PASSWORD および ES_USERNAME を定義するシークレット。kubectl create secret generic tracing-secret --from-literal=ES_PASSWORD=changeme --from-literal=ES_USERNAME=elastic により作成されます
4. すべてのストレージコンポーネントにマウントされるボリュームのマウントとボリューム。

1.26.4.7. Elasticsearch を使用した証明書の管理

Red Hat Elasticsearch Operator を使用して、証明書を作成および管理できます。Red Hat Elasticsearch Operator を使用して証明書を管理すると、複数の Jaeger Collector で単一の Elasticsearch クラスターを使用することもできます。

重要

Elasticsearch を使用した証明書の管理は、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。

テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

バージョン 2.4 以降、Red Hat OpenShift 分散トレースプラットフォーム Operator は、Elasticsearch カスタムリソースで次のアノテーションを使用して、証明書の作成を Red Hat Elasticsearch Operator に委譲します。

- **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"**

ここで、**<shared-es-node-name>** は Elasticsearch ノードの名前です。たとえば、 **custom-es** という名前の Elasticsearch ノードを作成する場合には、カスタムリソースは次の例のようになります。

アノテーションを表示する Elasticsearch CR の例

```
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
```

前提条件

- OpenShift Container Platform 4.7
- Red Hat OpenShift のロギングサブシステム: 5.2
- Elasticsearch ノードと Jaeger インスタンスは同じ namespace にデプロイする必要があります。(例: **traceing-system**)。

Jaeger カスタムリソースで **spec.storage.elasticsearch.useCertManagement** を **true** に設定して、証明書管理を有効にします。

useCertManagement を示す例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      name: custom-es
      doNotProvision: true
      useCertManagement: true
```

Red Hat OpenShift 分散トレースプラットフォーム Operator は、Elasticsearch カスタムリソース 名 を Jaeger カスタムリソースの **spec.storage.elasticsearch.name** の値に設定します。

証明書は Red Hat Operator によってプロビジョニングされ、Red Hat 分散トレースプラットフォーム Operator が証明書を挿入します。

Elasticsearch を OpenShift Container Platform で設定する方法については、ログストアの設定 または 分散トレースの設定とデプロイ を参照してください。

1.26.4.8. クエリー設定オプション

Query とは、ストレージからトレースを取得し、ユーザーインターフェイスをホストしてそれらを表示するサービスです。

表1.53 **Query** を定義するために **Red Hat OpenShift** 分散トレースプラットフォーム **Operator** によって使用されるパラメーター

パラメーター	説明	値	デフォルト値

`spec:`
`  query:`
`  replicas:`
	作成する Query レプリカの数を指定します。	整数 (例: **2**)	

表1.54 **Query** に渡される設定パラメーター

パラメーター	説明	値	デフォルト値

`spec:`
`  query:`
`  options: {}`
	Query サービスを定義する設定オプション。		

パラメーター	説明	値	デフォルト値
options: log-level:	Query のロギングレベル。	使用できる値は、**debug**、**info**、**warn**、**error**、**fatal**、**panic** です。	
options: query: base-path:	すべての jaeger-query HTTP ルートのベースパスは、root 以外の値に設定できます。たとえば、**/jaeger** ではすべての UI URL が **/jaeger** で開始するようになります。これは、リバースプロキシーの背後で jaeger-query を実行する場合に役立ちます。	/<path>	

Query 設定の例

```
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. Ingester 設定オプション

Ingester は、Kafka トピックから読み取り、Elasticsearch ストレージバックエンドに書き込むサービスです。**allInOne** または **production** デプロイメントストラテジーを使用している場合は、Ingester サービスを設定する必要はありません。

表1.55 Ingester に渡される Jaeger パラメーター

パラメーター	説明	値
spec: ingester: options: {}	Ingester サービスを定義する設定オプション。	

パラメーター	説明	値
options: deadlockInterval:	Ingester が終了するまでメッセージを待機する間隔 (秒単位または分単位) を指定します。システムの初期化中にメッセージが到達されない場合に Ingester が終了しないように、デッドロックの間隔はデフォルトで無効に (**0**に設定) されます。	分と秒 (例: **1m0s**)デフォルト値は **0** です。
options: kafka: consumer: topic:	**topic** パラメーターは、コレクターによってメッセージを生成するために使用され、Ingester によってメッセージを消費するために使用される Kafka 設定を特定します。	コンシューマーのラベル例: **jaeger-spans**
options: kafka: consumer: brokers:	メッセージを消費するために Ingester によって使用される Kafka 設定を特定します。	ブローカーのラベル (例: **my-cluster-kafka-brokers.kafka:9092**)
options: log-level:	Ingester のロギングレベル。	使用できる値は、**debug**、**info**、**warn**、**error**、**fatal**、**dpanic**、**panic** です。

ストリーミング Collector および Ingester の例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-streaming
spec:
  strategy: streaming
  collector:
    options:
      kafka:
        producer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
  ingester:
    options:
      kafka:
        consumer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
      ingester:
        deadlockInterval: 5
  storage:
```

```
type: elasticsearch
options:
  es:
    server-urls: http://elasticsearch:9200
```

1.27. SERVICE MESH のアンインストール

Red Hat OpenShift Service Mesh を既存の OpenShift Container Platform インスタンスからアンインストールし、そのリソースを削除するには、コントロールプレーンと Operator を削除してから、コマンドを実行してリソースを手動で削除する必要があります。

1.27.1. Red Hat OpenShift Service Mesh コントロールプレーンの削除

Service Mesh を既存の OpenShift Container Platform インスタンスからアンインストールするには、最初に Service Mesh コントロールプレーンおよび Operator を削除します。次に、コマンドを実行して残りのリソースを削除します。

1.27.1.1. Web コンソールを使用した Service Mesh コントロールプレーンの削除

Web コンソールを使用して Red Hat OpenShift Service Mesh コントロールプレーンを削除します。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
3. **Operators → Installed Operators** に移動します。
4. **Provided APIs** の **Service Mesh Control Plane** をクリックします。
5. **ServiceMeshControlPlane** メニュー ⋮ をクリックします。
6. **Delete Service Mesh Control Plane** をクリックします。
7. 確認ダイアログウィンドウで **Delete** をクリックし、**ServiceMeshControlPlane** を削除します。

1.27.1.2. CLI を使用した Service Mesh コントロールプレーンの削除

CLI を使用して Red Hat OpenShift Service Mesh コントロールプレーンを削除します。この例では、**istio-system** は、コントロールプレーンプロジェクトです。

手順

1. OpenShift Container Platform CLI にログインします。
2. 次のコマンドを実行して、**ServiceMeshMemberRoll** リソースを削除します。

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

3. 以下のコマンドを実行して、インストールした **ServiceMeshControlPlane** の名前を取得します。

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

4. **<name_of_custom_resource>** を先のコマンドの出力に置き換え、以下のコマンドを実行してカスタムリソースを削除します。

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

1.27.2. インストールされた Operator の削除

Red Hat OpenShift Service Mesh を正常に削除するには、Operator を削除する必要があります。Red Hat OpenShift Service Mesh Operator を削除したら、Kiali Operator、Red Hat OpenShift 分散トレースプラットフォーム Operator、および OpenShift Elasticsearch Operator を削除する必要があります。

1.27.2.1. Operator の削除

以下の手順に従って、Red Hat OpenShift Service Mesh を設定する Operator を削除します。以下の Operator ごとに手順を繰り返します。

- Red Hat OpenShift Service Mesh
- Kiali
- Red Hat OpenShift 分散トレースプラットフォーム
- OpenShift Elasticsearch

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operator → Installed Operators** ページから、スクロールするか、またはキーワードを **Filter by name** に入力して各 Operator を見つけます。次に、Operator 名をクリックします。
3. **Operator Details** ページで、**Actions** メニューから **Uninstall Operator** を選択します。プロンプトに従って各 Operator をアンインストールします。

1.27.3. Operator リソースのクリーンアップ

OpenShift Container Platform Web コンソールを使用して、Red Hat OpenShift Service Mesh Operator を削除した後に残ったリソースを手動で削除できます。

前提条件

- クラスター管理アクセスを持つアカウント。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
- **oc** として知られる OpenShift Container Platform コマンドラインインターフェイス (CLI) へのアクセス。

手順

1. クラスター管理者として OpenShift Container Platform CLI にログインします。

2. 以下のコマンドを実行して、Operator のアンインストール後にリソースをクリーンアップします。サービスメッシュなしで分散トレースプラットフォームをスタンドアロンのサービスとして引き続き使用する場合は、Jaeger リソースを削除しないでください。

注記

Openshift Elasticsearch Operator はデフォルトで **openshift-operators-redhat** にインストールされます。他の Operator はデフォルトで **openshift-operators** namespace にインストールされます。Operator を別の namespace にインストールしている場合、**openshift-operators** を Red Hat OpenShift Service Mesh Operator がインストールされていたプロジェクトの名前に置き換えます。

```
$ 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

$ oc delete sa -n openshift-operators istio-cni
```

第2章 SERVICE MESH 1.X

2.1. SERVICE MESH リリースノート

警告

こちらは、サポートされなくなった **Red Hat OpenShift Service Mesh** リリースのドキュメントです。

Service Mesh バージョン 1.0 および 1.1 コントロールプレーンはサポートされなくなりました。Service Mesh コントロールプレーンのアップグレードについては、Service Mesh の アップグレード を参照してください。

特定の Red Hat Service Mesh リリースのサポートステータスについては、製品ライフサイクルページ を参照してください。

2.1.1. 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

2.1.2. Red Hat OpenShift Service Mesh の概要

Red Hat OpenShift Service Mesh は、アプリケーションで一元化された制御ポイントを作成して、マイクロサービスアーキテクチャーのさまざまな問題に対応します。OpenShift Service Mesh はアプリケーションコードを変更せずに、既存の分散アプリケーションに透過的な層を追加します。

マイクロサービスアーキテクチャーは、エンタープライズアプリケーションの作業をモジュールサービスに分割するので、スケーリングとメンテナンスが容易になります。ただし、マイクロサービスアーキテクチャー上に構築されるエンタープライズアプリケーションはサイズも複雑性も増すため、マイクロサービスアーキテクチャーの理解と管理は困難です。Service Mesh は、サービス間のトラフィックをキャプチャーしたり、インターセプトしたりして、他のサービスへの新規要求を変更、リダイレクト、または作成することによってこれらのアーキテクチャーの問題に対応できます。

オープンソースの Istio project をベースにする Service Mesh では、検出、負荷分散、サービス間の認証、障害復旧、メトリクス、およびモニタリングを提供する、デプロイされたサービスのネットワークを簡単に作成できます。サービスメッシュは、A/B テスト、カナリアリリース、レート制限、アクセス制御、エンドツーエンド認証を含む、より複雑な運用機能を提供します。

2.1.3. サポート

本書で説明されている手順、または OpenShift Container Platform で問題が発生した場合は、Red Hat カスタマーポータル にアクセスしてください。カスタマーポータルでは、次のことができます。

- Red Hat 製品に関するアーティクルおよびソリューションについての Red Hat ナレッジベースの検索またはブラウズ。

- Red Hat サポートに対するサポートケースの送信。
- その他の製品ドキュメントへのアクセス。

クラスターの問題を特定するには、OpenShift Cluster Manager で Insights を使用できます。Insights により、問題の詳細と、利用可能な場合は問題の解決方法に関する情報が提供されます。

本書の改善への提案がある場合、またはエラーを見つけた場合は、最も関連性の高いドキュメントコンポーネントの Jira Issue を送信してください。セクション名や OpenShift Container Platform バージョンなどの具体的な情報を提供してください。

サポートケースを作成する際、ご使用のクラスターについてのデバッグ情報を Red Hat サポートに提供していただくと Red Hat のサポートに役立ちます。

must-gather ツールを使用すると、仮想マシンおよび Red Hat OpenShift Service Mesh に関する他のデータを含む、OpenShift Container Platform クラスターについての診断情報を収集できます。

迅速なサポートを得るには、OpenShift Container Platform と Red Hat OpenShift Service Mesh の両方の診断情報を提供してください。

2.1.3.1. must-gather ツールについて

oc adm must-gather CLI コマンドは、以下のような問題のデバッグに必要となる可能性のあるクラスターからの情報を収集します。

- リソース定義
- サービスログ

デフォルトで、**oc adm must-gather** コマンドはデフォルトのプラグインイメージを使用し、**./must-gather.local** に書き込みを行います。

または、以下のセクションで説明されているように、適切な引数を指定してコマンドを実行すると、特定の情報を収集できます。

- 1 つ以上の特定の機能に関連するデータを収集するには、以下のセクションに示すように、イメージと共に **--image** 引数を使用します。
  以下に例を示します。

  ```
  $ oc adm must-gather  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.9.0
  ```

- 監査ログを収集するには、以下のセクションで説明されているように **-- /usr/bin/gather_audit_logs** 引数を使用します。
  以下に例を示します。

  ```
  $ oc adm must-gather -- /usr/bin/gather_audit_logs
  ```

  注記

  ファイルのサイズを小さくするために、監査ログはデフォルトの情報セットの一部として収集されません。

oc adm must-gather を実行すると、ランダムな名前を持つ新規 Pod がクラスターの新規プロジェクトに作成されます。データは Pod で収集され、**must-gather.local** で始まる新規ディレクトリーに保存されます。このディレクトリーは、現行の作業ディレクトリーに作成されます。

以下に例を示します。

```
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. 前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift Container Platform CLI (**oc**) がインストールされていること。

2.1.3.3. サービスメッシュデータの収集について

oc adm must-gather CLI コマンドを使用してクラスターについての情報を収集できます。これには、Red Hat OpenShift Service Mesh に関連する機能およびオブジェクトが含まれます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift Container Platform CLI (**oc**) がインストールされていること。

手順

1. **must-gather** で Red Hat OpenShift Service Mesh データを収集するには、Red Hat OpenShift Service Mesh イメージを指定する必要があります。

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

2. **must-gather** で特定の Service Mesh コントロールプレーン namespace の Red Hat OpenShift Service Mesh データを収集するには、Red Hat OpenShift Service Mesh イメージおよび namespace を指定する必要があります。この例では、**<namespace>** を **istio-system** などの Service Mesh コントロールプレーンの namespace に置き換えます。

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

2.1.4. Red Hat OpenShift Service Mesh でサポートされている設定

以下は、Red Hat OpenShift Service Mesh で唯一サポートされている設定です。

- OpenShift Container Platform バージョン 4.6 以降。

注記

OpenShift Online および Red Hat OpenShift Dedicated は Red Hat OpenShift Service Mesh に対してはサポートされていません。

- デプロイメントは、フェデレーションされていない単一の OpenShift Container Platform クラスターに含まれる必要があります。
- Red Hat OpenShift Service Mesh の本リリースは、OpenShift Container Platform x86_64 でのみ利用できます。
- 本リリースでは、すべての Service Mesh コンポーネントが OpenShift Container Platform クラスターに含まれ、動作している設定のみをサポートしています。クラスター外にあるマイクロサービスの管理や、マルチクラスターシナリオにおけるマイクロサービスの管理はサポートしていません。
- 本リリースでは、仮想マシンなどの外部サービスを統合していない設定のみをサポートしています。

Red Hat OpenShift Service Mesh のライフサイクルおよびサポートされる設定についての詳細は、サポートポリシー について参照してください。

2.1.4.1. Red Hat OpenShift Service Mesh でサポートされている Kiali の設定

- Kiali の可観測性コンソールは Chrome、Edge、Firefox、または Safari ブラウザーの 2 つの最新リリースでのみサポートされています。

2.1.4.2. サポートされている Mixer アダプター

- 本リリースでは、次の Mixer アダプターのみをサポートしています。
  - 3scale Istio Adapter

2.1.5. 新機能

Red Hat OpenShift Service Mesh は、サービスのネットワーク全体で多数の主要機能を均一に提供します。

- トラフィック管理: サービス間でトラフィックおよび API 呼び出しのフローを制御し、呼び出しの安定度を高め、不利な条件下でもネットワークの堅牢性を維持します。
- サービス ID とセキュリティー: メッシュのサービスを検証可能な ID で指定でき、サービスのトラフィックがさまざまな信頼度のネットワークに送られる際にそのトラフィックを保護する機能を提供します。
- ポリシーの適用: サービス間の対話に組織のポリシーを適用し、アクセスポリシーが適用され、リソースはコンシューマー間で均等に分散されるようにします。ポリシー変更は、アプリケーションコードを変更するのではなく、メッシュを設定して行います。
- **Telemetry**: サービス間の依存関係やそれらの間のトラフィックの性質やフローを理解するのに役立ち、問題を素早く特定できます。

2.1.5.1. Red Hat OpenShift Service Mesh 1.1.18.2 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) に対応しています。

2.1.5.1.1. Red Hat OpenShift Service Mesh バージョン 1.1.18.2 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.4.10
Jaeger	1.30.2
Kiali	1.12.21.1
3scale Istio Adapter	1.0.0

2.1.5.2. Red Hat OpenShift Service Mesh 1.1.18.1 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) に対応しています。

2.1.5.2.1. Red Hat OpenShift Service Mesh バージョン 1.1.18.1 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.4.10
Jaeger	1.30.2
Kiali	1.12.20.1
3scale Istio Adapter	1.0.0

2.1.5.3. Red Hat OpenShift Service Mesh 1.1.18 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) に対応しています。

2.1.5.3.1. Red Hat OpenShift Service Mesh バージョン 1.1.18 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.4.10
Jaeger	1.24.0
Kiali	1.12.18

コンポーネント	バージョン
3scale Istio Adapter	1.0.0

2.1.5.4. Red Hat OpenShift Service Mesh 1.1.17.1 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) に対応しています。

2.1.5.4.1. Red Hat OpenShift Service Mesh が URI フラグメントを処理する方法の変更

Red Hat OpenShift Service Mesh には、リモートで悪用可能な脆弱性 CVE-2021-39156 が含まれており、URI パスにフラグメント (URI の末尾が # 文字で始まるセクション) を含む HTTP リクエストが Istio URI パスベースの認証ポリシーを無視する可能性があります。たとえば、Istio 認証ポリシーは URI パス **/user/profile** に送信される要求を拒否します。脆弱なバージョンでは、URI パス **/user/profile#section1** のリクエストは、deny ポリシーと、(正規化された URI **path /user/profile%23section1** を使用する) バックエンドへのルートを無視するため、セキュリティーのインシデントにつながる可能性があります。

DENY アクションおよび **operation.paths**、または ALLOW アクションおよび **operation.notPaths** で認可ポリシーを使用する場合は、この脆弱性の影響を受けます。

軽減策により、リクエストの URI の断片部分が、承認とルーティングの前に削除されます。これにより、URI のフラグメントを持つ要求が、フラグメントの一部のない URI をベースとする承認ポリシーが無視できなくなります。

2.1.5.4.2. 認証ポリシーに必要な更新

Istio はホスト名自体とすべての一致するポートの両方にホスト名を生成します。たとえば、httpbin.foo のホストの仮想サービスまたはゲートウェイは、httpbin.foo and httpbin.foo:*に一致する設定を生成します。ただし、完全一致許可ポリシーは、**hosts** または **notHosts** フィールドに指定された完全一致文字列にのみ一致します。

ルールの正確な文字列比較を使用して hosts または notHosts を決定する **AuthorizationPolicy** リソースがある場合、クラスターは影響を受けます。

完全一致ではなく接頭辞一致を使用するように、認証ポリシー ルール を更新する必要があります。たとえば、最初の **AuthorizationPolicy** の例で **hosts: ["httpbin.com"]** を **hosts: ["httpbin.com:*"]** に置き換えます。

接頭辞一致を使用した最初の AuthorizationPolicy の例

```
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:*"]
```

接頭辞一致を使用した 2 つ目の AuthorizationPolicy の例

```
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. Red Hat OpenShift Service Mesh 1.1.17 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

2.1.5.6. Red Hat OpenShift Service Mesh 1.1.16 の新機能

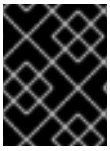
Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

2.1.5.7. Red Hat OpenShift Service Mesh 1.1.15 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

2.1.5.8. Red Hat OpenShift Service Mesh 1.1.14 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。



重要

CVE-2021-29492 および CVE-2021-31920 に対応するために、手動による手順を完了する必要があります。

2.1.5.8.1. CVE-2021-29492 および CVE-2021-31920 で必要な手動による更新

Istio にはリモートで悪用可能な脆弱性があり、複数のスラッシュまたはエスケープされたスラッシュ文字 (**%2F`** または %5C`) を持つ HTTP リクエストパスが、パスベースの認証ルールが使用される場合に Istio 認証ポリシーを無視する可能性があります。

たとえば、Istio クラスター管理者が、パス **/admin** での要求を拒否する認証 DENY ポリシーを定義すると仮定します。URL パス **//admin** に送信される要求は、認証ポリシーによって拒否されません。

RFC 3986 に応じて、複数のスラッシュを持つパス **//admin** は、**/admin** とは異なるパスとして処理される必要があります。ただし、一部のバックエンドサービスは、複数のスラッシュを単一のスラッシュ

にマージして URL パスを正規化することを選択します。これにより、認証ポリシーがバイパスされ (//**admin** は /**admin** に一致しません)、ユーザーはバックエンドのパス (/**admin**) でリソースにアクセスできます。これにより、セキュリティーのインシデントを示されます。

ALLOW action + notPaths フィールドまたは **DENY action + paths field** パターンを使用する認証ポリシーがある場合、クラスターはこの脆弱性の影響を受けます。これらのパターンは、予期しないポリシーのバイパスに対して脆弱です。

以下の場合、クラスターはこの脆弱性の影響を受けません。

- 認証ポリシーがありません。
- 認証ポリシーは、**paths** または **notPaths** フィールドを定義しません。
- 認証ポリシーは、**ALLOW action + paths** フィールドまたは **DENY action + notPaths** フィールドのパターンを使用します。これらのパターンは、ポリシーのバイパスではなく、予期しない拒否を生じさせる可能性があります。このような場合、アップグレードは任意です。

注記

パスの正規化向けの Red Hat OpenShift Service Mesh 設定の場所は、Istio 設定とは異なります。

2.1.5.8.2. パスの正規化設定の更新

Istio 認証ポリシーは、HTTP リクエストの URL パスをベースとする場合があります。URI の正規化として知られる パスの正規化 は、正規化されたパスを標準の方法で処理できるように、受信要求のパスを変更し、標準化します。構文の異なるパスは、パスの正規化後と同一になる場合があります。

Istio は、認証ポリシーに対して評価し、要求をルーティングする前の、要求パスでの以下の正規化スキームをサポートします。

表2.1 正規化スキーム

オプション	説明	例	注記
NONE	正規化は行われません。Envoy が受信する内容はそのまますべて、どのバックエンドサービスにも完全に転送されます。	**../%2fa../b** は認証ポリシーによって評価され、サービスに送信されます。	この設定は CVE-2021-31920 に対して脆弱です。
BASE	現時点で、これは Istio の **デフォルト** インストールで使用されるオプションです。これにより、Envoy プロキシーで **normalize_path** オプションが適用されます。これは、追加の正規化において RFC 3986 に従い、バックスラッシュをフォワードスラッシュに変換します。	**/a/../b** は **/b** に正規化されます。**\da** は **/da** に正規化されます。	この設定は CVE-2021-31920 に対して脆弱です。

オプション	説明	例	注記
MERGE_SLASHES	スラッシュは BASE の正規化後にマージされます。	**/a//b** は **/a/b** に正規化されます。	この設定に対して更新を行い、CVE-2021-31920 のリスクを軽減します。
DECODE_AND_MERGE_SLASHES	デフォルトですべてのトラフィックを許可する場合の最も厳密な設定です。この設定の場合、認証ポリシーのルートを詳細にテストする必要がある点に注意してください。パーセントでエンコードされた スラッシュおよびバックスラッシュ文字 (**%2F**、**%2f**、**%5C** および **%5c**) は、**MERGE_SLASHES** の正規化の前に / または \ にデコードされます。	**/a%2fb** は **/a/b** に正規化されます。	この設定に対して更新を行い、CVE-2021-31920 のリスクを軽減します。この設定はより安全ですが、アプリケーションが破損する可能性があります。実稼働環境にデプロイする前にアプリケーションをテストします。

正規化アルゴリズムは以下の順序で実行されます。

1. パーセントでデコードされた **%2F**、**%2f**、**%5C** および **%5c**。
2. Envoy の **normalize_path** オプションで実装された RFC 3986 およびその他の正規化。
3. スラッシュをマージします。

警告

これらの正規化オプションは HTTP 標準および一般的な業界プラクティスの推奨事項を表しますが、アプリケーションは独自に選択したいずれかの方法で URL を解釈する場合があります。拒否ポリシーを使用する場合には、アプリケーションの動作を把握している必要があります。

2.1.5.8.3. パスの正規化設定の例

Envoy がバックエンドサービスの期待値に一致するように要求パスを正規化することは、システムのセキュリティーを保護する上で非常に重要です。以下の例は、システムを設定するための参考として使用できます。正規化された URL パス、または **NONE** が選択されている場合は元の URL パスは以下のようになります。

1. 認証ポリシーの確認に使用されます。
2. バックエンドアプリケーションに転送されます。

表2.2 設定例

アプリケーションの条件	選択内容
プロキシーを使用して正規化を行う。	**BASE**、**MERGE_SLASHES**、または **DECODE_AND_MERGE_SLASHES**
RFC 3986 に基づいて要求パスを正規化し、スラッシュをマージしない。	**BASE**
RFC 3986 に基づいて要求パスを正規化し、スラッシュをマージするが、パーセントでエンコードされた スラッシュをデコードしない。	**MERGE_SLASHES**
RFC 3986 に基づいて要求パスを正規化し、パーセントでエンコードされた スラッシュをデコードし、スラッシュをマージする。	**DECODE_AND_MERGE_SLASHES**
RFC 3986 と互換性のない方法で要求パスを処理する。	**NONE**

2.1.5.8.4. パスを正規化するための SMCP の設定

Red Hat OpenShift Service Mesh のパスの正規化を設定するには、**ServiceMeshControlPlane** で以下を指定します。設定例を使用すると、システムの設定を判断するのに役立ちます。

SMCP v1 pathNormalization

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

2.1.5.9. Red Hat OpenShift Service Mesh 1.1.13 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

2.1.5.10. Red Hat OpenShift Service Mesh 1.1.12 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

2.1.5.11. Red Hat OpenShift Service Mesh 1.1.11 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

2.1.5.12. Red Hat OpenShift Service Mesh 1.1.10 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

2.1.5.13. Red Hat OpenShift Service Mesh 1.1.9 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

2.1.5.14. Red Hat OpenShift Service Mesh 1.1.8 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

2.1.5.15. Red Hat OpenShift Service Mesh 1.1.7 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

2.1.5.16. Red Hat OpenShift Service Mesh 1.1.6 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

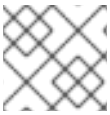
2.1.5.17. Red Hat OpenShift Service Mesh 1.1.5 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

本リリースでは、暗号化スイートの設定に対するサポートも追加しています。

2.1.5.18. Red Hat OpenShift Service Mesh 1.1.4 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。



注記

CVE-2020-8663 に対応するために、手動による手順を完了する必要があります。

2.1.5.18.1. CVE-2020-8663 で必要な手動による更新

CVE-2020-8663 の修正:**envoy: Resource exhaustion when accepting too many connections** により、ダウンストリーム接続に設定可能な制限が追加されました。この制限の設定オプションは、この脆弱性を軽減するように設定する必要があります。

重要

1.1 バージョンまたは 1.0 バージョンの Red Hat OpenShift Service Mesh を使用しているかどうかに関係なく、この CVE に対応するには、これらの手動の手順が必要です。

この新しい設定オプションは **overload.global_downstream_max_connections** と呼ばれ、プロキシーの **runtime** 設定として設定できます。Ingress ゲートウェイで制限を設定するには、以下の手順を実行します。

手順

1. 以下のテキストで **bootstrap-override.json** という名前のファイルを作成し、プロキシーがブートストラップテンプレートを上書きし、ディスクからランタイム設定を読み込むように強制します。

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

2. **bootstrap-override.json** ファイルからシークレットを作成し、<SMCPnamespace> を、サービスメッシュコントロールプレーン (SMCP) を作成した namespace に置き換えます。

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

3. SMCP 設定を更新して上書きを有効にします。

更新された SMCP 設定例 #1

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

4. 新規設定オプションを設定するには、**overload.global_downstream_max_connections** 設定の必要な値を持つシークレットを作成します。以下の例では、**10000** の値を使用します。

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

5. SMCP を再度更新して、Envoy がランタイム設定を検索する場所にシークレットをマウントします。

更新された SMCP 設定例 #2

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

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

2.1.5.18.2. Elasticsearch 5 から Elasticsearch 6 へのアップグレード

Elasticsearch 5 から Elasticsearch 6 に更新する場合、証明書に関する問題があるために Jaeger インスタンスを削除してから Jaeger インスタンスを再作成する必要があります。Jaeger インスタンスを再作成すると、証明書の新たなセットの作成がトリガーされます。永続ストレージを使用している場合、新規の Jaeger インスタンスの Jaeger 名および namespace が削除された Jaeger インスタンスと同じである限り、新規の Jaeger インスタンスについて同じボリュームをマウントできます。

Jaeger が Red Hat Service Mesh の一部としてインストールされている場合の手順

1. Jaeger カスタムリソースファイルの名前を判別します。

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

以下のような出力が表示されます。

```
NAME     AGE
jaeger   3d21h
```

2. 生成されたカスタムリソースファイルを一時ディレクトリーにコピーします。

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

3. Jaeger インスタンスを削除します。

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

4. カスタムリソースファイルのコピーから Jaeger インスタンスを再作成します。

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

5. 生成されたカスタムリソースファイルのコピーを削除します。

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

Jaeger が Red Hat Service Mesh の一部としてインストールされていない場合の手順

開始する前に、Jaeger カスタムリソースファイルのコピーを作成します。

1. カスタムリソースファイルを削除して Jaeger インスタンスを削除します。

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

以下に例を示します。

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

2. カスタムリソースファイルのバックアップコピーから Jaeger インスタンスを再作成します。

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

3. Pod が再起動したことを確認します。

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

2.1.5.19. Red Hat OpenShift Service Mesh 1.1.3 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、CVE (Common Vulnerabilities and Exposures) およびバグ修正に対応しています。

2.1.5.20. Red Hat OpenShift Service Mesh 1.1.2 の新機能

Red Hat OpenShift Service Mesh の本リリースは、セキュリティー脆弱性に対応しています。

2.1.5.21. Red Hat OpenShift Service Mesh 1.1.1 の新機能

Red Hat OpenShift Service Mesh のリリースでは、非接続インストールのサポートが追加されました。

2.1.5.22. Red Hat OpenShift Service Mesh 1.1.0 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、1.4.6 および Jaeger 1.17.1 のサポートが追加されました。

2.1.5.22.1. 1.0 から 1.1 への手動更新

Red Hat OpenShift Service Mesh 1.0 から 1.1 に更新する場合、**ServiceMeshControlPlane** リソースを更新してコントロールプレーンのコンポーネントを新規バージョンに更新する必要があります。

1. Web コンソールで、Red Hat OpenShift Service Mesh Operator をクリックします。
2. **Project** メニューをクリックし、一覧から **ServiceMeshControlPlane** がデプロイされているプロジェクト (例: **istio-system**) を選択します。
3. コントロールプレーンの名前 (**basic-install** など) をクリックします。
4. YAML をクリックし、バージョンフィールドを **ServiceMeshControlPlane** リソースの **spec:** に追加します。たとえば、Red Hat OpenShift Service Mesh 1.1.0 に更新するには、**version: v1.1** を追加します。

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

バージョンフィールドでは、インストールする ServiceMesh のバージョンを指定し、デフォルトは利用可能な最新バージョンに設定されます。

注記

Red Hat OpenShift Service Mesh v1.0 のサポートは 2020 年 10 月に終了していることに注意してください。v1.1 または v2.0 のいずれかにアップグレードする必要があります。

2.1.6. 非推奨の機能

以前のリリースで利用可能であった一部の機能が非推奨になるか、または削除されました。

非推奨の機能は依然として OpenShift Container Platform に含まれており、引き続きサポートされますが、本製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

2.1.6.1. 非推奨になった Red Hat OpenShift Service Mesh 1.1.5 の機能

以下のカスタムリソースはリリース 1.1.5 で非推奨となり、リリース 1.1.12 で削除されました。

- **Policy**: **Policy** リソースは非推奨となり、今後のリリースで **PeerAuthentication** リソースに置き換えられます。
- **MeshPolicy**: **MeshPolicy** リソースは非推奨となり、今後のリリースで **PeerAuthentication** リソースに置き換えられます。
- **v1alpha1** RBAC API: v1alpha1 RBAC ポリシーは v1beta1 **AuthorizationPolicy** で非推奨にされています。RBAC (ロールベースアクセス制御) は **ServiceRole** および **ServiceRoleBinding** オブジェクトを定義します。
  - **ServiceRole**
  - **ServiceRoleBinding**
- **RbacConfig**: **RbacConfig** は、Istio RBAC 動作を制御するためにカスタムリソース定義を実装します。
  - **ClusterRbacConfig** (Red Hat OpenShift Service Mesh 1.0 より前のバージョン)
  - **ServiceMeshRbacConfig** (Red Hat OpenShift Service Mesh バージョン 1.0 以降)
- Kiali では、**login** および **LDAP** ストラテジーは非推奨になりました。今後のバージョンでは、OpenID プロバイダーを使用した認証が導入されます。

以下のコンポーネントは本リリースで非推奨となり、今後のリリースで **Istiod** コンポーネントに置き換えられます。

- **Mixer**: アクセス制御および使用ポリシー
- **Pilot**: サービス検出およびプロキシー設定
- **Citadel**: 証明書の生成
- **Galley**: 設定の検証および分散

2.1.7. 既知の問題

Red Hat OpenShift Service Mesh には以下のような制限が存在します。

- アップストリームの Istio プロジェクトでサポートされておらず、また OpenShift Container Platform でも完全にサポートされていないため、Red Hat OpenShift Service Mesh は IPv6 をサポートしていません。

- グラフレイアウト: Kiali グラフのレイアウトは、アプリケーションのアーキテクチャーや表示データ (グラフィックノードとその対話の数) によって異なることがあります。すべての状況に適した単一のレイアウトを作成することは不可能ではないにしても困難であるため、Kiali は複数の異なるレイアウトの選択肢を提供します。別のレイアウトを選択するには、**Graph Settings** メニューから異なる **Layout Schema** を選択します。

- Kiali コンソールから Jaeger や Grafana などの関連サービスに初めてアクセスする場合、OpenShift Container Platform のログイン認証情報を使用して証明書を受け入れ、再認証する必要があります。これは、フレームワークが組み込まれたページをコンソールで表示する方法に問題があるために生じます。

2.1.7.1. Service Mesh の既知の問題

Red Hat OpenShift Service Mesh には次のような既知の問題が存在します。

- Jaeger/Kiali Operator のアップグレードが Operator の保留によりブロックされる Jaeger または Kiali Operator を Service Mesh 1.0.x がインストールされた状態でアップグレードすると、Operator のステータスは Pending と表示されます。
  回避策: 詳細は、関連するナレッジベースの記事を参照してください。

- Istio-14743 Red Hat OpenShift Service Mesh のこのリリースがベースとしている Istio のバージョンに制限があるため、現時点で Service Mesh と互換性のないアプリケーションが複数あります。詳細は、リンク先のコミュニティーの問題を参照してください。

- MAISTRA-858 Istio 1.1.x に関連する非推奨のオプションと設定 について説明する以下のような Envoy ログメッセージが予想されます。

  - [2019-06-03 07:03:28.943][19][warning][misc] [external/envoy/source/common/protobuf/utility.cc:129] Using deprecated option 'envoy.api.v2.listener.Filter.config'.この設定はまもなく Envoy から削除されます。

  - [2019-08-12 22:12:59.001][13][warning][misc] [external/envoy/source/common/protobuf/utility.cc:174] Using deprecated option 'envoy.api.v2.Listener.use_original_dst' from file lds.proto.この設定はまもなく Envoy から削除されます。

- MAISTRA-806 エビクトされた Istio Operator Pod により、メッシュおよび CNI はデプロイできなくなります。
  回避策: コントロールペインのデプロイ時に **istio-operator** Pod がエビクトされる場合は、エビクトされた **istio-operator** Pod を削除します。

- MAISTRA-681 コントロールプレーンに多くの namespace がある場合に、パフォーマンスの問題が発生する可能性があります。

- MAISTRA-465 Maistra Operator が、Operator メトリクスのサービスの作成に失敗します。

- MAISTRA-453 新規プロジェクトを作成して Pod を即時にデプロイすると、サイドカーコンテナーの挿入は発生しません。この Operator は Pod の作成前に **maistra.io/member-of** を追加できないため、サイドカーコンテナーの挿入を発生させるには Pod を削除し、再作成する必要があります。

- MAISTRA-158 同じホスト名を参照する複数のゲートウェイを適用すると、すべてのゲートウェイが機能しなくなります。

2.1.7.2. Kiali の既知の問題

注記

Kiali についての新たな問題は、**Component** が **Kiali** に設定された状態の OpenShift Service Mesh プロジェクトに作成される必要があります。

Kiali の既知の問題は以下のとおりです。

- KIALI-2206 初回の Kiali コンソールへのアクセス時に、Kiali のキャッシュされたブラウザーデータがない場合、Kiali サービスの詳細ページの Metrics タブにある View in Grafana リンクは誤った場所にリダイレクトされます。この問題は、Kiali への初回アクセス時にのみ生じます。
- KIALI-507 Kiali は Internet Explorer 11 に対応していません。これは、基礎となるフレームワークが Internet Explorer に対応していないためです。Kiali コンソールにアクセスするには、Chrome、Edge、Firefox、または Safari ブラウザーの最新の 2 バージョンのいずれかを使用します。

2.1.7.3. Red Hat OpenShift 分散トレースの既知の問題

Red Hat OpenShift 分散トレースには、以下の制限があります。

- Apache Spark はサポートされていません。
- AMQ/Kafka 経由のストリーミングデプロイメントは、IBM Z および IBM Power Systems ではサポートされません。

これらは Red Hat OpenShift 分散トレースの既知の問題です。

- TRACING-2057 Kafka API が **v1beta2** に更新され、Strimzi Kafka Operator 0.23.0 がサポートされるようになりました。ただし、この API バージョンは AMQ Streams 1.6.3 ではサポートされません。以下の環境がある場合、Jaeger サービスはアップグレードされず、新規の Jaeger サービスを作成したり、既存の Jaeger サービスを変更したりすることはできません。
  - Jaeger Operator チャネル: **1.17.x stable** または **1.20.x stable**
  - AMQ Streams Operator チャネル: **amq-streams-1.6.x**
    この問題を解決するには、AMQ Streams Operator のサブスクリプションチャネルを **amq-streams-1.7.x** または **stable** のいずれかに切り替えます。

2.1.8. 修正された問題

次の問題は、現在のリリースで解決されています。

2.1.8.1. Service Mesh の修正された問題

- MAISTRA-2371 は listerInformer で tombstones を処理します。更新されたキャッシュコードベースは、namespace キャッシュからのイベントを集約されたキャッシュに変換する際に tombstones を処理しないため、go ルーチンでパニックが生じました。
- OSSM-542 Galley はローテーション後、新しい証明書を使用していません。
- OSSM-99 ラベルを持たない直接の Pod から生成されたワークロードは Kiali をクラッシュさせる可能性があります。

- OSSM-93 IstioConfigList は複数の名前でフィルターをかけることができません。
- OSSM-92 VS/DR YAML 編集ページで未保存の変更をキャンセルしても、変更はキャンセルされません。
- OSSM-90 サービスの詳細ページでは、トレースは利用できません。
- MAISTRA-1649 ヘッドレス サービスは、異なる namespace にある場合に競合します。異なる namespace にヘッドレスサービスをデプロイする場合、エンドポイント設定はマージされ、無効な Envoy 設定がサイドカーコンテナーにプッシュされます。
- MAISTRA-1541 コントローラーが所有者の参照に設定されていない場合に kubernetesenv でパニックが生じます。Pod にコントローラーを指定しない ownerReference がある場合は、**kubernetesenv cache.go** コード内でパニックが生じます。
- 本リリースおよび今後のリリースでは、コントロールプレーンのインストールから MAISTRA-1352 Cert-manager カスタムリソース定義 (CRD) が削除されました。Red Hat OpenShift Service Mesh がすでにインストールされている場合、cert-manager が使用されていない場合には CRD を手動で削除する必要があります。
- MAISTRA-1001 HTTP/2 接続を閉じると、**istio-proxy** でセグメント化の障害が生じる可能性があります。
- MAISTRA-932 Jaeger Operator と OpenShift Elasticsearch Operator 間の依存関係を追加するために **requires** メタデータが追加されました。Jaeger Operator のインストール時に、これが利用不可能な場合は OpenShift Elasticsearch Operator を自動的にデプロイします。
- MAISTRA-862 Galley は namespace が数多く削除および再作成されると、 watch (監視) をドロップし、他のコンポーネントへの設定の提供を停止しました。
- MAISTRA-833 Pilot は namespace が数多く削除および再作成されると設定の配信を停止しました。
- MAISTRA-684 **istio-operator** のデフォルトの Jaeger バージョンは 1.12.0 で、Red Hat OpenShift Service Mesh 0.12.TechPreview で提供される Jaeger バージョン 1.13.1 と一致しません。
- MAISTRA-622 Maistra 0.12.0/TP12 では、パーミッシブモードは機能しません。ユーザーには Plain text モードまたは Mutual TLS モードを使用するオプションがありますが、パーミッシブモードのオプションはありません。
- MAISTRA-572 Jaeger を Kiali と併用できません。本リリースでは、Jaeger は OAuth プロキシーを使用するように設定されていますが、ブラウザーでのみ機能するようにも設定され、サービスアクセスを許可しません。Kiali は Jaeger エンドポイントと適切に通信できないため、Jaeger が無効であると見なします。TRACING-591 も参照してください。
- MAISTRA-357 AWS の OpenShift 4 Beta では、デフォルトでポート 80 以外のポートの Ingress ゲートウェイを介して TCP または HTTPS サービスにアクセスすることはできません。AWS ロードバランサーには、サービスエンドポイントのポート 80 がアクティブであるかどうかを検証するヘルスチェックがあります。サービスがポート 80 で実行されていないと、ロードバランサーのヘルスチェックは失敗します。
- MAISTRA-348 AWS の OpenShift 4 Beta は、80 または 443 以外のポートでの Ingress ゲートウェイトラフィックをサポートしません。Ingress ゲートウェイが 80 または 443 以外のポート番号の TCP トラフィックを処理するように設定する場合、回避策として OpenShift ルーターではなく AWS ロードバランサーによって提供されるサービスホスト名を使用する必要があります。

- MAISTRA-193 ヘルスチェックが citadel で有効になっていると、予期しないコンソール情報メッセージが表示されます。

- Bug 1821432: OpenShift Container Platform Control Resource の詳細ページのトグルコントロールで CR が正しく更新されない。OpenShift Container Platform Web コンソールの Service Mesh Control Plane (SMCP) Overview ページの UI のトグルコントロールにより、リソースの誤ったフィールドが更新されることがあります。ServiceMeshControlPlane を更新するには、YAML コンテンツを直接編集するか、トグルコントロールをクリックせずにコマンドラインからリソースを更新します。

2.1.8.2. Kiali の修正された問題

- KIALI-3239 Kiali Operator Pod が Evicted のステータスで失敗すると、Kiali Operator のデプロイがブロックされます。回避策として、エビクトされた Pod を削除して、Kiali Operator を再デプロイします。

- KIALI-3118 ServiceMeshMemberRoll の変更 (プロジェクトの追加または削除などの) 後、Kiali Pod が再起動し、その間にグラフページにエラーが表示されます。

- KIALI-3096 Service Mesh でラインタイムメトリクスが失敗します。Service Mesh と Prometheus 間には OAuth フィルターがあり、アクセスを付与するにはベアラートークンを Prometheus に渡す必要があります。Kiali は Prometheus サーバーと通信する際にこのトークンを使用するように更新されていますが、アプリケーションメトリクスは現在 403 エラーで失敗しています。

- KIALI-3070 このバグは、デフォルトのダッシュボードではなく、カスタムダッシュボードにのみ影響します。メトリクス設定でラベルを選択し、ページを更新すると、メニュー上でそれらの選択は保持されますが、その選択はチャート上に表示されません。

- KIALI-2686 コントロールプレーンに多くの namespace がある場合に、パフォーマンスの問題が発生する可能性があります。

2.1.8.3. Red Hat OpenShift 分散トレースの修正された問題

- TRACING-2337 Jaeger が、Jaeger ログに以下のような警告メッセージを繰り返しロギングする。

  ```
  {"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}
  ```

  この問題は、gRPC ポートではなく、クエリーサービスの HTTP(S) ポートのみを公開することで解決されました。

- TRACING-2009 Jaeger Operator が Strimzi Kafka Operator 0.23.0 のサポートを追加するように更新されました。

- TRACING-1907: アプリケーション namespace に Config Map がないため、Jaeger エージェントサイドカーの挿入が失敗していました。Config Map は正しくない **OwnerReference** フィールドの設定により自動的に削除され、そのため、アプリケーション Pod は ContainerCreating ステージから移動しませんでした。誤った設定が削除されました。

- TRACING-1725 は TRACING-1631 に対応しています。これはもう 1 つの修正であり、同じ名前だが異なる namespace にある複数の Jaeger 実稼働インスタンスがある場合に Elasticsearch 証明書を適切に調整することができるようになりました。BZ-1918920 も参照してください。

- TRACING-1631 同じ名前を使用するが、異なる namespace 内にある複数の Jaeger 実稼働インスタンスを使用すると、Elasticsearch 証明書に問題が発生します。複数のサービスメッシュがインストールされている場合、すべての Jaeger Elasticsearch インスタンスは個別のシークレットではなく同じ Elasticsearch シークレットを持ち、これにより、OpenShift Elasticsearch Operator がすべての Elasticsearch クラスターと通信できなくなりました。

- TRACING-1300 Istio サイドカーを使用する場合に、Agent と Collector 間の接続が失敗します。Jaeger Operator で有効にされた TLS 通信の更新は、Jaeger サイドカーエージェントと Jaeger Collector 間でデフォルトで提供されます。

- TRACING-1208 Jaeger UI にアクセスする際に、認証の 500 Internal Error が出されます。OAuth を使用して UI に対する認証を試行すると、oauth-proxy サイドカーが **additionalTrustBundle** でインストール時に定義されたカスタム CA バンドルを信頼しないため、500 エラーが出されます。

- TRACING-1166 現時点で、Jaeger ストリーミングストラテジーを非接続環境で使用することはできません。Kafka クラスターがプロビジョニングされる際に、以下のエラーが出されます:
Failed to pull image registry.redhat.io/amq7/amq-streams-kafka-24-rhel7@sha256:f9ceca004f1b7dccb3b82d9a8027961f9fe4104e0ed69752c0bdd8078b4a1076

- Trace-809 Jaeger Ingester には Kafka 2.3 との互換性がありません。Jaeger Ingester のインスタンスが複数あり、十分なトラフィックがある場合、リバランスメッセージがログに継続的に生成されます。これは、Kafka 2.3.1 で修正された Kafka 2.3 のリグレッションによって生じます。詳細は、Jaegertracing-1819 を参照してください。

- BZ-1918920/LOG-1619 Elasticsearch Pod は更新後に自動的に再起動しません。
回避策: Pod を手動で再起動します。

2.2. SERVICE MESH について

警告

こちらは、サポートされなくなった **Red Hat OpenShift Service Mesh** リリースのドキュメントです。

Service Mesh バージョン 1.0 および 1.1 コントロールプレーンはサポートされなくなりました。Service Mesh コントロールプレーンのアップグレードについては、Service Mesh の アップグレード を参照してください。

特定の Red Hat Service Mesh リリースのサポートステータスについては、製品ライフサイクルページ を参照してください。

Red Hat OpenShift Service Mesh は、サービスメッシュにおいてネットワーク化されたマイクロサービス全体の動作に関する洞察と運用管理のためのプラットフォームを提供します。Red Hat OpenShift Service Mesh では、OpenShift Container Platform 環境でマイクロサービスの接続、保護、監視を行うことができます。

2.2.1. サービスメッシュについて

サービスメッシュ は、分散したマイクロサービスアーキテクチャーの複数のアプリケーションを設定するマイクロサービスのネットワークであり、マイクロサービス間の対話を可能にします。Service Mesh のサイズとおよび複雑性が増大すると、これを把握し、管理することがより困難になる可能性があります。

オープンソースの Istio プロジェクトをベースとする Red Hat OpenShift Service Mesh は、サービスコードに変更を加えずに、既存の分散したアプリケーションに透過的な層を追加します。Red Hat OpenShift Service Mesh サポートは、特別なサイドカープロキシーをマイクロサービス間のネットワーク通信をすべてインターセプトするメッシュ内の関連サービスにデプロイすることで、サービスに追加できます。Service Mesh コントロールプレーンの機能を使用して Service Mesh を設定し、管理します。

Red Hat OpenShift Service Mesh により、以下を提供するデプロイされたサービスのネットワークを簡単に作成できます。

- 検出
- 負荷分散
- サービス間の認証
- 障害回復
- メトリクス
- モニタリング

Red Hat OpenShift Service Mesh は、以下を含むより複雑な運用機能も提供します。

- A/B テスト
- カナリアリリース
- アクセス制御
- エンドツーエンド認証

2.2.2. Red Hat OpenShift Service Mesh アーキテクチャー

Red Hat OpenShift Service Mesh は、データプレーンとコントロールプレーンに論理的に分割されます。

データプレーン は、サイドカーコンテナーとしてデプロイされたインテリジェントプロキシーのセットです。これらのプロキシーは、サービスメッシュ内のマイクロサービス間の受信および送信ネットワーク通信をすべてインターセプトし、制御します。サイドカープロキシーは、Mixer、汎用ポリシーおよび Telemetry ハブとも通信します。

- **Envoy** プロキシー は、サービスメッシュ内の全サービスの受信トラフィックおよび送信トラフィックをすべてインターセプトします。Envoy は、同じ Pod の関連するサービスに対してサイドカーコンテナーとしてデプロイされます。

コントロールプレーン は、プロキシーがトラフィックをルーティングするように管理および設定し、Mixer がポリシーを適用し、Telemetry を収集するように設定します。

- **Mixer** は、アクセス制御と使用ポリシー (認可、レート制限、クォータ、認証、および要求トレースなど) を適用し、Envoy プロキシーやその他のサービスから Telemetry データを収集します。

- **Pilot** はランタイム時にプロキシーを設定します。Pilot は、Envoy サイドカーコンテナーのサービス検出、インテリジェントルーティング (例: A/B テストまたはカナリアデプロイメント) のトラフィック管理機能、および回復性 (タイムアウト、再試行、サーキットブレーカー) を提供します。

- **Citadel** は証明書を発行し、ローテーションします。Citadel は、組み込み型のアイデンティティーおよび認証情報の管理機能を使用して、強力なサービス間認証およびエンドユーザー認証を提供します。Citadel を使用して、サービスメッシュで暗号化されていないトラフィックをアップグレードできます。Operator は、Citadel を使用して、ネットワーク制御ではなく、サービスアイデンティティーに基づいてポリシーを適用することができます。

- **Galley** は、サービスメッシュ設定を取り込み、その後設定を検証し、処理し、配布します。Galley は、他のサービスメッシュコンポーネントが OpenShift Container Platform からユーザー設定の詳細を取得できないようにします。

Red Hat OpenShift Service Mesh は、**istio-operator** を使用してコントロールプレーンのインストールも管理します。**Operator** は、OpenShift Container Platform クラスターで共通アクティビティーを実装し、自動化できるようにするソフトウェアの一部です。これはコントローラーとして動作し、クラスター内の必要なオブジェクトの状態を設定したり、変更したりできます。

2.2.3. Kiali について

Kiali は、サービスメッシュのマイクロサービスとそれらの接続方法を表示してサービスメッシュを可視化します。

2.2.3.1. Kiali の概要

Kiali では、OpenShift Container Platform で実行される Service Mesh の可観測性 (Observability) を提供します。Kiali は、Istio サービスメッシュの定義、検証、および確認に役立ちます。これは、トポロジーの推測によりサービスメッシュの構造を理解しやすくし、またサービスメッシュの健全性に関する情報も提供します。

Kiali は、サーキットブレーカー、要求レート、レイテンシー、トラフィックフローのグラフなどの機能を可視化する、namespace のインタラクティブなグラフビューをリアルタイムで提供します。Kiali では、異なるレベルのコンポーネント (アプリケーションからサービスおよびワークロードまで) についての洞察を提供し、選択されたグラフノードまたはエッジに関するコンテキスト情報やチャートを含む対話を表示できます。Kiali は、ゲートウェイ、宛先ルール、仮想サービス、メッシュポリシーなど、Istio 設定を検証する機能も提供します。Kiali は詳細なメトリクスを提供し、基本的な Grafana 統合は高度なクエリーに利用できます。Jaeger を Kiali コンソールに統合することで、分散トレースを提供します。

Kiali は、デフォルトで Red Hat OpenShift Service Mesh の一部としてインストールされます。

2.2.3.2. Kiali アーキテクチャー

Kiali はオープンソースの Kiali プロジェクト に基づいています。Kiali は Kiali アプリケーションと Kiali コンソールという 2 つのコンポーネントで設定されます。

- **Kiali** アプリケーション (バックエンド): このコンポーネントはコンテナーアプリケーションプラットフォームで実行され、サービスメッシュコンポーネントと通信し、データを取得し、処理し、そのデータをコンソールに公開します。Kiali アプリケーションはストレージを必要としません。アプリケーションをクラスターにデプロイする場合、設定は ConfigMap およびシークレットに設定されます。

- **Kiali** コンソール (フロントエンド): Kiali コンソールは Web アプリケーションです。Kiali アプリケーションは Kiali コンソールを提供し、データをユーザーに表示するためにバックエンドに対してデータのクエリーを実行します。

さらに Kiali は、コンテナーアプリケーションプラットフォームと Istio が提供する外部サービスとコンポーネントに依存します。

- **Red Hat Service Mesh** (Istio): Istio は Kiali の要件です。Istio はサービスメッシュを提供し、制御するコンポーネントです。Kiali と Istio を個別にインストールすることはできますが、Kiali は Istio に依存し、Istio が存在しない場合は機能しません。Kiali は、Prometheus および Cluster API 経由で公開される Istio データおよび設定を取得する必要があります。

- **Prometheus**: 専用の Prometheus インスタンスは Red Hat OpenShift Service Mesh インストールの一部として組み込まれています。Istio Telemetry が有効にされている場合、メトリクスデータは Prometheus に保存されます。Kiali はこの Prometheus データを使用して、メッシュトポロジーの判別、メトリクスの表示、健全性の算出、可能性のある問題の表示などを行います。Kiali は Prometheus と直接通信し、Istio Telemetry で使用されるデータスキーマを想定します。Prometheus は Istio に依存しており、Kiali と明示的な依存関係があるため、Kiali の機能の多くは Prometheus なしに機能しません。

- **Cluster API**: Kiali はサービスメッシュ設定を取得し、解決するために、OpenShift Container Platform (Cluster API) の API を使用します。Kiali は Cluster API に対してクエリーを実行し、たとえば、namespace、サービス、デプロイメント、Pod、その他のエンティティーの定義を取得します。Kiali はクエリーを実行して、異なるクラスターエンティティー間の関係も解決します。Cluster API に対してもクエリーを実行し、仮想サービス、宛先ルール、ルートルール、ゲートウェイ、クォータなどの Istio 設定を取得します。

- **Jaeger**: Jaeger はオプションですが、Red Hat OpenShift Service Mesh インストールの一部としてデフォルトでインストールされます。デフォルトの Red Hat OpenShift Service Mesh インストールの一部として分散トレースプラットフォームをインストールすると、Kiali コンソールには分散トレースデータを表示するタブが含まれます。Istio の分散トレース機能を無効にした場合、トレースデータは利用できないことに注意してください。また、トレースデータを表示するには、ユーザーは Service Mesh コントロールプレーンがインストールされている namespace にアクセスできる必要があります。

- **Grafana**: Grafana はオプションですが、デフォルトでは Red Hat OpenShift Service Mesh インストールの一部としてインストールされます。使用可能な場合、Kiali のメトリクスページには Grafana 内の同じメトリクスにユーザーを移動させるリンクが表示されます。Grafana ダッシュボードへのリンクと Grafana データを表示するには、Service Mesh コントロールプレーンがインストールされている namespace にユーザーがアクセスできる必要があることに注意してください。

2.2.3.3. Kiali の機能

Kiali コンソールは Red Hat Service Mesh に統合され、以下の機能を提供します。

- 健全性: アプリケーション、サービス、またはワークロードの問題を素早く特定します。

- トポロジー: Kiali グラフを使用して、アプリケーション、サービス、またはワークロードの通信方法を可視化します。

- メトリクス: 事前定義済みのメトリクスダッシュボードを使用すると、Go、Node.js、Quarkus、Spring Boot、Thorntail、および Vert.xまた、独自のカスタムダッシュボードを作成することもできます。

- トレース: Jaeger との統合により、アプリケーションを設定するさまざまなマイクロサービスで要求のパスを追跡できます。

- 検証: 最も一般的な Istio オブジェクト (宛先ルール、サービスエントリー、仮想サービスなど) で高度な検証を実行します。
- 設定: ウィザードを使用するか、または Kiali コンソールの YAML エディターを直接使用して、Istio ルーティング設定を作成し、更新し、削除できるオプションの機能です。

2.2.4. Jaeger について

ユーザーがアプリケーションでアクションを実行するたびに、応答を生成するために多数の異なるサービスに参加を要求する可能性のあるアーキテクチャーによって要求が実行されます。この要求のパスは分散トランザクションです。Jaeger を使用すると、分散トレースを実行できます。これは、アプリケーションを設定するさまざまなマイクロサービスを介して要求のパスを追跡します。

分散トレースは、さまざまな作業ユニットの情報を連携させるために使用される技術です。これは、分散トランザクションでのイベントのチェーン全体を理解するために、通常さまざまなプロセスまたはホストで実行されます。分散トレースを使用すると、開発者は大規模なサービス指向アーキテクチャーで呼び出しフローを可視化できます。シリアル化、並行処理、およびレイテンシーのソースについて理解しておくことも重要です。

Jaeger はマイクロサービスのスタック全体での個々の要求の実行を記録し、トレースとして表示します。トレース とは、システムにおけるデータ/実行パスです。エンドツーエンドトレースは、1 つ以上のスパンで設定されます。

スパン は、オペレーション名、オペレーションの開始時間および期間を持つ、Jaeger の作業の論理単位を表しています。スパンは因果関係をモデル化するためにネスト化され、順序付けられます。

2.2.4.1. 分散トレースの概要

サービスの所有者は、分散トレースを使用してサービスをインストルメント化し、サービスアーキテクチャーに関する洞察を得ることができます。分散トレースを使用して、現代的なクラウドネイティブのマイクロサービスベースのアプリケーションにおける、コンポーネント間の対話の監視、ネットワークプロファイリング、およびトラブルシューティングを行うことができます。

分散トレースを使用すると、以下の機能を実行できます。

- 分散トランザクションの監視
- パフォーマンスとレイテンシーの最適化
- 根本原因分析の実行

Red Hat OpenShift の分散トレースは、2 つの主要コンポーネントで設定されています。

- **Red Hat OpenShift** 分散トレースプラットフォーム: このコンポーネントは、オープンソースの Jaeger プロジェクト に基づいています。
- **Red Hat OpenShift** 分散トレースデータ収集: このコンポーネントは、オープンソースの OpenTelemetry プロジェクト に基づいています。

これらのコンポーネントは共に、特定のベンダーに依存しない OpenTracing API およびインストルメンテーションに基づいています。

2.2.4.2. 分散トレースのアーキテクチャー

分散トレースプラットフォームは、オープンソースの Jaeger プロジェクト に基づいています。分散トレースプラットフォームは、複数のコンポーネントで設定されており、トレースデータを収集し、保存し、表示するためにそれらが連携します。

- **Jaeger Client** (Tracer、Reporter、インストルメント化されたアプリケーション、クライアントライブラリー): Jaeger クライアントは、OpenTracing API の言語固有の実装です。それらは、手動または (Camel (Fuse)、Spring Boot (RHOAR)、MicroProfile (RHOAR/Thorntail)、Wildfly (EAP)、その他 OpenTracing にすでに統合されているものを含む) 各種の既存オープンソースフレームワークを使用して、分散トレース用にアプリケーションをインストルメント化するために使用できます。

- **Jaeger Agent** (Server Queue、Processor Worker): Jaeger エージェントは、User Datagram Protocol (UDP) で送信されるスパンをリッスンするネットワークデーモンで、コレクターにバッチ処理や送信を実行します。このエージェントは、インストルメント化されたアプリケーションと同じホストに配置されることが意図されています。これは通常、Kubernetes などのコンテナー環境にサイドカーコンテナーを配置することによって実行されます。

- **Jaeger Collector** (Queue、Worker): エージェントと同様に、コレクターはスパンを受信でき、これらを処理するために内部キューに配置できます。これにより、コレクターはスパンがストレージに移動するまで待機せずに、クライアント/エージェントにすぐに戻ることができます。

- **Storage** (Data Store): コレクターには永続ストレージのバックエンドが必要です。Jaeger には、スパンストレージ用のプラグ可能なメカニズムがあります。本リリースでは、サポートされているストレージは Elasticsearch のみであることに注意してください。

- **Query** (Query Service): Query は、ストレージからトレースを取得するサービスです。

- **Ingester** (Ingester Service): Jaeger は Apache Kafka をコレクターと実際のバッキングストレージ (Elasticsearch) 間のバッファーとして使用できます。Ingester は、Kafka からデータを読み取り、別のストレージバックエンド (Elasticsearch) に書き込むサービスです。

- **Jaeger Console**: Jaeger は、分散トレースデータを視覚化できるユーザーインターフェイスを提供します。検索ページで、トレースを検索し、個別のトレースを設定するスパンの詳細を確認することができます。

2.2.4.3. Red Hat OpenShift 分散トレースの機能

Red Hat OpenShift 分散トレースには、以下の機能が含まれます。

- Kiali との統合: 適切に設定されている場合、Kiali コンソールから分散トレースデータを表示できます。

- 高いスケーラビリティー: 分散トレースバックエンドは、単一障害点がなく、ビジネスニーズに合わせてスケーリングできるように設計されています。

- 分散コンテキストの伝播: さまざまなコンポーネントからのデータをつなぎ、完全なエンドツーエンドトレースを作成できます。

- Zipkin との後方互換性: Red Hat OpenShift 分散トレースには、Zipkin のドロップイン置き換えで使用できるようにする API がありますが、本リリースでは、Red Hat は Zipkin の互換性をサポートしていません。

2.2.5. 次のステップ

- OpenShift Container Platform 環境で Red Hat OpenShift Service Mesh をインストールする準備 をします。

2.3. SERVICE MESH と ISTIO の相違点

警告

こちらは、サポートされなくなった **Red Hat OpenShift Service Mesh** リリースのドキュメントです。

Service Mesh バージョン 1.0 および 1.1 コントロールプレーンはサポートされなくなりました。Service Mesh コントロールプレーンのアップグレードについては、Service Mesh の アップグレード を参照してください。

特定の Red Hat Service Mesh リリースのサポートステータスについては、製品ライフサイクルページ を参照してください。

Red Hat OpenShift Service Mesh のインストールは、多くの点でアップストリームの Istio コミュニティーインストールとは異なります。Red Hat OpenShift Service Mesh の変更点は、問題の解決、追加機能の提供、OpenShift Container Platform へのデプロイ時の差異の処理を実行するために必要になることがあります。

Red Hat OpenShift Service Mesh の現行リリースは、以下の点で現在のアップストリーム Istio コミュニティーのリリースとは異なります。

2.3.1. マルチテナントインストール

アップストリームの Istio は単一テナントのアプローチをとりますが、Red Hat OpenShift Service Mesh はクラスター内で複数の独立したコントロールプレーンをサポートします。Red Hat OpenShift Service Mesh はマルチテナント Operator を使用して、コントロールプレーンのライフサイクルを管理します。

Red Hat OpenShift Service Mesh は、デフォルトでマルチテナントコントロールプレーンをインストールします。Service Meshにアクセスできるプロジェクトを指定し、Service Meshを他のコントロールプレーンインスタンスから分離します。

2.3.1.1. マルチテナンシーとクラスター全体のインストールの比較

マルチテナントインストールとクラスター全体のインストールの主な違いは、istod で使用される権限の範囲です。コンポーネントでは、クラスタースコープのロールベースのアクセス制御 (RBAC) リソース **ClusterRoleBinding** が使用されなくなりました。

ServiceMeshMemberRoll members 一覧のすべてのプロジェクトには、コントロールプレーンのデプロイメントに関連付けられた各サービスアカウントの **RoleBinding** があり、各コントロールプレーンのデプロイメントはそれらのメンバープロジェクトのみを監視します。各メンバープロジェクトには **maistra.io/member-of** ラベルが追加されており、**member-of** の値はコントロールプレーンのインストールが含まれるプロジェクトになります。

Red Hat OpenShift Service Mesh は各メンバープロジェクトを設定し、それ自体、コントロールプレーン、および他のメンバープロジェクト間のネットワークアクセスを確保できるようにします。正確な設定は、OpenShift Container Platform のソフトウェア定義ネットワーク (SDN) の設定方法によって異なります。詳細は、OpenShift SDN についてを参照してください。

OpenShift Container Platform クラスターが SDN プラグインを使用するように設定されている場合:

- **NetworkPolicy**: Red Hat OpenShift Service Mesh は、各メンバープロジェクトで **NetworkPolicy** リソースを作成し、他のメンバーおよびコントロールプレーンからのすべての Pod に対する Ingress を許可します。Service Meshからメンバーを削除すると、この **NetworkPolicy** リソースがプロジェクトから削除されます。

  注記

  また、これにより Ingress がメンバープロジェクトのみに制限されます。メンバー以外のプロジェクトの Ingress が必要な場合は、**NetworkPolicy** を作成してそのトラフィックを許可する必要があります。

- **Multitenant**: Red Hat OpenShift Service Mesh は、各メンバープロジェクトの **NetNamespace** をコントロールプレーンプロジェクトの **NetNamespace** に追加します (**oc adm pod-network join-projects --to control-plane-project member-project** の実行と同じです)。Service Meshからメンバーを削除すると、その **NetNamespace** はコントロールプレーンから分離されます (**oc adm pod-network isolate-projects member-project** の実行と同じです)。

- **Subnet**: 追加の設定は実行されません。

2.3.1.2. クラスタースコープのリソース

アップストリーム Istio には、依存するクラスタースコープのリソースが 2 つあります。**MeshPolicy** および **ClusterRbacConfig**。これらはマルチテナントクラスターと互換性がなく、以下で説明されているように置き換えられました。

- コントロールプレーン全体の認証ポリシーを設定するために、MeshPolicy は **ServiceMeshPolicy** に置き換えられます。これは、コントロールプレーンと同じプロジェクトに作成する必要があります。

- コントロールプレーン全体のロールベースのアクセス制御を設定するために、ClusterRbacConfig は **ServicemeshRbacConfig** に置き換えられます。これは、コントロールプレーンと同じプロジェクトに作成する必要があります。

2.3.2. Istio と Red Hat OpenShift Service Mesh の相違点

Red Hat OpenShift Service Mesh のインストールは、多くの点で Istio のインストールとは異なります。Red Hat OpenShift Service Mesh の変更点は、問題の解決、追加機能の提供、OpenShift Container Platform へのデプロイ時の差異の処理を実行するために必要になることがあります。

2.3.2.1. コマンドラインツール

Red Hat OpenShift Service Mesh のコマンドラインツールは oc です。 Red Hat OpenShift Service Mesh は、istioctl をサポートしません。

2.3.2.2. 自動的な挿入

アップストリームの Istio コミュニティーインストールは、ラベル付けしたプロジェクト内の Pod にサイドカーコンテナーを自動的に挿入します。

Red Hat OpenShift Service Mesh は、サイドカーコンテナーをあらゆる Pod に自動的に挿入することはなく、プロジェクトにラベルを付けることなくアノテーションを使用して挿入をオプトインする必要があります。この方法で必要となる権限は少なく、ビルダー Pod などの他の OpenShift 機能と競合し

ません。自動挿入を有効にするには、サイドカーの自動挿入セクションで説明されているように **sidecar.istio.io/inject** アノテーションを指定します。

2.3.2.3. Istio ロールベースアクセス制御機能

Istio ロールベースアクセス制御機能 (RBAC) は、サービスへのアクセスを制御するために使用できるメカニズムを提供します。ユーザー名やプロパティーのセットを指定してサブジェクトを特定し、それに応じてアクセス制御を適用することができます。

アップストリームの Istio コミュニティーインストールには、ヘッダーの完全一致の実行、ヘッダーのワイルドカードの一致の実行、または特定の接頭辞または接尾辞を含むヘッダーの有無をチェックするオプションが含まれます。

Red Hat OpenShift Service Mesh は、正規表現を使用して要求ヘッダーと一致させる機能を拡張します。**request.regex.headers** のプロパティーキーを正規表現で指定します。

アップストリーム Istio コミュニティーの要求ヘッダーのマッチング例

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

Red Hat OpenShift Service Mesh の正規表現による要求ヘッダーのマッチング

```
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 では、BoringSSL を OpenSSL に置き換えます。OpenSSL は、Secure Sockets Layer (SSL) プロトコルおよび Transport Layer Security (TLS) プロトコルのオープンソース実装を含むソフトウェアライブラリーです。Red Hat OpenShift Service Mesh Proxy バイナリーは、基礎となる Red Hat Enterprise Linux オペレーティングシステムから OpenSSL ライブラリー (libssl および libcrypto) を動的にリンクします。

2.3.2.5. コンポーネントの変更

- すべてのリソースに **maistra-version** ラベルが追加されました。

- すべての Ingress リソースが OpenShift ルートリソースに変換されました。
- Grafana、トレース (Jaeger)、および Kiali はデフォルトで有効にされ、OpenShift ルート経由で公開されます。
- すべてのテンプレートから Godebug が削除されました。
- **istio-multi** ServiceAccount および ClusterRoleBinding が削除されました。また、 **istio-reader** ClusterRole も削除されました。

2.3.2.6. Envoy、シークレット検出サービス、および証明書

- Red Hat OpenShift Service Mesh は、QUIC ベースのサービスをサポートしません。
- Istio の Secret Discovery Service (SDS) 機能を使用した TLS 証明書のデプロイメントは、現在 Red Hat OpenShift Service Mesh ではサポートされていません。Istio 実装は、hostPath マウントを使用する nodeagent コンテナーに依存します。

2.3.2.7. Istio Container Network Interface (CNI) プラグイン

Red Hat OpenShift Service Mesh には CNI プラグインが含まれ、アプリケーション Pod ネットワーキングを設定する代替の方法が提供されます。CNI プラグインは **init-container** ネットワーク設定を置き換えます。これにより、昇格した権限でサービスアカウントおよびプロジェクトに SCC (Security Context Constraints) へのアクセスを付与する必要がなくなります。

2.3.2.8. Istio ゲートウェイのルート

Istio ゲートウェイの OpenShift ルートは、Red Hat OpenShift Service Mesh で自動的に管理されます。Istio ゲートウェイがサービスメッシュ内で作成され、更新され、削除されるたびに、OpenShift ルートが作成され、更新され、削除されます。

Istio OpenShift Routing (IOR) と呼ばれる Red Hat OpenShift Service Mesh コントロールプレーンコンポーネントは、ゲートウェイルートを同期させます。詳細は、自動ルートの作成について参照してください。

2.3.2.8.1. catch-all ドメイン

catch-all ドメイン ("*") はサポートされません。ゲートウェイ定義で catch-all ドメインが見つかった場合、Red Hat OpenShift Service Mesh はルートを 作成します が、デフォルトのホスト名を作成するには OpenShift に依存します。つまり、新たに作成されたルートは、catch all ("*") ルート ではなく、代わりに **<route-name>[-<project>].<suffix>** 形式のホスト名を持ちます。デフォルトのホスト名の仕組みや、クラスター管理者がカスタマイズできる仕組みについての詳細は、OpenShift ドキュメントを参照してください。

2.3.2.8.2. サブドメイン

サブドメイン (e.g.: "*.domain.com") はサポートされます。ただし、この機能は OpenShift Container Platform ではデフォルトで有効になっていません。つまり、Red Hat OpenShift Service Mesh はサブドメインを持つルートを 作成します が、これは OpenShift Container Platform が有効にするように設定されている場合にのみ有効になります。

2.3.2.8.3. トランスポート層セキュリティー

トランスポート層セキュリティー (TLS) がサポートされます。ゲートウェイに **tls** セクションが含まれる場合、OpenShift ルートは TLS をサポートするように設定されます。

関連情報

- 自動ルート作成

2.3.3. Kiali とサービスメッシュ

OpenShift Container Platform での Service Mesh を使用した Kiali のインストールは、複数の点でコミュニティーの Kiali インストールとは異なります。以下の変更点は、問題の解決、追加機能の提供、OpenShift Container Platform へのデプロイ時の差異の処理を実行するために必要になることがあります。

- Kiali はデフォルトで有効になっています。
- Ingress はデフォルトで有効になっている。
- Kiali ConfigMap が更新されている。
- Kiali の ClusterRole 設定が更新されている。
- 変更は Service Mesh または Kiali Operator によって上書きされる可能性があるため、ConfigMap を編集しないでください。Kiali Operator が管理するファイルには、**kiali.io**/ ラベルまたはアノテーションが付いています。Operator ファイルの更新は、**cluster-admin** 権限を持つユーザーに制限する必要があります。Red Hat OpenShift Dedicated を使用する場合に、**dedicated-admin** 権限のあるユーザーだけが Operator ファイルを更新できるようにする必要があります。

2.3.4. 分散トレースとサービスメッシュ

OpenShift Container Platform での Service Mesh を使用した分散トレースプラットフォームのインストールは、複数の点でコミュニティーの Jaeger インストールとは異なります。以下の変更点は、問題の解決、追加機能の提供、OpenShift Container Platform へのデプロイ時の差異の処理を実行するために必要になることがあります。

- 分散トレースは、Service Mesh に対してデフォルトで有効にされています。
- Ingress は、Service Mesh に対してデフォルトで有効にされています。
- Zipkin ポート名が、(**http** から) **jaeger-collector-zipkin** に変更されています。
- Jaeger は、**production** または **streaming** デプロイメントオプションのいずれかを選択する際に、デフォルトでストレージに Elasticsearch を使用します。
- Istio のコミュニティーバージョンは、一般的なトレースルートを提供します。Red Hat OpenShift Service Mesh は Red Hat OpenShift 分散トレーシプラットフォーム Operator によってインストールされ、OAuth によってすでに保護されている jaeger ルートを使用します。
- Red Hat OpenShift Service Mesh は Envoy プロキシーにサイドカーを使用し、Jaeger も Jaeger エージェントにサイドカーを使用します。両者は個別に設定し、混同しないようにしてください。プロキシーサイドカーは、Pod の Ingress および Egress トラフィックに関連するスパンを作成します。エージェントサイドカーは、アプリケーションによって出力されるスパンを受け取り、これらを Jaeger Collector に送信します。

2.4. SERVICE MESH のインストールの準備

警告

こちらは、サポートされなくなった **Red Hat OpenShift Service Mesh** リリースのドキュメントです。

Service Mesh バージョン 1.0 および 1.1 コントロールプレーンはサポートされなくなりました。Service Mesh コントロールプレーンのアップグレードについては、Service Mesh の アップグレード を参照してください。

特定の Red Hat Service Mesh リリースのサポートステータスについては、製品ライフサイクルページ を参照してください。

Red Hat OpenShift Service Mesh をインストールするには、インストールアクティビティーを確認し、前提条件を満たしていることを確認してください。

2.4.1. 前提条件

- お使いの Red Hat アカウントに有効な OpenShift Container Platform サブスクリプションを用意します。サブスクリプションをお持ちでない場合は、営業担当者にお問い合わせください。
- OpenShift Container Platform 4.7 の概要 を確認します。
- OpenShift Container Platform 4.7 をインストールします。
  - AWS への OpenShift Container Platform 4.7 のインストール
  - ユーザーによってプロビジョニングされた AWS への OpenShift Container Platform 4.7 のインストール
  - ベアメタルへの OpenShift Container Platform 4.7 のインストール
  - vSphere への OpenShift Container Platform 4.7 のインストール

注記

制限されたネットワーク に Red Hat OpenShift Service Mesh をインストールする場合、選択した OpenShift Container Platform インフラストラクチャーの手順に従います。

- OpenShift Container Platform バージョンに一致する OpenShift Container Platform コマンドラインユーティリティーのバージョン (**oc** クライアントツール) をインストールし、これをパスに追加します。
  - OpenShift Container Platform 4.7 を使用している場合は、OpenShift CLI について を参照してください。

2.4.2. Red Hat OpenShift Service Mesh でサポートされている設定

以下は、Red Hat OpenShift Service Mesh で唯一サポートされている設定です。

- OpenShift Container Platform バージョン 4.6 以降。

注記

OpenShift Online および Red Hat OpenShift Dedicated は Red Hat OpenShift Service Mesh に対してはサポートされていません。

- デプロイメントは、フェデレーションされていない単一の OpenShift Container Platform クラスターに含まれる必要があります。
- Red Hat OpenShift Service Mesh の本リリースは、OpenShift Container Platform x86_64 でのみ利用できます。
- 本リリースでは、すべての Service Mesh コンポーネントが OpenShift Container Platform クラスターに含まれ、動作している設定のみをサポートしています。クラスター外にあるマイクロサービスの管理や、マルチクラスターシナリオにおけるマイクロサービスの管理はサポートしていません。
- 本リリースでは、仮想マシンなどの外部サービスを統合していない設定のみをサポートしています。

Red Hat OpenShift Service Mesh のライフサイクルおよびサポートされる設定についての詳細は、サポートポリシー について参照してください。

2.4.2.1. Red Hat OpenShift Service Mesh でサポートされている Kiali の設定

- Kiali の可観測性コンソールは Chrome、Edge、Firefox、または Safari ブラウザーの 2 つの最新リリースでのみサポートされています。

2.4.2.2. サポートされている Mixer アダプター

- 本リリースでは、次の Mixer アダプターのみをサポートしています。
  - 3scale Istio Adapter

2.4.3. Operator の概要

Red Hat OpenShift Service Mesh には、以下の 4 つの Operator が必要です。

- **OpenShift Elasticsearch**:(オプション) 分散トレースプラットフォームでのトレースおよびロギング用にデータベースストレージを提供します。これはオープンソースの Elasticsearch プロジェクトに基づいています。
- **Red Hat OpenShift** 分散トレースプラットフォーム: 複雑な分散システムでのトランザクションを監視し、トラブルシューティングするための分散トレース機能を提供します。これはオープンソースの Jaeger プロジェクトに基づいています。
- **Kiali**: サービスメッシュの可観測性を実現します。これにより、単一のコンソールで設定を表示し、トラフィックを監視し、トレースの分析を実行できます。これはオープンソースの Kiali プロジェクトに基づいています。
- **Red Hat OpenShift Service Mesh**: アプリケーションを設定するマイクロサービスを接続し、保護し、制御し、観察することができます。Service Mesh Operator は、Service Mesh コンポーネントのデプロイメント、更新、および削除を管理する **ServiceMeshControlPlane** リソースを定義し、監視します。これはオープンソースの Istio プロジェクトに基づいています。

警告

実稼働環境で Elasticsearch のデフォルトの Jaeger パラメーターを設定する方法についての詳細は、ログストアの設定 を参照してください。

2.4.4. 次のステップ

- OpenShift Container Platform 環境に Red Hat OpenShift Service Mesh をインストール します。

2.5. SERVICE MESH のインストール

警告

こちらは、サポートされなくなった **Red Hat OpenShift Service Mesh** リリースのドキュメントです。

Service Mesh バージョン 1.0 および 1.1 コントロールプレーンはサポートされなくなりました。Service Mesh コントロールプレーンのアップグレードについては、Service Mesh の アップグレード を参照してください。

特定の Red Hat Service Mesh リリースのサポートステータスについては、製品ライフサイクルページ を参照してください。

Service Mesh のインストールには、OpenShift Elasticsearch、Jaeger、Kiali、Service Mesh Operator のインストール、コントロールプレーンをデプロイするための **ServiceMeshControlPlane** リソースの作成および管理、Service Mesh に関連する namespace を指定するための **ServiceMeshMemberRoll** リソースの作成が含まれます。

注記

Mixer のポリシーの適用はデフォルトで無効にされています。ポリシータスクを実行するには、これを有効にする必要があります。Mixer ポリシーの適用を有効にする方法については、Mixer ポリシーの適用の更新 を参照してください。

注記

マルチテナントコントロールプレーンのインストールがデフォルトの設定です。

注記

Service Mesh に関するドキュメントは **istio-system** をサンプルプロジェクトとして使用しますが、サービスメッシュを任意のプロジェクトにデプロイできます。

2.5.1. 前提条件

- Red Hat OpenShift Service Mesh のインストールの準備 のプロセスに従ってください。
- **cluster-admin** ロールを持つアカウントがある。

Service Mesh のインストールプロセスでは、OperatorHub を使用して **openshift-operators** プロジェクト内に **ServiceMeshControlPlane** カスタムリソース定義をインストールします。Red Hat OpenShift Service Mesh は、コントロールプレーンのデプロイメント、更新、および削除に関連する **ServiceMeshControlPlane** を定義し、監視します。

Red Hat OpenShift Service Mesh 1.1.18.2 以降では、Red Hat OpenShift Service Mesh Operator がコントロールプレーンをインストールする前に、OpenShift Elasticsearch Operator、Jaeger Operator、および Kiali Operator をインストールする必要があります。

2.5.2. OpenShift Elasticsearch Operator のインストール

デフォルトの Red Hat OpenShift 分散トレースプラットフォームのデプロイメントでは、インメモリーのストレージが使用されます。これは、Red Hat OpenShift 分散トレースの評価、デモの提供、またはテスト環境での Red Hat OpenShift 分散トレースプラットフォームの使用を希望するユーザー用に、迅速にインストール行うために設計されているためです。実稼働環境で Red Hat OpenShift 分散トレースプラットフォームを使用する予定がある場合、永続ストレージのオプション (この場合は Elasticsearch) をインストールし、設定する必要があります。

前提条件

- OpenShift Container Platform Web コンソールへのアクセスがある。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

警告

Operator のコミュニティーバージョンはインストールしないでください。コミュニティー Operator はサポートされていません。

注記

OpenShift Logging の一部として OpenShift Elasticsearch Operator がすでにインストールされている場合、OpenShift Elasticsearch Operator を再びインストールする必要はありません。Red Hat OpenShift 分散トレースプラットフォーム Operator はインストールされた OpenShift Elasticsearch Operator を使用して Elasticsearch インスタンスを作成します。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
2. **Operators → OperatorHub** に移動します。

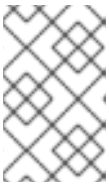
3. **Elasticsearch** とフィルターボックスに入力して、OpenShift Elasticsearch Operator を検索します。

4. Red Hat が提供する **OpenShift Elasticsearch Operator** をクリックし、Operator についての情報を表示します。

5. **Install** をクリックします。

6. **Install Operator** ページで、**stable** 更新チャネルを選択します。これにより、新しいバージョンがリリースされると Operator が自動的に更新されます。

7. デフォルトの **All namespaces on the cluster (default)**を受け入れます。これにより、Operator がデフォルトの **openshift-operators-redhat** プロジェクトにインストールされ、Operator はクラスター内のすべてのプロジェクトで利用可能になります。



注記

Elasticsearch インストールでは、 Elasticsearch Operator に **openshift-operators-redhat** namespace が必要です。他の Red Hat OpenShift 分散トレース Operator は、**openshift-operators** namespace にインストールされます。

- デフォルトの **Automatic** 承認ストラテジーを受け入れます。デフォルトを受け入れることで、Operator の新規バージョンが利用可能になると、Operator Lifecycle Manager (OLM) は人の介入なしに、Operator の実行中のインスタンスを自動的にアップグレードします。手動 更新を選択する場合、Operator の新規バージョンが利用可能になると、OLM は更新要求を作成します。クラスター管理者は、Operator が新規バージョンに更新されるように更新要求を手動で承認する必要があります。

注記

手動 の承認ストラテジーには、Operator のインストールおよびサブスクリプションプロセスを承認するための適切な認証情報を持つユーザーが必要です。

8. **Install** をクリックします。

9. **Installed Operators** ページで、**openshift-operators-redhat** プロジェクトを選択します。OpenShift Elasticsearch Operator が InstallSucceeded のステータスを表示するまで待機してから続行します。

2.5.3. Red Hat OpenShift 分散トレースプラットフォーム Operator のインストール

Red Hat OpenShift 分散トレースプラットフォームをインストールするには、OperatorHub を使用して Red Hat OpenShift 分散トレースプラットフォーム Operator をインストールします。

デフォルトでは、Operator は **openshift-operators** プロジェクトにインストールされます。

前提条件

- OpenShift Container Platform Web コンソールへのアクセスがある。

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

- 永続ストレージが必要な場合、Red Hat OpenShift 分散トレースプラットフォーム Operator をインストールする前に OpenShift Elasticsearch Operator もインストールする必要があります。

警告

Operator のコミュニティーバージョンはインストールしないでください。コミュニティー Operator はサポートされていません。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
2. **Operators → OperatorHub** に移動します。
3. フィルターに **distributing tracing platform** と入力して、Red Hat OpenShift 分散トレースプラットフォーム Operator を探します。
4. Red Hat が提供する **Red Hat OpenShift distributed tracing platform Operator**をクリックし、Operator についての情報を表示します。
5. **Install** をクリックします。
6. **Install Operator** ページで、**stable** 更新チャネルを選択します。これにより、新しいバージョンがリリースされると Operator が自動的に更新されます。
7. デフォルトの **All namespaces on the cluster (default)**を受け入れます。これにより、Operator がデフォルトの **openshift-operators** プロジェクトにインストールされ、Operator はクラスター内のすべてのプロジェクトで利用可能になります。
   - デフォルトの **Automatic** 承認ストラテジーを受け入れます。デフォルトを受け入れることで、Operator の新規バージョンが利用可能になると、Operator Lifecycle Manager (OLM) は人の介入なしに、Operator の実行中のインスタンスを自動的にアップグレードします。手動 更新を選択する場合、Operator の新規バージョンが利用可能になると、OLM は更新要求を作成します。クラスター管理者は、Operator が新規バージョンに更新されるように更新要求を手動で承認する必要があります。

     注記

     手動 の承認ストラテジーには、Operator のインストールおよびサブスクリプションプロセスを承認するための適切な認証情報を持つユーザーが必要です。
8. **Install** をクリックします。
9. **Operators → Installed Operators** に移動します。
10. **Installed Operators** ページで、**openshift-operators** プロジェクトを選択します。Red Hat OpenShift 分散トレースプラットフォーム Operator が Succeeded のステータスを表示するまで待機してから続行します。

2.5.4. Kiali Operator のインストール

Service Mesh コントロールプレーンをインストールするには、Red Hat OpenShift Service Mesh Operator の Kiali Operator をインストールする必要があります。

> **警告**
>
> Operator のコミュニティーバージョンはインストールしないでください。コミュニティー Operator はサポートされていません。

前提条件

- OpenShift Container Platform Web コンソールにアクセスします。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operators → OperatorHub** に移動します。
3. **Kiali** とフィルターボックスに入力して、Kiali Operator を検索します。
4. Red Hat が提供する **Kiali Operator** をクリックし、Operator についての情報を表示します。
5. **Install** をクリックします。
6. **Operator Installation** ページで、**stable** 更新チャネルを選択します。
7. **All namespaces on the cluster(default)** を選択します。これにより、Operator がデフォルトの **openshift-operators** プロジェクトにインストールされ、Operator はクラスター内のすべてのプロジェクトで利用可能になります。
8. **Automatic** Approval Strategy を選択します。

   > **注記**
   >
   > 手動の承認ストラテジーには、Operator のインストールおよびサブスクリプションプロセスを承認するための適切な認証情報を持つユーザーが必要です。

9. **Install** をクリックします。
10. **Installed Operators** ページには、Kiali Operator のインストールの進捗状況が表示されます。

2.5.5. Operator のインストール

Red Hat OpenShift Service Mesh をインストールするには、以下の Operator をこの順序でインストールします。Operator ごとに手順を繰り返します。

- OpenShift Elasticsearch

- Red Hat OpenShift 分散トレースプラットフォーム
- Kiali
- Red Hat OpenShift Service Mesh

注記

OpenShift Logging の一部として OpenShift Elasticsearch Operator がすでにインストールされている場合、OpenShift Elasticsearch Operator を再びインストールする必要はありません。Red Hat OpenShift 分散トレースプラットフォーム Operator はインストールされた OpenShift Elasticsearch Operator を使用して Elasticsearch インスタンスを作成します。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
2. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** をクリックします。
3. Operator のフィルターボックスに名前を入力し、Red Hat バージョンの Operator を選択します。Operator のコミュニティーバージョンはサポートされていません。
4. **Install** をクリックします。
5. 各 Operator の **Install Operator** ページで、デフォルト設定を受け入れます。
6. **Install** をクリックします。Operator がインストールされるまで待機してから、一覧で次に来る Operator で手順を繰り返します。
   - OpenShift Elasticsearch Operator は、**openshift-operators-redhat** namespace にインストールされ、クラスター内のすべての namespace で使用できます。
   - Red Hat OpenShift 分散トレースプラットフォームは、**openshift-distributed-tracing** namespace にインストールされ、クラスター内のすべての namespace で使用できます。
   - Kiali および Red Hat Service Mesh Operator は、**openshift-operators** namespace にインストールされ、クラスター内のすべての namespace で使用できます。
7. 4 つの Operator すべてをインストールしたら、**Operators → Installed Operators** をクリックし、Operator がインストールされていることを確認します。

2.5.6. Red Hat OpenShift Service Mesh コントロールプレーンのデプロイ

ServiceMeshControlPlane リソースは、インストール時に使用される設定を定義します。Red Hat が提供するデフォルト設定をデプロイするか、またはビジネスのニーズに合わせて **ServiceMeshControlPlane** ファイルをカスタマイズすることができます。

Web コンソールを使用するか、または **oc** クライアントツールを使用してコマンドラインから Service Mesh コントロールプレーンをデプロイすることができます。

2.5.6.1. Web コンソールを使用したコントロールプレーンのデプロイ

以下の手順に従って、Web コンソールを使用して Red Hat OpenShift Service Mesh コントロールプレーンをデプロイします。この例では、**istio-system** は、コントロールプレーンプロジェクトです。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている必要がある。
- Red Hat OpenShift Service Mesh のインストールのカスタマイズ方法についての手順を確認します。
- **cluster-admin** ロールを持つアカウントがある。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。
2. **istio-system** という名前のプロジェクトを作成します。
   a. **Home → Projects** に移動します。
   b. **Create Project** をクリックします。
   c. **Name** フィールドに **istio-system** を入力します。
   d. **Create** をクリックします。
3. **Operators → Installed Operators** に移動します。
4. 必要な場合は、Project メニューから **istio-system** を選択します。Operator が新規プロジェクトにコピーされるまでに数分待機する必要がある場合があります。
5. Red Hat OpenShift Service Mesh Operator をクリックします。**Provided APIs** の下に、Operator は 2 つのリソースタイプを作成するためのリンクを提供します。
   - **ServiceMeshControlPlane** リソース
   - **ServiceMeshMemberRoll** リソース
6. **Istio Service Mesh Control Plane**で **Create ServiceMeshControlPlane** をクリックします。
7. **Create Service Mesh Control Plane**ページで、必要に応じてデフォルト **ServiceMeshControlPlane** テンプレートの YAML を変更します。

   注記

   コントロールプレーンのカスタマイズについての詳細は、Red Hat OpenShift Service Mesh インストールのカスタマイズについて参照してください。実稼働環境の場合は、デフォルトの Jaeger テンプレートを変更する 必要 があります。

8. **Create** をクリックしてコントロールプレーンを作成します。Operator は、設定パラメーターに基づいて Pod、サービス、Service Mesh コントロールプレーンのコンポーネントを作成します。
9. **Istio Service Mesh Control Plane**タブをクリックします。

10. 新規コントロールプレーンの名前をクリックします。

11. **Resources** タブをクリックして、Red Hat OpenShift Service Mesh コントロールプレーンリソース (Operator が作成し、設定したもの) を表示します。

2.5.6.2. CLI からのコントロールプレーンのデプロイ

以下の手順に従って、CLI を使用して Red Hat OpenShift Service Mesh コントロールプレーンをデプロイします。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている必要がある。
- Red Hat OpenShift Service Mesh のインストールのカスタマイズ方法についての手順を確認します。
- **cluster-admin** ロールを持つアカウントがある。
- OpenShift CLI (**oc**) へのアクセスがある。

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。

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

2. **istio-system** という名前のプロジェクトを作成します。

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

3. Customize the Red Hat OpenShift Service Mesh installation にある例を使用して、**istio-installation.yaml** という名前の **ServiceMeshControlPlane** ファイルを作成します。必要に応じて値をカスタマイズして、ユースケースに合わせて使用することができます。実稼働デプロイメントの場合は、デフォルトの Jaeger テンプレートを変更する 必要 があります。

4. 以下のコマンドを実行してコントロールプレーンをデプロイします。

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

5. 以下のコマンドを実行して、コントロールプレーンのインストールのステータスを確認します。

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

STATUS 列が **ComponentsReady** の場合、インストールは正常に終了しています。

```
NAME           READY   STATUS           PROFILES      VERSION   AGE
basic-install   11/11   ComponentsReady   ["default"]   v1.1.18   4m25s
```

6. 以下のコマンドを実行して、インストールプロセス時の Pod の進捗を確認します。

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

以下のような出力が表示されるはずです。

出力例

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

マルチテナントインストールでは、Red Hat OpenShift Service Mesh はクラスター内で複数の独立したコントロールプレーンをサポートします。**ServiceMeshControlPlane** テンプレートを使用すると、再利用可能な設定を作成することができます。詳細は、コントロールプレーンテンプレートの作成 を参照してください。

2.5.7. Red Hat OpenShift Service Mesh メンバーロールの作成

ServiceMeshMemberRoll は、Service Mesh コントロールプレーンに属するプロジェクトを一覧表示します。**ServiceMeshMemberRoll** に一覧表示されているプロジェクトのみがコントロールプレーンの影響を受けます。プロジェクトは、特定のコントロールプレーンのデプロイメント用にメンバーロールに追加するまでサービスメッシュに属しません。

istio-system など、**ServiceMeshControlPlane** と同じプロジェクトに、 **default** という名前の **ServiceMeshMemberRoll** リソースを作成する必要があります。

2.5.7.1. Web コンソールからのメンバーロールの作成

Web コンソールを使用して 1 つ以上のプロジェクトを Service Mesh メンバーロールに追加します。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされ、検証されていること。
- サービスメッシュに追加する既存プロジェクトの一覧。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. メッシュのサービスがない場合や、ゼロから作業を開始する場合は、アプリケーションのプロジェクトを作成します。これは、Service Mesh コントロールプレーンをインストールしたプロジェクトとは異なる必要があります。

a. **Home → Projects** に移動します。

b. **Name** フィールドに名前を入力します。

c. **Create** をクリックします。

3. **Operators → Installed Operators** に移動します。

4. **Project** メニューをクリックし、一覧から **ServiceMeshControlPlane** リソースがデプロイされているプロジェクト (例: **istio-system**) を選択します。

5. Red Hat OpenShift Service Mesh Operator をクリックします。

6. **Istio Service Mesh Member Roll**タブをクリックします。

7. **Create ServiceMeshMemberRoll** をクリックします。

8. **Members** をクリックし、**Value** フィールドにプロジェクトの名前を入力します。任意の数のプロジェクトを追加できますが、プロジェクトは 単一 の **ServiceMeshMemberRoll** リソースしか属することができません。

9. **Create** をクリックします。

2.5.7.2. CLI からのメンバーロールの作成

コマンドラインからプロジェクトを **ServiceMeshMemberRoll** に追加します。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされ、検証されていること。
- サービスメッシュに追加するプロジェクトの一覧。
- OpenShift CLI (**oc**) へのアクセスがある。

手順

1. OpenShift Container Platform CLI にログインします。

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

2. メッシュのサービスがない場合や、ゼロから作業を開始する場合は、アプリケーションのプロジェクトを作成します。これは、Service Mesh コントロールプレーンをインストールしたプロジェクトとは異なる必要があります。

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

3. プロジェクトをメンバーとして追加するには、以下の YAML の例を変更します。任意の数のプロジェクトを追加できますが、プロジェクトは 単一 の **ServiceMeshMemberRoll** リソースしか属することができません。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

servicemeshmemberroll-default.yaml の例

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

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

4. 以下のコマンドを実行して、**istio-system** namespace に **ServiceMeshMemberRoll** リソースをアップロードおよび作成します。

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

5. 以下のコマンドを実行して、**ServiceMeshMemberRoll** が正常に作成されていることを確認します。

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

STATUS 列が **Configured** の場合には、インストールは正常に終了しています。

2.5.8. サービスメッシュからのプロジェクトの追加または削除

Web コンソールを使用して既存の Service Mesh **ServiceMeshMemberRoll** リソースを追加または削除できます。

- 任意の数のプロジェクトを追加できますが、プロジェクトは 単一 の **ServiceMeshMemberRoll** リソースしか属することができません。
- **ServiceMeshMemberRoll** リソースは、対応する **ServiceMeshControlPlane** リソースが削除されると削除されます。

2.5.8.1. Web コンソールを使用したメンバーロールからのプロジェクトの追加または削除

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされ、検証されていること。
- 既存の **ServiceMeshMemberRoll** リソース。
- **ServiceMeshMemberRoll** リソースを持つプロジェクトの名前。
- メッシュに/から追加または削除するプロジェクトの名前。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operators → Installed Operators** に移動します。
3. **Project** メニューをクリックし、一覧から **ServiceMeshControlPlane** リソースがデプロイされているプロジェクト (例: **istio-system**) を選択します。

4. Red Hat OpenShift Service Mesh Operator をクリックします。
5. **Istio Service Mesh Member Roll**タブをクリックします。
6. **default** リンクをクリックします。
7. YAML タブをクリックします。
8. YAML を変更して、プロジェクトをメンバーとして追加または削除します。任意の数のプロジェクトを追加できますが、プロジェクトは 単一 の **ServiceMeshMemberRoll** リソースしか属することができません。
9. **Save** をクリックします。
10. **Reload** をクリックします。

2.5.8.2. CLI を使用したメンバーロールからのプロジェクトの追加または削除

コマンドラインを使用して既存の Service Mesh Member Roll を変更できます。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされ、検証されていること。
- 既存の **ServiceMeshMemberRoll** リソース。
- **ServiceMeshMemberRoll** リソースを持つプロジェクトの名前。
- メッシュに/から追加または削除するプロジェクトの名前。
- OpenShift CLI (**oc**) へのアクセスがある。

手順

1. OpenShift Container Platform CLI にログインします。
2. **ServiceMeshMemberRoll** リソースを編集します。

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

3. YAML を変更して、プロジェクトをメンバーとして追加または削除します。任意の数のプロジェクトを追加できますが、プロジェクトは 単一 の **ServiceMeshMemberRoll** リソースしか属することができません。

servicemeshmemberroll-default.yaml の例

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

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

2.5.9. 手動更新

手動で更新することを選択する場合、Operator Lifecycle Manager (OLM) は、クラスター内の Operator のインストール、アップグレード、ロールベースのアクセス制御 (RBAC) を制御します。OLM はデフォルトで OpenShift Container Platform で実行されます。OLM は CatalogSource を使用します。これは Operator Registry API を使用して利用可能な Operator やインストールされた Operator のアップグレードについてクエリーします。

- OpenShift Container Platform のアップグレードの処理方法についての詳細は、Operator Lifecycle Manager のドキュメントを参照してください。

2.5.9.1. サイドカープロキシーの更新

サイドカープロキシーの設定を更新するには、アプリケーション管理者はアプリケーション Pod を再起動する必要があります。

デプロイメントで自動のサイドカーコンテナー挿入を使用する場合、アノテーションを追加または変更してデプロイメントの Pod テンプレートを更新することができます。以下のコマンドを実行して Pod を再デプロイします。

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

デプロイメントで自動のサイドカーコンテナー挿入を使用しない場合、デプロイメントまたは Pod で指定されたサイドカーコンテナーイメージを変更して Pod を再起動し、サイドカーコンテナーを手動で更新する必要があります。

2.5.10. 次のステップ

- Red Hat OpenShift Service Mesh で アプリケーションをデプロイする準備 をします。

2.6. SERVICE MESH のセキュリティーのカスタマイズ

警告

こちらは、サポートされなくなった **Red Hat OpenShift Service Mesh** リリースのドキュメントです。

Service Mesh バージョン 1.0 および 1.1 コントロールプレーンはサポートされなくなりました。Service Mesh コントロールプレーンのアップグレードについては、Service Mesh の アップグレード を参照してください。

特定の Red Hat Service Mesh リリースのサポートステータスについては、製品ライフサイクルページ を参照してください。

サービスメッシュアプリケーションが複雑な配列のマイクロサービスで構築されている場合、Red Hat OpenShift Service Mesh を使用してそれらのサービス間の通信のセキュリティーをカスタマイズできます。Service Mesh のトラフィック管理機能と共に OpenShift Container Platform のインフラストラクチャーを使用すると、アプリケーションの複雑性を管理し、マイクロサービスのサービスおよびアイデンティティーのセキュリティーを提供することができます。

2.6.1. 相互トランスポート層セキュリティー (mTLS) の有効化

相互トランスポート層セキュリティー (mTLS) は、二者が相互の認証を行うプロトコルです。これは、一部のプロトコル (IKE、SSH) での認証のデフォルトモードであり、他のプロトコル (TLS) ではオプションになります。

mTLS は、アプリケーションやサービスコードを変更せずに使用できます。TLS は、サービスメッシュインフラストラクチャーおよび 2 つのサイドカープロキシー間で完全に処理されます。

デフォルトで、Red Hat OpenShift Service Mesh は Permissive モードに設定されます。この場合、Service Mesh のサイドカーは、プレーンテキストのトラフィックと mTLS を使用して暗号化される接続の両方を受け入れます。メッシュのサービスがメッシュ外のサービスと通信している場合、厳密な mTLS によりサービス間の通信に障害が発生する可能性があります。ワークロードを Service Mesh に移行する間に Permissive モードを使用します。

2.6.1.1. メッシュ全体での厳密な mTLS の有効化

ワークロードがメッシュ外のサービスと通信せず、暗号化された接続のみを受け入れることで通信が中断されない場合は、メッシュ全体で mTLS をすぐに有効にできます。**ServiceMeshControlPlane** リソースで **spec.istio.global.mtls.enabled** を **true** に設定します。Operator は必要なリソースを作成します。

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

2.6.1.1.1. 特定のサービスの受信接続用のサイドカーの設定

ポリシーを作成して、個別のサービスまたは namespace に mTLS を設定することもできます。

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

2.6.1.2. 送信接続用のサイドカーの設定

宛先ルールを作成し、Service Mesh がメッシュ内の他のサービスに要求を送信する際に mTLS を使用するように設定します。

```
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. 最小および最大のプロトコルバージョンの設定

ご使用の環境のサービスメッシュに暗号化されたトラフィックの特定の要件がある場合、許可される暗号化機能を制御できます。これは、**ServiceMeshControlPlane** リソースに **spec.security.controlPlane.tls.minProtocolVersion** または **spec.security.controlPlane.tls.maxProtocolVersion** を設定して許可できます。コントロールプレーンリソースで設定されるこれらの値は、TLS 経由でセキュアに通信する場合にメッシュコンポーネントによって使用される最小および最大の TLS バージョンを定義します。

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    global:
      tls:
        minProtocolVersion: TLSv1_2
        maxProtocolVersion: TLSv1_3
```

デフォルトは **TLS_AUTO** であり、TLS のバージョンは指定しません。

表2.3 有効な値

値	説明
TLS_AUTO	default
TLSv1_0	TLS バージョン 1.0
TLSv1_1	TLS バージョン 1.1
TLSv1_2	TLS バージョン 1.2
TLSv1_3	TLS バージョン 1.3

2.6.2. 暗号化スイートおよび ECDH 曲線の設定

暗号化スイートおよび ECDH 曲線 (Elliptic-curve Diffie-Hellman) は、サービスメッシュのセキュリティーを保護するのに役立ちます。暗号化スイートのコンマ区切りの一覧を **spec.istio.global.tls.cipherSuites** を使用して定義し、 ECDH 曲線を **ServiceMeshControlPlane** リソースの **spec.istio.global.tls.ecdhCurves** を使用して定義できます。これらの属性のいずれかが空の場合、デフォルト値が使用されます。

サービスメッシュが TLS 1.2 以前のバージョンを使用する場合、**cipherSuites** 設定が有効になります。この設定は、TLS 1.3 でネゴシエートする場合は影響を与えません。

コンマ区切りの一覧に暗号化スイートを優先度順に設定します。たとえば、**ecdhCurves: CurveP256, CurveP384** は、**CurveP256** を **CurveP384** よりも高い優先順位として設定します。

注記

暗号化スイートを設定する場合は、**TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256** または **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256** のいずれかを追加する必要があります。HTTP/2 のサポートには、1 つ以上の以下の暗号スイートが必要です。

サポートされている暗号化スイートは以下になります。

- 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

サポートされる ECDH 曲線は以下のとおりです。

- CurveP256
- CurveP384

- CurveP521
- X25519

2.6.3. 外部認証局キーおよび証明書の追加

デフォルトで、Red Hat OpenShift Service Mesh は自己署名ルート証明書およびキーを生成し、それらを使用してワークロード証明書に署名します。ユーザー定義の証明書およびキーを使用して、ユーザー定義のルート証明書を使用してワークロード証明書に署名することもできます。このタスクは、証明書およびキーを Service Mesh にプラグインするサンプルを示しています。

前提条件

- 証明書を設定するには、相互 TLS を有効にして Red Hat OpenShift Service Mesh をインストールしておく必要があります。
- この例では、Maistra リポジトリー からの証明書を使用します。実稼働環境の場合は、認証局から独自の証明書を使用します。
- これらの手順で結果を確認するには、Bookinfo サンプルアプリケーションをデプロイする必要があります。

2.6.3.1. 既存の証明書およびキーの追加

既存の署名 (CA) 証明書およびキーを使用するには、CA 証明書、キー、ルート証明書が含まれる信頼ファイルのチェーンを作成する必要があります。それぞれの対応する証明書について以下のファイル名をそのまま使用する必要があります。CA 証明書は **ca-cert.pem** と呼ばれ、キーは **ca-key.pem** であり、**ca-cert.pem** を署名するルート証明書は **root-cert.pem** と呼ばれます。ワークロードで中間証明書を使用する場合は、**cert-chain.pem** ファイルでそれらを指定する必要があります。

以下の手順に従い、証明書を Service Mesh に追加します。Maistra リポジトリー からのサンプル証明書をローカルに保存し、**<path>** を証明書へのパスに置き換えます。

1. シークレット **cacert** を作成します。これには、入力ファイルの **ca-cert.pem**、**ca-key.pem**、**root-cert.pem** および **cert-chain.pem** が含まれます。

```
$ 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. **ServiceMeshControlPlane** リソースで **global.mtls.enabled** を **true** に設定し、**security.selfSigned** を **false** に設定します。Service Mesh は、secret-mount ファイルから証明書およびキーを読み取ります。

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

3. ワークロードが新規証明書を追加することをすぐに確認するには、**istio.*** という名前の Service Mesh によって生成されたシークレットを削除します。この例では **istio.default** です。Service Mesh はワークロードの新規の証明書を発行します。

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

2.6.3.2. 証明書の確認

Bookinfo サンプルアプリケーションを使用して、証明書が正しくマウントされていることを確認します。最初に、マウントされた証明書を取得します。次に、Pod にマウントされた証明書を確認します。

1. Pod 名を変数 **RATINGSPOD** に保存します。

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

2. 以下のコマンドを実行して、プロキシーにマウントされている証明書を取得します。

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

/tmp/pod-root-cert.pem ファイルには、Pod に伝播されるルート証明書が含まれます。

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

/tmp/pod-cert-chain.pem ファイルには、ワークロード証明書と Pod に伝播される CA 証明書が含まれます。

3. ルート証明書が Operator によって指定される証明書と同じであることを確認します。**<path>** を証明書へのパスに置き換えます。

```
$ 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
```

出力が空になることが予想されます。

4. CA 証明書が Operator で指定された証明書と同じであることを確認します。**<path>** を証明書へのパスに置き換えます。

```
$ 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
```

出力が空になることが予想されます。

5. ルート証明書からワークロード証明書への証明書チェーンを確認します。**<path>** を証明書へのパスに置き換えます。

```
$ 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
```

出力例

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

2.6.3.3. 証明書の削除

追加した証明書を削除するには、以下の手順に従います。

1. シークレット **cacerts** を削除します。

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

2. **ServiceMeshControlPlane** リソースで自己署名ルート証明書を使用して Service Mesh を再デプロイします。

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

2.7. トラフィック管理

警告

こちらは、サポートされなくなった **Red Hat OpenShift Service Mesh** リリースのドキュメントです。

Service Mesh バージョン 1.0 および 1.1 コントロールプレーンはサポートされなくなりました。Service Mesh コントロールプレーンのアップグレードについては、Service Mesh の アップグレード を参照してください。

特定の Red Hat Service Mesh リリースのサポートステータスについては、製品ライフサイクルページ を参照してください。

Red Hat OpenShift Service Mesh のサービス間のトラフィックのフローおよび API 呼び出しを制御できます。たとえば、サービスメッシュの一部のサービスはメッシュ内で通信する必要があり、他のサービスは非表示にする必要がある場合があります。トラフィックを管理して、特定のバックエンドサービスを非表示にし、サービスを公開し、テストまたはバージョン管理デプロイメントを作成し、または一連のサービスのセキュリティーの層を追加します。

2.7.1. ゲートウェイの使用

ゲートウェイを使用してメッシュの受信トラフィックおよび送信トラフィックを管理することで、メッシュに入るか、またはメッシュを出るトラフィックを指定できます。ゲートウェイ設定は、サービスワークロードと共に実行されるサイドカー Envoy プロキシーではなく、メッシュのエッジで実行されているスタンドアロンの Envoy プロキシーに適用されます。

Kubernetes Ingress API などのシステムに入るトラフィックを制御する他のメカニズムとは異なり、Red Hat OpenShift Service Mesh ゲートウェイではトラフィックのルーティングの機能および柔軟性を最大限に利用できます。Red Hat OpenShift Service Mesh ゲートウェイリソースは、Red Hat OpenShift Service Mesh TLS 設定を公開して設定するポートなど、4-6 の負荷分散プロパティーを階層化できます。アプリケーション層のトラフィックルーティング (L7) を同じ API リソースに追加する代わりに、通常の Red Hat OpenShift Service Mesh 仮想サービスをゲートウェイにバインドし、サービスメッシュ内の他のデータプレーントラフィックのようにゲートウェイトラフィックを管理することができます。

ゲートウェイは ingress トラフィックの管理に主に使用されますが、egress ゲートウェイを設定することもできます。egress ゲートウェイを使用すると、メッシュからのトラフィック専用の終了ノードを設定できます。これにより、サービスメッシュにセキュリティー制御を追加することで、外部ネットワークにアクセスできるサービスを制限できます。また、ゲートウェイを使用して完全に内部のプロキシーを設定することもできます。

ゲートウェイの例

ゲートウェイリソースは、着信または発信 HTTP/TCP 接続を受信するメッシュのエッジで動作するロードバランサーを表します。この仕様は、公開する必要のあるポートのセット、使用するプロトコルのタイプ、ロードバランサー用の SNI 設定などについて記述します。

以下の例は、外部 HTTPS Ingress トラフィックのゲートウェイ設定を示しています。

```
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
```

このゲートウェイ設定により、ポート 443 での **ext-host.example.com** からメッシュへの HTTPS トラフィックが可能になりますが、トラフィックのルーティングは指定されません。

ルーティングを指定し、ゲートウェイが意図される通りに機能するには、ゲートウェイを仮想サービスにバインドする必要もあります。これは、以下の例のように、仮想サービスのゲートウェイフィールドを使用して実行します。

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

次に、仮想サービスを外部トラフィックのルーティングルールを使用して設定できます。

2.7.2. Ingress ゲートウェイの設定

Ingress ゲートウェイは、受信 HTTP/TCP 接続を受信するメッシュのエッジで稼働するロードバランサーです。このゲートウェイは、公開されるポートおよびプロトコルを設定しますが、これにはトラフィックルーティングの設定は含まれません。Ingress トラフィックに対するトラフィックルーティングは、内部サービス要求の場合と同様に、ルーティングルールで設定されます。

以下の手順では、ゲートウェイを作成し、**/productpage** と **/login** のパスの外部トラフィックに、Bookinfo サンプルアプリケーションのサービスを公開するように、**VirtualService** を設定します。

手順

1. トラフィックを受け入れるゲートウェイを作成します。

   a. YAML ファイルを作成し、以下の YAML をこれにコピーします。

      ゲートウェイの例 (gateway.yaml)

      ```
      apiVersion: networking.istio.io/v1alpha3
      kind: Gateway
      metadata:
        name: bookinfo-gateway
      spec:
        selector:
          istio: ingressgateway
        servers:
        - port:
            number: 80
            name: http
            protocol: HTTP
          hosts:
          - "*"
      ```

   b. YAML ファイルを適用します。

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

2. **VirtualService** オブジェクトを作成し、ホストヘッダーを再作成します。

   a. YAML ファイルを作成し、以下の YAML をこれにコピーします。

   仮想サービスの例

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

   b. YAML ファイルを適用します。

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

3. ゲートウェイと VirtualService が正しく設定されていることを確認してください。

   a. ゲートウェイ URL を設定します。

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

   b. ポート番号を設定します。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

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

   c. 明示的に公開されているページをテストします。

```
curl -s -I "$GATEWAY_URL/productpage"
```

想定される結果は **200** です。

2.7.3. Ingress トラフィックの管理

Red Hat OpenShift Service Mesh では、Ingress Gateway は、モニタリング、セキュリティー、ルートルールなどの機能をクラスターに入るトラフィックに適用できるようにします。Service Meshゲートウェイを使用してサービスメッシュ外のサービスを公開します。

2.7.3.1. Ingress IP およびポートの判別

Ingress 設定は、環境が外部ロードバランサーをサポートするかどうかによって異なります。外部ロードバランサーはクラスターの Ingress IP およびポートに設定されます。クラスターの IP およびポートが外部ロードバランサーに設定されているかどうかを判別するには、以下のコマンドを実行します。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

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

このコマンドは、namespace のそれぞれの項目の **NAME**、**TYPE**、**CLUSTER-IP**、**EXTERNAL-IP**、**PORT(S)**、および **AGE** を返します。

EXTERNAL-IP 値が設定されている場合には、環境には Ingress ゲートウェイに使用できる外部ロードバランサーがあります。

EXTERNAL-IP の値が **<none>** または永続的に **<pending>** の場合、環境は Ingress ゲートウェイの外部ロードバランサーを提供しません。サービスの ノードポート を使用してゲートウェイにアクセスできます。

2.7.3.1.1. ロードバランサーを使用した Ingress ポートの判別

お使いの環境に外部ロードバランサーがある場合には、以下の手順に従います。

手順

1. 以下のコマンドを実行して Ingress IP およびポートを設定します。このコマンドは、ターミナルに変数を設定します。

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

2. 以下のコマンドを実行して Ingress ポートを設定します。

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

3. 以下のコマンドを実行してセキュアな Ingress ポートを設定します。

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

4. 以下のコマンドを実行して TCP Ingress ポートを設定します。

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

注記

一部の環境では、ロードバランサーは IP アドレスの代わりにホスト名を使用して公開される場合があります。この場合、Ingress ゲートウェイの **EXTERNAL-IP** 値は IP アドレスではありません。これはホスト名であり、直前のコマンドは **INGRESS_HOST** 環境変数の設定に失敗します。

失敗した場合には、以下のコマンドを使用して **INGRESS_HOST** 値を修正します。

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

2.7.3.1.2. ロードバランサーのない Ingress ポートの判別

お使いの環境に外部ロードバランサーがない場合は、Ingress ポートを判別し、代わりにノードポートを使用します。

手順

1. Ingress ポートを設定します。

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

2. 以下のコマンドを実行してセキュアな Ingress ポートを設定します。

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

3. 以下のコマンドを実行して TCP Ingress ポートを設定します。

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

2.7.4. 自動ルート作成

Istio ゲートウェイの OpenShift ルートは、Red Hat OpenShift Service Mesh で自動的に管理されます。Istio ゲートウェイがサービスメッシュ内で作成され、更新され、削除されるたびに、OpenShift ルートが作成され、更新され、削除されます。

2.7.4.1. 自動ルート作成の有効化

Istio OpenShift Routing (IOR) と呼ばれる Red Hat OpenShift Service Mesh コントロールプレーンコンポーネントは、ゲートウェイルートを同期させます。コントロールプレーンのデプロイメントの一部として IOR を有効にします。

ゲートウェイに TLS セクションが含まれる場合、OpenShift ルートは TLS をサポートするように設定されます。

1. **ServiceMeshControlPlane** リソースで、**ior_enabled** パラメーターを追加し、これを **true** に設定します。たとえば、以下のリソーススニペットを参照してください。

```
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. サブドメイン

Red Hat OpenShift Service Mesh はサブドメインでルートを作成しますが、OpenShift Container Platform はこれを有効にするように設定される必要があります。***.domain.com** などのサブドメインはサポートされますが、デフォルトでは設定されません。ワイルドカードホストゲートウェイを設定する前に OpenShift Container Platform ワイルドカードポリシーを設定します。詳細はリンクのセクションを参照してください。

以下のゲートウェイが作成される場合は、次のコマンドを実行します。

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

次に、以下の OpenShift ルートが自動的に作成されます。ルートが以下のコマンドを使用して作成されていることを確認できます。

```
$ oc -n <control_plane_namespace> get routes
```

予想される出力

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

このゲートウェイが削除されると、Red Hat OpenShift Service Mesh はルートを削除します。ただし、手動で作成されたルートは Red Hat OpenShift Service Mesh によって変更されることはありません。

2.7.5. サービスエントリーについて

サービスエントリーは、Red Hat OpenShift Service Mesh が内部で維持するサービスレジストリーにエントリーを追加します。サービスエントリーの追加後、Envoy プロキシーはメッシュ内のサービスであるかのようにトラフィックをサービスに送信できます。サービスエントリーを使用すると、以下が可能になります。

- サービスメッシュ外で実行されるサービスのトラフィックを管理します。
- Web から消費される API やレガシーインフラストラクチャーのサービスへのトラフィックなど、外部宛先のトラフィックをリダイレクトし、転送します。
- 外部宛先の再試行、タイムアウト、およびフォールトインジェクションポリシーを定義します。
- 仮想マシンをメッシュに追加して、仮想マシン (VM) でメッシュサービスを実行します。

注記

別のクラスターからメッシュにサービスを追加し、Kubernetes でマルチクラスター Red Hat OpenShift Service Mesh メッシュを設定します。

サービスエントリーの例

以下の mesh-external サービスエントリーの例では、 **ext-resource** の外部依存関係を Red Hat OpenShift Service Mesh サービスレジストリーに追加します。

```
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
```

hosts フィールドを使用して外部リソースを指定します。これを完全に修飾することも、ワイルドカードの接頭辞が付けられたドメイン名を使用することもできます。

仮想サービスおよび宛先ルールを設定して、メッシュ内の他のサービスのトラフィックを設定するのと同じように、サービスエントリーへのトラフィックを制御できます。たとえば、以下の宛先ルールでは、トラフィックルートを、サービスエントリーを使用して設定される **ext-svc.example.com** 外部サービスへの接続のセキュリティーを保護するために相互 TLS を使用するように設定します。

```
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. VirtualServices の使用

仮想サービスを使用して、Red Hat OpenShift Service Mesh で複数バージョンのマイクロサービスに要求を動的にルーティングできます。仮想サービスを使用すると、以下が可能になります。

- 単一の仮想サービスで複数のアプリケーションサービスに対応する。メッシュが Kubernetes を使用する場合などに、仮想サービスを特定の namespace のすべてのサービスを処理するように設定できます。仮想サービスを使用すると、モノリシックなアプリケーションをシームレスに、個別のマイクロサービスで設定されるサービスに変換できます。
- ingress および egress トラフィックを制御できるようにゲートウェイと組み合わせてトラフィックルールを設定する。

2.7.6.1. VirtualServices の設定

要求は、仮想サービスを使用してサービスメッシュ内のサービスにルーティングされます。それぞれの仮想サービスは、順番に評価される一連のルーティングルールで設定されます。Red Hat OpenShift Service Mesh は、仮想サービスへのそれぞれの指定された要求をメッシュ内の特定の実際の宛先に一致させます。

仮想サービスがない場合、Red Hat OpenShift Service Mesh はすべてのサービスインスタンス間のラウンドロビン負荷分散を使用してトラフィックを分散します。仮想サービスを使用すると、1 つ以上のホスト名のトラフィック動作を指定できます。仮想サービスのルーティングルールでは、仮想サービスのトラフィックを適切な宛先に送信する方法を Red Hat OpenShift Service Mesh に指示します。ルートの宛先は、同じサービスのバージョンまたは全く異なるサービスにすることができます。

手順

1. アプリケーションに接続するユーザーに基づき、異なるバージョンの Bookinfo アプリケーションサービスのサンプルに、要求をルーティングする以下の例を使用して、YAML ファイルを作成します。

   VirtualService.yaml の例

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

```
- route:
  - destination:
      host: reviews
      subset: v3
```

2. 以下のコマンドを実行して **VirtualService.yaml** を適用します。**VirtualService.yaml** はファイルへのパスです。

```
$ oc apply -f <VirtualService.yaml>
```

2.7.6.2. VirtualService 設定リファレンス

パラメーター	説明
`spec:` `  hosts:`	**hosts** フィールドには、ルーティングルールが適用される仮想サービスのユーザーの宛先アドレスが一覧表示されます。これは、サービスへの要求送信に使用するアドレスです。仮想サービスのホスト名は、IP アドレス、DNS 名または完全修飾ドメイン名に解決される短縮名になります。

`spec:`
`  http:`
`  - match:`
	http セクションには、ホストフィールドで指定された宛先に送信される HTTP/1.1、HTTP2、および gRPC トラフィックのルーティングの一致条件とアクションを記述する仮想サービスのルーティングルールが含まれます。ルーティングルールは、トラフィックの宛先と、指定の一致条件で設定されます。この例の最初のルーティングルールには条件があり、match フィールドで始まります。この例では、このルーティングはユーザー **jason** からの要求すべてに適用されます。**headers**、**end-user**、および **exact** フィールドを追加し、適切な要求を選択します。

`spec:`
`  http:`
`  - match:`
`  - destination:`
	route セクションの **destination** フィールドは、この条件に一致するトラフィックの実際の宛先を指定します。仮想サービスのホストとは異なり、宛先のホストは Red Hat OpenShift Service Mesh サービスレジストリーに存在する実際の宛先でなければなりません。これは、プロキシーが含まれるメッシュサービス、またはサービスエントリーを使用して追加されたメッシュ以外のサービスである可能性があります。この例では、ホスト名は Kubernetes サービス名です。

2.7.7. 宛先ルールについて

宛先ルールは仮想サービスのルーティングルールが評価された後に適用されるため、それらはトラフィックの実際の宛先に適用されます。仮想サービスはトラフィックを宛先にルーティングします。宛先ルールでは、その宛先のトラフィックに生じる内容を設定します。

デフォルトで、Red Hat OpenShift Service Mesh はラウンドロビンの負荷分散ポリシーを使用します。

このポリシーでは、プールの各サービスインスタンスが順番に要求を取得します。Red Hat OpenShift Service Mesh は以下のモデルもサポートします。このモデルは、特定のサービスまたはサービスサブセットへの要求の宛先ルールに指定できます。

- Random: 要求はプール内のインスタンスにランダムに転送されます。
- Weighted: 要求は特定のパーセンテージに応じてプールのインスタンスに転送されます。
- Least requests: 要求は要求の数が最も少ないインスタンスに転送されます。

宛先ルールの例

以下の宛先ルールの例では、異なる負荷分散ポリシーで **my-svc** 宛先サービスに 3 つの異なるサブセットを設定します。

```
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
```

本書では Bookinfo サンプルアプリケーションを参照して、サンプルアプリケーションでのルーティングの例を説明します。Bookinfo アプリケーション をインストールして、これらのルーティングのサンプルがどのように機能するかを確認します。

2.7.8. Bookinfo ルーティングチュートリアル

Service Mesh Bookinfo サンプルアプリケーションは、それぞれが複数のバージョンを持つ 4 つの別個のマイクロサービスで設定されます。Bookinfo サンプルアプリケーションをインストールした後に、**reviews** マイクロサービスの 3 つの異なるバージョンが同時に実行されます。

ブラウザーで Bookinfo アプリケーションの /**product** ページにアクセスして数回更新すると、書評の出力に星評価が含まれる場合と含まれない場合があります。ルーティング先の明示的なデフォルトサービスバージョンがない場合、Service Mesh は、利用可能なすべてのバージョンに要求をルーティングしていきます。

このチュートリアルは、すべてのトラフィックをマイクロサービスの **v1** (バージョン 1) にルーティングするルールを適用するのに役立ちます。後に、HTTP リクエストヘッダーの値に基づいてトラフィックをルーティングするためのルールを適用できます。

前提条件:

- 以下の例に合わせて Bookinfo サンプルアプリケーションをデプロイする。

2.7.8.1. 仮想サービスの適用

以下の手順では、マイクロサービスのデフォルトバージョンを設定する仮想サービスを適用して、各マイクロサービスの **v1** にすべてのトラフィックをルーティングします。

手順

1. 仮想サービスを適用します。

```
$ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.2/samples/bookinfo/networking/virtual-service-all-v1.yaml
```

2. 仮想サービスの適用を確認するには、以下のコマンドで定義されたルートを表示します。

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

このコマンドでは、YAML 形式で **kind: VirtualService** のリソースを返します。

Service Mesh を Bookinfo マイクロサービスの **v1** バージョン (例: **reviews** サービスバージョン 1) にルーティングするように設定しています。

2.7.8.2. 新規ルート設定のテスト

Bookinfo アプリケーションの /**productpage** を更新して、新しい設定をテストします。

手順

1. **GATEWAY_URL** パラメーターの値を設定します。この変数を使用して、Bookinfo 製品ページの URL を後で見つけることができます。この例では、istio-system はコントロールプレーンプロジェクトの名前です。

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

2. 以下のコマンドを実行して、製品ページの URL を取得します。

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

3. ブラウザーで Bookinfo サイトを開きます。

更新回数に関係なく、ページのレビュー部分は星評価なしに表示されます。これは、Service Mesh を、reviews サービスのすべてのトラフィックをバージョン **reviews:v1** にルーティングするように設定しているためであり、サービスのこのバージョンは星評価サービスにアクセスしません。

サービスメッシュは、トラフィックを 1 つのバージョンのサービスにルーティングするようになりました。

2.7.8.3. ユーザーアイデンティティーに基づくルート

ルート設定を変更して、特定のユーザーからのトラフィックすべてが特定のサービスバージョンにルーティングされるようにします。この場合、**jason** という名前のユーザーからのトラフィックはすべて、サービス **reviews:v2** にルーティングされます。

Service Mesh には、ユーザーアイデンティティーについての特別な組み込み情報はありません。この例は、**productpage** サービスが reviews サービスへのすべてのアウトバウンド HTTP リクエストにカスタム **end-user** ヘッダーを追加することで有効にされます。

手順

1. 以下のコマンドを実行して、Bookinfo アプリケーション例でユーザーベースのルーティングを有効にします。

```
$ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.2/samples/bookinfo/networking/virtual-service-reviews-test-v2.yaml
```

2. 以下のコマンドを実行して、ルールの作成を確認します。このコマンドは、**kind: VirtualService** のすべてのリソースを YAML 形式で返します。

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

3. Bookinfo アプリケーションの /**productpage** で、パスワードなしでユーザー **jason** としてログインします。

4. ブラウザーを更新します。各レビューの横に星評価が表示されます。

5. 別のユーザーとしてログインします (任意の名前を指定します)。ブラウザーを更新します。これで星がなくなりました。Jason 以外のすべてのユーザーのトラフィックが **reviews:v1** にルーティングされるようになりました。

ユーザーアイデンティティーに基づいてトラフィックをルーティングするように Bookinfo のアプリケーションサンプルが正常に設定されています。

2.7.9. 関連情報

OpenShift Container Platform ワイルドカードポリシーの設定に関する詳細は、ワイルドカードルートの使用 を参照してください。

2.8. SERVICE MESH へのアプリケーションのデプロイ

警告

こちらは、サポートされなくなった **Red Hat OpenShift Service Mesh** リリースのドキュメントです。

Service Mesh バージョン 1.0 および 1.1 コントロールプレーンはサポートされなくなりました。Service Mesh コントロールプレーンのアップグレードについては、Service Mesh の アップグレード を参照してください。

特定の Red Hat Service Mesh リリースのサポートステータスについては、製品ライフサイクルページ を参照してください。

アプリケーションを Service Mesh にデプロイする場合、Istio のアップストリームのコミュニティーバージョンのアプリケーションの動作と Red Hat OpenShift Service Mesh インストール内のアプリケーションの動作に違いがいくつかあります。

2.8.1. 前提条件

- Red Hat OpenShift Service Mesh とアップストリーム Istio コミュニティーインストールの比較 について確認する。
- Red Hat OpenShift Service Mesh のインストール について確認する

2.8.2. コントロールプレーンのテンプレートの作成

ServiceMeshControlPlane テンプレートを使用すると、再利用可能な設定を作成することができます。各ユーザーは、作成するテンプレートを独自の設定で拡張することができます。テンプレートは、他のテンプレートから設定情報を継承することもできます。たとえば、会計チーム用の会計コントロールプレーンとマーケティングチーム用のマーケティングコントロールプレーンを作成できます。開発プロファイルと実稼働テンプレートを作成する場合、マーケティングチームおよび会計チームのメンバーは、チーム固有のカスタマイズで開発および実稼働テンプレートを拡張することができます。

ServiceMeshControlPlane と同じ構文に従うコントロールプレーンのテンプレートを設定する場合、ユーザーは階層的に設定を継承します。Operator は、Red Hat OpenShift Service Mesh のデフォルト設定を使用する **default** テンプレートと共に提供されます。カスタムテンプレートを追加するには、**openshift-operators** プロジェクトで **smcp-templates** という名前の ConfigMap を作成し、**/usr/local/share/istio-operator/templates** で Operator コンテナーに ConfigMap をマウントする必要があります。

2.8.2.1. ConfigMap の作成

以下の手順に従って、ConfigMap を作成します。

前提条件

- Service Mesh Operator がインストールされ、検証されていること。
- **cluster-admin** ロールを持つアカウントがある。
- Operator デプロイメントの場所。

- **oc** として知られる OpenShift Container Platform コマンドラインインターフェイス (CLI) へのアクセス。

手順

1. クラスター管理者として OpenShift Container Platform CLI にログインします。
2. CLI で以下のコマンドを実行し、**openshift-operators** プロジェクトに **smcp-templates** という名前の ConfigMap を作成し、**<templates-directory>** をローカルディスクの **ServiceMeshControlPlane** ファイルの場所に置き換えます。

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

3. Operator ClusterServiceVersion 名を見つけます。

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

出力例

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

4. Operator クラスターサービスバージョンを編集して、Operator に **smcp-templates** ConfigMap を使用するよう指示します。

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

5. ボリュームマウントおよびボリュームを Operator デプロイメントに追加します。

```
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. 変更を保存し、エディターを終了します。
7. **ServiceMeshControlPlane** で **template** パラメーターを使用してテンプレートを指定できます。

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

2.8.3. サイドカーコンテナーの自動挿入の有効化

アプリケーションをデプロイする場合は、**deployment** オブジェクトで **spec.template.metadata.annotations** のアノテーション **sidecar.istio.io/inject** を **true** に設定して、インジェクションをオプトインする必要があります。オプトインにより、サイドカーの挿入が OpenShift Container Platform エコシステム内の複数のフレームワークが使用する、ビルダー Pod などの他の OpenShift Container Platform 機能に干渉しないようにします。

前提条件

- サービスメッシュの一部である namespace と、サイドカーの自動注入が必要なデプロイメントを特定しておく。

手順

1. デプロイメントを見つけるには、**oc get** コマンドを使用します。

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

たとえば、**bookinfo** namespaxce の ratings-v1 マイクロサービスのデプロイメントファイルを表示するには、次のコマンドを使用して YAML 形式でリソースを表示します。

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

2. エディターでアプリケーションのデプロイメント設定の YAML ファイルを開きます。

3. 次の例に示すように、**spec.template.metadata.annotations.sidecar.istio/inject** を Deployment YAML に追加し、**sidecar.istio.io/inject** を **true** に設定します。

bookinfo deployment-ratings-v1.yaml のスニペットの例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ratings-v1
  namespace: bookinfo
  labels:
    app: ratings
    version: v1
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: 'true'
```

4. デプロイメント設定ファイルを保存します。

5. ファイルをアプリケーションが含まれるプロジェクトに追加し直します。

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

この例では、**bookinfo** は **ratings-v1** アプリを含むプロジェクトの名前であり、**deployment-ratings-v1.yaml** は編集したファイルです。

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

6. リソースが正常にアップロードされたことを確認するには、以下のコマンドを実行します。

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

以下に例を示します。

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

2.8.4. アノテーションによるプロキシー環境変数の設定

Envoy サイドカープロキシーの設定は、**ServiceMeshControlPlane** によって管理されます。

デプロイメントの Pod アノテーションを **injection-template.yaml** ファイルに追加することにより、アプリケーションのサイドカープロキシーで環境変数を設定できます。環境変数がサイドカーコンテナーに挿入されます。

injection-template.yaml の例

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

警告

独自のカスタムリソースを作成するときは、**maistra.io/** ラベルとアノテーションを含めないでください。これらのラベルとアノテーションは、リソースが Operator によって生成および管理されていることを示しています。独自のリソースの作成時に Operator が生成したリソースからコンテンツをコピーする場合は、**maistra.io/** で始まるラベルやアノテーションを含めないでください。これらのラベルまたはアノテーションを含むリソースは、次回の調整時に Operator によって上書きまたは削除されます。

2.8.5. Mixer ポリシー適用の更新

以前のバージョンの Red Hat OpenShift Service Mesh では、Mixer のポリシーの適用がデフォルトで有効にされていました。Mixer ポリシーの適用はデフォルトで無効になりました。ポリシータスクを実行する前にこれを有効にする必要があります。

前提条件

- **oc** として知られる OpenShift Container Platform コマンドラインインターフェイス (CLI) へのアクセス。

注記

この例では、**<istio-system>** をコントロールプレーン namespace として使用します。この値は、Service Mesh コントロールプレーン (SMCP) をデプロイした namespace に置き換えます。

手順

1. OpenShift Container Platform CLI にログインします。
2. 以下のコマンドを実行して、現在の Mixer ポリシー適用のステータスを確認します。

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

3. **disablePolicyChecks: true** の場合、Service Mesh ConfigMap を編集します。

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

4. ConfigMap 内で **disablePolicyChecks: true** を見つけ、値を **false** に変更します。
5. 設定を保存してエディターを終了します。
6. Mixer ポリシー適用ステータスを再度チェックして、**false** に設定されていることを確認します。

2.8.5.1. 適切なネットワークポリシーの設定

Service Mesh は Service Mesh コントロールプレーンおよびメンバー namespace にネットワークポリシーを作成し、それらの間のトラフィックを許可します。デプロイする前に、以下の条件を考慮し、OpenShift Container Platform ルートで以前に公開されたサービスメッシュのサービスを確認します。

- Istio が適切に機能するには、サービスメッシュへのトラフィックが常に ingress-gateway を経由する必要があります。
- サービスメッシュ外のサービスは、サービスメッシュにない個別の namespace にデプロイします。
- サービスメッシュでリストされた namespace 内にデプロイする必要のあるメッシュ以外のサービスでは、それらのデプロイメント **maistra.io/expose-route: "true"** にラベルを付けます。これにより、これらのサービスへの OpenShift Container Platform ルートは依然として機能します。

2.8.6. Bookinfo のサンプルアプリケーション

Bookinfo のサンプルアプリケーションでは、OpenShift Container Platform での Red Hat OpenShift Service Mesh 2.2.3 のインストールをテストすることができます。

Bookinfo アプリケーションは、オンラインブックストアの単一カタログエントリーのように、書籍に関する情報を表示します。このアプリケーションでは、書籍の説明、書籍の詳細 (ISBN、ページ数その他の情報)、および書評のページが表示されます。

Bookinfo アプリケーションはこれらのマイクロサービスで設定されます。

- **productpage** マイクロサービスは、**details** と **reviews** マイクロサービスを呼び出して、ページを設定します。
- **details** マイクロサービスには書籍の情報が含まれています。
- **reviews** マイクロサービスには、書評が含まれます。これは **ratings** マイクロサービスも呼び出します。
- **ratings** マイクロサービスには、書評を伴う書籍のランキング情報が含まれます。

reviews マイクロサービスには、以下の 3 つのバージョンがあります。

- バージョン v1 は、**ratings** サービスを呼び出しません。
- バージョン v2 は、**ratings** サービスを呼び出して、各評価を 1 から 5 の黒い星で表示します。
- バージョン v3 は、**ratings** サービスを呼び出して、各評価を 1 から 5 の赤い星で表示します。

2.8.6.1. Bookinfo アプリケーションのインストール

このチュートリアルでは、プロジェクトの作成、そのプロジェクトへの Bookinfo アプリケーションのデプロイ、Service Mesh での実行中のアプリケーションの表示を行い、サンプルアプリケーションを作成する方法を説明します。

前提条件:

- OpenShift Container Platform 4.1 以降がインストールされている。
- Red Hat OpenShift Service Mesh 2.2.3 がインストールされている。
- OpenShift CLI (**oc**) へのアクセスがある。

- **cluster-admin** ロールを持つアカウントがある。

注記

Bookinfo サンプルアプリケーションは、IBM Z および IBM Power Systems にインストールできません。

注記

このセクションのコマンドは、Service Mesh コントロールプレーンプロジェクトが **istio-system** であると仮定します。コントロールプレーンを別の namespace にインストールしている場合は、実行する前にそれぞれのコマンドを編集します。

手順

1. cluster-admin 権限を持つユーザーとして OpenShift Container Platform Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。
2. **Home → Projects** をクリックします。
3. **Create Project** をクリックします。
4. **Project Name** として **bookinfo** を入力し、**Display Name** を入力します。その後、**Description** を入力し、**Create** をクリックします。
   - または、CLI からこのコマンドを実行して、**bookinfo** プロジェクトを作成できます。

     ```
     $ oc new-project bookinfo
     ```

5. **Operators → Installed Operators** をクリックします。
6. プロジェクト メニューをクリックし、Service Mesh コントロールプレーンの namespace を使用します。この例では **istio-system** を使用します。
7. **Red Hat OpenShift Service Mesh** Operator をクリックします。
8. **Istio Service Mesh Member Roll**タブをクリックします。
   a. Istio Service Mesh Member Roll がすでに作成されている場合には、名前をクリックしてから YAML タブをクリックし、YAML エディターを開きます。
   b. **ServiceMeshMemberRoll** を作成していない場合は、**Create ServiceMeshMemberRoll** をクリックします。
9. **Members** をクリックし、**Value** フィールドにプロジェクトの名前を入力します。
10. **Create** をクリックして、更新した Service Mesh Member Roll を保存します。
    a. または、以下のサンプルを YAML ファイルに保存します。

Bookinfo ServiceMeshMemberRoll の例 (servicemeshmemberroll-default.yaml)

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
```

```
  name: default
spec:
  members:
  - bookinfo
```

b. 以下のコマンドを実行して、そのファイルをアップロードし、**istio-system** namespace に **ServiceMeshMemberRoll** リソースを作成します。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

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

11. 以下のコマンドを実行して、**ServiceMeshMemberRoll** が正常に作成されていることを確認します。

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

STATUS 列が **Configured** の場合には、インストールは正常に終了しています。

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

12. CLI で **'bookinfo'** プロジェクトに Bookinfo アプリケーションをデプロイするには、**bookinfo.yaml** ファイルを適用します。

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

以下のような出力が表示されるはずです。

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

13. **bookinfo-gateway.yaml** ファイルを適用して Ingress ゲートウェイを作成します。

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

以下のような出力が表示されるはずです。

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

14. **GATEWAY_URL** パラメーターの値を設定します。

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

2.8.6.2. デフォルトの宛先ルールの追加

Bookinfo アプリケーションを使用するには、先にデフォルトの宛先ルールを追加する必要があります。相互トランスポート層セキュリティー (TLS) 認証を有効にしたかどうかによって、2 つの事前設定される YAML ファイルを使用できます。

手順

1. 宛先ルールを追加するには、以下のいずれかのコマンドを実行します。
   - 相互 TLS を有効にしていない場合:

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

   - 相互 TLS を有効にしている場合:

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

     以下のような出力が表示されるはずです。

     ```
     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. Bookinfo インストールの検証

Bookinfo アプリケーションのサンプルが正常にデプロイされたことを確認するには、以下の手順を実行します。

前提条件

- Red Hat OpenShift Service Mesh がインストールされている。
- Bookinfo サンプルアプリケーションのインストール手順を実行します。

CLI からの手順

1. OpenShift Container Platform CLI にログインします。
2. 以下のコマンドですべての Pod が準備状態にあることを確認します。

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

   すべての Pod のステータスは **Running** である必要があります。以下のような出力が表示されるはずです。

```
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. 以下のコマンドを実行して、製品ページの URL を取得します。

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

4. Web ブラウザーで出力をコピーして貼り付けて、Bookinfo の製品ページがデプロイされていることを確認します。

Kiali Web コンソールからの手順

1. Kiali Web コンソールのアドレスを取得します。

   a. **cluster-admin** 権限を持つユーザーとして OpenShift Container Platform Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合) **dedicated-admin** ロールがあるアカウント。

   b. **Networking** → **Routes** に移動します。

   c. **Routes** ページで、 **Namespace** メニューから Service Mesh コントロールプレーンプロジェクトを選択します (例: **istio-system**)。
   Location 列には、各ルートのリンク先アドレスが表示されます。

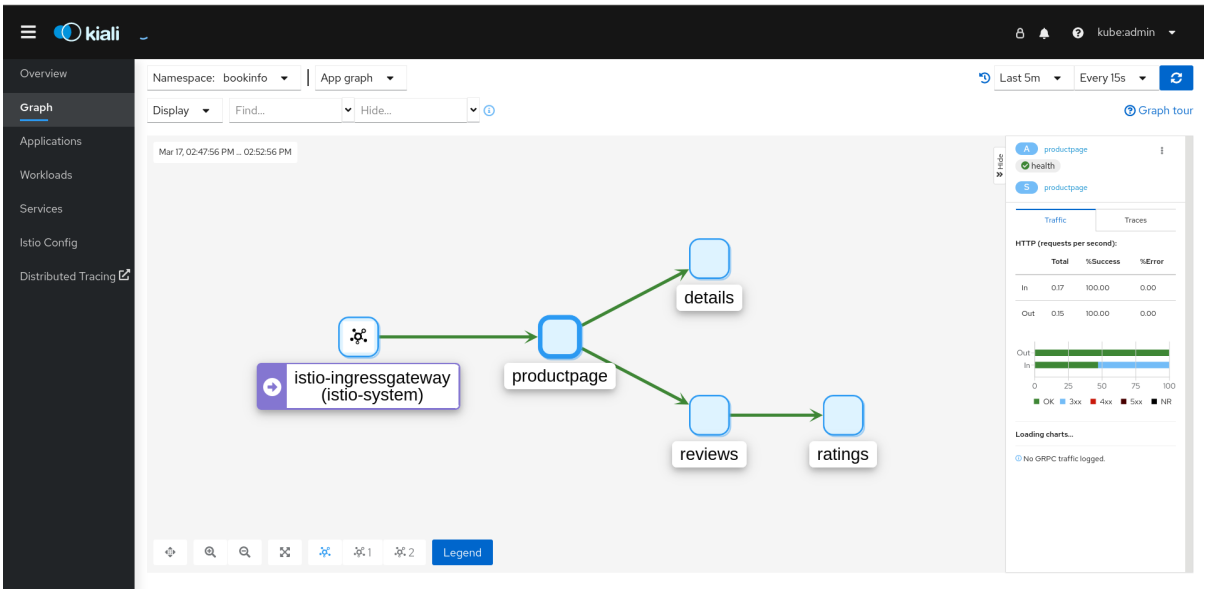
   d. Kiali の 場所 列のリンクをクリックします。

   e. **Log In With OpenShift** をクリックします。Kiali の 概要 画面には、各プロジェクトの namespace のタイルが表示されます。

2. Kiali で、グラフ をクリックします。

3. **Namespace** リストから bookinfo を選択し、 **Graph Type** リストから App graph を選択します。

4. **Display** メニューから **Display idle nodes** をクリックします。
   これにより、定義されているが要求を受信または送信していないノードが表示されます。アプリケーションが適切に定義されていることを確認できますが、要求トラフィックは報告されていません。



- 期間 メニューを使用して、期間を延ばして、古いトラフィックを取得できるようにします。
- **Refresh Rate** メニューを使用して、トラフィックを頻繁に更新するか、まったく更新しないようにします。

5. **Services**、**Workloads** または **Istio Config** をクリックして、bookinfo コンポーネントのリストビューを表示し、それらが正常であることを確認します。

2.8.6.4. Bookinfo アプリケーションの削除

以下の手順で、Bookinfo アプリケーションを削除します。

前提条件

- OpenShift Container Platform 4.1 以降がインストールされている。
- Red Hat OpenShift Service Mesh 2.2.3 がインストールされている。
- OpenShift CLI (**oc**) へのアクセスがある。

2.8.6.4.1. Bookinfo プロジェクトの削除

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Home** → **Projects** をクリックします。
3. **bookinfo** メニュー ⋮ をクリックしてから **Delete Project** をクリックします。
4. 確認ダイアログボックスに **bookinfo** と入力してから **Delete** をクリックします。
   - または、CLI を使用して以下のコマンドを実行し、**bookinfo** プロジェクトを作成できます。

```
$ oc delete project bookinfo
```

2.8.6.4.2. Service Mesh Member Roll からの Bookinfo プロジェクトの削除

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operators → Installed Operators** をクリックします。
3. **Project** メニューをクリックし、一覧から **istio-system** を選択します。
4. **Red Hat OpenShift Service Mesh** Operator の **Provided APIS** で、**Istio Service Mesh Member Roll** のリンクをクリックします。
5. **ServiceMeshMemberRoll** メニュー ⋮ をクリックし、**Edit Service Mesh Member Roll**を選択します。
6. デフォルトの Service Mesh Member Roll YAML を編集し、**members** 一覧から **bookinfo** を削除します。
   - または、CLI を使用して以下のコマンドを実行し、**ServiceMeshMemberRoll** から **bookinfo** プロジェクトを削除できます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前です。

     ```
     $ oc -n istio-system patch --type='json' smmr default -p '[{"op": "remove", "path": "/spec/members", "value":["""bookinfo"""]}]'
     ```
7. **Save** をクリックして、Service Mesh Member Roll を更新します。

2.8.7. サンプルトレースの生成とトレースデータの分析

Jaeger はオープンソースの分散トレースシステムです。Jaeger を使用すると、トレースを実行でき、アプリケーションを設定するさまざまなマイクロサービスで要求のパスを追跡します。Jaeger はデフォルトで Service Mesh の一部としてインストールされます。

このチュートリアルでは、Service Mesh と Bookinfo サンプルアプリケーションを使用して、Jaeger で分散トレースを実行する方法を示します。

前提条件:

- OpenShift Container Platform 4.1 以降がインストールされている。
- Red Hat OpenShift Service Mesh 2.2.3 がインストールされている。
- インストール時に Jaeger が有効にされている。
- Bookinfo のサンプルアプリケーションがインストールされている。

手順

1. Bookinfo サンプルアプリケーションのインストール後に、トラフィックをメッシュに送信します。以下のコマンドを数回入力します。

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

このコマンドはアプリケーションの **productpage** マイクロサービスにアクセスするユーザーをシミュレートします。

2. OpenShift Container Platform コンソールで、**Networking → Routes** に移動し、Jaeger ルートを検索します。これは **Location** に一覧される URL です。
   - または CLI を使用してルートの詳細のクエリーを実行します。この例では、**istio-system** が Service Mesh コントロールプレーンの namespace です。

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

     a. 以下のコマンドを実行して Jaeger コンソールの URL を表示します。結果をブラウザーに貼り付け、その URL に移動します。

     ```
     echo $JAEGER_URL
     ```

3. OpenShift Container Platform コンソールへアクセスするときに使用するものと同じユーザー名とパスワードを使用してログインします。
4. Jaeger ダッシュボードの左側のペインで、サービス メニューから **productpage.bookinfo** を選択し、ペイン下部の **Find Traces** をクリックします。トレースの一覧が表示されます。
5. 一覧のトレースのいずれかをクリックし、そのトレースの詳細ビューを開きます。一覧で最初の項目をクリックすると、**'/productpage** の最終更新に対応する詳細が表示されます。

2.9. データの可視化および可観測性

警告

こちらは、サポートされなくなった **Red Hat OpenShift Service Mesh** リリースのドキュメントです。

Service Mesh バージョン 1.0 および 1.1 コントロールプレーンはサポートされなくなりました。Service Mesh コントロールプレーンのアップグレードについては、Service Mesh の アップグレード を参照してください。

特定の Red Hat Service Mesh リリースのサポートステータスについては、製品ライフサイクルページ を参照してください。

Kiali コンソールでアプリケーションのトポロジー、健全性、およびメトリクスを表示できます。サービスに問題がある場合、Kiali コンソールは、サービス経由でデータフローを視覚化する方法を提供します。抽象アプリケーションからサービスおよびワークロードまで、さまざまなレベルでのメッシュコンポーネントに関する洞察を得ることができます。また Kiali は、リアルタイムで namespace のインタラクティブなグラフビューを提供します。

作業を開始する前に

アプリケーションがインストールされている場合には、アプリケーション経由でデータフローを確認できます。独自のアプリケーションがインストールされていない場合、Bookinfo サンプルアプリケーション をインストールして、Red Hat OpenShift Service Mesh での可観測性の機能を確認できます。

2.9.1. Service Mesh データの表示

Kiali Operator は、Red Hat OpenShift Service Mesh に収集される Telemetry データと連携して、namespace のアプリケーション、サービス、およびワークロードのグラフとリアルタイムのネットワーク図を提供します。

Kiali コンソールにアクセスするには、Red Hat OpenShift Service Mesh がインストールされており、サービスメッシュ用にプロジェクトを設定する必要があります。

手順

1. パースペクティブスイッチャーを使用して、**Administrator** パースペクティブに切り替えます。
2. **Home → Projects** をクリックします。
3. プロジェクトの名前をクリックします。たとえば、**bookinfo** をクリックします。
4. **Launcher** セクションで、**Kiali** をクリックします。
5. OpenShift Container Platform コンソールにアクセスするときに使用するものと同じユーザー名とパスワードを使用して Kiali コンソールにログインします。

初回の Kiali コンソールへのログイン時に、表示するパーミッションを持つサービスメッシュ内のすべての namespace を表示する **Overview** ページが表示されます。

コンソールのインストールを検証する場合は、表示するデータがないことがあります。

2.9.2. Kiali コンソールでのサービスメッシュデータの表示

Kiali グラフは、メッシュトラフィックの強力な視覚化を提供します。トポロジーは、リアルタイムの要求トラフィックを Istio 設定情報と組み合わせ、サービスメッシュの動作に関する洞察を即座に得て、問題を迅速に特定できるようにします。複数のグラフタイプを使用すると、トラフィックを高レベルのサービストポロジー、低レベルのワークロードトポロジー、またはアプリケーションレベルのトポロジーとして視覚化できます。

以下から選択できるグラフがいくつかあります。

- **App** グラフ は、同じラベルが付けられたすべてのアプリケーションの集約ワークロードを示します。
- **Service** グラフ は、メッシュ内の各サービスのノードを表示しますが、グラフからすべてのアプリケーションおよびワークロードを除外します。これは高レベルのビューを提供し、定義されたサービスのすべてのトラフィックを集約します。
- **Versioned App** グラフ は、アプリケーションの各バージョンのノードを表示します。アプリケーションの全バージョンがグループ化されます。
- **Workload** グラフ は、サービスメッシュの各ワークロードのノードを表示します。このグラフでは、app および version のラベルを使用する必要はありません。アプリケーションが version ラベルを使用しない場合は、このグラフを使用します。

グラフノードは、さまざまな情報で装飾され、仮想サービスやサービスエントリーなどのさまざまなルートルーティングオプションや、フォールトインジェクションやサーキットブレーカーなどの特別な設定を指定します。mTLS の問題、レイテンシーの問題、エラートラフィックなどを特定できます。グラフは高度な設定が可能で、トラフィックのアニメーションを表示でき、強力な検索機能や非表示機能があります。

Legend ボタンをクリックして、グラフに表示されるシェイプ、色、矢印、バッジに関する情報を表示します。

メトリクスの要約を表示するには、グラフ内のノードまたはエッジを選択し、そのメトリクスの詳細をサマリーの詳細パネルに表示します。

2.9.2.1. Kiali でのグラフレイアウトの変更

Kiali グラフのレイアウトは、アプリケーションのアーキテクチャーや表示データによって異なることがあります。たとえば、グラフノードの数およびそのインタラクションにより、Kiali グラフのレンダリング方法を判別できます。すべての状況に適した単一のレイアウトを作成することは不可能であるため、Kiali は複数の異なるレイアウトの選択肢を提供します。

前提条件

- 独自のアプリケーションがインストールされていない場合は、Bookinfo サンプルアプリケーションをインストールします。次に、以下のコマンドを複数回入力して Bookinfo アプリケーションのトラフィックを生成します。

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

  このコマンドはアプリケーションの **productpage** マイクロサービスにアクセスするユーザーをシミュレートします。

手順

1. Kiali コンソールを起動します。
2. **Log In With OpenShift** をクリックします。
3. Kiali コンソールで、**Graph** をクリックし、namespace グラフを表示します。
4. **Namespace** メニューから、アプリケーション namespace (例: **bookinfo**) を選択します。
5. 別のグラフレイアウトを選択するには、以下のいずれか、または両方を行います。
   - グラフの上部にあるメニューから、異なるグラフデータグループを選択します。
     - App graph
     - Service graph
     - Versioned App graph (デフォルト)
     - Workload graph
   - グラフの下部にある Legend から別のグラフレイアウトを選択します。
     - Layout default dagre
     - Layout 1 cose-bilkent

- Layout 2 cola

2.10. カスタムリソース

警告

こちらは、サポートされなくなった **Red Hat OpenShift Service Mesh** リリースのドキュメントです。

Service Mesh バージョン 1.0 および 1.1 コントロールプレーンはサポートされなくなりました。Service Mesh コントロールプレーンのアップグレードについては、Service Mesh の アップグレード を参照してください。

特定の Red Hat Service Mesh リリースのサポートステータスについては、製品ライフサイクルページ を参照してください。

デフォルトの Service Mesh のカスタムリソースを変更するか、または新規のカスタムリソースを作成して、Red Hat OpenShift Service Mesh をカスタマイズできます。

2.10.1. 前提条件

- **cluster-admin** ロールを持つアカウントがある。
- Red Hat OpenShift Service Mesh をインストールする準備 プロセスを完了していること。
- Operator がインストール済みである。

2.10.2. Red Hat OpenShift Service Mesh カスタムリソース

注記

istio-system プロジェクトは、Service Mesh のドキュメント全体でサンプルとして使用されますが、必要に応じて他のプロジェクトを使用できます。

カスタムリソース により、Red Hat OpenShift Service Mesh プロジェクトまたはクラスターで API を拡張することができます。Service Mesh をデプロイすると、プロジェクトパラメーターを変更するために変更できるデフォルトの **ServiceMeshControlPlane** が作成されます。

Service Mesh Operator は、**ServiceMeshControlPlane** リソースタイプを追加して API を拡張します。これにより、プロジェクト内に **ServiceMeshControlPlane** オブジェクトを作成できます。**ServiceMeshControlPlane** オブジェクトを作成することで、Operator にService Mesh コントロールプレーンをプロジェクトにインストールするよう指示でき、**ServiceMeshControlPlane** オブジェクトで設定したパラメーターを使用して設定できます。

この例の **ServiceMeshControlPlane** の定義には、サポートされるすべてのパラメーターが含まれ、これにより Red Hat Enterprise Linux (RHEL) をベースとした Red Hat OpenShift Service Mesh 1.1.18.2 イメージがデプロイされます。

重要

3scale の Istio Adapter は、カスタムリソースファイルでデプロイされ、設定されます。また、稼働している 3scale アカウント (SaaS または On-Premises) が必要になります。

istio-installation.yaml の詳細例

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

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

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

    mixer:
      policy:
        autoscaleEnabled: false

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

    pilot:
      autoscaleEnabled: false
      traceSampling: 100

    kiali:
      enabled: true

    grafana:
      enabled: true
```

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

2.10.3. ServiceMeshControlPlane パラメーター

以下の例は **ServiceMeshControlPlane** パラメーターの使用を示し、表はサポートされているパラメーターに関する追加情報を示しています。

重要

CPU、メモリー、Pod の数などのパラメーターを使用して Red Hat OpenShift Service Mesh に設定するリソースは、OpenShift Container Platform クラスターの設定をベースとしています。現在のクラスター設定で利用可能なリソースに基づいて、これらのパラメーターを設定します。

2.10.3.1. Istio グローバルの例

以下の例は、**ServiceMeshControlPlane** の Istio グローバルパラメーターと適切な値を持つ利用可能なパラメーターの説明を示しています。

注記

3scale Istio Adapter が機能するようするには、**disablePolicyChecks** は **false** である必要があります。

グローバルパラメーターの例

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

表2.4 グローバルパラメーター

パラメーター	説明	値	デフォルト値

パラメーター	説明	値	デフォルト値
disablePolicyChecks	このパラメーターは、ポリシーチェックを有効/無効にします。	**true/false**	**true**
policyCheckFailOpen	このパラメーターは、Mixer ポリシーサービスに到達できない場合にトラフィックを Envoy サイドカーコンテナーに通過させることができるかどうかを指定します。	**true/false**	**false**
tag	Operator が Istio イメージをプルするために使用するタグ。	有効なコンテナーイメージタグです。	**1.1.0**
hub	Operator が Istio イメージをプルするために使用するハブ。	有効なイメージリポジトリーです。	**maistra/** または **registry.redhat.io/openshift-service-mesh/**
mtls	このパラメーターは、デフォルトでサービス間での Mutual Transport Layer Security (mTLS) の有効化/無効化を制御します。	**true/false**	**false**
imagePullSecrets	Istio イメージを提供するレジストリーへのアクセスがセキュアな場合、ここに imagePullSecret を一覧表示します。	redhat-registry-pullsecret または quay-pullsecret	なし

これらのパラメーターは、グローバルパラメーターのプロキシーサブセットに固有のものです。

表2.5 プロキシーパラメーター

タイプ	パラメーター	説明	値	デフォルト値
requests	**cpu**	Envoy プロキシー用に要求される CPU リソースの量。	ご使用の環境設定に基づき、コアまたはミリコア (例: 200m、0.5、1) で指定される CPU リソース。	**10m**

タイプ	パラメーター	説明	値	デフォルト値
	memory	Envoy プロキシー用に要求されるメモリー量。	ご使用の環境設定に基づく、利用可能なバイト単位のメモリー (例: 200Ki、50Mi、5Gi)。	**128Mi**
limits	**cpu**	Envoy プロキシー用に要求される CPU リソースの最大量。	ご使用の環境設定に基づき、コアまたはミリコア (例: 200m、0.5、1) で指定される CPU リソース。	**2000m**
	memory	使用が許可されているメモリー Envoy プロキシーの最大量。	ご使用の環境設定に基づく、利用可能なバイト単位のメモリー (例: 200Ki、50Mi、5Gi)。	**1024Mi**

2.10.3.2. Istio ゲートウェイの設定

以下の例は、**ServiceMeshControlPlane** の Istio ゲートウェイパラメーターと適切な値を持つ利用可能なパラメーターの説明を示しています。

ゲートウェイパラメーターの例

```
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
```

表2.6 Istio ゲートウェイパラメーター

パラメーター	説明	値	デフォルト値
gateways.egress.runtime.deployment.autoScaling.enabled	このパラメーターは、自動スケーリングを有効/無効にします。	**true/false**	**true**
gateways.egress.runtime.deployment.autoScaling.minReplicas	**autoscaleEnabled** 設定に基づいて Egress ゲートウェイにデプロイする Pod の最小数。	ご使用の環境設定に基づく、有効な割り当て可能な Pod 数。	**1**
gateways.egress.runtime.deployment.autoScaling.maxReplicas	**autoscaleEnabled** 設定に基づいて Egress ゲートウェイにデプロイする Pod の最大数。	ご使用の環境設定に基づく、有効な割り当て可能な Pod 数。	**5**
gateways.ingress.runtime.deployment.autoScaling.enabled	このパラメーターは、自動スケーリングを有効/無効にします。	**true/false**	**true**
gateways.ingress.runtime.deployment.autoScaling.minReplicas	**autoscaleEnabled** 設定に基づいて Ingress ゲートウェイにデプロイする Pod の最小数。	ご使用の環境設定に基づく、有効な割り当て可能な Pod 数。	**1**
gateways.ingress.runtime.deployment.autoScaling.maxReplicas	**autoscaleEnabled** 設定に基づいて Ingress ゲートウェイにデプロイする Pod の最大数。	ご使用の環境設定に基づく、有効な割り当て可能な Pod 数。	**5**

クラスター管理者は、サブドメインを有効にする方法について、ワイルドカードルートの使用 を参照できます。

2.10.3.3. Istio Mixer 設定

以下の例は、**ServiceMeshControlPlane** の Mixer パラメーターと適切な値を持つ利用可能なパラメーターの説明を示しています。

Mixer パラメーターの例

```
mixer:
  enabled: true
  policy:
    autoscaleEnabled: false
  telemetry:
    autoscaleEnabled: false
    resources:
    requests:
```

```
        cpu: 10m
        memory: 128Mi
        limits:
```

表2.7 Istio Mixer ポリシーパラメーター

パラメーター	説明	値	デフォルト値
enabled	このパラメーターは、Mixer を有効/無効にします。	**true/false**	**true**
autoscaleEnabled	このパラメーターは、自動スケーリングを有効/無効にします。小規模な環境では、このパラメーターを無効にします。	**true/false**	**true**
autoscaleMin	**autoscaleEnabled** 設定に基づいてデプロイする Pod の最小数。	ご使用の環境設定に基づく、有効な割り当て可能な Pod 数。	**1**
autoscaleMax	**autoscaleEnabled** 設定に基づいてデプロイする Pod の最大数。	ご使用の環境設定に基づく、有効な割り当て可能な Pod 数。	**5**

表2.8 Istio Mixer Telemetry パラメーター

タイプ	パラメーター	説明	値	デフォルト
requests	**cpu**	Mixer Telemetry に要求される CPU リソースのパーセンテージ。	ご使用の環境設定に基づく、ミリコア単位の CPU リソース。	**10m**
	memory	Mixer Telemetry に要求されるメモリー量。	ご使用の環境設定に基づく、利用可能なバイト単位のメモリー (例: 200Ki、50Mi、5Gi)。	**128Mi**
limits	**cpu**	使用を許可された CPU リソース Mixer Telemetry の最大パーセンテージ。	ご使用の環境設定に基づく、ミリコア単位の CPU リソース。	**4800m**

タイプ	パラメーター	説明	値	デフォルト
	memory	使用を許可されているメモリー Mixer Telemetry の最大量です。	ご使用の環境設定に基づく、利用可能なバイト単位のメモリー (例: 200Ki、50Mi、5Gi)。	**4G**

2.10.3.4. Istio Pilot 設定

Pilot を、リソース割り当てのスケジュールまたはその制限を設定するように設定できます。以下の例は、**ServiceMeshControlPlane** の Pilot パラメーターと適切な値を持つ利用可能なパラメーターの説明を示しています。

Pilot パラメーターの例

```
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
```

表2.9 Istio Pilot パラメーター

パラメーター	説明	値	デフォルト値

パラメーター	説明	値	デフォルト値
cpu	Pilot に要求される CPU リソースのパーセンテージ。	ご使用の環境設定に基づく、ミリコア単位の CPU リソース。	**10m**
memory	Pilot に要求されるメモリー量。	ご使用の環境設定に基づく、利用可能なバイト単位のメモリー (例: 200Ki、50Mi、5Gi)。	**128Mi**
autoscaleEnabled	このパラメーターは、自動スケーリングを有効/無効にします。小規模な環境では、このパラメーターを無効にします。	**true/false**	**true**
traceSampling	この値は、無作為のサンプリングの発生頻度を制御します。**注:** 開発またはテストの場合はこの値を増やします。	有効なパーセンテージ。	**1.0**

2.10.4. Kiali の設定

Service Mesh Operator は **ServiceMeshControlPlane** を作成する際に、Kiali リソースも処理します。次に Kiali Operator は Kiali インスタンスの作成時にこのオブジェクトを使用します。

ServiceMeshControlPlane で指定されるデフォルトの Kiali パラメーターは以下のとおりです。

Kiali パラメーターの例

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

表2.10 Kiali パラメーター

パラメーター	説明	値	デフォルト値
enabled	このパラメーターは、Kiali を有効/無効にします。Kiali はデフォルトで有効です。	**true/false**	**true**

パラメーター	説明	値	デフォルト値
dashboard viewOnlyMode	このパラメーターは、Kiali コンソールの表示専用 (view-only) モードを有効/無効にします。表示専用モードを有効にすると、ユーザーはコンソールを使用して Service Mesh を変更できなくなります。	**true/false**	**false**
ingress enabled	このパラメーターは、Kiali の Ingress を有効/無効にします。	**true/false**	**true**

2.10.4.1. Grafana の Kiali の設定

Kiali および Grafana を Red Hat OpenShift Service Mesh の一部としてインストールする場合、Operator はデフォルトで以下を設定します。

- Grafana を Kiali の外部サービスとして有効化
- Kiali コンソールの Grafana 認証
- Kiali コンソールの Grafana URL

Kiali は Grafana URL を自動的に検出できます。ただし、Kiali で簡単に自動検出できないカスタムの Grafana インストールがある場合、**ServiceMeshControlPlane** リソースの URL の値を更新する必要があります。

追加の Grafana パラメーター

```
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. Jaeger についての Kiali の設定

Kiali および Jaeger を Red Hat OpenShift Service Mesh の一部としてインストールする場合、Operator はデフォルトで以下を設定します。

- Jaeger を Kiali の外部サービスとして有効化
- Kiali コンソールの Jaeger 認証
- Kiali コンソールの Jaeger URL

Kiali は Jaeger URL を自動的に検出できます。ただし、Kiali で簡単に自動検出できないカスタムの Jaeger インストールがある場合、**ServiceMeshControlPlane** リソースの URL の値を更新する必要があります。

追加の Jaeger パラメーター

```
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. Jaeger の設定

Service Mesh Operator は **ServiceMeshControlPlane** リソースを作成する際に、分散トレースのリソースも作成することができます。Service Mesh は分散トレースに Jaeger を使用します。

Jaeger 設定は、以下の 2 つの方法のいずれかで指定できます。

- **ServiceMeshControlPlane** リソースで Jaeger を設定します。この方法にはいくつかの制限があります。
- Jaeger をカスタム **Jaeger** リソースに設定し、**ServiceMeshControlPlane** リソースでその Jaeger インスタンスを参照します。**name** の値に一致する Jaeger リソースが存在する場合、コントロールプレーンは既存のインストールを使用します。この方法では、Jaeger 設定を完全にカスタマイズできます。

ServiceMeshControlPlane で指定されるデフォルトの Jaeger パラメーターは以下のとおりです。

デフォルトの all-in-one Jaeger パラメーター

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  version: v1.1
  istio:
    tracing:
      enabled: true
      jaeger:
        template: all-in-one
```

表2.11 Jaeger パラメーター

パラメーター	説明	値	デフォルト値

パラメーター	説明	値	デフォルト値
tracing: enabled:	このパラメーターは、Service Mesh Operator によるトレースのインストールおよびデプロイを有効/無効にします。Jaeger のインストールはデフォルトで有効にされます。既存の Jaeger デプロイメントを使用するには、この値を **false** に設定します。	**true**/**false**	**true**
jaeger: template:	このパラメーターは、使用する Jaeger デプロイメントストラテジーを指定します。	• **all-in-one**: 開発、テスト、デモおよび概念実証用。 • **production-elasticsearch**: 実稼働環境での使用。	**all-in-one**

注記

ServiceMeshControlPlane リソースのデフォルトのテンプレートは、インメモリーストレージを使用する **all-in-one** のデプロイメントストラテジーです。実稼働環境では、サポートされている唯一のストレージオプションが Elasticsearch であるため、実稼働環境内に Service Mesh をデプロイする際には、**production-elasticsearch** テンプレートを要求するように **ServiceMeshControlPlane** を設定する必要があります。

2.10.5.1. Elasticsearch の設定

デフォルトの Jaeger デプロイメントストラテジーでは、**all-in-one** テンプレートを使用するため、最小のリソースでインストールを完了できます。ただし、**all-in-one** テンプレートはインメモリーストレージを使用するので、開発、デモまたはテスト目的での使用を推奨しています。実稼働環境には使用しないでください。

実稼働環境で Service Mesh および Jaeger をデプロイする場合、テンプレートを **production-elasticsearch** テンプレートに変更する必要があります。これは Jaeger のストレージのニーズに対応するために Elasticsearch を使用します。

Elasticsearch はメモリー集約型アプリケーションです。デフォルトの OpenShift Container Platform インストールで指定されたノードの初期セットは、Elasticsearch クラスターをサポートするのに十分な大きさではない場合があります。デフォルトの Elasticsearch 設定は、ユースケースと OpenShift Container Platform インストール用に必要とするリソースに一致するように変更する必要があります。resources ブロックを有効な CPU 値およびメモリー値で変更することにより、各コンポーネントの CPU およびメモリーの制限の両方を調整することができます。推奨容量 (以上) のメモリーを使用して実行する場合は、追加のノードをクラスターに追加する必要があります。OpenShift Container Platform インストールに必要となるリソースを超えていないことを確認してください。

Elasticsearch を使用したデフォルトの実稼働 Jaeger パラメーター

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

表2.12 Elasticsearch パラメーター

パラメーター	説明	値	デフォルト値	例
`tracing: enabled:`	このパラメーターは、Service Mesh でトレースを有効/無効にします。Jaeger がデフォルトでインストールされます。	**true/false**	**true**	
`ingress: enabled:`	このパラメーターは、Jaeger の Ingress を有効/無効にします。	**true/false**	**true**	
`jaeger: template:`	このパラメーターは、使用する Jaeger デプロイメントストラテジーを指定します。	**all-in-one/production-elasticsearch**	**all-in-one**	
`elasticsearch: nodeCount:`	作成する Elasticsearch ノードの数。	整数値。	1	概念実証 = 1、最小デプロイメント = 3
`requests: cpu:`	ご使用の環境設定に基づく、要求に対する中央処理単位の数。	コアまたはミリコアで指定されます (例: 200m、0.5、1)。	1Gi	概念実証 = 500m、最小デプロイメント = 1

パラメーター	説明	値	デフォルト値	例
requests: memory:	ご使用の環境設定に基づく、要求に使用できるメモリー。	バイト単位で指定されます (例: 200Ki、50Mi、5Gi)。	500m	概念実証 = 1Gi、最小デプロイメント = 16Gi*
limits: cpu:	ご使用の環境設定に基づく、中央処理単位数の制限。	コアまたはミリコアで指定されます (例: 200m、0.5、1)。		概念実証 = 500m、最小デプロイメント = 1
limits: memory:	ご使用の環境設定に基づく、利用可能なメモリー制限。	バイト単位で指定されます (例: 200Ki、50Mi、5Gi)。		概念実証 = 1Gi、最小デプロイメント = 16Gi*
	* 各 Elasticsearch ノードはこれより低い値のメモリー設定でも動作しますが、これは実稼働環境でのデプロイメントには推奨され**ません**。実稼働環境で使用する場合、デフォルトで各 Pod に割り当てる設定を 16Gi 未満にすることはできず、Pod ごとに最大 64Gi を割り当てることを推奨します。			

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。
2. **Operators → Installed Operators** に移動します。
3. Red Hat OpenShift Service Mesh Operator をクリックします。
4. **Istio Service Mesh Control Plane**タブをクリックします。
5. コントロールプレーンのファイル名 (**basic-install** など) をクリックします。
6. **YAML** タブをクリックします。
7. Jaeger パラメーターを編集し、デフォルトの **all-in-one** テンプレートを **production-elasticsearch** テンプレートのパラメーターに置き換え、ユースケースに合わせて変更します。インデントが正しいことを確認します。
8. **Save** をクリックします。
9. **Reload** をクリックします。OpenShift Container Platform は Jaeger を再デプロイし、指定されたパラメーターに基づいて Elasticsearch リソースを作成します。

2.10.5.2. 既存の Jaeger インスタンスへの接続

SMCP が既存の Jaeger インスタンスに接続できるようにするには、以下が true である必要があります。

- Jaeger インスタンスは、コントロールプレーンと同じ namespace にデプロイされます (例: **istio-system** namespace)。

- サービス間でのセキュアな通信を有効にするには、Jaeger インスタンスとの通信のセキュリティーを保護する oauth-proxy を有効にし、シークレットが Jaeger インスタンスにマウントされ、Kiali がこれと通信できるようにする必要があります。

- カスタムまたは既存の Jaeger インスタンスを使用するには、**spec.istio.tracing.enabled** を false に設定し、Jaeger インスタンスのデプロイメントを無効にします。

- **spec.istio.global.tracer.zipkin.address** を **jaeger-collector** を jaeger-collector サービスのホスト名およびポートに設定して、正しい jaeger-collector エンドポイントを Mixer に指定します。通常、サービスのホスト名は **<jaeger-instance-name>-collector.<namespace>.svc.cluster.local** です。

- **spec.istio.kiali.jaegerInClusterURL** を jaeger-query サービスのホスト名に設定し、トレースを収集するために正しい jaeger-query エンドポイントを Kiali に指定します。ポートは、デフォルトで 443 を使用するため、通常は不要です。通常、サービスのホスト名は **<jaeger-instance-name>-query.<namespace>.svc.cluster.local** です。

- Jaeger インスタンスのダッシュボード URL を Kiali に指定し、Kiali コンソールから Jaeger にアクセスできるようにします。Jaeger Operator によって作成される OpenShift ルートから URL を取得できます。Jaeger リソースが **external-jaeger** で、**istio-system** プロジェクトにある場合は、以下のコマンドを使用してルートを取得できます。

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

出力例

```
NAME                HOST/PORT                                  PATH   SERVICES              [...]
external-jaeger     external-jaeger-istio-system.apps.test            external-jaeger-query  [...]
```

HOST/PORT の値は、Jaeger ダッシュボードの外部アクセス可能な URL です。

例: Jaeger リソース

```
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
```

以下の **ServiceMeshControlPlane** の例では、Jaeger Operator および Jaeger リソースのサンプルを使用して Jaeger をデプロイしていることを前提としています。

外部 Jaeger を使用した ServiceMeshControlPlane の例

```
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. Elasticsearch の設定

デフォルトの Jaeger デプロイメントストラテジーでは、**all-in-one** テンプレートを使用するため、最小のリソースでインストールを完了できます。ただし、**all-in-one** テンプレートはインメモリーストレージを使用するので、開発、デモまたはテスト目的での使用を推奨しています。実稼働環境には使用しないでください。

実稼働環境で Service Mesh および Jaeger をデプロイする場合、テンプレートを **production-elasticsearch** テンプレートに変更する必要があります。これは Jaeger のストレージのニーズに対応するために Elasticsearch を使用します。

Elasticsearch はメモリー集約型アプリケーションです。デフォルトの OpenShift Container Platform インストールで指定されたノードの初期セットは、Elasticsearch クラスターをサポートするのに十分な大きさではない場合があります。デフォルトの Elasticsearch 設定は、ユースケースと OpenShift Container Platform インストール用に必要とするリソースに一致するように変更する必要があります。resources ブロックを有効な CPU 値およびメモリー値で変更することにより、各コンポーネントの CPU およびメモリーの制限の両方を調整することができます。推奨容量 (以上) のメモリーを使用して実行する場合は、追加のノードをクラスターに追加する必要があります。OpenShift Container Platform インストールに必要となるリソースを超えていないことを確認してください。

Elasticsearch を使用したデフォルトの実稼働 Jaeger パラメーター

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

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

表2.13 Elasticsearch パラメーター

パラメーター	説明	値	デフォルト値	例
`tracing: enabled:`	このパラメーターは、Service Mesh でトレースを有効/無効にします。Jaeger がデフォルトでインストールされます。	**true/false**	**true**	
`ingress: enabled:`	このパラメーターは、Jaeger の Ingress を有効/無効にします。	**true/false**	**true**	
`jaeger: template:`	このパラメーターは、使用する Jaeger デプロイメントストラテジーを指定します。	**all-in-one/production-elasticsearch**	**all-in-one**	
`elasticsearch: nodeCount:`	作成する Elasticsearch ノードの数。	整数値。	1	概念実証 = 1、最小デプロイメント = 3
`requests: cpu:`	ご使用の環境設定に基づく、要求に対する中央処理単位の数。	コアまたはミリコアで指定されます (例: 200m、0.5、1)。	1Gi	概念実証 = 500m、最小デプロイメント = 1

パラメーター	説明	値	デフォルト値	例
requests: memory:	ご使用の環境設定に基づく、要求に使用できるメモリー。	バイト単位で指定されます (例: 200Ki、50Mi、5Gi)。	500m	概念実証 = 1Gi、最小デプロイメント = 16Gi*
limits: cpu:	ご使用の環境設定に基づく、中央処理単位数の制限。	コアまたはミリコアで指定されます (例: 200m、0.5、1)。		概念実証 = 500m、最小デプロイメント = 1
limits: memory:	ご使用の環境設定に基づく、利用可能なメモリー制限。	バイト単位で指定されます (例: 200Ki、50Mi、5Gi)。		概念実証 = 1Gi、最小デプロイメント = 16Gi*
	* 各 Elasticsearch ノードはこれより低い値のメモリー設定でも動作しますが、これは実稼働環境でのデプロイメントには推奨され**ません**。実稼働環境で使用する場合、デフォルトで各 Pod に割り当てる設定を 16Gi 未満にすることはできず、Pod ごとに最大 64Gi を割り当てることを推奨します。			

手順

1. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform Web コンソールにログインします。
2. **Operators → Installed Operators** に移動します。
3. Red Hat OpenShift Service Mesh Operator をクリックします。
4. **Istio Service Mesh Control Plane**タブをクリックします。
5. コントロールプレーンのファイル名 (**basic-install** など) をクリックします。
6. **YAML** タブをクリックします。
7. Jaeger パラメーターを編集し、デフォルトの **all-in-one** テンプレートを **production-elasticsearch** テンプレートのパラメーターに置き換え、ユースケースに合わせて変更します。インデントが正しいことを確認します。
8. **Save** をクリックします。
9. **Reload** をクリックします。OpenShift Container Platform は Jaeger を再デプロイし、指定されたパラメーターに基づいて Elasticsearch リソースを作成します。

2.10.5.4. Elasticsearch インデックスクリーナージョブの設定

Service Mesh Operator は **ServiceMeshControlPlane** を作成した際に Jaeger のカスタムリソース (CR) も作成します。次に、Red Hat OpenShift 分散トレースプラットフォーム Operator は Jaeger インスタンスの作成時にこの CR を使用します。

Elasticsearch ストレージを使用する場合、デフォルトでジョブが作成され、古いトレースをストレージからクリーンアップします。このジョブのオプションを設定するには、Jaeger カスタムリソース (CR) を編集して、ユースケースに合わせてカスタマイズします。関連するオプションを以下に示します。

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

表2.14 Elasticsearch インデックスクリーナーのパラメーター

パラメーター	値	説明
enabled	true/ false	インデックスクリーナージョブを有効または無効にします。
numberOfDays	整数値	インデックスの削除を待機する日数。
schedule	"55 23 * * *"	実行するジョブの cron 式

Elasticsearch を OpenShift Container Platform で設定する方法については、ログストアの設定 について参照してください。

2.10.6. 3scale の設定

以下の表では、**ServiceMeshControlPlane** リソースの 3scale Istio アダプターのパラメーターについて説明しています。

3scale パラメーターの例

```
spec:
  addons:
    3Scale:
      enabled: false
      PARAM_THREESCALE_LISTEN_ADDR: 3333
      PARAM_THREESCALE_LOG_LEVEL: info
      PARAM_THREESCALE_LOG_JSON: true
      PARAM_THREESCALE_LOG_GRPC: false
      PARAM_THREESCALE_REPORT_METRICS: true
      PARAM_THREESCALE_METRICS_PORT: 8080
      PARAM_THREESCALE_CACHE_TTL_SECONDS: 300
      PARAM_THREESCALE_CACHE_REFRESH_SECONDS: 180
      PARAM_THREESCALE_CACHE_ENTRIES_MAX: 1000
      PARAM_THREESCALE_CACHE_REFRESH_RETRIES: 1
      PARAM_THREESCALE_ALLOW_INSECURE_CONN: false
      PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS: 10
```

```
PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS: 60
PARAM_USE_CACHED_BACKEND: false
PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS: 15
PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED: true
```

表2.15 3scale パラメーター

パラメーター	説明	値	デフォルト値
enabled	3scale アダプターを使用するかどうか	**true/false**	**false**
PARAM_THREESCALE_LISTEN_ADDR	gRPC サーバーのリッスンアドレスを設定します。	有効なポート番号	**3333**
PARAM_THREESCALE_LOG_LEVEL	ログ出力の最小レベルを設定します。	**debug**、**info**、**warn**、**error**、または **none**	**info**
PARAM_THREESCALE_LOG_JSON	ログが JSON としてフォーマットされるかどうかを制御します。	**true/false**	**true**
PARAM_THREESCALE_LOG_GRPC	ログに gRPC 情報を含むかどうかを制御します。	**true/false**	**true**
PARAM_THREESCALE_REPORT_METRICS	3scale システムおよびバックエンドメトリクスが収集され、Prometheus に報告されるかどうかを制御します。	**true/false**	**true**
PARAM_THREESCALE_METRICS_PORT	3scale /**metrics** エンドポイントをスクラップできるポートを設定します。	有効なポート番号	**8080**
PARAM_THREESCALE_CACHE_TTL_SECONDS	キャッシュから期限切れのアイテムを消去するまで待機する時間 (秒単位)。	時間 (秒単位)	**300**
PARAM_THREESCALE_CACHE_REFRESH_SECONDS	キャッシュ要素の更新を試行する場合の期限	時間 (秒単位)	**180**
PARAM_THREESCALE_CACHE_ENTRIES_MAX	キャッシュにいつでも保存できるアイテムの最大数。キャッシュを無効にするには **0** に設定します。	有効な数字	**1000**

パラメーター	説明	値	デフォルト値
PARAM_THREESCALE_CACHE_REFRESH_RETRIES	キャッシュ更新ループ時に到達できないホストが再試行される回数	有効な数字	**1**
PARAM_THREESCALE_ALLOW_INSECURE_CONN	**3scale** API 呼び出し時の証明書の検証を省略できるようにします。この有効化は推奨されていません。	**true/false**	**false**
PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS	3scale システムおよびバックエンドへの要求を終了するまで待機する秒数を設定します。	時間 (秒単位)	**10**
PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS	接続を閉じるまでの最大秒数 (+/-10% のジッター) を設定します。	時間 (秒単位)	60
PARAM_USE_CACHE_BACKEND	true の場合、承認要求のインメモリー apisonator キャッシュの作成を試行します。	**true/false**	**false**
PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS	バックエンドキャッシュが有効な場合には、3scale に対してキャッシュをフラッシュする間隔を秒単位で設定します。	時間 (秒単位)	15
PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED	バックエンドキャッシュが承認データを取得できない場合は常に、要求を拒否する (クローズする) か、許可する (オープンする) かどうか。	**true/false**	**true**

2.11. 3SCALE ISTIO アダプターの使用

警告

こちらは、サポートされなくなった **Red Hat OpenShift Service Mesh** リリースのドキュメントです。

Service Mesh バージョン 1.0 および 1.1 コントロールプレーンはサポートされなくなりました。Service Mesh コントロールプレーンのアップグレードについては、Service Mesh の アップグレード を参照してください。

特定の Red Hat Service Mesh リリースのサポートステータスについては、製品ライフサイクルページ を参照してください。

3scale Istio アダプターはオプションのアダプターであり、これを使用すると、Red Hat OpenShift Service Mesh 内で実行中のサービスにラベルを付け、そのサービスを 3scale API Management ソリューションと統合できます。これは Red Hat OpenShift Service Mesh には必要ありません。

2.11.1. 3scale アダプターと Red Hat OpenShift Service Mesh の統合

これらの例を使用して、3scale Istio アダプターを使用してサービスに対する要求を設定できます。

前提条件:

- Red Hat OpenShift Service Mesh バージョン 1.x
- 稼働している 3scale アカウント (SaaS または 3scale 2.5 On-Premises)
- バックエンドキャッシュを有効にするには 3scale 2.9 以上が必要です。
- Red Hat OpenShift Service Mesh の前提条件

注記

3scale Istio アダプターを設定するために、アダプターパラメーターをカスタムリソースファイルに追加する手順については、Red Hat OpenShift Service Mesh カスタムリソースを参照してください。

注記

kind: handler リソースにとくに注意してください。これを 3scale アカウントの認証情報を使用して更新する必要があります。オプションで **service_id** をハンドラーに追加できますが、この設定は 3scale アカウントの 1 つのサービスに対してのみ有効で、後方互換性を確保するためにだけ維持されています。**service_id** をハンドラーに追加する場合は、他のサービスに対して 3scale を有効にする必要がある場合は、別の **service_ids** で追加のハンドラーを作成する必要があります。

以下の手順に従って、3scale アカウントごとに単一のハンドラーを使用します。

手順

1. 3scale アカウントのハンドラーを作成し、アカウントの認証情報を指定します。サービス識別子を省略します。

```
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"
```

オプションで、**params** セクション内の **backend_url** フィールドを指定して、3scale 設定によって提供される URL を上書きできます。これは、アダプターが 3scale のオンプレミスインスタンスと同じクラスターで実行され、内部クラスター DNS を利用する必要がある場合に役立ちます。

2. 3scale アカウントに属するサービスの Deployment リソースを編集するか、またはパッチを適用します。

   a. 有効な **service_id** に対応する値を使用して **"service-mesh.3scale.net/service-id"** ラベルを追加します。

   b. 手順 1 の ハンドラーリソースの名前 の値を使用して **"service-mesh.3scale.net/credentials"** ラベルを追加します。

3. 他のサービスを追加する場合には、手順 2 を実行して、3scale アカウントの認証情報とそのサービス識別子にリンクします。

4. 3scale 設定でルールの設定を変更し、ルールを 3scale ハンドラーにディスパッチします。

ルール設定の例

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

2.11.1.1. 3scale カスタムリソースの生成

アダプターには、**handler**、**instance**、および **rule** カスタムリソースの生成を可能にするツールが含まれます。

表2.16 使用法

オプション	説明	必須	デフォルト値
-h, --help	利用可能なオプションについてのヘルプ出力を生成します	いいえ	
--name	この URL の一意の名前、トークンのペア	はい	
-n, --namespace	テンプレートを生成する namespace	いいえ	istio-system
-t, --token	3scale アクセストークン	はい	
-u, --url	3scale 管理ポータル URL	はい	
--backend-url	3scale バックエンド URL。これが設定されている場合、システム設定から読み込まれる値がオーバーライドされます。	いいえ	
-s, --service	3scale API/サービス ID	いいえ	
--auth	指定する 3scale 認証パターン (1=API Key, 2=App Id/App Key, 3=OIDC)	いいえ	ハイブリッド
-o, --output	生成されたマニフェストを保存するファイル	いいえ	標準出力
--version	CLI バージョンを出力し、即座に終了する	いいえ	

2.11.1.1.1. URL サンプルからのテンプレートの生成

注記

- デプロイされたアダプターからのマニフェストの生成 で、3scale アダプターコンテナーイメージからの **oc exec** を使用して以下のコマンドを実行します。
- **3scale-config-gen** コマンドを使用すると、YAML 構文とインデントエラーを回避するのに役立ちます。
- このアノテーションを使用する場合は **--service** を省略できます。
- このコマンドは、**oc exec** を使用してコンテナーイメージ内から起動する必要があります。

手順

- **3scale-config-gen** コマンドを使用して、トークンと URL のペアを 1 つのハンドラーとして複数のサービスで共有できるようにテンプレートを自動生成します。

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

- 以下の例では、ハンドラーに埋め込まれたサービス ID を使用してテンプレートを生成します。

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

関連情報

- トークン

2.11.1.2. デプロイされたアダプターからのマニフェストの生成

注記

- **NAME** は、3scale で管理するサービスの識別に使用する識別子です。
- **CREDENTIALS_NAME** 参照は、ルール設定の **match** セクションに対応する識別子です。CLI ツールを使用している場合は、**NAME** 識別子に自動設定されます。
- この値は具体的なものでなくても構いませんが、ラベル値はルールの内容と一致させる必要があります。詳細は、アダプター経由でのサービストラフィックのルーティング を参照してください。

1. このコマンドを実行して、**istio-system** namespace でデプロイされたアダプターからマニフェストを生成します。

```
$ 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. これでターミナルにサンプル出力が生成されます。必要に応じて、これらのサンプルを編集し、**oc create** コマンドを使用してオブジェクトを作成します。
3. 要求がアダプターに到達すると、アダプターはサービスが 3scale の API にどのようにマッピングされるかを認識している必要があります。この情報は、以下のいずれかの方法で提供できます。
   a. ワークロードにラベルを付ける (推奨)
   b. ハンドラーを **service_id** としてハードコーディングする
4. 必要なアノテーションでワークロードを更新します。

注記

ハンドラーにまだ組み込まれていない場合は、このサンプルで提供されたサービス ID のみを更新する必要があります。ハンドラーの設定が優先されます。

```
$ 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. アダプター経由でのサービストラフィックのルーティング

以下の手順に従って、3scale アダプターを使用してサービスのトラフィックを処理します。

前提条件

- 3scale 管理者から受け取る認証情報とサービス ID

手順

1. **kind: rule** リソース内で、以前に設定で作成した **destination.labels["service-mesh.3scale.net/credentials"] == "threescale"** ルールと一致させます。
2. 上記のラベルを、ターゲットワークロードのデプロイメントで **PodTemplateSpec** に追加し、サービスを統合します。値 **threescale** は生成されたハンドラーの名前を参照します。このハンドラーは、3scale を呼び出すのに必要なアクセストークンを保存します。
3. **destination.labels["service-mesh.3scale.net/service-id"] == "replace-me"** ラベルをワークロードに追加し、要求時にサービス ID をインスタンス経由でアダプターに渡します。

2.11.2. 3scale での統合設定

以下の手順に従って、3scale の統合設定を行います。

注記

3scale SaaS を使用している場合、Red Hat OpenShift Service Mesh は Early Access プログラムの一部として有効にされています。

手順

1. **[your_API_name] → Integration** の順に移動します。
2. **Settings** をクリックします。
3. **Deployment** で **Istio** オプションを選択します。
   - デフォルトでは **Authentication** の **API Key (user_key)** オプションが選択されます。

4. **Update Product** をクリックして選択内容を保存します。

5. **Configuration** をクリックします。

6. 設定の更新 をクリックします。

2.11.3. キャッシング動作

3scale System API からの応答は、アダプター内でデフォルトでキャッシュされます。**cacheTTLSeconds** 値よりも古いと、エントリーはキャッシュから消去されます。また、デフォルトでキャッシュされたエントリーの自動更新は、**cacheRefreshSeconds** 値に基づいて、期限が切れる前に数秒間試行されます。**cacheTTLSeconds** 値よりも高い値を設定することで、自動更新を無効にできます。

cacheEntriesMax を正の値以外に設定すると、キャッシングを完全に無効にできます。

更新プロセスを使用すると、到達不能になるホストのキャッシュされた値が、期限が切れて最終的に消去される前に再試行されます。

2.11.4. 認証要求

本リリースでは、以下の認証方法をサポートします。

- 標準 **API** キー: 単一のランダム文字列またはハッシュが識別子およびシークレットトークンとして機能します。
- アプリケーション **ID** とキーのペア: イミュータブルな識別子とミュータブルなシークレットキー文字列。
- **OpenID** 認証方法: JSON Web トークンから解析されるクライアント ID 文字列。

2.11.4.1. 認証パターンの適用

以下の認証方法の例に従って **instance** カスタムリソースを変更し、認証動作を設定します。認証情報は、以下から受け取ることができます。

- 要求ヘッダー
- 要求パラメーター
- 要求ヘッダーとクエリーパラメーターの両方

注記

ヘッダーの値を指定する場合、この値は小文字である必要があります。たとえば、ヘッダーを **User-Key** として送信する必要がある場合、これは設定で **request.headers["user-key"]** として参照される必要があります。

2.11.4.1.1. API キー認証方法

Service Mesh は、**subject** カスタムリソースパラメーターの **user** オプションで指定されたクエリーパラメーターと要求ヘッダーで API キーを検索します。これは、カスタムリソースファイルで指定される順序で値をチェックします。不要なオプションを省略することで、API キーの検索をクエリーパラメーターまたは要求ヘッダーに制限できます。

この例では、Service Mesh は **user_key** クエリーパラメーターの API キーを検索します。API キーがクエリーパラメーターにない場合、Service Mesh は **x-user-key** ヘッダーを確認します。

API キー認証方法の例

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

アダプターが異なるクエリーパラメーターまたは要求ヘッダーを検査するようにする場合は、名前を適宜変更します。たとえば、key というクエリーパラメーターの API キーを確認するには、**request.query_params["user_key"]** を **request.query_params["key"]** に変更します。

2.11.4.1.2. アプリケーション ID およびアプリケーションキーペアの認証方法

Service Mesh は、**subject** カスタムリソースパラメーターの **properties** オプションで指定されるように、クエリーパラメーターと要求ヘッダーでアプリケーション ID とアプリケーションキーを検索します。アプリケーションキーはオプションです。これは、カスタムリソースファイルで指定される順序で値をチェックします。不要なオプションを含めないことで、認証情報の検索をクエリーパラメーターまたは要求ヘッダーのいずれかに制限できます。

この例では、Service Mesh は最初にクエリーパラメーターのアプリケーション ID とアプリケーションキーを検索し、必要に応じて要求ヘッダーに移動します。

アプリケーション ID およびアプリケーションキーペアの認証方法の例

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

アダプターが異なるクエリーパラメーターまたは要求ヘッダーを検査するようにする場合は、名前を適宜変更します。たとえば、**identification** という名前のクエリーパラメーターのアプリケーション ID を確認するには、**request.query_params["app_id"]** を **request.query_params["identification"]** に変更します。

2.11.4.1.3. OpenID 認証方法

OpenID Connect (OIDC) 認証方法 を使用するには、**subject** フィールドで **properties** 値を使用して **client_id** および任意で **app_key** を設定します。

このオブジェクトは、前述の方法を使用して操作することができます。以下の設定例では、クライアント識別子 (アプリケーション ID) は、**azp** ラベルの JSON Web Token (JWT) から解析されます。これは必要に応じて変更できます。

OpenID 認証方法の例

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

この統合を正常に機能させるには、クライアントがアイデンティティープロバイダー (IdP) で作成されるよう OIDC を 3scale で実行する必要があります。保護するサービスと同じ namespace でサービスの に要求の認証 を作成する必要があります。JWT は要求の **Authorization** ヘッダーに渡されます。

以下に定義されるサンプル **RequestAuthentication** で、**issuer**、**jwksUri**、および **selector** を適宜置き換えます。

OpenID Policy の例

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: bookinfo
spec:
  selector:
    matchLabels:
      app: productpage
  jwtRules:
  - issuer: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak
    jwksUri: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak/protocol/openid-
connect/certs
```

2.11.4.1.4. ハイブリッド認証方法

特定の認証方法を適用せず、いずれかの方法の有効な認証情報を受け入れる方法を選択できます。API

特定の認証方法を適用せず、いずれかの方法の有効な認証情報を受け入れる方法を選択できます。API キーとアプリケーション ID/アプリケーションキーペアの両方が提供される場合、Service Mesh は API キーを使用します。

この例では、Service Mesh がクエリーパラメーターの API キーをチェックし、次に要求ヘッダーを確認します。API キーがない場合、クエリーパラメーターのアプリケーション ID とキーをチェックし、次に要求ヘッダーを確認します。

ハイブリッド認証方法の例

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

2.11.5. 3scale アダプターメトリクス

アダプターはデフォルトで、**/metrics** エンドポイントのポート **8080** で公開されるさまざまな Prometheus メトリクスを報告します。これらのメトリクスから、アダプターと 3scale 間の対話方法についての洞察が提供されます。サービスには、自動的に検出され、Prometheus によって収集されるようにラベルが付けられます。

2.11.6. 3scale Istio Adapter の検証

3scale Istio Adapter が予想通りに機能しているかどうかを確認します。アダプターが機能しない場合は、以下の手順に従って問題のトラブルシューティングを行うことができます。

手順

1. **3scale-adapter** Pod が Service Mesh コントロールプレーン namespace で実行されていることを確認します。

```
$ oc get pods -n <istio-system>
```

2. そのバージョンなど、**3scale-adapter** Pod が起動に関する情報を出力したことを確認します。

```
$ oc logs <istio-system>
```

3. 3scale Adapter の統合で保護されているサービスに対して要求を実行すると、正しい認証情報がかけているという要求を必ず試し、その要求が失敗することを確認します。3scale Adapter ログをチェックして、追加情報を収集します。

関連情報

- Pod およびコンテナーログの検査

2.11.7. 3scale Istio adapter のトラブルシューティングのチェックリスト

管理者が 3scale Istio adapter をインストールすると、統合が適切に機能しなくなる可能性のあるシナリオが複数あります。以下の一覧を使用して、インストールのトラブルシューティングを行います。

- YAML のインデントが間違っている。
- YAML セクションがない。
- YAML の変更をクラスターに適用するのを忘れている。
- **service-mesh.3scale.net/credentials** キーでサービスのワークロードにラベルを付けるのを忘れている。
- **service_id** が含まれないハンドラーを使用してアカウントごとに再利用できるようにする時に **service-mesh.3scale.net/service-id** サービスワークロードにラベルを付けるのを忘れている。
- **Rule** カスタムリソースが誤ったハンドラーまたはインスタンスカスタムリソースを参照しているか、対応する namespace の接尾辞がかけている参照を指定している。
- **Rule** カスタムリソースの **match** セクションは、設定中のサービスと同じでない可能性があるか、現在実行中でない、または存在しない宛先ワークロードを参照している。
- ハンドラーの 3scale 管理ポータルのアクセストークンまたは URL が正しくない。
- クエリーパラメーター、ヘッダー、承認要求などの誤った場所を指定しているか、パラメーター名がテストで使用する要求と一致しないため、インスタンス のカスタムリソースの **params/subject/properties** セクションで、 **app_id**、**app_key** または **client_id** の正しいパラメーターの表示に失敗する。
- 設定ジェネレーターがアダプターコンテナーイメージに実際に存在しており、**oc exec** で呼び出す必要があることに気づかなかったため、設定ジェネレーターの使用に失敗する。

2.12. SERVICE MESH の削除

警告

こちらは、サポートされなくなった **Red Hat OpenShift Service Mesh** リリースのドキュメントです。

Service Mesh バージョン 1.0 および 1.1 コントロールプレーンはサポートされなくなりました。Service Mesh コントロールプレーンのアップグレードについては、Service Mesh の アップグレード を参照してください。

特定の Red Hat Service Mesh リリースのサポートステータスについては、製品ライフサイクルページ を参照してください。

既存の OpenShift Container Platform インスタンスから Red Hat OpenShift Service Mesh を削除するには、コントロールプレーンを削除してから、Operator を削除します。

2.12.1. Red Hat OpenShift Service Mesh コントロールプレーンの削除

Service Mesh を既存の OpenShift Container Platform インスタンスからアンインストールするには、最初に Service Mesh コントロールプレーンおよび Operator を削除します。次に、コマンドを実行して残りのリソースを削除します。

2.12.1.1. Web コンソールを使用した Service Mesh コントロールプレーンの削除

Web コンソールを使用して Red Hat OpenShift Service Mesh コントロールプレーンを削除します。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
3. **Operators → Installed Operators** に移動します。
4. **Provided APIs** の **Service Mesh Control Plane** をクリックします。
5. **ServiceMeshControlPlane** メニュー ⋮ をクリックします。
6. **Delete Service Mesh Control Plane**をクリックします。
7. 確認ダイアログウィンドウで **Delete** をクリックし、**ServiceMeshControlPlane** を削除します。

2.12.1.2. CLI を使用した Service Mesh コントロールプレーンの削除

CLI を使用して Red Hat OpenShift Service Mesh コントロールプレーンを削除します。この例では、**istio-system** は、コントロールプレーンプロジェクトです。

手順

1. OpenShift Container Platform CLI にログインします。
2. 次のコマンドを実行して、**ServiceMeshMemberRoll** リソースを削除します。

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

3. 以下のコマンドを実行して、インストールした **ServiceMeshControlPlane** の名前を取得します。

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

4. **<name_of_custom_resource>** を先のコマンドの出力に置き換え、以下のコマンドを実行してカスタムリソースを削除します。

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

■

2.12.2. インストールされた Operator の削除

Red Hat OpenShift Service Mesh を正常に削除するには、Operator を削除する必要があります。Red Hat OpenShift Service Mesh Operator を削除したら、Kiali Operator、Red Hat OpenShift 分散トレースプラットフォーム Operator、および OpenShift Elasticsearch Operator を削除する必要があります。

2.12.2.1. Operator の削除

以下の手順に従って、Red Hat OpenShift Service Mesh を設定する Operator を削除します。以下の Operator ごとに手順を繰り返します。

- Red Hat OpenShift Service Mesh
- Kiali
- Red Hat OpenShift 分散トレースプラットフォーム
- OpenShift Elasticsearch

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operator → Installed Operators** ページから、スクロールするか、またはキーワードを **Filter by name** に入力して各 Operator を見つけます。次に、Operator 名をクリックします。
3. **Operator Details** ページで、**Actions** メニューから **Uninstall Operator** を選択します。プロンプトに従って各 Operator をアンインストールします。

2.12.2.2. Operator リソースのクリーンアップ

以下の手順に従って、OpenShift Container Platform Web コンソールを使用して、Red Hat OpenShift Service Mesh Operator を削除した後に残ったリソースを手動で削除します。

前提条件

- クラスター管理アクセスを持つアカウント。
- **oc** として知られる OpenShift Container Platform コマンドラインインターフェイス (CLI) へのアクセス。

手順

1. クラスター管理者として OpenShift Container Platform CLI にログインします。
2. 以下のコマンドを実行して、Operator のアンインストール後にリソースをクリーンアップします。サービスメッシュなしで Jaeger をスタンドアロンのサービスとして引き続き使用する場合は、Jaeger リソースを削除しないでください。

注記

Operator はデフォルトで **openshift-operators** namespace にインストールされます。Operator を別の namespace にインストールしている場合、**openshift-operators** を Red Hat OpenShift Service Mesh Operator がインストールされていたプロジェクトの名前に置き換えます。

```
$ 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>
```