# OpenShift Container Platform 4.3

## サービスメッシュ

サービスメッシュのインストール、使用法、およびリリースノート

Last Updated: 2020-09-21

# OpenShift Container Platform 4.3 サービスメッシュ

サービスメッシュのインストール、使用法、およびリリースノート

## 法律上の通知



## 概要

本書では、OpenShift Container Platform でサービスメッシュを使用する方法について説明します。

# 目次

# 第1章 SERVICE MESH リリースノート

## 1.1. RED HAT OPENSHIFT SERVICE MESH の概要

Red Hat OpenShift Service Mesh は動作についての洞察、およびサービスメッシュに対する運用上の制御を提供するプラットフォームであり、マイクロサービスアプリケーションへの接続、そのセキュリティー保護、およびモニターを実行するための統一した方法を提供します。

**サービスメッシュ** という用語は、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構成するマイクロサービスのネットワークおよびマイクロサービス間の対話を説明するために使用されます。サービスメッシュ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。

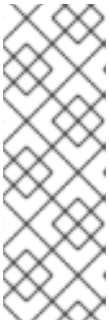
オープンソースの Istio プロジェクトをベースとする Red Hat OpenShift Service Mesh は、サービスコードに変更を加えずに、既存の分散したアプリケーションに透過的な層を追加します。マイクロサービス間のネットワーク通信をすべてインターセプトする環境全体に特別なサイドカープロキシーをデプロイすることで、Red Hat OpenShift Service Mesh のサポートをサービスに追加することができます。コントロールプレーンの機能を使用してサービスメッシュを設定し、管理します。

Red Hat OpenShift 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。

## 1.2. サポート

このドキュメントで説明する手順に関連した問題が発生する場合は、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から、以下を行うことができます。

- Red Hat 製品に関する技術サポート記事の 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。



**注記**

サポートケースを送信する際、Red Hat サポートのトラブルシューティングに役立つ以下の情報を提供していただくことをお勧めします。

- **oc adm must-gather** コマンドを使用して収集されるデータ
- 一意のクラスター ID。**(?) Help → Open Support Case** に移動すると、ケースの送信時にクラスター ID が自動入力されます。

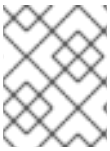
- 他の製品ドキュメントへのアクセス。

本書の改善が提案されている場合や、エラーが見つかった場合は、**Documentation** コンポーネントの **OpenShift Container Platform** 製品に対して、Bugzilla レポート 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。

## 1.3. 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構成です。

- Red Hat OpenShift Container Platform バージョン 4.x。



**注記**

OpenShift Online および OpenShift Dedicated は Red Hat OpenShift Service Mesh 1.1.7 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、すべてのサービスメッシュコンポーネントが OpenShift 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。

### 1.3.1. 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。

### 1.3.2. 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

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。

### 1.3.3. Red Hat OpenShift Service Mesh バージョン 1.1.7 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 | 1.4.8 |

| コンポーネント | バージョン |
| --- | --- |
| Jaeger | 1.17.4 |
| Kiali | 1.12.7 |
| 3scale Istio Adapter | 1.0.0 |

### 1.3.4. 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。

### 1.3.5. 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。

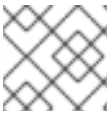
### 1.3.6. Red Hat OpenShift Service Mesh 1.1.5 の新機能

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。詳細は、暗号化スイートおよび ECDH 曲線の設定について参照してください。

### 1.3.7. 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。

### 1.3.8. 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。

### 1.3.9. 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。

### 1.3.10. 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。

### 1.3.11. Red Hat OpenShift Service Mesh 1.1.0 の新機能

Red Hat OpenShift Service Mesh のリリースでは、1.4.6 および Jaeger 1.17.1 のサポートが追加されました。

### 1.3.12. Red Hat OpenShift Service Mesh 1.0.11 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。

### 1.3.13. Red Hat OpenShift Service Mesh 1.0.10 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。

### 1.3.14. Red Hat OpenShift Service Mesh 1.0.9 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。

### 1.3.15. Red Hat OpenShift Service Mesh 1.0.8 の新機能

Red Hat OpenShift Service Mesh の本リリースは、OpenShift Container Platform 4.4 との互換性に関する問題に対応しています。OpenShift Container Platform 4.3 を 4.4 にアップグレードする前に、Red Hat OpenShift Service Mesh を 1.0.8 にアップグレードする必要があります。

### 1.3.16. Red Hat OpenShift Service Mesh 1.0.7 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。

### 1.3.17. Red Hat OpenShift Service Mesh 1.0.6 の新機能

本リリースには、内部で改良された機能が含まれています。

### 1.3.18. Red Hat OpenShift Service Mesh 1.0.5 の新機能

本リリースには、内部で改良された機能が含まれています。

### 1.3.19. Red Hat OpenShift Service Mesh 1.0.4 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、Kiali 1.0.9 のサポートが追加されました。さらに、CVE (Common Vulnerabilities and Exposures) に対応しています。

### 1.3.20. Red Hat OpenShift Service Mesh 1.0.3 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、Kiali 1.0.8 のサポートが追加されました。さらに、CVE (Common Vulnerabilities and Exposures) に対応しています。

### 1.3.21. Red Hat OpenShift Service Mesh 1.0.2 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、Istio 1.1.17、Jaeger 1.13.1、Kiali 1.0.7、および 3scale Istio Adapter 1.0 および OpenShift Container Platform 4.2 のサポートが追加されました。

### 1.3.22. Red Hat OpenShift Service Mesh 1.0.1 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、Istio 1.1.11、Jaeger 1.13.1、Kiali 1.0.6、3scale Istio Adapter 1.0 および OpenShift Container Platform 4.1 のサポートが追加されました。

### 1.3.23. Red Hat OpenShift Service Mesh 1.0 の新機能

Red Hat OpenShift Service Mesh の本リリースでは、Istio 1.1.11、Jaeger 1.13.1、Kiali 1.0.5、3scale Istio Adapter 1.0 および OpenShift Container Platform 4.1 のサポートが追加されました。

このリリースにおけるその他の主な変更点は、以下のとおりです。

- Kubernetes Container Network Interface (CNI) プラグインは常にオンになっています。
- コントロールプレーンは、デフォルトでマルチテナンシーに対して設定されます。単一テナントの、クラスター全体でのコントロールプレーン設定は推奨されていません。
- Elasticsearch、Jaeger、Kiali、および Service Mesh Operator は OperatorHub からインストールされます。
- コントロールプレーンのテンプレートを作成し、指定することができます。
- 本リリースから自動ルート作成が削除されました。

## 1.4. 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。

### 1.4.1. 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本リリースで非推奨となり、今後のリリースで削除されます。

- **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

## 1.5. 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 プロジェクトでサポートされておらず、また OpenShift 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。

### 1.5.1. サービスメッシュの既知の問題

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

- 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 ダッシュボードは依然として存在しています。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。
- 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。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。
- 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。これについては、進行中のソリューションと回避策があります。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、現時点でサービスメッシュ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。
  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-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。

- 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。

### 1.5.2. Kiali の既知の問題

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。

### 1.5.3. Jaeger の既知の問題

Jaeger には、以下の制限があります。

- Kafka パブリッシャーは Jaeger の一部として組み込まれていますが、サポートされていません。
- Apache Spark はサポートされていません。
- セルフプロビジョニングされた Elasticsearch インスタンスのみがサポートされます。本リリースでは、外部 Elasticsearch インスタンスはサポートされません。

Jaeger の既知の問題は以下のとおりです。

- 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。

## 1.6. 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。

### 1.6.1. サービスメッシュの修正された問題

- 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。
CRD を削除するには、以下のコマンドを実行します。

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

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

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

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

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

- MAISTRA-1649 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。
- 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。

- 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-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 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 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。

### 1.6.2. 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 サービスメッシュでラインタイムメトリクスが失敗します。サービスメッシュと 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。

- 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。

# 第2章 サービスメッシュアーキテクチャー

## 2.1. RED HAT OPENSHIFT 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。

### 2.1.1. 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構成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。サービスメッシュ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。コントロールプレーンの機能を使用してサービスメッシュを設定し、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
- レート制限
- アクセス制御
- エンドツーエンド認証

### 2.1.2. 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 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構成要素です。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。

### 2.1.3. Red Hat OpenShift Service Mesh コントロールプレーン

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。サービスメッシュにアクセスできるプロジェクトを指定し、サービスメッシュ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。

### 2.1.4. Red Hat OpenShift Service Mesh におけるマルチテナンシーとクラスター全体のインストール

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、コントロールプレーンのデプロイメント (Galley や Pilot など) 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が、コンポーネントはプロジェクトスコープの **RoleBinding** に依存します。

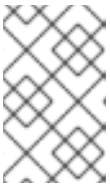
**members** 一覧のすべてのプロジェクトには、コントロールプレーンのデプロイメントに関連付けられた各サービスアカウントの **RoleBinding** があり、各コントロールプレーンのデプロイメントはそれらのメンバープロジェクトのみを監視します。各メンバープロジェクトには **maistra.io/member-of** ラベルが追加されており、**member-of** の値はコントロールプレーンのインストールが含まれるプロジェクトになります。

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。詳細な設定は、OpenShift SDN (Software-defined Networking) 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。サービスメッシュ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)。サービスメッシュ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。

### 2.1.5. 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どの Pod にも自動的に挿入しませんが、自動的なサイドカーコンテナーの挿入についての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する必要があります。

### 2.1.6. 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、または特定のプレフィックスまたはサフィックス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>"
```

### 2.1.7. 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。

### 2.1.8. 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。

Istio CNI プラグインは Multus CNI を使用して有効にされます。Istio Operator は、メッシュの一部である各プロジェクトに **NetworkAttachmentDefinition** オブジェクトを作成します。このオブジェクトは **k8s.v1.cni.cncf.io/networks** アノテーションで参照され、挿入時に Pod に追加されます。

#### 2.1.8.1. 他の Multus CNI プラグインでの Istio CNI の使用

デフォルトで、Pod に既存の **k8s.v1.cni.cncf.io/networks** アノテーションが含まれる場合（Multus CNI を使用して macvlan ネットワークを Pod に追加する場合など)、アノテーションの値は上書きされます。値を保持し、Istio CNI を最後に追加するには、以下の例のように、**spec.istio.sidecarInjectorWebhook.injectPodRedirectAnnot** フィールドを **ServiceMeshControlPlane** オブジェクトで **true** に設定する必要があります。

```
kind: ServiceMeshControlPlane
...
spec:
  istio:
    sidecarInjectorWebhook:
      injectPodRedirectAnnot: true
...
```

JSON 形式のサポートが Red Hat OpenShift Service Mesh バージョン 1.1.5 で導入されました。以前の Red Hat OpenShift Service Mesh バージョンでは、**k8s.v1.cni.cncf.io/networks** アノテーションのテキスト形式のみがサポートされました。

### 2.1.9. 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。

### 2.1.10. 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。

## 2.2. KIALI の概要

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。

### 2.2.1. Kiali の概要

Kiali では、OpenShift Container Platform で実行されるサービスメッシュの可観測性 (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。

### 2.2.2. Kiali アーキテクチャー

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構成されます。

- **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 Telemetery 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 Jaeger をインストールすると、Kiali コンソールには Jaeger の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、Jaeger データを表示するには、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、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. 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 のサービスメッシュおよびアプリケーションのパフォーマンスをチャートに表示できます。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構成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。

## 2.3. 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構成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 つ以上のスパンで構成されます。

**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。

### 2.3.1. Jaeger の概要

サービスの所有者は、Jaeger 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。Jaeger は、最新のクラウドネイティブ、マイクロサービスベースのアプリケーションにおいてコンポーネント間の対話のモニタリング、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に使用できる、オープンソースの分散トレースプラットフォームです。

Jaeger 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

Jaeger は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。

### 2.3.2. Jaeger アーキテクチャー

Jaeger は、複数のコンポーネントで構成されており、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ースを提供します。検索ページで、トレースを検索し、個別のトレースを構成するスパンの詳細を確認することができます。

### 2.3.3. Jaeger の機能

Jaeger のトレース機能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 Jaeger データを表示できます。

- 高いスケーラビリティー: Jaeger 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します。

- Zipkin との後方互換性: Jaeger 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。

## 2.4. サービスメッシュと ISTIO の比較

Red Hat OpenShift Service Mesh のインストールは、複数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。

### 2.4.1. Red Hat OpenShift Service Mesh コントロールプレーン

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。サービスメッシュにアクセスできるプロジェクトを指定し、サービスメッシュ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。

### 2.4.2. Red Hat OpenShift Service Mesh におけるマルチテナンシーとクラスター全体のインストール

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、コントロールプレーンのデプロイメント (Galley や Pilot など) 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

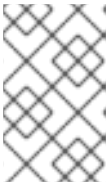
コープのロールベースのアクセス制御 (RBAC) リソース **ClusterRoleBinding** が使用されなくなりましたが、コンポーネントはプロジェクトスコープの **RoleBinding** に依存します。

**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。詳細な設定は、OpenShift SDN (Software-defined Networking) 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。サービスメッシュ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)。サービスメッシュ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。

### 2.4.3. 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どの Pod にも自動的に挿入しませんが、自動的なサイドカーコンテナーの挿入についての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する必要があります。

### 2.4.4. 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、または特定のプレフィックスまたはサフィックス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>"
```

### 2.4.5. 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。

### 2.4.6. 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。

Istio CNI プラグインは Multus CNI を使用して有効にされます。Istio Operator は、メッシュの一部である各プロジェクトに **NetworkAttachmentDefinition** オブジェクトを作成します。このオブジェクトは **k8s.v1.cni.cncf.io/networks** アノテーションで参照され、挿入時に Pod に追加されます。

#### 2.4.6.1. 他の Multus CNI プラグインでの Istio CNI の使用

デフォルトで、Pod に既存の **k8s.v1.cni.cncf.io/networks** アノテーションが含まれる場合（Multus CNI を使用して macvlan ネットワークを Pod に追加する場合など)、アノテーションの値は上書きされます。値を保持し、Istio CNI を最後に追加するには、以下の例のよう

に、**spec.istio.sidecarInjectorWebhook.injectPodRedirectAnnot** フィールドを **ServiceMeshControlPlane** オブジェクトで **true** に設定する必要があります。

```
kind: ServiceMeshControlPlane
...
spec:
  istio:
    sidecarInjectorWebhook:
      injectPodRedirectAnnot: true
...
```

JSON 形式のサポートが Red Hat OpenShift Service Mesh バージョン 1.1.5 で導入されました。以前の Red Hat OpenShift Service Mesh バージョンでは、**k8s.v1.cni.cncf.io/networks** アノテーションのテキスト形式のみがサポートされました。

### 2.4.7. 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。

### 2.4.8. Kiali とサービスメッシュ

OpenShift Container Platform でのサービスメッシュを使用した 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ます。
- Kiali ConfigMap が更新されています。
- Kiali の ClusterRole 設定が更新されています。
- ユーザーは、ConfigMap または Kiali カスタムリソースファイルを手動で編集できません。そのような変更はサービスメッシュまたは Kiali Operator によって上書きされる可能性があるためです。Red Hat OpenShift Service Mesh で実行している Kiali の設定はすべて **ServiceMeshControlPlane** カスタムリソースファイルで行われ、設定オプションは制限されています。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。

### 2.4.9. Jaeger とサービスメッシュ

OpenShift Container Platform でのサービスメッシュを使用した Jaeger 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。

- Jaeger はサービスメッシュに対してデフォルトで有効にされています。
- Ingress は、サービスメッシュ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 Jaeger 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。

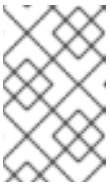
# 第3章 サービスメッシュのインストール

## 3.1. RED HAT OPENSHIFT 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。

### 3.1.1. 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 4.3 の概要」を確認します。
- OpenShift Container Platform 4.3 をインストールします。
  - AWS への OpenShift Container Platform 4.3 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 4.3 のインストール
  - ベアメタルへの OpenShift Container Platform 4.3 のインストール
  - vSphere への OpenShift Container Platform 4.3 のインストール



**注記**

ネットワークが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 4.3 を使用している場合は、「CLI について」を参照してください。

### 3.1.2. 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構成です。

- Red Hat OpenShift Container Platform バージョン 4.x。

**注記**

OpenShift Online および OpenShift Dedicated は Red Hat OpenShift Service Mesh 1.1.7 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、すべてのサービスメッシュコンポーネントが OpenShift 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。

#### 3.1.2.1. 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。

#### 3.1.2.2. 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

### 3.1.3. Red Hat OpenShift Service Mesh のインストールアクティビティー

Red Hat OpenShift Service Mesh Operator をインストールするには、まず以下の Operator をインストールする必要があります。

- **Elasticsearch**: オープンソースの Elasticsearch プロジェクトをベースとし、Jaeger を使用してトレースとロギングを行うために Elasticsearch クラスターを設定し、管理することができます。
- **Jaeger**: オープンソース Jaeger プロジェクトをベースとし、トレースを実行して、複雑な分散システムでトランザクションを監視し、トラブルシューティングできます。
- **Kiali**: オープンソースの Kiali プロジェクトをベースとしており、サービスメッシュの可観測性を提供します。Kiali を使用すると、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、トレースの表示と分析を実行できます。

Elasticsearch、Jaeger、Kiali Operator のインストール後に、Red Hat OpenShift Service Mesh Operator をインストールします。Service Mesh Operator は、サービスメッシュ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。

- **Red Hat OpenShift Service Mesh**: オープンソースの Istio プロジェクトに基づき、アプリケーションを構成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。

**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、Elasticsearch の設定について参照してください。

### 3.1.4. 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。

## 3.2. RED HAT OPENSHIFT SERVICE MESH のインストール

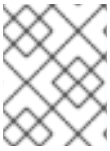
サービスメッシュのインストールには、Elasticsearch、Jaeger、Kiali、Service Mesh Operator のインストール、コントロールプレーンをデプロイするための **ServiceMeshControlPlane** リソースの作成および管理、サービスメッシュに関連する namespace を指定するための **ServiceMeshMemberRoll** リソースの作成が含まれます。

**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について参照してください。

**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は、Red Hat OpenShift Service Mesh 1.0 以降のデフォルト設定です。



**注記**

サービスメッシュ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。

### 3.2.1. 前提条件

- Red Hat OpenShift Service Mesh のインストールの準備 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。

### 3.2.2. OperatorHub からの Operator のインストール

サービスメッシュ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 1.1.7 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には、Elasticsearch Operator、Jaeger Operator、および Kiali Operator をインストールをする必要があります。

#### 3.2.2.1. Elasticsearch Operator のインストール

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 Elasticsearch 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 コンソールへのアクセス。

**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. **Elasticsearch** とフィルターボックスに入力して、Elasticsearch Operator を検索します。
4. **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. **Create Operator Subscription** ページで、**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。
7. Update Channel で最新のバージョンを選択します。
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. **Subscribe** をクリックします。
10. **Installed Operators** ページには、Elasticsearch Operator のインストールの進捗状況が表示されます。

#### 3.2.2.2. Jaeger Operator のインストール

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 Jaeger 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 コンソールへのアクセス。
- Elasticsearch Operator がインストールされていること。

**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. **Jaeger** とフィルターボックスに入力して、Jaeger Operator を検索します。
4. Red Hat が提供する **Jaeger 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. **Create Operator Subscription** ページで、**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。
7. **stable** Update Channel を選択します。
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. **Subscribe** をクリックします。
10. **Subscription Overview** ページには、Jaeger Operator のインストールの進捗状況が表示されます。

#### 3.2.2.3. Kiali Operator のインストール

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 コンソールへのアクセス。

**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. **Create Operator Subscription** ページで、**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。
7. **stable** Update Channel を選択します。
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. **Subscribe** をクリックします。
10. **Subscription Overview** 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。

#### 3.2.2.4. Red Hat OpenShift Service Mesh Operator のインストール

**前提条件**

- OpenShift Container Platform Web コンソールへのアクセス。
- Elasticsearch Operator がインストールされていること。
- Jaeger Operator がインストールされていること。
- Kiali Operator がインストールされていること。

**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. **Red Hat OpenShift Service Mesh**とフィルターボックスに入力して、Red Hat OpenShift Service Mesh Operator を検索します。
4. Red Hat OpenShift Service Mesh Operator をクリックし、Operator についての情報を表示します。
5. **Create Operator Subscription** ページで、**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。
6. **Install** をクリックします。
7. **stable** Update Channel を選択します。

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. **Subscribe** をクリックします。

10. **Subscription Overview** ページには、Red Hat OpenShift Service Mesh Operator のインストールの進捗状況が表示されます。

#### 3.2.2.5. 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サービスメッシュコントロールプレーンをデプロイすることができます。

##### 3.2.2.5.1. 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。

**前提条件**

- Red Hat OpenShift Service Mesh Operator がインストールされていること。
- 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。

**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、サービスメッシュ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。

#### 3.2.2.5.2. 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こと。
- 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。
- **oc** として知られる OpenShift Container Platform コマンドラインインターフェース (CLI) にアクセスします。

**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。

```
$ oc login 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
```

READY 列が true の場合、インストールは正常に終了しています。

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

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。詳しい情報は、「コントロールプレーンのテンプレートの作成」を参照してください。

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

**ServiceMeshMemberRoll** は、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。

**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。

**注記**

メンバープロジェクトは、サービスメッシュコントロールプレーンのインストールが成功した場合にのみ更新されます。

#### 3.2.2.6.1. Web コンソールからのメンバーロールの作成

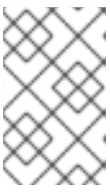
以下の手順に従って、Web コンソールを使用して 1 つ以上のプロジェクトを Service Mesh Member Roll に追加します。

**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。
- インストールされた **ServiceMeshControlPlane**の場所。
- 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. メッシュのプロジェクトがない場合や、ゼロから作業を開始する場合は、プロジェクトを作成します。これは **istio-system** とは異なるものである必要があります。
   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。
2. OpenShift Container Platform Web コンソールにログインします。
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** 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. **All Instances** タブをクリックします。
7. **Create New** をクリックしてから **Create Istio Service Mesh Member Roll**を選択します。



**注記**

Operator がリソースのコピーを終了するまでに少し時間がかかる場合があります。そのため、**Create Istio Service Mesh Member Roll**ボタンが表示されるように画面を更新する必要がある場合があります。

8. **Create Service Mesh Member Roll**ページで、YAML を変更してプロジェクトをメンバーとして追加します。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** をクリックして、Service Mesh Member Roll を保存します。

#### 3.2.2.6.2. CLI からのメンバーロールの作成

以下の手順に従って、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。
- インストールされた **ServiceMeshControlPlane**の場所。
- サービスメッシュに追加するプロジェクトの一覧。
- **oc** として知られる OpenShift Container Platform コマンドラインインターフェース (CLI) にアクセスします。

**手順**

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

```
$ oc login
```

2. **istio-system** の例では、**ServiceMeshMemberRoll** リソースを **ServiceMeshControlPlane** リソースと同じプロジェクトに作成します。リソースの名前は **default** にする必要があります。

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

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

3. デフォルトの YAML を変更して、プロジェクトを **members**として追加します。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。

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

**ServiceMeshMember** リソースは、**ServiceMeshMemberRoll** にメンバーを直接追加する特権を持たないサービスメッシュユーザーが作成できます。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、以下のようにユーザーに **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。

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

管理者は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、プロジェクトをそれが参照する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: control-plane-namespace
    name: minimal-install
```

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

#### 3.2.2.7. 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

以下の手順に従って、Web コンソールを使用して既存の Service Mesh **ServiceMeshMemberRoll** リソースを変更します。

- 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。

- **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。

#### 3.2.2.7.1. Web コンソールからのメンバーロールの変更

**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** 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。

#### 3.2.2.7.2. CLI からのメンバーロールの変更

以下の手順に従って、コマンドラインを使用して既存の Service Mesh Member Roll を変更します。

**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。
- 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。
- **oc** として知られる OpenShift Container Platform コマンドラインインターフェース (CLI) にアクセスします。

**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。

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

#### 3.2.2.8. Red Hat OpenShift Service Mesh メンバーロールの削除

**ServiceMeshMemberRoll** リソースは、関連付けられた **ServiceMeshControlPlane** リソースを削除する際に自動的に削除されます。

### 3.2.3. アプリケーション Pod の更新

Operator をインストールする際に Automatic Approval Strategy を選択した場合には、Operator はコントロールプレーンを自動的に更新しますが、アプリケーションを更新しません。既存のアプリケーションは、引き続きメッシュの一部になり、それに応じて機能します。アプリケーション管理者は、サイドカーコンテナーをアップグレードするためにアプリケーション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。

### 3.2.4. 次のステップ

- Red Hat OpenShift Service Mesh インストールをカスタマイズ します。
- 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。

## 3.3. RED HAT OPENSHIFT SERVICE MESH インストールのカスタマイズ

デフォルトのサービスメッシュ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。

### 3.3.1. 前提条件

- **cluster-admin** ロールを持つアカウント。
- 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こと。

### 3.3.2. Red Hat OpenShift Service Mesh カスタムリソース

**注記**

**istio-system** プロジェクトは、サービスメッシュ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。サービスメッシュをデプロイすると、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 にサービスメッシュ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 1.1.7 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。

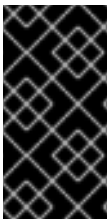
**istio-installation.yaml の例**

```
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
```

### 3.3.3. 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。

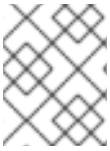


**重要**

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 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。

#### 3.3.3.1. 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3.1 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 |
|---|---|---|---|
| **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/** |

| パラメーター | 説明 | 値 | デフォルト値 |
| --- | --- | --- | --- |
| **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3.2 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 |
| --- | --- | --- | --- | --- |
| リソース | **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** |
| | **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)。 | **1024Mi** |
| 制限 | **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)。 | **128Mi** |

#### 3.3.3.2. 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:
    istio-egressgateway:
      autoscaleEnabled: false
      autoscaleMin: 1
      autoscaleMax: 5
    istio-ingressgateway:
      autoscaleEnabled: false
      autoscaleMin: 1
      autoscaleMax: 5
      ior_enabled: true
```

**表3.3 Istio ゲートウェイパラメーター**

| タイプ | パラメーター | 説明 | 値 | デフォルト値 |
|---|---|---|---|---|
| **istio-egressgateway** | **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。 | **true/false** | **true** |
| | **autoscaleMin** | **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** |
| | **autoscaleMax** | **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** |
| **istio-ingressgateway** | **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。 | **true/false** | **true** |
| | **autoscaleMin** | **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** |

| タイプ | パラメーター | 説明 | 値 | デフォルト値 |
|---|---|---|---|---|
| | **autoscaleMax** | **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** |
| | **ior_enabled** | 自動ルート作成を有効にするかどうかを制御します。 | **true**/**false** | **false** |

#### 3.3.3.3. 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3.4 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** |

| パラメーター | 説明 | 値 | デフォルト値 |
| --- | --- | --- | --- |
| **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3.5 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 |
| --- | --- | --- | --- | --- |
| リソース | **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** |
| 制限 | **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** |
| | **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** |

#### 3.3.3.4. Istio Pilot 設定

以下の例は、**ServiceMeshControlPlane** の Istio 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**

```
pilot:
  resources:
    requests:
      cpu: 100m
      memory: 128Mi
  autoscaleEnabled: false
  traceSampling: 100
```

表3.6 Istio Pilot パラメーター

| 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** |

### 3.3.4. 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3.7 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<br>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サービスメッシュを変更できなくなります。 | **true/false** | **false** |
| ingress<br>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** |

#### 3.3.4.1. 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
```

#### 3.3.4.2. 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
```

### 3.3.5. Jaeger の設定

Service Mesh Operator は **ServiceMeshControlPlane** リソースを作成する際に、Jaeger リソースも作成します。次に Jaeger Operator は Jaeger 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 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:
  istio:
    tracing:
      enabled: true
      jaeger:
        template: all-in-one
```

**表3.8 Jaeger パラメーター**

| パラメーター | 説明 | 値 | デフォルト値 |
|---|---|---|---|
| tracing<br>enabled | このパラメーターは、サービスメッシュでトレースを有効/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** |
| jaeger<br>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。<br>• **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、実稼働環境内にサービスメッシュ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。

#### 3.3.5.1. 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。

実稼働環境でサービスメッシュおよび 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3.9 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、サービスメッシュでトレースを有効/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 |
| 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。

Elasticsearch を OpenShift Container Platform で設定する方法については、「Elasticsearch の設定」を参照してください。

### 3.3.6. 3scale の設定

以下の例は、Red Hat OpenShift Service Mesh カスタムリソースの 3scale Istio Adapter 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。

**3scale パラメーターの例**

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

**表3.10 3scale 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 |
| --- | --- | --- | --- |
| **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** |
| **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** |

| パラメーター | 説明 | 値 | デフォルト値 |
| --- | --- | --- | --- |
| **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 |

### 3.3.7. 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。

## 3.4. RED HAT OPENSHIFT SERVICE MESH のアップグレード

### 3.4.1. 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
```

### 3.4.2. 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>
```

   以下は例になります。

```
$ 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
```

### 3.4.3. 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。

### 3.4.4. 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。

## 3.5. RED HAT OPENSHIFT SERVICE MESH の削除

このプロセスでは、既存の OpenShift Container Platform インスタンスから Red Hat OpenShift Service Mesh を削除できます。Operator を削除する前にコントロールプレーンを削除します。

### 3.5.1. Red Hat OpenShift Service Mesh コントロールプレーンの削除

OpenShift Container Platform Web コンソールまたは CLI を使用してサービスメッシュコントロールプレーンを削除できます。

#### 3.5.1.1. Web コンソールを使用したコントロールプレーンの削除

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削除します。

**前提条件**

- Red Hat OpenShift Service Mesh コントロールプレーンがデプロイされている必要があります。

**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、一覧から **istio-system** プロジェクトを選択します。
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** メニューで  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。

#### 3.5.1.2. CLI からのコントロールプレーンの削除

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削除します。

**前提条件**

- Red Hat OpenShift Service Mesh コントロールプレーンがデプロイされている必要があります。
- **oc** として知られる OpenShift Container Platform コマンドラインインターフェース (CLI) へのアクセス。

**手順**

**ServiceMeshControlPlane** を削除すると、サービスメッシュは Operator にインストールしたすべてのものをアンインストールするよう指示します。

**ヒント**

**servicemeshcontrolplane** の代わりに短い **smcp** エイリアスを使用できます。

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、インストールした **ServiceMeshControlPlane** の名前を取得します。

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

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

### 3.5.2. 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、Jaeger Operator、Kiali Operator、および Elasticsearch Operator を削除する必要があります。

#### 3.5.2.1. Red Hat OpenShift Service Mesh Operator の削除

以下の手順に従って、Red Hat OpenShift Service Mesh Operator を削除します。

**前提条件**

- OpenShift Container Platform Web コンソールへのアクセス。
- Red Hat OpenShift Service Mesh Operator がインストールされていること。

**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 Red Hat OpenShift Service Mesh Operator を見つけます。次に、それをクリックします。
3. **Operator Details** ページの右側で、**Actions** ドロップダウンメニューから **Uninstall Operator** を選択します。
4. **Remove Operator Subscription** ウィンドウでプロンプトが表示されたら、インストールに関連するすべてのコンポーネントを削除する場合は、**Also completely remove the Operator from the selected namespace** チェックボックスをオプションで選択します。これにより CSV が削除され、次に Operator に関連付けられた Pod、Deployment、CRD および CR が削除されます。

#### 3.5.2.2. Jaeger Operator の削除

以下の手順に従って、Jaeger Operator を削除します。

**前提条件**

- OpenShift Container Platform Web コンソールへのアクセス。
- Jaeger Operator がインストールされていること。

**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 Jaeger Operator を見つけます。次に、それをクリックします。
3. **Operator Details** ページの右側で、**Actions** ドロップダウンメニューから **Uninstall Operator** を選択します。
4. **Remove Operator Subscription** ウィンドウでプロンプトが表示されたら、インストールに関連するすべてのコンポーネントを削除する場合は、**Also completely remove the Operator from the selected namespace** チェックボックスをオプションで選択します。これにより CSV

が削除され、次に Operator に関連付けられた Pod、Deployment、CRD および CR が削除されます。

#### 3.5.2.3. Kiali Operator の削除

以下の手順に従って、Kiali Operator を削除します。

**前提条件**

- OpenShift Container Platform Web コンソールへのアクセス。
- Kiali Operator がインストールされていること。

**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 Kiali Operator を見つけます。次に、それをクリックします。
3. **Operator Details** ページの右側で、**Actions** ドロップダウンメニューから **Uninstall Operator** を選択します。
4. **Remove Operator Subscription** ウィンドウでプロンプトが表示されたら、インストールに関連するすべてのコンポーネントを削除する場合は、**Also completely remove the Operator from the selected namespace** チェックボックスをオプションで選択します。これにより CSV が削除され、次に Operator に関連付けられた Pod、Deployment、CRD および CR が削除されます。

#### 3.5.2.4. Elasticsearch Operator の削除

以下の手順に従って、Elasticsearch Operator を削除します。

**前提条件**

- OpenShift Container Platform Web コンソールへのアクセス。
- Elasticsearch Operator がインストールされていること。

**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 Elasticsearch Operator を見つけます。次に、それをクリックします。
3. **Operator Details** ページの右側で、**Actions** ドロップダウンメニューから **Uninstall Operator** を選択します。
4. **Remove Operator Subscription** ウィンドウでプロンプトが表示されたら、インストールに関連するすべてのコンポーネントを削除する場合は、**Also completely remove the Operator from the selected namespace** チェックボックスをオプションで選択します。これにより CSV が削除され、次に Operator に関連付けられた Pod、Deployment、CRD および CR が削除されます。

#### 3.5.2.5. Operator リソースのクリーンアップ

以下の手順に従って、OperatorHub インターフェースを使用して、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ース (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。

**注記**

**<operator-project>** を、Red Hat OpenShift Service Mesh Operator がインストールされているプロジェクトの名前に置き換えます。通常、これは **openshift-operators** になります。

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

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

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

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

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

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

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

# 第4章 DAY TWO

## 4.1. RED HAT OPENSHIFT SERVICE MESH へのアプリケーションのデプロイ

アプリケーションをサービスメッシュ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。

### 4.1.1. 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

### 4.1.2. 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。開発テンプレートと本番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本番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。

#### 4.1.2.1. 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。
- 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ース (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。

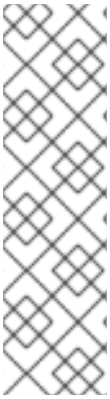
7. **ServiceMeshControlPlane** で **template** パラメーターを使用してテンプレートを指定できます。

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

### 4.1.3. Red Hat OpenShift Service Mesh のサイドカーコンテナー挿入

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コンテナー

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。自動のサイドカーコンテナー挿入を有効にしたり、手動で管理したりできます。Red Hat では、プロジェクトにラベルを付ける必要のない、アノテーションを使用した自動挿入を推奨しています。これにより、デプロイメント時に、アプリケーションにサービスメッシュの適切な設定が含まれるようになります。この方法で必要となる権限が少なく、ビルダー Pod などの他の OpenShift 機能と競合しません。



**注記**

プロジェクトにラベルを付けている場合、デフォルトで Istio のアップストリームバージョンはサイドカーコンテナーを挿入します。Red Hat OpenShift Service Mesh では、サイドカーコンテナーがデプロイメントに自動的に挿入されるようにオプトインすることが求められるため、プロジェクトにラベルを付ける必要はありません。これにより、(Pod のビルドまたはデプロイの場合など) 不要な場合にはサイドカーコンテナーを挿入しないようにできます。

Webhook はすべてのプロジェクトにデプロイする Pod の設定をチェックし、これらの Pod が適切なアノテーションで挿入をオプトインしているかどうかを確認します。

#### 4.1.3.1. アノテーションを使用したアプリケーションのプロキシーでの環境変数の設定

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。

```
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\" }"
```

#### 4.1.3.2. 自動のサイドカーコンテナー挿入の有効化

アプリケーションを Red Hat OpenShift Service Mesh にデプロイする場合は、**sidecar.istio.io/inject** アノテーションに値 **"true"** を指定して、挿入をオプトインする必要があります。オプトインにより、サイドカーコンテナーの挿入が OpenShift 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 機能に干渉しないようにします。

**前提条件**

- 自動のサイドカーコンテナー挿入を有効にするデプロイメントを特定します。
- アプリケーションの YAML 設定ファイルを見つけます。

**手順**

1. エディターでアプリケーションの設定 YAML ファイルを開きます。
2. **sidecar.istio.io/inject** を、以下に示すように **"true"** の値が含まれる設定 YAML に追加します。

   **スリープテストアプリケーションの例**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sleep
spec:
  replicas: 1
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true"
      labels:
        app: sleep
    spec:
      containers:
      - name: sleep
        image: tutum/curl
        command: ["/bin/sleep","infinity"]
        imagePullPolicy: IfNotPresent
```

3. 設定ファイルを保存します。

### 4.1.4. 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ース (CLI) へのアクセス。

**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。

### 4.1.5. 適切なネットワークポリシーの設定

サービスメッシュは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ホワイトリスト化します。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、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。
- メッシュ外のサービスは、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。

### 4.1.6. 次のステップ

- Red Hat OpenShift Service Mesh で Bookinfo をデプロイします。

## 4.2. 分散トレースのためのサービスメッシュの設定

このセクションでは、CRD または CR ファイルで実行される設定について説明します。

### 4.2.1. 前提条件

- cluster-admin ユーザー権限を使用した OpenShift Container Platform クラスターへのアクセス
- Elasticsearch Operator がクラスターにインストールされていること。
- Jaeger Operator がクラスターにインストールされていること。

### 4.2.2. 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。Jaeger 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4.1 Elasticsearch インデックスクリーナーパラメーター

| 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 |

## 4.3. アプリケーションの例

**警告**

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 1.1.7 のインストールをテストすることができます。

Red Hat では、Bookinfo アプリケーションをサポートしていません。

### 4.3.1. Bookinfo アプリケーション

アップストリームの Istio プロジェクトには Bookinfo というチュートリアルのサンプルがあり、これは各種の Istio 機能を示すために使用される 4 つの異なるマイクロサービスで構成されています。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。ページに表示される内容は、書籍の説明、書籍の詳細 (ISBN、ページ数その他の情報)、および書評です。

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

- **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。

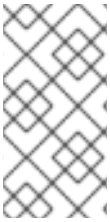
- バージョン v1 は、**ratings** サービスを呼び出しません。
- バージョン v2 は、**ratings** サービスを呼び出して、各評価を 1 から 5 の黒い星で表示します。
- バージョン v3 は、**ratings** サービスを呼び出して、各評価を 1 から 5 の赤い星で表示します。

### 4.3.2. 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、Bookinfo プロジェクトの作成、Bookinfo アプリケーションのデプロイ、および Service Mesh 1.1.7 を含む OpenShift Container Platform での Bookinfo の実行について説明します。

**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 1.1.7 がインストールされている。
- **oc** として知られる OpenShift Container Platform コマンドラインインターフェース (CLI) へのアクセス。



**注記**

Red Hat OpenShift Service Mesh は、アップストリームの Istio プロジェクトとは別の自動挿入を実装します。そのため、この手順では Red Hat OpenShift Service Mesh の Istio サイドカーコンテナーの自動挿入を有効にするためのアノテーションが付けられた **bookinfo.yaml** ファイルのバージョンを使用します。

**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。
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. **Project** メニューをクリックし、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** リンクをクリックします。
   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. Istio Service Mesh Member Roll を作成していない場合は、**Create Service Mesh Member Roll** をクリックします。

**注記**

Istio Service Mesh Member Roll を編集するには cluster-admin 権限が必要になります。

9. デフォルトの Service Mesh Member Roll YAML を編集し、**bookinfo** を **members** 一覧に追加します。

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

- 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 **ServiceMeshMemberRoll** に追加できます。**<control_plane_project>** をコントロールプレーンプロジェクトの名前に置き換えます。

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

10. **Create** をクリックして、更新した Service Mesh Member Roll を保存します。

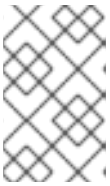
11. CLI で **'bookinfo'** プロジェクトに Bookinfo アプリケーションをデプロイするには、**bookinfo.yaml** ファイルを適用します。

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

12. **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-1.1/samples/bookinfo/networking/bookinfo-gateway.yaml
```

13. **GATEWAY_URL** パラメーターの値を設定します。



**注記**

**<control_plane_project>** をコントロールプレーンプロジェクトの名前に置き換えます。この例では、コントロールプレーンプロジェクトは **istio-system** です。

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

### 4.3.3. 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、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-1.1/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-1.1/samples/bookinfo/networking/destination-rule-all-mtls.yaml
     ```

### 4.3.4. Bookinfo インストールの検証

アプリケーションを設定する前に、正しくデプロイされていることを確認します。

**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 1.1.7 がインストールされている。
- **oc** として知られる OpenShift Container Platform コマンドラインインターフェース (CLI) へのアクセス。

**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、Bookinfo がデプロイされていることを確認します。

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

   - または、ブラウザーで **http://$GATEWAY_URL/productpage** を開くことができます。
   - 以下のコマンドでもすべての Pod が準備状態にあることを確認できます。

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

### 4.3.5. 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 1.1.7 がインストールされている。
- **oc** として知られる OpenShift Container Platform コマンドラインインターフェース (CLI) へのアクセス。

#### 4.3.5.1. 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** メニューで  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 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 delete project bookinfo
```

#### 4.3.5.2. 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 **openshift-operators** を選択します。
4. **Red Hat OpenShift Service Mesh Operator**の **Provided APIS** で、**Istio Service Mesh Member Roll** リンクをクリックします。
5. **ServiceMeshMemberRoll** メニューの  をクリックし、**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 からこのコマンドを実行して、**ServiceMeshMemberRoll** から **bookinfo** プロジェクトを削除できます。**<control_plane_project>** をコントロールプレーンプロジェクトの名前に置き換えます。

```
$ oc -n <control_plane_project> 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。

## 4.4. 分散トレースのチュートリアル

Jaeger はオープンソースの分散トレースシステムです。Jaeger はマイクロサービスベースの分散システムの監視およびトラブルシューティングに使用します。Jaeger を使用すると、トレースを実行できます。これは、アプリケーションを構成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 はデフォルトでサービスメッシュ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、サービスメッシュと bookinfo のチュートリアルを使用して、Jaeger で分散トレースを実行する方法を示します。

**注記**

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 1.1.7 のインストールをテストすることができます。

Red Hat では、Bookinfo アプリケーションをサポートしていません。

### 4.4.1. トレースの生成とトレースデータの分析

このチュートリアルでは、サービスメッシュおよび Bookinfo チュートリアルを使用して、Red Hat OpenShift Service Mesh の Jaeger コンポーネントでトレースを実行する方法を説明します。

**前提条件:**

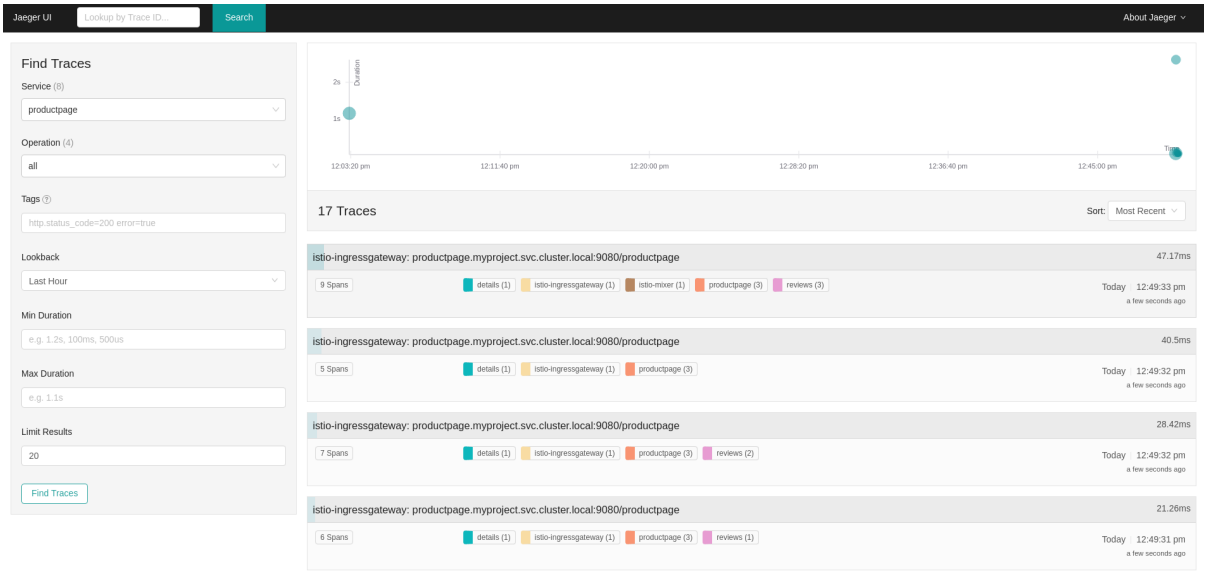
- OpenShift Container Platform 4.1 以降がインストールされている。
- Red Hat OpenShift Service Mesh 1.1.7 がインストールされている。
- インストール時に Jaeger が有効にされている。
- Bookinfo のサンプルアプリケーションがインストールされている。

**手順**

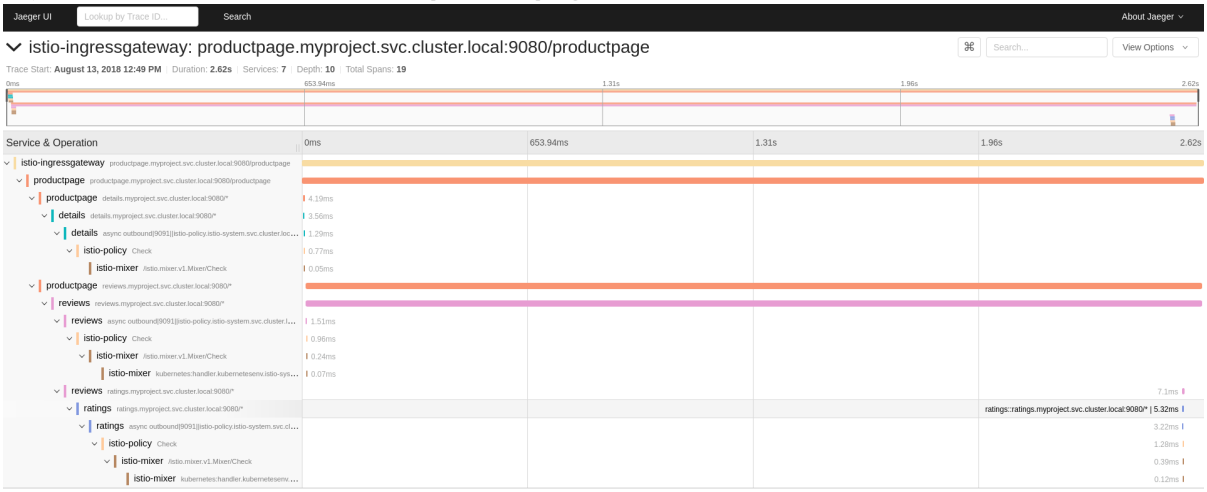
1. Bookinfo アプリケーションをデプロイした後、いくつかのトレースデータを分析できるように、Bookinfo アプリケーションへの呼び出しを生成する必要があります。http://<GATEWAY_URL>/productpage にアクセスし、ページを数回更新すると、トレースデータを生成することができます。
2. インストールプロセスにより、Jaeger コンソールにアクセスするためのルートが作成されます。
   a. OpenShift Container Platform コンソールで、**Networking → Routes** に移動し、Jaeger ルートを検索します。これは **Location** に一覧される URL です。
   b. CLI を使用してルートの詳細のクエリーを実行します。

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

3. ブラウザーを起動して、**https://<JAEGER_URL>** に移動します。
4. 必要に応じて、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. 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」を選択し、ペイン下部の **Find Traces** ボタンをクリックします。以下のイメージに示されているように、トレースの一覧が表示されます。




6. 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。最上部 (最新の) トレースをクリックすると、**'/productpage** の最終更新に対応する詳細が表示されます。




先の図のトレースは、一部のネストされたスパンで構成されており、各スパンは Bookinfo サービス呼び出しに対応し、すべてが **'/productpage** 要求の応答で実行されます。全体的な処理時間は 2.62s で、**details** サービスは 3.56ms、**reviews** サービスは 2.6s、**ratings** サービスは 5.32ms かかりました。リモートサービスへの各呼び出しは、それぞれクライアント側とサーバー側のスパンで表されます。たとえば、**details** クライアント側スパンには **productpage details.myproject.svc.cluster.local:9080** というラベルが付けられます。その下にネスト化されるスパンには、**details details.myproject.svc.cluster.local:9080** というラベルが付けられ、要求のサーバー側の処理に対応します。トレースは **istio-policy** への呼び出しも表示し、これには Istio による認可チェックが反映されます。

## 4.5. 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。

### 4.5.1. 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
```

詳細は、Istio ゲートウェイの設定について参照してください。

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 <your-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。

# 第5章 SERVICE MESH ユーザーガイド

## 5.1. トラフィック管理

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。

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。

### 5.1.1. トラフィックのルーティングおよび管理

YAML ファイルのカスタムリソース定義を使用して、独自のトラフィック設定を Red Hat OpenShift Service Mesh に追加してサービスメッシュを設定します。

#### 5.1.1.1. 仮想サービスの使用によるトラフィック管理

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。単一の仮想サービスを数多くのサービスにマッピングすることは、モノリシックなアプリケーションの、別個のマイクロサービスから構築される複合サービスへの移行を容易にするために役立ちます。この際、サービスのコンシューマーには移行に伴う適応が必要となりません。
- 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。

##### 5.1.1.1.1. 仮想サービスの設定

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構成されます。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。

以下の例では、どのユーザーがアプリケーションに接続するかに応じて、異なるバージョンのサービスに要求をルーティングします。このコマンドを使用して、このサンプル YAML ファイル、または各自が作成する YAML ファイルを適用します。

```
$ oc apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
```

```
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v3
EOF
```

#### 5.1.1.2. 仮想ホストの設定

以下のセクションでは、YAML ファイルの各フィールド、および仮想サービスで仮想ホストを作成する方法について説明します。

##### 5.1.1.2.1. ホスト

**hosts** フィールドには、これらの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アドレス指定可能な宛先が一覧表示されます。これは、要求をサービスに送信する際にクライアントが使用する 1 つ以上のアドレスです。

仮想サービスのホスト名は、IP アドレス、DNS 名、またはプラットフォームによっては、完全修飾ドメイン名に解決される短縮名になります。

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

##### 5.1.1.2.2. ルーティングルール

**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、ユースケースに応じてゼロまたは 1 つ以上の一致条件で構成されます。

**一致条件**

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:
  hosts:
  - reviews
  http:
```

```
  - match:
    - headers:
      end-user:
        exact: jason
```

#### 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。

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

##### 5.1.1.2.3. 宛先ルール

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。

###### 5.1.1.2.3.1. 負荷分散オプション

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、インスタンス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
```

#### 5.1.1.2.4. ゲートウェイ

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、公開するポート、Red Hat OpenShift Service Mesh TLS 設定などの 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、メッシュから出るトラフィックの専用の出口ノードを設定し、外部ネットワークにアクセスできるサービスを制限したり、egress トラフィックのセキュアな制御を有効にしてメッシュにセキュリティーを追加することなどが可能になります。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**

以下の例は、外部 HTTPS Ingress トラフィックの予想される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。

#### 5.1.1.2.5. サービスエントリー

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設定すると、(以下のタスクを含め) メッシュの外部で実行されているサービスのトラフィックを管理できます。

- 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。
- 別のクラスターからメッシュにサービスを論理的に追加し、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。
- メッシュサービスが使用するすべての外部サービスのサービスエントリーを追加する必要はありません。Red Hat OpenShift Service Mesh はデフォルトで、Envoy プロキシーを不明なサービスへの要求をパススルーするように設定します。ただし、Red Hat OpenShift Service Mesh 機能を使用して、メッシュに登録されていない宛先へのトラフィックを制御することはできません。

**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、ワイルドカードのプレフィックス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
```

#### 5.1.1.2.6. サイドカー

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。
- より大規模なアプリケーションではこのようなサイドカーの到達可能性を制限する必要がある場合があります。この場合、すべてのプロキシーがメッシュ内の他のすべてのサービスに到達するように設定されると、メモリー使用率が高くなるためにメッシュのパフォーマンスに影響する可能性があります。

**サイドカーの例**

サイドカー設定を特定の namespace のすべてのワークロードに適用するように指定するか、または **workloadsSelector** を使用して特定のワークロードを選択することができます。たとえば、以下のサイドカー設定では **bookinfo** namespace 内のすべてのサービスを、同じ namespace および Red Hat OpenShift Service Mesh コントロールプレーン（現時点では Red Hat OpenShift Service Mesh ポリシーおよび Telemetry 機能を使用するために必要) で実行されるサービスのみに到達するように設定します。

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

### 5.1.2. 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サービスメッシュ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。サービスメッシュを、サービスメッシュ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するように設定します。

#### 5.1.2.1. Ingress IP およびポートの判別

以下のコマンドを実行して、Kubernetes クラスターが外部ロードバランサーをサポートする環境で実行されているかどうかを判別します。

```
$ 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。

環境の手順を選択します。

#### ロードバランサーを使用したルーティングの設定

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。

Ingress IP およびポートを設定します。

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

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

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

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。

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}')
```

#### ロードバランサーを使用しないルーティングの設定

お使いの環境に外部のロードバランサーがない場合は、以下の手順に従います。ノードポートを代わりに使用する必要があります。

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}')
```

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

### 5.1.3. Bookinfo アプリケーションを使用したルーティングの例

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構成されます。3 つの異なるバージョン (マイクロサービスの 1 つは **reviews** と呼ばれる) はデプロイされており、同時に実行されます。

**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します。

#### このタスクについて

これによって生じる問題を確認するには、ブラウザーで bookinfo アプリの **/product page** にアクセスし、複数回更新を実行します。

書評の出力に星評価が含まれる場合や、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、サービスメッシュは、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 要求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。

#### 5.1.3.1. 仮想サービスの適用

1 つのバージョンにのみルーティングするには、マイクる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ます。以下の例では、仮想サービスはすべてのトラフィックを各マイクロサービスの **v1** にルーティングします。

1. 以下のコマンドを実行して仮想サービスを適用します。

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

2. コマンドが正常に実行されることをテストするには、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
```

このコマンドは以下の YAML ファイルを返します。

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

```
  ...
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
      host: reviews
      subset: v1
```

サービスメッシュを Bookinfo マイクロサービスの **v1** バージョン (最も重要な点として **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。

#### 5.1.3.2. 新規ルーティング設定のテスト

Bookinfo アプリケーションの /**productpage** を再度更新すると、新しい設定を簡単にテストできます。

1. ブラウザーで Bookinfo サイトを開きます。URL は **http://$GATEWAY_URL/productpage** です。ここで、**$GATEWAY_URL** は Ingress の外部 IP アドレスです。
   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、サービスメッシュを、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。

#### 5.1.3.3. ユーザーアイデンティティーに基づくルート

次に、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。

サービスメッシュには、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がないことに注意してください。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 要求にカスタム **end-user** ヘッダーを追加することで有効にされます。

1. 以下のコマンドを実行して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-
1.1/samples/bookinfo/networking/virtual-service-reviews-test-v2.yaml
```

2. ルールが作成されていることを確認します。

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

このコマンドは以下の YAML ファイルを返します。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
  ...
spec:
  hosts:
```

```
    - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v1
```

3. Bookinfo アプリケーションの **/productpage** で、ユーザー **jason** としてログインします。ブラウザーを更新します。表示内容を確認してください。各レビューの横に星評価が表示されます。

4. 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。これは、Jason 以外のすべてのユーザーについてトラフィックが **reviews:v1** にルーティングされるためです。

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サービスメッシュが正常に設定されています。

## 5.2. データの可視化および可観測性

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。

アプリケーション経由でのデータフローは、アプリケーションがインストールされている場合に確認することができます。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。

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
```

サンプルアプリケーションが正しく設定されている場合、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。

### 5.2.1. Kiali コンソールへのアクセス

コンソールにアクセスするには、メニューバーで **Application launcher** > **Kiali** をクリックします。

1. OpenShift Container Platform メニューバーで、**Application launcher** > **Kiali** をクリックします。

2. 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。

3. **Namespace** フィールドでサービスのプロジェクトを選択します。Bookinfo のサンプルをインストールしている場合は、**bookinfo** を選択します。

**コマンドラインでの手順**

1. CLI からこのコマンドを実行して、ルートおよび Kiali URL を取得します。

```
$ oc get routes
```

**kiali** 行の出力で、HOST/PORT 列の URL を使用し、Kiali コンソールを開きます。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。**Namespace** フィールドでサービスのプロジェクトを選択します。

初回ログイン時に、表示するパーミッションを持つ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。

## 5.2.2. サービスの可視化

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。

Overview ページには、メッシュにサービスが含まれるすべての namespace が表示されます。サービスメッシュを通過するデータに関するより深い洞察を得ることや、以下のグラフや視覚化を使用してサービスメッシュ内のサービスやワークロードの問題を特定することができます。

### 5.2.2.1. namespace グラフ

namespace グラフは、namespace のサービス、デプロイメント、およびワークフローのマップであり、それらを通過するデータフローを示す矢印が表示されます。namespace グラフを表示するには、以下を実行します。

1. メインのナビゲーションにある **Graph** をクリックします。

2. **Namespace** メニューから **bookinfo** を選択します。

アプリケーションが Bookinfo サンプルアプリケーションなどのバージョンタグを使用する場合は、Version グラフが表示されます。Graph Type ドロップダウンメニューからグラフを選択します。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。

- 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すべての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。

- 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。

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。

## 5.3. サービスメッシュのセキュリティーのカスタマイズ

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。サービスメッシュ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。

### 5.3.1. 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、二者が同時に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、サービスメッシュのサイドカーは、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。ワークロードをサービスメッシュに移行する間に Permissive モードを使用します。

#### 5.3.1.1. 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
```

##### 5.3.1.1.1. 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.maistra.io/v1"
kind: "Policy"
metadata:
  name: "default"
  namespace: <NAMESPACE>
```

```
spec:
  peers:
  - mtls: {}
```

#### 5.3.1.2. 送信接続用のサイドカーの設定

宛先ルールを作成し、サービスメッシュ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
```

#### 5.3.1.3. 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.istio.global.tls.minProtocolVersion** または **spec.istio.global.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_0
```

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5.1 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 |

### 5.3.2. 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

### 5.3.3. 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。Operator 指定のルート証明書およびキーを使用して、Operator 指定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サービスメッシュ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。

#### 5.3.3.1. 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、証明書をサービスメッシュに追加します。Maistra リポジトリーから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。サービスメッシュは、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.*** という名前のサービスメッシュ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。サービスメッシュ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
```

#### 5.3.3.2. 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}'`
```

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。

2. 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。

3. 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。

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。

```
$ 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
```

#### 5.3.3.3. 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サービスメッシュを再デプロイします。

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

# 第6章 サポート

## 6.1. RED HAT サポート向けの RED HAT OPENSHIFT SERVICE MESH データの収集

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。

### 6.1.1. 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
- 監査ログ
- サービスログ

**--image** 引数を指定してコマンドを実行する際にイメージを指定できます。イメージを指定する際、ツールはその機能または製品に関連するデータを収集します。

**oc adm must-gather**を実行すると、新しい Pod がクラスターに作成されます。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。

### 6.1.2. 前提条件

- **cluster-admin** ロールを持つユーザーとしてのクラスターへのアクセスがあること。
- OpenShift Container Platform CLI（**oc**）がインストールされていること。

### 6.1.3. 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。

**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-rhel7
```

# 第7章 3SCALE アダプター

## 7.1. 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。

### 7.1.1. 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 0.12.0+
- 稼働している 3scale アカウント (SaaS または 3scale 2.5 On-Premises)
- 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。

**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。3scale の認証情報と管理する API のサービス ID を使用して、これを更新する必要があります。

1. 3scale 設定でハンドラー設定を変更します。

**ハンドラー設定の例**

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

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。

1. 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
```

#### 7.1.1.1. 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7.1 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 | No | |
| **--name** | この URL の一意の名前、トークンのペア | Yes | |
| **-n, --namespace** | テンプレートを生成する namespace | No | istio-system |
| **-t, --token** | 3scale アクセストークン | Yes | |
| **-u, --url** | 3scale 管理ポータル URL | Yes | |
| **--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。 | No | |
| **-s, --service** | 3scale API/サービス ID | No | |

| オプション | 説明 | 必須 | デフォルト値 |
|---|---|---|---|
| **--auth** | 指定する 3scale 認証パターン (1=API Key, 2=App Id/App Key, 3=OIDC) | No | 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 | No | 標準出力 |
| **--version** | CLI バージョンを出力し、即座に終了する | No | |

##### 7.1.1.1.1. URL サンプルからのテンプレートの生成

- この例では、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生成します。

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

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

#### 7.1.1.2. デプロイされたアダプターからのマニフェストの生成

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}""
```

#### 7.1.1.3. 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**

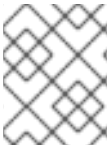
- 3scale 管理者から受け取る認証情報とサービス ID

**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。

### 7.1.2. 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 → Configuration** の順に移動します。
2. **Integration** ページの上部で、右上隅の **edit integration settings** をクリックします。
3. **Service Mesh** の見出しで、**Istio** オプションをクリックします。
4. ページの下部までスクロールし、**Update Service** をクリックします。

### 7.1.3. 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。

### 7.1.4. 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。

#### 7.1.4.1. 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。

##### 7.1.4.1.1. API キー認証方法

サービスメッシュは、**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、サービスメッシュは **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、サービスメッシュは **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。

#### 7.1.4.1.2. 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

サービスメッシュは、**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、サービスメッシュ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。

#### 7.1.4.1.3. 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 でサービスのエンドユーザー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。

以下で定義されるサンプル **Policy** では、**issuer** と **jwksUri** を適宜置き換えます。

**OpenID Policy の例**

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

#### 7.1.4.1.4. 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、サービスメッシュは API キーを使用します。

この例では、サービスメッシュ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"] | ""
```

### 7.1.5. 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。