



Red Hat OpenShift Service on AWS 4

Service Mesh

Service Mesh のインストール、使用法、およびリリースノート

Red Hat OpenShift Service on AWS 4 Service Mesh

Service Mesh のインストール、使用法、およびリリースノート

法律上の通知

Copyright © 2025 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

このドキュメントでは、Red Hat OpenShift Service on AWS で Service Mesh を使用する方法を説明します。

目次

第1章 SERVICE MESH 2.X	3
1.1. OPENSIFT SERVICE MESH について	3
1.2. SERVICE MESH リリースノート	4
1.3. SERVICE MESH について	69
1.4. SERVICE MESH のデプロイメントモデル	76
1.5. SERVICE MESH と ISTIO の相違点	85
1.6. SERVICE MESH のインストールの準備	91
1.7. OPERATOR のインストール	93
1.8. SERVICEMESHCONTROLPLANE の作成	96
1.9. SERVICE MESH へのサービスの追加	106
1.10. サイドカーコンテナの挿入の有効化	123
1.11. SERVICE MESH のアップグレード	127
1.12. ユーザーおよびプロファイルの管理	146
1.13. セキュリティー	148
1.14. SERVICE MESH でのトラフィックの管理	166
1.15. メトリクス、ログ、およびトレース	180
1.16. パフォーマンスおよびスケーラビリティ	200
1.17. 実稼働環境の SERVICE MESH の設定	203
1.18. SERVICE MESH の接続	204
1.19. 拡張	235
1.20. 3SCALE WEBASSEMBLY モジュールの使用	248
1.21. 3SCALE ISTIO アダプターの使用	269
1.22. SERVICE MESH のトラブルシューティング	281
1.23. ENVOY プロキシのトラブルシューティング	290
1.24. SERVICE MESH コントロールプレーン設定の参照	293
1.25. KIALI 設定リファレンス	307
1.26. JAEGER 設定リファレンス	311
1.27. SERVICE MESH のアンインストール	343

第1章 SERVICE MESH 2.X

1.1. OPENSIFT SERVICE MESH について



注記

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

Red Hat OpenShift Service Mesh のライフサイクルとサポートされるプラットフォームに関する追加情報は、[プラットフォームライフサイクルポリシー](#) を参照してください。

1.1.1. Red Hat OpenShift Service Mesh の概要

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

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

オープンソースの [Istio プロジェクト](#) をベースにする Service Mesh では、検出、負荷分散、サービス間の認証、障害復旧、メトリクス、およびモニタリングを提供する、デプロイされたサービスのネットワークを簡単に作成できます。Service Mesh 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。



注記

Red Hat OpenShift Service Mesh 3 が一般公開されました。詳細は、[Red Hat OpenShift Service Mesh 3.0](#) を参照してください。

1.1.2. コア機能

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

- **トラフィック管理:** サービス間でトラフィックおよび API 呼び出しのフローを制御し、呼び出しの安定度を高め、不利な条件下でもネットワークの堅牢性を維持します。
- **サービス ID とセキュリティ:** 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。
- テレメトリー: サービス間の依存関係やそれらの間のトラフィックの性質やフローを理解するのに役立ち、問題を素早く特定できます。

1.2. SERVICE MESH リリースノート

1.2.1. Red Hat OpenShift Service Mesh バージョン 2.6.7

この Red Hat OpenShift Service Mesh リリースでは、Red Hat OpenShift Service Mesh Operator バージョンが 2.6.7 に更新され、**ServiceMeshControlPlane** リソースバージョン 2.6.7 および 2.5.10 の更新が含まれています。

このリリースは、Common Vulnerabilities and Exposures (CVE) への対応を含んでおり、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。

Red Hat OpenShift Service Mesh Operator の最新バージョンは、サポートされているすべてのバージョンの Service Mesh で使用できます。Service Mesh のバージョンは、**ServiceMeshControlPlane** を使用して指定されます。

Red Hat が提供する Kiali Operator の最新バージョンは、サポートされているすべてのバージョンの Red Hat OpenShift Service Mesh で使用できます。Service Mesh のバージョンは、**ServiceMeshControlPlane** リソースを使用して指定されます。Service Mesh のバージョンにより、互換性のあるバージョンの Kiali が自動的に確保されます。

1.2.1.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.20.8
Envoy プロキシ	1.28.7
Kiali Server	1.73.20

1.2.2. Red Hat OpenShift Service Mesh バージョン 2.5.10

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.7 に含まれており、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。このリリースでは、Common Vulnerabilities and Exposures (CVE) に対応しています。

1.2.2.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.18.7
Envoy プロキシ	1.26.8

コンポーネント	バージョン
Kiali Server	1.73.20

1.2.3. Red Hat OpenShift Service Mesh バージョン 2.6.6

この Red Hat OpenShift Service Mesh リリースでは、Red Hat OpenShift Service Mesh Operator バージョンが 2.6.6 に更新され、**ServiceMeshControlPlane** リソースバージョン 2.6.6、2.5.9、および 2.4.15 の更新が含まれています。

このリリースは、Common Vulnerabilities and Exposures (CVE) への対応を含んでおり、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。

Red Hat が提供する Kiali Operator の最新バージョンは、サポートされているすべてのバージョンの Red Hat OpenShift Service Mesh で使用できます。Service Mesh のバージョンは、**ServiceMeshControlPlane** リソースを使用して指定されます。Service Mesh のバージョンにより、互換性のあるバージョンの Kiali が自動的に確保されます。

1.2.3.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.20.8
Envoy プロキシ	1.28.7
Kiali Server	1.73.19

1.2.3.2. 新機能

- この更新により、Red Hat OpenShift Service Mesh 2.6 の Operator の名前が Red Hat OpenShift Service Mesh 2 に変更され、Red Hat OpenShift Service Mesh 3.0 のリリースに合わせて明確さが向上しました。

1.2.4. Red Hat OpenShift Service Mesh バージョン 2.5.9

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.6 に含まれており、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。このリリースでは、Common Vulnerabilities and Exposures (CVE) に対応しています。

1.2.4.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.18.7
Envoy プロキシ	1.26.8

コンポーネント	バージョン
Kiali Server	1.73.19

1.2.5. Red Hat OpenShift Service Mesh バージョン 2.4.15

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.6 に含まれており、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。

1.2.5.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.16.7
Envoy プロキシ	1.24.12
Kiali Server	1.65.20

1.2.6. Red Hat OpenShift Service Mesh バージョン 2.6.5

この Red Hat OpenShift Service Mesh リリースでは、Red Hat OpenShift Service Mesh Operator バージョンが 2.6.5 に更新され、2.6.5、2.5.8、および 2.4.14 の **ServiceMeshControlPlane** リソースバージョンの更新が含まれています。

このリリースは、Common Vulnerabilities and Exposures (CVE) への対応を含んでおり、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。

Red Hat が提供する Kiali Operator の最新バージョンは、サポートされているすべてのバージョンの Red Hat OpenShift Service Mesh で使用できます。Service Mesh のバージョンは、**ServiceMeshControlPlane** リソースを使用して指定されます。Service Mesh のバージョンにより、互換性のあるバージョンの Kiali が自動的に確保されます。

1.2.6.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.20.8
Envoy プロキシ	1.28.7
Kiali Server	1.73.18

1.2.6.2. 新機能

- Red Hat OpenShift 分散トレーシング Platform Stack が IBM Z でサポートされるようになりました。

1.2.6.3. 修正された問題

- [OSSM-8608](#): 以前は、バイナリーをコピーしながらインストールフェーズで Container Network Interface (CNI) Pod を終了すると、ノードファイルシステムに Istio-CNI の一時ファイルが残る可能性があります。繰り返し発生すると、最終的にはノードのディスク領域がいっぱいになる可能性があります。現在、インストールフェーズ中に CNI Pod を終了する際に、CNI バイナリーをコピーする前に既存の一時ファイルが削除され、ノードファイルシステムには Istio バージョンごとに1つの一時ファイルのみが存在するようになりました。

1.2.7. Red Hat OpenShift Service Mesh バージョン 2.5.8

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.5 に含まれており、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。このリリースでは、Common Vulnerabilities and Exposures (CVE) に対応しています。

1.2.7.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.18.7
Envoy プロキシ	1.26.8
Kiali Server	1.73.18

1.2.7.2. 修正された問題

- [OSSM-8608](#): 以前は、バイナリーをコピーしながらインストールフェーズで Container Network Interface (CNI) Pod を終了すると、ノードファイルシステムに Istio-CNI の一時ファイルが残る可能性があります。繰り返し発生すると、最終的にはノードのディスク領域がいっぱいになる可能性があります。現在、インストールフェーズ中に CNI Pod を終了する際に、CNI バイナリーをコピーする前に既存の一時ファイルが削除され、ノードファイルシステムには Istio バージョンごとに1つの一時ファイルのみが存在するようになりました。

1.2.8. Red Hat OpenShift Service Mesh バージョン 2.4.14

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.5 に含まれており、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。このリリースでは、Common Vulnerabilities and Exposures (CVE) に対応しています。

1.2.8.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.16.7
Envoy プロキシ	1.24.12
Kiali Server	1.65.19

1.2.8.2. 修正された問題

- [OSSM-8608](#): 以前は、バイナリーをコピーしながらインストールフェーズで Container Network Interface (CNI) Pod を終了すると、ノードファイルシステムに Istio-CNI の一時ファイルが残る可能性があります。繰り返し発生すると、最終的にはノードのディスク領域がいっぱいになる可能性があります。現在、インストールフェーズ中に CNI Pod を終了する際に、CNI バイナリーをコピーする前に既存の一時ファイルが削除され、ノードファイルシステムには Istio バージョンごとに1つの一時ファイルのみが存在するようになりました。

1.2.9. Red Hat OpenShift Service Mesh バージョン 2.6.4

この Red Hat OpenShift Service Mesh リリースでは、Red Hat OpenShift Service Mesh Operator バージョンが 2.6.4 に更新され、2.6.4、2.5.7、2.4.13 の **ServiceMeshControlPlane** リソースバージョンの更新が含まれています。

このリリースは、Common Vulnerabilities and Exposures (CVE) への対応を含んでおり、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。

Red Hat が提供する Kiali Operator の最新バージョンは、サポートされているすべてのバージョンの Red Hat OpenShift Service Mesh で使用できます。Service Mesh のバージョンは、**ServiceMeshControlPlane** リソースを使用して指定されます。Service Mesh のバージョンにより、互換性のあるバージョンの Kiali が自動的に確保されます。

1.2.9.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.20.8
Envoy プロキシ	1.28.7
Kiali Server	1.73.17

1.2.10. Red Hat OpenShift Service Mesh バージョン 2.5.7

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.4 に含まれており、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。このリリースでは、Common Vulnerabilities and Exposures (CVE) に対応しています。

1.2.10.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.18.7
Envoy プロキシ	1.26.8
Kiali Server	1.73.17

1.2.11. Red Hat OpenShift Service Mesh バージョン 2.4.13

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.4 に含まれており、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。このリリースでは、Common Vulnerabilities and Exposures (CVE) に対応しています。

1.2.11.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.16.7
Envoy プロキシ	1.24.12
Kiali Server	1.65.18

1.2.12. Red Hat OpenShift Service Mesh バージョン 2.6.3

この Red Hat OpenShift Service Mesh リリースでは、Red Hat OpenShift Service Mesh Operator バージョンが 2.6.3 に更新され、2.6.3、2.5.6、2.4.12 の **ServiceMeshControlPlane** リソースバージョンの更新が含まれています。

このリリースは、Common Vulnerabilities and Exposures (CVE) への対応を含んでおり、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。

Red Hat が提供する Kiali Operator の最新バージョンは、サポートされているすべてのバージョンの Red Hat OpenShift Service Mesh で使用できます。Service Mesh のバージョンは、**ServiceMeshControlPlane** リソースを使用して指定されます。Service Mesh のバージョンにより、互換性のあるバージョンの Kiali が自動的に確保されます。

1.2.12.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.20.8
Envoy プロキシ	1.28.7
Kiali Server	1.73.16

1.2.13. Red Hat OpenShift Service Mesh バージョン 2.5.6

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.3 に含まれ、Common Vulnerabilities and Exposures (CVE) に対処し、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。

1.2.13.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.18.7
Envoy プロキシ	1.26.8
Kiali Server	1.73.16

1.2.14. Red Hat OpenShift Service Mesh バージョン 2.4.12

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.3 に含まれ、Common Vulnerabilities and Exposures (CVE) に対処し、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。

1.2.14.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.16.7
Envoy プロキシ	1.24.12
Kiali Server	1.65.17

1.2.15. Red Hat OpenShift Service Mesh バージョン 2.6.2

この Red Hat OpenShift Service Mesh リリースでは、Red Hat OpenShift Service Mesh Operator バージョンが 2.6.2 に更新され、2.6.2、2.5.5、2.4.11 の **ServiceMeshControlPlane** リソースバージョンの更新が含まれています。

このリリースは、Common Vulnerabilities and Exposures (CVE) への対応とバグ修正を含んでおり、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。

Red Hat が提供する Kiali Operator の最新バージョンは、サポートされているすべてのバージョンの Red Hat OpenShift Service Mesh で使用できます。Service Mesh のバージョンは、**ServiceMeshControlPlane** リソースを使用して指定されます。Service Mesh のバージョンにより、互換性のあるバージョンの Kiali が自動的に確保されます。

1.2.15.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.20.8
Envoy プロキシ	1.28.7
Kiali Server	1.73.15

1.2.15.2. 新機能

- `cert-manager Operator for Red Hat OpenShift` が、IBM Power、IBM Z、IBM® LinuxONE でサポートされるようになりました。

1.2.15.3. 修正された問題

- [OSSM-8099](#) 以前は、エンドポイントがドレインフェーズにある場合に、永続セッションラベルのサポートで問題が発生していました。現在は、ステートフルヘッダーセッションのドレインエンドポイントを処理する方法があります。
- [OSSM-8001](#) 以前は、Pod で `runAsUser` と `runAsGroup` が同じ値に設定されていた場合、プロキシ GID がコンテナの GID と一致するように誤って設定され、Istio CNI によって適用された iptables ルールでトラフィック傍受の問題が発生していました。これで、コンテナは `runAsUser` と `runAsGroup` に同じ値を持つことができ、iptables ルールが正しく適用されるようになりました。
- [OSSM-8074](#) 以前は、Service Mesh に数値のみの namespace (例: **12345**) があった場合、Kiali Operator は Kiali サーバーのインストールに失敗していました。現在は、数字のみを含む namespace が正しく機能します。

1.2.16. Red Hat OpenShift Service Mesh バージョン 2.5.5

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.2 に含まれ、Common Vulnerabilities and Exposures (CVE) に対処し、OpenShift Container Platform 4.14 以降でサポートされています。

1.2.16.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.18.7
Envoy プロキシ	1.26.8
Kiali Server	1.73.15

1.2.16.2. 修正された問題

- [OSSM-8001](#) 以前は、Pod で `runAsUser` および `runAsGroup` パラメーターが同じ値に設定されていた場合、プロキシ GID がコンテナの GID と一致するように誤って設定され、Istio CNI によって適用された iptables ルールでトラフィック傍受の問題が発生していました。これで、コンテナは `runAsUser` パラメーターと `runAsGroup` パラメーターに同じ値を持つことができ、iptables ルールが正しく適用されるようになりました。
- [OSSM-8074](#) 以前は、Service Mesh に数値のみの namespace (例: **12345**) があった場合、Red Hat が提供する Kiali Operator は Kiali Server のインストールに失敗していました。現在は、数字のみを含む namespace が正しく機能します。

1.2.17. Red Hat OpenShift Service Mesh バージョン 2.4.11

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.2 に含まれ、Common Vulnerabilities and Exposures (CVE) に対処し、OpenShift Container Platform 4.14 以降でサポートされています。

1.2.17.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.16.7
Envoy プロキシ	1.24.12
Kiali Server	1.65.16

1.2.17.2. 修正された問題

- [OSSM-8001](#) 以前は、Pod で **runAsUser** および **runAsGroup** パラメーターが同じ値に設定されていた場合、プロキシの GID がコンテナの GID と一致するように誤って設定され、Istio CNI によって適用された iptables ルールでトラフィック傍受の問題が発生していました。これで、コンテナは **runAsUser** パラメーターと **runAsGroup** パラメーターに同じ値を持つことができ、iptables ルールが正しく適用されるようになりました。
- [OSSM-8074](#) 以前は、Service Mesh に数値のみの namespace (例: **12345**) があった場合、Red Hat が提供する Kiali Operator は Kiali Server のインストールに失敗していました。現在は、数字のみを含む namespace が正しく機能します。

1.2.18. Red Hat OpenShift Service Mesh バージョン 2.6.1

この Red Hat OpenShift Service Mesh リリースでは、Red Hat OpenShift Service Mesh Operator バージョンが 2.6.1 に更新され、2.6.1、2.5.4、2.4.10 の **ServiceMeshControlPlane** リソースバージョンの更新が含まれています。このリリースは、Common Vulnerabilities and Exposures (CVE) への対応とバグ修正を含んでおり、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。

Red Hat が提供する Kiali Operator の最新バージョンは、サポートされているすべてのバージョンの Red Hat OpenShift Service Mesh で使用できます。Service Mesh のバージョンは、**ServiceMeshControlPlane** リソースを使用して指定されます。Service Mesh のバージョンにより、互換性のあるバージョンの Kiali が自動的に確保されます。

1.2.18.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.20.8
Envoy プロキシ	1.28.5
Kiali Server	1.73.14

1.2.18.2. 修正された問題

- [OSSM-6766](#) 以前は、ユーザーが namespace を更新するか (インジェクションの有効化または無効化など)、Istio オブジェクト (トラフィックポリシーなど) を作成しようとする、OpenShift Service Mesh Console (OSSMC) プラグインが失敗していました。現在は、ユーザーが namespace を更新しても、Istio オブジェクトを作成しても、OpenShift Service Mesh Console (OSSMC) プラグインは失敗しません。

1.2.19. Red Hat OpenShift Service Mesh バージョン 2.5.4

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.1 に含まれ、Common Vulnerabilities and Exposures (CVE) に対処し、OpenShift Container Platform 4.14 以降でサポートされています。

1.2.19.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.18.7
Envoy プロキシ	1.26.8
Kiali Server	1.73.14

1.2.20. Red Hat OpenShift Service Mesh バージョン 2.4.10

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.1 に含まれ、Common Vulnerabilities and Exposures (CVE) に対処し、OpenShift Container Platform 4.14 以降でサポートされています。

1.2.20.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.16.7
Envoy プロキシ	1.24.12
Kiali Server	1.65.15

1.2.21. Red Hat OpenShift Service Mesh バージョン 2.6.0

この Red Hat OpenShift Service Mesh リリースでは、Red Hat OpenShift Service Mesh Operator バージョンが 2.6.0 に更新され、2.6.0、2.5.3、2.4.9 の **ServiceMeshControlPlane** リソースバージョンの更新が含まれています。このリリースは、新機能と Common Vulnerabilities and Exposures (CVE) への対応を含んでおり、Red Hat OpenShift Service on AWS 4.14 以降でサポートされています。

このリリースで、Red Hat OpenShift Service Mesh バージョン 2.3 のメンテナンスサポートは終了します。Service Mesh バージョン 2.3 を使用している場合は、サポートされているバージョンに更新する必要があります。



重要

Red Hat OpenShift Service Mesh は FIPS 用に設計されています。Service Mesh は、x86_64、ppc64le、および s390x アーキテクチャーでの FIPS 140-2/140-3 検証のために NIST に提出された RHEL 暗号化ライブラリーを使用します。NIST の検証プログラムの詳細は、[Cryptographic Module Validation Program](#) を参照してください。検証のために提出された RHEL 暗号化ライブラリーの個々のバージョンの最新の NIST ステータスは、[Compliance Activities and Government Standards](#) を参照してください。

1.2.21.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.20.8
Envoy プロキシ	1.28.5
Kiali	1.73.9

1.2.21.2. Istio 1.20 サポート

Service Mesh 2.6 は Istio 1.20 をベースとしており、次のような新機能と製品の機能強化が提供されています。

- ネイティブサイドカーは Red Hat OpenShift Service on AWS 4.16 以降でサポートされています。

ServiceMeshControlPlane リソースの例

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

- Istio 1.20 のトラフィックミラーリングでは、複数の宛先がサポートされるようになりました。この機能により、さまざまなエンドポイントへのトラフィックのミラーリングが可能になり、異なるサービスバージョンや設定にわたる同時監視が可能になります。

Red Hat OpenShift Service Mesh は多くの Istio 1.20 機能をサポートしていますが、次の例外に注意してください。

- Ambient mesh はサポートされていません
- Istio の QuickAssist Technology (QAT) PrivateKeyProvider はサポートされていません

1.2.21.3. Istio および Kiali バンドルイメージ名の変更

このリリースでは、Istio バンドルイメージ名と Kiali バンドルイメージ名が更新され、Red Hat の命名規則により適合するようになりました。

- Istio バンドルイメージ名: **openshift-service-mesh/istio-operator-bundle**
- Kiali バンドルイメージ名: **openshift-service-mesh/kiali-operator-bundle**

1.2.21.4. Red Hat OpenShift 分散トレーシングプラットフォームと Red Hat build of OpenTelemetry との統合

このリリースでは、トレース拡張プロバイダー Red Hat OpenShift 分散トレーシング Platform と Red Hat build of OpenTelemetry の一般公開された統合が導入されています。

ServiceMeshControlPlane リソースの **spec.meshConfig.extensionProviders** 仕様に名前付き要素と **opentelemetry** プロバイダーを追加することで、トレースデータを Red Hat OpenShift 分散トレーシング Platform に公開できます。次に、テレメトリカスタムリソースは、トレーススパンを収集して OpenTelemetry Collector エンドポイントに送信するように Istio プロキシを設定します。

メッシュ namespace に Red Hat build of OpenTelemetry を作成し、トレースデータをトレースプラットフォームバックエンドサービスに送信するように設定できます。

1.2.21.5. Red Hat OpenShift 分散トレーシング Platform (Jaeger) のデフォルト設定の変更

このリリースでは、**ServiceMeshControlPlane** リソースの新しいインスタンスに対して、Red Hat OpenShift 分散トレーシング Platform (Jaeger) がデフォルトで無効になります。

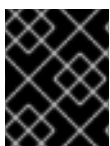
ServiceMeshControlPlane リソースの既存のインスタンスを Red Hat OpenShift Service Mesh バージョン 2.6 に更新すると、分散トレーシング Platform (Jaeger) はデフォルトで有効のままになります。

Red Hat OpenShift Service Mesh 2.6 は、Red Hat OpenShift 分散トレーシング Platform (Jaeger) と OpenShift Elasticsearch Operator のサポートが含まれる最後のリリースです。分散トレーシング Platform (Jaeger) と OpenShift Elasticsearch Operator は両方とも次のリリースで削除されます。現在、分散トレーシング Platform (Jaeger) と OpenShift Elasticsearch Operator を使用している場合は、Red Hat OpenShift 分散トレーシング Platform と Red Hat build of OpenTelemetry に切り替える必要があります。

1.2.21.6. Gateway API の使用は、Red Hat OpenShift Service Mesh のクラスター全体のデプロイメントで一般提供となりました。

このリリースにより、Kubernetes Gateway API バージョン 1.0.0 と Red Hat OpenShift Service Mesh 2.6 の使用が一般提供となりました。この API の使用は Red Hat OpenShift Service Mesh に限定されます。Gateway API カスタムリソース定義 (CRD) はサポートされていません。

クラスター全体モードが有効になっている場合 (**spec.mode: ClusterWide**) は、Gateway API がデフォルトで有効になりました。クラスターにカスタムリソース定義 (CRD) がインストールされていない場合でも有効にできます。



重要

マルチテナントメッシュデプロイメント用のゲートウェイ API は現在もテクノロジープレビュー段階です。

使用している OpenShift Service Mesh バージョンとともにどの Gateway API バージョンをインストールする必要があるかを判断するには、次の表を参照してください。

Service Mesh のバージョン	Istio バージョン	ゲートウェイ API バージョン	注記
2.6	1.20.x	1.0.0	該当なし
2.5.x	1.18.x	0.6.2	v0.6.2 には ReferenceGrand がないため、実験ブランチを使用します。
2.4.x	1.16.x	0.5.1	マルチテナントメッシュデプロイメントの場合、ゲートウェイ API CRD がすべて存在する必要があります。実験的なブランチを使用します。

PILOT_ENABLE_GATEWAY_API を **false** に設定することで、この機能を無効にできます。

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

1.2.21.7. 修正された問題

- [OSSM-6754](#) 以前は、Red Hat OpenShift Service on AWS 4.15 で、ユーザーが **Service details** ページに移動し、**Service Mesh** タブをクリックしてページを更新すると、アクティブなタブがデフォルトの **Details** タブであったにもかかわらず、**Service Mesh details** ページはサービスメッシュのコンテンツ情報のままになりました。更新後、ユーザーは問題なく **Service details** ページのさまざまなタブ間を移動できるようになりました。
- [OSSM-2101](#) 以前は、Istio Operator は、不要になった **istio-cni-node** DaemonSet やその他の CNI リソースを削除しませんでした。現在、Operator をアップグレードした後、クラスターに少なくとも1つの SMCP がインストールされている場合、Operator はこの SMCP を調整し、未使用の CNI インストール (v2.0 などの非常に古い CNI バージョンも含む) をすべて削除します。

1.2.21.8. Kiali の既知の問題

- [OSSM-6099](#) IPv6 クラスターで OpenShift Service Mesh Console (OSSMC) プラグインのインストールが失敗する。

回避策: IPv4 クラスターに OSSMC プラグインをインストールします。

1.2.22. Red Hat OpenShift Service Mesh バージョン 2.5.3

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.0 に含まれ、Common Vulnerabilities and Exposures (CVE) に対処し、OpenShift Container Platform 4.12 以降でサポートされています。

1.2.22.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.18.7
Envoy プロキシ	1.26.8
Kiali	1.73.9

1.2.23. Red Hat OpenShift Service Mesh バージョン 2.4.9

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.6.0 に含まれ、Common Vulnerabilities and Exposures (CVE) に対処し、OpenShift Container Platform 4.12 以降でサポートされています。

1.2.23.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.16.7
Envoy プロキシ	1.24.12
Kiali	1.65.11

1.2.24. Red Hat OpenShift Service Mesh バージョン 2.5.2

この Red Hat OpenShift Service Mesh リリースでは、Red Hat OpenShift Service Mesh Operator バージョンが 2.5.2 に更新され、2.5.2、2.4.8、2.3.12 の **ServiceMeshControlPlane** リソースバージョンの更新が含まれています。このリリースは、Common Vulnerabilities and Exposures (CVE) への対応とバグ修正を含んでおり、Red Hat OpenShift Service on AWS 4.12 以降でサポートされています。

1.2.24.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.18.7

コンポーネント	バージョン
Envoy プロキシ	1.26.8
Kiali	1.73.8

1.2.24.2. 修正された問題

- [OSSM-6331](#): 以前は、**smcp.general.logging.componentLevels** 仕様は無効な **LogLevel** 値を受け入れ、その場合でも **ServiceMeshControlPlane** リソースが作成されていました。現在、無効な値が使用されるとターミナルにエラーメッセージが表示され、コントロールプレーンは作成されません。
- [OSSM-6290](#) 以前は、**Istio Config** リストページの **Project** フィルタードロップダウンが正しく機能していませんでした。ドロップダウンメニューで特定のプロジェクトを選択した場合でも、すべての namespace のすべての **istio config** 項目が表示されていました。これで、フィルタードロップダウンで選択したプロジェクトに属する **istio config** 項目のみが表示されます。
- [OSSM-6298](#): 以前は、OpenShift Service Mesh Console (OSSMC) プラグイン内のアイテム参照をクリックすると、目的のページが開く前にコンソールが複数のリダイレクトを実行することがありました。その結果、コンソールで開いた前のページに戻ると、Web ブラウザーで間違ったページが開かれていました。現在、これらのリダイレクトは発生せず、Web ブラウザーで **Back** をクリックすると正しいページが開きます。
- [OSSM-6299](#): 以前は、Red Hat OpenShift Service on AWS 4.15 でトラフィックグラフ内の任意のノードメニューの **Node graph** メニューオプションをクリックしても、ノードグラフが表示されませんでした。代わりに、ページが更新されて同じトラフィックグラフが表示されていました。現在は、**Node graph** メニューオプションをクリックするとノードグラフが正しく表示されます。
- [OSSM-6267](#) 以前は、Red Hat OpenShift Service Mesh 2.5 Grafana でデータソースを設定すると、データクエリー認証エラーが発生し、ユーザーは Istio サービスおよびワークロードダッシュボードでデータを表示できませんでした。現在は、既存の 2.5 SMCP をバージョン 2.5.2 以降にアップグレードすることで、Grafana エラーは解決します。

1.2.25. Red Hat OpenShift Service Mesh バージョン 2.4.8

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.5.2 に含まれています。Common Vulnerabilities and Exposures (CVE) への対応とバグ修正を含んでおり、OpenShift Container Platform 4.12 以降でサポートされています。

Red Hat OpenShift Service Mesh Operator の最新バージョンは、サポートされているすべてのバージョンの Service Mesh で使用できます。Service Mesh のバージョンは、**ServiceMeshControlPlane** を使用して指定されます。

1.2.25.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.16.7

コンポーネント	バージョン
Envoy プロキシ	1.24.12
Kiali	1.65.11

1.2.26. Red Hat OpenShift Service Mesh バージョン 2.3.12

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.5.2 に含まれています。Common Vulnerabilities and Exposures (CVE) への対応とバグ修正を含んでおり、OpenShift Container Platform 4.12 以降でサポートされています。

Red Hat OpenShift Service Mesh Operator の最新バージョンは、サポートされているすべてのバージョンの Service Mesh で使用できます。Service Mesh のバージョンは、**ServiceMeshControlPlane** リソースを使用して指定します。

1.2.26.1. コンポーネントの更新

コンポーネント	バージョン
Istio	1.14.5
Envoy プロキシ	1.22.11
Kiali	1.57.14

1.2.27. 以前のリリース

これらの以前のリリースには、機能と改良が追加されました。

1.2.27.1. Red Hat OpenShift Service Mesh バージョン 2.5.1 の新機能

この Red Hat OpenShift Service Mesh リリースでは、Red Hat OpenShift Service Mesh Operator バージョンが 2.5.1 に更新され、2.5.1、2.4.7、2.3.11 の **ServiceMeshControlPlane** リソースバージョンの更新が含まれています。

このリリースは、Common Vulnerabilities and Exposures (CVE) への対応とバグ修正を含んでおり、Red Hat OpenShift Service on AWS 4.12 以降でサポートされています。

1.2.27.1.1. Red Hat OpenShift Service Mesh バージョン 2.5.1 の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	1.18.7
Envoy プロキシ	1.26.8
Kiali	1.73.7

1.2.27.2. Red Hat OpenShift Service Mesh バージョン 2.5 の新機能

この Red Hat OpenShift Service Mesh リリースでは、Red Hat OpenShift Service Mesh Operator バージョンが 2.5.0 に更新され、2.5.0、2.4.6、2.3.10 の **ServiceMeshControlPlane** リソースバージョンの更新が含まれています。

このリリースは、新機能と Common Vulnerabilities and Exposures (CVE) への対応を含んでおり、Red Hat OpenShift Service on AWS 4.12 以降でサポートされています。

このリリースで、OpenShift Service Mesh バージョン 2.2 のメンテナンスサポートが終了します。OpenShift Service Mesh バージョン 2.2 を使用している場合は、サポートされているバージョンに更新する必要があります。

1.2.27.2.1. Red Hat OpenShift Service Mesh バージョン 2.5 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.18.7
Envoy プロキシ	1.26.8
Kiali	1.73.4

1.2.27.2.2. Istio 1.18 サポート

Service Mesh 2.5 は Istio 1.18 に基づいており、新機能と製品の機能強化をもたらします。Red Hat OpenShift Service Mesh は多くの Istio 1.18 機能をサポートしていますが、次の例外に注意する必要があります。

- Ambient mesh はサポートされていません
- Istio の QuickAssist Technology (QAT) PrivateKeyProvider はサポートされていません

1.2.27.2.3. クラスター全体のメッシュ移行

このリリースでは、マルチテナントメッシュからクラスター全体のメッシュへの移行に関するドキュメントが追加されています。詳細は、以下のドキュメントを参照してください。

- 「クラスター全体のメッシュへの移行について」
- 「クラスター全体のメッシュからの namespace の除外」
- 「クラスター全体のメッシュにおいてサイドカーインジェクションを受け取る namespace の定義」
- 「クラスター全体のメッシュからの Pod の個別除外」

1.2.27.2.4. ARM ベースのクラスターに対する Red Hat OpenShift Service Mesh Operator

このリリースでは、ARM ベースのクラスター上で Red Hat OpenShift Service Mesh Operator が一般提供機能として提供されます。

1.2.27.2.5. Red Hat OpenShift 分散トレーシングプラットフォーム (Tempo) スタックとの統合

このリリースでは、一般に利用可能なトレースエクステンションプロバイダーの統合が導入されています。名前付き要素と **zipkin** プロバイダーを **spec.meshConfig.extensionProviders** 仕様に追加することで、トレースデータを Red Hat OpenShift 分散トレーシング Platform (Tempo) スタックに公開できます。次に、テレメトリカスタムリソースは、トレーススパンを収集し、Tempo ディストリビュータサービスエンドポイントに送信するように Istio プロキシを設定します。



注記

Red Hat OpenShift 分散トレーシング Platform (Tempo) スタックは、IBM Z ではサポートされていません。

1.2.27.2.6. OpenShift Service Mesh Console プラグイン

このリリースでは、OpenShift Service Mesh Console (OSSMC) プラグインの一般提供バージョンが導入されています。

OSSMC プラグインは、サービスメッシュへの可視性を提供する OpenShift コンソールのエクステンションです。OSSMC プラグインをインストールすると、Web コンソールのナビゲーションペインで新しいサービスメッシュメニューオプションが使用可能になり、既存のワークロードおよびサービスコンソールページを強化する新しいサービスメッシュタブも使用できるようになります。

OSSMC プラグインの機能は、スタンドアロン Kiali コンソールの機能とよく似ています。OSSMC プラグインは Kiali コンソールに代わるものではありません。OSSMC プラグインをインストールした後も、スタンドアロンの Kiali コンソールにアクセスできます。

1.2.27.2.7. Istio OpenShift Routing (IOR) のデフォルト設定の変更

Istio OpenShift Routing (IOR) のデフォルト設定が変更されました。今回のリリース以降、**ServiceMeshControlPlane** リソースの新しいインスタンスでは自動ルートがデフォルトで無効になっています。

ServiceMeshControlPlane リソースの新しいインスタンスの場合、**ServiceMeshControlPlane** リソースの **gateways.openshiftRoute** 仕様が **enabled** フィールドを **true** に設定することで、自動ルートを使用できます。

ServiceMeshControlPlane リソースの例

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

ServiceMeshControlPlane リソースの既存のインスタンスを Red Hat OpenShift Service Mesh バージョン 2.5 に更新すると、自動ルートはデフォルトで有効なままになります。

1.2.27.2.8. Istio プロキシ同時実行設定の強化

networking.istio API の **concurrency** パラメーターは、Istio プロキシが実行するワーカースレッドの数を設定します。

デプロイメント間の一貫性を確保するために、Istio はプロキシコンテナに割り当てられた CPU 制限に基づいて **concurrency** パラメーターを設定するようになりました。たとえば、制限が 2500m の場合、**concurrency** パラメーターは **3** に設定されます。**concurrency** パラメーターを別の値に設定す

ると、Istio は CPU 制限を使用する代わりに、その値を使用してプロキシーが実行するスレッドの数を設定します。

以前のリリースでは、このパラメーターのデフォルト設定は **2** でした。

1.2.27.2.9. ゲートウェイ API CRD バージョン



重要

Red Hat OpenShift Service on AWS Gateway API のサポートはテクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

Gateway API カスタムリソース定義 (CRD) の新しいバージョンが利用可能になりました。使用している OpenShift Service Mesh バージョンとともにどの Gateway API バージョンをインストールする必要があるかを判断するには、次の表を参照してください。

Service Mesh のバージョン	Istio バージョン	ゲートウェイ API バージョン	注記
2.5.x	1.18.x	0.6.2	v0.6.2 には ReferenceGrant がいないため、実験的ブランチを使用します。
2.4.x	1.16.x	0.5.1	マルチテナントメッシュデプロイメントの場合、ゲートウェイ API CRD がすべて存在する必要があります。実験的なブランチを使用します。

1.2.27.3. Red Hat OpenShift Service Mesh バージョン 2.4.7 の新機能

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.5.1 に含まれています。Common Vulnerabilities and Exposures (CVE) への対応と修正を含んでおり、Red Hat OpenShift Service on AWS 4.12 以降でサポートされています。

1.2.27.3.1. Red Hat OpenShift Service Mesh バージョン 2.4.7 の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	1.16.7

コンポーネント	バージョン
Envoy プロキシ	1.24.12
Kiali	1.65.11

1.2.27.4. Red Hat OpenShift Service Mesh バージョン 2.4.6 の新機能

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.5.0 に含まれています。Common Vulnerabilities and Exposures (CVE) への対応とバグ修正を含んでおり、OpenShift Container Platform 4.12 以降でサポートされています。

1.2.27.4.1. Red Hat OpenShift Service Mesh バージョン 2.4.6 の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	1.16.7
Envoy プロキシ	1.24.12
Kiali	1.65.11

1.2.27.5. Red Hat OpenShift Service Mesh バージョン 2.4.5 の新機能

この Red Hat OpenShift Service Mesh リリースでは、Red Hat OpenShift Service Mesh Operator バージョンが 2.4.5 に更新され、2.4.5、2.3.9、2.2.12 の **ServiceMeshControlPlane** リソースバージョンの更新が含まれています。

このリリースは、Common Vulnerabilities and Exposures (CVE) への対応を含んでおり、Red Hat OpenShift Service on AWS 4.11 以降でサポートされています。

1.2.27.5.1. Red Hat OpenShift Service Mesh バージョン 2.4.5 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.16.7
Envoy プロキシ	1.24.12
Kiali	1.65.11

1.2.27.6. Red Hat OpenShift Service Mesh バージョン 2.4.4 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.11 以降でサポートされます。

1.2.27.6.1. Red Hat OpenShift Service Mesh バージョン 2.4.4 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.16.7
Envoy プロキシ	1.24.12
Jaeger	1.47.0
Kiali	1.65.10

1.2.27.7. Red Hat OpenShift Service Mesh バージョン 2.4.3 の新機能

- **envoyExtAuthzGrpc** フィールドが追加されました。これは、gRPC API を使用して外部承認プロバイダーを設定するために使用されます。
- Common Vulnerabilities and Exposures (CVE) が解決されました。
- このリリースは、Red Hat OpenShift Service on AWS 4.10 以降のバージョンでサポートされています。

1.2.27.7.1. Red Hat OpenShift Service Mesh バージョン 2.4.3 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.16.7
Envoy プロキシ	1.24.10
Jaeger	1.42.0
Kiali	1.65.8

1.2.27.7.2. 外部認可設定に対するリモートプロシージャコール (gRPC) API のサポート

今回の機能拡張により、**envoyExtAuthzGrpc** フィールドが追加され、gRPC API を使用して外部認可プロバイダーが設定されます。

1.2.27.8. Red Hat OpenShift Service Mesh バージョン 2.4.2 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.10 以降でサポートされます。

1.2.27.8.1. Red Hat OpenShift Service Mesh バージョン 2.4.2 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.16.7
Envoy プロキシ	1.24.10
Jaeger	1.42.0
Kiali	1.65.7

1.2.27.9. Red Hat OpenShift Service Mesh バージョン 2.4.1 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.10 以降でサポートされます。

1.2.27.9.1. Red Hat OpenShift Service Mesh バージョン 2.4.1 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.16.5
Envoy プロキシ	1.24.8
Jaeger	1.42.0
Kiali	1.65.7

1.2.27.10. Red Hat OpenShift Service Mesh バージョン 2.4 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.10 以降でサポートされます。

1.2.27.10.1. Red Hat OpenShift Service Mesh バージョン 2.4 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.16.5
Envoy プロキシ	1.24.8
Jaeger	1.42.0
Kiali	1.65.6

1.2.27.10.2. クラスター全体のデプロイメント

この機能拡張により、クラスター全体のデプロイメントの一般利用可能なバージョンが導入されます。クラスター全体のデプロイメントには、クラスター全体のリソースを監視する Service Mesh Control Plane が含まれます。コントロールプレーンは、すべての namespace で単一のクエリーを使用して、メッシュ設定に影響を与える各 Istio または Kubernetes リソースの種類を監視します。クラスター全体のデプロイメントでコントロールプレーンが実行するクエリーの数を減らすと、パフォーマンスが向上します。

1.2.27.10.3. ディスカバリーセクターのサポート

この機能強化により、**meshConfig.discoverySelectors** フィールドの一般利用可能なバージョンが導入され、これをクラスター全体のデプロイメントで使用して、Service Mesh コントロールプレーンが検出できるサービスを制限できます。

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

1.2.27.10.4. cert-manager istio-csr との統合

この更新により、Red Hat OpenShift Service Mesh は **cert-manager** コントローラーおよび **istio-csr** エージェントと統合されます。**cert-manager** は、証明書と証明書発行者を Kubernetes クラスター内のリソースタイプとして追加し、それらの証明書の取得、更新、使用のプロセスを簡素化します。**cert-manager** は、Istio の中間 CA 証明書を提供し、ローテーションします。**istio-csr** との統合により、ユーザーは Istio プロキシからの署名証明書要求を **cert-manager** に委任できます。**ServiceMeshControlPlane** v2.4 は **cert-manager** によって提供された CA 証明書を **cacerts** シークレットとして受け入れます。



注記

cert-manager および **istio-csr** との統合は、IBM Power®、IBM Z®、および IBM® LinuxONE ではサポートされていません。

1.2.27.10.5. 外部認証システムとの統合

この機能強化により、**AuthorizationPolicy** リソースの **action:CUSTOM** フィールドを使用して、Red Hat OpenShift Service Mesh を外部認可システムと統合する一般に利用可能な方法が導入されました。**envoyExtAuthzHttp** フィールドを使用して、アクセス制御を外部認証システムに委任します。

1.2.27.10.6. 外部 Prometheus インストールとの統合

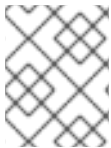
この機能拡張により、Prometheus 拡張プロバイダーの一般利用可能なバージョンが導入されます。**spec.meshConfig** 仕様で **extensionProviders** フィールドの値を **prometheus** に設定することで、Red Hat OpenShift Service on AWS モニタリングスタックまたはカスタム Prometheus インストー

ルにメトリクスを公開できます。テレメトリオブジェクトは、トラフィックメトリクスを収集するように Istio プロキシを設定します。Service Mesh は、Prometheus メトリクスの Telemetry API のみをサポートします。

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

1.2.27.10.7. シングルスタック IPv6 のサポート

この機能拡張により、シングルスタック IPv6 クラスターの一般利用可能なサポートが導入され、より広範囲の IP アドレスへのアクセスが提供されます。デュアルスタック IPv4 または IPv6 クラスターはサポートされていません。



注記

シングルスタック IPv6 サポートは、IBM Power®、IBM Z®、および IBM® LinuxONE では利用できません。

1.2.27.10.7.1. Red Hat OpenShift Service on AWS ゲートウェイ API の有効化

Red Hat OpenShift Service on AWS Gateway API を有効にするには、**ServiceMeshControlPlane** リソースの **techPreview.gatewayAPI** 仕様で、**enabled** フィールドの値を **true** に設定します。

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

以前は、ゲートウェイ API を有効にするために環境変数が使用されていました。

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

1.2.27.10.8. Istio 1.16 サポート

Service Mesh 2.4 は Istio 1.16 に基づいており、新機能と製品の機能強化をもたらします。多くの Istio 1.16 機能がサポートされていますが、次の例外に注意する必要があります。

- サイドカーの HBONE プロトコルは実験的な機能であり、サポートされていません。
- ARM64 アーキテクチャー上の Service Mesh はサポートされていません。

1.2.27.11. Red Hat OpenShift Service Mesh バージョン 2.3.11 の新機能

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.5.1 に含まれています。Common Vulnerabilities and Exposures (CVE) への対応と修正を含んでおり、Red Hat OpenShift Service on AWS 4.12 以降でサポートされています。

1.2.27.11.1. Red Hat OpenShift Service Mesh バージョン 2.3.11 の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	1.14.5
Envoy プロキシ	1.22.11
Kiali	1.57.14

1.2.27.12. Red Hat OpenShift Service Mesh バージョン 2.3.10 の新機能

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.5.0 に含まれています。Common Vulnerabilities and Exposures (CVE) への対応とバグ修正を含んでおり、OpenShift Container Platform 4.12 以降でサポートされています。

1.2.27.12.1. Red Hat OpenShift Service Mesh バージョン 2.3.10 の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	1.14.5
Envoy プロキシ	1.22.11
Kiali	1.57.14

1.2.27.13. Red Hat OpenShift Service Mesh バージョン 2.3.9 の新機能

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.4.5 に含まれています。Common Vulnerabilities and Exposures (CVE) への対応とバグ修正を含んでおり、OpenShift Container Platform 4.11 以降でサポートされています。

1.2.27.13.1. Red Hat OpenShift Service Mesh バージョン 2.3.9 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.14.5
Envoy プロキシ	1.22.11
Jaeger	1.47.0
Kiali	1.57.14

1.2.27.14. Red Hat OpenShift Service Mesh バージョン 2.3.8 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.11 以降でサポートされます。

1.2.27.14.1. Red Hat OpenShift Service Mesh バージョン 2.3.8 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	1.14.5
Envoy プロキシ	1.22.11
Jaeger	1.47.0
Kiali	1.57.13

1.2.27.15. Red Hat OpenShift Service Mesh バージョン 2.3.7 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.10 以降でサポートされます。

1.2.27.15.1. Red Hat OpenShift Service Mesh バージョン 2.3.7 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	1.14.6
Envoy プロキシ	1.22.11
Jaeger	1.42.0
Kiali	1.57.11

1.2.27.16. Red Hat OpenShift Service Mesh バージョン 2.3.6 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.10 以降でサポートされます。

1.2.27.16.1. Red Hat OpenShift Service Mesh バージョン 2.3.6 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	1.14.5
Envoy プロキシ	1.22.11
Jaeger	1.42.0
Kiali	1.57.10

1.2.27.17. Red Hat OpenShift Service Mesh バージョン 2.3.5 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.10 以降でサポートされます。

1.2.27.17.1. Red Hat OpenShift Service Mesh バージョン 2.3.5 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.14.5
Envoy プロキシ	1.22.9
Jaeger	1.42.0
Kiali	1.57.10

1.2.27.18. Red Hat OpenShift Service Mesh バージョン 2.3.4 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.10 以降でサポートされます。

1.2.27.18.1. Red Hat OpenShift Service Mesh バージョン 2.3.4 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.14.6

コンポーネント	バージョン
Envoy プロキシ	1.22.9
Jaeger	1.42.0
Kiali	1.57.9

1.2.27.19. Red Hat OpenShift Service Mesh バージョン 2.3.3 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.9 以降でサポートされます。

1.2.27.19.1. Red Hat OpenShift Service Mesh バージョン 2.3.3 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.14.5
Envoy プロキシ	1.22.9
Jaeger	1.42.0
Kiali	1.57.7

1.2.27.20. Red Hat OpenShift Service Mesh バージョン 2.3.2 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.9 以降でサポートされます。

1.2.27.20.1. Red Hat OpenShift Service Mesh バージョン 2.3.2 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.14.5
Envoy プロキシ	1.22.7
Jaeger	1.39
Kiali	1.57.6

1.2.27.21. Red Hat OpenShift Service Mesh バージョン 2.3.1 の新機能

このリリースの Red Hat OpenShift Service Mesh には、新機能の導入、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれ、Red Hat OpenShift Service on AWS 4.9 以降でサポートされます。

1.2.27.21.1. Red Hat OpenShift Service Mesh バージョン 2.3.1 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.14.5
Envoy プロキシ	1.22.4
Jaeger	1.39
Kiali	1.57.5

1.2.27.22. Red Hat OpenShift Service Mesh バージョン 2.3 の新機能

このリリースの Red Hat OpenShift Service Mesh には、新機能の導入、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれ、Red Hat OpenShift Service on AWS 4.9 以降でサポートされます。

1.2.27.22.1. Red Hat OpenShift Service Mesh バージョン 2.3 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.14.3
Envoy プロキシ	1.22.4
Jaeger	1.38
Kiali	1.57.3

1.2.27.22.2. 新しい Container Network Interface (CNI) DaemonSet コンテナと ConfigMap

openshift-operators namespace には、新しい istio CNI DaemonSet **istio-cni-node-v2-3**、新しい **ConfigMap** リソース、**istio-cni-config-v2-3** が含まれています。

Service Mesh Control Plane 2.3 にアップグレードすると、既存の **istio-cni-node** DaemonSet は変更されず、新しい **istio-cni-node-v2-3** DaemonSet が作成されます。

この名称変更は、以前のリリースや、以前のリリースを使用してデプロイされた Service Mesh Control Plane に関連付けられた **istio-cni-node** CNI DaemonSet には影響しません。

1.2.27.22.3. ゲートウェイ挿入のサポート

このリリースでは、ゲートウェイ挿入の一般利用可能なサポートが導入されています。ゲートウェイ設定は、サービスワークロードとともに実行するサイドカー Envoy プロキシではなく、メッシュのエッジで実行するスタンドアロン Envoy プロキシに適用されます。これにより、ゲートウェイオプション

のカスタマイズ機能が有効になります。ゲートウェイ挿入を使用する場合は、ゲートウェイプロキシを実行する namespace にリソース (**Service**、**Deployment**、**Role**、および **RoleBinding**) を作成する必要があります。

1.2.27.22.4. Istio 1.14 サポート

Service Mesh 2.3 は Istio 1.14 に基づいており、新機能と製品の機能強化をもたらします。多くの Istio 1.14 機能がサポートされていますが、次の例外に注意する必要があります。

- ProxyConfig API はサポートされていますが、image フィールドは例外です。
- SPIRE ランタイムはサポートされていない機能です。

1.2.27.22.4.1. クラスター全体のデプロイメントの設定

次の **ServiceMeshControlPlane** オブジェクトの例では、クラスター全体のデプロイを設定します。

クラスター全体のデプロイメント用に SMCP を作成する場合、ユーザーは **cluster-admin** ClusterRole に属している必要があります。SMCP がクラスター全体のデプロイメント用に設定されている場合は、それがクラスター内の唯一の SMCP である必要があります。コントロールプレーンモードをマルチテナントからクラスター全体 (またはクラスター全体からマルチテナント) に変更することはできません。マルチテナントコントロールプレーンがすでに存在する場合は、それを削除して、新しいコントロールプレーンを作成します。

この例では、クラスター全体のデプロイメント用に SMCP を設定します。

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

- ❶ Istiod が、個々の namespace を監視するのではなく、クラスターレベルでリソースを監視できるようにします。

さらに、SMMR もクラスター全体のデプロイメント用に設定する必要があります。この例では、クラスター全体のデプロイメント用に SMMR を設定します。

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

- ❶ 後で作成する namespace を含め、すべての namespace をメッシュに追加します。kube、openshift、kube-*、および openshift-* の namespace は、メッシュの一部ではありません。

1.2.27.23. Red Hat OpenShift Service Mesh バージョン 2.2.12 の新機能

この Red Hat OpenShift Service Mesh リリースは、Red Hat OpenShift Service Mesh Operator 2.4.5 に含まれています。Common Vulnerabilities and Exposures (CVE) への対応とバグ修正を含んでおり、OpenShift Container Platform 4.11 以降でサポートされています。

1.2.27.23.1. Red Hat OpenShift Service Mesh バージョン 2.2.12 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.12.9
Envoy プロキシ	1.20.8
Jaeger	1.47.0
Kiali	1.48.11

1.2.27.24. Red Hat OpenShift Service Mesh バージョン 2.2.11 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.11 以降でサポートされます。

1.2.27.24.1. Red Hat OpenShift Service Mesh バージョン 2.2.11 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.12.9
Envoy プロキシ	1.20.8
Jaeger	1.47.0
Kiali	1.48.10

1.2.27.25. Red Hat OpenShift Service Mesh バージョン 2.2.10 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.10 以降でサポートされます。

1.2.27.25.1. Red Hat OpenShift Service Mesh バージョン 2.2.10 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.12.9
Envoy プロキシ	1.20.8
Jaeger	1.42.0
Kiali	1.48.8

1.2.27.26. Red Hat OpenShift Service Mesh バージョン 2.2.9 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.10 以降でサポートされます。

1.2.27.26.1. Red Hat OpenShift Service Mesh バージョン 2.2.9 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.12.9
Envoy プロキシ	1.20.8
Jaeger	1.42.0
Kiali	1.48.7

1.2.27.27. Red Hat OpenShift Service Mesh バージョン 2.2.8 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.10 以降でサポートされます。

1.2.27.27.1. Red Hat OpenShift Service Mesh バージョン 2.2.8 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.12.9
Envoy プロキシ	1.20.8
Jaeger	1.42.0
Kiali	1.48.7

1.2.27.28. Red Hat OpenShift Service Mesh バージョン 2.2.7 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.10 以降でサポートされます。

1.2.27.28.1. Red Hat OpenShift Service Mesh バージョン 2.2.7 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	1.12.9
Envoy プロキシ	1.20.8
Jaeger	1.42.0
Kiali	1.48.6

1.2.27.29. Red Hat OpenShift Service Mesh バージョン 2.2.6 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.9 以降でサポートされます。

1.2.27.29.1. Red Hat OpenShift Service Mesh バージョン 2.2.6 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.12.9
Envoy プロキシ	1.20.8
Jaeger	1.39
Kiali	1.48.5

1.2.27.30. Red Hat OpenShift Service Mesh バージョン 2.2.5 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.9 以降でサポートされます。

1.2.27.30.1. Red Hat OpenShift Service Mesh バージョン 2.2.5 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.12.9

コンポーネント	バージョン
Envoy プロキシ	1.20.8
Jaeger	1.39
Kiali	1.48.3

1.2.27.31. Red Hat OpenShift Service Mesh バージョン 2.2.4 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.9 以降でサポートされます。

1.2.27.31.1. Red Hat OpenShift Service Mesh バージョン 2.2.4 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.12.9
Envoy プロキシ	1.20.8
Jaeger	1.36.14
Kiali	1.48.3

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

このリリースの Red Hat OpenShift Service Mesh は、共通脆弱性および漏洩 (CVE) に対処し、バグ修正を行っており、Red Hat OpenShift Service on AWS 4.9 以降でサポートされています。

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

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

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

このリリースの Red Hat OpenShift Service Mesh は、共通脆弱性および漏洩 (CVE) に対処し、バグ修正を行っており、Red Hat OpenShift Service on AWS 4.9 以降でサポートされています。

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

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

1.2.27.33.2. ルートラベルのコピー

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

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

このリリースの Red Hat OpenShift Service Mesh は、共通脆弱性および漏洩 (CVE) に対処し、バグ修正を行っており、Red Hat OpenShift Service on AWS 4.9 以降でサポートされています。

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

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

1.2.27.35. Red Hat OpenShift Service Mesh 2.2 の新機能

このリリースの Red Hat OpenShift Service Mesh は、新しい機能と拡張機能を追加し、Red Hat OpenShift Service on AWS 4.9 以降のバージョンでサポートされています。

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

コンポーネント	バージョン
Istio	1.12.7

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

1.2.27.35.2. WasmPlugin API

このリリースでは、**WasmPlugin** API のサポートが追加され、**ServiceMeshExtension** が非推奨になりました。

1.2.27.35.3. ROSA サポート

このリリースでは、マルチクラスターフェデレーションを含む Red Hat OpenShift on AWS (ROSA) の Service Mesh サポートが導入されています。

1.2.27.35.4. istio-node DaemonSet の名前変更

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

1.2.27.35.5. エンボイサイドカーネットワークの変更

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

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

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

1.2.27.35.7. Istio 1.12 サポート

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

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

1.2.27.35.7.1. Gateway API CRD のインストール

Gateway API CRD は、デフォルトでは OpenShift クラスターにプリインストールされていません。SMCP で Gateway API サポートを有効にする前に、CRD をインストールします。

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

1.2.27.35.7.2. Kubernetes Gateway API の有効化

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

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

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

1.2.27.35.7.3. 手動によるゲートウェイリソースへの既存ゲートウェイのリンク

Kubernetes API デプロイメントコントローラーが無効になっている場合は、ingress ゲートウェイを手動でデプロイし、作成されたゲートウェイリソースにリンクする必要があります。

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

1.2.27.36. Red Hat OpenShift Service Mesh 2.1.6 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.9 以降でサポートされます。

1.2.27.36.1. Red Hat OpenShift Service Mesh バージョン 2.1.6 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.9.9

コンポーネント	バージョン
Envoy プロキシ	1.17.5
Jaeger	1.36
Kiali	1.36.16

1.2.27.37. Red Hat OpenShift Service Mesh 2.1.5.2 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.9 以降でサポートされます。

1.2.27.37.1. Red Hat OpenShift Service Mesh バージョン 2.1.5.2 に含まれるコンポーネントのバージョン

コンポーネント	バージョン
Istio	1.9.9
Envoy プロキシ	1.17.5
Jaeger	1.36
Kiali	1.24.17

1.2.27.38. Red Hat OpenShift Service Mesh 2.1.5.1 の新機能

このリリースの Red Hat OpenShift Service Mesh は、共通脆弱性および漏洩 (CVE) に対処し、バグ修正を行っており、Red Hat OpenShift Service on AWS 4.9 以降でサポートされています。

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

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

1.2.27.39. Red Hat OpenShift Service Mesh 2.1.5 の新機能

このリリースの Red Hat OpenShift Service Mesh は、共通脆弱性および漏洩 (CVE) に対処し、バグ修正を行っており、Red Hat OpenShift Service on AWS 4.9 以降でサポートされています。

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

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

1.2.27.40. Red Hat OpenShift Service Mesh 2.1.4 の新機能

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

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

コンポーネント	バージョン
Istio	1.9.9
Envoy プロキシ	1.17.1
Jaeger	1.30.2
Kiali	1.36.12-1

1.2.27.41. Red Hat OpenShift Service Mesh 2.1.3 の新機能

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

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

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

コンポーネント	バージョン
Kiali	1.36.10-2

1.2.27.42. Red Hat OpenShift Service Mesh 2.1.2.1 の新機能

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

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

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

1.2.27.43. Red Hat OpenShift Service Mesh 2.1.2 の新機能

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

このリリースでは、Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator が、デフォルトで **openshift-distributed-tracing** 名前空間にインストールされるようになりました。以前のリリースでは、デフォルトのインストールは **openshift-operator** namespace にありました。

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

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

1.2.27.44. Red Hat OpenShift Service Mesh 2.1.1 の新機能

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

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

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

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

1.2.27.44.2. ネットワークポリシーの無効化

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

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



注記

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

手順

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

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
```



```
security:
  trust:
    manageNetworkPolicy: false
```

6. **Save** をクリックします。

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

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

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

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

1.2.27.45.2. Service Mesh のフェデレーション

Service Mesh をフェデレーションできるように新規のカスタムリソース定義 (CRD) が追加されました。Service Mesh は、同じクラスター内または異なる OpenShift クラスター間でフェデレーションを行うことができます。これらの新規リソースには以下が含まれます。

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

Service Mesh フェデレーションは、Red Hat OpenShift Service on AWS (ROSA) または OpenShift Dedicated (OSD) 上のクラスター間ではサポートされていません。

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

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

1.2.27.45.4. Service Mesh WebAssembly (WASM) 拡張

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

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

1.2.27.45.5. 3scale WebAssembly Adapter (WASM)

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

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

1.2.27.45.6. Istio 1.9 サポート

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

- 仮想マシンの統合はまだサポートされていません。
- Kubernetes Gateway API はまだサポートされていません。
- WebAssembly HTTP フィルターのリモートフェッチおよびロードはサポートされていません。
- Kubernetes CSR API を使用したカスタム CA 統合はまだサポートされていません。
- トラフィック監視要求の分類機能はテクノロジープレビュー機能です。
- Authorization ポリシーの CUSTOM アクションによる外部承認システムとの統合はテクノロジープレビュー機能です。

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

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

1.2.27.45.8. Kiali の更新

Kiali 1.36 には、以下の機能と拡張機能が含まれています。

- Service Mesh のトラブルシューティング機能
 - コントロールプレーンおよびゲートウェイの監視
 - プロキシの同期ステータス
 - Envoy 設定ビュー
 - Envoy プロキシおよびアプリケーションログのインターリーブを示す統合ビュー

- フェデレーションされた Service Mesh ビューをサポートする namespace およびクラスターボックス
- 新しい検証、ウィザード、および分散トレースの機能拡張

1.2.27.46. Red Hat OpenShift Service Mesh 2.0.11.1 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.9 以降でサポートされます。

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

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

1.2.27.47. Red Hat OpenShift Service Mesh 2.0.11 の新機能

このリリースの Red Hat OpenShift Service Mesh には、CVE (Common Vulnerabilities and Exposures) への対応、バグ修正が含まれています。このリリースは、Red Hat OpenShift Service on AWS 4.9 以降でサポートされます。

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

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

1.2.27.48. Red Hat OpenShift Service Mesh 2.0.10 の新機能

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

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

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

1.2.27.49. Red Hat OpenShift Service Mesh 2.0.9 の新機能

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

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

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

1.2.27.50. Red Hat OpenShift Service Mesh 2.0.8 の新機能

このリリースの Red Hat OpenShift Service Mesh では、バグ修正に対応しています。

1.2.27.51. Red Hat OpenShift Service Mesh 2.0.7.1 の新機能

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

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

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

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

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

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



警告

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

ServiceMeshControlPlane の変更例

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

1.2.27.51.2. 認証ポリシーに必要な更新

Istio はホスト名自体とすべての一致するポートの両方にホスト名を生成します。たとえば、"httpbin.foo" のホストの仮想サービスまたはゲートウェイは、"httpbin.foo and httpbin.foo:*" に一致する設定を生成します。ただし、完全一致許可ポリシーは、**hosts** または **notHosts** フィールドに指定された完全一致文字列にのみ一致します。

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

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

接頭辞一致を使用した最初の AuthorizationPolicy の例

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

```

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

```

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

```

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

```

1.2.27.52. Red Hat OpenShift Service Mesh 2.0.7 の新機能

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

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

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

1.2.27.54. Red Hat OpenShift Service Mesh 2.0.6 の新機能

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

1.2.27.55. Red Hat OpenShift Service Mesh 2.0.5 の新機能

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

1.2.27.56. Red Hat OpenShift Service Mesh 2.0.4 の新機能

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



重要

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

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

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

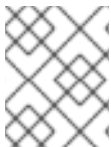
たとえば、Istio クラスター管理者が、パス **/admin** での要求を拒否する認証 DENY ポリシーを定義すると仮定します。URL パス **//admin** に送信される要求は、認証ポリシーには拒否されません。

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

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

以下の場合、クラスターはこの脆弱性の影響を受けません。

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



注記

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

1.2.27.56.2. パスの正規化設定の更新

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

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

表1.1 正規化スキーム

オプション	説明	例	注記
NONE	正規化は行われません。Envoy が受信する内容はそのまますべて、どのバックエンドサービスにも完全に転送されます。	../%2fa../b は認証ポリシーによって評価され、サービスに送信されます。	この設定は CVE-2021-31920 に対して脆弱です。

オプション	説明	例	注記
BASE	現時点で、これは Istio の デフォルト インストールで使用されるオプションです。これにより、Envoy プロキシで normalize_path オプションが適用されます。これは、追加の正規化において RFC 3986 に従い、バックスラッシュをフォワードスラッシュに変換します。	/a/./b は /b に正規化されます。 \da は /da に正規化されます。	この設定は CVE-2021-31920 に対して脆弱です。
MERGE_SLASHES	スラッシュは BASE の正規化後にマージされます。	/a//b は /a/b に正規化されます。	この設定に対して更新を行い、CVE-2021-31920 のリスクを軽減します。
DECODE_AND_MERGE_SLASHES	デフォルトですべてのトラフィックを許可する場合の最も厳密な設定です。この設定の場合は、認証ポリシーのルートの詳細にテストする必要があります。パーセントでエンコードされたスラッシュおよびバックスラッシュ文字 (%2F 、 %2f 、 %5C および %5c) は、 MERGE_SLASHES の正規化の前に / または \ にデコードされます。	/a%2fb は /a/b に正規化されます。	この設定に対して更新を行い、CVE-2021-31920 のリスクを軽減します。この設定はより安全ですが、アプリケーションが破損する可能性があります。実稼働環境にデプロイする前にアプリケーションをテストします。

正規化アルゴリズムは以下の順序で実行されます。

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



警告

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

1.2.27.56.3. パスの正規化設定の例

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

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

表1.2 設定例

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

1.2.27.56.4. パスを正規化するための SMCP の設定

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

SMCP v2 pathNormalization

```
spec:
  techPreview:
    global:
```

```
pathNormalization: <option>
```

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

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

EnvoyFilter をファイルに保存して、以下のコマンドを実行します。

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

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

1.2.27.57. Red Hat OpenShift Service Mesh 2.0.3 の新機能

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

また、このリリースには以下の新機能があります。

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

1.2.27.58. Red Hat OpenShift Service Mesh 2.0.2 の新機能

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

1.2.27.59. Red Hat OpenShift Service Mesh 2.0.1 の新機能

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

1.2.27.60. Red Hat OpenShift Service Mesh 2.0 の新機能

このリリースの Red Hat OpenShift Service Mesh では、Istio 1.6.5、Jaeger 1.20.0、Kiali 1.24.2、3scale Istio Adapter 2.0、および Red Hat OpenShift Service on AWS 4.6 のサポートが追加されました。

また、このリリースには以下の新機能があります。

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

1.2.28. 非推奨の機能と削除された機能

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

非推奨の機能は依然として Red Hat OpenShift Service on AWS に含まれており、引き続きサポートされますが、この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。

この製品では、削除機能が除去されています。

1.2.28.1. Red Hat OpenShift Service Mesh 2.5 で非推奨化および削除された機能

v2.2 の **ServiceMeshControlPlane** リソースはサポートされなくなりました。お客様は、新しいバージョンの **ServiceMeshControlPlane** リソースを使用するようにメッシュデプロイメントをアップグレードする必要があります。

Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator のサポートは非推奨になりました。トレーススパンを収集するには、Red Hat OpenShift 分散トレーシング Platform (Tempo) スタックを使用します。

OpenShift Elasticsearch Operator のサポートが非推奨になりました。

Istio は、ファーストパーティ JSON Web トークン (JWT) のサポートを削除します。Istio は今後もサードパーティの JWT をサポートします。

1.2.28.2. Red Hat OpenShift Service Mesh 2.4 で非推奨化および削除された機能

v2.1 の **ServiceMeshControlPlane** リソースはサポートされなくなりました。お客様は、新しいバージョンの **ServiceMeshControlPlane** リソースを使用するようにメッシュデプロイメントをアップグレードする必要があります。

Istio OpenShift Routing (IOR) のサポートは非推奨となり、将来のリリースでは削除される予定です。

Grafana のサポートは非推奨となり、将来のリリースでは削除される予定です。

Red Hat OpenShift Service Mesh 2.3 で非推奨となった以下の暗号スイートのサポートは、クライアント側とサーバー側の両方で TLS ネゴシエーションで使用される暗号のデフォルトのリストから削除されました。これらの暗号スイートのいずれかを必要とするサービスにアクセスする必要があるアプリケーションは、プロキシから TLS 接続が開始されると接続に失敗します。

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

1.2.28.3. Red Hat OpenShift Service Mesh 2.3 で非推奨化および削除された機能

次の暗号スイートのサポートは非推奨になりました。将来のリリースでは、クライアント側とサーバー側の両方で TLS ネゴシエーションに使用されるデフォルトの暗号リストから削除される予定です。

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

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

Red Hat OpenShift Service Mesh バージョン 2.2 で非推奨化された **ServiceMeshExtension** API は、Red Hat OpenShift Service Mesh バージョン 2.3 で廃止されました。**ServiceMeshExtension** API を使用している場合、WebAssembly エクステンションを引き続き使用するには **WasmPlugin** API に移行する必要があります。

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

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

1.2.28.5. Red Hat OpenShift Service Mesh 2.2 で削除された機能

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

1.2.28.6. Red Hat OpenShift Service Mesh 2.1 で削除された機能

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

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

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

Mixer コンポーネントはリリース 2.0 で非推奨となり、リリース 2.1 で削除されます。Mixer を使用したエクステンションの実装はリリース 2.0 でも引き続きサポートされますが、エクステンションを新規の [WebAssembly](#) メカニズムに移行する必要があります。

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

- **Policy** (authentication.istio.io/v1alpha1) はサポートされなくなりました。Policy リソースの特定の設定によっては、同じ効果を実現するために複数のリソースを設定しなければならない場合があります。
 - **RequestAuthentication** (security.istio.io/v1beta1) の使用
 - **PeerAuthentication** (security.istio.io/v1beta1) の使用
- **ServiceMeshPolicy** (maistra.io/v1) はサポートされなくなりました。
 - 上記のように **RequestAuthentication** または **PeerAuthentication** を使用しますが、Service Mesh コントロールプレーン namespace に配置します。
- **RbacConfig** (rbac.istio.io/v1alpha1) はサポートされなくなりました。

- **AuthorizationPolicy** (security.istio.io/v1beta1) に置き換わります。これは **RbacConfig**、**ServiceRole**、および **ServiceRoleBinding** の動作を包含します。
- **ServiceMeshRbacConfig** (maistra.io/v1) がサポートされなくなりました。
 - 上記のように **AuthorizationPolicy** を使用しますが、Service Mesh コントロールプレーンの namespace に配置します。
- **ServiceRole** (rbac.istio.io/v1alpha1) がサポートされなくなりました。
- **ServiceRoleBinding** (rbac.istio.io/v1alpha1) がサポートされなくなりました。
- Kiali では、**login** および **LDAP** ストラテジーは非推奨になりました。今後のバージョンでは、OpenID プロバイダーを使用した認証が導入されます。

1.2.29. 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。

- Red Hat OpenShift Service Mesh はまだ **IPv6** をフルサポートを提供していません。その結果、Red Hat OpenShift Service Mesh ではデュアルスタッククラスターはサポート対象外です。
- グラフレイアウト: Kiali グラフのレイアウトは、アプリケーションのアーキテクチャーや表示データ (グラフィックノードとその対話の数) によって異なることがあります。すべての状況に適した単一のレイアウトを作成することは不可能ではないにしても困難であるため、Kiali は複数の異なるレイアウトの選択肢を提供します。別のレイアウトを選択するには、**Graph Settings** メニューから異なる **Layout Schema** を選択します。
- Kiali コンソールから分散トレーシング Platform (Jaeger) や Grafana などの関連サービスに初めてアクセスする場合は、証明書を受け入れ、Red Hat OpenShift Service on AWS のログイン認証情報を使用して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。

1.2.29.1. Service Mesh の既知の問題

これらは Red Hat OpenShift Service Mesh の既知の問題です: * [OSSM-5556](#) istio-system ラベルが検出セクターと一致しない場合、ゲートウェイはスキップされます。

+ 回避策: ゲートウェイ設定がスキップされないように、検出セクターと一致するようにコントロールプレーンの namespace にラベルを付けます。

+ **ServiceMeshControlPlane** リソースの例

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

```
gateways:
  ingress:
    enabled: true
```

+ 次に、コマンドラインで次のコマンドを実行します。

+

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

- [OSSM-3890](#) マルチテナントメッシュデプロイメントでゲートウェイ API を使用しようとする
と、次のようなエラーメッセージが生成されます。

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

マルチテナントメッシュデプロイメントでゲートウェイ API をサポートするには、すべてのゲートウェイ API カスタムリソース定義 (CRD) ファイルがクラスター内に存在する必要があります。

マルチテナントメッシュデプロイメントでは、CRD スキャンが無効になり、Istio はクラスター内にどの CRD が存在するかを検出する方法がありません。その結果、Istio はサポートされているすべての Gateway API CRD を監視しようとしませんが、それらの CRD の一部が存在しない場合はエラーが生成されます。

Service Mesh 2.3.1 以降のバージョンは、**v1alpha2** と **v1beta1** の両方の CRD をサポートします。したがって、マルチテナントメッシュデプロイメントでゲートウェイ API をサポートするには、両方の CRD バージョンが存在する必要があります。

回避策: 次の例では、**kubectl get** 操作により **v1alpha2** および **v1beta1** CRD がインストールされます。URL には追加の **experimental** セグメントが含まれており、それに応じて既存のスク립トが更新されることに注意してください。

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

- [OSSM-2042](#) **default** という名前の SMCP のデプロイメントが失敗します。SMCP オブジェクトを作成し、そのバージョンフィールドを v2.3 に設定する場合、オブジェクトの名前は **default** にできません。名前が **default** の場合、コントロールプレーンはデプロイに失敗し、OpenShift は次のメッセージを含む **Warning** イベントを生成します。

```
Error processing component mesh-config: error: [mesh-
config/templates/telemetryv2_1.6.yaml: Internal error occurred: failed calling webhook
"rev.validation.istio.io": Post "https://istiod-default.istio-system.svc:443/validate?
timeout=10s": x509: certificate is valid for istiod.istio-system.svc, istiod-remote.istio-
system.svc, istio-pilot.istio-system.svc, not istiod-default.istio-system.svc, mesh-
config/templates/enable-mesh-permissive.yaml]
```

- [OSSM-1655](#) SMCP で **mTLS** を有効にした後に、Kiali ダッシュボードにエラーが表示されま
す。

SMCP で **spec.security.controlPlane.mtls** 設定を有効にすると、Kiali コンソールにエラー
メッセージ **No subsets defined** が表示されます。

- [OSSM-1505](#) この問題は、OpenShift Container Platform 4.11 で **ServiceMeshExtension** リソースを使用する場合にのみ発生します。OpenShift Container Platform 4.11 で **ServiceMeshExtension** を使用すると、リソースの準備が整いません。**oc describe ServiceMeshExtension** を使用して問題を調べると、**stderr: Error creating mount namespace before pivot: function not implemented** エラーが表示されます。

回避策: **ServiceMeshExtension** は Service Mesh 2.2 で廃止されました。**ServiceMeshExtension** から **WasmPlugin** リソースに移行します。詳細は、**ServiceMeshExtension** から **WasmPlugin** リソースへの移行を参照してください。
- [OSSM-1396](#) ゲートウェイリソースに **spec.externalIPs** 設定が含まれていると、ゲートウェイは、**ServiceMeshControlPlane** の更新時に再作成されず削除され、再作成されることはありません。
- [OSSM-1168](#) Service Mesh リソースが単一の YAML ファイルとして作成される場合は、Envoy プロキシサイドカーが Pod に確実に挿入されません。SMCP、SMMR、およびデプロイメントリソースを個別に作成すると、デプロイメントは想定どおりに機能します。
- [OSSM-1115](#) **spec.proxy** API の **concurrency** フィールドが istio-proxy に伝播されませんでした。**concurrency** フィールドは、**ProxyConfig** で設定すると機能します。**concurrency** フィールドは、実行するワーカースレッドの数を指定します。フィールドが **0** に設定されている場合、使用可能なワーカースレッドの数は CPU コアの数と等しくなります。フィールドが設定されていない場合、使用可能なワーカースレッドの数はデフォルトで **2** になります。

次の例では、**concurrency** フィールドが **0** に設定されています。

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

- [OSSM-1052](#) Service Mesh コントロールプレーンで入力ゲートウェイのサービス **ExternalIP** を設定すると、サービスは作成されません。SMCP のスキーマには、サービスのパラメーターがありません。

回避策: SMCP 仕様でゲートウェイの作成を無効にして、(サービス、ロール、および RoleBinding など) ゲートウェイのデプロイメントを完全に手動で管理します。
- [OSSM-882](#) これは、Service Mesh 2.1 以前に適用されます。namespace は **accessible_namespace** リストにありますが、Kiali UI には表示されません。デフォルトでは、Kiali は "kube" で始まる namespace を表示しません。これらの namespace は通常内部使用のみであり、メッシュの一部ではないためです。

たとえば、'akube-a' という名前の namespace を作成し、これを Service Mesh メンバーロールに追加すると、Kiali UI は namespace を表示しません。定義された除外パターンの場合、ソフトウェアは、このパターンで始まるか、そのパターンを含む namespace を除外します。

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

```
api:
  namespaces:
    exclude:
      - "^istio-operator"
```



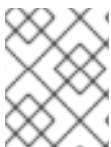
```
- "^kube-.*"
- "^openshift.*"
- "^kiali-operator"
```

- [MAISTRA-2692](#) Mixer が削除されると、Service Mesh 2.0.x で定義されたカスタムメトリクスを 2.1 で使用できません。カスタムメトリクスは **EnvoyFilter** を使用して設定できます。明示的に文書化されている場合を除き、Red Hat は **EnvoyFilter** 設定をサポートできません。これは、下層の Envoy API と疎結合されており、後方互換性を保持できないためです。
- [MAISTRA-1959](#) 2.0 への移行 Prometheus の収集 (`spec.addons.prometheus.scrape` が `true` に設定される) は mTLS が有効になっていると機能しません。また、Kiali は、mTLS が無効になっている場合に余分なグラフデータを表示します。
この問題は、たとえば、プロキシー設定からポート 15020 を除外して対応できます。

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

- [MAISTRA-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の挿入を発生させるには Pod を削除し、再作成する必要があります。
- [MAISTRA-158](#) 同じホスト名を参照する複数のゲートウェイを適用すると、すべてのゲートウェイが機能しなくなります。

1.2.29.2. Kiali の既知の問題



注記

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

Kiali の既知の問題は以下のとおりです。

- [OSSM-6299](#) Red Hat OpenShift Service on AWS 4.15 では、トラフィックグラフ内の任意の **Node graph** メニューオプションをクリックしても、ノードグラフが表示されません。代わりに、ページが同じトラフィックグラフで更新されます。現在、この問題に対する回避策はありません。
- [OSSM-6298](#) OpenShift Service Mesh Console (OSSMC) プラグイン内の項目参照 (特定のサービスに関連するワークロードリンクなど) をクリックすると、コンソールが目的のページを開く前に複数回のリダイレクトを実行することがあります。Web ブラウザーで **Back** をクリックすると、前のページではなく、コンソールの別のページが開きます。回避策として、**Back** を 2 回クリックして前のページに移動してください。
- [OSSM-6290](#) Red Hat OpenShift Service on AWS 4.15 では、**Istio Config** リストページの **Project** フィルターが正しく機能しません。ドロップダウンから特定のプロジェクトを選択した場合でも、すべての **istio** の項目が表示されます。現在、この問題に対する回避策はありません。

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

1.2.30. 修正された問題

以下の問題は、これまでのリリースで解決されています。

1.2.30.1. Service Mesh の修正された問題

- [OSSM-6177](#) 以前は、**ServiceMeshControlPlane** (SMCP) で検証メッセージが有効になっている場合、**GatewayAPI** サポートが有効になっていない限り、**istiod** が継続的にクラッシュしていました。現在は、検証メッセージが有効になっている場合、**GatewayAPI** サポートが有効になっていなくても、**istiod** が継続的にクラッシュすることはなくなりました。
- [OSSM-6163](#) 次の問題が解決されました。
 - 以前は、不安定な Prometheus イメージが Service Mesh コントロールプレーン (SMCP) v2.5 に含まれていたため、ユーザーが Prometheus ダッシュボードにアクセスできませんでした。現在は、Service Mesh Operator 2.5.1 で、Prometheus イメージが更新されています。
 - 以前は、Service Mesh コントロールプレーン (SMCP) で、Grafana データソースが Basic 認証のパスワードを自動的に設定できず、ユーザーが Grafana メッシュダッシュボードで Prometheus からのメトリクスを表示できませんでした。現在は、Grafana データソースのパスワードが **secureJsonData** フィールドで設定されます。メトリクスはダッシュボードに正しく表示されます。
- [OSSM-6148](#) 以前は、ユーザーが **Traffic Graph** ページのノードのメニューでオプションをクリックしても、OpenShift Service Mesh Console (OSSMC) プラグインが応答しませんでした。現在は、プラグインがメニューで選択されたオプションに応答し、対応する詳細ページにリダイレクトします。
- [OSSM-6099](#) 以前は、OpenShift Service Mesh Console (OSSMC) プラグインが IPv6 クラスターで正しくロードされませんでした。現在は、OSSMC プラグインの設定が変更され、IPv6 クラスターで適切にロードされるようになりました。
- [OSSM-5960](#) 以前は、OpenShift Service Mesh Console (OSSMC) プラグインが、バックエンドエラーや Istio 検証などの通知メッセージを表示しませんでした。現在は、これらの通知がプラグインページの上部に正しく表示されます。
- [OSSM-5959](#) 以前は、OpenShift Service Mesh Console (OSSMC) プラグインの **Overview** ページに TLS および Istio の認証情報が表示されませんでした。現在は、この情報が正しく表示されます。
- [OSSM-5902](#) 以前は、ユーザーが **Overview** ページで **Istio config** のヘルスシンボルをクリックすると、OpenShift Service Mesh Console (OSSMC) プラグインが “Not Found Page” エラーにリダイレクトされていました。現在は、プラグインが正しい **Istio config** 詳細ページにリダイレクトされます。

- [OSSM-5541](#) 以前は、一部の再起動状態で、Istio Operator Pod がリーダーリースを待機し続けることがありました。現在は、この問題を回避するためにリーダー選出の実装が強化されています。
- [OSSM-1397](#) 以前は、namespace から **maistra.io/member-of** ラベルを削除した場合、サービスメッシュ Operator により、ラベルは namespace に自動的に再適用されませんでした。その結果、サイドカーインジェクションは対象の namespace で機能しませんでした。

ServiceMeshMember オブジェクトに変更を加えた場合、Operator はラベルを namespace に再適用します。これにより、このメンバーオブジェクトの調整がトリガーされます。

今後は、namespace に変更を加えると、メンバーオブジェクトの調整もトリガーされます。

- [OSSM-3647](#) 以前は、Service Mesh コントロールプレーン (SMCP) v2.2 (Istio 1.12) では、WasmPlugin は受信リスナーにのみ適用されていました。SMCP v2.3 (Istio 1.14) 以降、WasmPlugin はデフォルトでインバウンドおよびアウトバウンドのリスナーに適用されるようになり、3scale WasmPlugin のユーザーにリグレーションが発生しました。環境変数 **APPLY_WASM_PLUGINS_TO_INBOUND_ONLY** が追加され、SMCP v2.2 から v2.3 および v2.4 への安全な移行が可能になります。

次の設定を SMCP config に追加する必要があります。

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

安全な移行を確保するには、次の手順を実行します。

1. SMCP v2.2 で **APPLY_WASM_PLUGINS_TO_INBOUND_ONLY** を設定します。
 2. 2.4 にアップグレードします。
 3. WasmPlugins で **spec.match[].mode: SERVER** を設定します。
 4. 以前に追加した環境変数を削除します。
- [OSSM-4851](#) 以前は、**runAsGroup**、**runAsUser**、または **fsGroup** パラメーターが **nil** の場合、メッシュ内をスコープとする namespace に新しい Pod をデプロイするオペレータでエラーが発生しました。**nil** 値を回避するために yaml 検証が追加されました。
 - [OSSM-3771](#) 以前は、Service Mesh Control Plane (SMCP) で定義された追加の Ingress ゲートウェイに対して OpenShift ルートを無効にすることができませんでした。現在、**routeConfig** ブロックを追加の各 **additionalIngress** ゲートウェイに追加できるため、ゲートウェイごとに OpenShift ルートの作成を有効または無効にできます。
 - [OSSM-4197](#) では、'ServiceMeshControlPlane' リソースの v2.2 または v2.1 をデプロイしても、**/etc/cni/multus/net.d/** ディレクトリーは作成されませんでした。その結果、**istio-cni** Pod の準備ができず、**istio-cni** Pod のログに次のメッセージが含まれていました。

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

ここで、'ServiceMeshControlPlane' リソースの v2.2 または v2.1 をデプロイすると、**/etc/cni/multus/net.d/** ディレクトリーが作成され、**istio-cni** Pod の準備が整います。

- [OSSM-3993](#) 以前は、Kiali は標準 HTTPS ポート **443** 上のプロキシ経由の OpenShift OAuth のみをサポートしていました。現在、Kiali は非標準の HTTPS ポートを介した OpenShift OAuth をサポートします。ポートを有効にするには、Kiali CR で **spec.server.web_port** フィールドをプロキシの非標準 HTTPS ポートに設定する必要があります。
- [OSSM-3936](#) 以前は、**injection_label_rev** および **injection_label_name** 属性の値がハードコーディングされていました。これにより、カスタム設定が Kiali カスタムリソース定義 (CRD) で有効になりませんでした。今回のリリースより、属性値はハードコーディングされなくなりました。**spec.istio_labels** 仕様の **injection_label_rev** および **injection_label_name** 属性の値をカスタマイズできます。
- [OSSM-3644](#) これまで、フェデレーション egress-gateway はネットワークゲートウェイエンドポイントの誤った更新を受信し、余分なエンドポイントエントリが発生していました。現在は、federation-egress ゲートウェイがサーバー側で更新され、正しいネットワークゲートウェイエンドポイントを受信するようになりました。
- [OSSM-3595](#) これまでは、**iptables-restore** ユーティリティーが **/tmp** ディレクトリー内のファイルを開くことを SELinux が許可しなかったため、RHEL 上で **istio-cni** プラグインが失敗することがありました。SELinux は、ファイル経由ではなく **stdin** 入力ストリーム経由で **iptables-restore** を渡すようになりました。
- [OSSM-3586](#) 以前は、Google Cloud Platform (GCP) メタデータサーバーが利用できない場合に Istio プロキシの起動が遅くなりました。Istio 1.14.6 にアップグレードすると、メタデータサーバーが利用できない場合でも、Istio プロキシは GCP 上で期待どおりに起動します。
- [OSSM-3025](#) Istiod が準備完了にならないことがあります。メッシュに多くのメンバーの namespace が含まれている場合は、Istiod 内のデッドロックが原因で Istiod Pod が準備完了にならないことがありました。デッドロックが解決され、Pod が期待どおりに起動するようになりました。
- [OSSM-2493](#) SMCP のデフォルトの **nodeSelector** と **tolerations** が Kiali に渡されません。**SMCP.spec.runtime.defaults** に追加する **nodeSelector** と **tolerations** が Kiali リソースに渡されるようになりました。
- [OSSM-2492](#) SMCP のデフォルトの toleration が Jaeger に渡されません。**SMCP.spec.runtime.defaults** に追加する **nodeSelector** と **tolerations** が Jaeger リソースに渡されるようになりました。
- [OSSM-2374](#) **ServiceMeshMember** リソースの1つを削除すると、Service Mesh Operator が **ServiceMeshMemberRoll** を削除しました。これは、最後の **ServiceMeshMember** を削除する際に期待される動作ですが、削除されたメンバーに加えて、メンバーが含まれている場合、Operator は **ServiceMeshMemberRoll** を削除しないようにする必要があります。この問題は修正され、Operator は最後の **ServiceMeshMember** リソースが削除された場合のみ、**ServiceMeshMemberRoll** を削除するようになりました。
- [OSSM-2373](#) ログイン時に OAuth メタデータを取得しようとして、エラーが発生しました。クラスターのバージョンを取得するには、**system:anonymous** アカウントが使用されます。クラスターのデフォルトのバンドルされた ClusterRole と ClusterRoleBinding を使用すると、匿名アカウントはバージョンを正しく取得できます。**system:anonymous** アカウントがクラスターバージョンを取得する権限を失うと、OpenShift 認証は使用できなくなります。これは、Kiali SA を使用して、クラスターのバージョンを取得することで修正されます。これにより、クラスターのセキュリティも向上します。
- [OSSM-2371](#) Kiali が "view-only" として設定されているにもかかわらず、ユーザーはワークロードの詳細の Logs タブの kebab メニューからプロキシログレベルを変更できます。この問題は修正されており、Kiali が "view-only" として設定されている場合は、"Set Proxy Log Level" の下のオプションが無効になります。

- [OSSM-2344](#) Istiod を再起動すると、Kiali によって CRI-O がポート転送リクエストでいっぱいになります。この問題は、Kiali が Istiod に接続できず、Kiali が同時に大量のリクエストを istiod に発行したときに発生しました。Kiali が istiod に送信するリクエストの数が制限されるようになりました。
- [OSSM-2335](#) Traces の散布図のプロット上でマウスポインターをドラッグすると、同時バックエンドリクエストが原因で Kiali コンソールが応答を停止する場合があります。
- [OSSM-2221](#) 以前は、デフォルトで **ignore-namespace** ラベルが namespace に適用されていたため、**ServiceMeshControlPlane** namespace へのゲートウェイインジェクションはできませんでした。
v2.4 コントロールプレーンを作成すると、namespace に **ignore-namespace** ラベルが適用されなくなり、ゲートウェイインジェクションが可能になります。

以下の例では、**oc label** コマンドは既存のデプロイメントの namespace から **ignore-namespace** ラベルを削除します。

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

ここでは、以下のようになります。

istio_system

ServiceMeshControlPlane namespace の名前を指定します。

- [OSSM-2053](#) Red Hat OpenShift Service Mesh Operator 2.2 または 2.3 を使用すると、SMCP の調整中に、SMMR コントローラーがメンバーの namespace を **SMMR.status.configuredMembers** から削除しました。これにより、メンバーの namespace のサービスがしばらく利用できなくなりました。
Red Hat OpenShift Service Mesh Operator 2.2 または 2.3 を使用すると、SMMR コントローラーは **SMMR.status.configuredMembers** から namespace を削除しなくなります。代わりに、コントローラーは namespace を **SMMR.status.pendingMembers** に追加して、それらが最新ではないことを示します。調整中に、各 namespace が SMCP と同期されると、namespace は **SMMR.status.pendingMembers** から自動的に削除されます。
- [OSSM-1962](#) フェデレーションコントローラーで **EndpointSlices** を使用します。フェデレーションコントローラーが **EndpointSlices** を使用するようになりました。これにより、大規模なデプロイメントでのスケーラビリティとパフォーマンスが向上します。
PILOT_USE_ENDPOINT_SLICE フラグはデフォルトで有効になっています。フラグを無効にすると、フェデレーションデプロイメントを使用できなくなります。
- [OSSM-1668](#) 新しいフィールド **spec.security.jwksResolverCA** がバージョン 2.1 **SMCP** に追加されましたが、2.2.0 および 2.2.1 リリースにはありませんでした。このフィールドが存在する Operator バージョンから、このフィールドが存在しなかった Operator バージョンにアップグレードする場合は、**SMCP** で **.spec.security.jwksResolverCA** フィールドを使用できませんでした。
- [OSSM-1325](#) istiod Pod がクラッシュし、**fatal error: concurrent map iteration and map write** のエラーメッセージが表示されます。
- [OSSM-1211](#) フェイルオーバー用のフェデレーション Service Mesh の設定が想定どおりに機能しません。
Istiod パイロットログに、**envoy connection [C289] TLS error: 337047686:SSL routines:tls_process_server_certificate:certificate verify failed** のエラーが表示されます。
- [OSSM-1099](#) Kiali コンソールに **Sorry, there was a problem. Try a refresh or navigate to a different page.** メッセージが表示されました

- [OSSM-1074](#) SMCP で定義された Pod アノテーションが Pod に注入されません。
- [OSSM-999](#) Kiali は想定どおりに保持されませんでした。ダッシュボードグラフでは、カレンダーの時刻がグレイアウトされています。
- [OSSM-797](#) Kiali Operator Pod は、Operator のインストールまたはアップグレード時に **CreateContainerConfigError** を生成します。
- **kube** で始まる [OSSM-722](#) namespace は Kiali には表示されません。
- [OSSM-569](#): Prometheus **istio-proxy** コンテナには CPU メモリ制限がありません。Prometheus **istio-proxy** サイドカーは、**spec.proxy.runtime.container** で定義されたリソース制限を使用するようになりました。
- [OSSM-535](#) SMCP での validationMessages のサポート。Service Mesh コントロールプレーンの **ValidationMessages** フィールドを **True** に設定できるようになりました。これにより、問題のトラブルシューティングに役立つ、リソースのステータスのログが書き込まれます。
- [OSSM-449](#) VirtualService および Service により、"Only unique values for domains are permitted.Duplicate entry of domain." エラーが生じます。
- 同様の名前を持つ [OSSM-419](#) namespace は、namespace が Service Mesh Member Role で定義されていない場合でも、Kiali namespace の一覧に表示されます。
- [OSSM-296](#) ヘルス設定を Kiali カスタムリソース (CR) に追加する場合、これは Kiali configmap にレプリケートされません。
- [OSSM-291](#) Kiali コンソールの、Applications、Services、および Workloads ページの "Remove Label from Filters" が機能しません。
- [OSSM-289](#) Kiali コンソールの 'istio-ingressgateway' および 'jaeger-query' サービスの Service Details ページにはトレースが表示されません。トレースは Jaeger にあります。
- [OSSM-287](#) Kiali コンソールでは、トレースが Graph Service に表示されません。
- [OSSM-285](#) Kiali コンソールにアクセスしようとする、"Error trying to get OAuth Metadata" エラーメッセージが表示されます。
回避策: Kiali Pod を再起動します。
- [MAISTRA-2735](#) Red Hat OpenShift Service Mesh バージョン 2.1 では、SMCP の調整時に Service Mesh Operator が削除するリソースが変更されました。以前のバージョンでは、Operator は以下のラベルを持つリソースを削除しました。
 - **maistra.io/owner**
 - **app.kubernetes.io/version**

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

- [MAISTRA-2687](#) 外部証明書を使用する場合は、Red Hat OpenShift Service Mesh 2.1 フェデレーションゲートウェイでは、証明書チェーンが完全に送信されません。Service Mesh フェデレーション egress ゲートウェイはクライアント証明書のみを送信します。フェデレーション Ingress ゲートウェイはルート証明書のみを認識するため、ルート証明書をフェデレーションインポート **ConfigMap** に追加しない限り、クライアント証明書を検証できません。

- [MAISTRA-2635](#) 非推奨の Kubernetes API が置き換えられました。Red Hat OpenShift Service on AWS 4.8 との互換性を維持するために、**apiextensions.k8s.io/v1beta1** API は Red Hat OpenShift Service Mesh 2.0.8 で非推奨になりました。
- [MAISTRA-2631](#) nsenter バイナリーが存在しないことが原因で Podman に問題が発生しているため、WASM 機能は使用できません。Red Hat OpenShift Service Mesh は **Error: error configuring CNI network plugin exec: "nsenter": executable file not found in \$PATH** エラーメッセージを生成します。コンテナイメージには nsenter が含まれ、WASM が予想通りに機能するようになりました。
- [MAISTRA-2534](#) istiod が JWT ルールで指定された発行者の JWKS の取得を試行する際に、発行者サービスは 502 で応答します。これにより、プロキシコンテナの準備ができなくなり、デプロイメントがハングしていました。[コミュニティバグ](#) の修正は、Service Mesh 2.0.7 リリースに含まれています。
- [MAISTRA-2411](#) Operator が **ServiceMeshControlPlane** で **spec.gateways.additionalIngress** を使用して新規 ingress ゲートウェイを作成する場合、Operator はデフォルトの istio-ingressgateway の場合と同様に追加の Ingress ゲートウェイの **NetworkPolicy** を作成しません。これにより、新規ゲートウェイのルートから 503 応答が生じました。
回避策: **istio-system** namespace に **NetworkPolicy** を手動で作成します。
- [MAISTRA-2401](#) CVE-2021-3586 servicemesh-operator: NetworkPolicy リソースが Ingress リソースのポートを誤って指定しています。Red Hat OpenShift Service Mesh にインストールされた NetworkPolicy リソースでは、アクセス可能なポートが適切に指定されませんでした。これにより、任意の Pod からこれらのリソースの全ポートにアクセスできるようになりました。以下のリソースに適用されるネットワークポリシーが影響を受けます。
 - Galley
 - Grafana
 - Istiod
 - Jaeger
 - Kiali
 - Prometheus
 - サイドカーインジェクター
- [MAISTRA-2378](#) クラスターが **ovs-multitenant** で OpenShift SDN を使用するように設定されており、メッシュに多数の namespace (200+) が含まれる場合に、Red Hat OpenShift Service on AWS ネットワークプラグインは namespace を迅速に設定できません。Service Mesh がタイムアウトになると、namespace が Service Mesh から継続的にドロップされ、再リストされます。
- [MAISTRA-2370](#) は listerInformer で tombstones を処理します。更新されたキャッシュコードベースは、namespace キャッシュからのイベントを集約されたキャッシュに変換する際に tombstones を処理しないため、go ルーチンでパニックが生じました。
- [MAISTRA-2117](#) オプションの **ConfigMap** マウントを Operator に追加します。CSV にはオプションの **ConfigMap** ボリュームマウントが含まれるようになり、**smcp-templates ConfigMap** (存在する場合) をマウントします。**smcp-templates ConfigMap** が存在しないと、マウントされたディレクトリーは空になります。**ConfigMap** を作成すると、ディレクトリーには **ConfigMap** からのエントリーが設定され、**SMCP.spec.profiles** で参照できます。Service Mesh Operator の再起動は必要ありません。

CSV を変更して 2.0 Operator を使用して smcp-templates ConfigMap をマウントしている場合は、Red Hat OpenShift Service Mesh 2.1 にアップグレードできます。アップグレード後は、CSV を編集せずに、既存の ConfigMap およびこれに含まれるプロファイルを引き続き使用できます。以前別の名前で ConfigMap を使用していた場合は、ConfigMap の名前を変更するか、アップグレード後に CSV を更新する必要があります。

- [MAISTRA-2010](#) AuthorizationPolicy は **request.regex.headers** フィールドをサポートしません。**validatingwebhook** はこのフィールドのある AuthorizationPolicy を拒否し、これを無効にした場合でも、パイロットは同じコードを使用してこの検証を試行し、機能しません。
- [MAISTRA-1979](#) 2.0 への移行 変換 webhook は、**SMCP.status** を v2 から v1 に変換する際に以下の重要なフィールドをドロップします。
 - conditions
 - components
 - observedGeneration
 - annotations

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

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

- [MAISTRA-1983](#) 2.0 への移行 既存の無効な **ServiceMeshControlPlane** を使用した 2.0.0 へのアップグレードは修復できません。**ServiceMeshControlPlane** リソース内の無効な項目により、回復不可能なエラーが発生しました。修正により、エラーが回復可能になりました。無効なリソースを削除してこれを新しいリソースに置き換えるか、リソースを編集してエラーを修正できます。リソースの編集に関する詳細は、[Red Hat OpenShift Service Mesh インストールの設定] を参照してください。
- [MAISTRA-1502](#) バージョン 1.0.10 の CVE の修正により、Istio ダッシュボードは Grafana の **Home Dashboard** メニューから利用できなくなりました。Istio ダッシュボードにアクセスするには、ナビゲーションパネルの **Dashboard** メニューをクリックし、**Manage** タブを選択します。
- [MAISTRA-1399](#) Red Hat OpenShift Service Mesh では、サポート対象外の CNI プロトコルがインストールされなくなりました。サポート対象のネットワーク設定は変更されていません。
- [MAISTRA-1089](#) 2.0 への移行 コントロールプレーン以外の namespace で作成されたゲートウェイは自動的に削除されます。SMCP 仕様からゲートウェイ定義を削除した後にこれらのリソースを手動で削除する必要があります。
- [MAISTRA-858](#) Istio 1.1.x に関連する非推奨のオプションと設定 を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] 非推奨の 'envoy.api.v2.listener.Filter.config' オプションの使用。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] lds.proto ファイルから非推奨の 'envoy.api.v2.Listener.use_original_dst' オプションを使用。この設定はまもなく Envoy から削除されます。

- [MAISTRA-806](#) エビクトされた Istio Operator Pod により、メッシュおよび CNi はデプロイできなくなります。
回避策: コントロールペインのデプロイ時に **istio-operator** Pod がエビクトされる場合は、エビクトされた **istio-operator** Pod を削除します。
- [MAISTRA-681](#) Service Mesh コントロールプレーンに多くの namespace がある場合に、パフォーマンスの問題が発生する可能性があります。
- [MAISTRA-193](#) ヘルスチェックが citadel で有効になっていると、予期しないコンソール情報メッセージが表示されます。
- [Bugzilla 1821432](#) Red Hat OpenShift Service on AWS カスタムリソースの詳細ページのトグルコントロールで CR が正しく更新されません。Red Hat OpenShift Service on AWS Web コンソールの Service Mesh Control Plane (SMCP) Overview ページの UI のトグルコントロールにより、リソースの誤ったフィールドが更新されることがあります。SMCP を更新するには、YAML コンテンツを直接編集するか、トグルコントロールをクリックせずにコマンドラインからリソースを更新します。

1.3. SERVICE MESH について

Red Hat OpenShift Service Mesh は、Service Mesh 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 を使用すると、Red Hat OpenShift Service on AWS 環境でマイクロサービスを接続、保護、監視できます。

1.3.1. Red Hat OpenShift Service Mesh とは

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

オープンソースの [Istio](#) プロジェクトをベースとする Red Hat OpenShift Service Mesh は、サービスコードに変更を加えずに、既存の分散したアプリケーションに透過的な層を追加します。Red Hat OpenShift Service Mesh サポートをサービスに追加するには、マイクロサービス間のすべてのネットワーク通信を傍受する特別なサイドカープロキシをメッシュ内の関連サービスにデプロイします。Service Mesh コントロールプレーンの機能を使用して Service Mesh を設定し、管理します。

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

- 検出
- 負荷分散
- サービス間の認証
- 障害回復
- メトリクス
- モニタリング

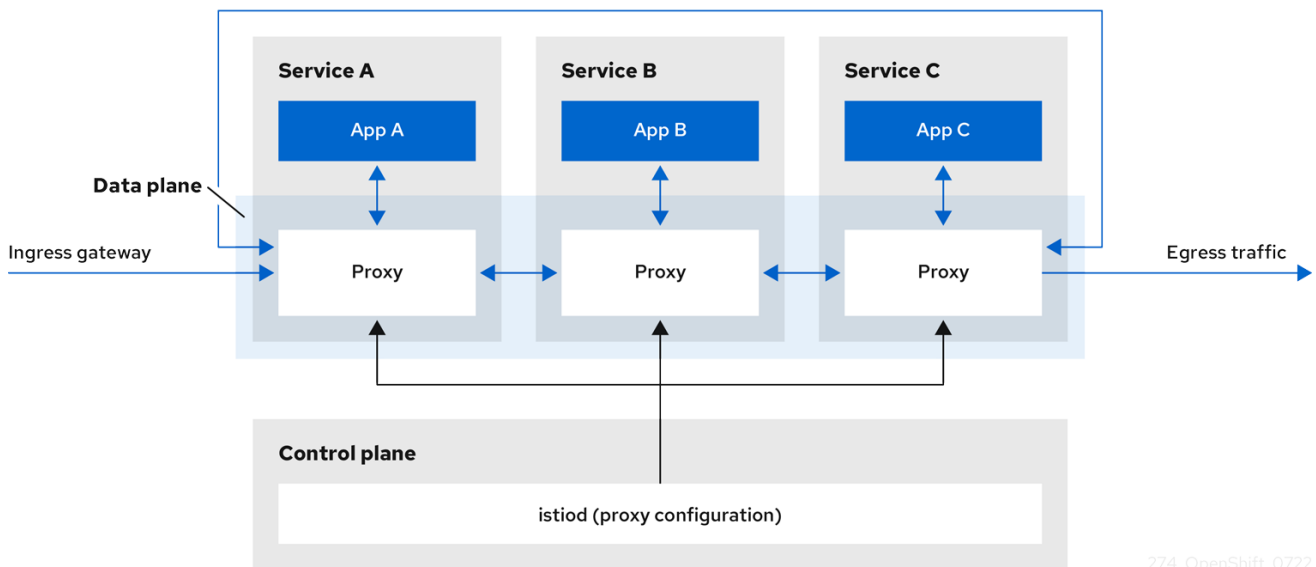
Red Hat OpenShift Service Mesh は、以下を含むより複雑な運用機能も提供します。

- A/B テスト

- カナリアリリース
- アクセス制御
- エンドツーエンド認証

1.3.2. Service Mesh アーキテクチャー

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



274_OpenShift_0722

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

データプレーンは、Pod のアプリケーションコンテナとともに実行するインテリジェントプロキシのセットであり、Service Mesh 内のマイクロサービス間で起こる受信および送信ネットワーク通信をすべて傍受し、制御します。データプレーンは、すべての受信 (ingress) および送信 (egress) ネットワークトラフィックを傍受する方法で実装されます。Istio データプレーンは、Pod 内のアプリケーションコンテナと並行して実行される Envoy コンテナで構成されます。Envoy コンテナはプロキシとして機能し、すべてのネットワーク通信を Pod に対して制御します。

- **Envoy** プロキシは、データプレーントラフィックと対話する唯一の Istio コンポーネントです。プロキシ経由でサービスフロー間の受信 (ingress) および送信 (egress) ネットワークトラフィックはすべて、そのプロキシを介して行われます。また、Envoy プロキシは、メッシュ内のサービストラフィックに関連するすべてのメトリクスを収集します。Envoy プロキシはサイドカーとしてデプロイされ、サービスと同じ Pod で実行されます。Envoy プロキシは、メッシュゲートウェイの実装にも使用されます。
 - **Sidecar** プロキシは、ワークロードインスタンスのインバウンドおよびアウトバウンド通信を管理します。
 - **ゲートウェイ** は、受信または送信 HTTP/TCP 接続を受信するロードバランサーとして動作するプロキシです。ゲートウェイ設定は、サービスワークロードとともに実行するサイドカー Envoy プロキシではなく、メッシュのエッジで実行するスタンドアロン Envoy プロキシに適用されます。ゲートウェイを使用してメッシュの受信トラフィックおよび送信トラフィックを管理することで、メッシュに入るか、メッシュを出るトラフィックを指定できます。

- **Ingress-gateway** - ingress コントローラーとしても知られる、Ingress ゲートウェイは Service Mesh に入るトラフィックを受信し、制御する専用の Envoy プロキシです。Ingress ゲートウェイは、モニタリングおよびルーティングルールなどの機能をクラスターに入るトラフィックに適用できるようにします。
- **Egress-gateway** - egress コントローラーとしても知られる、Egress Gateway は Service Mesh からトラフィックを管理する専用の Envoy プロキシです。Egress Gateway は、モニタリングおよびルートルールなどの機能をメッシュのトラフィックに適用できるようにします。

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

- Istio コントロールプレーンは、以前の複数のコントロールプレーンコンポーネント (Citadel、Galley、Pilot) を単一バイナリに統合する **Istiod** で構成されています。Istiod は、サービス検出、設定、および証明書の管理を行います。これは、高レベルのルーティングルールを Envoy 設定に変換し、それらをランタイム時にサイドカーコンテナに伝播します。
 - Istiod は認証局 (CA) として機能し、データプレーンでセキュアな mTLS 通信に対応する証明書を生成します。この場合は、外部 CA を使用することもできます。
 - Istiod は、サイドカーコンテナを OpenShift クラスターにデプロイされたワークロードに挿入します。

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

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

- **Kiali**: Kiali は Red Hat OpenShift Service Mesh の管理コンソールです。ダッシュボード、可観測性、および堅牢な設定、ならびに検証機能を提供します。これは、トラフィックトポロジーを推測して Service Mesh の構造を示し、メッシュの正常性を表示します。Kiali は、詳細なメトリクス、強力な検証、Grafana へのアクセス、分散トレーシングプラットフォーム (Jaeger) との強力な統合を提供します。
- **Prometheus**: Red Hat OpenShift Service Mesh は Prometheus を使用してサービスからのテレメトリ情報を保存します。Kiali は、メトリクス、ヘルスステータス、およびメッシュトポロジーを取得するために Prometheus に依存します。
- **Jaeger** - Red Hat OpenShift Service Mesh は、分散トレーシングプラットフォーム (Jaeger) をサポートしています。Jaeger はオープンソースのトレース機能で、複数のサービス間の単一要求に関連付けられたトレースを一元管理し、表示します。分散トレーシングプラットフォーム (Jaeger) を使用すると、マイクロサービスベースの分散システムを監視およびトラブルシューティングできます。
- **Elasticsearch**: Elasticsearch は、オープンソースの分散型 JSON ベースの検索および解析エンジンです。分散トレーシングプラットフォーム (Jaeger) は、永続ストレージに Elasticsearch を使用します。
- **Grafana**: Grafana は、Istio データの高度なクエリーおよびメトリクス分析、ならびにダッシュボードを使用してメッシュ管理者を提供します。任意で、Grafana を使用して Service Mesh メトリクスを分析できます。

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

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

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

1.3.3. Kiali について

Kiali は、Service Mesh のマイクロサービスとそれらの接続方法を表示して Service Mesh を可視化します。

1.3.3.1. Kiali の概要

Kiali は、Red Hat OpenShift Service on AWS で実行されている Service Mesh に可観測性を提供します。Kiali は、Istio Service Mesh の定義、検証、および確認に役立ちます。トポロジを推測することで Service Mesh の構造を理解するのに役立ち、Service Mesh の正常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) に関する洞察を提供し、選択されたグラフノードまたはエッジに関するコンテキスト情報やチャートを含む対話を表示できます。Kiali は、ゲートウェイ、宛先ルール、仮想サービス、メッシュポリシーなど、Istio 設定を検証する機能も提供します。Kiali は詳細なメトリクスを提供し、基本的な Grafana 統合は高度なクエリーに利用できます。Jaeger を Kiali コンソールに統合することで、分散トレースを提供します。

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

1.3.3.2. Kiali アーキテクチャー

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

- **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アプリケーションプラットフォームで実行され、Service Mesh 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 Service Mesh 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 およびクラスター 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 が有効になっている場合、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。

す。Kiali は Prometheus と直接通信し、Istio Telemetry で使用されるデータスキーマを想定します。Prometheus は Istio に依存しており、Kiali と明示的な依存関係があるため、Kiali の機能の多くは Prometheus なしに機能しません。

- **Cluster API** - Kiali は Service Mesh 設定を取得し、解決するために、Red Hat OpenShift Service on AWS (クラスター API) の API を使用します。Kiali はクラスター 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間の関係も解決します。クラスター API に対してもクエリーを実行し、仮想サービス、宛先ルール、ルートルール、ゲートウェイ、クォータなどの Istio 設定を取得します。
- **Jaeger**: Jaeger はオプションですが、Red Hat OpenShift Service Mesh インストールの一部としてデフォルトでインストールされます。デフォルトの Red Hat OpenShift Service Mesh インストールの一部として分散トレーシング Platform (Jaeger) をインストールすると、Kiali コンソールに分散トレーシングデータを表示するタブが含まれます。Istio の分散トレース機能を無効にした場合、トレースデータは利用できないことに注意してください。また、トレースデータを表示するには、ユーザーは Service Mesh コントロールプレーンがインストールされている namespace にアクセスできる必要があります。
- **Grafana**: Grafana はオプションですが、デフォルトでは Red Hat OpenShift Service Mesh インストールの一部としてインストールされます。使用可能な場合は、Kiali のメトリクスページに Grafana 内の同じメトリクスにユーザーを移動させるリンクが表示されます。Grafana ダッシュボードへのリンクと Grafana データを表示するには、Service Mesh コントロールプレーンがインストールされている namespace にユーザーがアクセスできる必要があることに注意してください。

1.3.3.3. Kiali の機能

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

- **健全性**: アプリケーション、サービス、またはワークロードの問題を素早く特定します。
- **トポロジー**: Kiali グラフを使用して、アプリケーション、サービス、またはワークロードの通信方法を可視化します。
- **メトリクス**: 事前定義済みのメトリクスダッシュボードを使用すると、Go、Node.js、Quarkus、Spring Boot、Thorntail、および Vert.x また、独自のカスタムダッシュボードを作成することもできます。
- **トレース**: Jaeger との統合により、アプリケーションを設定するさまざまなマイクロサービスで要求のパスを追跡できます。
- **検証**: 最も一般的な Istio オブジェクト (宛先ルール、サービスエントリ、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、Kiali コンソールの YAML エディターを直接使用して、Istio ルーティング設定を作成し、更新し、削除できるオプションの機能です。

1.3.4. 分散トレースについて

ユーザーがアプリケーションでアクションを実行するたびに、応答を生成するために多数の異なるサービスに参加を要求する可能性のあるアーキテクチャによって要求が実行されます。この要求のパスは分散トランザクションです。分散トレーシングプラットフォーム (Jaeger) を使用すると、アプリケーションを設定するさまざまなマイクロサービスを通じてリクエストのパスを追跡する分散トレーシングを実行できます。

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

分散トレーシングプラットフォーム (Jaeger) は、マイクロサービスのスタック全体にわたる個々のリクエストの実行を記録し、それらを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、作業の論理単位を表しています。スパンは因果関係をモデル化するためにネスト化され、順序付けられます。

1.3.4.1. 分散トレースの概要

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

分散トレーシングプラットフォームを使用すると、次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

1.3.4.2. Red Hat OpenShift 分散トレーシングプラットフォームアーキテクチャー

Red Hat OpenShift 分散トレーシング Platform は、トレースデータを収集、保存、表示するために連携して動作する複数のコンポーネントで設定されています。

- **Red Hat OpenShift 分散トレーシング Platform**- このコンポーネントは、オープンソースの [Grafana Tempo プロジェクト](#) に基づいています。
 - **Gateway:** ゲートウェイは、認証、認可、およびディストリビューターまたはクエリーフロントエンドサービスへのリクエストの転送を処理します。
 - **Distributor:** ディストリビューターは、Jaeger、OpenTelemetry、Zipkin などの複数の形式のスパンを受け入れます。**traceID** をハッシュ化し、分散コンシステントハッシュリングを使用して、スパンを Ingester にルーティングします。
 - **Ingester:** Ingester はトレースをブロックにバッチ化し、ブルームフィルターとインデックスを作成してすべてバックエンドにフラッシュします。
 - **Query Frontend:** Query Frontend は、受信クエリーの検索スペースをシャーディングします。次に、検索クエリーが Querier に送信されます。Query Frontend のデプロイメントでは、Tempo Query サイドカーを介して Jaeger UI が公開されます。
 - **Querier:** Querier は、Ingester またはバックエンドストレージで要求されたトレース ID を検索します。パラメーターに応じて、Ingester にクエリーを実行し、バックエンドから Bloom インデックスを取得して、オブジェクトストレージ内のブロックを検索できます。
 - **Compactor:** Compactor は、ブロックをバックエンドストレージとの間でストリーミングして、ブロックの総数を減らします。

- **Red Hat build of OpenTelemetry**- このコンポーネントは、オープンソースの [OpenTelemetry プロジェクト](#) に基づいています。
 - **OpenTelemetry Collector**: OpenTelemetry Collector は、テレメトリデータを受信、処理、エクスポートするためのベンダーに依存しない方法です。OpenTelemetry Collector は、Jaeger や Prometheus などのオープンソースの可観測性データ形式をサポートし、1 つ以上のオープンソースまたは商用バックエンドに送信します。Collector は、インストールメンテーションライブラリーがテレメトリデータをエクスポートするデフォルトの場所です。
- **Red Hat OpenShift 分散トレーシング Platform (Jaeger)**- このコンポーネントはオープンソースの [Jaeger プロジェクト](#) に基づいています。

重要

Red Hat OpenShift 分散トレーシング Platform (Jaeger) は非推奨の機能です。非推奨の機能は依然として Red Hat OpenShift Service on AWS に含まれており、引き続きサポートされますが、この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。

Red Hat OpenShift 分散トレーシング Platform Operator (Jaeger) は、今後のリリースの **redhat-operators** カタログから削除されます。詳細は、Red Hat ナレッジベースソリューション [Jaeger Deprecation and Removal in OpenShift](#) を参照してください。

分散トレーシングの収集と保存を行うには、Tempo Operator と Red Hat build of OpenTelemetry に移行する必要があります。

- クライアント (Jaeger クライアント、Tracer、Reporter、インストルメント化されたアプリケーション、クライアントライブラリー) - 分散トレーシング Platform (Jaeger) クライアントは、OpenTracing API の言語固有の実装です。クライアントは、手動で、またはさまざまな既存のオープンソースフレームワーク (Camel (Fuse)、Spring Boot (RHOAR)、MicroProfile (RHOAR/Thorntail)、Wildfly (EAP) など、OpenTracing にすでに統合されているもの) を使用して、分散トレーシング用にアプリケーションを計装するために使用できます。
- エージェント (Jaeger エージェント、サーバーキュー、プロセッサーワーカー) - 分散トレーシング Platform (Jaeger) エージェントは、User Datagram Protocol (UDP) 経由で送信されるスパンをリッスンし、それをバッチ処理してコレクターに送信するネットワークデーモンです。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 などのコンテナ環境にサイドカーコンテナを配置することによって実行されます。
- **Jaeger Collector** (Collector、Queue、Worker): Jaeger エージェントと同様に、Jaeger Collector はスパンを受信し、これら进行处理するために内部キューに配置します。これにより、Jaeger Collector はスパンがストレージに移動するまで待機せずに、クライアント/エージェントにすぐに戻るができます。
- **Storage** (Data Store): コレクターには永続ストレージのバックエンドが必要です。Red Hat OpenShift 分散トレーシング Platform (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。Red Hat OpenShift 分散トレーシング Platform (Jaeger) は、Elasticsearch ストレージをサポートしています。
- **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。
- **Ingestor** (Ingestor サービス) - Red Hat OpenShift 分散トレーシング Platform は、

Collector と実際の Elasticsearch バックエンドストレージ間のバッファとして Apache Kafka を使用できます。Ingestor は、Kafka からデータを読み取り、Elasticsearch ストレージバックエンドに書き込むサービスです。

- **Jaeger コンソール** - Red Hat OpenShift 分散トレーシング Platform (Jaeger) ユーザーインターフェイスを使用すると、分散トレーシングデータを視覚化できます。検索ページで、トレースを検索し、個別のトレースを設定するスパンの詳細を確認できます。

1.3.4.3. Red Hat OpenShift 分散トレーシングプラットフォームの機能

Red Hat OpenShift 分散トレーシング Platform は、次の機能を提供します。

- **Kiali との統合** - 適切に設定すると、Kiali コンソールから分散トレーシング Platform データを表示できます。
- **High スケーラビリティ** - 分散トレーシングプラットフォームのバックエンドは、単一障害点がなく、ビジネスニーズに合わせて拡張できるように設計されています。
- **分散コンテキストの伝播**: さまざまなコンポーネントからのデータをつなぎ、完全なエンドツーエンドトレースを作成できます。
- **Zipkin との下位互換性** - Red Hat OpenShift 分散トレーシング Platform には、Zipkin の代替として使用できる API がありますが、Red Hat はこのリリースでは Zipkin との互換性をサポートしていません。

1.3.5. 次のステップ

- Red Hat OpenShift Service on AWS 環境で [Red Hat OpenShift Service Mesh をインストールする準備](#) をします。

1.4. SERVICE MESH のデプロイメントモデル

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

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

1.4.1. クラスター全体 (シングルテナント) メッシュデプロイメントモデル

クラスター全体のデプロイメントには、クラスター全体のリソースを監視する Service Mesh Control Plane が含まれます。クラスター全体のリソースのモニタリングは、コントロールプレーンがすべての namespace にわたって単一のクエリーを使用して Istio および Kubernetes リソースを監視するという点で、Istio の機能によく似ています。その結果、クラスター全体のデプロイメントにより、API サーバーに送信されるリクエストの数が減少します。

Istio と同様に、クラスター全体のメッシュには、デフォルトで **istio-injection=enabled** namespace ラベルが付いた namespace が含まれます。このラベルを変更するには、**ServiceMeshMemberRoll** リソースの **spec.memberSelectors** フィールドを変更します。

1.4.2. マルチテナントデプロイメントモデル

Red Hat OpenShift Service Mesh は、デフォルトでマルチテナントとして設定される

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

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

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

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

指定の namespace または namespace 設定だけにワークロードをデプロイするチームパーミッションを付与できます。Service Mesh 管理者が **mesh-user** ロールを付与していると、ユーザーは **ServiceMeshMember** リソースを作成して namespace を **ServiceMeshMemberRoll** に追加できます。

1.4.2.1. クラスター全体のメッシュへの移行について

クラスター全体のメッシュでは、1つの **ServiceMeshControlPlane** (SMCP) がクラスター全体のすべての namespace を監視します。Red Hat OpenShift Service Mesh バージョン 2.5 以降を使用して、既存のクラスターをマルチテナントメッシュからクラスター全体のメッシュに移行できます。



注記

クラスターに複数の SMCP が必要な場合、クラスター全体のメッシュに移行できません。

デフォルトでは、クラスター全体のメッシュはクラスターを構成するすべての namespace を検出します。ただし、限られた namespace セットにアクセスするようにメッシュを設定できます。デフォルトでは、namespace はサイドカーインジェクションを受け取りません。どの namespace がサイドカーインジェクションを受け取るかを指定する必要があります。

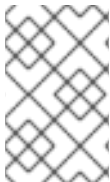
同様に、どの Pod がサイドカーインジェクションを受け取るかを指定する必要があります。サイドカーインジェクションを受け取る namespace に存在する Pod は、サイドカーインジェクションを継承しません。サイドカーインジェクションを namespace と Pod に適用するのは別の操作です。

クラスター全体のメッシュに移行するときに Istio のバージョンを変更した場合は、アプリケーションを再起動する必要があります。同じ Istio バージョンを使用する場合、アプリケーションプロキシはクラスター全体のメッシュの新規 SMCP に接続し、マルチテナントメッシュの場合と同じように動作します。

1.4.2.1.1. Web コンソールを使用してクラスター全体のメッシュに namespace を含めたり除外したりする

Red Hat OpenShift Service on AWS Web コンソールを使用して、クラスター全体のメッシュの **ServiceMeshControlPlane** リソースに検出セレクターを追加できます。ディスカバリーセレクターはコントロールプレーンが検出できる namespace を定義します。コントロールプレーンは、メッシュが

ら namespace を除外する検出セクターのいずれにも一致しない namespace を無視します。



注記

コントロールプレーンの namespace に Ingress または Egress ゲートウェイをインストールする場合は、コントロールプレーンの namespace を検出セクターに含める必要があります。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- **ServiceMeshControlPlane** リソースがデプロイされている。
- **cluster-admin** ロールを持つユーザーとしてログインしている。Red Hat OpenShift Dedicated を使用する場合は、**dedicated-admin** ロールを持つユーザーとしてログインします。

手順

1. Red Hat OpenShift Service on AWS 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** をクリックします。
5. コントロールプレーンの名前をクリックします。
6. **YAML** をクリックします。
7. YAML ファイルを変更して、**ServiceMeshControlPlane** リソースの **spec.meshConfig** フィールドに検出セクターが含まれるようにします。



注記

Istiod サービスが検出できる namespace を設定する場合は、他のメッシュには公開すべきではない機密サービスが含まれる可能性のある namespace を除外します。

以下の例では、**Istiod** サービスは **istio-discovery: enabled** のラベルが付いた namespace、または **info**、**httpbin**、または **istio-system** の名前を持つ namespace を検出します。

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  mode: ClusterWide
  meshConfig:
    discoverySelectors:
      - matchLabels:
          istio-discovery: enabled ❶
      - matchExpressions:
          - key: kubernetes.io/metadata.name ❷
```

```
operator: In
values:
- info
- httpbin
- istio-system
```

- 1 メッシュがラベル **istio-discovery:enabled** を含む namespace を検出するようにします。
- 2 メッシュが namespace **info**、**httpbin**、および **istio-system** を検出するようにします。

namespace が検出セクターのいずれかに一致する場合、メッシュは namespace を検出します。メッシュは、どの検出セクターにも一致しない namespace を除外します。

8. ファイルを保存します。

1.4.2.1.2. CLI を使用したクラスター全体のメッシュへの namespace の組み込みと除外

Red Hat OpenShift Service on AWS CLI を使用して、クラスター全体のメッシュの **ServiceMeshControlPlane** リソースに検出セクターを追加できます。ディスカバリーセクターはコントロールプレーンが検出できる namespace を定義します。コントロールプレーンは、メッシュから namespace を除外する検出セクターのいずれにも一致しない namespace を無視します。



注記

コントロールプレーンの namespace に Ingress または Egress ゲートウェイをインストールする場合は、コントロールプレーンの namespace を検出セクターに含める必要があります。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- **ServiceMeshControlPlane** リソースがデプロイされている。
- **cluster-admin** ロールを持つユーザーとしてログインしている。Red Hat OpenShift Dedicated を使用する場合は、**dedicated-admin** ロールを持つユーザーとしてログインします。

手順

1. Red Hat OpenShift Service on AWS CLI にログインします。
2. 次のコマンドを実行して、**ServiceMeshControlPlane** リソースを YAML ファイルとして開きます。

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

- 1 **<name>** は、**ServiceMeshControlPlane** リソースの名前を表します。

3. YAML ファイルを変更して、**ServiceMeshControlPlane** リソースの **spec.meshConfig** フィールドに検出セクターが含まれるようにします。



注記

Istiod サービスが検出できる namespace を設定する場合は、他のメッシュには公開すべきではない機密サービスが含まれる可能性のある namespace を除外します。

以下の例では、**Istiod** サービスは **istio-discovery: enabled** のラベルが付いた namespace、または **info**、**httpbin**、または **istio-system** の名前を持つ namespace を検出します。

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

- ❶ メッシュがラベル **istio-discovery:enabled** を含む namespace を検出するようにします。
- ❷ メッシュが namespace **info**、**httpbin**、および **istio-system** を検出するようにします。

namespace が検出セクターのいずれかに一致する場合、メッシュは namespace を検出します。メッシュは、どの検出セクターにも一致しない namespace を除外します。

4. ファイルを保存して、エディターを終了します。

1.4.2.1.3. Web コンソールを使用して、クラスター全体のメッシュでサイドカーインジェクションを受け取る namespace を定義する

デフォルトでは、Red Hat OpenShift Service Mesh Operator はメンバーセクターを使用して、サイドカーインジェクションを受け取る namespace を識別します。**ServiceMeshMemberRoll** リソースで定義されている **istio-injection=enabled** ラベルと一致しない namespace は、サイドカーインジェクションを受け取りません。



注記

検出セクターを使用してメッシュが検出できる namespace を決定しても、サイドカーインジェクションには影響しません。namespace の検出とサイドカーインジェクションの設定は別の操作です。

前提条件

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

- **ServiceMeshControlPlane** リソースを **mode: ClusterWide** アノテーションが指定してデプロイした。
- **cluster-admin** ロールを持つユーザーとしてログインしている。Red Hat OpenShift Dedicated を使用する場合は、**dedicated-admin** ロールを持つユーザーとしてログインします。

手順

1. Red Hat OpenShift Service on AWS 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 Member Roll** をクリックします。
5. **ServiceMeshMemberRoll** リソースをクリックします。
6. **YAML** をクリックします。
7. **inject** ラベルと一致するメンバーセクターを追加して、**ServiceMeshMemberRoll** リソースの **spec.memberSelectors** フィールドを変更します。次の例では、**istio-injection: enabled** を使用します。

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

- ❶ namespace がサイドカーインジェクションを確実に受け取るようにします。

8. ファイルを保存します。

1.4.2.1.4. CLI を使用して、クラスター全体のメッシュでサイドカーインジェクションを受け取る namespace を定義する

デフォルトでは、Red Hat OpenShift Service Mesh Operator はメンバーセクターを使用して、サイドカーインジェクションを受け取る namespace を識別します。**ServiceMeshMemberRoll** リソースで定義されている **istio-injection=enabled** ラベルと一致しない namespace は、サイドカーインジェクションを受け取りません。



注記

検出セクターを使用してメッシュが検出できる namespace を決定しても、サイドカーインジェクションには影響しません。namespace の検出とサイドカーインジェクションの設定は別の操作です。

前提条件

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

- **ServiceMeshControlPlane** リソースを **mode: ClusterWide** アノテーションが指定してデプロイした。
- **cluster-admin** ロールを持つユーザーとしてログインしている。Red Hat OpenShift Dedicated を使用する場合は、**dedicated-admin** ロールを持つユーザーとしてログインします。

手順

1. Red Hat OpenShift Service on AWS 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. **inject** ラベルと一致するメンバーセクターを追加して、**ServiceMeshMemberRoll** リソースの **spec.memberSelectors** フィールドを変更します。次の例では、**istio-injection: enabled** を使用します。

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

- ❶ namespace がサイドカーインジェクションを確実に受け取るようにします。

4. ファイルを保存して、エディターを終了します。

1.4.2.1.5. Web コンソールを使用してクラスター全体のメッシュから個々の Pod を除外する

Pod に **sidecar.istio.io/inject: true** アノテーションが適用されており、Pod が **ServiceMeshMemberRoll** リソースで定義されているラベルセクターまたはメンバーリストのいずれかに一致する namespace に存在する場合、Pod はサイドカーインジェクションを受け取ります。

Pod に **sidecar.istio.io/inject** アノテーションが適用されていない場合、サイドカーインジェクションを受け取ることができません。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- **ServiceMeshControlPlane** リソースを **mode: ClusterWide** アノテーションが指定してデプロイした。
- **cluster-admin** ロールを持つユーザーとしてログインしている。Red Hat OpenShift Dedicated を使用する場合は、**dedicated-admin** ロールを持つユーザーとしてログインします。

手順

1. Red Hat OpenShift Service on AWS Web コンソールにログインします。

2. **Workloads** → **Deployments** に移動します。
3. デプロイメントの名前をクリックします。
4. **YAML** をクリックします。
5. 次の例に示すように、YAML ファイルを変更して、サイドカーインジェクションを受け取るアプリケーションと受け取らないアプリケーションを1つずつデプロイします。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: 'true' ❶
    labels:
      app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
          - containerPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-without-sidecar
spec:
  selector:
    matchLabels:
      app: nginx-without-sidecar
  template:
    metadata:
      labels:
        app: nginx-without-sidecar ❷
  spec:
    containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
          - containerPort: 80
```

- ❶ この Pod には、**sidecar.istio.io/inject** アノテーションが適用されているため、サイドカーインジェクションを受け取ります。
- ❷ この Pod にはアノテーションがないため、サイドカーインジェクションを受け取りません。

6. ファイルを保存します。

1.4.2.1.6. CLI を使用してクラスター全体のメッシュから個々の Pod を除外する

Pod に **sidecar.istio.io/inject: true** アノテーションが適用されており、Pod が **ServiceMeshMemberRoll** リソースで定義されているラベルセクターまたはメンバーリストのいずれかに一致する namespace に存在する場合、Pod はサイドカーインジェクションを受け取ります。

Pod に **sidecar.istio.io/inject** アノテーションが適用されていない場合、サイドカーインジェクションを受け取ることができません。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- **ServiceMeshControlPlane** リソースを **mode: ClusterWide** アノテーションが指定してデプロイした。
- **cluster-admin** ロールを持つユーザーとしてログインしている。Red Hat OpenShift Dedicated を使用する場合は、**dedicated-admin** ロールを持つユーザーとしてログインします。

手順

1. Red Hat OpenShift Service on AWS CLI にログインします。
2. 次のコマンドを実行して、デプロイメントを編集します。

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

3. 次の例に示すように、YAML ファイルを変更して、サイドカーインジェクションを受け取るアプリケーションと受け取らないアプリケーションを1つずつデプロイします。

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



```

metadata:
  name: nginx-without-sidecar
spec:
  selector:
    matchLabels:
      app: nginx-without-sidecar
  template:
    metadata:
      labels:
        app: nginx-without-sidecar ❷
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80

```

- ❶ この Pod には、**sidecar.istio.io/inject** アノテーションが適用されているため、サイドカーインジェクションを受け取ります。
- ❷ この Pod にはアノテーションがないため、サイドカーインジェクションを受け取りません。

4. ファイルを保存します。

1.4.3. マルチテーマまたはフェデレーションされたデプロイメントモデル

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

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

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

1.5. SERVICE MESH と ISTIO の相違点

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

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

以下の機能は Service Mesh と Istio で異なります。

1.5.1.1. コマンドラインツール

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

1.5.1.2. インストールおよびアップグレード

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

Red Hat OpenShift Service Mesh は Service Mesh のカナリアアップグレードをサポートしません。

1.5.1.3. 自動的な挿入

アップストリームの Istio コミュニティインストールは、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をあらゆる Pod に自動的に挿入することではなく、プロジェクトにラベルを付けることなくアノテーションを使用して挿入をオプトインする必要があります。この方法は必要な権限が少なく、ビルダー Pod などの他の Red Hat OpenShift Service on AWS 機能と競合しません。自動挿入を有効にするには、サイドカーの自動挿入 セクションで説明されているとおり、**sidecar.istio.io/inject** ラベルまたはアノテーションを指定します。

表1.3 サイドカーインジェクションのラベルとアノテーションの設定

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

1.5.1.4. Istio ロールベースアクセス制御機能

Istio ロールベースアクセス制御機能 (RBAC) は、サービスへのアクセスを制御するために使用できるメカニズムを提供します。ユーザー名やプロパティ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できます。

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

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

アップストリーム Istio コミュニティの要求ヘッダーのマッチング例

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: httpbin-usernamepolicy
spec:
```

```

action: ALLOW
rules:
  - when:
    - key: 'request.regex.headers[username]'
      values:
        - "allowed.*"
selector:
  matchLabels:
    app: httpbin

```

1.5.1.5. OpenSSL

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

1.5.1.6. 外部ワークロード

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

1.5.1.7. 仮想マシンのサポート

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

1.5.1.8. コンポーネントの変更

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

1.5.1.9. Envoy フィルター

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

1.5.1.10. Envoy サービス

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

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

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



注記

デフォルトでは、Istio Container Network Interface (CNI) Pod は、Red Hat OpenShift Service on AWS ノード上に作成されます。特定のノードでの CNI Pod の作成を除外するには、**maistra.io/exclude-cni=true** ラベルをノードに適用します。このラベルを追加すると、以前にデプロイされた Istio CNI Pod がノードから削除されます。

1.5.1.12. グローバル mTLS 設定

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

1.5.1.13. ゲートウェイ

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

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



注記

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

1.5.1.14. マルチクラスター設定

マルチクラスター設定における Red Hat OpenShift Service Mesh のサポートは、複数のクラスターにわたる Service Mesh のフェデレーションに限定されます。

1.5.1.15. カスタム証明書署名要求 (CSR)

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

1.5.1.16. Istio ゲートウェイのルート

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

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

1.5.1.16.1. catch-all ドメイン

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

1.5.1.16.2. サブドメイン

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

1.5.1.16.3. トランスポート層セキュリティー

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

関連情報

- [自動ルート作成](#)

1.5.2. マルチテナントインストール

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

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

1.5.2.1. マルチテナンシーとクラスター全体のインストールの比較

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

ServiceMeshMemberRoll members リスト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に **NetworkPolicy** リソースを作成し、他のメンバーとコントロールプレーンからのすべての Pod への Ingress を許可することで、各メンバー

プロジェクトを設定して、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します。Service Mesh からメンバーを削除すると、この **NetworkPolicy** リソースがプロジェクトから削除されます。



注記

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

1.5.2.2. クラスタースコープのリソース

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

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

1.5.3. Kiali と Service Mesh

Red Hat OpenShift Service on AWS での Service Mesh を使用した Kiali のインストールは、複数の点でコミュニティの Kiali インストールとは異なります。以下のような変更が、問題の解決、追加機能の提供、Red Hat OpenShift Service on AWS 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る。
- Ingress はデフォルトで有効になっている。
- Kiali ConfigMap が更新されている。
- Kiali の ClusterRole 設定が更新されている。
- 変更は Service Mesh または Kiali Operator によって上書きされる可能性があるため、ConfigMap を編集しないでください。Kiali Operator が管理するファイルには、**kiali.io/** ラベルまたはアノテーションが付いています。Operator ファイルの更新は、**cluster-admin** 権限を持つユーザーに制限する必要があります。Red Hat OpenShift Dedicated を使用する場合には、**dedicated-admin** 権限のあるユーザーだけが Operator ファイルを更新できるようにする必要があります。

1.5.4. 分散トレースと Service Mesh

Red Hat OpenShift Service on AWS にサービスメッシュを使用して分散トレーシングプラットフォーム (Jaeger) をインストールすることは、コミュニティの Jaeger のインストールとはさまざまな点で異なります。以下のような変更が、問題の解決、追加機能の提供、Red Hat OpenShift Service on AWS へのデプロイ時の差異の処理を実行するために必要になることがあります。

- 分散トレースは、Service Mesh に対してデフォルトで有効になっています。

- Ingress は、Service Mesh に対してデフォルトで有効になっています。
- Zipkin ポート名が、(**http** から) **jaeger-collector-zipkin** に変更されています。
- Jaeger は、**production** または **streaming** デプロイメントオプションのいずれかを選択する際に、デフォルトでストレージに Elasticsearch を使用します。
- Istio のコミュニティーバージョンは、一般的な "トレース" ルートを提供します。Red Hat OpenShift Service Mesh は、Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator によってインストールされ、すでに OAuth によって保護されている 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。プロキシサイドカーは、Pod の Ingress および Egress トラフィックに関連するスパンを作成します。エージェントサイドカーは、アプリケーションによって出力されるスパンを受け取り、これらを Jaeger Collector に送信します。

1.6. SERVICE MESH のインストールの準備

Red Hat OpenShift Service Mesh をインストールする前に、Red Hat OpenShift Service on AWS にサブスクライブし、サポートされている設定で Red Hat OpenShift Service on AWS をインストールする必要があります。

1.6.1. 前提条件

- Red Hat アカウントにアクティブな Red Hat OpenShift Service on AWS サブスクリプションがある。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。
- Red Hat OpenShift Service on AWS のバージョンに一致する Red Hat OpenShift Service on AWS コマンドラインユーティリティ (**oc** クライアントツール) のバージョンをインストールし、それをパスに追加する。

Red Hat OpenShift Service Mesh のライフサイクルおよびサポートされるプラットフォームの詳細は、[サポートポリシー](#) を参照してください。

1.6.2. サポートされる構成

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

1.6.2.1. サポート対象のプラットフォーム

Red Hat OpenShift Service Mesh Operator は、複数のバージョンの **ServiceMeshControlPlane** リソースをサポートします。バージョン 2.6 の Service Mesh コントロールプレーンは、次のプラットフォームバージョンでサポートされています。

- Red Hat OpenShift Container Platform バージョン 4.10 以降
- Red Hat OpenShift Dedicated バージョン 4
- Azure Red Hat OpenShift (ARO) バージョン 4
- Red Hat OpenShift Service on AWS (ROSA)

1.6.2.2. サポートされない設定

明示的にサポート対象外とされているケースには、以下が含まれます。

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

1.6.2.3. サポートされるネットワーク設定

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

- OpenShift-SDN
- OVN-Kubernetes は、サポートされているすべてのバージョンの Red Hat OpenShift Service on AWS で利用できます。
- Red Hat OpenShift Service on AWS で認定され、さらに Service Mesh 適合テストに合格したサードパーティーの Container Network Interface (CNI) プラグイン。詳細は、[認定 OpenShift CNI プラグイン](#) を参照してください。

1.6.2.4. Service Mesh でサポートされる設定

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

1.6.2.5. Kiali のサポートされる設定

- Kiali コンソールは、Google Chrome、Microsoft Edge、Mozilla Firefox、または Apple Safari ブラウザーの最新の 2 つのリリースでのみサポートされています。
- **openshift** 認証戦略は、Kiali が Red Hat OpenShift Service Mesh (OSSM) とともにデプロイされている場合にサポートされる唯一の認証設定です。**openshift** 戦略は、Red Hat OpenShift Service on AWS の個人のロールベースのアクセス制御 (RBAC) ロールに基づいてアクセスを制御します。

1.6.2.6. 分散トレースのサポートされる設定

- サイドカーとしての Jaeger エージェントは、Jaeger でサポートされる唯一の設定です。デーモンセットとしての Jaeger はマルチテナントインストールまたは OpenShift Dedicated ではサポートされません。

1.6.2.7. サポート対象の WebAssembly モジュール

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

1.6.3. 次のステップ

- Red Hat OpenShift Service on AWS 環境に [Red Hat OpenShift Service Mesh をインストール](#) します。

1.7. OPERATOR のインストール

Red Hat OpenShift Service Mesh をインストールするには、まず Red Hat OpenShift Service Mesh Operator とオプションの Operator を Red Hat OpenShift Service on AWS にインストールします。次に、コントロールプレーンをデプロイするための **ServiceMeshControlPlane** リソースを作成します。



注記

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

前提条件

- [Red Hat OpenShift Service Mesh のインストールの準備](#) のプロセスを確認している。
- **cluster-admin** ロールを持つアカウントがある。

以下の手順では、Red Hat OpenShift Service on AWS に Red Hat OpenShift Service Mesh の基本的なインスタンスをインストールする方法を説明します。



重要

Red Hat OpenShift Service Mesh 2.5 以降、Red Hat OpenShift 分散トレーシング Platform (Jaeger) および OpenShift Elasticsearch Operator は非推奨となり、今後のリリースで削除される予定です。Red Hat は、現行リリースのライフサイクル中はこれらの機能のバグ修正とサポートを提供しますが、今後これらの機能に対する機能強化は行われません。Red Hat OpenShift 分散トレーシング Platform (Jaeger) の代わりに、Red Hat OpenShift 分散トレーシング Platform を使用することもできます。

1.7.1. Service Mesh Operator の概要

Red Hat OpenShift Service Mesh では、Red Hat OpenShift Service Mesh Operator を使用する必要があります。この Operator により、アプリケーションを構成するマイクロサービスの接続、保護、制御、監視が可能になります。他の Operator をインストールして、Service Mesh のエクスペリエンスを強化することもできます。



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。

次の Operator が必要です。

Red Hat OpenShift Service Mesh Operator

アプリケーションを構成するマイクロサービスの接続、保護、制御、監視を可能にします。また、Service Mesh コンポーネントのデプロイ、更新、削除を管理する **ServiceMeshControlPlane** リソースを定義および監視します。これはオープンソースの [Istio](#) プロジェクトに基づいています。

次の Operator はオプションです。

Red Hat が提供する Kiali Operator

Service Mesh の可観測性を提供します。これにより、単一のコンソールで設定を表示し、トラフィックを監視し、トレースの分析を実行できます。これはオープンソースの [Kiali](#) プロジェクトに基づいています。

Red Hat OpenShift 分散トレーシングプラットフォーム

複雑な分散システム内のトランザクションを監視およびトラブルシューティングするための分散トレーシングを提供します。これはオープンソースの [Grafana Tempo](#) プロジェクトに基づいています。

次のオプションの Operators は非推奨です。



重要

Red Hat OpenShift Service Mesh 2.5 以降、Red Hat OpenShift 分散トレーシング Platform (Jaeger) および OpenShift Elasticsearch Operator は非推奨となり、今後のリリースで削除される予定です。Red Hat は、現行リリースのライフサイクル中はこれらの機能のバグ修正とサポートを提供しますが、今後これらの機能に対する機能強化は行われません。Red Hat OpenShift 分散トレーシング Platform (Jaeger) の代わりに、Red Hat OpenShift 分散トレーシング Platform を使用することもできます。

Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)

複雑な分散システム内のトランザクションを監視およびトラブルシューティングするための分散トレーシングを提供します。これはオープンソースの [Jaeger](#) プロジェクトに基づいています。

OpenShift Elasticsearch Operator

分散トレーシング Platform (Jaeger) を使用したトレースおよびログ記録用のデータベースストレージを提供します。これはオープンソースの [Elasticsearch](#) プロジェクトに基づいています。

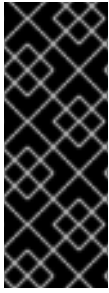
1.7.2. Operator のインストール

Red Hat OpenShift Service Mesh をインストールするには、Red Hat OpenShift Service Mesh Operator をインストールする必要があります。この手順を、インストールする追加の Operator ごとに繰り返し行ってください。

追加の Operators には以下のものがあります。

- Red Hat が提供する Kiali Operator
- Tempo Operator

非推奨となった追加の Operators は次のとおりです。



重要

Red Hat OpenShift Service Mesh 2.5 以降、Red Hat OpenShift 分散トレーシング Platform (Jaeger) および OpenShift Elasticsearch Operator は非推奨となり、今後のリリースで削除される予定です。Red Hat は、現行リリースのライフサイクル中はこれらの機能のバグ修正とサポートを提供しますが、今後これらの機能に対する機能強化は行われません。Red Hat OpenShift 分散トレーシング Platform (Jaeger) の代わりに、Red Hat OpenShift 分散トレーシング Platform を使用することもできます。

- Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)
- OpenShift Elasticsearch Operator



注記

OpenShift Logging の一部として OpenShift Elasticsearch Operator がすでにインストールされている場合は、OpenShift Elasticsearch Operator を再びインストールする必要はありません。Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator は、インストールされた OpenShift Elasticsearch Operator を使用して Elasticsearch インスタンスを作成します。

手順

1. **dedicated-admin** ロールを持つユーザーとして Red Hat OpenShift Service on AWS Web コンソールにログインします。
2. Red Hat OpenShift Service on AWS Web コンソールで、**Operator → OperatorHub** をクリックします。
3. Operator のフィルターボックスに名前を入力し、Red Hat バージョンの Operator を選択します。Operator のコミュニティーバージョンはサポートされていません。
4. **Install** をクリックします。
5. 各 Operator の **Install Operator** ページで、デフォルト設定を受け入れます。
6. **Install** をクリックします。Operator がインストールされるまで待機してから、同じ手順を繰り返して、次の Operator をインストールします。
 - Red Hat OpenShift Service Mesh Operator は、**openshift-operators** namespace にインストールされ、クラスター内のすべての namespace で使用できます。
 - Red Hat が提供する Kiali Operator は、**openshift-operators** namespace にインストールされ、クラスター内のすべての namespace で使用できます。
 - Tempo Operator は、**openshift-tempo-operator** namespace にインストールされ、クラスター内のすべての namespace で使用できます。
 - Red Hat OpenShift 分散トレーシング Platform (Jaeger) は、**openshift-distributed-tracing** 名前空間にインストールされ、クラスター内のすべての名前空間で使用できます。



重要

Red Hat OpenShift Service Mesh 2.5 以降、Red Hat OpenShift 分散トレーシング Platform (Jaeger) は非推奨となり、今後のリリースで削除される予定です。Red Hat は、現在のリリースのライフサイクル中にこの機能のバグ修正とサポートを提供しますが、この機能は今後、機能拡張を受け取らず、削除されます。Red Hat OpenShift 分散トレーシング Platform (Jaeger) の代わりに、Red Hat OpenShift 分散トレーシング Platform を使用することもできます。

- OpenShift Elasticsearch Operator は、**openshift-operators-redhat** namespace にインストールされ、クラスター内のすべての namespace で使用できます。



重要

Red Hat OpenShift Service Mesh 2.5 以降、OpenShift Elasticsearch Operator が非推奨になりました。これは今後のリリースで削除される予定です。Red Hat は、現在のリリースのライフサイクル中にこの機能のバグ修正とサポートを提供しますが、この機能は今後、機能拡張を受け取らず、削除されます。

検証

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

1.7.3. 次のステップ

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

1.8. SERVICEMESHCONTROLPLANE の作成

1.8.1. About ServiceMeshControlPlane

コントロールプレーンには、Istiod、Ingress および Egress Gateway、Kiali や Jaeger などの他のコンポーネントが含まれます。コントロールプレーンは、Service Mesh Operator やデータプレーンアプリケーションおよびサービスとは別の namespace にデプロイする必要があります。Red Hat OpenShift Service on AWS Web コンソールを使用するか、**oc** クライアントツールを使用してコマンドラインから **ServiceMeshControlPlane** (SMCP) の基本的なインストールをデプロイできます。



注記

この基本インストールは、デフォルトの Red Hat OpenShift Service on AWS 設定に基づいて設定されており、本番環境での使用を目的として設計されていません。このデフォルトのインストールを使用してインストールを確認し、環境に合わせて **ServiceMeshControlPlane** 設定を設定します。

Red Hat OpenShift Service on AWS で使用するためにコントロールプレーンをデプロイしている場合は、新しいプロジェクトの追加と Pod の開始を説明している Red Hat ナレッジベースの記事 [OpenShift service mesh operator Istio basic not starting due to authentication errors](#) を参照してください。

い。

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

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

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- Red Hat OpenShift Service on AWS Web コンソールに、**dedicated-admin** ロールを持つユーザーとしてログインしている。

手順

1. **cluster-admin** ロールを持つユーザーとして Red Hat OpenShift Service on AWS Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合は **dedicated-admin** ロールがあるアカウント。
2. **istio-system** という名前のプロジェクトを作成します。
 - a. **Home** → **Projects** に移動します。
 - b. **Create Project** をクリックします。
 - c. **Name** フィールドに **istio-system** と入力します。**ServiceMeshControlPlane** リソースは、マイクロサービスおよび Operator とは別に、**istio-system** プロジェクトにインストールする必要があります。
 - d. **Create** をクリックします。
3. **Operators** → **Installed Operators** に移動します。
4. Red Hat OpenShift Service Mesh Operator をクリックした後に、**Istio Service Mesh Control Plane** をクリックします。
5. **Istio Service Mesh Control Plane** タブで **Create ServiceMeshControlPlane** をクリックします。
 - a. 製品の最新バージョンで利用できる機能を利用するには、デフォルトの Service Mesh コントロールプレーンバージョンを受け入れます。コントロールプレーンのバージョンは、Operator のバージョンに関係なく利用可能な機能を判別します。
 - b. Red Hat OpenShift Service on AWS に必要な **spec.security.identity.type.ThirdParty** フィールドを追加します。
 - c. **Create** をクリックします。

Operator は、設定パラメーターに基づいて Pod、サービス、Service Mesh コントロールプレーンのコンポーネントを作成します。**ServiceMeshControlPlane** 設定は後で設定できます。

検証

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

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

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

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

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- OpenShift CLI (**oc**) へのアクセスがある。
- Red Hat OpenShift Service on AWS に、**dedicated-admin** ロールを持つユーザーとしてログインしている。

手順

1. **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
```

ServiceMeshControlPlane リソースは、マイクロサービスおよび Operator とは別に、**istio-system** プロジェクトにインストールする必要があります。

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

ServiceMeshControlPlane リソースの例

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

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

- 1 Red Hat OpenShift Service on AWS に必要な設定を指定します。

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

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

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

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

以下のような出力が表示されるはずです。

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

1.8.1.3. CLI を使用した SMCP インストールの検証

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

1. 前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- OpenShift CLI (**oc**) へのアクセスがある。
- Red Hat OpenShift Service on AWS に、**dedicated-admin** ロールを持つユーザーとしてログインしている。

手順

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

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

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

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

1.8.2. コントロールプレーンとクラスター全体のデプロイメントについて

クラスター全体のデプロイメントには、クラスター全体のリソースを監視する Service Mesh Control Plane が含まれます。クラスター全体のリソースのモニタリングは、コントロールプレーンがすべての namespace にわたって単一のクエリーを使用して Istio および Kubernetes リソースを監視するという点で、Istio の機能によく似ています。その結果、クラスター全体のデプロイメントにより、API サーバーに送信されるリクエストの数が減少します。

Red Hat OpenShift Service on AWS Web コンソールまたは CLI を使用して、クラスター全体のデプロイメント用に Service Mesh コントロールプレーンを設定できます。

1.8.2.1. Web コンソールを使用したクラスター全体のデプロイメント用のコントロールプレーンの設定

Red Hat OpenShift Service on AWS Web コンソールを使用して、クラスター全体のデプロイメント用に **ServiceMeshControlPlane** リソースを設定できます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前となります。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- Red Hat OpenShift Service on AWS に、**dedicated-admin** ロールを持つユーザーとしてログインしている。

手順

1. **istio-system** という名前のプロジェクトを作成します。
 - a. **Home** → **Projects** に移動します。
 - b. **Create Project** をクリックします。
 - c. **Name** フィールドに **istio-system** と入力します。**ServiceMeshControlPlane** リソースは、マイクロサービスおよび Operator とは異なるプロジェクトにインストールする必要があります。
これらの手順では、**istio-system** をサンプルとして使用します。Service Mesh コントロールプレーンは、サービスが含まれるプロジェクトから分離されている限り、任意のプロジェクトにデプロイできます。
 - d. **Create** をクリックします。
2. **Operators** → **Installed Operators** に移動します。
3. Red Hat OpenShift Service Mesh Operator をクリックした後に、**Istio Service Mesh Control Plane** をクリックします。
4. **Istio Service Mesh Control Plane** タブで **Create ServiceMeshControlPlane** をクリックします。
5. **YAML view** をクリックします。Service Mesh コントロールプレーンのバージョンは、Operator のバージョンに関係なく利用可能な機能を判別します。
6. **spec.mode** フィールドを変更し、**spec.security.identity.type.ThirdParty** フィールドを追加します。

ServiceMeshControlPlane リソースの例


```

apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.6
  mode: ClusterWide ❶
  security:
    identity:
      type: ThirdParty ❷
  tracing:
    type: Jaeger
    sampling: 10000
  policy:
    type: Istiod
  addons:
    grafana:
      enabled: true
    jaeger:
      install:
        storage:
          type: Memory
    kiali:
      enabled: true
    prometheus:
      enabled: true
  telemetry:
    type: Istiod

```

❶ リソースがクラスター全体のデプロイメント用であることを指定します。

❷ Red Hat OpenShift Service on AWS に必要な設定を指定します。

7. **Create** をクリックします。Operator は、設定パラメーターに基づいて Pod、サービス、Service Mesh コントロールプレーンのコンポーネントを作成します。**ServiceMeshMemberRoll** がデフォルト設定の一部として存在しない場合、Operator はそれも作成します。

検証

- コントロールプレーンが正しくインストールされていることを確認するには、次の手順を実行します。
 - a. **Istio Service Mesh Control Plane** タブをクリックします。
 - b. 新しい **ServiceMeshControlPlane** オブジェクトの名前をクリックします。
 - c. **Resources** タブをクリックして、Red Hat OpenShift Service Mesh コントロールプレーンリソース (Operator が作成し、設定したもの) を表示します。

1.8.2.2. CLI を使用したクラスター全体のデプロイメント用のコントロールプレーンの設定

CLI を使用して、クラスター全体のデプロイメント用に **ServiceMeshControlPlane** リソースを設定できます。この例では、**istio-system** は Service Mesh コントロールプレーンの namespace の名前です。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- OpenShift CLI (**oc**) にアクセスできる。
- Red Hat OpenShift Service on AWS に、**dedicated-admin** ロールを持つユーザーとしてログインしている。

手順

1. **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
```

2. 以下の例を使用して **istio-installation.yaml** という名前の **ServiceMeshControlPlane** ファイルを作成します。

ServiceMeshControlPlane リソースの例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.6
  mode: ClusterWide ①
  security:
    identity:
      type: ThirdParty ②
```

- ① リソースがクラスター全体のデプロイメント用であることを指定します。

- ② Red Hat OpenShift Service on AWS に必要な設定を指定します。

3. 次のコマンドを実行して、Service Mesh コントロールプレーンをデプロイします。

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

ここでは、以下ようになります。

<istio_installation.yaml>

ファイルへの完全パスを指定します。

検証

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

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

次の例のような出力が表示されるはずですが。

出力例

NAME	READY	STATUS	RESTARTS	AGE
grafana-b4d59bd7-mrgbr	2/2	Running	0	65m
istio-egressgateway-678dc97b4c-wrjqp	1/1	Running	0	108s
istio-ingressgateway-b45c9d54d-4qg6n	1/1	Running	0	108s
istiod-basic-55d78bbbcd-j5556	1/1	Running	0	108s
jaeger-67c75bd6dc-jv6k6	2/2	Running	0	65m
kiali-6476c7656c-x5msp	1/1	Running	0	43m
prometheus-58954b8d6b-m5std	2/2	Running	0	66m

1.8.2.3. クラスター全体のメッシュのメンバーロールのカスタマイズ

クラスター全体モードでは、**ServiceMeshControlPlane** リソースを作成すると、**ServiceMeshMemberRoll** リソースも作成されます。**ServiceMeshMemberRoll** リソースは、作成後に変更できます。リソースを変更すると、Service Mesh オペレーターはそのリソースを変更しなくなります。Red Hat OpenShift Service on AWS Web コンソールを使用して **ServiceMeshMemberRoll** リソースを変更する場合は、変更を上書きするプロンプトを受け入れます。

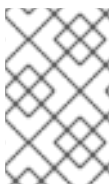
または、**ServiceMeshControlPlane** リソースをデプロイする前に **ServiceMeshMemberRoll** リソースを作成することもできます。**ServiceMeshControlPlane** リソースを作成するとき、Service Mesh Operator は **ServiceMeshMemberRoll** を変更しません。



注記

ServiceMeshMemberRoll リソース名には、**default** という名前を付け、**ServiceMeshControlPlane** リソースと同じプロジェクト namespace に作成する必要があります。

メッシュに namespace を追加するには2つの方法があります。**spec.members** リストで名前を指定して namespace を追加することも、ラベルに基づいて namespace を含めるか除外するように一連の namespace ラベルセレクターを設定することもできます。



注記

ServiceMeshMemberRoll リソースでメンバーがどのように指定されているかに関係なく、各 namespace に **ServiceMeshMember** リソースを作成してメッシュにメンバーを追加することもできます。

1.8.3. Kiali を使用した SMCP インストールの検証

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

1. 前提条件

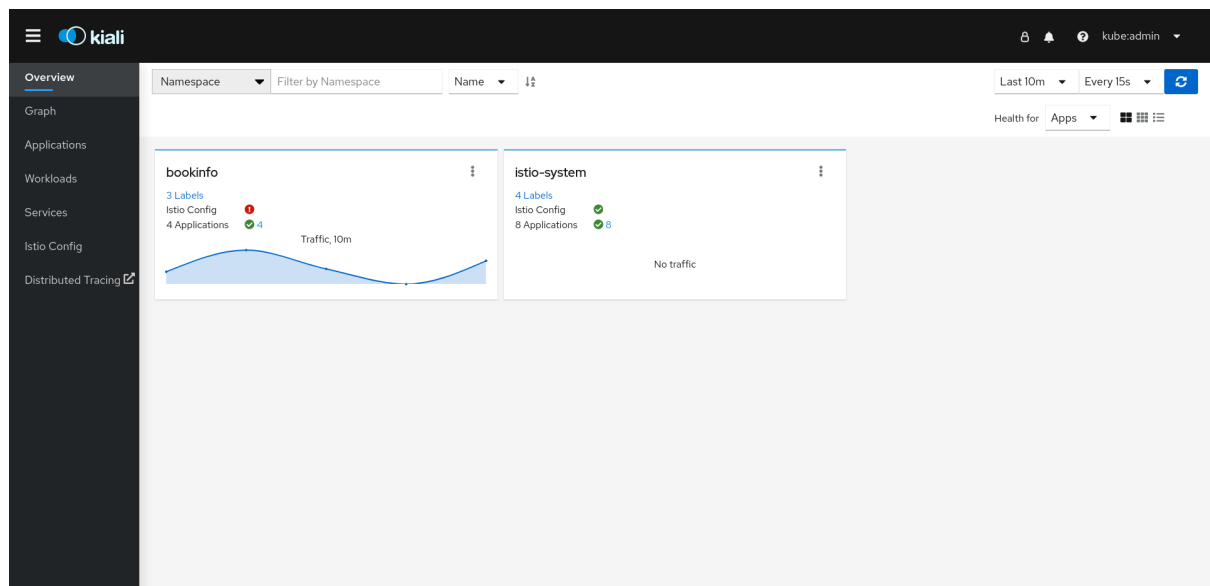
- Red Hat OpenShift Service Mesh Operator がインストールされている。
- OpenShift CLI (**oc**) へのアクセスがある。

- Red Hat OpenShift Service on AWS に、**dedicated-admin** ロールを持つユーザーとしてログインしている。


手順

- Red Hat OpenShift Service on AWS Web コンソールで、**Networking → Routes** に移動します。
- Routes** ページで、**Namespace** メニューから Service Mesh コントロールプレーンプロジェクトを選択します (例: **istio-system**)。
Location 列には、各ルートリンク先アドレスが表示されます。
- 必要に応じて、フィルターを使用して Kiali コンソールのルートを見つけます。ルートの **Location** をクリックしてコンソールを起動します。
- Log In With OpenShift** をクリックします。
初回の Kiali コンソールへのログイン時に、表示するパーミッションを持つ Service Mesh 内のすべての namespace を表示する **Overview** ページが表示されます。概要 ページに複数の namespace が表示されている場合は、Kiali は最初に正常性または検証に問題がある namespace を表示します。

図1.1 Kiali の概要ページ



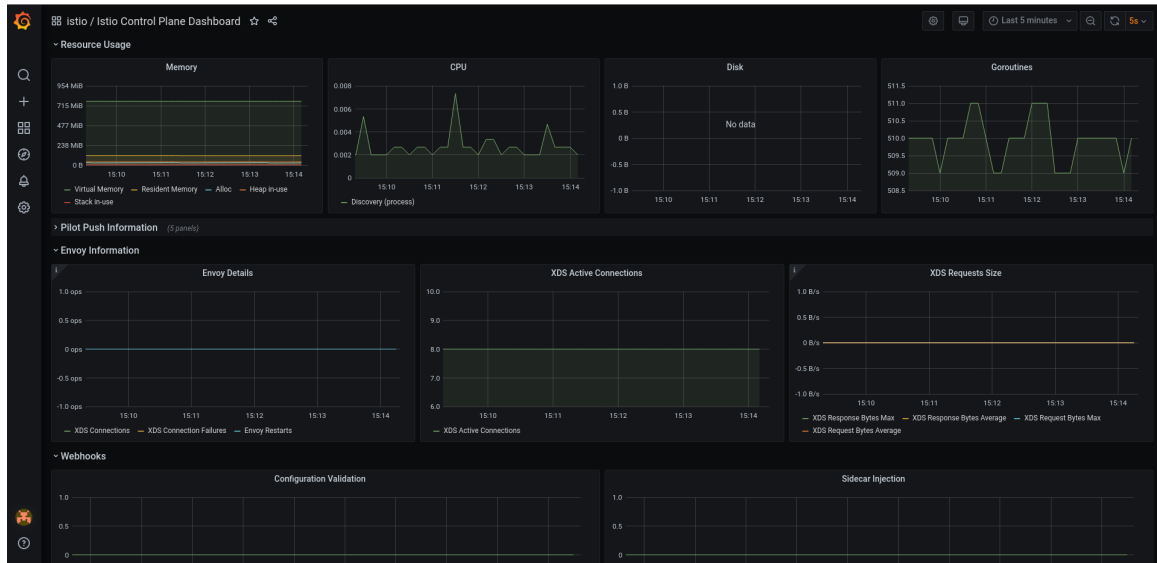
各 namespace のタイルには、ラベルの数、Istio Config の状態、アプリケーションの状態と数、namespace のトラフィックが表示されます。コンソールのインストールを検証中で、namespace がまだメッシュに追加されていないと、**istio-system** 以外のデータは表示されない可能性があります。

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

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

- Istio Wasm Exetension ダッシュボード

図1.2 Grafana Istio コントロールプレーンダッシュボード



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

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

- Service Mesh コントロールプレーンノードを表示するには、グラフ ページをクリックし、メニューから **ServiceMeshControlPlane** をインストールした **Namespace** を選択します (例: **istio-system**)。
 - 必要に応じて、**Display idle nodes** をクリックします。
 - グラフ ページの詳細は、グラフツアー リンクをクリックしてください。
 - メッシュトポロジーを表示するには、namespace メニューの Service Mesh メンバーロールから追加の **namespace** を1つまたは複数選択します。
- istio-system** namespace 内のアプリケーションのリストを表示するには、アプリケーション ページをクリックします。Kiali は、アプリケーションの状態を表示します。
 - 情報アイコンの上にマウスをかざすと、詳細 列に記載されている追加情報が表示されます。
- istio-system** namespace のワークロードのリストを表示するには、ワークロード ページをクリックします。Kiali は、ワークロードの状態を表示します。
 - 情報アイコンの上にマウスをかざすと、詳細 列に記載されている追加情報が表示されます。
- istio-system** namespace のサービスのリストを表示するには、サービス ページをクリックします。Kiali は、サービスと設定の状態を表示します。
 - 情報アイコンの上にマウスをかざすと、詳細 列に記載されている追加情報が表示されます。

10. **istio-system** namespace の Istio 設定オブジェクトのリストを表示するには、**Istio Config** ページをクリックします。Kiali は、設定の正常性を表示します。

- a. 設定エラーがある場合は、行をクリックすると、Kiali が設定ファイルを開き、エラーが強調表示されます。

1.8.4. 関連情報

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

1.8.5. 次のステップ

- プロジェクトを Service Mesh に追加してアプリケーションを利用可能にします。詳細は、[Service Mesh へのサービスの追加](#) を参照してください。

1.9. SERVICE MESH へのサービスの追加

プロジェクトにはサービスが含まれますが、そのプロジェクトを Service Mesh に追加していなければサービスは使用できません。

1.9.1. Service Mesh へのプロジェクトの追加

Operator をインストールして **ServiceMeshControlPlane** リソースを作成した後、1つ以上のプロジェクトを Service Mesh に追加します。



注記

Red Hat OpenShift Service on AWS では、プロジェクトは基本的に、プロジェクトで使用するユーザー ID の範囲などの追加のアノテーションを備えた Kubernetes 名前空間です。通常、Red Hat OpenShift Service on AWS Web コンソールではプロジェクトという用語が使用され、CLI では namespace という用語が使用されますが、この2つの用語は基本的に同義です。

Red Hat OpenShift Service on AWS Web コンソールまたは CLI を使用して、既存の Service Mesh にプロジェクトを追加できます。プロジェクトを Service Mesh に追加するには、次の3つの方法があります。

- **ServiceMeshMemberRoll** リソースでプロジェクト名を指定する方法。
- **ServiceMeshMemberRoll** リソースの **spec.memberSelectors** フィールドでラベルセクターを設定します。
- プロジェクトで **ServiceMeshMember** リソースを作成する方法。

最初の方法を使用する場合は、**ServiceMeshMemberRoll** リソースを作成する必要があります。

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

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

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

1.9.2.1. Web コンソールからのメンバーロールの作成

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

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。
- Service Mesh に追加する既存プロジェクトの一覧。

手順

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

1.9.2.2. CLI からのメンバーロールの作成

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

前提条件

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

手順

1. Red Hat OpenShift Service on AWS CLI にログインします。

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

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

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

3. プロジェクトをメンバーとして追加するには、以下の YAML の例を変更します。任意の数のプロジェクトを追加できますが、プロジェクトは1つの **ServiceMeshMemberRoll** リソースにしかなることができません。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前となります。

servicemeshmemberroll-default.yaml の例

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

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

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

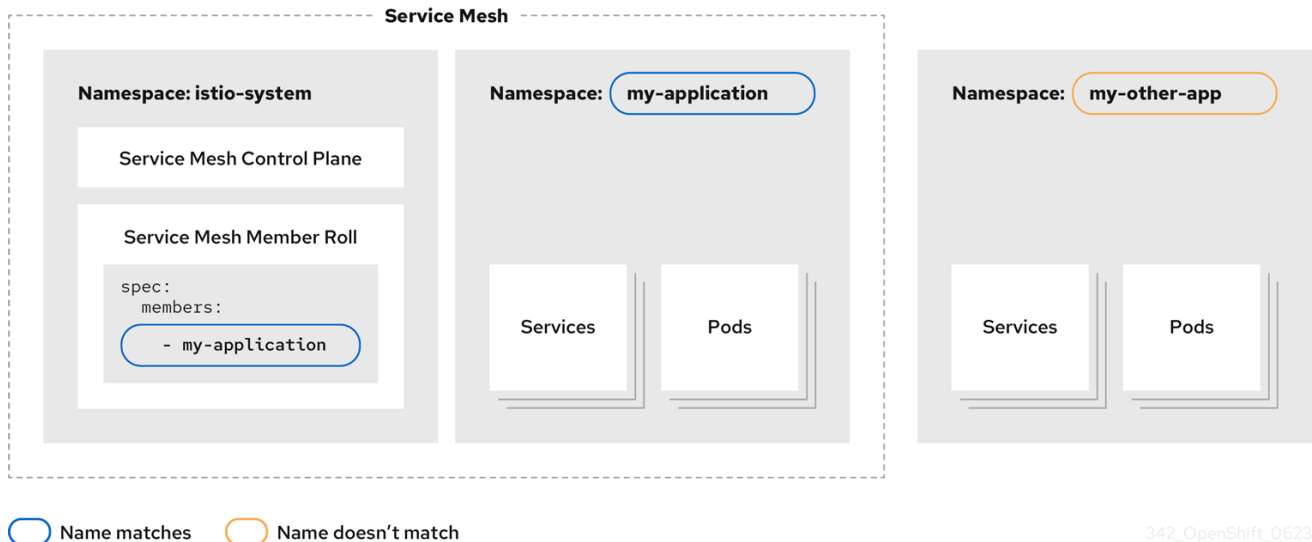
5. 以下のコマンドを実行して、**ServiceMeshMemberRoll** が正常に作成されていることを確認します。

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

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

1.9.3. ServiceMeshMemberRoll リソースを使用したプロジェクトの追加について

ServiceMeshMemberRoll リソースを使用するのが、プロジェクトを Service Mesh に追加する最も簡単な方法です。プロジェクトを追加するには、**ServiceMeshMemberRoll** リソースの **spec.members** フィールドにプロジェクト名を指定します。**ServiceMeshMemberRoll** リソースは、**ServiceMeshControlPlane** リソースによって制御されるプロジェクトを指定します。



342_OpenShift_0623



注記

この方法を使用してプロジェクトを追加するには、追加するプロジェクトの **servicemeshmemberrolls** 権限と **update pods** 権限をユーザーが持っている必要があります。

- Service Mesh に追加するアプリケーション、ワークロード、またはサービスがすでにある場合は、次を参照してください。
 - Web コンソールで **ServiceMeshMemberRoll** リソースを使用してメッシュにプロジェクトを追加または削除する
 - CLI で **ServiceMeshMemberRoll** リソースを使用してメッシュにプロジェクトを追加または削除する
- あるいは、Bookinfo というサンプルアプリケーションをインストールして **ServiceMeshMemberRoll** リソースに追加するには、Bookinfo サンプルアプリケーションのチュートリアルを参照してください。

1.9.3.1. Web コンソールで ServiceMeshMemberRoll リソースを使用してメッシュにプロジェクトを追加または削除する

Red Hat OpenShift Service on AWS Web コンソールで **ServiceMeshMemberRoll** リソースを使用して、メッシュにプロジェクトを追加または削除できます。任意の数のプロジェクトを追加できますが、プロジェクトは1つのメッシュに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。

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。
- メッシュに追加する、またはメッシュから削除するプロジェクトの名前。

手順

1. Red Hat OpenShift Service on AWS 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、リストから **ServiceMeshControlPlane** リソースがデプロイされているプロジェクトを選択します。たとえば、**istio-system** です。
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します (またはプロジェクトを削除して既存メンバーを削除します)。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1つの **ServiceMeshMemberRoll** リソースにしか属することができません。

servicemeshmemberroll-default.yaml の例

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

9. **Save** をクリックします。
10. **Reload** をクリックします。

1.9.3.2. CLI で ServiceMeshMemberRoll リソースを使用してメッシュにプロジェクトを追加または削除する

CLI で **ServiceMeshMemberRoll** リソースを使用して、1つ以上のプロジェクトをメッシュに追加できます。任意の数のプロジェクトを追加できますが、プロジェクトは1つのメッシュに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。

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。

- メッシュに追加する、またはメッシュから削除するプロジェクトの名前。
- OpenShift CLI (**oc**) へのアクセスがある。

手順

1. Red Hat OpenShift Service on AWS 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1つの **ServiceMeshMemberRoll** リソースにしか属することができません。

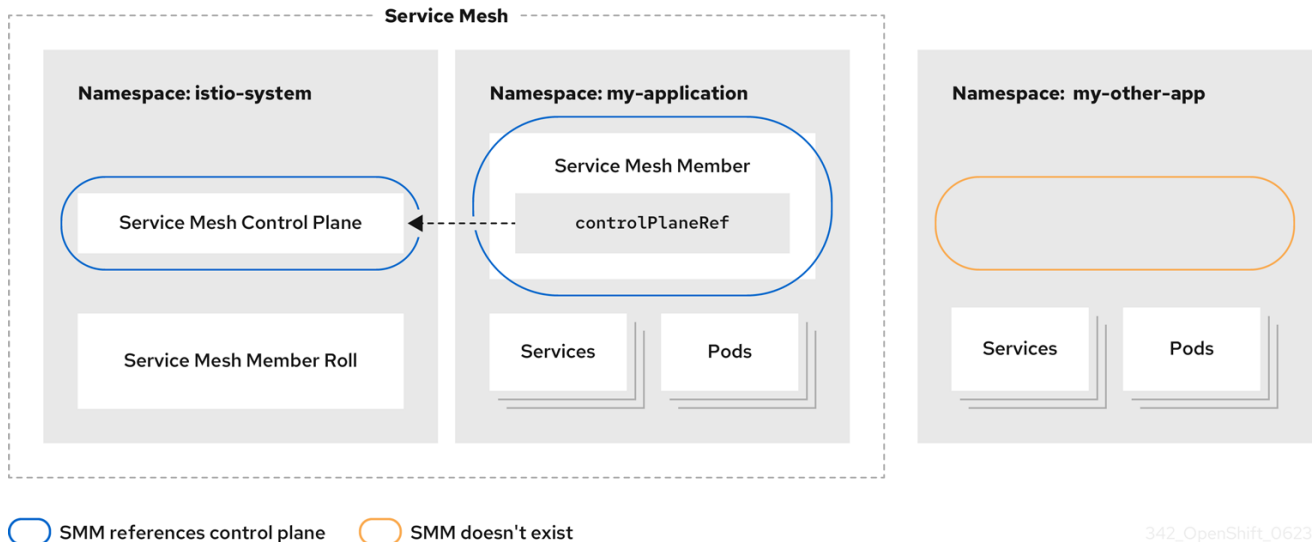
servicemeshmemberroll-default.yaml の例

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

4. ファイルを保存して、エディターを終了します。

1.9.4. ServiceMeshMember リソースを使用したプロジェクトの追加について

ServiceMeshMember リソースを使用すると、**ServiceMeshMemberRoll** リソースを変更せずにプロジェクトを Service Mesh に追加できます。プロジェクトを追加するには、Service Mesh に追加するプロジェクトに **ServiceMeshMember** リソースを作成します。Service Mesh Operator が **ServiceMeshMember** オブジェクトを処理すると、**ServiceMeshMemberRoll** リソースの **status.members** リストにプロジェクトが表示されます。次に、プロジェクトに存在するサービスがメッシュで利用可能になります。



メッシュ管理者は、各メッシュユーザーに **ServiceMeshMember** リソースの **ServiceMeshControlPlane** リソースを参照する権限を付与する必要があります。この権限を設定すると、メッシュユーザーが Service Mesh プロジェクトまたは **ServiceMeshMemberRoll** リソースへの直接アクセス権を持っていない場合でも、メッシュユーザーはプロジェクトをメッシュに追加できます。詳細は、Red Hat OpenShift Service Mesh メンバーの作成を参照してください。

1.9.4.1. Web コンソールで ServiceMeshMember リソースを使用してメッシュにプロジェクトを追加

Red Hat OpenShift Service on AWS Web コンソールで **ServiceMeshMember** リソースを使用して、1 つ以上のプロジェクトをメッシュに追加できます。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- ServiceMeshControlPlane** リソースの名前と、リソースが属するプロジェクトの名前はわかっています。
- メッシュに追加するプロジェクトの名前はわかっています。
- Service Mesh 管理者は、Service Mesh へのアクセスを明示的に付与する必要があります。管理者は、**RoleBinding** または **ClusterRoleBinding** を使用して **mesh-user** ロール をユーザーに割り当てて、メッシュにアクセスする権限を付与できます。詳細は、**Red Hat OpenShift Service Mesh** メンバーの作成 を参照してください。

手順

- Red Hat OpenShift Service on AWS Web コンソールにログインします。
- Operators** → **Installed Operators** に移動します。
- Project** メニューをクリックし、ドロップダウンリストからメッシュに追加するプロジェクトを選択します。たとえば、**istio-system** です。
- Red Hat OpenShift Service Mesh Operator をクリックします。
- Istio Service Mesh Member** タブをクリックします。

6. **Create ServiceMeshMember** をクリックします。
7. **ServiceMeshMember** のデフォルト名を許可します。
8. クリックして **ControlPlaneRef** を展開します。
9. **Namespace** フィールドで、**ServiceMeshControlPlane** リソースが属するプロジェクトを選択します。たとえば、**istio-system** です。
10. **Name** フィールドに、この namespace が属する **ServiceMeshControlPlane** リソースの名前を入力します。たとえば、**basic** です。
11. **Create** をクリックします。

検証

1. 次の手順を使用して、**ServiceMeshMember** リソースが作成され、プロジェクトがメッシュに追加されたことを確認します。
 - a. リソース名 (例: **default**) をクリックします。
 - b. 画面の最後に表示される **Conditions** セクションを表示します。
 - c. **Reconciled** および **Ready** の条件の **Status** が **True** であることを確認します。
Status が **False** の場合は、**Reason** 列および **Message** 列で詳細を確認してください。

1.9.4.2. CLI で ServiceMeshMember リソースを使用してメッシュにプロジェクトを追加する

CLI で **ServiceMeshMember** リソースを使用して、1つ以上のプロジェクトをメッシュに追加できます。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- **ServiceMeshControlPlane** リソースの名前と、それが属するプロジェクトの名前はわかっています。
- メッシュに追加するプロジェクトの名前はわかっています。
- Service Mesh 管理者は、Service Mesh へのアクセスを明示的に付与する必要があります。管理者は、**RoleBinding** または **ClusterRoleBinding** を使用して **mesh-user** ロール をユーザーに割り当てて、メッシュにアクセスする権限を付与できます。詳細は、**Red Hat OpenShift Service Mesh** メンバーの作成 を参照してください。

手順

1. Red Hat OpenShift Service on AWS CLI にログインします。
2. **ServiceMeshMember** マニフェストの YAML ファイルを作成します。マニフェストは、**istio-system** namespace にデプロイされた **ServiceMeshControlPlane** リソースが作成した Service Mesh に **my-application** プロジェクトを追加します。

```
apiVersion: maistra.io/v1
kind: ServiceMeshMember
metadata:
```

```

name: default
namespace: my-application
spec:
  controlPlaneRef:
    namespace: istio-system
    name: basic

```

3. YAML ファイルを適用して **ServiceMeshMember** リソースを作成します。

```
$ oc apply -f <file-name>
```

検証

- 次のコマンドを実行して、namespace がメッシュに含まれていることを確認します。 **READY** 列に **True** の値が表示されていることを確認します。

```
$ oc get smm default -n my-application
```

出力例

```

NAME      CONTROL PLANE    READY  AGE
default   istio-system/basic True    2m11s

```

- または、**ServiceMeshMemberRoll** リソースを表示して、**my-application** namespace が **ServiceMeshMemberRoll** リソースの **status.members** フィールドと **status.configuredMembers** フィールドに表示されることを確認します。

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

出力例

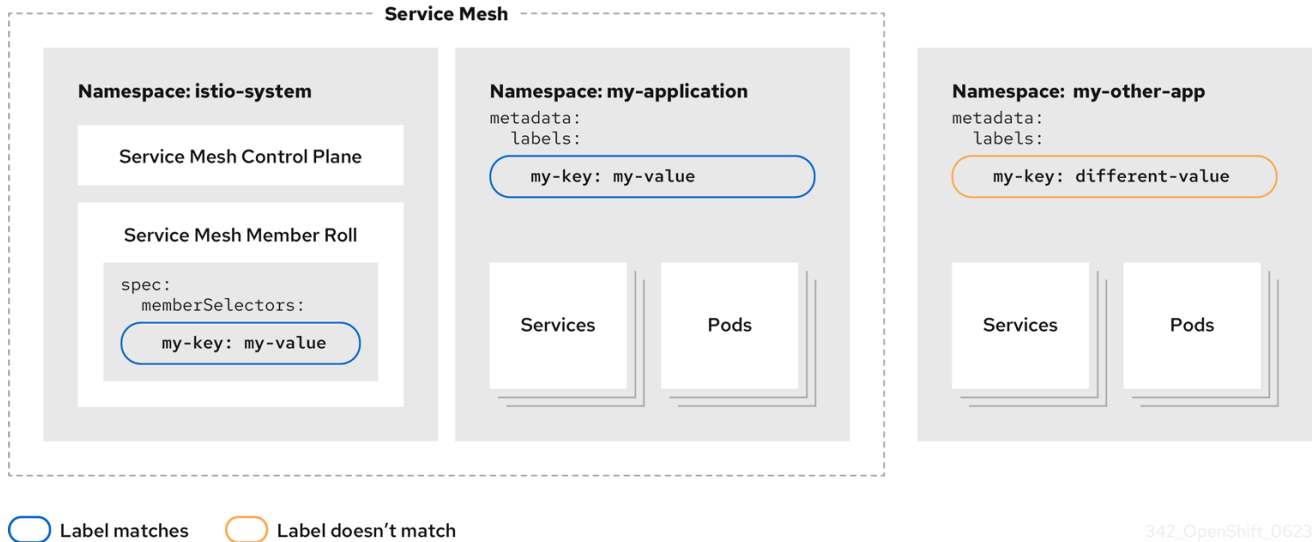
```

Name:      default
Namespace: istio-system
Labels:    <none>
# ...
Status:
# ...
  Configured Members:
    default
    my-application
# ...
  Members:
    default
    my-application

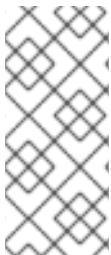
```

1.9.5. ラベルセクターを使用したプロジェクトの追加について

クラスター全体のデプロイメントの場合は、ラベルセクターを使用してプロジェクトをメッシュに追加できます。**ServiceMeshMemberRoll** リソースで指定されたラベルセクターを使用すると、Service Mesh Operator は、namespace ラベルに基づいてメッシュに namespace を追加またはメッシュから namespace を削除できます。単一のラベルセクターを指定するために使用できる他の標準的な Red Hat OpenShift Service on AWS リソースとは異なり、**ServiceMeshMemberRoll** リソースを使用して複数のラベルセクターを指定できます。



namespace のラベルが **ServiceMeshMemberRoll** リソースで指定されたセレクトアのいずれかに一致する場合、その namespace はメッシュに含まれます。



注記

Red Hat OpenShift Service on AWS では、プロジェクトは基本的に、プロジェクトで利用できるユーザー ID の範囲などの追加のアノテーションを備えた Kubernetes 名前空間です。通常、Red Hat OpenShift Service on AWS Web コンソールではプロジェクトという用語が使用され、CLI では 名前空間 という用語が使用されますが、この 2 つの用語は基本的に同義です。

1.9.5.1. Web コンソールでラベルセクターを使用してメッシュにプロジェクトを追加する

ラベルセクターを使用して、Red Hat OpenShift Service on AWS Web コンソールで Service Mesh にプロジェクトを追加できます。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- デプロイメントには既存の **ServiceMeshMemberRoll** リソースがあります。
- Red Hat OpenShift Service on AWS Web コンソールに、**dedicated-admin** ロールを持つユーザーとしてログインしている。

手順

- Operators → Installed Operators に移動します。
- Project メニューをクリックし、ドロップダウンリストから **ServiceMeshMemberRoll** リソースがデプロイされているプロジェクトを選択します。たとえば、istio-system です。
- Red Hat OpenShift Service Mesh Operator をクリックします。
- Istio Service Mesh Member Roll タブをクリックします。
- Create ServiceMeshMember Roll をクリックします。

6. **ServiceMeshMemberRoll** のデフォルト名を受け入れます。
7. **Labels** フィールドにキーと値のペアを入力して、Service Mesh に含める namespace を識別するラベルを定義します。プロジェクト namespace にセクターで指定されたラベルがある場合、プロジェクト namespace は Service Mesh に含まれます。両方のラベルを含める必要はありません。
たとえば、**mykey=myvalue** と入力すると、このラベルを持つすべての namespace がメッシュの一部として含まれます。セクターが一致を識別すると、プロジェクト namespace が Service Mesh に追加されます。

myotherkey=myothervalue と入力すると、このラベルを持つすべての namespace がメッシュの一部として含まれます。セクターが一致を識別すると、プロジェクト namespace が Service Mesh に追加されます。
8. **Create** をクリックします。

1.9.5.2. CLI でラベルセクターを使用してメッシュにプロジェクトを追加する

ラベルセクターを使用して、CLI でプロジェクトを Service Mesh に追加できます。

前提条件

- Red Hat OpenShift Service Mesh Operator がインストールされている。
- デプロイメントには既存の **ServiceMeshMemberRoll** リソースがあります。
- Red Hat OpenShift Service on AWS に、**dedicated-admin** ロールを持つユーザーとしてログインしている。

手順

1. Red Hat OpenShift Service on AWS CLI にログインします。
2. **ServiceMeshMemberRoll** リソースを編集します。

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

Service Mesh コントロールプレーンは、サービスが含まれるプロジェクトから分離されていれば、任意のプロジェクトにデプロイできます。

3. YAML ファイルを変更して、**ServiceMeshMemberRoll** リソースの **spec.memberSelectors** フィールドに namespace ラベルセクターを含めます。



注記

matchLabels フィールドを使用する代わりに、セクターで **matchExpressions** フィールドを使用することもできます。

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



```

memberSelectors: ❶
- matchLabels: ❷
  mykey: myvalue ❸
- matchLabels: ❹
  myotherkey: myothervalue ❺

```

- ❶ Service Mesh に含まれるプロジェクト namespace を識別するために使用されるラベルセクターが含まれます。プロジェクト namespace にセクターで指定されたラベルがある場合、プロジェクト namespace は Service Mesh に含まれます。プロジェクト namespace には両方のラベルを含める必要はありません。
- ❷ ❸ **mykey=myvalue** ラベルを持つすべての namespace を指定します。セクターが一致を識別すると、プロジェクト namespace が Service Mesh に追加されます。
- ❹ ❺ **myotherkey=myothervalue** ラベルを持つすべての namespace を指定します。セクターが一致を識別すると、プロジェクト namespace が Service Mesh に追加されます。

1.9.6. Bookinfo サンプルアプリケーション

Bookinfo サンプルアプリケーションを使用すると、Red Hat OpenShift Service on AWS 上の Red Hat OpenShift Service Mesh 2.6.7 のインストールをテストできます。

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

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

- **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。

1.9.6.1. Bookinfo アプリケーションのインストール

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

前提条件

- Red Hat OpenShift Service on AWS 4.1 以降がインストールされている。
- Red Hat OpenShift Service Mesh 2.6.7 がインストールされている。
- OpenShift CLI (**oc**) へのアクセスがある。
- Red Hat OpenShift Service on AWS に、**dedicated-admin** ロールを持つユーザーとしてログインしている。



注記

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

手順

1. **Home** → **Projects** をクリックします。
2. **Create Project** をクリックします。
3. **Project Name** として **info** 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、**info** プロジェクトを作成できます。

```
$ oc new-project info
```

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

Bookinfo ServiceMeshMemberRoll の例 (servicemeshmemberroll-default.yaml)

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
```

```
spec:
  members:
  - info
```

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

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

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

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

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

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

11. CLI で ``info`` 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 info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.6/samples/bookinfo/platform/kube/bookinfo.yaml
```

以下のような出力が表示されるはずです。

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

12. **info-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 info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.6/samples/bookinfo/networking/bookinfo-gateway.yaml
```

以下のような出力が表示されるはずです。

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

13. GATEWAY_URL パラメーターの値を設定します。

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

1.9.6.2. デフォルトの宛先ルールの追加

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

手順

- 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 info -f https://raw.githubusercontent.com/Maistra/istio/maistra-
2.6/samples/bookinfo/networking/destination-rule-all.yaml
```

- 相互 TLS を有効にしている場合:

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

以下のような出力が表示されるはずです。

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

1.9.6.3. Bookinfo インストールの検証

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

前提条件

- Red Hat OpenShift Service Mesh がインストールされている。
- Bookinfo サンプルアプリケーションのインストール手順を実行します。
- Red Hat OpenShift Service on AWS に、**dedicated-admin** ロールを持つユーザーとしてログインしている。

CLI からの手順

- 以下のコマンドですべての Pod が準備状態にあることを確認します。

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

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

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

2. 以下のコマンドを実行して、製品ページの URL を取得します。

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

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

Kiali Web コンソールからの手順

1. Kiali Web コンソールのアドレスを取得します。
 - a. Red Hat OpenShift Service on AWS Web コンソールにログインします。
 - b. **Networking** → **Routes** に移動します。
 - c. **Routes** ページで、**Namespace** メニューから Service Mesh コントロールプレーンプロジェクトを選択します (例: **istio-system**)。
Location 列には、各ルートのリンク先アドレスが表示されます。
 - d. Kiali の **場所** 列のリンクをクリックします。
 - e. **Log In With OpenShift** をクリックします。Kiali の **概要** 画面には、各プロジェクトの namespace のタイルが表示されます。
2. Kiali で、**グラフ** をクリックします。
3. **Namespace** リストから **info** を選択し、**Graph Type** リストから **App graph** を選択します。
4. **Display** メニューから **Display idle nodes** をクリックします。
これにより、定義されているが要求を受信または送信していないノードが表示されます。アプリケーションが適切に定義されていることを確認できますが、要求トラフィックは報告されていません。



- 期間 メニューを使用して、期間を延ばして、古いトラフィックを取得できるようにします。

- **Refresh Rate** メニューを使用して、トラフィックを頻繁に更新するか、まったく更新しないようにします。
5. **Services**、**Workloads** または **Istio Config** をクリックして、info コンポーネントのリストビューを表示し、それらが正常であることを確認します。

1.9.6.4. Bookinfo アプリケーションの削除

以下の手順で、Bookinfo アプリケーションを削除します。

前提条件

- Red Hat OpenShift Service on AWS 4.1 以降がインストールされている。
- Red Hat OpenShift Service Mesh 2.6.7 がインストールされている。
- OpenShift CLI (**oc**) へのアクセスがある。

1.9.6.4.1. Bookinfo プロジェクトの削除

手順


1. Red Hat OpenShift Service on AWS 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. **info** メニュー  をクリックしてから **Delete Project** をクリックします。
4. 確認ダイアログボックスに **info** と入力してから **Delete** をクリックします。
 - または、CLI を使用して次のコマンドを実行し、**info** プロジェクトを作成できます。

```
$ oc delete project info
```

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

手順

1. Red Hat OpenShift Service on AWS Web コンソールにログインします。
2. **Operators** → **Installed Operators** をクリックします。
3. **Project** メニューをクリックし、一覧から **istio-system** を選択します。
4. Red Hat OpenShift Service Mesh Operator の **Provided APIS** で、**Istio Service Mesh Member Roll** のリンクをクリックします。

5. **ServiceMeshMemberRoll** メニュー  をクリックし、**Edit Service Mesh Member Roll** を選択します。

6. デフォルトの Service Mesh Member Roll YAML を編集し、**members** 一覧から **info** を削除します。
 - または、CLI を使用して次のコマンドを実行し、**ServiceMeshMemberRoll** から **info** プロジェクトを削除できます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前となります。

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

7. **Save** をクリックして、Service Mesh Member Roll を更新します。

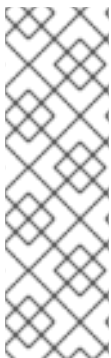
1.9.7. 次のステップ

- インストールプロセスを続行するには、[サイドカーインジェクションを有効化](#) を行う必要があります。

1.10. サイドカーコンテナの挿入の有効化

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

Bookinfo サンプルアプリケーションをインストールした場合は、アプリケーションがデプロイされ、インストール手順の一部としてサイドカーが注入されています。独自のプロジェクトとサービスを使用している場合は、Red Hat OpenShift Service on AWS にアプリケーションをデプロイします。



注記

Init Containers (Pod 内のアプリケーションコンテナの前に実行される特殊なコンテナ) によって開始されたトラフィックは、デフォルトでサービスメッシュの外に移動できません。Init Container が実行する、メッシュ外のネットワークトラフィック接続の確立を必要とするアクションはすべて失敗します。

Init Container をサービスに接続する方法の詳細は Red Hat ナレッジベースソリューション [initContainer in CrashLoopBackOff on pod with Service Mesh sidecar injected](#) を参照してください。

1.10.1. 前提条件

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

1.10.2. サイドカーコンテナの自動挿入の有効化

アプリケーションをデプロイする場合は、**deployment** オブジェクトで **spec.template.metadata.labels** の **sidecar.istio.io/inject** ラベルを **true** に設定して、インジェクションをオプトインする必要があります。オプトインにより、サイドカーの挿入が Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 機能に干渉しないようにします。

前提条件

- Service Mesh の一部である namespace と、サイドカーの自動注入が必要なデプロイメントを特定しておく。

手順

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

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

たとえば、**info** namespace の 'ratings-v1' マイクロサービスの **Deployment** YAML ファイルを表示するには、次のコマンドを使用して YAML 形式でリソースを表示します。

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

2. エディターでアプリケーションの **Deployment** YAML ファイルを開きます。
3. 次の例に示すように、**spec.template.metadata.labels.sidecar.istio/inject** を Deployment YAML ファイルに追加し、**sidecar.istio.io/inject** を **true** に設定します。

info deployment-ratings-v1.yaml からのスニペットの例

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



注記

自動サイドカーコンテナ注入を有効にする際の **annotations** パラメーターの使用は非推奨となり、**labels** パラメーターの使用に置き換えられます。

4. **Deployment** YAML ファイルを保存します。
5. ファイルをアプリケーションが含まれるプロジェクトに追加し直します。

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

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

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

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


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

以下に例を示します。

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

1.10.3. サイドカーインジェクションの検証

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

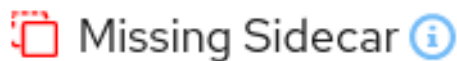
図1.3 サイドカーバッジがない



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

- App graph
- Versioned app graph
- Workload graph

図1.4 サイドカーアイコンがない



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

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

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

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

ワークロードの詳細 ページには、Envoy プロキシであるか、Envoy プロキシが注入されたワークロード用の **Envoy** タブもあります。このタブには、**Clusters**、**Listeners**、**Routes**、**Bootstrap**、**Config**、および **Metrics** のサブタブなど、組み込みの Envoy ダッシュボードが表示されます。

Envoy アクセスログを有効にする方法は、[トラブルシューティング](#) のセクションを参照してください。

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

1.10.4. アノテーションによるプロキシー環境変数の設定

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

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

injection-template.yaml の例

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



警告

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

1.10.5. サイドカープロキシーの更新

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

デプロイメントで自動のサイドカーコンテナ挿入を使用する場合は、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":{"kubectrl.kubernetes.io/restartedAt": ""`date -lseconds`"}}}}}'
```

デプロイメントで自動のサイドカーコンテナ挿入を使用しない場合は、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イメージを変更して Pod を再起動し、サイドカーコンテナを手動で更新する必要があります。

1.10.6. 次のステップ

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

- [セキュリティ](#)
- [トラフィック管理](#)
- [メトリクス、ログ、およびトレース](#)

1.11. SERVICE MESH のアップグレード

Red Hat OpenShift Service Mesh の最新機能にアクセスするには、最新バージョンである 2.6.7 にアップグレードしてください。

1.11.1. バージョニングについて

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

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

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

実行する更新のバージョンによって、アップグレードプロセスが異なります。

- パッチの更新: パッチのアップグレードは、Operator Lifecycle Manager (OLM) によって管理されます。Operator を更新すると自動的に発生します。
- マイナーアップグレード: マイナーアップグレードでは、最新の Red Hat OpenShift Service Mesh Operator バージョンに更新することと、**ServiceMeshControlPlane** リソースの **spec.version** 値を手動で変更することの両方が必要です。
- メジャーアップグレード: メジャーアップグレードでは、最新の Red Hat OpenShift Service Mesh Operator バージョンに更新することと、**ServiceMeshControlPlane** リソースの **spec.version** 値を手動で変更することの両方が必要です。メジャーアップグレードには後方互換性のない変更が含まれている可能性があるため、手動による追加の変更が必要になる場合があります。

1.11.1.2. Service Mesh のバージョンについて

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

- **Operator バージョン** - 最新の Operator バージョンは 2.6.7 です。Operator バージョン番号は、現在インストールされている Operator のバージョンのみを示します。Red Hat OpenShift Service Mesh Operator は Service Mesh コントロールプレーンの複数のバージョンをサポートするため、Operator のバージョンはデプロイされた **ServiceMeshControlPlane** リソースのバージョンを決定しません。



重要

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

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

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

1.11.2. アップグレードに関する考慮事項

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



警告

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

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

- **maistra-istio-operator** (Red Hat OpenShift Service Mesh) に設定された **maistra.io/** および **app.kubernetes.io/managed-by** ラベル
- **kiali.io/** (Kiali)

- jaegertracing.io/ (Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger))
- logging.openshift.io/ (Red Hat Elasticsearch)

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

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

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

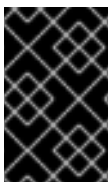
1.11.2.1. アップグレードに影響する可能性のある既知の問題

アップグレードに影響する可能性がある既知の問題には、次のものがあります。

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

1.11.3. Operator のアップグレード

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



重要

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

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

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

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

表1.4 更新チャネルおよび承認ストラテジーのインタラクション

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

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

Kiali および Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator をアップグレードすると、OLM 調整プロセスによってクラスターがスキャンされ、管理対象インスタンスが新しい Operator のバージョンにアップグレードされます。たとえば、Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator をバージョン 1.30.2 からバージョン 1.34.1 に更新する場合、Operator は実行中の分散トレーシング Platform (Jaeger) のインスタンスをスキャンし、それらも 1.34.1 にアップグレードします。

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

1.11.4. コントロールプレーンのアップグレード

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

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

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

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

1.11.4.1. バージョン 2.5 からバージョン 2.6 へのアップグレードの変更

1.11.4.1.1. Red Hat OpenShift 分散トレーシング Platform (Jaeger) のデフォルト設定の変更

このリリースでは、**ServiceMeshControlPlane** リソースの新しいインスタンスに対して、Red Hat OpenShift 分散トレーシング Platform (Jaeger) がデフォルトで無効になります。

ServiceMeshControlPlane リソースの既存のインスタンスを Red Hat OpenShift Service Mesh バージョン 2.6 に更新すると、分散トレーシング Platform (Jaeger) はデフォルトで有効のままになります。

Red Hat OpenShift Service Mesh 2.6 は、Red Hat OpenShift 分散トレーシング Platform (Jaeger) と OpenShift Elasticsearch Operator のサポートが含まれる最後のリリースです。分散トレーシング Platform (Jaeger) と OpenShift Elasticsearch Operator は両方とも次のリリースで削除されます。現在、分散トレーシング Platform (Jaeger) と OpenShift Elasticsearch Operator を使用している場合は、Red Hat OpenShift 分散トレーシング Platform と Red Hat build of OpenTelemetry に移行する必要があります。

1.11.4.1.2. Envoy サイドカーコンテナのデフォルト設定の変更

Pod の起動時間を短縮するために、Istio ではサイドカーコンテナに **startupProbe** がデフォルトで含まれるようになりました。Pod の readiness プローブは、Envoy サイドカーが起動するまで起動しません。

1.11.4.2. バージョン 2.4 から 2.5 へのアップグレードに伴う変更

1.11.4.2.1. Istio OpenShift Routing (IOR) のデフォルト設定の変更

Istio OpenShift Routing (IOR) のデフォルト設定が変更されました。この設定はデフォルトで無効になりました。

ServiceMeshControlPlane リソースの **spec.gateways.openshiftRoute** 仕様で、**enabled** フィールドを **true** に設定することで、IOR を使用できます。

```

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

```

1.11.4.2.2. Istio プロキシ同時実行設定の強化

デプロイメント間の一貫性を確保するために、Istio はプロキシコンテナに割り当てられた CPU 制限に基づいて **concurrency** パラメーターを設定するようになりました。たとえば、制限が 2500m の場合、**concurrency** パラメーターは 3 に設定されます。**concurrency** パラメーターを別の値に設定すると、Istio は CPU 制限を使用する代わりに、その値を使用してプロキシが実行するスレッドの数を設定します。

以前のリリースでは、このパラメーターのデフォルト設定は 2 でした。

1.11.4.3. バージョン 2.3 から 2.4 へのアップグレードに伴う変更

Service Mesh コントロールプレーンをバージョン 2.3 から 2.4 にアップグレードすると、次の動作変更が導入されます。

- Istio OpenShift Routing (IOR) のサポートは非推奨になりました。IOR 機能は引き続き有効ですが、将来のリリースでは削除される予定です。
- 次の暗号スイートはサポートされなくなり、クライアント側とサーバー側の TLS ネゴシエーションで使用される暗号のリストから削除されました。
 - ECDHE-ECDSA-AES128-SHA
 - ECDHE-RSA-AES128-SHA
 - AES128-GCM-SHA256
 - AES128-SHA
 - ECDHE-ECDSA-AES256-SHA
 - ECDHE-RSA-AES256-SHA
 - AES256-GCM-SHA384
 - AES256-SHA
 これらの暗号スイートのいずれかを使用するサービスにアクセスする必要があるアプリケーションは、プロキシが TLS 接続を開始すると接続に失敗します。

1.11.4.4. バージョン 2.2 から 2.3 へのアップグレードに伴う変更

Service Mesh コントロールプレーンをバージョン 2.2 から 2.3 にアップグレードすると、次の動作変更が導入されます。

- このリリースでは、**WasmPlugin** API を使用する必要があります。2.2 で非推奨化された **ServiceMeshExtension** API のサポートが廃止されました。**ServiceMeshExtension** API の使用中にアップグレードを試みると、アップグレードは失敗します。

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

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

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

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

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

アーキテクチャーの変更

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

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

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

以下に例を示します。

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  policy:
    type: Istiod
  telemetry:
    type: Istiod
  version: v2.6
```

動作上の変更

- **AuthorizationPolicy** の更新
 - PROXY プロトコルでは、**ipBlocks** および **notIpBlocks** を使用してリモート IP アドレスを指定する場合は、代わりに **remotelpBlocks** および **notRemotelpBlocks** を使用するよう設定を更新します。
 - ネストされた JSON Web Token(JWT) 要求のサポートが追加されました。
- **EnvoyFilter** の重大な変更

- **typed_config** を使用する必要があります。
- xDS v2 はサポート対象外になりました。
- フィルター名が非推奨になりました。
- 以前のバージョンのプロキシは、新しいプロキシから 1xx または 204 ステータスコードを受信すると、503 ステータスコードを報告する場合があります。

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

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

前提条件

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

手順

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

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

2. v2 **ServiceMeshControlPlane** リソース設定をチェックし、これが有効であることを確認します。
 - a. 以下のコマンドを実行して、**ServiceMeshControlPlane** リソースを v2 リソースとして表示します。

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

ヒント

Service Mesh コントロールプレーン設定をバックアップします。

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

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
```

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

- a. **Operators** → **Installed Operators** をクリックします。
- b. **ServiceMeshControlPlane** インスタンスを見つけます。
- c. **YAML view** を選択し、直前の例のように YAML ファイルのテキストを更新します。
- d. **Save** をクリックします。

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

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

前提条件

- Red Hat OpenShift Service Mesh 2.0 にアップグレードする前に Red Hat OpenShift Service on AWS 4.7 にアップグレードする必要がある。
- Red Hat OpenShift Service Mesh のバージョン 2.0 Operator が必要である。自動 アップグレードパスを選択した場合、Operator は最新情報を自動的にダウンロードします。ただし、Red Hat OpenShift Service Mesh バージョン 2.0 で機能を使用するために実行する必要がある手順があります。

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

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

手順

1. v1 **ServiceMeshControlPlane** リソース設定をチェックし、これが有効であることを確認します。
 - a. 以下のコマンドを実行して、**ServiceMeshControlPlane** リソースを v2 リソースとして表示します。


```
$ oc get smcp -o yaml
```
 - b. 無効なフィールドの情報は、出力の **spec.techPreview.errorred.message** フィールドを確認してください。
 - c. v1 リソースに無効なフィールドがある場合は、リソースは調整されず、v2 リソースとして編集することはできません。v2 フィールドへの更新はすべて、元の v1 設定で上書きされます。無効なフィールドを修正するには、リソースの v1 バージョンを置き換えるか、パッチを適用するか、編集できます。リソースを修正せずに削除することもできます。リソースの修正後に調整でき、リソースの v2 バージョンを変更または表示できます。
 - d. ファイルを編集してリソースを修正するには、**oc get** を使用してリソースを取得し、テキストファイルをローカルで編集し、リソースを編集したファイルに置き換えます。

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

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

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

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

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

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

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

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

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

- ServiceMeshControlPlane** を、開始点としての設定に関する情報が含まれる v2 のコントロールプレーンバージョンに変換します。

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

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

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

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

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

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

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

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

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

1.11.4.8.2. 2.0 ServiceMeshControlPlane の設定

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

1.11.4.8.2.1. アーキテクチャーの変更

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

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

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

1.11.4.8.2.2. アノテーションの変更

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

- **sidecar.maistra.io/proxyCPULimit** は **sidecar.istio.io/proxyCPULimit** に置き換えられました。ワークロードで **sidecar.maistra.io** アノテーションを使用していた場合は、代わりに **sidecar.istio.io** を使用するようにこれらのワークロードを変更する必要があります。
- **sidecar.maistra.io/proxyMemoryLimit** が **sidecar.istio.io/proxyMemoryLimit** に置き換えられました。
- **sidecar.istio.io/discoveryAddress** はサポートされなくなりました。また、デフォルトの検出アドレスが **pilot.<control_plane_namespace>.svc:15010** (または mtls が有効にされている場合はポート 15011) から **istiod-<smcp_name>.<control_plane_namespace>.svc:15012** に移行しました。
- ヘルスステータスポートは設定できなくなり、15021 にハードコーディングされています。* カスタムステータスポート (例: **status.sidecar.istio.io/port**) を定義している場合は、ワークロー

ドを v2.0 Service Mesh コントロールプレーンに移行する前にオーバーライドを削除する必要があります。ステータスポートを **0** に設定すると、readiness チェックを依然として無効にできません。

- Kubernetes シークレットリソースは、サイドカーのクライアント証明書を配信するために使用されなくなりました。証明書は Istiod の SDS サービスを介して配信されるようになりました。マウントされたシークレットに依存している場合、それらは v2.0 Service Mesh コントロールプレーンのワークロードで利用不可になります。

1.11.4.8.2.3. 動作上の変更

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

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

1.11.4.8.2.4. サポート対象外のリソースの移行情報

Policy (authentication.istio.io/v1alpha1)

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

相互 TLS

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

認証

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

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

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

ServiceMeshPolicy (maistra.io/v1)

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

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

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

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

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

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

ServiceMeshRbacConfig (maistra.io/v1)

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

1.11.4.8.2.5. Mixer プラグイン

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

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

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

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

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

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

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

1.11.4.8.2.6. 相互 TLS の変更

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

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

mTLS に関する詳細は、[相互トランスポート層セキュリティ \(mTLS\) の有効化](#) を参照してください。

1.11.4.8.2.6.1. その他の mTLS の例

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

Policy リソースの例

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

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

PeerAuthentication リソースの例


```

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

```

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

Policy リソースの例

```

apiVersion: authentication.istio.io/v1alpha1
kind: Policy
metadata:
  name: productpage-mTLS-with-JWT
  namespace: <namespace>
spec:
  targets:
    - name: productpage
  ports:
    - number: 9000
  peers:
    - mtls:
        origins:
          - jwt:
              issuer: "https://securetoken.google.com"
              audiences:
                - "productpage"
              jwksUri: "https://www.googleapis.com/oauth2/v1/certs"
              jwtHeaders:
                - "x-goog-iap-jwt-assertion"
            triggerRules:
              - excludedPaths:
                  - exact: /health_check
        principalBinding: USE_ORIGIN

```

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

PeerAuthentication リソースの例

```

#require mtls for productpage:9000
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: productpage-mTLS-with-JWT
  namespace: <namespace>

```

```

spec:
  selector:
    matchLabels:
      # this should match the selector for the "productpage" service
      app: productpage
  portLevelMtls:
    9000:
      mode: STRICT
---
#JWT authentication for productpage
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: productpage-mTLS-with-JWT
  namespace: <namespace>
spec:
  selector:
    matchLabels:
      # this should match the selector for the "productpage" service
      app: productpage
  jwtRules:
    - issuer: "https://securetoken.google.com"
      audiences:
        - "productpage"
      jwksUri: "https://www.googleapis.com/oauth2/v1/certs"
      fromHeaders:
        - name: "x-goog-iap-jwt-assertion"
---
#Require JWT token to access product page service from
#any client to all paths except /health_check
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: productpage-mTLS-with-JWT
  namespace: <namespace>
spec:
  action: ALLOW
  selector:
    matchLabels:
      # this should match the selector for the "productpage" service
      app: productpage
  rules:
    - to: # require JWT token to access all other paths
      - operation:
          notPaths:
            - /health_check
      from:
        - source:
            # if using principalBinding: USE_PEER in the Policy,
            # then use principals, e.g.
            # principals:
            # - "*"
          requestPrincipals:
            - "*"
    - to: # no JWT token required to access health_check

```

```
- operation:
  paths:
    - /health_check
```

1.11.4.8.3. 設定レシピ

これらの設定レシピを使用して、以下の項目を設定できます。

1.11.4.8.3.1. データプレーンでの相互 TLS

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

1.11.4.8.3.2. カスタム署名キー

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

1.11.4.8.3.3. トレーシング

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

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

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

```
spec:
  addons:
    jaeger:
      name: jaeger
      install:
        storage:
          type: Memory # or Elasticsearch for production mode
          memory:
            maxTraces: 100000
          elasticsearch: # the following values only apply if storage.type:=Elasticsearch
            storage: # specific storageclass configuration for the Jaeger Elasticsearch (optional)
              size: "100G"
              storageClassName: "storageclass"
            nodeCount: 3
            redundancyPolicy: SingleRedundancy
      runtime:
        components:
          tracing.jaeger: {} # general Jaeger specific runtime configuration (optional)
          tracing.jaeger.elasticsearch: #runtime configuration for Jaeger Elasticsearch deployment
            (optional)
          container:
```

```
resources:
  requests:
    memory: "1Gi"
    cpu: "500m"
  limits:
    memory: "1Gi"
```

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

1.11.4.8.3.4. 可視化

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

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

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

1.11.4.8.3.5. リソース使用状況とスケジューリング

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

コンポーネント	説明	サポート対象バージョン
セキュリティ	Citadel コンテナ	v1.0/1.1
galley	Galley コンテナ	v1.0/1.1
pilot	Pilot/Istiod コンテナ	v1.0/1.1/2.0
mixer	istio-telemetry および istio-policy コンテナ	v1.0/1.1
mixer.policy	istio-policy コンテナ	v2.0

コンポーネント	説明	サポート対象バージョン
mixer.telemetry	istio-telemetry コンテナ	v2.0
global.oauthproxy	各種アドオンと共に使用する oauth-proxy コンテナ	v1.0/1.1/2.0
sidecarInjectorWebhook	サイドカーインジェクター Webhook コンテナ	v1.0/1.1
tracing.jaeger	一般的な Jaeger コンテナ: すべての設定が適用されない可能性があります。Jaeger インストールの完全なカスタマイズは、既存の Jaeger リソースを Service Mesh コントロールプレーンの設定に指定することでサポートされます。	v1.0/1.1/2.0
tracing.jaeger.agent	Jaeger エージェントに固有の設定	v1.0/1.1/2.0
tracing.jaeger.allInOne	Jaeger allInOne に固有の設定	v1.0/1.1/2.0
tracing.jaeger.collector	Jaeger コレクターに固有の設定	v1.0/1.1/2.0
tracing.jaeger.elasticsearch	Jaeger elasticsearch デプロイメントに固有の設定	v1.0/1.1/2.0
tracing.jaeger.query	Jaeger クエリーに固有の設定	v1.0/1.1/2.0
prometheus	prometheus コンテナ	v1.0/1.1/2.0
kiali	Kiali コンテナ: Kiali インストールの完全なカスタマイズは、既存の Kiali リソースを Service Mesh コントロールプレーン設定に指定してサポートされます。	v1.0/1.1/2.0
grafana	Grafana コンテナ	v1.0/1.1/2.0
3scale	3scale コンテナ	v1.0/1.1/2.0
wasmExtensions.cacher	WASM 拡張キャッシュコンテナ	V2.0: テクノロジープレビュー

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

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

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

1.11.5. データプレーンのアップグレード

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

1.11.5.1. アプリケーションおよびワークロードの更新

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

デプロイメントのローリング更新を実行するには、以下のコマンドを使用します。

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

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

1.12. ユーザーおよびプロファイルの管理

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

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

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

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

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

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

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

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

この例では、**alice** のロールバインディングを追加します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  namespace: istio-system
  name: mesh-users
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: mesh-user
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: alice
```

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

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

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

1.12.2.1. ConfigMap の作成

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

前提条件

- Service Mesh Operator がインストールされ、検証されていること。
- **cluster-admin** ロールを持つアカウントがある。(Red Hat OpenShift Dedicated を使用する場合は) **dedicated-admin** ロールがあるアカウント。
- Operator デプロイメントの場所。
- OpenShift CLI (**oc**) へのアクセスがある。

手順

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

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

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

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

1.12.2.2. 適切なネットワークポリシーの設定

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

- Istio が適切に機能するには、Service Mesh 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。
- Service Mesh 外のサービスは、Service Mesh がない個別の namespace にデプロイします。
- Service Mesh 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 Red Hat OpenShift Service on AWS ルートは依然として機能します。

1.13. セキュリティー

Service Mesh 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は、Red Hat OpenShift Service Mesh を使用してそれらのサービス間の通信のセキュリティーをカスタマイズできます。Service Mesh のトラフィック管理機能と共に Red Hat OpenShift Service on AWS のインフラストラクチャーを使用すると、アプリケーションの複雑性を管理し、マイクロサービスのセキュリティーを確保できるようにします。

作業を始める前に

プロジェクトがある場合は、プロジェクトを [ServiceMeshMemberRoll リソース](#) に追加します。

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

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

Mutual Transport Layer Security (mTLS) は、二者が相互認証できるようにするプロトコルです。これは、一部のプロトコル (IKE、SSH) での認証のデフォルトモードであり、他のプロトコル (TLS) ではオプションになります。mTLS は、アプリケーションやサービスコードを変更せずに使用できます。TLS は、Service Mesh 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間で完全に処理されます。

デフォルトで、Red Hat OpenShift Service Mesh の mTLS は有効になっており、Permissive モードに設定されます。この場合、Service Mesh のサイドカーは、プレーンテキストのトラフィックと mTLS を使用して暗号化される接続の両方を受け入れます。厳密な mTLS を使用するように設定されたメッシュ内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 においてクライアントとサーバーの両方が互いの ID を検証できる必要があるため、これらのサービス間の通信が中断される可能性があります。ワークロードを Service Mesh に移行する間に Permissive モードを使用します。次に、メッシュ、namespace、またはアプリケーションで厳密な mTLS を有効にできます。

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

1.13.1.1. Service Mesh 全体での厳密な mTLS の有効化

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

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  version: v2.6
  security:
    dataPlane:
      mtls: true
```

Red Hat OpenShift Service on AWS Web コンソールを使用して mTLS を有効にすることもできます。

手順

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

1.13.1.1.1. 特定のサービスの受信接続用のサイドカーの設定

ポリシーを作成して、個別のサービスに mTLS を設定することもできます。

手順

1. 以下のサンプルを使用して YAML ファイルを作成します。

PeerAuthentication ポリシーの例 (policy.yaml)

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

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

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

**注記**

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

1.13.1.1.2. 送信接続用のサイドカーの設定

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

手順

1. 以下のサンプルを使用して YAML ファイルを作成します。

DestinationRule の例 (destination-rule.yaml)

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

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

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

1.13.1.1.3. 最小および最大のプロトコルバージョンの設定

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

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

表1.5 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

手順

1. Web コンソールにログインします。
2. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
3. **Operators** → **Installed Operators** をクリックします。
4. **Provided APIs** の **Service Mesh Control Plane** をクリックします。
5. **ServiceMeshControlPlane** リソースの名前 (例: **basic**) をクリックします。
6. **YAML** タブをクリックします。
7. 以下のコードスニペットを YAML エディターに挿入します。**minProtocolVersion** の値は、TLS バージョンの値に置き換えます。この例では、最小の TLS バージョンは **TLSv1_2** に設定されます。

ServiceMeshControlPlane スニペット

```
kind: ServiceMeshControlPlane
spec:
  security:
```

```
controlPlane:
  tls:
    minProtocolVersion: TLSv1_2
```

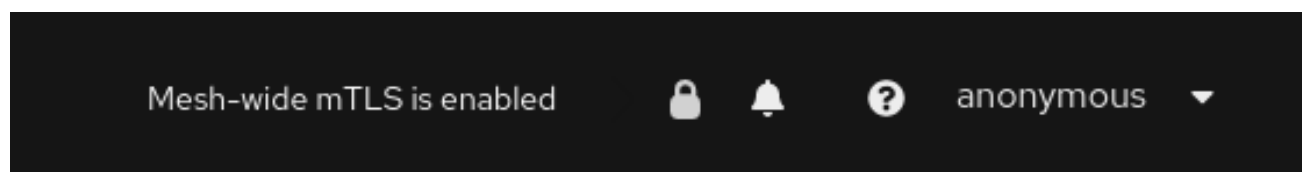
8. **Save** をクリックします。

9. **Refresh** をクリックし、変更が正しく更新されたことを確認します。

1.13.1.2. Kiali による暗号化の検証

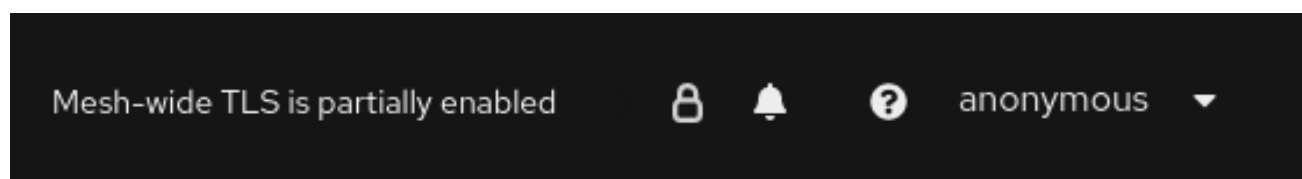
Kiali コンソールは、アプリケーション、サービス、ワークロードが mTLS 暗号化を有効にしているかどうかを検証するためのいくつかの方法を提供します。

図1.5 マストヘッドアイコンメッシュワイド mTLS が有効



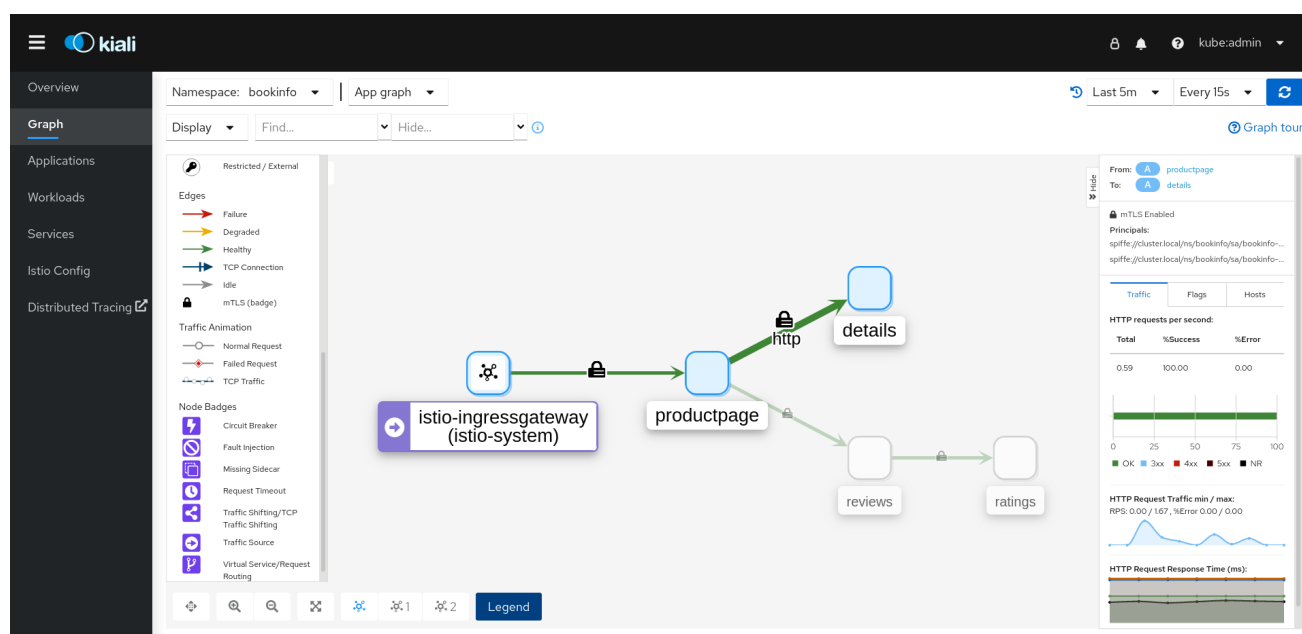
サービスメッシュ全体で厳密に mTLS が有効化されている場合、Kiali はマストヘッドの右側にロックアイコンを表示します。これは、メッシュ内のすべての通信に mTLS が使用されていることを意味します。

図1.6 マストヘッドアイコンメッシュワイド mTLS が一部有効



メッシュが **PERMISSIVE** モードに設定されているか、メッシュ全体の mTLS 設定にエラーが発生している場合、Kiali は中空ロックアイコンを表示します。

図1.7 セキュリティバッジ



Graph ページには、mTLS が有効であることを示すために、グラフの端に **Security** バッジを表示する

オプションがあります。グラフにセキュリティーバッジを表示するには、**Display** メニューの **Show Badges** で **Security** チェックボックスをオンにします。エッジにロックアイコンが表示されている場合は、mTLS が有効なリクエストが少なくとも1つ存在することを意味します。mTLS と non-mTLS の両方のリクエストがある場合、サイドパネルには mTLS を使用するリクエストのパーセンテージが表示されます。

Applications Detail Overview ページでは、mTLS が有効なリクエストが1つ以上存在するグラフの端に **Security** アイコンが表示されます。

Workloads Detail Overview ページでは、mTLS が有効なリクエストが1つ以上存在するグラフの端に **Security** アイコンが表示されます。

Services Detail Overview ページでは、mTLS が有効なリクエストが1つ以上存在するグラフの端に **Security** アイコンが表示されます。また、Kiali では、mTLS を設定したポートの横の **Network** セクションにロックアイコンが表示されることに注意してください。

1.13.2. ロールベースアクセス制御 (RBAC) の設定

ロールベースアクセス制御 (RBAC) オブジェクトは、ユーザーまたはサービスがプロジェクト内で所定のアクションを実行することが許可されるかどうかを決定します。メッシュでワークロードのメッシュ全体、namespace 全体、およびワークロード全体のアクセス制御を定義できます。

RBAC を設定するには、アクセスを設定する namespace で **AuthorizationPolicy** リソースを作成します。メッシュ全体のアクセスを設定する場合は、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を使用します。

たとえば、RBAC を使用して以下を実行するポリシーを作成できます。

- プロジェクト内通信を設定します。
- デフォルト namespace のすべてのワークロードへの完全アクセスを許可または拒否します。
- ingress ゲートウェイアクセスを許可または拒否します。
- アクセスにはトークンが必要です。

認証ポリシーには、セクター、アクション、およびルールのリストが含まれます。

- **selector** フィールドは、ポリシーのターゲットを指定します。
- **action** フィールドは、要求を許可または拒否するかどうかを指定します。
- **rules** フィールドは、アクションをトリガーするタイミングを指定します。
 - **from** フィールドは、要求元の制約を指定します。
 - **to** フィールドは、要求のターゲットおよびパラメーターの制約を指定します。
 - **when** フィールドは、ルールを適用する追加の条件を指定します。

手順

1. **AuthorizationPolicy** リソースを作成します。以下の例は、ingress-policy **AuthorizationPolicy** を更新して、IP アドレスが Ingress ゲートウェイにアクセスすることを拒否するリソースを示しています。

```
apiVersion: security.istio.io/v1beta1
```

```

kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: DENY
  rules:
  - from:
    - source:
      ipBlocks: ["1.2.3.4"]

```

- リソースを作成した後に以下のコマンドを実行して、namespace にリソースを作成します。
namespace は、**AuthorizationPolicy** リソースの **metadata.namespace** フィールドと一致する必要があります。

```
$ oc create -n istio-system -f <filename>
```

次のステップ

その他の一般的な設定は、以下の例を参照してください。

1.13.2.1. プロジェクト内通信の設定

AuthorizationPolicy を使用して Service Mesh コントロールプレーンを設定し、メッシュまたはメッシュ内のサービスとの通信トラフィックを許可したり、拒否したりできます。

1.13.2.1.1. namespace 外のサービスへのアクセス制限

以下の **AuthorizationPolicy** リソースの例を使用して、**info** namespace がないソースからの要求を拒否できます。

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: httpbin-deny
  namespace: info
spec:
  selector:
    matchLabels:
      app: httpbin
      version: v1
  action: DENY
  rules:
  - from:
    - source:
      notNamespaces: ["info"]

```

1.13.2.1.2. allow-all およびデフォルトの deny-all 認可ポリシーの作成

以下の例は、**info** namespace のすべてのワークロードへの完全なアクセスを許可する allow-all 認可ポリシーを示しています。

-

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-all
  namespace: info
spec:
  action: ALLOW
  rules:
  - {}

```

以下の例は、**info** namespace のすべてのワークロードへのアクセスを拒否するポリシーを示しています。

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: info
spec:
  {}

```

1.13.2.2. ingress ゲートウェイへのアクセスの許可または拒否

認証ポリシーを設定し、IP アドレスに基づいて許可または拒否リストを追加できます。

```

apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: ALLOW
  rules:
  - from:
    - source:
      ipBlocks: ["1.2.3.4", "5.6.7.0/24"]

```

1.13.2.3. JSON Web トークンを使用したアクセスの制限

JSON Web Token (JWT) を使用してメッシュにアクセスできる内容を制限できます。認証後に、ユーザーまたはサービスはそのトークンに関連付けられたルート、サービスにアクセスできます。

ワークロードでサポートされる認証方法を定義する **RequestAuthentication** リソースを作成します。以下の例では、<http://localhost:8080/auth/realms/master> で実行される JWT を受け入れます。

```

apiVersion: "security.istio.io/v1beta1"
kind: "RequestAuthentication"
metadata:
  name: "jwt-example"
  namespace: info
spec:

```

```

selector:
  matchLabels:
    app: httpbin
jwtRules:
- issuer: "http://localhost:8080/auth/realms/master"
  jwksUri: "http://keycloak.default.svc:8080/auth/realms/master/protocol/openid-connect/certs"

```

次に、同じ namespace に **AuthorizationPolicy** リソースを作成し、作成した **RequestAuthentication** リソースと連携させます。以下の例では、要求を **httpbin** ワークロードに送信する際に、JWT は **Authorization** ヘッダーになければなりません。

```

apiVersion: "security.istio.io/v1beta1"
kind: "AuthorizationPolicy"
metadata:
  name: "frontend-ingress"
  namespace: info
spec:
  selector:
    matchLabels:
      app: httpbin
  action: DENY
  rules:
  - from:
    - source:
        notRequestPrincipals: [""]

```

1.13.3. 暗号化スイートおよび ECDH 曲線の設定

暗号化スイートおよび ECDH 曲線 (Elliptic-curve Diffie-Hellman) は、Service Mesh のセキュリティーを保護するのに役立ちます。暗号化スイートのコンマ区切りの一覧を **spec.security.controlplane.tls.cipherSuites** を使用して定義し、ECDH 曲線を **ServiceMeshControlPlane** リソースの **spec.security.controlplane.tls.ecdhCurves** を使用して定義できます。これらの属性のいずれかが空の場合は、デフォルト値が使用されます。

Service Mesh が TLS 1.2 以前のバージョンを使用する場合は、**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。

コンマ区切りのリストに暗号化スイートを優先度順に設定します。たとえば、**ecdhCurves: CurveP256, CurveP384** は、**CurveP256** を **CurveP384** よりも高い優先順位として設定します。



注記

暗号化スイートを設定する場合は、**TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256** または **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256** のいずれかを追加する必要があります。HTTP/2 のサポートには、1つ以上の以下の暗号スイートが必要です。

サポートされている暗号化スイートは以下になります。

- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA

サポートされる ECDH 曲線は以下のとおりです。

- CurveP256
- CurveP384
- CurveP521
- X25519

1.13.4. JSON Web キーセットリゾルバー認証局の設定

ServiceMeshControlPlane (SMCP) 仕様から独自の JSON Web Key Sets (JWKS) リゾルバー認証局 (CA) を設定できます。

手順

1. **ServiceMeshControlPlane** 仕様ファイルを編集します。

```
$ oc edit smcp <smcp-name>
```

2. 次の例に示すように、**ServiceMeshControlPlane** 仕様で **mtls** フィールドの値を **true** に設定して、データプレーンの **mtls** を有効にします。

```
spec:
  security:
```

```

dataPlane:
  mtls: true # enable mtls for data plane
  # JWKSResolver extra CA
  # PEM-encoded certificate content to trust an additional CA
  jwksResolverCA: |
    -----BEGIN CERTIFICATE-----
    [...]
    [...]
    -----END CERTIFICATE-----
...

```

3. 変更を保存します。Red Hat OpenShift Service on AWS が変更を自動的に適用します。

pilot-jwks-cacerts-[<SMCP name>](#) などの **ConfigMap** が CA **.pem data** を使用して作成されます。

ConfigMap **pilot-jwks-cacerts-[<SMCP name>](#)** の例

```

kind: ConfigMap
apiVersion: v1
data:
  extra.pem: |
    -----BEGIN CERTIFICATE-----
    [...]
    [...]
    -----END CERTIFICATE-----

```

1.13.5. 外部認証局キーおよび証明書の追加

デフォルトで、Red Hat OpenShift Service Mesh は自己署名ルート証明書およびキーを生成し、それらを使用してワークロード証明書に署名します。ユーザー定義の証明書およびキーを使用して、ユーザー定義のルート証明書を使用してワークロード証明書に署名することもできます。このタスクは、証明書およびキーを Service Mesh にプラグインするサンプルを示しています。

前提条件

- 相互 TLS を有効にして Red Hat OpenShift Service Mesh をインストールし、証明書を設定する。
- この例では、[Maistra リポジトリ](#) からの証明書を使用します。実稼働環境の場合は、認証局から独自の証明書を使用します。
- Bookinfo サンプルアプリケーションをデプロイして以下の手順で結果を確認しておく。
- OpenSSL は、証明書を検証するために必要です。

1.13.5.1. 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。対応する証明書ごとに、以下のファイル名をそのまま使用する必要があります。CA 証明書は **ca-cert.pem** と呼ばれ、キーは **ca-key.pem** であり、**ca-cert.pem** を署名するルート証明書は **root-cert.pem** と呼ばれます。ワークロードで中間証明書を使用する場合は、**cert-chain.pem** ファイルでそれらを指定する必要があります。

1. [Maistra リポジトリ](#) のサンプル証明書をローカルに保存し、**<path>** を証明書へのパスに置き換えます。

2. **cacert** という名前のシークレットを作成します。これには、入力ファイルの **ca-cert.pem**、**ca-key.pem**、**root-cert.pem** および **cert-chain.pem** が含まれます。

```
$ oc create secret generic cacerts -n istio-system --from-file=<path>/ca-cert.pem \
--from-file=<path>/ca-key.pem --from-file=<path>/root-cert.pem \
--from-file=<path>/cert-chain.pem
```

3. **ServiceMeshControlPlane** リソースで、**spec.security.dataPlane.mtls true** を **true** に設定し、以下の例のように **certificateAuthority** フィールドを設定します。デフォルトの **rootCADir** は **/etc/cacerts** です。キーおよび証明書がデフォルトの場所にマウントされている場合は、**privateKey** を設定する必要はありません。Service Mesh は、secret-mount ファイルから証明書およびキーを読み取ります。

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  security:
    dataPlane:
      mtls: true
  certificateAuthority:
    type: Istiod
    istiod:
      type: PrivateKey
      privateKey:
        rootCADir: /etc/cacerts
```

4. **cacert** シークレットを作成/変更/削除した後に、変更を有効にするために、Service Mesh コントロールプレーンの **istiod** と **gateway** Pod を再起動する必要があります。以下のコマンドで Pod を再起動します。

```
$ oc -n istio-system delete pods -l 'app in (istiod,istio-ingressgateway, istio-egressgateway)'
```

Operator は、Pod を削除した後、自動的に再作成します。

5. info アプリケーションの Pod を再起動し、sidecar プロキシがシークレットの変更を取り込むようにします。以下のコマンドで Pod を再起動します。

```
$ oc -n info delete pods --all
```

以下のような出力が表示されるはずです。

```
pod "details-v1-6cd699df8c-j54nh" deleted
pod "productpage-v1-5ddcb4b84f-mtmf2" deleted
pod "ratings-v1-bdbcc68bc-kmng4" deleted
pod "reviews-v1-754ddd7b6f-lqhsv" deleted
pod "reviews-v2-675679877f-q67r2" deleted
pod "reviews-v3-79d7549c7-c2gjs" deleted
```

6. 以下のコマンドで、Pod が作成され、準備ができたことを確認します。

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

1.13.5.2. 証明書の確認

Bookinfo サンプルアプリケーションを使用して、ワークロード証明書が CA に差し込まれた証明書によって署名されていることを確認します。このプロセスでは、マシンに **openssl** がインストールされている必要があります。

1. info ワークロードから証明書を抽出するには、以下のコマンドを使用します。

```
$ sleep 60
$ oc -n info exec "$(oc -n bookinfo get pod -l app=productpage -o jsonpath=
{.items..metadata.name})" -c istio-proxy -- openssl s_client -showcerts -connect details:9080
> bookinfo-proxy-cert.txt
$ sed -n '/-----BEGIN CERTIFICATE-----/{:start /-----END CERTIFICATE-----/!N;b
start};/.*p}' info-proxy-cert.txt > certs.pem
$ awk 'BEGIN {counter=0;} /BEGIN CERT/{counter++} { print > "proxy-cert-" counter ".pem"}'
< certs.pem
```

コマンドを実行すると、作業ディレクトリーに **proxy-cert-1.pem**、**proxy-cert-2.pem**、**proxy-cert-3.pem** の 3 つのファイルが作成されるはずです。

2. ルート証明書が管理者が指定したものと同一であることを確認します。**<path>** を証明書へのパスに置き換えます。

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

ターミナルウィンドウで次の構文を実行します。

```
$ openssl x509 -in ./proxy-cert-3.pem -text -noout > /tmp/pod-root-cert.crt.txt
```

ターミナルウィンドウで以下の構文を実行して、証明書を比較します。

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

以下のような結果が表示されるはずです: **Files /tmp/root-cert.crt.txt and /tmp/pod-root-cert.crt.txt are identical**

3. CA 証明書が管理者が指定したものと同一であることを確認します。**<path>** を証明書へのパスに置き換えます。

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

ターミナルウィンドウで次の構文を実行します。

```
$ openssl x509 -in ./proxy-cert-2.pem -text -noout > /tmp/pod-cert-chain-ca.crt.txt
```

ターミナルウィンドウで以下の構文を実行して、証明書を比較します。

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

以下のような結果が表示されるはずです: **Files /tmp/ca-cert.crt.txt and /tmp/pod-cert-chain-ca.crt.txt are identical.**

4. ルート証明書からワークロード証明書への証明書チェーンを確認します。**<path>** を証明書へのパスに置き換えます。

```
$ openssl verify -CAfile <(cat <path>/ca-cert.pem <path>/root-cert.pem) ./proxy-cert-1.pem
```

以下のような出力が表示されるはず: `./proxy-cert-1.pem: OK`

1.13.5.3. 証明書の削除

追加した証明書を削除するには、以下の手順に従います。

1. シークレット **cacerts** を削除します。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前となります。

```
$ oc delete secret cacerts -n istio-system
```

2. **ServiceMeshControlPlane** リソースで自己署名ルート証明書を使用して Service Mesh を再デプロイします。

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

1.13.6. Service Mesh と cert-manager および istio-csr の統合について

cert-manager ツールは、Kubernetes での X.509 証明書管理のソリューションです。Vault、Google Cloud Certificate Authority Service、Let's Encrypt、その他のプロバイダーなどの秘密キーまたは公開キーインフラストラクチャー (PKI) とアプリケーションを統合するための統合 API を提供します。

cert-manager ツールは、証明書の有効期限が切れる前に、設定された時間に証明書の更新を試行することで、証明書が有効で最新であることを確認します。

Istio ユーザーの場合、cert-manager は、Istio プロキシからの証明書署名要求 (CSR) を処理する認証局 (CA) サーバーである **istio-csr** との統合も提供します。次に、サーバーは署名を cert-manager に委任し、設定された CA サーバーに CSR を転送します。



注記

Red Hat は、**istio-csr** および cert-manager との統合のサポートを提供します。Red Hat は **istio-csr** またはコミュニティ cert-manager コンポーネントに対する直接サポートを提供しません。ここで示すコミュニティ cert-manager の使用は、デモンストレーションのみを目的としています。

前提条件

- cert-manager の次のいずれかのバージョン:
 - Red Hat OpenShift 1.10 以降の cert-manager Operator
 - コミュニティ cert-manager Operator 1.11 以降
 - cert-manager 1.11 以降
- OpenShift Service Mesh Operator 2.4 以降
- **istio-csr** 0.6.0 以降



注記

istio-csr サーバーが **jetstack/cert-manager-istio-csr** Helm チャートとともにインストールされているときに、すべての namespace で config map が作成されないようにするには、**istio-csr.yaml** ファイル内の **app.controller.configmapNamespaceSelector:** **"maistra.io/member-of: <istio-namespace>"** 設定を使用します。

1.13.6.1. cert-manager のインストール

cert-manager ツールをインストールすると、TLS 証明書のライフサイクルを管理し、証明書が有効で最新であることを確認できます。環境内で Istio を実行している場合は、Istio プロキシからの証明書署名要求 (CSR) を処理する **istio-csr** 認証局 (CA) サーバーをインストールすることもできます。**istio-csr** CA は署名を **cert-manager** ツールに委任し、cert-manager ツールは設定された CA に委任します。

手順

1. ルートクラスターの発行者を作成します。
 - a. 次の例のように **cluster-issuer** オブジェクトを作成します。

cluster-issuer.yaml の例

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned-root-issuer
  namespace: cert-manager
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: root-ca
  namespace: cert-manager
spec:
  isCA: true
  duration: 21600h # 900d
  secretName: root-ca
  commonName: root-ca.my-company.net
  subject:
    organizations:
      - my-company.net
  issuerRef:
    name: selfsigned-root-issuer
    kind: Issuer
    group: cert-manager.io
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: root-ca
spec:
  ca:
    secretName: root-ca
```



注記

root-ca がクラスター発行者であるため、**selfsigned-root-issuer** の発行者と **root-ca** 証明書の namespace は **cert-manager** です。そのため、cert-manager は参照されるシークレットを独自の namespace で検索します。Red Hat OpenShift の cert-manager Operator の場合、namespace の名前は **cert-manager** です。

- b. 次のコマンドを使用してオブジェクトを作成します。

```
$ oc apply -f cluster-issuer.yaml
```

- c. 次の例のように **istio-ca** オブジェクトを作成します。

istio-ca.yaml の例

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: istio-ca
  namespace: istio-system
spec:
  isCA: true
  duration: 21600h
  secretName: istio-ca
  commonName: istio-ca.my-company.net
  subject:
    organizations:
      - my-company.net
  issuerRef:
    name: root-ca
    kind: ClusterIssuer
    group: cert-manager.io
---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: istio-ca
  namespace: istio-system
spec:
  ca:
    secretName: istio-ca
```

- d. 次のコマンドを使用してオブジェクトを作成します。

```
$ oc apply -n istio-system -f istio-ca.yaml
```

2. **istio-csr** をインストールします。

```
$ helm install istio-csr jetstack/cert-manager-istio-csr \
  -n istio-system \
  -f deploy/examples/cert-manager/istio-csr/istio-csr.yaml
```

istio-csr.yaml の例

```

replicaCount: 2

image:
  repository: quay.io/jetstack/cert-manager-istio-csr
  tag: v0.6.0
  pullSecretName: ""

app:
  certmanager:
    namespace: istio-system
    issuer:
      group: cert-manager.io
      kind: Issuer
      name: istio-ca

  controller:
    configmapNamespaceSelector: "maistra.io/member-of=istio-system"
    leaderElectionNamespace: istio-system

  istio:
    namespace: istio-system
    revisions: ["basic"]

  server:
    maxCertificateDuration: 5m

  tls:
    certificateDNSNames:
      # This DNS name must be set in the SMCP spec.security.certificateAuthority.cert-
      # manager.address
      - cert-manager-istio-csr.istio-system.svc

```

3. SMCP をデプロイメントします。

```
$ oc apply -f mesh.yaml -n istio-system
```

例 mesh.yaml

```

apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  addons:
    grafana:
      enabled: false
    kiali:
      enabled: false
    prometheus:
      enabled: false
  proxy:
    accessLogging:
      file:

```



```

    name: /dev/stdout
  security:
    certificateAuthority:
      cert-manager:
        address: cert-manager-istio-csr.istio-system.svc:443
        type: cert-manager
    dataPlane:
      mtls: true
    identity:
      type: ThirdParty
    tracing:
      type: None
  ---
  apiVersion: maistra.io/v1
  kind: ServiceMeshMemberRoll
  metadata:
    name: default
  spec:
    members:
      - httpbin
      - sleep

```



注記

security.identity.type: ThirdParty は **security.certificateAuthority.type: cert-manager** が設定されている場合に設定する必要があります。

検証

サンプル **httpbin** サービスと **sleep** アプリを使用して、インGRESSゲートウェイからの mTLS トラフィックをチェックし、**cert-manager** ツールがインストールされていることを確認します。

1. HTTP アプリと **sleep** アプリをデプロイします。

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

```
$ oc apply -f https://raw.githubusercontent.com/maistra/istio/maistra-2.4/samples/httpbin/httpbin.yaml
```

```
$ oc apply -f https://raw.githubusercontent.com/maistra/istio/maistra-2.4/samples/sleep/sleep.yaml
```

2. **sleep** が **httpbin** サービスにアクセスできることを確認します。

```
$ oc exec "$(oc get pod -l app=sleep -n <namespace> \
-o jsonpath={.items..metadata.name})" -c sleep -n <namespace> -- \
curl http://httpbin.<namespace>:8000/ip -s -o /dev/null \
-w "%{http_code}\n"
```

出力例

```
200
```

3. Ingress ゲートウェイから **httpbin** サービスへの mTLS トラフィックを確認します。

```
$ oc apply -n <namespace> -f https://raw.githubusercontent.com/maistra/istio/maistra-2.4/samples/httpbin/httpbin-gateway.yaml
```

4. **istio-ingressgateway** ルートを取得します。

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

5. Ingress ゲートウェイから **httpbin** サービスへの mTLS トラフィックを確認します。

```
$ curl -s -I http://$INGRESS_HOST/headers -o /dev/null -w "%{http_code}" -s
```

1.13.7. 関連情報

Red Hat OpenShift Service on AWS の cert-manager Operator をインストールする方法は、[Installing the cert-manager Operator for Red Hat OpenShift](#) を参照してください。

1.14. SERVICE MESH でのトラフィックの管理

Red Hat OpenShift Service Mesh のサービス間における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。Service Mesh 内の一部のサービスはメッシュ内で通信する必要があり、その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の層を追加できます。

1.14.1. ゲートウェイの使用

ゲートウェイを使用してメッシュの受信トラフィックおよび送信トラフィックを管理することで、メッシュに入るか、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ともに実行するサイドカー Envoy プロキシではなく、メッシュのエッジで実行するスタンドアロン Envoy プロキシに適用されます。

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します。

Red Hat OpenShift Service Mesh ゲートウェイリソースは、Red Hat OpenShift Service Mesh TLS 設定を公開して設定するポートなど、レイヤー 4-6 の負荷分散プロパティを使用できます。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、Service Mesh 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できます。

ゲートウェイは ingress トラフィックの管理に主に使用されますが、egress ゲートウェイを設定することもできます。egress ゲートウェイを使用すると、メッシュからのトラフィック専用の終了ノードを設定できます。これにより、Service Mesh にセキュリティ制御を追加することで、外部ネットワークにアクセスできるサービスを制限できます。また、ゲートウェイを使用して完全に内部のプロキシを設定することもできます。

ゲートウェイの例

ゲートウェイリソースは、着信または発信 HTTP/TCP 接続を受信するメッシュのエッジで動作するロードバランサーを表します。この仕様には、公開する必要があるポートのセット、使用するプロトコルのタイプ、ロードバランサー用の SNI 設定などが記述されています。

以下の例は、外部 HTTPS Ingress トラフィックのゲートウェイ設定を示しています。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: ext-host-gwy
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 443
      name: https
      protocol: HTTPS
    hosts:
    - ext-host.example.com
    tls:
      mode: SIMPLE
      serverCertificate: /tmp/tls.crt
      privateKey: /tmp/tls.key
```

このゲートウェイ設定により、ポート 443 での **ext-host.example.com** からメッシュへの HTTPS トラフィックが可能になりますが、トラフィックのルーティングは指定されません。

ルーティングを指定し、ゲートウェイが意図される通りに機能するには、ゲートウェイを仮想サービスにバインドする必要もあります。これは、以下の例のように、仮想サービスのゲートウェイフィールドを使用して実行します。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: virtual-svc
spec:
  hosts:
  - ext-host.example.com
  gateways:
  - ext-host-gwy
```

次に、仮想サービスを外部トラフィックのルーティングルールを使用して設定できます。

1.14.1.1. Ingress トラフィックの管理

Red Hat OpenShift Service Mesh では、Ingress Gateway は、モニタリング、セキュリティ、ルートルールなどの機能をクラスターに入るトラフィックに適用できるようにします。Service Mesh ゲートウェイを使用して Service Mesh 外のサービスを公開します。

1.14.1.1.1. Ingress IP およびポートの判別

Ingress 設定は、環境が外部ロードバランサーをサポートするかどうかによって異なります。外部ロードバランサーはクラスターの Ingress IP およびポートに設定されます。クラスターの IP およびポートが外部ロードバランサーに設定されているかどうかを判別するには、以下のコマンドを実行します。こ

の例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前となります。

```
$ 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、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。

1.14.1.1.1.1. ロードバランサーを使用した Ingress ポートの判別

お使いの環境に外部ロードバランサーがある場合は、以下の手順に従います。

手順

1. 以下のコマンドを実行して Ingress IP およびポートを設定します。このコマンドは、ターミナルに変数を設定します。

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

2. 以下のコマンドを実行して Ingress ポートを設定します。

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

3. 以下のコマンドを実行してセキュアな Ingress ポートを設定します。

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

4. 以下のコマンドを実行して TCP Ingress ポートを設定します。

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

注記

一部の環境では、ロードバランサーは IP アドレスの代わりにホスト名を使用して公開される場合があります。この場合、Ingress ゲートウェイの **EXTERNAL-IP** 値は IP アドレスではありません。これはホスト名であり、直前のコマンドは **INGRESS_HOST** 環境変数の設定に失敗します。

失敗した場合は、以下のコマンドを使用して **INGRESS_HOST** 値を修正します。

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

1.14.1.1.2. ロードバランサーのない Ingress ポートの判別

お使いの環境に外部ロードバランサーがない場合は、Ingress ポートを判別し、代わりにノードポートを使用します。

手順

1. Ingress ポートを設定します。

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

2. 以下のコマンドを実行してセキュアな Ingress ポートを設定します。

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

3. 以下のコマンドを実行して TCP Ingress ポートを設定します。

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

1.14.1.2. Ingress ゲートウェイの設定

Ingress ゲートウェイは、受信 HTTP/TCP 接続を受信するメッシュのエッジで稼働するロードバランサーです。このゲートウェイは、公開されるポートおよびプロトコルを設定しますが、これにはトラフィックルーティングの設定は含まれません。Ingress トラフィックに対するトラフィックルーティングは、内部サービス要求の場合と同様に、ルーティングルールで設定されます。

以下の手順では、ゲートウェイを作成し、**/productpage** と **/login** のパスの外部トラフィックに、Bookinfo サンプルアプリケーションのサービスを公開するように、**VirtualService** を設定します。

手順

1. トラフィックを受け入れるゲートウェイを作成します。
 - a. YAML ファイルを作成し、以下の YAML をこれにコピーします。

ゲートウェイの例 (gateway.yaml)

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: info-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
      - "*"

```

- b. YAML ファイルを適用します。

```
$ oc apply -f gateway.yaml
```

2. **VirtualService** オブジェクトを作成し、ホストヘッダーを再作成します。

- a. YAML ファイルを作成し、以下の YAML をこれにコピーします。

仮想サービスの例

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: info
spec:
  hosts:
  - "*"
  gateways:
  - info-gateway
  http:
  - match:
    - uri:
        exact: /productpage
    - uri:
        prefix: /static
    - uri:
        exact: /login
    - uri:
        exact: /logout
    - uri:
        prefix: /api/v1/products
    route:
    - destination:
        host: productpage
        port:
          number: 9080
```

- b. YAML ファイルを適用します。

```
$ oc apply -f vs.yaml
```

3. ゲートウェイと VirtualService が正しく設定されていることを確認してください。

- a. ゲートウェイ URL を設定します。

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

- b. ポート番号を設定します。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前となります。

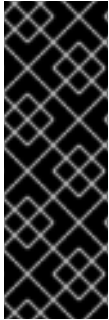
```
export TARGET_PORT=$(oc -n istio-system get route istio-ingressgateway -o
jsonpath='{.spec.port.targetPort}')
```

- c. 明示的に公開されているページをテストします。

```
curl -s -I "$GATEWAY_URL/productpage"
```

想定される結果は **200** です。

1.14.2. 自動ルートについて



重要

Istio OpenShift Routing (IOR) は非推奨の機能です。非推奨の機能は依然として Red Hat OpenShift Service on AWS に含まれており、引き続きサポートされますが、この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。

Red Hat OpenShift Service on AWS で非推奨化または削除された主な機能の最新のリストについては、Red Hat OpenShift Service on AWS リリースノートの [非推奨および削除された機能 セクション](#)を参照してください。

ゲートウェイの OpenShift ルートは Service Mesh で自動的に管理されます。Istio ゲートウェイが Service Mesh 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。



注記

Service Mesh 2.5 以降、**ServiceMeshControlPlane** リソースの新しいインスタンスでは自動ルートがデフォルトで無効になっています。

1.14.2.1. サブドメインのあるルート

Red Hat OpenShift Service Mesh はサブドメインを使用してルートを作成しますが、Red Hat OpenShift Service on AWS はそのサブドメインが有効になるように設定する必要があります。***.domain.com** などのサブドメインはサポートされますが、デフォルトでは設定されません。ワイルドカードホスト Gateway を設定する前に、Red Hat OpenShift Service on AWS ワイルドカードポリシーを設定します。

1.14.2.2. サブドメインルートの作成

以下の例では、サブドメインルートを作成する Bookinfo サンプルアプリケーションにゲートウェイを作成します。

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

```
hosts:
- www.info.com
- info.example.com
```

Gateway リソースは、次の OpenShift ルートを作成します。ルートが以下のコマンドを使用して作成されていることを確認できます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前となります。

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

予想される出力

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

このゲートウェイが削除されると、Red Hat OpenShift Service Mesh はルートを削除します。ただし、手動で作成したルートは Red Hat OpenShift Service Mesh によって変更されることはありません。

1.14.2.3. ルートラベルとアノテーション

OpenShift ルートでは、特定のラベルまたはアノテーションが必要になる場合があります。

このユースケースおよび他のユースケースでは、Red Hat OpenShift Service Mesh は (**kubectl.kubernetes.io** で始まるものを除く) Istio Gateway リソースにあるすべてのラベルとアノテーションを管理対象の OpenShift Route リソースにコピーします。

Service Mesh によって作成される OpenShift ルートで特定のラベルまたはアノテーションが必要な場合は、それらを Istio Gateway リソースで作成すると、Service Mesh で管理される OpenShift ルートリソースにコピーされます。

1.14.2.4. 自動ルート作成の無効化

デフォルトで、**ServiceMeshControlPlane** リソースは Istio ゲートウェイリソースと OpenShift ルートを自動的に同期します。自動ルート作成を無効にすると、特殊なケースがある場合やルートを手動で制御する場合に、ルートをより柔軟に制御できます。

1.14.2.4.1. 特定のケースでの自動ルート作成の無効化

特定の Istio ゲートウェイの OpenShift ルートの自動管理を無効にする場合は、アノテーション **maistra.io/manageRoute: false** をゲートウェイのメタデータ定義に追加する必要があります。Red Hat OpenShift Service Mesh は、他の Istio ゲートウェイの自動管理を維持しつつ、このアノテーションの付いた Istio ゲートウェイを無視します。

1.14.2.4.2. すべてのケースでの自動ルート作成の無効化

メッシュ内のすべてのゲートウェイの OpenShift ルートの自動管理を無効にできます。

Istio ゲートウェイと OpenShift ルート間の統合を無効にするには、**ServiceMeshControlPlane** フィールド **gateways.openshiftRoute.enabled** を **false** に設定します。たとえば、以下のリソーススニペットを参照してください。

```
apiVersion: maistra.io/v1alpha1
kind: ServiceMeshControlPlane
```



```

metadata:
  namespace: istio-system
spec:
  gateways:
    openshiftRoute:
      enabled: false

```

1.14.3. サービスエントリーについて

サービスエントリーは、Red Hat OpenShift Service Mesh が内部で維持するサービスレジストリーにエントリーを追加します。サービスエントリーの追加後、Envoy プロキシはメッシュ内のサービスであるかのようにトラフィックをサービスに送信できます。サービスエントリーを使用すると、以下が可能になります。

- Service Mesh 外で実行されるサービスのトラフィックを管理します。
- Web から消費される API やレガシーインフラストラクチャーのサービスへのトラフィックなど、外部宛先のトラフィックをリダイレクトし、転送します。
- 外部宛先の再試行、タイムアウト、およびフォールトインジェクションポリシーを定義します。
- 仮想マシンをメッシュに追加して、仮想マシン (VM) でメッシュサービスを実行します。



注記

別のクラスターからメッシュにサービスを追加し、Kubernetes でマルチクラスター Red Hat OpenShift Service Mesh メッシュを設定します。

サービスエントリーの例

以下の mesh-external サービスエントリーの例では、**ext-resource** の外部依存関係を Red Hat OpenShift Service Mesh サービスレジストリーに追加します。

```

apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: svc-entry
spec:
  hosts:
    - ext-svc.example.com
  ports:
    - number: 443
      name: https
      protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS

```

hosts フィールドを使用して外部リソースを指定します。これを完全に修飾することも、ワイルドカードの接頭辞が付けられたドメイン名を使用することもできます。

仮想サービスおよび宛先ルールを設定して、メッシュ内の他のサービスのトラフィックを設定すると同じように、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を保護するために相互 TLS を使用するように設定します。

```

apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: ext-res-dr
spec:
  host: ext-svc.example.com
  trafficPolicy:
    tls:
      mode: MUTUAL
      clientCertificate: /etc/certs/myclientcert.pem
      privateKey: /etc/certs/client_private_key.pem
      caCertificates: /etc/certs/rootcacerts.pem

```

1.14.4. VirtualServices の使用

仮想サービスを使用して、Red Hat OpenShift Service Mesh で複数バージョンのマイクロサービスに要求を動的にルーティングできます。仮想サービスを使用すると、以下が可能になります。

- 単一の仮想サービスで複数のアプリケーションサービスに対応する。メッシュが Kubernetes を使用する場合などに、仮想サービスを特定の namespace のすべてのサービス処理するように設定できます。仮想サービスを使用すると、モノリシックなアプリケーションをシームレスに、個別のマイクロサービスで構成されるサービスに変換できます。
- ingress および egress トラフィックを制御できるようにゲートウェイと組み合わせてトラフィックルールを設定する。

1.14.4.1. VirtualServices の設定

要求は、仮想サービスを使用して Service Mesh 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で最小要求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できます。

手順

1. アプリケーションに接続するユーザーに基づき、異なるバージョンの Bookinfo アプリケーションサービスのサンプルに、要求をルーティングする以下の例を使用して、YAML ファイルを作成します。

VirtualService.yaml の例

```

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

```

```

- match:
  - headers:
      end-user:
        exact: jason
    route:
  - destination:
      host: reviews
      subset: v2
- route:
  - destination:
      host: reviews
      subset: v3

```

2. 以下のコマンドを実行して **VirtualService.yaml** を適用します。 **VirtualService.yaml** はファイルへのパスです。

```
$ oc apply -f <VirtualService.yaml>
```

1.14.4.2. VirtualService 設定リファレンス

パラメーター	説明
<pre>spec: hosts:</pre>	<p>hosts フィールドには、ルーティングルールが適用される仮想サービスのユーザーの宛先アドレスが一覧表示されます。これは、サービスへの要求送信に使用するアドレスです。仮想サービスのホスト名は、IP アドレス、DNS 名または完全修飾ドメイン名に解決される短縮名になります。</p>
<pre>spec: http: - match:</pre>	<p>http セクションには、ホストフィールドで指定された宛先に送信される HTTP/1.1、HTTP2、および gRPC トラフィックのルーティングの一致条件とアクションを記述する仮想サービスのルーティングルールが含まれます。ルーティングルールは、トラフィックの宛先と、指定の一致条件で構成されます。この例の最初のルーティングルールには条件があり、match フィールドで始まります。この例では、このルーティングはユーザー jason からの要求すべてに適用されます。headers、end-user、および exact フィールドを追加し、適切な要求を選択します。</p>
<pre>spec: http: - match: - destination:</pre>	<p>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。</p>

1.14.5. 宛先ルールについて

宛先ルールは仮想サービスのルーティングルールが評価された後に適用されるため、それらはトラフィックの実際の宛先に適用されます。仮想サービスはトラフィックを宛先にルーティングします。宛先ルールでは、その宛先のトラフィックに生じる内容を設定します。

デフォルトでは、Red Hat OpenShift Service Mesh は最小要求負荷分散ポリシーを使用します。その場合は、アクティブな接続の数が最も少ないプール内のサービスインスタンスが要求を受け取ります。Red Hat OpenShift Service Mesh は以下のモデルもサポートします。このモデルは、特定のサービスまたはサービスサブセットへの要求の宛先ルールに指定できます。

- Random: 要求はプール内のインスタンスにランダムに転送されます。
- Weighted: 要求は特定のパーセンテージに応じてプールのインスタンスに転送されます。
- Least requests: 要求は要求の数が最も少ないインスタンスに転送されます。

宛先ルールの例

以下の宛先ルールの例では、異なる負荷分散ポリシーで **my-svc** 宛先サービスに 3 つの異なるサブセットを設定します。

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: my-destination-rule
spec:
  host: my-svc
  trafficPolicy:
    loadBalancer:
      simple: RANDOM
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
    trafficPolicy:
      loadBalancer:
        simple: ROUND_ROBIN
  - name: v3
    labels:
      version: v3
```

1.14.6. ネットワークポリシーについて

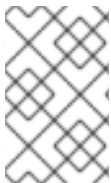
Red Hat OpenShift Service Mesh は、Service Mesh コントロールプレーンおよびアプリケーションネームスペースで多数の **NetworkPolicies** リソースを自動的に作成し、管理します。これは、アプリケーションとコントロールプレーンが相互に通信できるようにするために使用されます。

たとえば、Red Hat OpenShift Service on AWS クラスターが SDN プラグインを使用するように設定されている場合、Red Hat OpenShift Service Mesh は各メンバープロジェクトで **NetworkPolicy** リソースを作成します。これにより、他のメッシュメンバーおよびコントロールプレーンからのメッシュ内のすべての Pod に対する ingress が有効になります。また、これにより Ingress がメンバープロジェクト

のみに制限されます。メンバー以外のプロジェクトの Ingress が必要な場合は、**NetworkPolicy** を作成してそのトラフィックを許可する必要があります。Service Mesh から namespace を削除する場合、この **NetworkPolicy** リソースはプロジェクトから削除されます。

1.14.6.1. NetworkPolicy 自動作成の無効化

NetworkPolicy リソースの自動作成および管理を無効にする場合 (例: 会社のセキュリティポリシーを適用したり、メッシュ内の Pod への直接アクセスを許可する場合など) はこれを実行できます。**ServiceMeshControlPlane** を編集し、**spec.security.manageNetworkPolicy** を **false** に設定できます。



注記

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

前提条件

- Red Hat OpenShift Service Mesh Operator バージョン 2.1.1 以降がインストールされている。
- ServiceMeshControlPlane** リソースはバージョン 2.1 以降に更新されている。

手順

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

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

- Save** をクリックします。

1.14.7. トラフィック管理のサイドカーの設定

デフォルトでは、Red Hat OpenShift Service Mesh は、関連するワークロードのすべてのポートでトラフィックを受け入れ、トラフィックの転送時にメッシュ内のすべてのワークロードに到達するように、すべての Envoy プロキシを設定します。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 プロキシが到達できるサービスのセットを制限します。



注記

Service Mesh のパフォーマンスを最適化するには、Envoy プロキシ設定の制限を検討してください。

Bookinfo サンプルアプリケーションで、同じ namespace およびコントロールプレーンで実行されている他のサービスに度のサービスからでもアクセスできるように Sidecar を設定します。この Sidecar 設定は、Red Hat OpenShift Service Mesh ポリシーおよび Telemetry 機能での使用に必要になります。

手順

1. 以下の例を使用して YAML ファイルを作成し、サイドカー設定を特定の namespace の全ワークロードに適用するように指定します。それ以外の場合は、**workloadSelector** を使用して特定のワークロードを選択します。

sidecar.yaml の例

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

2. 以下のコマンドを実行して **sidecar.yaml** を適用します。ここでは、**sidecar.yaml** はファイルへのパスです。

```
$ oc apply -f sidecar.yaml
```

3. サイドカーが正常に作成されたことを確認するには、以下のコマンドを実行します。

```
$ oc get sidecar
```

1.14.8. ルーティングチュートリアル

このガイドでは Bookinfo サンプルアプリケーションを参照して、サンプルアプリケーションでのルーティングの例を説明します。[Bookinfo アプリケーション](#) をインストールして、これらのルーティングのサンプルがどのように機能するかを確認します。

1.14.8.1. Bookinfo ルーティングチュートリアル

Service Mesh Bookinfo サンプルアプリケーションは、それぞれが複数のバージョンを持つ 4 つの別個のマイクロサービスで構成されます。Bookinfo サンプルアプリケーションをインストールした後、**reviews** マイクロサービスの 3 つの異なるバージョンが同時に実行されます。

ブラウザーで Bookinfo アプリケーションの **/product** ページにアクセスして数回更新すると、書評の出力に星評価が含まれる場合と含まれない場合があります。ルーティング先の明示的なデフォルトサービ

スバージョンがない場合、Service Mesh は、利用可能なすべてのバージョンに要求をルーティングしていきます。

このチュートリアルは、すべてのトラフィックをマイクロサービスの **v1** (バージョン 1) にルーティングするルールを適用するのに役立ちます。後に、HTTP リクエストヘッダーの値に基づいてトラフィックをルーティングするためのルールを適用できます。

前提条件

- 以下の例に合わせて Bookinfo サンプルアプリケーションをデプロイする。

1.14.8.2. 仮想サービスの適用

以下の手順では、マイクロサービスのデフォルトバージョンを設定する仮想サービスを適用して、各マイクロサービスの **v1** にすべてのトラフィックをルーティングします。

手順

1. 仮想サービスを適用します。

```
$ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.6/samples/info/networking/virtual-service-all-v1.yaml
```

2. 仮想サービスの適用を確認するには、以下のコマンドで定義されたルートを表示します。

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

このコマンドでは、YAML 形式で **kind: VirtualService** のリソースを返します。

Service Mesh を Bookinfo マイクロサービスの **v1** バージョン (例: **reviews** サービスバージョン 1) にルーティングするように設定しています。

1.14.8.3. 新規ルート設定のテスト

Bookinfo アプリケーションの **/productpage** を更新して、新しい設定をテストします。

手順

1. **GATEWAY_URL** パラメーターの値を設定します。この変数を使用して、Bookinfo 製品ページの URL を後で見つけることができます。この例では、istio-system はコントロールプレーンプロジェクトの名前です。

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

2. 以下のコマンドを実行して、製品ページの URL を取得します。

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

3. ブラウザーで Bookinfo サイトを開きます。

更新回数に関係なく、ページのレビュー部分は星評価なしに表示されます。これは、Service Mesh を、reviews サービスのすべてのトラフィックをバージョン **reviews:v1** にルーティングするように設定しているためであり、サービスのこのバージョンは星評価サービスにアクセスしません。

Service Mesh は、トラフィックを1つのバージョンのサービスにルーティングするようになりました。

1.14.8.4. ユーザーアイデンティティーに基づくルート

ルート設定を変更して、特定のユーザーからのトラフィックすべてが特定のサービスバージョンにルーティングされるようにします。この場合、**jason** という名前のユーザーからのトラフィックはすべて、サービス **reviews:v2** にルーティングされます。

Service Mesh には、ユーザーアイデンティティーに関する特別な組み込み情報はありません。この例は、**productpage** サービスが reviews サービスへのすべてのアウトバウンド HTTP リクエストにカスタム **end-user** ヘッダーを追加することで有効になります。

手順

1. 以下のコマンドを実行して、Bookinfo アプリケーション例でユーザーベースのルーティングを有効にします。

```
$ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.6/samples/info/networking/virtual-service-reviews-test-v2.yaml
```

2. 以下のコマンドを実行して、ルールの作成を確認します。このコマンドは、**kind: VirtualService** のすべてのリソースを YAML 形式で返します。

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

3. Bookinfo アプリケーションの **/productpage** で、パスワードなしでユーザー **jason** としてログインします。
4. ブラウザーを更新します。各レビューの横に星評価が表示されます。
5. 別のユーザーとしてログインします (任意の名前を指定します)。ブラウザーを更新します。これで星がなくなりました。Jason 以外のすべてのユーザーのトラフィックが **reviews:v1** にルーティングされるようになりました。

ユーザーアイデンティティーに基づいてトラフィックをルーティングするように Bookinfo のアプリケーションサンプルが正常に設定されています。

1.15. メトリクス、ログ、およびトレース

アプリケーションをメッシュに追加したら、アプリケーション経由でデータフローを確認できます。独自のアプリケーションがインストールされていない場合は、[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 で可観測性がどのように機能するかを確認できます。

1.15.1. コンソールアドレスの検出

Red Hat OpenShift Service Mesh は、Service Mesh データを表示する以下のコンソールを提供します。

- **Kiali** コンソール: Kiali は Red Hat OpenShift Service Mesh の管理コンソールです。
- **Jaeger** コンソール - Jaeger は、Red Hat OpenShift 分散トレーシング Platform の管理コンソールです。

- **Grafana** コンソール: Grafana は、Istio データの高度なクエリーとメトリクス分析およびダッシュボードをメッシュ管理者に提供します。任意で、Grafana を使用して Service Mesh メトリクスを分析できます。
- **Prometheus** コンソール: Red Hat OpenShift Service Mesh は Prometheus を使用してサービスからのテレメトリー情報を保存します。

Service Mesh コントロールプレーンのインストール時に、インストールされた各コンポーネントのルートを自動的に生成します。ルートアドレスを作成したら、Kiali、Jaeger、Prometheus、または Grafana コンソールにアクセスして、Service Mesh データを表示および管理できます。

前提条件

- コンポーネントが有効で、インストールされていること。たとえば、分散トレースをインストールしていない場合、Jaeger コンソールにはアクセスできません。

OpenShift コンソールからの手順

1. cluster-admin 権限を持つユーザーとして Red Hat OpenShift Service on AWS Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合は **dedicated-admin** ロールがあるアカウント。
2. **Networking** → **Routes** に移動します。
3. **Routes** ページで、**Namespace** メニューから Service Mesh コントロールプレーンプロジェクトを選択します (例: **istio-system**)。
Location 列には、各ルートのリンク先アドレスが表示されます。
4. 必要な場合は、フィルターを使用して、アクセスするルートを持つコンポーネントコンソールを検索します。ルートの **Location** をクリックしてコンソールを起動します。
5. **Log In With OpenShift** をクリックします。

CLI からの手順

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

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

2. Service Mesh コントロールプレーンプロジェクトに切り替えます。この例では、**istio-system** は Service Mesh コントロールプレーンプロジェクトです。以下のコマンドを実行します。

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

3. 各種 Red Hat OpenShift Service Mesh コンソールのルートを取得するには、以下のコマンドを実行します。

```
$ oc get routes
```

このコマンドは、Kiali、Jaeger、Prometheus、および Grafana Web コンソールの URL と、Service Mesh 内の他のルートの URL を返します。以下のような出力が表示されるはずです。

NAME	HOST/PORT	SERVICES	PORT	TERMINATION
------	-----------	----------	------	-------------

info-gateway	bookinfo-gateway-yourcompany.com	istio-ingressgateway	http2
grafana	grafana-yourcompany.com	grafana	<all>
reencrypt/Redirect			
istio-ingressgateway	istio-ingress-yourcompany.com	istio-ingressgateway	8080
jaeger	jaeger-yourcompany.com	jaeger-query	<all> reencrypt
kiali	kiali-yourcompany.com	kiali	20001 reencrypt/Redirect
prometheus	prometheus-yourcompany.com	prometheus	<all>
reencrypt/Redirect			

4. **HOST/PORT** コラムからアクセスするコンソールの URL をブラウザにコピーして、コンソールを開きます。
5. **Log In With OpenShift** をクリックします。

1.15.2. Kiali コンソールへのアクセス

Kiali コンソールでアプリケーションのトポロジー、健全性、およびメトリクスを表示できます。サービスで問題が発生した場合、Kiali コンソールは、サービス経由でデータフローを表示できます。抽象アプリケーションからサービスおよびワークロードまで、さまざまなレベルでのメッシュコンポーネントに関する洞察を得ることができます。Kiali は、リアルタイムで namespace のインタラクティブなグラフィックビューも提供します。

Kiali コンソールにアクセスするには、Red Hat OpenShift Service Mesh がインストールされ、Kiali がインストールおよび設定されている必要があります。

インストールプロセスにより、Kiali コンソールにアクセスするためのルートが作成されます。

Kiali コンソールの URL が分かっている場合は、直接アクセスできます。URL が分からない場合は、以下の指示を使用します。

管理者の手順

1. 管理者ロールで Red Hat OpenShift Service on AWS Web コンソールにログインします。
2. **Home → Projects** をクリックします。
3. **Projects** ページで、必要に応じてフィルターを使用してプロジェクトの名前を検索します。
4. プロジェクトの名前 (例: **info**) をクリックします。
5. **Project details** ページの **Launcher** セクションで、**Kiali** リンクをクリックします。
6. Red Hat OpenShift Service on AWS コンソールにアクセスするときに使用するものと同じユーザー名とパスワードを使用して Kiali コンソールにログインします。
初回の Kiali コンソールへのログイン時に、表示するパーミッションを持つ Service Mesh 内のすべての namespace を表示する **Overview** ページが表示されます。

コンソールのインストールを検証中で、namespace がまだメッシュに追加されていないと、**istio-system** 以外のデータは表示されない可能性があります。

開発者の手順

1. 開発者ロールで Red Hat OpenShift Service on AWS Web コンソールにログインします。
2. **Project** をクリックします。

3. 必要に応じて、**Project Details** ページで、フィルターを使用してプロジェクトの名前を検索します。
4. プロジェクトの名前 (例: **info**) をクリックします。
5. **Project** ページの **Launcher** セクションで、**Kiali** リンクをクリックします。
6. **Log In With OpenShift** をクリックします。

1.15.3. Kiali コンソールでの Service Mesh データの表示

Kiali グラフは、メッシュトラフィックの強力な視覚化を提供します。このトポロジは、リアルタイムのリクエストトラフィックと Istio 設定情報を組み合わせて、Service Mesh の動作を即座に把握し、問題を迅速に特定できるようにします。複数のグラフタイプを使用すると、トラフィックを高レベルのサービストポロジ、低レベルのワークロードトポロジ、またはアプリケーションレベルのトポロジとして視覚化できます。

以下から選択できるグラフがいくつかあります。

- **App** グラフ は、同じラベルが付けられたすべてのアプリケーションの集約ワークロードを示します。
- **Service** グラフ は、メッシュ内の各サービスのノードを表示しますが、グラフからすべてのアプリケーションおよびワークロードを除外します。これは高レベルのビューを提供し、定義されたサービスのすべてのトラフィックを集約します。
- **Versioned App** グラフ は、アプリケーションの各バージョンのノードを表示します。アプリケーションの全バージョンがグループ化されます。
- **Workload** グラフ は、Service Mesh の各ワークロードのノードを表示します。このグラフでは、app および version のラベルを使用する必要はありません。アプリケーションが version ラベルを使用しない場合は、このグラフを使用します。

グラフノードは、さまざまな情報で装飾され、仮想サービスやサービスエントリなどのさまざまなルートルーティングオプションや、フォールトインジェクションやサーキットブレーカーなどの特別な設定を指定します。mTLS の問題、レイテンシーの問題、エラートラフィックなどを特定できます。グラフは高度な設定が可能で、トラフィックのアニメーションを表示でき、強力な検索機能や非表示機能があります。

Legend ボタンをクリックして、グラフに表示されるシェイプ、色、矢印、バッジに関する情報を表示します。

メトリクスの要約を表示するには、グラフ内のノードまたはエッジを選択し、そのメトリクスの詳細をサマリーの詳細パネルに表示します。

1.15.3.1. Kiali でのグラフレイアウトの変更

Kiali グラフのレイアウトは、アプリケーションのアーキテクチャーや表示データによって異なる場合があります。たとえば、グラフノードの数およびそのインタラクションにより、Kiali グラフのレンダリング方法を判別できます。すべての状況に適した単一のレイアウトを作成することは不可能であるため、Kiali は複数の異なるレイアウトの選択肢を提供します。

前提条件

- 独自のアプリケーションがインストールされていない場合は、Bookinfo サンプルアプリケーションをインストールします。次に、以下のコマンドを複数回入力して Bookinfo アプリケーションのトラフィックを生成します。

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

このコマンドはアプリケーションの **productpage** マイクロサービスにアクセスするユーザーをシミュレートします。

手順

1. Kiali コンソールを起動します。
2. **Log In With OpenShift** をクリックします。
3. Kiali コンソールで、**Graph** をクリックし、namespace グラフを表示します。
4. **Namespace** メニューから、アプリケーション namespace (例: **info**) を選択します。
5. 別のグラフレイアウトを選択するには、以下のいずれか、両方を行います。
 - グラフの上部にあるメニューから、異なるグラフデータグループを選択します。
 - App graph
 - Service graph
 - Versioned App graph (デフォルト)
 - Workload graph
 - グラフの下部にある Legend から別のグラフレイアウトを選択します。
 - Layout default dagre
 - Layout 1 cose-bilkent
 - Layout 2 cola

1.15.3.2. Kiali コンソールでのログの表示

Kiali コンソールでワークロードのログを表示できます。**Workload Detail** ページには **Logs** タブが含まれており、アプリケーションとプロキシログの両方を表示する統一されたログビューが表示されます。Kiali でログ表示を更新する頻度を選択できます。

Kiali に表示されるログのロギングレベルを変更するには、ワークロードまたはプロキシのロギング設定を変更します。

前提条件

- Service Mesh がインストールされ、設定されている。
- Kiali がインストールされ、設定されている。
- Kiali コンソールのアドレス。
- アプリケーションまたは Bookinfo サンプルアプリケーションがメッシュに追加されました。

手順

1. Kiali コンソールを起動します。
2. **Log In With OpenShift** をクリックします。
Kiali Overview ページには、閲覧権限を持つメッシュに追加された namespace が表示されません。
3. **Workloads** をクリックします。
4. **Workloads** ページで、**Namespace** メニューからプロジェクトを選択します。
5. 必要な場合は、フィルターを使用して、表示するログがあるワークロードを見つけます。ワークロードの名前をクリックします。たとえば、**ratings-v1** をクリックします。
6. **Workload Details** ページで、**Logs** タブをクリックしてワークロードのログを表示します。

ヒント

ログエントリが表示されない場合は、時間範囲または更新間隔のいずれかの調整が必要になる場合があります。

1.15.3.3. Kiali コンソールでのメトリクスの表示

Kiali コンソールで、アプリケーション、ワークロード、サービスのインバウンドおよびアウトバウンドメトリクスを表示できます。詳細ページには、以下のタブが含まれます。

- inbound Application metrics
- outbound Application metrics
- inbound Workload metrics
- outbound Workload metrics
- inbound Service metrics

これらのタブには、関連するアプリケーション、ワークロード、またはサービスレベルに合わせて調整された事前定義済みのメトリクスダッシュボードが表示されます。アプリケーションおよびワークロードの詳細ビューには、ボリューム、期間、サイズ、TCP トラフィックなどの要求および応答メトリクスが表示されます。サービスの詳細ビューには、インバウンドトラフィックの要求および応答メトリクスのみ表示されます。

Kiali では、チャート化されたディメンションを選択してチャートをカスタマイズできます。Kiali は、ソースまたは宛先プロキシメトリクスによって報告されるメトリクスを表示することもできます。また、トラブルシューティングのために、Kiali はメトリクスでトレースをオーバーレイできます。

前提条件

- Service Mesh がインストールされ、設定されている。
- Kiali がインストールされ、設定されている。
- Kiali コンソールのアドレス。
- (オプション): 分散トレースがインストールされ、設定されます。

手順

1. Kiali コンソールを起動します。
2. **Log In With OpenShift** をクリックします。
Kiali Overview ページには、閲覧権限を持つメッシュに追加された namespace が表示されません。
3. **Applications**、**Workloads**、または **Services** のいずれかをクリックします。
4. **Applications**、**Workloads**、または **Services** ページで、**Namespace** メニューからプロジェクトを選択します。
5. 必要な場合は、フィルターを使用して、ログを表示するアプリケーション、ワークロード、またはサービスを検索します。名前をクリックします。
6. **Application Detail**、**Workload Details**、または **Service Details** ページで、**Inbound Metrics** または **Outbound Metrics** タブをクリックしてメトリクスを表示します。

1.15.4. 分散トレース

分散トレースは、アプリケーションのサービス呼び出しのパスを追跡して、アプリケーション内の個々のサービスのパフォーマンスを追跡するプロセスです。アプリケーションでユーザーがアクションを起こすたびに、要求が実行され、多くのサービスが応答を生成するために対話が必要になる場合があります。この要求のパスは、分散トランザクションと呼ばれます。

Red Hat OpenShift Service Mesh は、Red Hat OpenShift 分散トレーシング Platform を使用して、開発者がマイクロサービスアプリケーション内の呼び出しフローを表示できるようにします。

1.15.4.1. Red Hat OpenShift 分散トレーシングプラットフォームと Red Hat build of OpenTelemetry の設定

ServiceMeshControlPlane の **spec.meshConfig.extensionProviders** 仕様に名前付き要素と **opentelemetry** プロバイダーを追加することで、トレースデータを Red Hat OpenShift 分散トレーシング Platform に公開できます。次に、テレメトリーカスタムリソースは、トレーススパンを収集して OpenTelemetry Collector エンドポイントに送信するように Istio プロキシを設定します。

メッシュ namespace に Red Hat build of OpenTelemetry を作成し、トレースデータをトレースプラットフォームバックエンドサービスに送信するように設定できます。

前提条件

- **tracing-system** namespace で Red Hat Tempo Operator を使用して TempoStack インスタンスを作成しました。詳細は、Red Hat OpenShift 分散トレーシング Platform のインストールを参照してください。
- Red Hat build of OpenTelemetry Operator を、推奨される namespace または **openshift-operators** namespace のいずれかにインストールしました。詳細は、「Red Hat build of OpenTelemetry のインストール」を参照してください。
- Red Hat OpenShift Service Mesh 2.5 以前を使用している場合は、**ServiceMeshControlPlane** リソースの **spec.tracing.type** パラメーターを **None** に設定して、トレースデータを OpenTelemetry Collector に送信できるようにします。

手順

1. メッシュ namespace に OpenTelemetry Collector インスタンスを作成します。この例では、**info** namespace を使用します。

OpenTelemetry Collector の設定例

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otel
  namespace: info ❶
spec:
  mode: deployment
  config: |
    receivers:
      otlp:
        protocols:
          grpc:
            endpoint: 0.0.0.0:4317
    exporters:
      otlp:
        endpoint: "tempo-sample-distributor.tracing-system.svc.cluster.local:4317" ❷
        tls:
          insecure: true
  service:
    pipelines:
      traces:
        receivers: [otlp]
        processors: []
        exporters: [otlp]
```

- ❶ **ServiceMeshMemberRoll** メンバーリストに namespace を含めます。
- ❷ この例では、TempoStack インスタンスが **tracing-system** namespace で実行しています。**ServiceMeshMemberRoll** メンバーリストに、`tracing-system` などの TempoStack namespace を含める必要はありません。



注記

- **ServiceMeshMemberRoll** メンバー namespace の1つに OpenTelemetry Collector のインスタンスを1つ作成してください。
- **OpenTelemetryCollector** リソースに **sidecar.istio.io/inject: 'true'** を追加することで、メッシュの一部として **otel-collector** を追加できます。

2. **otel-collector** Pod ログを確認し、Pod が実行中であることを確認します。

otel-collector Pod ログチェックの例

```
$ oc logs -n info -l app.kubernetes.io/name=otel-collector
```

3. **istio-system** namespace で **ServiceMeshControlPlane** カスタムリソース (CR) を作成または既存の ServiceMeshControlPlane を更新します。

SMCP カスタムリソースの例

```
kind: ServiceMeshControlPlane
apiVersion: maistra.io/v2
metadata:
  name: basic
  namespace: istio-system
spec:
  addons:
    grafana:
      enabled: false
    kiali:
      enabled: true
    prometheus:
      enabled: true
  meshConfig:
    extensionProviders:
      - name: otel
        opentelemetry:
          port: 4317
          service: otel-collector.info.svc.cluster.local
  policy:
    type: Istiod
  telemetry:
    type: Istiod
  version: v2.6
```

注記

SMCP 2.5 から 2.6 にアップグレードする場合は、**spec.tracing.type** パラメーターを **None** に設定します。

SMCP **spec.tracing.type** パラメーターの例

```
spec:
  tracing:
    type: None
```

4. **istio-system** namespace に Telemetry リソースを作成します。

Telemetry リソースの例

```
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: mesh-default
  namespace: istio-system
spec:
  tracing:
    - providers:
        - name: otel
      randomSamplingPercentage: 100
```


5. **istiod** ログを確認します。
6. Kiali リソース仕様を設定して、Kiali ワークロードトレースダッシュボードを有効にします。
ダッシュボードを使用して、トレースクエリーの結果を表示できます。

Kiali リソースの例

```
apiVersion: kiali.io/v1alpha1
kind: Kiali
# ...
spec:
  external_services:
    tracing:
      query_timeout: 30 ❶
      enabled: true
      in_cluster_url: 'http://tempo-sample-query-frontend.tracing-
system.svc.cluster.local:16685'
      url: '[Tempo query frontend Route url]'
      use_grpc: true ❷
```

- ❶ デフォルトの **query_timeout** 整数値は 30 秒です。値を 30 秒より大きく設定する場合は、Kiali CR の **.spec.server.write_timeout** を更新し、Kiali ルートに **haproxy.router.openshift.io/timeout=50s** アノテーションを追加する必要があります。**.spec.server.write_timeout** と **haproxy.router.openshift.io/timeout=** は両方とも **query_timeout** より大きくする必要があります。
- ❷ デフォルトの HTTP または gRPC ポートを使用していない場合は、**in_cluster_url**: ポートをカスタムポートに置き換えます。



注記

Kiali 1.73 は、Jaeger Query API を使用するため、Tempo リソースの制限に応じて応答時間が長くなります。Kiali UI に **Could not fetch spans** のエラーメッセージが表示された場合は、Tempo 設定を確認するか、Kiali のクエリーごとの制限を減らしてください。

7. アプリケーションにリクエストを送信します。
8. **istiod** Pod ログと **otel-collector** Pod ログを確認します。

1.15.4.1.1. mTLS で暗号化された Service Mesh メンバー namespace での **OpenTelemetryCollector** の設定

Service Mesh **dataPlane** の mTLS 暗号化を有効にすると、すべてのトラフィックが TLS で暗号化されます。

メッシュが **OpenTelemetryCollector** サービスと通信できるようにするには、**OpenTelemetryCollector** サービスに **DestinationRule** を適用して TLS **trafficPolicy** を無効にします。

DestinationRule Tempo CR の例

```
apiVersion: networking.istio.io/v1alpha3
```

```
kind: DestinationRule
metadata:
  name: otel-disable-tls
spec:
  host: "otel-collector.info.svc.cluster.local"
  trafficPolicy:
    tls:
      mode: DISABLE
```

1.15.4.1.2. mTLS で暗号化されたサービスマッシュメンバー名前空間での Red Hat OpenShift 分散トレーシングプラットフォームの設定



注記

Service Mesh メンバー namespace ではない namespace に TempoStack インスタンスを作成した場合、この追加の **DestinationRule** 設定は必要ありません。

Service Mesh **dataPlane** mTLS 暗号化を有効にし、**tracing-system-mtls** などの Service Mesh メンバー namespace に TempoStack インスタンスを作成すると、すべてのトラフィックが TLS で暗号化されます。この暗号化は Tempo 分散サービスでは予期されていないため、TLS エラーが返されます。

TLS エラーを修正するには、Tempo と Kiali に **DestinationRule** を適用して TLS **trafficPolicy** を無効にします。

DestinationRule Tempo の例

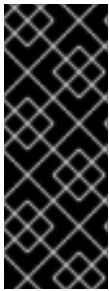
```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: tempo
  namespace: tracing-system-mtls
spec:
  host: ".*tracing-system-mtls.svc.cluster.local"
  trafficPolicy:
    tls:
      mode: DISABLE
```

DestinationRule Kiali の例

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: kiali
  namespace: istio-system
spec:
  host: kiali.istio-system.svc.cluster.local
  trafficPolicy:
    tls:
      mode: DISABLE
```

1.15.4.2. 既存の分散トレーシングの Jaeger インスタンスへの接続

Red Hat OpenShift Service on AWS に既存の Red Hat OpenShift 分散トレーシング Platform (Jaeger) インスタンスがある場合は、そのインスタンスを分散トレーシング Platform に使用するように **ServiceMeshControlPlane** リソースを設定できます。



重要

Red Hat OpenShift Service Mesh 2.5 以降、Red Hat OpenShift 分散トレーシング Platform (Jaeger) および OpenShift Elasticsearch Operator は非推奨となり、今後のリリースで削除される予定です。Red Hat は、現行リリースのライフサイクル中はこれらの機能のバグ修正とサポートを提供しますが、今後これらの機能に対する機能強化は行われません。Red Hat OpenShift 分散トレーシング Platform (Jaeger) の代わりに、Red Hat OpenShift 分散トレーシング Platform を使用することもできます。

前提条件

- Red Hat OpenShift 分散トレーシング Platform インスタンスがインストールおよび設定されています。

手順

- Red Hat OpenShift Service on AWS Web コンソールで、**Operators → Installed Operators** をクリックします。
- Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
- Red Hat OpenShift Service Mesh Operator をクリックします。**Istio Service Mesh Control Plane** 列で、**ServiceMeshControlPlane** リソースの名前 (**basic** など) をクリックします。
- 分散トレーシング Platform (Jaeger) インスタンスの名前を **ServiceMeshControlPlane** に追加します。
 - YAML** タブをクリックします。
 - ServiceMeshControlPlane** リソースの **spec.addons.jaeger.name** に、分散トレーシング Platform (Jaeger) インスタンスの名前を追加します。次の例では、**distr-tracing-production** は分散トレーシング Platform (Jaeger) インスタンスの名前です。

分散トレースの設定例

```
spec:
  addons:
    jaeger:
      name: distr-tracing-production
```

- Save** をクリックします。
- Reload** をクリックして、**ServiceMeshControlPlane** リソースが正しく設定されていることを確認します。

1.15.4.3. サンプリングレートの調整

トレースは、Service Mesh 内のサービス間の実行パスです。トレースは1つ以上のスパンで構成されます。スパンは、名前、開始時間、および期間を持つ作業の論理単位です。サンプリングレートは、トレースが永続化される頻度を決定します。

Envoy プロキシのサンプリングレートは、デフォルトで Service Mesh でトレースの 100% をサンプリングするように設定されています。サンプリングレートはクラスターリソースおよびパフォーマンスを消費しますが、問題のデバッグを行う場合に役立ちます。Red Hat OpenShift Service Mesh を実稼働環境でデプロイする前に、値を小さめのトレースサイズに設定します。たとえば、**spec.tracing.sampling** を **100** に設定し、トレースの 1% をサンプリングします。

Envoy プロキシサンプリングレートを、0.01% の増分を表すスケールリングされた整数として設定します。

基本的なインストールでは、**spec.tracing.sampling** は **10000** に設定され、トレースの 100% をサンプリングします。以下に例を示します。

- この値を 10 サンプル (トレースの 0.1%) に設定します。
- この値を 500 サンプル (トレースの 5%) に設定します。



注記

Envoy プロキシサンプリングレートは、Service Mesh で利用可能なアプリケーションに適用され、Envoy プロキシを使用します。このサンプリングレートは、Envoy プロキシが収集および追跡するデータ量を決定します。

Jaeger リモートサンプリングレートは、Service Mesh の外部にあるアプリケーションに適用され、データベースなどの Envoy プロキシを使用しません。このサンプリングレートは、分散トレースシステムが収集および保存するデータ量を決定します。

手順

1. Red Hat OpenShift Service on AWS Web コンソールで、**Operators → Installed Operators** をクリックします。
2. **Project** メニューをクリックし、コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
3. Red Hat OpenShift Service Mesh Operator をクリックします。 **Istio Service Mesh Control Plane** 列で、**ServiceMeshControlPlane** リソースの名前 (**basic** など) をクリックします。
4. サンプリングレートを調整するには、**spec.tracing.sampling** に別の値を設定します。
 - a. **YAML** タブをクリックします。
 - b. **ServiceMeshControlPlane** リソースで **spec.tracing.sampling** の値を設定します。以下の例では、**100** に設定します。

Jaeger サンプリングの例

```
spec:
  tracing:
    sampling: 100
```

- c. **Save** をクリックします。
5. **Reload** をクリックして、**ServiceMeshControlPlane** リソースが正しく設定されていることを確認します。

1.15.5. Jaeger コンソールへのアクセス

Jaeger コンソールにアクセスするには、Red Hat OpenShift Service Mesh がインストールされ、Red Hat OpenShift 分散トレーシング Platform (Jaeger) がインストールおよび設定されている必要があります。

インストールプロセスにより、Jaeger コンソールにアクセスするためのルートが作成されます。

Jaeger コンソールの URL が分かっている場合は、これに直接アクセスできます。URL が分からない場合は、以下の指示を使用します。



重要

Red Hat OpenShift Service Mesh 2.5 以降、Red Hat OpenShift 分散トレーシング Platform (Jaeger) および OpenShift Elasticsearch Operator は非推奨となり、今後のリリースで削除される予定です。Red Hat は、現在のリリースのライフサイクル中にこの機能のバグ修正とサポートを提供しますが、この機能は今後、機能拡張を受け取らず、削除されます。Red Hat OpenShift 分散トレーシング Platform (Jaeger) の代わりに、Red Hat OpenShift 分散トレーシング Platform を使用することもできます。

OpenShift コンソールからの手順

1. cluster-admin 権限を持つユーザーとして Red Hat OpenShift Service on AWS Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合は **dedicated-admin** ロールがあるアカウント。
2. **Networking** → **Routes** に移動します。
3. **Routes** ページで、**Namespace** メニューから Service Mesh コントロールプレーンプロジェクトを選択します (例: **istio-system**)。
Location 列には、各ルートのリンク先アドレスが表示されます。
4. 必要な場合は、フィルターを使用して **jaeger** ルートを検索します。ルートの **Location** をクリックしてコンソールを起動します。
5. **Log In With OpenShift** をクリックします。

Kiali コンソールからの手順

1. Kiali コンソールを起動します。
2. 左側のナビゲーションペインで **Distributed Tracing** をクリックします。
3. **Log In With OpenShift** をクリックします。

CLI からの手順

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

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

2. コマンドラインを使用してルートの詳細をクエリーするには、以下のコマンドを入力します。この例では、**istio-system** が Service Mesh コントロールプレーンの namespace です。

```
$ oc get route -n istio-system jaeger -o jsonpath='{.spec.host}'
```

3. ブラウザーを起動し、**https://<JAEGER_URL>** に移動します。ここで、**<JAEGER_URL>** は直前の手順で検出されたルートです。
4. Red Hat OpenShift Service on AWS 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ログインします。
5. Service Mesh にサービスを追加し、トレースを生成している場合は、フィルターと **Find Traces** ボタンを使用してトレースデータを検索します。
コンソールインストールを検証すると、表示するトレースデータはありません。

1.15.6. Grafana コンソールへのアクセス

Grafana は、Service Mesh メトリクスの表示、クエリー、および分析に使用できる解析ツールです。この例では、**istio-system** が Service Mesh コントロールプレーンの namespace です。Grafana にアクセスするには、以下の手順を実施します。

手順

1. Red Hat OpenShift Service on AWS Web コンソールにログインします。
2. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
3. **Routes** をクリックします。
4. **Grafana** 行の **Location** コラムのリンクをクリックします。
5. Red Hat OpenShift Service on AWS の認証情報を使用して Grafana コンソールにログインします。

1.15.7. Prometheus コンソールへのアクセス

Prometheus は、マイクロサービスに関する多次元データの収集に使用できるモニタリングおよびアラートツールです。この例では、**istio-system** が Service Mesh コントロールプレーンの namespace です。

手順

1. Red Hat OpenShift Service on AWS Web コンソールにログインします。
2. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
3. **Routes** をクリックします。
4. **Prometheus** 行の **Location** コラムのリンクをクリックします。
5. Red Hat OpenShift Service on AWS の認証情報を使用して Prometheus コンソールにログインします。

1.15.8. ユーザーのワークロード監視との統合

デフォルトでは、Red Hat OpenShift Service Mesh (OSSM) は、メッシュからメトリクスを収集するた

めの Prometheus の専用インスタンスを含む Service Mesh コントロールプレーン (SMCP) をインストールします。ただし、運用システムには、ユーザー定義プロジェクトの Red Hat OpenShift Service on AWS 監視など、より高度な監視システムが必要です。

次の手順では、Service Mesh をユーザーワークロードの監視と統合する方法を示します。

前提条件

- ユーザーのワークロードの監視が有効になっている。
- Red Hat OpenShift Service Mesh Operator 2.4 がインストールされている。
- Kiali Operator 1.65 がインストールされている。

手順

1. **cluster-monitoring-view** ロールを Kiali サービスアカウントに付与します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kiali-monitoring-rbac
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-monitoring-view
subjects:
- kind: ServiceAccount
  name: kiali-service-account
  namespace: istio-system
```

2. ユーザーワークロードを監視するために Kiali を設定します。

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

- Istio Operator 2.4 を使用する場合は、次の設定を使用して Kiali をユーザーワークロードモニタリング用に設定します。

```
apiVersion: kiali.io/v1alpha1
kind: Kiali
metadata:
  name: kiali-user-workload-monitoring
```

```

namespace: istio-system
spec:
  external_services:
    istio:
      config_map_name: istio-<smcp-name>
      istio_sidecar_injector_config_map_name: istio-sidecar-injector-<smcp-name>
      istiod_deployment_name: istiod-<smcp-name>
      url_service_version: 'http://istiod-<smcp-name>.istio-system:15014/version'
    prometheus:
      auth:
        token: secret:thanos-querier-web-token:token
        type: bearer
        use_kiali_token: false
      query_scope:
        mesh_id: "basic-istio-system"
      thanos_proxy:
        enabled: true
      url: https://thanos-querier.openshift-monitoring.svc.cluster.local:9091
  version: v1.65

```

注記

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

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

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

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

3. 外部 Prometheus 用に SMCP を設定します。

```

apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  addons:
    prometheus:
      enabled: false ❶
    grafana:
      enabled: false ❷

```



```
kiali:
  name: kiali-user-workload-monitoring
meshConfig:
  extensionProviders:
    - name: prometheus
    prometheus: {}
```

- ❶ OSSM によって提供されるデフォルトの Prometheus インスタンスを無効にします。
- ❷ Grafana を無効にします。外部 Prometheus インスタンスではサポートされていません。

4. カスタムネットワークポリシーを適用して、モニタリング namespace からの受信トラフィックを許可します。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: user-workload-access
  namespace: istio-system ❶
spec:
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
                network.openshift.io/policy-group: monitoring
  podSelector: {}
  policyTypes:
    - Ingress
```

- ❶ カスタムネットワークポリシーはすべての namespace に適用する必要があります。

5. **Telemetry** オブジェクトを適用して、Istio プロキシのトラフィックメトリクスを有効にします。

```
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: enable-prometheus-metrics
  namespace: istio-system ❶
spec:
  selector: ❷
    matchLabels:
      app: info
  metrics:
    - providers:
        - name: prometheus
```

- ❶ コントロールプレーンの namespace で作成された **Telemetry** オブジェクトは、メッシュ内のすべてのワークロードに適用されます。Telemetry を1つの namespace のみに適用するには、ターゲット namespace にオブジェクトを作成します。
- ❷ オプション: **selector.matchLabels** 仕様を設定すると、ターゲット namespace 内の特定のワークロードに **Telemetry** オブジェクトが適用されます。

6. **ServiceMonitor** オブジェクトを適用して Istio コントロールプレーンを監視します。

```

apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: istiod-monitor
  namespace: istio-system ❶
spec:
  targetLabels:
    - app
  selector:
    matchLabels:
      istio: pilot
  endpoints:
    - port: http-monitoring
      interval: 30s
      relabelings:
        - action: replace
          replacement: "basic-istio-system" ❷
          targetLabel: mesh_id

```

- ❶ この **ServiceMonitor** オブジェクトは Istiod サービスを監視するため、Istio コントロールプレーン namespace に作成します。この例では、namespace は **istio-system** です。
- ❷ 文字列 "**basic-istio-system**" は、SMCP 名とその namespace の組み合わせですが、クラスター内のユーザーワークロード監視を使用するメッシュごとに一意である限り、任意のラベルを使用できます。ステップ 2 で設定した Kiali リソースの **spec.prometheus.query_scope** は、この値と一致する必要があります。



注記

ユーザーワークロードモニタリングを使用するメッシュが1つだけの場合、Kiali リソースの **Mesh_id** の再ラベル付けと **spec.prometheus.query_scope** フィールドは両方ともオプションです (ただし、**mesh_id** ラベルが削除される場合は、ここで指定される **query_scope** フィールドも削除される必要があります)。

クラスター上の複数のメッシュインスタンスがユーザーワークロードのモニタリングを使用する可能性がある場合は、Kiali リソースの **mesh_id** の再ラベル付けと **spec.prometheus.query_scope** フィールドの両方が必要です。これにより、Kiali は関連付けられたメッシュからのメトリクスのみ参照します。

Kiali をデプロイしていない場合でも、別のメッシュのメトリクスを区別できるように **mesh_id** の再ラベル付けを適用できます。

7. **PodMonitor** オブジェクトを適用して、Istio プロキシからメトリクスを収集します。

```

apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: istio-proxies-monitor
  namespace: istio-system ❶
spec:
  selector:
    matchExpressions:

```

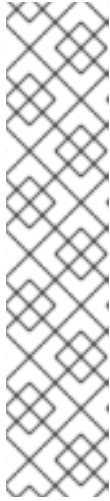
```

- key: istio-prometheus-ignore
  operator: DoesNotExist
podMetricsEndpoints:
- path: /stats/prometheus
  interval: 30s
  relabelings:
  - action: keep
    sourceLabels: [__meta_kubernetes_pod_container_name]
    regex: "istio-proxy"
  - action: keep
    sourceLabels: [__meta_kubernetes_pod_annotationpresent_prometheus_io_scrape]
  - action: replace
    regex: (\d+);((([A-Fa-f0-9]{1,4}::?){1,7}[A-Fa-f0-9]{1,4}))
    replacement: '[$2]:$1'
    sourceLabels: [__meta_kubernetes_pod_annotation_prometheus_io_port,
      __meta_kubernetes_pod_ip]
    targetLabel: __address__
  - action: replace
    regex: (\d+);((([0-9]+?)\.|\$)){4})
    replacement: $2:$1
    sourceLabels: [__meta_kubernetes_pod_annotation_prometheus_io_port,
      __meta_kubernetes_pod_ip]
    targetLabel: __address__
  - action: labeldrop
    regex: "__meta_kubernetes_pod_label_(.+)\"
  - sourceLabels: [__meta_kubernetes_namespace]
    action: replace
    targetLabel: namespace
  - sourceLabels: [__meta_kubernetes_pod_name]
    action: replace
    targetLabel: pod_name
  - action: replace
    replacement: "basic-istio-system" ❷
    targetLabel: mesh_id

```

❶ Red Hat OpenShift Service on AWS モニタリングは **ServiceMonitor** オブジェクトと **PodMonitor** オブジェクトの **namespaceSelector** 仕様を無視するため、コントロールプレーンの namespace を含むすべてのメッシュ namespace に **PodMonitor** オブジェクトを適用する必要があります。

❷ 文字列 **"basic-istio-system"** は、SMCP 名とその namespace の組み合わせですが、クラスター内のユーザーワークロード監視を使用するメッシュごとに一意である限り、任意のラベルを使用できます。ステップ 2 で設定した Kiali リソースの **spec.prometheus.query_scope** は、この値と一致する必要があります。



注記

ユーザーワークロードモニタリングを使用するメッシュが1つだけの場合、Kiali リソースの **Mesh_id** の再ラベル付けと **spec.prometheus.query_scope** フィールドは両方ともオプションです (ただし、**mesh_id** ラベルが削除される場合は、ここで指定される **query_scope** フィールドも削除される必要があります)。

クラスター上の複数のメッシュインスタンスがユーザーワークロードのモニタリングを使用する可能性がある場合は、Kiali リソースの **mesh_id** の再ラベル付けと **spec.prometheus.query_scope** フィールドの両方が必要です。これにより、Kiali は関連付けられたメッシュからのメトリクスのみ参照します。

Kiali をデプロイしていない場合でも、別のメッシュのメトリクスを区別できるように **mesh_id** の再ラベル付けを適用できます。

8. Red Hat OpenShift Service on AWS Web コンソールを開き、メトリックが表示されていることを確認します。

1.16. パフォーマンスおよびスケーラビリティ

デフォルトの **ServiceMeshControlPlane** 設定は実稼働環境での使用を目的としていません。それらはリソース面で制限のあるデフォルトの Red Hat OpenShift Service on AWS インストールに正常にインストールされるように設計されています。SMCP インストールに成功したことを確認したら、SMCP 内で定義した設定をお使いの環境に合わせて変更する必要があります。

1.16.1. コンピュートリソースでの制限の設定

デフォルトでは、**spec.proxy** には **cpu:10m** および **memory:128M** の設定があります。Pilot を使用している場合、**spec.runtime.components.pilot** には同じデフォルト値があります。

以下の例の設定は、1秒あたり1,000サービスおよび1,000要求をベースとしています。**ServiceMeshControlPlane** で **cpu** および **memory** の値を変更できます。

手順

1. Red Hat OpenShift Service on AWS Web コンソールで、**Operators → Installed Operators** をクリックします。
2. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
3. Red Hat OpenShift Service Mesh Operator をクリックします。Istio **Service Mesh Control Plane** 列で、**ServiceMeshControlPlane** の名前 (**basic** など) をクリックします。
4. スタンドアロンの Jaeger インスタンスの名前を **ServiceMeshControlPlane** に追加します。
 - a. **YAML** タブをクリックします。
 - b. **ServiceMeshControlPlane** リソースの **spec.proxy.runtime.container.resources.requests.cpu**、**spec.proxy.runtime.container.resources.requests.memory**、**components.kiali.container**、および **components.global.oauthproxy** の値を設定します。

サンプルバージョン 2.6 ServiceMeshControlPlane

```

apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.6
  proxy:
    runtime:
      container:
        resources:
          requests:
            cpu: 600m
            memory: 50Mi
          limits: {}
  runtime:
    components:
      pilot:
        container:
          resources:
            requests:
              cpu: 1000m
              memory: 1.6Gi
            limits: {}
      kiali:
        container:
          resources:
            limits:
              cpu: "90m"
              memory: "245Mi"
            requests:
              cpu: "30m"
              memory: "108Mi"
      global.observabilityproxy:
        container:
          resources:
            requests:
              cpu: "101m"
              memory: "256Mi"
            limits:
              cpu: "201m"
              memory: "512Mi"

```

- c. Red Hat OpenShift 分散トレーシング Platform (Jaeger) の値を設定するには、分散トレーシングプラットフォーム Jaeger の設定とデプロイを参照してください。
- d. **Save** をクリックします。

検証

- **Reload** をクリックして、**ServiceMeshControlPlane** リソースが正しく設定されたことを確認します。

1.16.2. テスト結果の読み込み

関連情報

アップストリームの Istio コミュニティの負荷テストのメッシュは、1 秒あたり 70,000 のメッシュ全体の要求を持つ 1000 サービスと 2000 サイドカーで構成されます。Istio 1.12.3 を使用してテストを実行後、以下の結果が生成されました。

- Envoy プロキシは、プロキシを通過する 1 秒あたり/要求 1000 件あたり **0.35 vCPU** および **40 MB** メモリーを使用します。
- Istiod は **1 vCPU** および **1.5 GB** のメモリーを使用します。
- Envoy プロキシは **2.65 ms** を 90 % レイテンシーに追加します。
- 従来の **istio-telemetry** サービス (Service Mesh 2.0 ではデフォルトで無効になっています) は、Mixer を使用するデプロイメントの場合は、1 秒あたり 1000 メッシュ全体のリクエストごとに **0.6 vCPU** を使用します。データプレーンのコンポーネントである Envoy プロキシは、システムを通過するデータフローを処理します。Service Mesh コントロールプレーンコンポーネントである Istiod は、データプレーンを設定します。データプレーンおよびコントロールプレーンには、さまざまなパフォーマンスに関する懸念点があります。

1.16.2.1. Service Mesh コントロールプレーンのパフォーマンス

Istiod は、ユーザーが作成する設定ファイルおよびシステムの現在の状態に基づいてサイドカープロキシを設定します。Kubernetes 環境では、カスタムリソース定義 (CRD) およびデプロイメントはシステムの設定および状態を構成します。ゲートウェイや仮想サービスなどの Istio 設定オブジェクトは、ユーザーが作成する設定を提供します。プロキシの設定を生成するために、Istiod は Kubernetes 環境およびユーザー作成の設定から、組み合わせた設定およびシステムの状態を処理します。

Service Mesh コントロールプレーンは、数千のサービスをサポートし、これらは同様の数のユーザーが作成する仮想サービスおよびその他の設定オブジェクトと共に数千の Pod 全体に分散されます。Istiod の CPU およびメモリー要件は、設定数および使用可能なシステムの状態と共にスケーリングされます。CPU の消費は、以下の要素でスケーリングします。

- デプロイメントの変更レート。
- 設定の変更レート。
- Istiod へのプロキシ数。

ただし、この部分は水平的にスケーリングが可能です。

1.16.2.2. データプレーンのパフォーマンス

データプレーンのパフォーマンスは、以下を含む数多くの要因によって変わります。

- クライアント接続の数
- ターゲットの要求レート
- 要求サイズおよび応答サイズ
- プロキシワーカーのスレッド数
- プロトコル
- CPU コア数

- プロキシフィルターの数およびタイプ (とくに Telemetry v2 関連のフィルター)。

レイテンシー、スループット、およびプロキシの CPU およびメモリーの消費は、これらの要素の関数として測定されます。

1.16.2.2.1. CPU およびメモリーの消費

Sidecar プロキシはデータパスで追加の作業を実行するため、CPU およびメモリーを消費します。Istio 1.12.3 の時点で、プロキシは 1 秒あたり 1000 要求ベースで 0.5 vCPU を消費します。

プロキシのメモリー消費は、プロキシが保持する設定の状態の合計数によって異なります。多数のリスナー、クラスター、およびルートは、メモリーの使用量を増やす可能性があります。

通常、プロキシは通過するデータをバッファに入れられないため、要求レートはメモリー消費には影響を及ぼしません。

1.16.2.2.2. その他のレイテンシー

Istio はデータパスにサイドカープロキシを挿入するため、レイテンシーは重要な考慮事項になります。Istio は認証フィルター、Telemetry フィルター、およびメタデータ交換フィルターをプロキシに追加します。すべての追加フィルターはプロキシ内のパスの長さに追加され、これはレイテンシーに影響を及ぼします。

Envoy プロキシは、応答がクライアントに送信された後に未加工の Telemetry データを収集します。リクエストの生のテレメトリーの収集に費やされた時間は、そのリクエストの完了にかかる合計時間には影響しません。ただし、ワーカーは要求処理にビジー状態になるため、ワーカーは次の要求の処理をすぐに開始しません。このプロセスは、次の要求のキューの待機時間に追加され、平均のレイテンシーおよびテイルレイテンシー (Tail Latency) に影響を及ぼします。実際のテイルレイテンシーは、トラフィックのパターンによって異なります。

メッシュ内で、要求はクライアント側のプロキシを通過してから、サーバー側のプロキシを通過します。Istio 1.12.3 のデフォルト設定 (Istio と Telemetry v2) では、2 つのプロキシは、ベースラインのデータプレーンのレイテンシーに対してそれぞれ約 1.7 ms および 2.7 ms を 90 および 99 番目のパーセントテイルレイテンシーに追加します。

1.17. 実稼働環境の SERVICE MESH の設定

基本インストールから実稼働環境に移行する準備ができれば、実稼働環境の要件を満たすようにコントロールプレーン、トレーシング、およびセキュリティ証明書を設定する必要があります。

前提条件

- Red Hat OpenShift Service Mesh をインストールして設定しておく。
- ステージング環境で設定をテストしておく

1.17.1. 実稼働環境用の ServiceMeshControlPlane リソース設定

Service Mesh をテストするために基本的な **ServiceMeshControlPlane** リソースをインストールしている場合は、実稼働環境で Red Hat OpenShift Service Mesh を使用する前にこれを実稼働仕様に設定する必要があります。

既存の **ServiceMeshControlPlane** リソースの **metadata.name** フィールドを変更できません。実稼働デプロイメントの場合は、デフォルトのテンプレートをカスタマイズする必要があります。

手順

1. 実稼働用に分散トレーシングプラットフォーム (Jaeger) を設定します。
 - a. **ServiceMeshControlPlane** リソースは、**production** デプロイメントストラテジーを使用するように編集するには、**spec.addons.jaeger.install.storage.type** を **Elasticsearch** に設定し、**install** で追加設定オプションを指定します。Jaeger インスタンスを作成および設定し、**spec.addons.jaeger.name** を Jaeger インスタンスの名前に設定できます。

デフォルト Jaeger パラメーター (例: Elasticsearch)

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
  tracing:
    sampling: 100
    type: Jaeger
  addons:
    jaeger:
      name: MyJaeger
      install:
        storage:
          type: Elasticsearch
        ingress:
          enabled: true
      runtime:
        components:
          tracing.jaeger.elasticsearch: # only supports resources and image name
          container:
            resources: {}
```

- b. 実稼働環境のサンプリングレートを設定します。詳細は、パフォーマンスおよびスケーラビリティセクションを参照してください。
2. 外部認証局からセキュリティ証明書をインストールして、セキュリティ証明書が実稼働可能であることを確認します。詳細は、セキュリティのセクションを参照してください。

検証

1. 以下のコマンドを実行して、**ServiceMeshControlPlane** リソースが適切に更新されていることを確認します。この例では、**basic** は **ServiceMeshControlPlane** リソースの名前です。

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

1.17.2. 関連情報

- パフォーマンス用の Service Mesh のチューニングに関する詳細は、[パフォーマンスおよびスケーラビリティ](#) を参照してください。

1.18. SERVICE MESH の接続

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

1.18.1. フェデレーションの概要

フェデレーションは、個別のメッシュ間でサービスを接続できるようにする機能セットであり、複数の別個の管理対象ドメイン間での認証、認可、およびトラフィック管理などの Service Mesh 機能を使用できるようにします。

フェデレーションメッシュを実装すると、複数の OpenShift クラスター全体で実行されている単一の Service Mesh を実行、管理、および監視できます。Red Hat OpenShift Service Mesh のフェデレーションは、メッシュ間の最小限の信頼を前提とする Service Mesh のマルチクラスター実装に対して独自のアプローチを採用しています。

Service Mesh フェデレーションでは、各メッシュが個別に管理され、独自の管理者を用意することが前提です。デフォルトの動作では、通信が許可されず、メッシュ間で情報を共有されません。メッシュ間の情報、オプトインを明示することで共有されます。共有設定されていない限り、フェデレーションメッシュでは共有されません。証明書の生成、メトリクス、トレース収集などのサポート機能は、それぞれのメッシュのローカルで機能します。

各 Service Mesh で **ServiceMeshControlPlane** が、フェデレーション専用の ingress および egress ゲートウェイを作成してメッシュの信頼ドメインを指定するように設定します。

フェデレーションでは、追加でフェデレーションファイルも作成されます。以下のリソースを使用して、2つ以上のメッシュ間のフェデレーションを設定します。

- **ServiceMeshPeer** リソースは、Service Mesh のペア間のフェデレーションを宣言します。
- **ExportedServiceSet** リソース: メッシュからの1つ以上のサービスがピアメッシュで使用できることを宣言します。
- **ImportedServiceSet** リソース: ピアメッシュでエクスポートされたどのサービスがメッシュにインポートされるかを宣言します。

1.18.2. フェデレーション機能

メッシュに参加するための Red Hat OpenShift Service Mesh フェデレーションアプローチの機能は以下のとおりです。

- 各メッシュの共通ルート証明書をサポートします。
- 各メッシュの異なるルート証明書をサポートします。
- メッシュ管理者は、フェデレーションメッシュの外部にあるメッシュの証明書チェーン、サービス検出エンドポイント、信頼ドメインを手動で設定する必要があります。
- メッシュ間で共有するサービスのみをエクスポート/インポートします。
 - デフォルトは、デプロイされたワークロードに関する情報は、フェデレーション内の他のメッシュとは共有されません。サービスをエクスポートして他のメッシュに公開し、独自のメッシュ外のワークロードから要求できるようにします。
 - エクスポートされたサービスは別のメッシュにインポートでき、そのメッシュのワークロードをインポートされたサービスに送信できるようにします。
- メッシュ間の通信を常時暗号化します。

- ローカルにデプロイされたワークロードおよびフェデレーション内の別のメッシュにデプロイされたワークロードの間における負荷分散の設定をサポートします。

メッシュが別のメッシュに参加すると、以下を実行できます。

- フェデレーションメッシュに対して自信の信頼情報を提供します。
- フェデレーションメッシュに関する信頼情報を検出します。
- 独自にエクスポートされたサービスに関する情報をフェデレーションメッシュに提供します。
- フェデレーションメッシュでエクスポートされるサービスの情報を検出します。

1.18.3. フェデレーションセキュリティ

Red Hat OpenShift Service Mesh のフェデレーションは、メッシュ間の最小限の信頼を前提とする Service Mesh のマルチクラスター実装に対して独自のアプローチを採用しています。データセキュリティは、フェデレーション機能の一部として組み込まれています。

- メッシュごとに、テナントや管理が一意となっています。
- フェデレーションで各メッシュに一意の信頼ドメインを作成します。
- フェデレーションメッシュ間のトラフィックは、相互トランスポート層セキュリティ (mTLS) を使用して自動的に暗号化されます。
- Kiali グラフは、インポートしたメッシュとサービスのみを表示します。メッシュにインポートされていない他のメッシュまたはサービスを確認できません。

1.18.4. フェデレーションの制限

メッシュに参加するための Red Hat OpenShift Service Mesh フェデレーションアプローチには、以下の制限があります。

- メッシュのフェデレーションは OpenShift Dedicated ではサポートされていません。

1.18.5. フェデレーションの前提条件

メッシュに参加するための Red Hat OpenShift Service Mesh フェデレーションアプローチには、以下の前提条件があります。

- 2 つ以上の Red Hat OpenShift Service on AWS 4.6 以降のクラスター。
- フェデレーションは、Red Hat OpenShift Service Mesh 2.1 以降で導入されました。フェデレーションする必要のある各メッシュに Red Hat OpenShift Service Mesh 2.1 Operator がインストールされている必要があります。
- フェデレーションする必要のある各メッシュにバージョン 2.1 以降の **ServiceMeshControlPlane** をデプロイする必要があります。
- 生の TLS トラフィックをサポートするように、フェデレーションゲートウェイに関連付けられたサービスをサポートするロードバランサーを設定する必要があります。フェデレーショントラフィックは、検出用の HTTPS と、サービストラフィック用の生の暗号化 TCP で構成されます。
- 別のメッシュに公開するサービスは、エクスポートおよびインポートする前にデプロイする必

要があります。ただし、これは厳密な要件ではありません。エクスポート/インポート用にまだ存在していないサービス名を指定できます。**ExportedServiceSet** と **ImportedServiceSet** という名前のサービスをデプロイする場合に、これらのサービスは自動的にエクスポート/インポートで利用可能になります。

1.18.6. メッシュフェデレーションのプランニング

メッシュフェデレーションの設定を開始する前に、実装の計画に時間をかけるようにしてください。

- フェデレーションに参加させる予定のメッシュは何個ありますか？まずは、2 つから 3 つ程度の限られた数のメッシュで開始する必要があります。
- 各メッシュにどの命名規則を使用する予定ですか？事前定義の命名規則があると、設定とトラブルシューティングに役立ちます。このドキュメントの例では、メッシュごとに異なる色を使用します。各メッシュと以下のフェデレーションリソースの所有者および管理者を判断できるように、命名規則を決定する必要があります。
 - クラスター名
 - クラスターネットワーク名
 - メッシュ名と namespace
 - フェデレーション Ingress ゲートウェイ
 - フェデレーション egress ゲートウェイ
 - セキュリティー信頼ドメイン



注記

フェデレーションの各メッシュには、一意の信頼ドメインが必要です。

- 各メッシュのどのサービスをフェデレーションメッシュにエクスポートする予定ですか？各サービスは個別にエクスポートすることも、ラベルを指定したり、ワイルドカードを使用したりすることもできます。
 - サービスの namespace にエイリアスを使用しますか？
 - エクスポートされたサービスにエイリアスを使用しますか？
- 各メッシュでどのエクスポートサービスを、インポートする予定ですか？各メッシュは必要なサービスのみをインポートします。
 - インポートしたサービスにエイリアスを使用しますか？

1.18.7. クラスター全体でのメッシュフェデレーション

OpenShift Service Mesh のインスタンスを別のクラスターで実行されているインスタンスに接続するには、同じクラスターにデプロイされた 2 つのメッシュに接続する場合とほぼ変わりません。ただし、メッシュの Ingress ゲートウェイに、他のメッシュから到達できる必要があります。確実に到達できるようにするには、クラスターがこのタイプのサービスをサポートする場合に、ゲートウェイサービスを **LoadBalancer** サービスとして設定します。

サービスは、OSI モデルのレイヤー 4 で動作するロードバランサー経由で公開する必要があります。

1.18.7.1. ベアメタルで実行されるクラスターでのフェデレーション Ingress の公開

クラスターがベアメタルで実行され、**LoadBalancer** サービスを完全にサポートする場合は、ingress ゲートウェイ **Service** オブジェクトの **.status.loadBalancer.ingress.ip** フィールドにある IP アドレスを **ServiceMeshPeer** オブジェクトの **.spec.remote.addresses** フィールドにあるエントリーの1つとして指定する必要があります。

クラスターが **LoadBalancer** サービスをサポートしない場合は、他のメッシュを実行するクラスターからノードにアクセスできるのであれば **NodePort** サービスを使用することも可能です。**ServiceMeshPeer** オブジェクトで、**.spec.remote.addresses** フィールドのノードの IP アドレスと、**.spec.remote.discoveryPort** と **.spec.remote.servicePort** フィールドのサービスのノードポートを指定します。

1.18.7.2. Amazon Web Services(AWS) でのフェデレーション Ingress の公開

デフォルトでは、AWS で実行されるクラスターの LoadBalancer サービスで L4 負荷分散はサポートされていません。Red Hat OpenShift Service Mesh フェデレーションを正常に機能させるには、以下のアノテーションを Ingress ゲートウェイサービスに追加する必要があります。

```
service.beta.kubernetes.io/aws-load-balancer-type: nlb
```

ingress ゲートウェイ **Service** オブジェクトの **.status.loadBalancer.ingress.hostname** フィールドにある完全修飾ドメイン名は、**ServiceMeshPeer** オブジェクトの **.spec.remote.addresses** フィールドにあるエントリーの1つとして指定する必要があります。

1.18.8. フェデレーション実装のチェックリスト

Service Mesh のフェデレーションには、以下のアクティビティーが含まれます。

- ☐ フェデレーションする予定のクラスター間のネットワークを設定する。
 - ☐ 生の TLS トラフィックをサポートするように、フェデレーションゲートウェイに関連付けられたサービスをサポートするロードバランサーを設定する。
- ☐ Red Hat OpenShift Service Mesh バージョン 2.1 以降の Operator を各クラスターにインストールする。
- ☐ バージョン 2.1 以降の **ServiceMeshControlPlane** を各クラスターにデプロイする。
- ☐ フェデレーションする各メッシュのフェデレーションに SMCP を設定する。
 - ☐ フェデレーションする各メッシュにフェデレーション egress ゲートウェイを作成する。
 - ☐ フェデレーションする各メッシュにフェデレーション Ingress ゲートウェイを作成する。
 - ☐ 一意の信頼ドメインを設定する。
- ☐ 各メッシュのペアの **ServiceMeshPeer** リソースを作成して、2 つ以上のメッシュをフェデレーションする。
- ☐ **ExportedServiceSet** リソースを作成してサービスをエクスポートし、1 つのメッシュからピアメッシュにサービスが利用できるようにします。
- ☐ **ImportedServiceSet** リソースを作成してサービスをインポートし、メッシュピアで共有されるサービスをインポートします。

1.18.9. フェデレーション用の Service Mesh コントロールプレーンの設定

メッシュをフェデレーションする前に、メッシュフェデレーションの **ServiceMeshControlPlane** を設定する必要があります。フェデレーションに所属する全メッシュは同等で、各メッシュは個別に管理されるため、フェデレーションに参加する各メッシュに SMCP を設定する必要があります。

以下の例では、**red-mesh** の管理者は **green-mesh** と **blue-mesh** の両方を使用して、フェデレーションに SMCP を設定します。

red-mesh のサンプル SMCP

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: red-mesh
  namespace: red-mesh-system
spec:
  version: v2.6
  runtime:
    defaults:
      container:
        imagePullPolicy: Always
  gateways:
    additionalEgress:
      egress-green-mesh:
        enabled: true
        requestedNetworkView:
          - green-network
        routerMode: sni-dnat
      service:
        metadata:
          labels:
            federation.maistra.io/egress-for: egress-green-mesh
      ports:
        - port: 15443
          name: tls
        - port: 8188
          name: http-discovery #note HTTP here
      egress-blue-mesh:
        enabled: true
        requestedNetworkView:
          - blue-network
        routerMode: sni-dnat
      service:
        metadata:
          labels:
            federation.maistra.io/egress-for: egress-blue-mesh
      ports:
        - port: 15443
          name: tls
        - port: 8188
          name: http-discovery #note HTTP here
    additionalIngress:
      ingress-green-mesh:
        enabled: true
        routerMode: sni-dnat
```

```

service:
  type: LoadBalancer
  metadata:
    labels:
      federation.maistra.io/ingress-for: ingress-green-mesh
  ports:
    - port: 15443
      name: tls
    - port: 8188
      name: https-discovery #note HTTPS here
ingress-blue-mesh:
  enabled: true
  routerMode: sni-dnat
  service:
    type: LoadBalancer
    metadata:
      labels:
        federation.maistra.io/ingress-for: ingress-blue-mesh
    ports:
      - port: 15443
        name: tls
      - port: 8188
        name: https-discovery #note HTTPS here
security:
  identity:
    type: ThirdParty
  trust:
    domain: red-mesh.local

```

表1.6 ServiceMeshControlPlane フェデレーション設定パラメーター

パラメーター	説明	値	デフォルト値
spec: cluster: name:	クラスターの名前。クラスター名は指定する必要はありませんが、トラブルシューティングに役立ちます。	文字列	該当なし
spec: cluster: network:	クラスターネットワークの名前。ネットワークの名前は指定する必要はありませんが、設定およびトラブルシューティングに役立ちます。	文字列	該当なし

1.18.9.1. フェデレーションゲートウェイについて

ゲートウェイ を使用してメッシュの受信トラフィックおよび送信トラフィックを管理することで、メッシュに入るか、メッシュを出るトラフィックを指定できます。

Ingress および egress ゲートウェイを使用して、Service Mesh (North-South トラフィック) に入退出するトラフィックを管理します。フェデレーションメッシュの作成時に、追加の Ingress/egress ゲートウェイを作成し、フェデレーションメッシュ間のサービス検出や通信を用意にして、Service Mesh 間の

トラフィックフロー (East-West トラフィック) を管理します。

メッシュ間の命名の競合を回避するには、各メッシュに個別の egress および ingress ゲートウェイを作成する必要があります。たとえば、**red-mesh** には、**green-mesh** および **blue-mesh** に移動するトラフィックに対して個別の egress ゲートウェイがあります。

表1.7 フェデレーションゲートウェイパラメーター

パラメーター	説明	値	デフォルト値
spec: gateways: additionalEgress: <egress_name>:	フェデレーションの各メッシュピアの egress ゲートウェイを追加で定義します。		
spec: gateways: additionalEgress: <egress_name>: enabled:	このパラメーターは、フェデレーションの egress を有効または無効にします。	true/false	true
spec: gateways: additionalEgress: <egress_name>: requestedNetwork View:	エクスポートされたサービスに関連付けられたネットワーク。	メッシュの SMCP で spec.cluster.network の値に設定します。それ以外の場合は、<ServiceMeshPeer-name>-network を使用します。たとえば、メッシュの ServiceMeshPeer リソースの名前が west の場合、ネットワークは west-network になります。	

パラメーター	説明	値	デフォルト値
<pre>spec: gateways: additionalEgress: <egress_name>: service: metadata: labels: federation.maistra.io/egress-for:</pre>	<p>フェデレーショントラフィックがクラスターのデフォルトのシステムゲートウェイを通過しないように、ゲートウェイに一意的ラベルを指定します。</p>		
<pre>spec: gateways: additionalEgress: <egress_name>: service: ports:</pre>	<p>TLS およびサービス検出用の port: と name: を指定するのに使用します。フェデレーショントラフィックは、サービストラフィック用の生の暗号化 TCP で構成されます。</p>	<p>ポート 15443 は、フェデレーションで TLS サービス要求を他のメッシュに送信するために必要です。ポート 8188 は、フェデレーションでサービス検出要求を他のメッシュに送信するために必要です。</p>	
<pre>spec: gateways: additionalIngress:</pre>	<p>フェデレーションで、各メッシュピアの追加の Ingress ゲートウェイを定義します。</p>		
<pre>spec: gateways: additionalIngress: <ingress_name>: enabled:</pre>	<p>このパラメーターは、フェデレーション Ingress を有効または無効にします。</p>	true/false	true

パラメーター	説明	値	デフォルト値
spec: gateways: additionalIngress: <ingress_name>: service: type:	Ingress ゲートウェイサービスは、OSI モデルのレイヤー 4 で動作し、一般公開されているロードバランサー経由で公開する必要があります。	LoadBalancer	
spec: gateways: additionalIngress: <ingress_name>: service: type:	クラスターが LoadBalancer サービスをサポートしていない場合、入力ゲートウェイサービスは NodePort サービスを介して公開できます。	NodePort	
spec: gateways: additionalIngress: <ingress_name>: service: metadata: labels: federation.maistra.io/ingress-for:	フェデレーショントラフィックがクラスターのデフォルトのシステムゲートウェイを通過しないように、ゲートウェイに一意のラベルを指定します。		
spec: gateways: additionalIngress: <ingress_name>: service: ports:	TLS およびサービス検出用の port: と name: を指定するのに使用します。フェデレーショントラフィックは、サービストラフィック用の生の暗号化 TCP で構成されます。フェデレーショントラフィックは、検出用に HTTPS で構成されます。	ポート 15443 は、フェデレーションの他のメッシュへの TLS サービス要求を受信するために必要です。ポート 8188 は、フェデレーションの他のメッシュへのサービス検出要求を受信するために必要です。	

パラメーター	説明	値	デフォルト値
spec: gateways: additionalIngress: <ingress_name>: service: ports: nodePort:	クラスターが LoadBalancer サービスをサポートしていない場合は、 nodePort を指定するために使用されます。	指定した場合は、 port: と name: 以外に、TLS とサービス検出の両方でこれが必須です。 nodePort: 30000-32767 の範囲である必要があります。	

次の例では、管理者は **NodePort** サービスを使用して グリーンメッシュ とのフェデレーション用に SMCP を設定しています。

NodePort の SMCP 例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: green-mesh
  namespace: green-mesh-system
spec:
  # ...
  gateways:
    additionalIngress:
      ingress-green-mesh:
        enabled: true
        service:
          type: NodePort
          metadata:
            labels:
              federation.maistra.io/ingress-for: ingress-green-mesh
          ports:
            - port: 15443
              nodePort: 30510
              name: tls
            - port: 8188
              nodePort: 32359
              name: https-discovery
```

1.18.9.2. フェデレーション信頼ドメインパラメーターについて

フェデレーションの各メッシュには、一意の信頼ドメインが必要です。この値は、ServiceMesh **Peer** リソースでメッシュフェデレーションを設定する時に使用されます。

```
kind: ServiceMeshControlPlane
metadata:
  name: red-mesh
  namespace: red-mesh-system
spec:
```

```
security:
trust:
  domain: red-mesh.local
```

表1.8 フェデレーションセキュリティパラメーター

パラメーター	説明	値	デフォルト値
spec: security: trust: domain:	メッシュの信頼ドメインの一意の名前を指定するために使用されます。ドメインは、フェデレーション内のすべてのメッシュで一意である必要があります。	<mesh-name>.local	該当なし

コンソールからの手順

この手順に従って、Red Hat OpenShift Service on AWS Web コンソールで **ServiceMeshControlPlane** を編集します。この例では、**red-mesh** をサンプルとして使用しています。

1. cluster-admin ロールを持つユーザーとして Red Hat OpenShift Service on AWS Web コンソールにログインします。
2. **Operators** → **Installed Operators** に移動します。
3. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクトを選択します。例: **red-mesh-system**
4. Red Hat OpenShift Service Mesh Operator をクリックします。
5. **Istio Service Mesh Control Plane** タブで、**ServiceMeshControlPlane** の名前 (**red-mesh** など) をクリックします。
6. **Create ServiceMeshControlPlane Details** ページで、**YAML** をクリックして設定を変更します。
7. **ServiceMeshControlPlane** を変更してフェデレーション Ingress および egress ゲートウェイを追加し、信頼ドメインを指定します。
8. **Save** をクリックします。

CLI からの手順

以下の手順に従って、コマンドラインで **ServiceMeshControlPlane** を作成するか、編集します。この例では、**red-mesh** をサンプルとして使用しています。

1. **cluster-admin** ロールを持つユーザーとして Red Hat OpenShift Service on AWS CLI にログインします。以下のコマンドを入力します。次に、プロンプトが表示されたら、ユーザー名とパスワードを入力します。

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

2. Service Mesh コントロールプレーンをインストールしたプロジェクト (例: red-mesh-system) に切り替えます。

```
$ oc project red-mesh-system
```

3. **ServiceMeshControlPlane** ファイルを編集し、フェデレーション Ingress および egress ゲートウェイを追加して信頼ドメインを指定します。
4. 以下のコマンドを実行して Service Mesh コントロールプレーンを編集します。ここで、**red-mesh-system** はシステムの namespace であり、**red-mesh** は **ServiceMeshControlPlane** オブジェクトの名前になります。

```
$ oc edit -n red-mesh-system smcp red-mesh
```

5. 以下のコマンドを実行して、Service Mesh コントロールプレーンのインストールのステータスを確認します。このコマンドでは、**red-mesh-system** がシステム namespace に置き換えます。

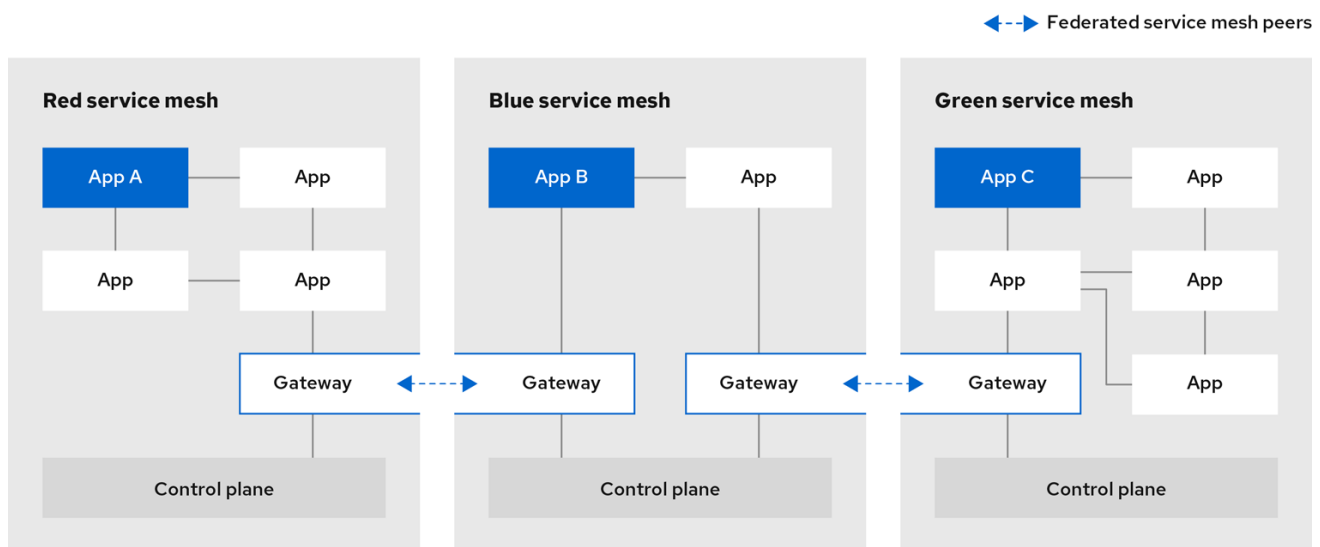
```
$ oc get smcp -n red-mesh-system
```

READY 列にすべてのコンポーネントが準備状態であることが示されると、インストールは正常に終了しています。

```
NAME      READY STATUS      PROFILES  VERSION AGE
red-mesh  10/10 ComponentsReady ["default"] 2.1.0 4m25s
```

1.18.10. フェデレーションメッシュへの参加

ServiceMeshPeer リソースを作成して、2つのメッシュ間のフェデレーションを宣言します。**ServiceMeshPeer** リソースは、2つのメッシュ間のフェデレーションを定義し、これを使用してピアメッシュの検出設定、ピアメッシュへのアクセス、および他のメッシュのクライアントの検証に使用される証明書を定義します。



182_OpenShift_0921

メッシュは1対1でフェデレーションされるため、ピアの各ペアでは、他のサービスメッシュへのフェデレーション接続を指定する **ServiceMeshPeer** リソースのペアが必要です。たとえば、**red** および **green** という名前の2つのメッシュには2つの **ServiceMeshPeer** ファイルが必要です。

1. red-mesh-system で、green メッシュの **ServiceMeshPeer** を作成します。

2. green-mesh-system で、Red メッシュの **ServiceMeshPeer** を作成します。

red、**blue** および **green** という名前の 3 つのメッシュのフェデレーションには 6 つの **ServiceMeshPeer** ファイルが必要になります。

1. red-mesh-system で、green メッシュの **ServiceMeshPeer** を作成します。
2. red-mesh-system で、blue メッシュの **ServiceMeshPeer** を作成します。
3. green-mesh-system で、Red メッシュの **ServiceMeshPeer** を作成します。
4. green-mesh-system で、blue メッシュの **ServiceMeshPeer** を作成します。
5. blue-mesh-system で、Red メッシュの **ServiceMeshPeer** を作成します。
6. blue-mesh-system で、green メッシュの **ServiceMeshPeer** を作成します。

ServiceMeshPeer リソースの設定には、以下が含まれます。

- 検出およびサービス要求に使用される他のメッシュの Ingress ゲートウェイのアドレス。
- 指定のピアメッシュとの対話に使用されるローカル ingress および egress ゲートウェイの名前。
- このメッシュへの要求の送信時に他のメッシュで使われるクライアント ID。
- 他のメッシュで使われる信頼ドメイン。
- **ConfigMap** の名前。これには、他のメッシュで使われる信頼ドメインのクライアント証明書の検証に使用するルート証明書が含まれます。

以下の例では、**red-mesh** の管理者は **green-mesh** でフェデレーションを設定します。

red-mesh の ServiceMeshPeer リソースの例

```
kind: ServiceMeshPeer
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  remote:
    addresses:
      - ingress-red-mesh.green-mesh-system.apps.domain.com
  gateways:
    ingress:
      name: ingress-green-mesh
    egress:
      name: egress-green-mesh
  security:
    trustDomain: green-mesh.local
    clientID: green-mesh.local/ns/green-mesh-system/sa/egress-red-mesh-service-account
    certificateChain:
      kind: ConfigMap
      name: green-mesh-ca-root-cert
```

表1.9 ServiceMeshPeer 設定パラメーター

パラメーター	説明	値
<code>metadata: name:</code>	このリソースがフェデレーションを設定するピアメッシュの名前。	文字列
<code>metadata: namespace:</code>	このメッシュのシステム namespace (Service Mesh コントロールプレーンのインストール先)。	文字列
<code>spec: remote: addresses:</code>	このメッシュからの要求に対応するピアメッシュの Ingress ゲートウェイのパブリックアドレスリスト。	
<code>spec: remote: discoveryPort:</code>	アドレスが検出要求を処理するポート。	デフォルトは 8188 です。
<code>spec: remote: servicePort:</code>	アドレスがサービス要求を処理するポート。	デフォルトは 15443 です。
<code>spec: gateways: ingress: name:</code>	ピアメッシュからの受信要求に対応するこのメッシュの Ingress の名前。例: ingress-green-mesh	
<code>spec: gateways: egress: name:</code>	ピアメッシュに送信される要求に対応するこのメッシュ上の egress の名前。例: egress-green-mesh	
<code>spec: security: trustDomain:</code>	ピアメッシュで使用する信頼ドメイン。	<code><peerMeshName>.local</code>
<code>spec: security: clientId:</code>	このメッシュの呼び出し時にピアメッシュが使用するクライアント ID。	<code><peerMeshTrustDomain>/ns/<peerMeshSystem>/sa/<peerMeshEgressGatewayName>-service-account</code>

パラメーター	説明	値
<pre>spec: security: certificateChain: kind: ConfigMap name:</pre>	ピアメッシュがこのメッシュに提示したクライアント証明書の検証に使用されるルート証明書が含まれるリソースの種類 (例: ConfigMap) と名前。証明書が含まれる config map エントリーの鍵は root-cert.pem である必要があります。	kind: ConfigMap name: <peerMesh>-ca-root-cert

1.18.10.1. ServiceMeshPeer リソースの作成

前提条件

- 2 つ以上の Red Hat OpenShift Service on AWS 4.6 以降のクラスター。
- クラスターのネットワーク設定が完了している。
- 生の TLS トラフィックをサポートするように、フェデレーションゲートウェイに関連付けられたサービスをサポートするロードバランサーを設定する必要があります。
- 各クラスターには、フェデレーションデプロイをサポートするようにバージョン 2.1 以降の **ServiceMeshControlPlane** が設定されている必要があります。
- **cluster-admin** ロールを持つアカウントがある。

CLI からの手順

以下の手順に従って、コマンドラインから **ServiceMeshPeer** リソースを作成します。以下の例では、**red-mesh** が **green-mesh** のピアリソースを作成しています。

1. **cluster-admin** ロールを持つユーザーとして Red Hat OpenShift Service on AWS CLI にログインします。以下のコマンドを入力します。次に、プロンプトが表示されたら、ユーザー名とパスワードを入力します。

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

2. コントロールプレーンをインストールしたプロジェクト (例: **red-mesh-system**) に切り替えます。

```
$ oc project red-mesh-system
```

3. フェデレーションを行う 2 つのメッシュに対して、以下の例に基づいて **ServiceMeshPeer** ファイルを作成します。

red-mesh から green-mesh への ServiceMeshPeer リソースのサンプル

```
kind: ServiceMeshPeer
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
```

```
spec:
  remote:
    addresses:
      - ingress-red-mesh.green-mesh-system.apps.domain.com
  gateways:
    ingress:
      name: ingress-green-mesh
    egress:
      name: egress-green-mesh
  security:
    trustDomain: green-mesh.local
    clientID: green-mesh.local/ns/green-mesh-system/sa/egress-red-mesh-service-account
    certificateChain:
      kind: ConfigMap
      name: green-mesh-ca-root-cert
```

4. 以下のコマンドを実行してリソースをデプロイします。ここで、**red-mesh-system** はシステムの namespace に置き換え、**servicemeshpeer.yaml** には編集したファイルへのフルパスが含まれます。

```
$ oc create -n red-mesh-system -f servicemeshpeer.yaml
```

5. red メッシュと green メッシュ間の接続確立を確認するには、red-mesh-system namespace の green-mesh **ServiceMeshPeer** のステータスを調べます。

```
$ oc -n red-mesh-system get servicemeshpeer green-mesh -o yaml
```

red-mesh と green-mesh 間の ServiceMeshPeer 接続の例

```
status:
  discoveryStatus:
    active:
      - pod: istiod-red-mesh-b65457658-9wq5j
        remotes:
          - connected: true
            lastConnected: "2021-10-05T13:02:25Z"
            lastFullSync: "2021-10-05T13:02:25Z"
            source: 10.128.2.149
        watch:
          connected: true
          lastConnected: "2021-10-05T13:02:55Z"
          lastDisconnectStatus: 503 Service Unavailable
          lastFullSync: "2021-10-05T13:05:43Z"
```

status.discoveryStatus.active.remotes フィールドは、ピアメッシュ (この例では green メッシュ) が現在のメッシュ (この例では red メッシュ) の istiod に接続されていることを示します。

status.discoveryStatus.active.watch フィールドは、現在のメッシュの istiod がピアメッシュで istiod に接続されていることを示します。

green-mesh-system で **red-mesh** という名前の **servicemeshpeer** を確認すると、green メッシュの観点からの 2 つの同じ接続に関する情報が表示されます。

2つのメッシュ間の接続が確立されていない場合、**ServiceMeshPeer** ステータスは、**status.discoveryStatus.inactive** フィールドにこれを示します。

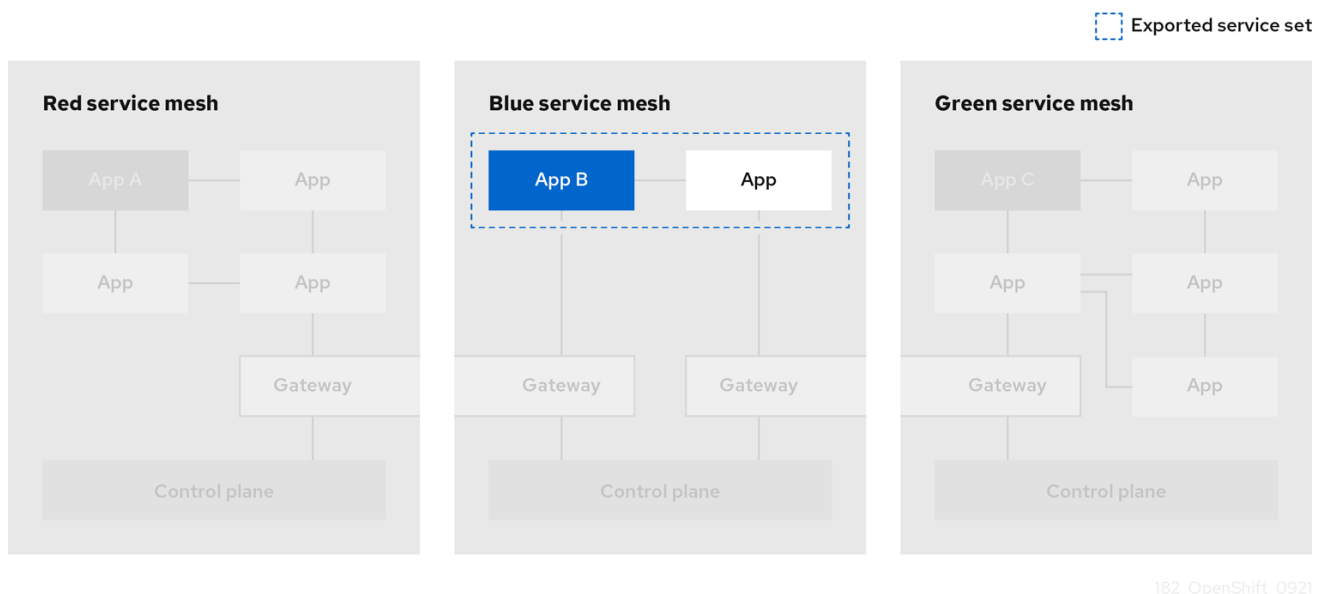
接続の試みが失敗した理由の詳細は、Istiod ログ、ピアの egress トラフィックを処理する egress ゲートウェイのアクセスログ、およびピアメッシュ内の現在のメッシュの ingress トラフィックを処理する ingress ゲートウェイのアクセスログを調べてください。

たとえば、red メッシュが green メッシュに接続できない場合は、以下のログを確認します。

- red-mesh-system の istiod-red-mesh
- red-mesh-system の Egress-green-mesh
- green-mesh-system の ingress-red-mesh

1.18.11. フェデレーションメッシュからのサービスのエクスポート

サービスをエクスポートすると、メッシュは、フェデレーションされたメッシュの別のメンバーとサービスを共有できます。



ExportedServiceSet リソースを使用して、フェデレーションメッシュ内の別のピアから利用できるように指定したメッシュからサービスを宣言します。ピアと共有される各サービスを明示的に宣言する必要があります。

- サービスは、namespace または名前別に選択できます。
- ワイルドカードを使用してサービスを選択できます。たとえば、namespace 内のすべてのサービスをエクスポートします。
- エイリアスを使用してサービスをエクスポートできます。たとえば、**foo/bar** サービスを **custom-ns/bar** としてエクスポートできます。
- メッシュのシステム namespace に表示されるサービスのみをエクスポートできます。たとえば、**networking.istio.io/exportTo** ラベルが '!' に設定された別の namespace のサービスは、エクスポートの候補にはなりません。

- エクスポートされたサービスの場合、それらのターゲットサービスは、元の要求元 (他のメッシュの egress ゲートウェイや、要求元のワークロードのクライアント ID は表示されない) ではなく、ingress ゲートウェイからのトラフィックのみが表示されます。

以下の例は、**red-mesh** が **green-mesh** にエクスポートするサービス向けです。

ExportedServiceSet リソースの例

```
kind: ExportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  exportRules:
    # export ratings.mesh-x-info as ratings.bookinfo
    - type: NameSelector
      nameSelector:
        namespace: red-mesh-info
        name: red-ratings
        alias:
          namespace: info
          name: ratings
    # export any service in red-mesh-info namespace with label export-service=true
    - type: LabelSelector
      labelSelector:
        namespace: red-mesh-info
        selector:
          matchLabels:
            export-service: "true"
        aliases: # export all matching services as if they were in the info namespace
        - namespace: "*"
          name: "*"
          alias:
            namespace: info
```

表1.10 ExportedServiceSet パラメーター

パラメーター	説明	値
metadata: name:	このサービスを公開する ServiceMeshPeer の名前。	ServiceMeshPeer リソースのメッシュの name 値と一致する必要があります。
metadata: namespace:	このリソースを含むプロジェクト/namespace の名前 (メッシュのシステム namespace を指定する必要があります)。	
spec: exportRules: - type:	このサービスのエクスポートを管理するルールタイプ。サービスで最初に一致するルールがエクスポートに使用されます。	NameSelector, LabelSelector

パラメーター	説明	値
<pre>spec: exportRules: - type: NameSelector nameSelector: namespace: name:</pre>	NameSelector ルールを作成するには、サービスの namespace およびサービスの namespace を Service リソースで定義されたとおり指定します。	
<pre>spec: exportRules: - type: NameSelector nameSelector: alias: namespace: name:</pre>	サービスの namespace と name を指定した後、 namespace のエイリアスと、サービスの name として使用するエイリアスを指定し、このサービスのエイリアスに使用する NameSelector ルールを作成します。	
<pre>spec: exportRules: - type: LabelSelector labelSelector: namespace: <exportingMesh> selector: matchLabels: <labelKey>: <labelValue></pre>	LabelSelector ルールを作成するには、サービスの namespace を指定し、 Service リソースで定義されているラベルを指定します。上記の例では、ラベルは export-service です。	
<pre>spec: exportRules: - type: LabelSelector labelSelector: namespace: <exportingMesh> selector: matchLabels: <labelKey>: <labelValue> aliases: - namespace: name: alias: namespace: name:</pre>	サービスのエイリアスを使用する LabelSelector ルールを作成するには、 selector を指定した後で、サービスの name または namespace に使用するエイリアスを指定します。上記の例では、一致するすべてのサービスの namespace エイリアスは info です。	

red-mesh のすべての namespace から blue-mesh へ "ratings" という名前のサービスをエクスポートします。

```
kind: ExportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: blue-mesh
  namespace: red-mesh-system
spec:
  exportRules:
  - type: NameSelector
    nameSelector:
      namespace: "*"
      name: ratings
```

すべてのサービスを `west-data-center` namespace から `green-mesh` にエクスポートします。

```
kind: ExportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  exportRules:
  - type: NameSelector
    nameSelector:
      namespace: west-data-center
      name: "*"

```

1.18.11.1. ExportedServiceSet の作成

ExportedServiceSet リソースを作成し、メッシュピアで利用可能なサービスを明示的に宣言します。

サービスは `<export-name>.<export-namespace>.svc.<ServiceMeshPeer.name>-exports.local` としてエクスポートされ、ターゲットサービスに自動的にルーティングされます。これは、エクスポートメッシュでエクスポートされたサービスで認識される名前です。Ingress ゲートウェイが宛先がこの名前の要求を受信すると、エクスポートされる実際のサービスにルーティングされます。たとえば、`ratings.red-mesh-info` という名前のサービスが `ratings.bookinfo` として `green-mesh` にエクスポートされると、サービスは `ratings.bookinfo.svc.green-mesh-exports.local` という名前でエクスポートされ、そのホスト名の ingress ゲートウェイが受信するトラフィックは `ratings.red-mesh-bookinfo` サービスにルーティングされます。

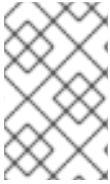


注記

importAsLocal パラメーターを **true** に設定して、リモートエンドポイントをローカルサービスで集約する場合は、サービスのエイリアスを使用する必要があります。パラメーターを **false** に設定した場合、エイリアスは必要ありません。

前提条件

- クラスタおよび **ServiceMeshControlPlane** がメッシュフェデレーション用に設定されている。
- **cluster-admin** ロールを持つアカウントがある。



注記

サービスがまだ存在していない場合でも、エクスポート用にサービスを設定できます。ExportedServiceSet で指定された値に一致するサービスがデプロイされ、自動的にエクスポートされます。

CLI からの手順

以下の手順に従って、コマンドラインから **ExportedServiceSet** を作成します。

1. **cluster-admin** ロールを持つユーザーとして Red Hat OpenShift Service on AWS CLI にログインします。以下のコマンドを入力します。次に、プロンプトが表示されたら、ユーザー名とパスワードを入力します。

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

2. Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **red-mesh-system**) に切り替えます。

```
$ oc project red-mesh-system
```

3. 以下の例に基づいて、**ExportedServiceSet** ファイルを作成します。ここでは、**red-mesh** がサービスを **green-mesh** にエクスポートします。

red-mesh から green-mesh への ExportedServiceSet リソースの例

```
apiVersion: federation.maistra.io/v1
kind: ExportedServiceSet
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  exportRules:
    - type: NameSelector
      nameSelector:
        namespace: red-mesh-info
        name: ratings
      alias:
        namespace: info
        name: red-ratings
    - type: NameSelector
      nameSelector:
        namespace: red-mesh-info
        name: reviews
```

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

```
$ oc create -n <ControlPlaneNamespace> -f <ExportedServiceSet.yaml>
```

以下に例を示します。

```
$ oc create -n red-mesh-system -f export-to-green-mesh.yaml
```

5. フェデレーションメッシュのメッシュピアごとに、必要に応じて追加の **ExportedServiceSets** を作成します。

検証

- 以下のコマンドを実行して、red-mesh が green-mesh と共有するためにエクスポートしたサービスを検証します。

```
$ oc get exportedserviceset <PeerMeshExportedTo> -o yaml
```

以下に例を示します。

```
$ oc -n red-mesh-system get exportedserviceset green-mesh -o yaml
```

red メッシュからエクスポートして green メッシュに共有したサービスの検証例。

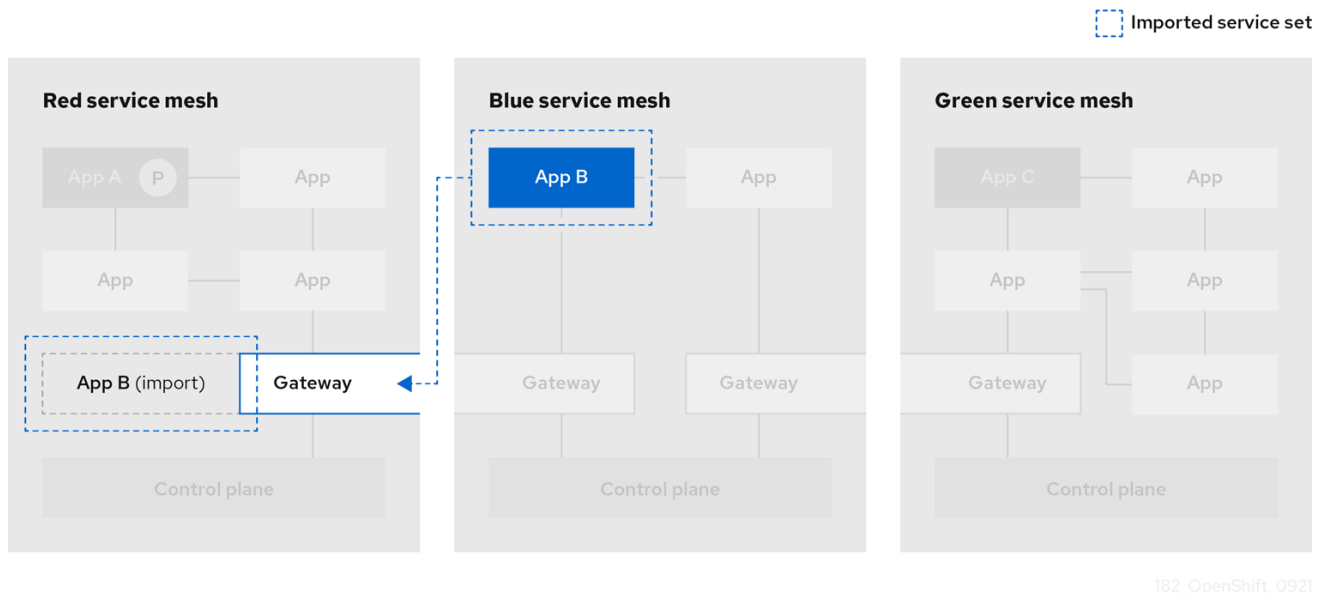
```
status:
  exportedServices:
  - exportedName: red-ratings.info.svc.green-mesh-exports.local
    localService:
      hostname: ratings.red-mesh-info.svc.cluster.local
      name: ratings
      namespace: red-mesh-info
  - exportedName: reviews.red-mesh-info.svc.green-mesh-exports.local
    localService:
      hostname: reviews.red-mesh-info.svc.cluster.local
      name: reviews
      namespace: red-mesh-info
```

status.exportedServices 配列では、現在エクスポートされているサービス (これらのサービスは **ExportedServiceSet** オブジェクト のエクスポートルールに一致) をリスト表示します。配列の各エントリーは、エクスポートされたサービスの名前と、エクスポートされたローカルサービスの詳細を示します。

エクスポート予定のサービスがない場合は、Service オブジェクトが存在すること、その名前またはラベルが **ExportedServiceSet** オブジェクトで定義される **exportRules** と一致すること、および Service オブジェクトの namespace が **ServiceMeshMemberRoll** または **ServiceMeshMember** オブジェクトを使用して Service Mesh のメンバーとして設定されることを確認します。

1.18.12. サービスのフェデレーションメッシュへのインポート

サービスをインポートすると、別のメッシュからエクスポートされたサービスの内、サービスメッシュ内でアクセスできるものを明示的に指定できます。



ImportedServiceSet リソースを使用して、インポートするサービスを選択します。メッシュピアによってエクスポートされ、明示的にインポートされるサービスのみがメッシュで利用できます。明示的にインポートしない場合、サービスは、メッシュ内で利用できません。

- サービスは、namespace または名前別に選択できます。
- namespace にエクスポートされたすべてのサービスをインポートするなど、ワイルドカードを使用してサービスを選択できます。
- メッシュにグローバルであるか、特定のメンバーの namespace の範囲内にあるラベルセクターを使用してエクスポートするサービスを選択できます。
- エイリアスを使用してサービスをインポートできます。たとえば、**custom-ns/bar** サービスを **other-mesh/bar** としてインポートできます。
- カスタムドメイン接尾辞を指定できます。これは、**bar.other-mesh.imported.local** など、インポートされたサービスの **name.namespace** に、完全修飾ドメイン名として追加されます。

以下の例は、**red-mesh** でエクスポートされたサービスをインポートする **green-mesh** の例です。

ImportedServiceSet の例

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh #name of mesh that exported the service
  namespace: green-mesh-system #mesh namespace that service is being imported into
spec:
  importRules: # first matching rule is used
    # import ratings.info as ratings.bookinfo
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: info
      name: ratings
      alias:
```

```
# service will be imported as ratings.info.svc.red-mesh-imports.local
namespace: info
name: ratings
```

表1.11 ImportedServiceSet パラメーター

パラメーター	説明	値
metadata: name:	サービスをフェデレーションメッシュにエクスポートした ServiceMeshPeer の名前。	
metadata: namespace:	ServiceMeshPeer リソース (メッシュシステム namespace) を含む namespace の名前。	
spec: importRules: - type:	サービスのインポートを管理するルールタイプ。サービスで最初に一致するルールがインポートに使用されます。	NameSelector
spec: importRules: - type: NameSelector nameSelector: namespace: name:	NameSelector ルールを作成するには、 namespace およびエクスポートされたサービスの name を指定します。	
spec: importRules: - type: NameSelector importAsLocal:	リモートエンドポイントをローカルサービスで集約するには、 true に設定します。 true の場合、サービスは <name> . <namespace>.svc.cluster.local としてインポートされます。 true の場合、エイリアスが必要です。 false の場合、エイリアスは必要ありません。	true/false
spec: importRules: - type: NameSelector nameSelector: namespace: name: alias: namespace: name:	サービスの namespace と name を指定した後、 namespace のエイリアスと、サービスの name として使用するエイリアスを指定し、このサービスのエイリアスに使用する NameSelector ルールを作成します。	

red-mesh から blue-mesh への "info/ratings" サービスのインポート

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh
  namespace: blue-mesh-system
spec:
  importRules:
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: info
      name: ratings
```

red-mesh の west-data-center namespace からすべてのサービスを green-mesh にインポートします。これらのサービスは、<name>.west-data-center.svc.red-mesh-imports.local としてアクセスできます。

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh
  namespace: green-mesh-system
spec:
  importRules:
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: west-data-center
      name: "*"
```

1.18.12.1. ImportedServiceSet の作成

ImportedServiceSet リソースを作成し、メッシュにインポートするサービスを明示的に宣言します。

サービスは、<exported-name>.<exported-namespace>.svc.<ServiceMeshPeer.name>.remote という名前でインポートされます。これは "非表示" のサービスで、egress ゲートウェイ namespace にのみ表示され、エクスポートされたサービスのホスト名に関連付けられます。このサービスは、ローカルから <export-name>.<export-namespace>.<domainSuffix> として利用可能になります。ここでは、**importAsLocal** が **true** に設定されていない限り、**domainSuffix** はデフォルトで **svc.<ServiceMeshPeer.name>-imports.local** となっています。true の場合、**domainSuffix** は **svc.cluster.local** となります。**importAsLocal** が **false** に設定されている場合は、インポートルールのドメイン接尾辞が適用されます。ローカルインポートは、メッシュ内の他のサービスと同様に扱うことができます。これは egress ゲートウェイを介して自動的にルーティングされ、エクスポートされたサービスのリモート名にリダイレクトされます。

前提条件

- クラスターおよび **ServiceMeshControlPlane** がメッシュフェデレーション用に設定されている。
- **cluster-admin** ロールを持つアカウントがある。



注記

サービスがまだエクスポートされていない場合でも、インポートするように設定できます。ImportedServiceSet で指定された値に一致するサービスがデプロイされてエクスポートされると、これは自動的にインポートされます。

手順

この手順に従って、コマンドラインから **ImportedServiceSet** を作成します。

1. **cluster-admin** ロールを持つユーザーとして Red Hat OpenShift Service on AWS CLI にログインします。以下のコマンドを入力します。次に、プロンプトが表示されたら、ユーザー名とパスワードを入力します。

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

2. Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **green-mesh-system**) に切り替えます。

```
$ oc project green-mesh-system
```

3. 以下の例に基づいて **ImportedServiceSet** ファイルを作成します。ここでは、**green-mesh** が、**red-mesh** によって以前にエクスポートされたサービスをインポートします。

red-mesh から green-mesh への ImportedServiceSet リソースの例

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh
  namespace: green-mesh-system
spec:
  importRules:
  - type: NameSelector
    importAsLocal: false
    nameSelector:
      namespace: info
      name: red-ratings
    alias:
      namespace: info
      name: ratings
```

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

```
$ oc create -n <ControlPlaneNamespace> -f <ImportedServiceSet.yaml>
```

以下に例を示します。

```
$ oc create -n green-mesh-system -f import-from-red-mesh.yaml
```

5. フェデレーションメッシュ内のメッシュピアごとに、必要に応じて追加の **ImportedServiceSet** リソースを作成します。

検証

- 次のコマンドを実行して、サービスが **green-mesh** にインポートされたことを確認します。

```
$ oc get importedserviceset <PeerMeshImportedInto> -o yaml
```

'green-mesh-system' namespace の **importedserviceset/red-mesh** オブジェクトの **status** セクションを使用して、red mesh からエクスポートされたサービスが green mesh にインポートされたことを確認する例

```
$ oc -n green-mesh-system get importedserviceset/red-mesh -o yaml
```

```
status:
  importedServices:
  - exportedName: red-ratings.info.svc.green-mesh-exports.local
    localService:
      hostname: ratings.info.svc.red-mesh-imports.local
      name: ratings
      namespace: info
  - exportedName: reviews.red-mesh-info.svc.green-mesh-exports.local
    localService:
      hostname: ""
      name: ""
      namespace: ""
```

上記の例では、**localService** の入力済みフィールドで示されているように、ratings サービスのみがインポートされます。Reviews サービスはインポートできますが、**ImportedServiceSet** オブジェクトの **importRules** と一致しないため、現時点ではインポートされません。

1.18.13. フェイルオーバー用のフェデレーションメッシュの設定

フェイルオーバーとは、別のサーバーなどの信頼性の高いバックアップシステムに自動的かつシームレスに切り替える機能です。フェデレーションメッシュの場合は、あるメッシュのサービスを設定して、別のメッシュのサービスにフェイルオーバーできます。

ImportedServiceSet リソースで **importAsLocal** と **locality** の設定を指定してから、**ImportedServiceSet** で指定されたローカリティへのサービスのフェイルオーバーを設定する **DestinationRule** を指定することにより、フェイルオーバーのフェデレーションを設定します。

前提条件

- 2 つ以上の Red Hat OpenShift Service on AWS 4.6 以降のクラスターがすでにネットワーク化およびフェデレーションされている。
- フェデレーションメッシュのメッシュピアごとにすでに作成されている **ExportedServiceSet** リソース。
- フェデレーションメッシュのメッシュピアごとにすでに作成されている **ImportedServiceSet** リソース。
- **cluster-admin** ロールを持つアカウントがある。

1.18.13.1. フェイルオーバー用の ImportedServiceSet の設定

ローカリティ加重負荷分散を使用すると、管理者は、トラフィックの発信元と終了場所のローカリティに基づいて、エンドポイントへのトラフィックの分散を制御できます。これらのローカリティは、`{region}/{zone}/{sub-zone}` 形式でローカリティの階層を指定する任意のラベルを使用して指定します。

このセクションの例では、**green-mesh**は米国**us-east**地域にあり、**us-east**は**us-east**地域にあります。

red-mesh から green-mesh への ImportedServiceSet リソースの例

```
kind: ImportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: red-mesh #name of mesh that exported the service
  namespace: green-mesh-system #mesh namespace that service is being imported into
spec:
  importRules: # first matching rule is used
    # import ratings.info as ratings.bookinfo
  - type: NameSelector
    importAsLocal: true
    nameSelector:
      namespace: info
      name: ratings
    alias:
      # service will be imported as ratings.info.svc.red-mesh-imports.local
      namespace: info
      name: ratings
    #Locality within which imported services should be associated.
  locality:
    region: us-west
```

表1.12 ImportedServiceLocality フィールドテーブル

名前	説明	型
region:	インポートされたサービスが配置されている地域。	string
サブゾーン:	インポートされたサービスが配置されているサブゾーン。サブゾーンが指定されている場合は、ゾーンも指定する必要があります。	string
zone:	インポートされたサービスが配置されているゾーン。ゾーンを指定する場合は、リージョンも指定する必要があります。	string

手順

1. **cluster-admin** ロールを持つユーザーとして Red Hat OpenShift Service on AWS CLI にログインし、次のコマンドを入力します。

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

2. Service Mesh コントロールプレーンをインストールしたプロジェクトに変更し、次のコマンドを入力します。

```
$ oc project <smcp-system>
```

たとえば、**green-mesh-system** です。

```
$ oc project green-mesh-system
```

3. **ImportedServiceSet** ファイルを編集します。ここで、**<ImportedServiceSet.yaml>** には、編集するファイルへのフルパスが含まれています。以下のコマンドを入力してください。

```
$ oc edit -n <smcp-system> -f <ImportedServiceSet.yaml>
```

たとえば、前の **ImportedServiceSet** の例で示したように、red-mesh-system から green-mesh-system にインポートするファイルを変更する場合は、以下のようになります。

```
$ oc edit -n green-mesh-system -f import-from-red-mesh.yaml
```

4. ファイルを変更します。

- a. **spec.importRules.importAsLocal** を **true** に設定します。
- b. **spec.locality** を **region**、**zone**、または **subzone** に設定します。
- c. 変更を保存します。

1.18.13.2. フェイルオーバー用の DestinationRule の設定

以下を設定する **DestinationRule** リソースを作成します。

- サービスの外れ値の検出。これは、フェイルオーバーを正しく機能させるには必要です。特に、サービスのエンドポイントが異常である場合は把握できるようにサイドカープロキシを設定し、最終的に次のローカリティへのフェイルオーバーをトリガーします。
- リージョン間のフェイルオーバーポリシー。これにより、リージョンの境界を超えたフェイルオーバーが予測どおりに動作することが保証されます。

手順

1. **cluster-admin** ロールを持つユーザーとして Red Hat OpenShift Service on AWS CLI にログインします。以下のコマンドを入力します。次に、プロンプトが表示されたら、ユーザー名とパスワードを入力します。

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

2. Service Mesh コントロールプレーンをインストールしたプロジェクトに変更します。

```
$ oc project <smcp-system>
```

たとえば、**green-mesh-system** です。

```
$ oc project green-mesh-system
```

3. 次の例に基づいて **DestinationRule** ファイルを作成します。ここで、green-mesh が使用できない場合、トラフィックは **us-east** リージョンの green-mesh から **us-west** の red-mesh にルーティングされます。

DestinationRule の例

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: default-failover
  namespace: info
spec:
  host: "ratings.info.svc.cluster.local"
  trafficPolicy:
    loadBalancer:
      localityLbSetting:
        enabled: true
        failover:
          - from: us-east
            to: us-west
    outlierDetection:
      consecutive5xxErrors: 3
      interval: 10s
      baseEjectionTime: 1m
```

4. **DestinationRule** をデプロイします。次のコマンドを入力します。<**DestinationRule**> にはファイルへのフルパスを追加します。

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

以下に例を示します。

```
$ oc create -n info -f green-mesh-us-west-DestinationRule.yaml
```

1.18.14. フェデレーションメッシュからのサービスの削除

フェデレーションメッシュからサービスを削除する必要がある場合 (非推奨になるか、別のサービスで置き換えられる場合など)、削除できます。

1.18.14.1. 単一のメッシュからサービスを削除するには、以下を実行します。

サービスにアクセスすべきでないメッシュピアの **ImportedServiceSet** リソースからサービスのエントリーを削除します。

1.18.14.2. フェデレーションメッシュ全体からサービスを削除するには、以下を実行します。

サービスを所有するメッシュの **ExportedServiceSet** リソースからサービスのエントリーを削除します。

1.18.15. フェデレーションメッシュからのメッシュの削除

フェデレーションからメッシュを削除する必要がある場合は、削除できます。

1. 削除されたメッシュの **ServiceMeshControlPlane** リソースを編集して、ピアメッシュのすべてのフェデレーション Ingress ゲートウェイを削除します。
2. 削除されたメッシュがフェデレーションされているメッシュピアごとに、以下を実行します。
 - a. 2つのメッシュをリンクする **ServiceMeshPeer** リソースを削除します。
 - b. ピアメッシュの **ServiceMeshControlPlane** リソースを編集して、削除されたメッシュに対応する egress ゲートウェイを削除します。

1.19. 拡張

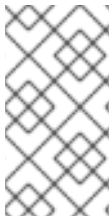
WebAssembly エクステンションを使用して、Red Hat OpenShift Service Mesh プロキシに直接新しい機能を追加できます。これにより、お使いのアプリケーションから、さらに一般的な機能を移動して、単一の言語で実装して、WebAssembly bytecode にコンパイルできます。

1.19.1. WebAssembly モジュールの概要

WebAssembly モジュールは、プロキシなどの多くのプラットフォームで実行でき、これには、幅広い言語サポート、高速実行、および sandboxed-by-default (デフォルトでサンドボックス化される) セキュリティモデルが含まれます。

Red Hat OpenShift Service Mesh エクステンションとして [Envoy HTTP フィルター](#) を使用でき、幅広い機能を提供します。

- 要求と応答の本体とヘッダーの操作
- 認証やポリシーのチェックなど、要求パスにないサービスへの帯域外 HTTP 要求
- 相互に通信するフィルター用のサイドチャネルデータストレージおよびキュー



注記

新しい WebAssembly エクステンションを作成するときは、**WasmPlugin** API を使用してください。**ServiceMeshExtension** API は Red Hat OpenShift Service Mesh バージョン 2.2 で非推奨化され、Red Hat OpenShift Service Mesh バージョン 2.3 で廃止されました。

Red Hat OpenShift Service Mesh エクステンションの作成には 2 つの部分があります。

1. [proxy-wasm API](#) を公開する SDK を使用してエクステンションを記述し、それを WebAssembly モジュールにコンパイルする必要があります。
2. 次に、モジュールをコンテナにパッケージ化する必要があります。

サポートされる言語

WebAssembly バイトコードにコンパイルする言語を使用して Red Hat OpenShift Service Mesh 拡張を作成できますが、以下の言語には proxy-wasm API を公開する既存の SDK があるため、これを直接使用できます。

表1.13 サポートされる言語

言語	保守管理者	リポジトリ
AssemblyScript	solo.io	solo-io/proxy-runtime
C++	proxy-wasm チーム (Istio コミュニティー)	proxy-wasm/proxy-wasm-cpp-sdk
Go	tetrade.io	tetratelabs/proxy-wasm-go-sdk
Rust	proxy-wasm チーム (Istio コミュニティー)	proxy-wasm/proxy-wasm-rust-sdk

1.19.2. WasmPlugin コンテナ形式

Istio は、Wasm プラグインメカニズムで Open Container Initiative (OCI) イメージをサポートしています。Wasm プラグインをコンテナイメージとして配布でき、**spec.url** フィールドを使用してコンテナレジストリーの場所を参照できます。たとえば、**quay.io/my-username/my-plugin:latest** です。

WASM モジュールの各実行環境 (ランタイム) にはランタイム固有の設定パラメーターを設定できるため、WASM イメージは次の 2 つの階層で構成できます。

- **plugin.wasm** (必須) - コンテンツレイヤー。このレイヤーは、ランタイムによってロードされる WebAssembly モジュールのバイトコードを含む **.wasm** バイナリーで構成されます。このファイルには **plugin.wasm** という名前を付ける必要があります。
- **runtime-config.json** (オプション) - 設定レイヤー。このレイヤーは、ターゲットランタイムのモジュールに関するメタデータを記述する JSON 形式の文字列で構成されます。ターゲットランタイムによっては、設定レイヤーに追加のデータが含まれる場合もあります。たとえば、WASM Envoy Filter の設定には、フィルターで使用可能な **root_ids** が含まれています。

1.19.3. WasmPlugin API リファレンス

WasmPlugins API には、WebAssembly フィルターを介して Istio プロキシによって提供される機能を拡張するメカニズムがあります。

複数の WasmPlugin をデプロイできます。**phase** および **priority** の設定により、実行の順序が (Envoy のフィルターチェーンの一部として) 決定され、ユーザー提供の WasmPlugins と Istio の内部フィルター間の複雑な対話設定が可能になります。

次の例では、認証フィルターが OpenID フローを実装し、Authorization ヘッダーに JSON Web Token (JWT) を入力します。Istio 認証はこのトークンを消費し、ingress ゲートウェイにデプロイします。WasmPlugin ファイルはプロキシサイドカーファイルシステムに存在します。フィールドの **URL** に注意してください。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: openid-connect
  namespace: istio-ingress
spec:
  selector:
    matchLabels:
      istio: ingressgateway
```



```

url: file:///opt/filters/openid.wasm
sha256: 1ef0c9a92b0420cf25f7fe5d481b231464bc88f486ca3b9c83ed5cc21d2f6210
phase: AUTHN
pluginConfig:
  openid_server: authn
  openid_realm: ingress

```

以下は同じ例ですが、今回はファイルシステム内のファイルの代わりに Open Container Initiative (OCI) イメージが使用されています。フィールド **url**、**imagePullPolicy**、および **imagePullSecret** に注意してください。

```

apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: openid-connect
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  url: oci://private-registry:5000/openid-connect/openid:latest
  imagePullPolicy: IfNotPresent
  imagePullSecret: private-registry-pull-secret
  phase: AUTHN
  pluginConfig:
    openid_server: authn
    openid_realm: ingress

```

表1.14 WasmPlugin フィールドリファレンス

フィールド	型	説明	必須
spec.selector	WorkloadSelector	このプラグイン設定を適用する必要がある Pod/VM の特定のセットを選択するために使用される基準。省略した場合に、この設定は同じ namespace 内のすべてのワークロードインスタンスに適用されます。 WasmPlugin フィールドが config root namespace に存在する場合は、任意の namespace の該当するすべてのワークロードに適用されます。	いいえ

フィールド	型	説明	必須
spec.url	string	Wasm モジュールまたは OCI コンテナの URL。スキームが存在しない場合は、デフォルトで oci:// になり、OCI イメージを参照します。その他の有効なスキームとして、プロキシコンテナ内にローカルに存在する.wasm モジュール ファイルを参照するための file:// と、リモートでホストされる.wasm モジュール ファイルを参照するための http[s]:// があります。	いいえ
spec.sha256	string	Wasm モジュールまたは OCI コンテナの検証に使用される SHA256 チェックサム。 url フィールドがすでに SHA256 を参照している場合 (@sha256: 表記を使用) には、このフィールドの値と一致する必要があります。OCI イメージがタグによって参照され、このフィールドが設定されている場合は、プル後にそのチェックサムがこのフィールドの内容に対して検証されます。	いいえ

フィールド	型	説明	必須
spec.imagePullPolicy	PullPolicy	OCI イメージをフェッチするときに適用されるプル動作。イメージがSHA ではなくタグで参照されている場合にのみ参照します。デフォルト値は IfNotPresent です。ただし、 URL フィールドで OCI イメージが参照され、 latest タグが使用されている場合は、 Always の値がデフォルトで、K8s の動作を反映しています。 url フィールドが file:// または http[s]:// を使用して Wasm モジュールを直接参照している場合は、設定が無視されます。	いいえ
spec.imagePullSecret	string	OCI イメージのプルに使用するクレデンシャル。イメージをプルするときにレジストリーに対して認証するためのプルシークレットなど、 WasmPlugin オブジェクトと同じ namespace 内のシークレットの名前。	いいえ
spec.phase	PluginPhase	フィルターチェーンのどこにこの WasmPlugin オブジェクトを挿入するかを決定します。	いいえ

フィールド	型	説明	必須
spec.priority	int64	phase の値が同じ WasmPlugins オブジェクトの順序を決定します。複数の WasmPlugins オブジェクトが同じフェーズで同じワークロードに適用される場合、それらは優先度と降順をもとに適用されます。優先度フィールドが設定されていない場合や値が同じ WasmPlugins オブジェクトが2つある場合、順序は WasmPlugins オブジェクトの名前と namespace をもとに決定されます。デフォルト値は 0 です。	いいえ
spec.pluginName	string	Envoy 設定で使用するプラグイン名。一部の Wasm モジュールでは、実行する Wasm プラグインを選択するためにこの値が必要になる場合があります。	いいえ
spec.pluginConfig	Struct	プラグインに渡される設定。	いいえ
spec.pluginConfig.verificationKey	string	署名された OCI イメージまたは Wasm モジュールの署名を検証するために使用される公開鍵。PEM 形式で指定する必要があります。	いいえ

WorkloadSelector オブジェクトは、フィルターをプロキシに適用できるかどうかを判別するために使用される条件を指定します。一致の条件には、プロキシに関連付けられたメタデータ、Pod/VM に添付されたラベルなどのワークロードインスタンス情報、またはプロキシが最初のハンドシェイク中に Istio に提供するその他の情報が含まれます。複数の条件が指定されている場合、ワークロードインスタンスを選択するには、すべての条件が一致する必要があります。現在、ラベルベースの選択メカニズムのみがサポートされています。

表1.15 WorkloadSelector

フィールド	型	説明	必須
matchLabels	map<string, string>	ポリシーを適用する必要がある Pod/VM の特定のセットを示す1つ以上のラベル。ラベル検索の範囲は、リソースが存在する設定 namespace に限定されます。	はい

PullPolicy オブジェクトは、OCI イメージをフェッチするときに適用されるプル動作を指定します。

表1.16 PullPolicy

値	説明
<empty>	デフォルト値は IfNotPresent です。ただし、OCI イメージのタグが latest の場合、デフォルト値は Always です。
IfNotPresent	イメージの既存のバージョンが以前にプルされている場合は、それが使用されます。イメージのバージョンがローカルに存在しない場合は、最新バージョンをプルします。
Always	このプラグインを適用するときは、常に最新バージョンのイメージをプルします。

Struct は、動的に型付けされた値にマップされるフィールドで構成される構造化データ値を表します。一部の言語では、Struct はネイティブ表現でサポートされている場合があります。たとえば、JavaScript のようなスクリプト言語では、構造体はオブジェクトとして表されます。

表1.17 Struct

フィールド	型	説明
fields	map<string, Value>	動的に型付けされた値のマップ。

PluginPhase は、プラグインが注入されるフィルターチェーンのフェーズを指定します。

表1.18 PluginPhase

フィールド	説明
-------	----

フィールド	説明
<empty>	コントロールプレーンは、プラグインを挿入する場所を決定します。これは通常、フィルターチェーンの最後かつ、ルーターの直前にあります。プラグインが他のプラグインから独立している場合は、PluginPhase を指定しないでください。
AUTHN	Istio 認証フィルターの前にプラグインを挿入します。
AUTHZ	Istio 認証フィルターの前と Istio 認証フィルターの後にプラグインを挿入します。
STATS	Istio 統計フィルターの前と Istio 認証フィルターの後にプラグインを挿入します。

1.19.3.1. WasmPlugin リソースのデプロイ

WasmPlugin リソースを使用して、Red Hat OpenShift Service Mesh エクステンションを有効にできます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前となります。次の例では、OpenID Connect フローを実行してユーザーを認証する **openid-connect** フィルターを作成します。

手順

1. 以下のリソース例を作成します。

plugin.yaml の例

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: openid-connect
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  url: oci://private-registry:5000/openid-connect/openid:latest
  imagePullPolicy: IfNotPresent
  imagePullSecret: private-registry-pull-secret
  phase: AUTHN
  pluginConfig:
    openid_server: authn
    openid_realm: ingress
```

2. 次のコマンドを使用して **plugin.yaml** ファイルを適用します。

```
$ oc apply -f plugin.yaml
```

1.19.4. ServiceMeshExtension コンテナ形式

コンテナイメージを有効な拡張イメージにするために、WebAssembly モジュールのバイトコードを含む **.wasm** ファイルとコンテナファイルシステムのルートに **manifest.yaml** ファイルが必要です。



注記

新しい WebAssembly エクステンションを作成するときは、**WasmPlugin** API を使用してください。**ServiceMeshExtension** API は Red Hat OpenShift Service Mesh バージョン 2.2 で非推奨化され、Red Hat OpenShift Service Mesh バージョン 2.3 で廃止されました。

manifest.yaml

```
schemaVersion: 1

name: <your-extension>
description: <description>
version: 1.0.0
phase: PreAuthZ
priority: 100
module: extension.wasm
```

表1.19 manifest.yaml フィールドの参照情報

フィールド	説明	必須
schemaVersion	マニフェストスキーマのバージョン管理に使用されます。現時点で利用できる値は 1 のみです。	これは必須フィールドです。
name	拡張の名前です。	このフィールドは単なるメタデータであり、現時点では使用されていません。
description	拡張の説明。	このフィールドは単なるメタデータであり、現時点では使用されていません。
version	拡張のバージョンです。	このフィールドは単なるメタデータであり、現時点では使用されていません。
phase	拡張のデフォルトの実行フェーズです。	これは必須フィールドです。
priority	拡張のデフォルトの優先度です。	これは必須フィールドです。
module	コンテナファイルシステムのルートから WebAssembly モジュールへの相対パスです。	これは必須フィールドです。

1.19.5. ServiceMeshExtension リファレンス

ServiceMeshExtension API には、WebAssembly フィルターを介して Istio プロキシによって提供される機能を拡張するメカニズムがあります。WebAssembly エクステンションの作成には 2 つの部分があります。

1. proxy-wasm API を公開する SDK を使用してエクステンションを記述し、それを WebAssembly モジュールにコンパイルします。
2. コンテナにパッケージ化します。



注記

新しい WebAssembly エクステンションを作成するときは、**WasmPlugin** API を使用してください。Red Hat OpenShift Service Mesh バージョン 2.2 で非推奨化された **ServiceMeshExtension** API は、Red Hat OpenShift Service Mesh バージョン 2.3 で廃止されました。

表1.20 ServiceMeshExtension フィールドの参照情報

フィールド	説明
metadata.namespace	ServiceMeshExtension ソースの metadata.namespace フィールドには特殊なセマンティクスが含まれます。これが Control Plane Namespace と等しい場合、拡張はその workloadSelector の値に一致する Service Mesh のすべてのワークロードに適用されます。これは、その他の Mesh namespace にデプロイされる場合に、同じ namespace のワークロードにのみ適用されます。
spec.workloadSelector	spec.workloadSelector フィールドには、 Istio Gateway リソース の spec.selector フィールドと同じセマンティクスが含まれます。これは Pod ラベルに基づいてワークロードに一致します。 workloadSelector の値が指定されていない場合、拡張は namespace のすべてのワークロードに適用されます。
spec.config	これは、拡張に転送される構造化フィールドで、セマンティクスはデプロイしている拡張に依存します。
spec.image	拡張を保持するイメージを参照するコンテナイメージ URI です。

フィールド	説明
spec.phase	このフェーズでは、認証、認可、メトリクスの生成などの既存の Istio 機能に関連して、拡張が挿入されるフィルターチェーン内の場所を決定します。有効な値は、PreAuthN、PostAuthN、PreAuthZ、PostAuthZ、PreStats、PostStats です。このフィールドは、拡張の manifest.yaml ファイルで設定される値にデフォルト設定されますが、ユーザーが上書きできます。
spec.priority	同じ spec.phase の値を持つ複数のエクステンションが同じワークロードインスタンスに適用される場合、 spec.priority は実行の順序を決定します。優先度が高いエクステンションが最初の実行されます。これにより、相互に依存する拡張が使用可能になります。このフィールドは、拡張の manifest.yaml ファイルで設定される値にデフォルト設定されますが、ユーザーが上書きできます。

1.19.5.1. ServiceMeshExtension リソースのデプロイ

Red Hat OpenShift Service Mesh エクステンションは **ServiceMeshExtension** リソースを使用して有効にできます。この例では、**istio-system** が Service Mesh コントロールプレーンプロジェクトの名前となります。



注記

新しい WebAssembly エクステンションを作成するときは、**WasmPlugin** API を使用してください。**ServiceMeshExtension** API は Red Hat OpenShift Service Mesh バージョン 2.2 で非推奨化され、Red Hat OpenShift Service Mesh バージョン 2.3 で廃止されました。

Rust SDK を使用してビルドされる完全なサンプルは、[header-append-filter](#) を参照してください。これは、拡張の **config** フィールドから取られた名前および値で HTTP 応答に1つ以上のヘッダーを追加する単純なフィルターです。以下のスニペットの設定例を参照してください。

手順

1. 以下のリソース例を作成します。

ServiceMeshExtension リソースエクステンションの例

```
apiVersion: maistra.io/v1
kind: ServiceMeshExtension
metadata:
  name: header-append
  namespace: istio-system
spec:
  workloadSelector:
```

```

labels:
  app: httpbin
config:
  first-header: some-value
  another-header: another-value
image: quay.io/maistra-dev/header-append-filter:2.1
phase: PostAuthZ
priority: 100

```

2. 以下のコマンドを使用して **extension.yaml** ファイルを適用します。

```
$ oc apply -f <extension>.yaml
```

1.19.6. ServiceMeshExtension リソースから WasmPlugin リソースへの移行

Red Hat OpenShift Service Mesh バージョン 2.2 で非推奨化された **ServiceMeshExtension** API は、Red Hat OpenShift Service Mesh バージョン 2.3 で廃止されました。**ServiceMeshExtension** API を使用している場合、WebAssembly エクステンションを引き続き使用するには **WasmPlugin** API に移行する必要があります。

API は非常に似ています。移行の手順は、以下の 2 つのステップで構成されます。

1. プラグインファイルの名前を変更し、モジュールパッケージを更新する。
2. 更新されたコンテナイメージを参照する **WasmPlugin** リソースを作成する。

1.19.6.1. API の変更

新しい **WasmPlugin** API は **ServiceMeshExtension** に似ていますが、いくつかの違いがあります (特にフィールド名)。

表1.21 **ServiceMeshExtensions** と **WasmPlugin** の間のフィールドの変更

ServiceMeshExtension	WasmPlugin
spec.config	spec.pluginConfig
spec.workloadSelector	spec.selector
spec.image	spec.url
spec.phase 有効な値: PreAuthN、PostAuthN、PreAuthZ、PostAuthZ、PreStats、PostStats	spec.phase 有効な値: <empty>、AUTHN、AUTHZ、STATS

以下は、**ServiceMeshExtension** リソースを **WasmPlugin** リソースに変換する方法の例になります。

ServiceMeshExtension リソース

```

apiVersion: maistra.io/v1
kind: ServiceMeshExtension
metadata:
  name: header-append

```

```

namespace: istio-system
spec:
  workloadSelector:
    labels:
      app: httpbin
  config:
    first-header: some-value
    another-header: another-value
  image: quay.io/maistra-dev/header-append-filter:2.2
  phase: PostAuthZ
  priority: 100

```

上記の ServiceMeshExtension と等価な新しい WasmPlugin リソース

```

apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: header-append
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: httpbin
  url: oci://quay.io/maistra-dev/header-append-filter:2.2
  phase: STATS
  pluginConfig:
    first-header: some-value
    another-header: another-value

```

1.19.6.2. コンテナイメージの形式の変更

新しい **WasmPlugin** コンテナイメージの形式は **ServiceMeshExtensions** と似ていますが、以下の様な違いがあります。

- **ServiceMeshExtension** コンテナ形式には、コンテナファイルシステムのルートディレクトリーに **manifest.yaml** という名前のメタデータファイルが必要でした。**WasmPlugin** コンテナ形式には **manifest.yaml** ファイルは必要ありません。
- 任意のファイル名を付けることができた **.wasm** ファイル (実際のプラグイン) には **plugin.wasm** という名前を付け、コンテナファイルシステムのルートディレクトリーに配置する必要があります。

1.19.6.3. WasmPlugin リソースへの移行

WebAssembly 拡張を **ServiceMeshExtension** API から **WasmPlugin** API にアップグレードするには、プラグインファイルの名前を変更します。

前提条件

- **ServiceMeshControlPlane** がバージョン 2.2 以降にアップグレードされる。

手順

1. コンテナイメージを更新します。プラグインがすでにコンテナ内の `/plugin.wasm` にある場合は、次のステップに進みます。そうでない場合は、以下を行います。
 - a. プラグインファイルの名前が **plugin.wasm** であることを確認します。拡張ファイルには **plugin.wasm** という名前を付ける必要があります。
 - b. プラグインファイルがルート (/) ディレクトリーにあることを確認します。拡張ファイルをコンテナファイルシステムのルートに配置する必要があります。
 - c. コンテナイメージを再ビルドして、これをコンテナレジストリーにプッシュします。
2. **ServiceMeshExtension** リソースを削除し、ビルドした新しいコンテナイメージを参照する **WasmPlugin** リソースを作成します。

1.20. 3SCALE WEBASSEMBLY モジュールの使用



注記

threescale-wasm-auth モジュールは、3scale API Management 2.11 以降と Red Hat OpenShift Service Mesh 2.1.0 以降のインテグレーションで実行されます。

threescale-wasm-auth モジュールは、アプリケーションバイナリーインターフェイス (ABI) と呼ばれるインターフェイスセットを使用する **WebAssembly** モジュールです。これは、ABI を実装するソフトウェアの一部を駆動する **Proxy-WASM** 仕様によって定義され、3scale に対する HTTP リクエストを承認できます。

ABI 仕様として、Proxy-WASM は、**host** という名前のソフトウェアと、**module**、**program**、または **extension** という名前のソフトウェアの間の相互作用を定義します。ホストは、モジュールが使用するサービスセットを公開してタスクを実行し、この場合はプロキシリクエストを処理します。

ホスト環境は、ソフトウェアの一部 (この場合は HTTP プロキシ) と対話する WebAssembly 仮想マシンで構成されています。

モジュール自体は、仮想マシンで実行する手順と Proxy-WASM によって指定された ABI を除き、外部環境とは独立して実行されます。これは、ソフトウェアを参照する拡張を提供するのに安全な方法です。拡張は、仮想マシンおよびホストと明確に定義された方法でのみ対話できます。対話により、コンピューティングモデルと、プロキシが持つ外部への接続が提供されます。

1.20.1. 互換性

threescale-wasm-auth モジュールは、**Proxy-WASM** ABI 仕様のすべての実装と完全に互換性を持つように設計されています。ただし、この時点で、連携することが完全にテストされているのは **Envoy** リバースプロキシだけです。

1.20.2. スタンドアロンモジュールとしての使用

その自己完結型設計により、このモジュールが Service Mesh と 3scale Istio アダプターのデプロイメントとは独立して Proxy-WASM プロキシと連携するように設定できます。

1.20.3. threescale-wasm-auth モジュールの設定

Red Hat OpenShift Service on AWS のクラスター管理者は、**threescale-wasm-auth** モジュールを設定し、アプリケーションバイナリーインターフェイス (ABI) を使用して 3scale API Management への HTTP 要求を承認できます。ABI は、ホストとモジュール間の対話を定義し、ホストサービスを公開

し、モジュールを使用してプロキシ要求を処理できます。

1.20.3.1. WasmPlugin API エクステンション

Service Mesh は、**WasmPlugin** と呼ばれるサイドカープロキシに Proxy-WASM エクステンションを指定して適用するためのカスタムリソース定義を提供します。Service Mesh は、3scale での HTTP API 管理を必要とするワークロードのセットにこのカスタムリソースを適用します。



注記

WebAssembly 拡張の設定は、現在手動プロセスです。3scale システムからサービスの設定を取得するサポートは、今後のリリースでご利用いただけます。

前提条件

- このモジュールを適用する Service Mesh デプロイメントで Kubernetes ワークロードおよび namespace を特定します。
- 3scale テナントアカウントが必要です。適合するサービスならびに該当するアプリケーションおよびメトリクスが定義された [SaaS](#) または [オンプレミス型 3scale 2.11](#) を参照してください。
 - 以下の例は、**threescale-wasm-auth** モジュールのカスタムリソースの YAML 形式です。この例では、アップストリームの Maistra バージョンの Service Mesh **WasmPlugin** API を参照します。モジュールが適用されるアプリケーションのセットを特定する **selector** とともに **threescale-wasm-auth** モジュールがデプロイされる namespace を宣言する必要があります。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
  namespace: <info> 1
spec:
  selector: 2
  labels:
    app: <product_page>
  pluginConfig: <yaml_configuration>
  url: oci://registry.redhat.io/3scale-amp2/3scale-auth-wasm-rhel8:0.0.3
  phase: AUTHZ
  priority: 100
```

1 namespace

2 selector

- **spec.pluginConfig** フィールドはモジュール設定に依存し、直前の例では入力されません。この例では、**<yaml_configuration>** プレースホルダーの値を使用します。このカスタムリソースの例の形式を使用できます。
 - **spec.pluginConfig** フィールドはアプリケーションによって異なります。その他のフィールドはすべて、このカスタムリソースの複数のインスタンス間で永続します。以下に例を示します。
 - **url**: 新しいバージョンのモジュールがデプロイされる場合にのみ変更されます。

- **phase:** このモジュールは、OpenID Connect (OIDC) トークンの検証など、プロキシがローカルの承認を行った後に呼び出す必要があるため、同じままです。
- **spec.pluginConfig** と残りのカスタムリソースにモジュール設定を追加したら、**oc apply** コマンドでこれを適用します。

```
$ oc apply -f threescale-wasm-auth-info.yaml
```

関連情報

- [ServiceMeshExtension](#) リソースから [WasmPlugin](#) リソースへの移行
- [Custom Resources](#)

1.20.4. 3scale 外部 ServiceEntry オブジェクトの適用

threescale-wasm-auth モジュールに 3scale に対するクエストを承認させるには、モジュールは 3scale サービスにアクセスする必要があります。Red Hat OpenShift Service Mesh 内でこれを行うには、外部の **ServiceEntry** オブジェクトと対応する **DestinationRule** オブジェクトを TLS 設定に適用して、HTTPS プロトコルを使用します。

カスタムリソース (CR) は、Service Management API および Account Management API のバックエンドおよびシステムコンポーネントのために、Service Mesh 内から 3scale Hosted (SaaS) への安全なアクセスのためのサービスエントリーと宛先ルールを設定します。Service Management API は、各リクエストの承認ステータスのクエリーを受信します。Account Management API は、サービスの API 管理設定を提供します。

手順

1. 以下の外部 **ServiceEntry** CR および関連する 3scale Hosted バックエンド用の **DestinationRule** CR をクラスターに適用します。
 - a. **ServiceEntry** CR を **service-entry-threescale-saas-backend.yaml** というファイルに追加します。

ServiceEntry CR

```
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: service-entry-threescale-saas-backend
spec:
  hosts:
  - su1.3scale.net
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
```

- b. **DestinationRule** CR を **destination-rule-threescale-saas-backend.yaml** というファイルに追加します。

DestinationRule CR

```

apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: destination-rule-threescale-saas-backend
spec:
  host: su1.3scale.net
  trafficPolicy:
    tls:
      mode: SIMPLE
      sni: su1.3scale.net

```

- c. 以下のコマンドを実行して、3scale Hosted バックエンドの外部 **ServiceEntry** CR をクラスターに適用して保存します。

```
$ oc apply -f service-entry-threescale-saas-backend.yml
```

- d. 以下のコマンドを実行して、3scale Hosted バックエンドの外部 **DestinationRule** CR をクラスターに適用して保存します。

```
$ oc apply -f destination-rule-threescale-saas-backend.yml
```

2. 以下の外部 **ServiceEntry** CR および関連する 3scale Hosted システム 用の **DestinationRule** CR をクラスターに適用します。

- a. **ServiceEntry** CR を **service-entry-threescale-saas-system.yml** というファイルに追加します。

ServiceEntry CR

```

apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: service-entry-threescale-saas-system
spec:
  hosts:
    - multitenant.3scale.net
  ports:
    - number: 443
      name: https
      protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS

```

- b. **DestinationRule** CR を **destination-rule-threescale-saas-system.yml** というファイルに追加します。

DestinationRule CR

```

apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: destination-rule-threescale-saas-system
spec:
  host: multitenant.3scale.net

```



```
trafficPolicy:
  tls:
    mode: SIMPLE
    sni: multitenant.3scale.net
```

- c. 以下のコマンドを実行して、3scale Hosted システムの外部 **ServiceEntry** CR をクラスターに適用して保存します。

```
$ oc apply -f service-entry-threescale-saas-system.yml
```

- d. 以下のコマンドを実行して、3scale Hosted システムの外部 **DestinationRule** CR をクラスターに適用して保存します。

```
$ oc apply -f <destination-rule-threescale-saas-system.yml>
```

または、メッシュ内の 3scale サービスをデプロイできます。メッシュ内 3scale サービスをデプロイするには、3scale をデプロイしてデプロイにリンクすることにより、CR 内のサービスの場所を変更します。

関連情報

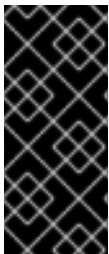
- [サービスエントリーと宛先ルールのドキュメント](#)

1.20.5. 3scale WebAssembly モジュール設定

WasmPlugin カスタムリソース仕様は、**Proxy-WASM** モジュールが読み取る設定を提供します。

仕様はホストに組み込まれ、**Proxy-WASM** モジュールによって読み取られます。通常、設定は、解析するモジュールの JSON ファイル形式ですが、**WasmPlugin** リソースは仕様値を YAML として解釈し、モジュールで使用するために JSON に変換できます。

スタンドアロンモードで **Proxy-WASM** モジュールを使用する場合は、JSON 形式を使用して設定を作成する必要があります。JSON 形式を使用する場合は、**host** 設定ファイル内の必要な場所で、エスケープと引用を使用できます (例:**Envoy**)。 **WasmPlugin** リソースで WebAssembly モジュールを使用する場合、設定は YAML 形式になります。この場合は、無効な設定により、JSON 表現に基づいて診断がモジュールによって強制的にサイドカーコンテナのログイングストリームに表示されます。



重要

EnvoyFilter カスタムリソースは一部の 3scale Istio アダプターまたは Service Mesh リリースで使用できますが、このカスタムリソースはサポートされる API ではありません。**EnvoyFilter** カスタムリソースの使用は推奨されていません。**EnvoyFilter** カスタムリソースの代わりに **WasmPlugin** API を使用します。**EnvoyFilter** カスタムリソースを使用する必要がある場合は、仕様を JSON 形式で指定する必要があります。

1.20.5.1. 3scale WebAssembly モジュールの設定

3scale の WebAssembly モジュール設定のアーキテクチャーは、3scale アカウントおよび承認サービスや処理するサービスのリストによって異なります。

前提条件

前提条件は、すべてのケースで最小の必須フィールドのセットです。

- 3scale アカウントおよび承認サービス:**backend-listener** URL。
- 処理するサービスリスト: サービス ID と少なくとも1つの認証情報の検索方法、およびその検索場所。
- **userkey**、**appid**、**appkey**、および OpenID Connect (OIDC) パターンを処理する例があります。
- WebAssembly モジュールは、静的設定で指定した設定を使用します。たとえば、モジュールにマッピングルール設定を追加する場合は、3scale 管理ポータルにこのようなマッピングルールが設定されていない場合でも、常に適用されます。残りの **WasmPlugin** リソースは **spec.pluginConfig** YAML エントリーに存在します。

1.20.5.2. 3scale WebAssembly モジュール api オブジェクト

3scale WebAssembly モジュールからの **api** 最上位文字列は、モジュールが使用する設定のバージョンを定義します。



注記

存在しないバージョンまたはサポート対象外のバージョンの **api** オブジェクトの場合、レンダリングされた 3scale WebAssembly モジュールは操作不能になります。

api 最上位文字列の例

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
  namespace: <info>
spec:
  pluginConfig:
    api: v1
# ...
```

api エントリーは、設定の残りの値を定義します。許可される値は **v1** のみです。現在の設定との互換性を壊す、または **v1** を使用するモジュールが処理できないロジックを必要とする新しい設定には、異なる値が必要になります。

1.20.5.3. 3scale WebAssembly モジュール system オブジェクト

system 最上位オブジェクトは、特定のアカウントの 3scale Account Management API にアクセスする方法を指定します。**upstream** フィールドは、オブジェクトの最も重要な部分です。**system** オブジェクトはオプションですが、3scale の**system**コンポーネントへの接続を提供しない場合のオプションである 3scale WebAssembly モジュールに完全な静的設定を提供する場合を除き、推奨されます。

system オブジェクトに加えて静的設定オブジェクトを指定する場合は、静的な設定オブジェクトが優先されます。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
```

```

pluginConfig:
  system:
    name: <saas_porta>
    upstream: <object>
    token: <my_account_token>
    ttl: 300
# ...

```

表1.22 **system** オブジェクトフィールド

名前	説明	必須
name	3scale サービスの識別子 (現在、参照されていません)。	任意
upstream	問い合わせるネットワークホストの詳細。 upstream は、system として知られる 3scale Account Management API ホストを参照します。	はい
token	読み取り権限を持つ 3scale の個人アクセストークン。	はい
ttl	新規の変更を取得する前に、このホストから取得した設定を有効なものとする最小時間 (秒数)。デフォルトは 600 秒 (10 分) です。 注記: 最大の期間はありますが、モジュールは通常、この TTL が経過した後に妥当な時間内に設定を取得します。	任意

1.20.5.4. 3scale WebAssembly モジュール upstream オブジェクト

upstream オブジェクトは、プロキシが呼び出しを実行できる外部ホストを説明しています。

```

apiVersion: maistra.io/v1
upstream:
  name: outbound|443||multitenant.3scale.net
  url: "https://myaccount-admin.3scale.net/"
  timeout: 5000
# ...

```

表1.23 **upstream** オブジェクトフィールド

名前	説明	必須
----	----	----

名前	説明	必須
name	name は自由形式の識別子ではありません。これは、プロキシ設定で定義される外部ホストの識別子です。スタンドアロン Envoy 設定の場合は、これは他のプロキシの upstream と呼ばれる クラスター 名にマッピングします。 注記: Service Mesh および 3scale Istio アダプターコントロールプレーンは、複数のフィールドの区切り文字として垂直バー () を使用する形式に従って名前を設定します。この統合の目的上、常に outbound <port> <hostname> の形式を使用します。	はい
url	記述されたサービスにアクセスするための完全な URL。スキームによって暗示されていない限り、TCP ポートが含まれている必要があります。	はい
Timeout	応答にかかる時間がこの設定を超えたこのサービスへの接続がエラーとみなされるためのタイムアウト (ミリ秒単位)。デフォルトは 1000 秒です。	任意

1.20.5.5. 3scale WebAssembly モジュール backend オブジェクト

backend 最上位オブジェクトは、HTTP リクエストの承認および報告のために 3scale Service Management API にアクセスする方法を指定します。このサービスは、3scale のバックエンドコンポーネントによって提供されます。

```

apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    # ...
    backend:
      name: backend
      upstream: <object>
    # ...

```

表1.24 **backend** オブジェクトフィールド

名前	説明	必須
name	3scale バックエンドの識別子 (現在、参照されていません)。	任意
upstream	問い合わせるネットワークホストの詳細。これは、system として知られる 3scale Account Management API ホストを参照する必要があります。	有効。最も重要な必須フィールドです。

1.20.5.6. 3scale WebAssembly モジュール services オブジェクト

services の最上位オブジェクトは、**module** のこの特定のインスタンスで処理されるサービス識別子を指定します。

アカウントには複数のサービスがあるため、どのサービス进行处理するかを指定する必要があります。残りの設定は、サービスの設定方法に関するものです。

services フィールドは必須です。有用とするサービスを少なくとも1つ含める必要がある配列です。

```

apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    # ...
    services:
      - id: "2555417834789"
        token: service_token
        authorities:
          - "*.app"
          - 0.0.0.0
          - "0.0.0.0:8443"
        credentials: <object>
        mapping_rules: <object>
    # ...

```

services 配列の各要素は、3scale サービスを表します。

表1.25 **services** オブジェクトフィールド

名前	説明	必須
ID	この 3scale サービスの識別子 (現在、参照されていません)。	はい

名前	説明	必須
token	<p>この token は、System 内のサービスのプロキシ設定にあるか、以下の curl コマンドを使用して System から取得できます。</p> <pre>curl https://<system_host>/admin /api/services/<service_id>/pr oxy/configs/production/late st.json?access_token= <access_token>" jq '.proxy_config.content.backe nd_authentication_value</pre>	任意
authorities	文字列の配列。それぞれが一致する URL の 認証局 を表します。これらの文字列は、アスタリスク (*)、正符号 (+)、および疑問符 (?) マッチャーに対応する glob パターンを受け入れます。	はい
credentials	検索する認証情報の種類と場所を定義するオブジェクト。	はい
mapping_rules	ヒットするマッピングルールおよび 3scale メソッドを表すオブジェクトの配列。	任意

1.20.5.7. 3scale WebAssembly モジュール credentials オブジェクト

credentials オブジェクトは **service** オブジェクトのコンポーネントです。**credentials** は、検索する認証情報の種類と、このアクションを実行する手順を指定します。

すべてのフィールドはオプションですが、少なくとも1つの **user_key** または **app_id** を指定する必要があります。各認証情報を指定する順番は、モジュールによって事前確立されているために無関係です。各認証情報の1つのインスタンスのみを指定します。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    # ...
    services:
      - credentials:
          user_key: <array_of_lookup_queries>
          app_id: <array_of_lookup_queries>
          app_key: <array_of_lookup_queries>
    # ...
```

表1.26 **credentials** オブジェクトフィールド

名前	説明	必須
user_key	これは、3scale ユーザーキーを定義する検索クエリーの配列です。ユーザーキーは、一般に API キーと呼ばれます。	任意
app_id	これは、3scale のアプリケーション識別子を定義する検索クエリーの配列です。アプリケーションの識別子は、3scale または Red Hat Single Sign-On (RH-SSO) や OpenID Connect (OIDC) などのアイデンティティプロバイダーを使用して提供されます。成功して2つの値に解決するたびに、ここで指定された検索クエリーの解決で、 app_id と app_key が設定されます。	任意
app_key	これは、3scale のアプリケーションキーを定義する検索クエリーの配列です。解決される app_id のないアプリケーションキーは無意味なため、 app_id が指定されている場合のみこのフィールドを指定します。	任意

1.20.5.8. 3scale WebAssembly モジュール検索クエリー

lookup query オブジェクトは、**credentials** オブジェクトのフィールドの一部になります。特定の認証情報フィールドが検出され、処理される方法を指定します。評価されると、解決に成功すると、1つ以上の値が見つかったことを意味します。解決に失敗したことは、値が見つからなかったことを意味します。

lookup queries の配列は、ショートサーキットまたは関係を定義しています。いずれかのクエリーの正常な解決により、残りのクエリーの評価が停止され、値を指定の credential-type に割り当てます。アレイの各クエリーは、互いに独立しています。

lookup query は、1つのフィールド (ソースオブジェクト) で構成されています。これは、複数のソースタイプの1つになります。以下の例を参照してください。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    # ...
    services:
      - credentials:
          user_key:
```

```

- <source_type>: <object>
- <source_type>: <object>
# ...
app_id:
- <source_type>: <object>
# ...
app_key:
- <source_type>: <object>
# ...

```

1.20.5.9. 3scale WebAssembly モジュール source オブジェクト

source オブジェクトは、任意の **credentials** オブジェクトフィールド内のソースの配列の一部として存在します。**source**-type として参照されるオブジェクトフィールド名は、以下のいずれかになります。

- **header**: 検索クエリーは、HTTP リクエストヘッダーを入力として受け取ります。
- **query_string**: **lookup query** は、URL クエリー文字列パラメーターを入力として受け取ります。
- **filter**: **lookup query** は、フィルターメタデータをインプットとして受け取ります。

すべての **source**-type オブジェクトには、少なくとも以下の2つのフィールドがあります。

表1.27 **source**-type オブジェクトフィールド

名前	説明	必須
keys	文字列の配列。それぞれが key で、入力データで検索されたエントリーを参照します。	はい
ops	key エントリーの照合を行う operations の配列。配列は、操作が入力を受け取り、次の操作の出力を生成するパイプラインです。出力に失敗した operation は、 lookup query を失敗として解決します。操作のパイプライン順序によって評価順序が決まります。	任意

filter フィールド名には、データの検索に使用するメタデータのパスを表示するのに必要な **path** エントリーがあります。

key が入力データと一致する場合は、残りの鍵は評価されず、ソース解決アルゴリズムは、指定した **operations (ops)** の実行にジャンプします。**ops** を指定しないと、一致する **key** の結果値 (ある場合) が返されます。

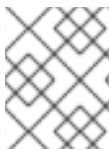
Operations は、最初のフェーズが **key** を検索した後に、入力に対する特定の条件および変換を指定する方法を提供します。プロパティを変換、デコード、および要求する必要があるときに、**operations** を使用しますが、すべてのニーズに対応する成熟した言語は提供されず、**Turing-completeness** はありません。

スタックは **operations** の出力を保存します。評価されると、認証情報が消費する値の数に応じて、スタックの下部に値を割り当てて、**lookup query** は終了します。

1.20.5.10. 3scale WebAssembly モジュール operations オブジェクト

特定の **source type** に属する **ops** 配列の各要素は、値に変換を適用するか、テストを実行する **operation** オブジェクトです。このようなオブジェクトに使用するフィールド名は **operation** 自体の名前で、値は **operation** に対するパラメーターです。これは、フィールドと値のマップ、リスト、または文字列など、構造化オブジェクトになります。

ほとんどの **operations** は、1つ以上の入力を処理し、1つ以上の出力を生成します。入力を使用したり、出力を生成したりする場合、それらは値のスタックで作業します。操作によって消費される各値は、値のスタックからポップアップされ、**source** マッチと共に初期入力されます。出力される値はスタックにプッシュされます。他の **operations** は、特定のプロパティを要求する以外、出力を使用または生成しませんが、値のスタックを検査します。



注記

解決が完了すると、次の手順 (値を **app_id**、**app_key**、または **user_key** に割り当てるなど) でピックアップされる値はスタックの下部の値から取得されます。

operations カテゴリーはいくつかあります。

- **decode**: 別の形式を取得するために、入力値をデコードして変換します。
- **string**: 文字列値を入力として取り、変換を実行し、確認します。
- **stack**: 入力の値のセットを取得し、複数のスタック変換とスタック内の特定の位置の選択を実行します。
- **check**: 影響を及ぼさない方法で、操作セットに関するプロパティを要求します。
- **control**: 評価フローを変更できる操作を実施します。
- **format**: 入力値の形式固有の構造を解析し、その値を検索します。

すべての操作は、name 識別子で文字列として指定されます。

関連情報

- 利用可能な [操作](#)

1.20.5.11. 3scale WebAssembly モジュール mapping_rules オブジェクト

mapping_rules オブジェクトは **service** オブジェクトの一部です。これは、REST パスパターンのセットならびに関連する 3scale メトリクスおよびパターンが一致する時に使用するカウント増分を指定します。

system 最上位オブジェクトに動的設定が提供されていない場合は、値が必要です。**system** 最上位エントリーに加えてオブジェクトが提供されると、**mapping_rules** オブジェクトが最初に評価されます。

mapping_rules は配列オブジェクトです。そのアレイの各要素は **mapping_rule** オブジェクトです。受信したリクエストの評価されたマッチするマッピングルールにより、承認および **APIManager** へのレポート用の 3scale **methods** のセットが提供されます。複数のマッチングルールが同じ **methods** を

参照する場合は、3scale への呼び出し時に **deltas** の合算があります。たとえば、2つのルールが、1と3の **deltas** で Hits メソッドを2回増やすと、3scale にレポートする Hits の単一のメソッドエントリーの **delta** は4になります。

1.20.5.12. 3scale WebAssembly モジュール mapping_rule オブジェクト

mapping_rule オブジェクトは **mapping_rules** オブジェクトの配列の一部です。

mapping_rule オブジェクトフィールドは、以下の情報を指定します。

- 照合する HTTP 要求メソッド。
- パスに一致するパターン。
- 報告する量と共にレポートする 3scale メソッド。フィールドを指定する順序によって評価順序が決まります。

表1.28 **mapping_rule** オブジェクトフィールド

名前	説明	必須
メソッド	HTTP リクエストメソッド (動詞) を表す文字列を指定します。許可される値は、許可される HTTP メソッド名の1つと一致し、大文字と小文字を区別しません。すべてのマッチのすべてのメソッドの特殊な値。	はい
pattern	HTTP リクエストの URI パスコンポーネントに一致するパターン。このパターンは、3scale で説明されている構文に従います。 {this} などの中括弧間の文字のシーケンスを使用するワイルドカード (アスタリスク (*) 文字の使用) が許可されます。	はい
usages	<p>usage オブジェクトのリスト。ルールがマッチすると、deltas を持つすべてのメソッドが、承認およびレポートのために 3scale に送信されるメソッドのリストに追加されます。</p> <p>以下の必須フィールドに usages オブジェクトを埋め込みます。</p> <ul style="list-style-type: none"> ● name: レポートする method のシステム名。 ● delta: その method の増分。 	はい

名前	説明	必須
last	このルールが正常にマッチした場合に、それ以外のマッピングルールの評価を停止する必要があるかどうか。	任意のブール値。デフォルトは false です。

以下の例は、3scale のメソッド間の既存の階層とは独立しています。つまり、3scale 側で実行されたすべての内容はこれには影響しません。たとえば、**Hits** メトリクスは、それらすべての親となる可能性があるため、承認されたリクエストで報告されたすべてのメソッドの合計により 4 ヒットを保管し、3scale **Authrep** API エンドポイントを呼び出します。

以下の例では、すべてのルールに一致する、パス **/products/1/sold** への **GET** リクエストを使用します。

mapping_rules GET リクエストの例

```

apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    # ...
    mapping_rules:
      - method: GET
        pattern: /
        usages:
          - name: hits
            delta: 1
      - method: GET
        pattern: /products/
        usages:
          - name: products
            delta: 1
      - method: ANY
        pattern: /products/{id}/sold
        usages:
          - name: sales
            delta: 1
          - name: products
            delta: 1
    # ...

```

すべての **usages** は、モジュールが使用状況データを使用して 3scale に実施するリクエストに追加されます。

- Hits: 1
- products: 2
- sales: 1

1.20.6. 認証情報ユースケースの 3scale WebAssembly モジュールの例

ほとんどの時間を費やして、設定手順を適用してサービスへのリクエストの認証情報を取得します。

以下は **credentials** の例です。これは、特定のユースケースに合わせて変更できます。

複数のソースオブジェクトと独自の **lookup queries** を指定する場合、これらはすべて組み合わせることができ、いずれか1つが正しく解決されるまで、それらは順番に評価されます。

1.20.6.1. クエリー文字列パラメーターの API キー (user_key)

以下の例では、クエリー文字列パラメーターまたは同じ名前のヘッダーで **user_key** を検索します。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  # ...
  services:
  # ...
    credentials:
      user_key:
        - query_string:
            keys:
              - <user_key>
        - header:
            keys:
              - <user_key>
  # ...
```

1.20.6.2. アプリケーション ID およびキー

以下の例では、クエリーまたはヘッダーの **app_key** および **app_id** 認証情報を検索します。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  # ...
  services:
  # ...
    credentials:
      app_id:
        - query_string:
            keys:
              - <app_id>
        - header:
            keys:
              - <app_id>
      app_key:
        - query_string:
            keys:
              - <app_key>
```

```
- header:
  keys:
    - <app_key>
```

```
# ...
```

1.20.6.3. 認証ヘッダー

リクエストには、**authorization** ヘッダーに **app_id** および **app_key** が含まれます。最後に出力される値が1つまたは2つある場合は、**app_key** を割り当てることができます。

ここでの解決は、最後に出力された1つまたは2つの出力がある場合は **app_key** を割り当てます。

authorization ヘッダーは承認の種類で値を指定し、その値は **Base64** としてエンコードされます。つまり、値を空白文字で分割し、2番目の出力を取得して、コロン (:) をセパレーターとして使用して再度分割できます。たとえば、**app_id:app_key** という形式を使用する場合、ヘッダーは以下の **credential** の例のようになります。

```
aladdin:opensesame: Authorization: Basic YWxhZGRpbjpvGVuc2VzYW1l
```

以下の例のように、小文字のヘッダーフィールド名を使用する必要があります。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  # ...
  services:
  # ...
    credentials:
      app_id:
        - header:
            keys:
              - authorization
        ops:
          - split:
              separator: " "
              max: 2
          - length:
              min: 2
          - drop:
              head: 1
          - base64_urlsafe
          - split:
              max: 2
      app_key:
        - header:
            keys:
              - app_key
  # ...
```

前述のユースケースの例は、**authorization** のヘッダーを確認します。

1. これは文字列の値を取り、スペースで分割し、**credential**-type および **credential** 自体の少なくとも2つの値を生成することを確認してから、**credential**-type をドロップします。

2. 次に、必要なデータが含まれる 2 番目の値をデコードし、最初の **app_id** の後にもしあれば **app_key** が含まれる操作スタックとなるように、コロン (:) 文字を使用して分割します。
 - a. **app_key** が認証ヘッダーに存在しない場合は、特定のソースがチェックされます (この場合は、キー **app_key** のヘッダーなど)。
3. **credentials** に追加の条件を追加するには、**Basic** 認証を許可します。ここで、**app_id** は **aladdin** もしくは **admin**、または長さが 8 文字以上の任意の **app_id** になります。
4. **app_key** には値が含まれ、以下の例のように最小で 64 文字を指定する必要があります。

```

apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  # ...
  services:
  # ...
  credentials:
    app_id:
      - header:
          keys:
            - authorization
          ops:
            - split:
                separator: " "
                max: 2
            - length:
                min: 2
            - reverse
            - glob:
                - Basic
            - drop:
                tail: 1
            - base64_urlsafe
            - split:
                max: 2
            - test:
                if:
                  length:
                    min: 2
                then:
                  - strlen:
                      max: 63
                  - or:
                      - strlen:
                          min: 1
                      - drop:
                          tail: 1
            - assert:
            - and:
            - reverse
            - or:
            - strlen:
                min: 8

```

```

- glob:
  - aladdin
  - admin
# ...

```

5. **authorization** ヘッダーの値を選択したら、タイプが上部に配置されるようにスタックを逆にして **Basic credential**-type を取得します。
6. glob マッチを実行します。検証し、認証情報がデコードされ、分割されると、スタックの下部に **app_id** を取得し、上部に **app_key** を取得する可能性があります。
7. **test:** を実行します。スタックに 2 つの値がある場合は、**app_key** が取得されたことになりません。
 - a. **app_id** および **app_key** を含め、文字列の長さが 1 から 63 文字になるようにします。キーの長さがゼロの場合は破棄し、キーが存在しないものとして続行します。**app_id** のみがあり、**app_key** がない場合、不明なブランチは、テストに成功し、評価が続行されます。

最後の操作は **assert** で、スタックに副作用がないことを示します。その後、スタックを変更できません。

1. **app_id** が最上部になるように、スタックを逆にします。
 - a. **app_key** が存在するかどうかで、スタックを逆にすると、**app_id** が上部になります。
2. **and** を使用して、テスト間でスタックの内容を保持します。次に、以下のいずれかの方法を使用します。
 - **app_id** に 8 文字以上の文字列が設定されていることを確認してください。
 - **app_id** が **aladdin** または **admin** と一致していることを確認します。

1.20.6.4. OpenID Connect (OIDC) のユースケース

Service Mesh および 3scale Istio アダプターの場合は、以下の例のように **RequestAuthentication** をデプロイし、独自のワークロードデータおよび **jwtRules** を入力する必要があります。

```

apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: info
spec:
  selector:
    matchLabels:
      app: productpage
  jwtRules:
    - issuer: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak
      jwksUri: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak/protocol/openid-connect/certs

```

RequestAuthentication を適用するとき、**JWT** トークンを検証するために **ネイティブプラグイン** で **Envoy** を設定します。プロキシは、モジュールを実行する前にすべてを検証します。したがって、失敗したリクエストが 3scale WebAssembly モジュールに実行されません。

JWT トークンが検証されると、プロキシーはそのコンテンツを内部メタデータオブジェクトに格納します。エントリーのキーは、プラグインの特定の設定に依存します。このユースケースでは、不明なキー名が含まれる単一のエントリーを持つ構造化オブジェクトを検索できます。

OIDC の 3scale **app_id** は、OAuth **client_id** と一致します。これは **JWT** トークンの **azp** フィールドまたは **aud** フィールドにあります。

Envoy のネイティブ **JWT** 認証フィルターから **app_id** フィールドを取得するには、以下の例を参照してください。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  # ...
  services:
  # ...
    credentials:
      app_id:
        - filter:
            path:
              - envoy.filters.http.jwt_authn
              - "0"
            keys:
              - azp
              - aud
            ops:
              - take:
                  head: 1
        # ...
```

この例では、モジュールに対し、**filter** ソースタイプを使用して **Envoy** 固有の **JWT** 認証ネイティブプラグインからオブジェクトのフィルターメタデータを検索するよう指示します。このプラグインには、1つのエントリーと事前に設定された名前を持つ構造化オブジェクトの一部として **JWT** トークンが含まれます。**0** を使用して、単一のエントリーのみにアクセスするように指定します。

結果の値は、以下の2つのフィールドを解決する構造です。

- **azp: app_id** が見つけれられる値。
- **aud:** この情報も見つけれられる値。

この操作により、割り当て用に1つの値のみが保持されます。

1.20.6.5. ヘッダーからの JWT トークンの取得

一部のセットアップには、**JWT** トークンの検証プロセスがあり、検証されたトークンが JSON 形式のヘッダーを介してこのモジュールに到達する場合があります。

app_id を取得するには、以下の例を参照してください。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
```

```
spec:
# ...
  services:
# ...
    credentials:
      app_id:
        - header:
            keys:
              - x-jwt-payload
            ops:
              - base64_urlsafe
              - json:
                  - keys:
                      - azp
                      - aud
                  - take:
                      head: 1
# ,,,
```

1.20.7. 3scale WebAssembly モジュールの機能する最低限の設定

以下は、3scale WebAssembly モジュールの機能する最低限の設定の例です。これをコピーアンドペーストし、これを独自の設定で機能するように編集できます。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  url: oci://registry.redhat.io/3scale-amp2/3scale-auth-wasm-rhel8:0.0.3
  imagePullSecret: <optional_pull_secret_resource>
  phase: AUTHZ
  priority: 100
  selector:
    labels:
      app: <product_page>
  pluginConfig:
    api: v1
    system:
      name: <system_name>
      upstream:
        name: outbound|443|multitenant.3scale.net
        url: https://istiodevel-admin.3scale.net/
        timeout: 5000
        token: <token>
    backend:
      name: <backend_name>
      upstream:
        name: outbound|443|su1.3scale.net
        url: https://su1.3scale.net/
        timeout: 5000
    extensions:
      - no_body
  services:
    - id: '2555417834780'
```



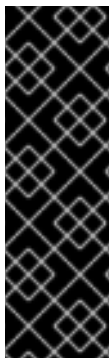
```

authorities:
- ""
credentials:
  user_key:
    - query_string:
        keys:
          - <user_key>
    - header:
        keys:
          - <user_key>
  app_id:
    - query_string:
        keys:
          - <app_id>
    - header:
        keys:
          - <app_id>
  app_key:
    - query_string:
        keys:
          - <app_key>
    - header:
        keys:
          - <app_key>

```

1.21. 3SCALE ISTIO アダプターの使用

3scale Istio アダプターはオプションのアダプターであり、これを使用すると、Red Hat OpenShift Service Mesh 内で実行中のサービスにラベルを付け、そのサービスを 3scale API Management ソリューションと統合できます。これは Red Hat OpenShift Service Mesh には必要ありません。



重要

3scale Istio アダプターは、Red Hat OpenShift Service Mesh バージョン 2.0 以前でのみ使用できます。Mixer コンポーネントはリリース 2.0 で非推奨となり、リリース 2.1 で削除されました。Red Hat OpenShift Service Mesh バージョン 2.1.0 以降では、[3scale WebAssembly モジュール](#)を使用する必要があります。

3scale Istio アダプターで 3scale バックエンドキャッシュを有効にする必要がある場合は、Mixer ポリシーと Mixer Telemetry も有効にする必要があります。[Red Hat OpenShift Service Mesh コントロールプレーンのデプロイ](#)を参照してください。

1.21.1. 3scale アダプターと Red Hat OpenShift Service Mesh の統合

これらの例を使用して、3scale Istio アダプターを使用してサービスに対する要求を設定できます。

前提条件

- Red Hat OpenShift Service Mesh バージョン 2.x
- 稼働している 3scale アカウント (SaaS または [3scale 2.9 On-Premises](#))
- バックエンドキャッシュを有効にするには 3scale 2.9 以上が必要です。
- Red Hat OpenShift Service Mesh の前提条件

- Mixer ポリシーの適用が有効になっていることを確認します。「Mixer ポリシー適用の更新」セクションでは、現在の Mixer ポリシーの適用ステータスをチェックし、ポリシーの適用を有効にする手順が説明されています。
- Mixer プラグインを使用している場合は、Mixer ポリシーと Telemetry は有効にする必要があります。
 - アップグレード時に、Service Mesh コントロールプレーン (SMCP) を適切に設定する必要があります。



注記

3scale Istio アダプターを設定するために、アダプターパラメーターをカスタムリソースファイルに追加する手順は、Red Hat OpenShift Service Mesh カスタムリソースを参照してください。



注記

kind: handler リソースにとくに注意してください。これを 3scale アカountの認証情報を使用して更新する必要があります。オプションで **service_id** をハンドラーに追加できますが、この設定は 3scale アカountの1つのサービスに対してのみ有効で、後方互換性を確保するためにだけ維持されています。**service_id** をハンドラーに追加し、他のサービスに対して 3scale を有効にする必要がある場合は、別の **service_ids** で追加のハンドラーを作成する必要があります。

以下の手順に従って、3scale アカountごとに単一のハンドラーを使用します。

手順

1. 3scale アカountのハンドラーを作成し、アカountの認証情報を指定します。サービス識別子を省略します。

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

オプションで、**params** セクション内の **backend_url** フィールドを指定して、3scale 設定によって提供される URL を上書きできます。これは、アダプターが 3scale のオンプレミスインスタンスと同じクラスターで実行され、内部クラスター DNS を利用する必要がある場合に役立ちます。

2. 3scale アカountに属するサービスの Deployment リソースを編集するか、パッチを適用します。
 - a. 有効な **service_id** に対応する値を使用して **"service-mesh.3scale.net/service-id"** ラベルを追加します。

- b. 手順1のハンドラーリソースの名前の値を使用して **"service-mesh.3scale.net/credentials"** ラベルを追加します。
3. 他のサービスを追加する場合は、手順2を実行して、3scale アカウントの認証情報とそのサービス識別子にリンクします。
4. 3scale 設定でルールの変更し、ルールを 3scale ハンドラーにディスパッチします。

ルール設定の例

```
apiVersion: "config.istio.io/v1alpha2"
kind: rule
metadata:
  name: threescale
spec:
  match: destination.labels["service-mesh.3scale.net"] == "true"
  actions:
    - handler: threescale.handler
      instances:
        - threescale-authorization.instance
```

1.21.1.1. 3scale カスタムリソースの生成

アダプターには、**handler**、**instance**、および **rule** カスタムリソースの生成を可能にするツールが含まれます。

表1.29 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のヘルプ出力を生成します	いいえ	
--name	この URL の一意の名前、トークンのペア	はい	
-n, --namespace	テンプレートを生成する namespace	いいえ	istio-system
-t, --token	3scale アクセストークン	はい	
-u, --url	3scale 管理ポータル URL	はい	
--backend-url	3scale バックエンド URL。これが設定されている場合は、システム設定から読み込まれる値がオーバーライドされます。	いいえ	
-s, --service	3scale API/サービス ID	いいえ	

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)	いいえ	ハイブリッド
-o, --output	生成されたマニフェストを保存するファイル	いいえ	標準出力
--version	CLI バージョンを出力し、即座に終了する	いいえ	

1.21.1.1.1. URL サンプルからのテンプレートの生成



注記

- [デプロイされたアダプターからのマニフェストの生成](#) で、3scale アダプターコンテナイメージからの **oc exec** を使用して以下のコマンドを実行します。
- **3scale-config-gen** コマンドを使用すると、YAML 構文とインデントエラーを回避するのに役立ちます。
- このアノテーションを使用する場合は **--service** を省略できます。
- このコマンドは、**oc exec** を使用してコンテナイメージ内から起動する必要があります。

手順

- **3scale-config-gen** コマンドを使用して、トークンと URL のペアを1つのハンドラーとして複数のサービスで共有できるようにテンプレートを自動生成します。

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

- 以下の例では、ハンドラーに埋め込まれたサービス ID を使用してテンプレートを生成します。

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

関連情報

- [トークン](#)

1.21.1.2. デプロイされたアダプターからのマニフェストの生成



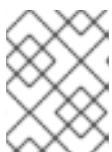
注記

- **NAME** は、3scale で管理するサービスの識別に使用する識別子です。
- **CREDENTIALS_NAME** 参照は、ルール設定の **match** セクションに対応する識別子です。CLI ツールを使用している場合は、**NAME** 識別子に自動設定されます。
- この値は具体的なものでなくても構いませんが、ラベル値はルールの内容と一致させる必要があります。詳細は、[アダプター経由でのサービストラフィックのルーティング](#)を参照してください。

1. このコマンドを実行して、**istio-system** namespace でデプロイされたアダプターからマニフェストを生成します。

```
$ export NS="istio-system" URL="https://replaceme-admin.3scale.net:443" NAME="name"
TOKEN="token"
oc exec -n ${NS} $(oc get po -n ${NS} -o jsonpath='{.items[?
(@.metadata.labels.app=="3scale-istio-adapter")].metadata.name}') \
-it -- ./3scale-config-gen \
--url ${URL} --name ${NAME} --token ${TOKEN} -n ${NS}
```

2. これでターミナルにサンプル出力が生成されます。必要に応じて、これらのサンプルを編集し、**oc create** コマンドを使用してオブジェクトを作成します。
3. 要求がアダプターに到達すると、アダプターはサービスが 3scale の API にどのようにマッピングされるかを認識している必要があります。この情報は、以下のいずれかの方法で提供できます。
 - a. ワークロードにラベルを付ける (推奨)
 - b. ハンドラーを **service_id** としてハードコーディングする
4. 必要なアノテーションでワークロードを更新します。



注記

ハンドラーにまだ組み込まれていない場合は、このサンプルで提供されたサービス ID のみを更新する必要があります。ハンドラーの設定が優先されます。

```
$ export CREDENTIALS_NAME="replace-me"
export SERVICE_ID="replace-me"
export DEPLOYMENT="replace-me"
patch="$(oc get deployment "${DEPLOYMENT}"
patch="$(oc get deployment "${DEPLOYMENT}" --template='{ "spec": { "template": { "metadata":
{ "labels": { { range $k,$v := .spec.template.metadata.labels }} { $k }}: { { $v }}', { end
}} "service-mesh.3scale.net/service-id": "${SERVICE_ID}", "service-
mesh.3scale.net/credentials": "${CREDENTIALS_NAME}" } }' )"
oc patch deployment "${DEPLOYMENT}" --patch "${patch}"
```

1.21.1.3. アダプター経由でのサービストラフィックのルーティング

以下の手順に従って、3scale アダプターを使用してサービスのトラフィックを処理します。

前提条件

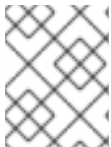
- 3scale 管理者から受け取る認証情報とサービス ID

手順

1. **kind: rule** リソース内で、以前に設定で作成した **destination.labels["service-mesh.3scale.net/credentials"] == "threescale"** ルールと一致させます。
2. 上記のラベルを、ターゲットワークロードのデプロイメントで **PodTemplateSpec** に追加し、サービスを統合します。値 **threescale** は生成されたハンドラーの名前を参照します。このハンドラーは、3scale を呼び出すのに必要なアクセストークンを保存します。
3. **destination.labels["service-mesh.3scale.net/service-id"] == "replace-me"** ラベルをワークロードに追加し、要求時にサービス ID をインスタンス経由でアダプターに渡します。

1.21.2. 3scale での統合設定

以下の手順に従って、3scale の統合設定を行います。



注記

3scale SaaS を使用している場合は、Red Hat OpenShift Service Mesh は Early Access プログラムの一部として有効になっています。

手順

1. **[your_API_name]** → **Integration** の順に移動します。
2. **Settings** をクリックします。
3. **Deployment** で **Istio** オプションを選択します。
 - デフォルトでは **Authentication** の **API Key (user_key)** オプションが選択されます。
4. **Update Product** をクリックして選択内容を保存します。
5. **Configuration** をクリックします。
6. **設定の更新** をクリックします。

1.21.3. キャッシング動作

3scale System API からの応答は、アダプター内でデフォルトでキャッシュされます。**cacheTTLSeconds** 値よりも古いと、エントリ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の自動更新は、**cacheRefreshSeconds** 値に基づいて、期限が切れる前に数秒間試行されます。**cacheTTLSeconds** 値よりも高い値を設定することで、自動更新を無効にできます。

cacheEntriesMax を正の値以外に設定すると、キャッシングを完全に無効にできます。

更新プロセスを使用すると、到達不能になるホストのキャッシュされた値が、期限が切れて最終的に消去される前に再試行されます。

1.21.4. 認証要求

このリリースでは、以下の認証方法をサポートします。

- **標準 API キー**: 単一のランダム文字列またはハッシュが識別子およびシークレットトークンとして機能します。
- **アプリケーション ID とキーのペア**: イミュータブルな識別子とミュータブルなシークレットキー文字列。
- **OpenID 認証方法**: JSON Web トークンから解析されるクライアント ID 文字列。

1.21.4.1. 認証パターンの適用

以下の認証方法の例に従って **instance** カスタムリソースを変更し、認証動作を設定します。認証情報は、以下から受け取ることができます。

- 要求ヘッダー
- 要求パラメーター
- 要求ヘッダーとクエリーパラメーターの両方



注記

ヘッダーの値を指定する場合、この値は小文字である必要があります。たとえば、ヘッダーを **User-Key** として送信する必要がある場合、これは設定で **request.headers["user-key"]** として参照される必要があります。

1.21.4.1.1. API キー認証方法

Service Mesh は、**subject** カスタムリソースパラメーターの **user** オプションで指定されたクエリーパラメーターと要求ヘッダーで API キーを検索します。これは、カスタムリソースファイルで指定される順序で値をチェックします。不要なオプションを省略することで、API キーの検索をクエリーパラメーターまたは要求ヘッダーに制限できます。

この例では、Service Mesh は **user_key** クエリーパラメーターの API キーを検索します。API キーがクエリーパラメーターにない場合、Service Mesh は **user-key** ヘッダーを確認します。

API キー認証方法の例

```
apiVersion: "config.istio.io/v1alpha2"
kind: instance
metadata:
  name: threescale-authorization
  namespace: istio-system
spec:
  template: authorization
  params:
    subject:
      user: request.query_params["user_key"] | request.headers["user-key"] | ""
    action:
      path: request.url_path
      method: request.method | "get"
```

アダプターが異なるクエリーパラメーターまたは要求ヘッダーを検査するようにする場合は、名前を適宜変更します。たとえば、“key”というクエリーパラメーターの API キーを確認するには、**request.query_params["user_key"]** を **request.query_params["key"]** に変更します。

1.21.4.1.2. アプリケーション ID およびアプリケーションキーペアの認証方法

Service Mesh は、**subject** カスタムリソースパラメーターの **properties** オプションで指定されるように、クエリーパラメーターと要求ヘッダーでアプリケーション ID とアプリケーションキーを検索します。アプリケーションキーはオプションです。これは、カスタムリソースファイルで指定される順序で値をチェックします。不要なオプションを含めないことで、認証情報の検索をクエリーパラメーターまたは要求ヘッダーのいずれかに制限できます。

この例では、Service Mesh は最初にクエリーパラメーターのアプリケーション ID とアプリケーションキーを検索し、必要に応じて要求ヘッダーに移動します。

アプリケーション ID およびアプリケーションキーペアの認証方法の例

```
apiVersion: "config.istio.io/v1alpha2"
kind: instance
metadata:
  name: threescale-authorization
  namespace: istio-system
spec:
  template: authorization
  params:
    subject:
      app_id: request.query_params["app_id"] | request.headers["app-id"] | ""
      app_key: request.query_params["app_key"] | request.headers["app-key"] | ""
    action:
      path: request.url_path
      method: request.method | "get"
```

アダプターが異なるクエリーパラメーターまたは要求ヘッダーを検査するようにする場合は、名前を適宜変更します。たとえば、**identification** という名前のクエリーパラメーターのアプリケーション ID を確認するには、**request.query_params["app_id"]** を **request.query_params["identification"]** に変更します。

1.21.4.1.3. OpenID 認証方法

OpenID Connect (OIDC) 認証方法を使用するには、**subject** フィールドで **properties** 値を使用して **client_id** および任意で **app_key** を設定します。

このオブジェクトは、前述の方法を使用して操作できます。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プロバイダー (IdP) で作成されるよう OIDC を 3scale で実行する必要があります。保護するサービスと同じ namespace でサービスの [に要求の認証](#) を作成する必要があります。JWT は要求の **Authorization** ヘッダーに渡されます。

以下に定義されるサンプル **RequestAuthentication** で、**issuer**、**jwksUri**、および **selector** を適宜置き換えます。

OpenID Policy の例

```

apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: info
spec:
  selector:
    matchLabels:
      app: productpage
  jwtRules:
    - issuer: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak
      jwksUri: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak/protocol/openid-connect/certs

```

1.21.4.1.4. 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は、Service Mesh は API キーを使用します。

この例では、Service Mesh がクエリーパラメーターの API キーをチェックし、次に要求ヘッダーを確認します。API キーがない場合は、クエリーパラメーターのアプリケーション ID とキーをチェックし、次に要求ヘッダーを確認します。

ハイブリッド認証方法の例

```

apiVersion: "config.istio.io/v1alpha2"
kind: instance
metadata:
  name: threescale-authorization
spec:
  template: authorization
  params:
    subject:
      user: request.query_params["user_key"] | request.headers["user-key"] |

```

```

properties:
  app_id: request.query_params["app_id"] | request.headers["app-id"] | ""
  app_key: request.query_params["app_key"] | request.headers["app-key"] | ""
  client_id: request.auth.claims["azp"] | ""
action:
  path: request.url_path
  method: request.method | "get"
  service: destination.labels["service-mesh.3scale.net/service-id"] | ""

```

1.21.5. 3scale アダプターメトリクス

アダプターはデフォルトで、**/metrics** エンドポイントのポート **8080** で公開されるさまざまな Prometheus メトリクスを報告します。これらのメトリクスは、アダプターと 3scale の間の相互作用がどのように実行されているかに関する洞察を提供します。サービスには、自動的に検出され、Prometheus によって収集されるようにラベルが付けられます。



注記

3scale Istio Adapter メトリクスには、Service Mesh 1.x の以前のリリース以降、互換性のない変更があります。

Prometheus では、以下のメトリクスが Service Mesh 2.0 の時点で使用できるように、バックエンドキャッシュの1つの追加と共にメトリクスの名前が変更されています。

表1.30 Prometheus メトリクス

メトリクス	型	説明
threescale_latency	ヒストグラム	アダプターと 3scale 間の要求レイテンシーです。
threescale_http_total	カウンター	3scale バックエンドへの要求に関する HTTP ステータスの応答コード。
threescale_system_cache_hits	カウンター	設定キャッシュからフェッチされる 3scale システムへの要求の合計数。
threescale_backend_cache_hits	カウンター	バックエンドキャッシュからフェッチされる 3scale バックエンドへの要求の合計数。

1.21.6. 3scale バックエンドキャッシュ

3scale バックエンドキャッシュは、3scale Service Management API のクライアントの認証およびポートキャッシュを提供します。このキャッシュはアダプターに組み込まれ、管理者がトレードオフを受け入れることが予想される特定の状況での応答の低レイテンシーが可能になります。



注記

3scale バックエンドキャッシュはデフォルトで無効になっています。3scale バックエンドキャッシュ機能では、低レイテンシーとプロセッサおよびメモリーのリソースの高い使用率と引き換えに、速度制限における不正確な状況が生じたり、フラッシュの最後の実行からのヒットを失う可能性があります。

1.21.6.1. バックエンドキャッシュを有効にする利点

バックエンドキャッシュを有効にする利点には以下が含まれます。

- 3scale Istio Adapter が管理するサービスへのアクセス時にレイテンシーが高くなる場合にバックエンドキャッシュを有効にします。
- バックエンドキャッシュを有効にすると、アダプターは 3scale API マネージャーによるリクエストの承認の継続的なチェックを停止し、レイテンシーが短縮されます。
 - これにより、3scale API マネージャーにアクセスして認証を試行する前に、3scale Istio アダプターが保存し、再利用する 3scale 認証のインメモリーキャッシュが作成されます。これにより、認証の許可または拒否にかかる時間が大幅に少なくなります。
- バックエンドキャッシュは、3scale Istio アダプターを実行する Service Mesh とは異なる地理的な場所で 3scale API マネージャーをホストする場合に役立ちます。
 - 通常、これは 3scale のホスト型 (SaaS) プラットフォームに該当しますが、ユーザーが異なるアベイラビリティゾーンの地理的に異なる場所にある別のクラスターで 3scale API マネージャーをホストする場合や、3scale API マネージャーに到達するためのネットワークのオーバーヘッドを考慮する必要がある場合にも使用できます。

1.21.6.2. 低レイテンシーを確保するためのトレードオフ

以下は、低レイテンシーを確保するためのトレードオフです。

- フラッシュが発生するたびに、3scale アダプターの承認状態が更新されます。
 - つまり、アダプターのインスタンスが 2 つ以上あると、フラッシュ期間の間隔に関する不正確さがさらに大きくなることを意味します。
 - 要求が過剰になり、制限を超過し、誤った動作を生じさせ、さらには各要求を処理するアダプターによって処理される要求と処理されない要求とが発生する高い可能性があります。
- データをフラッシュできず、その認証情報を更新できないアダプターキャッシュは、その情報を API マネージャーに報告せずにシャットダウンまたはクラッシュする可能性があります。
- アダプターのキャッシュで、API マネージャーと通信する際のネットワーク接続が原因と予想される問題などにより、要求を許可/拒否する必要があるかどうかを判別できない場合に、fail open または fail closed ポリシーが適用されます。
- キャッシュミスが発生すると、通常はアダプターの起動直後、または接続なしの状態が長く続いた後に、API マネージャーのクエリーを行うためにレイテンシーが増加します。
- アダプターキャッシュでは、キャッシュを有効にしない場合よりも、認証の計算により多くの作業が必要になります。これにより、より多くのプロセッサリソースが必要になります。
- メモリー要件は、キャッシュで管理される制限、アプリケーションおよびサービスの量の組み合わせに比例して増加します。

1.21.6.3. バックエンドキャッシュ設定

以下では、バックエンドキャッシュの設定を説明します。

- 3scale 設定オプションでバックエンドキャッシュを設定するための設定を見つけます。
- 最後の 3 つの設定では、バックエンドキャッシュの有効化を制御します。
 - **PARAM_USE_CACHE_BACKEND**: バックエンドキャッシュを有効にするには true に設定します。
 - **PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS**: キャッシュデータの API マネージャーへのフラッシュの試行間の時間 (秒単位) を設定します。
 - **PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED**: キャッシュされたデータが十分になく、3scale API マネージャーに到達できない場合にサービスへの要求を許可/オープン/または拒否/クローズするかどうかを設定します。

1.21.7. 3scale Istio Adapter APIcast エミュレーション

3scale Istio アダプターは、以下の条件が満たされる場合に APIcast と同様に動作します。

- 要求が定義されるマッピングルールと一致しない場合、返される HTTP コードは 404 Not Found になります。これは、以前は 403 Forbidden でした。
- 要求が制限を超えるために拒否されると、返される HTTP コードは 429 Too Many Requests になります。これは、以前は 403 Forbidden でした。
- CLI でデフォルトのテンプレートを生成する場合、ヘッダーにはハイフンではなくアンダースコアが使用されます (例: **user-key** ではなく **user_key** が使用されます)。

1.21.8. 3scale Istio Adapter の検証

3scale Istio Adapter が予想通りに機能しているかどうかを確認します。アダプターが機能しない場合は、以下の手順に従って問題のトラブルシューティングを行うことができます。

手順

1. **3scale-adapter** Pod が Service Mesh コントロールプレーン namespace で実行されていることを確認します。

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

2. そのバージョンなど、**3scale-adapter** Pod が起動に関する情報を出力したことを確認します。

```
$ oc logs istio-system
```

3. 3scale アダプターの統合で保護されているサービスに対して要求を実行すると、正しい認証情報がかけられているという要求を必ず試し、その要求が失敗することを確認します。3scale Adapter ログをチェックして、追加情報を収集します。

関連情報

- [Pod およびコンテナログの検査](#)

1.21.9. 3scale Istio adapter のトラブルシューティングのチェックリスト

管理者が 3scale Istio adapter をインストールすると、統合が適切に機能しなくなる可能性のあるシナリオが複数あります。以下のリストを使用して、インストールのトラブルシューティングを行います。

- YAML のインデントが間違っている。
- YAML セクションがない。
- YAML の変更をクラスターに適用するのを忘れている。
- **service-mesh.3scale.net/credentials** キーでサービスのワークロードにラベルを付けるのを忘れている。
- **service_id** が含まれないハンドラーを使用してアカウントごとに再利用できるようにする時に **service-mesh.3scale.net/service-id** サービスワークロードにラベルを付けるのを忘れている。
- **Rule** カスタムリソースが誤ったハンドラーまたはインスタンスカスタムリソースを参照しているか、対応する namespace の接尾辞がかけられている参照を指定している。
- **Rule** カスタムリソースの **match** セクションは、設定中のサービスと同じでない可能性があるか、現在実行中でない、または存在しない宛先ワークロードを参照している。
- ハンドラーの 3scale 管理ポータルへのアクセストークンまたは URL が正しくない。
- クエリーパラメーター、ヘッダー、認可要求などの誤った場所を指定しているか、パラメーター名がテストで使用する要求と一致しないため、インスタンスのカスタムリソースの **params/subject/properties** セクションで、**app_id**、**app_key** または **client_id** の正しいパラメーターの表示に失敗する。
- 設定ジェネレーターがアダプターコンテナイメージに実際に存在しており、**oc exec** で呼び出す必要があることに気づけなかったため、設定ジェネレーターの使用に失敗する。

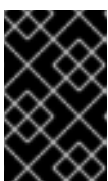
1.22. SERVICE MESH のトラブルシューティング

このセクションでは、Red Hat OpenShift Service Mesh で一般的な問題を特定し、解決する方法を説明します。以下のセクションを使用して、Red Hat OpenShift Service on AWS に Red Hat OpenShift Service Mesh をデプロイする際の問題のトラブルシューティングおよびデバッグに役立ちます。

1.22.1. Service Mesh のバージョンについて

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

- **Operator** バージョン - 最新の Operator バージョンは 2.6.7 です。Operator バージョン番号は、現在インストールされている Operator のバージョンのみを示します。Red Hat OpenShift Service Mesh Operator は Service Mesh コントロールプレーンの複数のバージョンをサポートするため、Operator のバージョンはデプロイされた **ServiceMeshControlPlane** リソースのバージョンを決定しません。



重要

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

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

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

1.22.2. Operator インストールのトラブルシューティング

1.22.2.1. Operator インストールの検証

Red Hat OpenShift Service Mesh Operator のインストール時に、OpenShift は正常な Operator インストールの一部として以下のオブジェクトを自動的に作成します。

- config map
- カスタムリソース定義
- デプロイメント
- pods
- レプリカセット
- roles
- ロールバインディング
- secrets
- サービスアカウント
- services

Red Hat OpenShift Service on AWS コンソールの使用

Red Hat OpenShift Service on AWS コンソールを使用して、Operator Pod が使用可能であり、実行していることを確認できます。

1. **Workloads** → **Pods** に移動します。
2. **openshift-operators** namespace を選択します。
3. 以下の Pod が存在し、ステータスが **running** であることを確認します。
 - **istio-operator**
 - **jaeger-operator**

- **kiali-operator**

4. **openshift-operators-redhat** namespace を選択します。
5. **elasticsearch-operator** Pod が存在し、ステータスが **running** であることを確認します。

コマンドラインで以下を行います。

1. 以下のコマンドを使用して、Operator Pod が利用可能で、**openshift-operators** namespace で実行していることを確認します。

```
$ oc get pods -n openshift-operators
```

出力例

NAME	READY	STATUS	RESTARTS	AGE
istio-operator-bb49787db-zgr87	1/1	Running	0	15s
jaeger-operator-7d5c4f57d8-9xphf	1/1	Running	0	2m42s
kiali-operator-f9c8d84f4-7xh2v	1/1	Running	0	64s

2. 以下のコマンドを使用して Elasticsearch Operator を確認します。

```
$ oc get pods -n openshift-operators-redhat
```

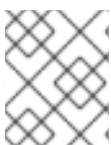
出力例

NAME	READY	STATUS	RESTARTS	AGE
elasticsearch-operator-d4f59b968-796vq	1/1	Running	0	15s

1.22.2.2. Service Mesh Operator のトラブルシューティング

Operator に問題が発生した場合は、以下を実行します。

- Operator サブスクリプションのステータスを確認します。
- サポートされる Red Hat バージョンではなく、コミュニティバージョンの Operator をインストールしていないことを確認します。
- Red Hat OpenShift Service Mesh をインストールするために **cluster-admin** ロールがあることを確認します。
- 問題が Operator のインストールに関連する場合は、Operator Pod ログでエラーの有無を確認します。



注記

Operator は OpenShift コンソールからのみインストールでき、OperatorHub はコマンドラインからアクセスできません。

1.22.2.2.1. Operator Pod ログの表示

oc logs コマンドを使用して、Operator ログを表示できます。Red Hat は、サポートケースの解決に役立つログをリクエストする場合があります。

手順

- Operator Pod ログを表示するには、以下のコマンドを入力します。

```
$ oc logs -n openshift-operators <podName>
```

以下に例を示します。

```
$ oc logs -n openshift-operators istio-operator-bb49787db-zgr87
```

1.22.3. コントロールプレーンのトラブルシューティング

Service Mesh コントロールプレーンは Istiod で構成されており、以前のいくつかのコントロールプレーンコンポーネント (Citadel、Galley、Pilot) を単一バイナリーに統合します。**ServiceMeshControlPlane** をデプロイすると、[アーキテクチャー](#) のトピックで説明されているように、Red Hat OpenShift Service Mesh を設定する他のコンポーネントも作成されます。

1.22.3.1. Service Mesh コントロールプレーンのインストールの検証

Service Mesh コントロールプレーンの作成時に、Service Mesh Operator は **ServiceMeshControlPlane** リソースファイルに指定したパラメーターを使用して以下を実行します。

- Istio コンポーネントを作成し、以下の Pod をデプロイします。
 - istiod**
 - istio-ingressgateway**
 - istio-egressgateway**
 - grafana**
 - prometheus**
- SMCP または Kiali カスタムリソースのいずれかの設定に基づいて Kiali デプロイメントを作成するには、Kiali Operator を呼び出します。



注記

Service Mesh Operator ではなく、Kiali Operator で Kiali コンポーネントを表示します。

- Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator を呼び出して、SMCP または Jaeger カスタムリソースの設定に基づいて分散トレーシング Platform (Jaeger) コンポーネントを作成します。



注記

Jaeger コンポーネントは、Service Mesh Operator ではなく、Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator の下に表示されます。Elasticsearch コンポーネントは、Red Hat Elasticsearch Operator の下に表示されます。

Red Hat OpenShift Service on AWS コンソールの使用

Red Hat OpenShift Service on AWS Web コンソールで Service Mesh コントロールプレーンのインストールを確認できます。

1. **Operators → Installed Operators** に移動します。
2. **istio-system** namespace を選択します。
3. Red Hat OpenShift Service Mesh Operator を選択します。
 - a. **Istio Service Mesh Control Plane** タブをクリックします。
 - b. コントロールプレーンの名前 (**basic** など) をクリックします。
 - c. デプロイメントによって作成されたリソースを表示するには、**Resources** タブをクリックします。フィルターを使用してビューを絞り込むことができます。たとえば、すべての **Pod** のステータスが **running** になっていることを確認できます。
 - d. SMCP のステータスが問題を示す場合は、YAML ファイルの **status:** 出力で詳細を確認してください。
4. **Operators → Installed Operators** に戻ります。
5. OpenShift Elasticsearch Operator を選択します。
 - a. **Elasticsearch** タブをクリックします。
 - b. デプロイメントの名前をクリックします (例: **elasticsearch**)。
 - c. デプロイメントによって作成されたリソースを表示するには、**Resources** タブをクリックします。
 - d. ステータス 列に問題がある場合は、**YAML** タブの **ステータス:** 出力で詳細を確認してください。
6. **Operators → Installed Operators** に戻ります。
7. Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator を選択します。
 - a. **Jaeger** タブをクリックします。
 - b. デプロイメントの名前をクリックします (例: **jaeger**)。
 - c. デプロイメントによって作成されたリソースを表示するには、**Resources** タブをクリックします。
 - d. ステータス 列に問題がある場合は、**YAML** タブの **status:** 出力で詳細を確認してください。
8. **Operators → Installed Operators** に移動します。
9. Kiali Operator を選択します。
 - a. **Istio Service Mesh Control Plane** タブをクリックします。
 - b. デプロイメントの名前をクリックします (例: **kiali**)。
 - c. デプロイメントによって作成されたリソースを表示するには、**Resources** タブをクリックします。

- d. ステータス 列に問題がある場合は、**YAML** タブの **ステータス:** 出力で詳細を確認してください。

コマンドラインで以下を行います。

1. 以下のコマンドを実行して、Service Mesh コントロールプレーン Pod が利用可能かどうか確認します。**istio-system** は SMCP をインストールした namespace になります。

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

出力例

NAME	READY	STATUS	RESTARTS	AGE
grafana-6776785cfc-6fz7t	2/2	Running	0	102s
istio-egressgateway-5f49dd99-l9ppq	1/1	Running	0	103s
istio-ingressgateway-6dc885c48-jjd8r	1/1	Running	0	103s
istiod-basic-6c9cc55998-wg4zq	1/1	Running	0	2m14s
jaeger-6865d5d8bf-zrfss	2/2	Running	0	100s
kiali-579799fbb7-8mwc8	1/1	Running	0	46s
prometheus-5c579dfb-6qhjk	2/2	Running	0	115s

2. 次のコマンドを使用して、Service Mesh コントロールプレーンのデプロイのステータスを確認します。**istio-system** は、SMCP をデプロイした namespace に置き換えます。

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

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

出力例

NAME	READY	STATUS	PROFILES	VERSION	AGE
basic	10/10	ComponentsReady	["default"]	2.1.3	4m2s

Service Mesh コントロールプレーンを変更および再デプロイする場合、ステータスは **UpdateSuccessful** が表示されるはずです。

出力例

NAME	READY	STATUS	TEMPLATE	VERSION	AGE
basic-install	10/10	UpdateSuccessful	default	v1.1	3d16h

3. SMCP のステータスが **ComponentsReady** 以外の場合は、SCMP リソースの **status:** 出力で詳細を確認してください。

```
$ oc describe smcp <smcp-name> -n <controlplane-namespace>
```

出力例

```
$ oc describe smcp basic -n istio-system
```

4. 以下のコマンドを使用して、Jaeger デプロイメントのステータスを確認します。ここでは、**istio-system** は SMCP をデプロイした namespace に置き換えます。

-

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

出力例

```
NAME      STATUS   VERSION  STRATEGY  STORAGE  AGE
jaeger    Running  1.30.0   allinone  memory   15m
```

- 以下のコマンドを使用して、Kiali デプロイメントのステータスを確認します。ここでは、**istio-system** は SMCP をデプロイした namespace に置き換えます。

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

出力例

```
NAME    AGE
kiali   15m
```

1.22.3.1.1. Kiali コンソールへのアクセス

Kiali コンソールでアプリケーションのトポロジー、健全性、およびメトリクスを表示できます。サービスで問題が発生した場合、Kiali コンソールは、サービス経由でデータフローを表示できます。抽象アプリケーションからサービスおよびワークロードまで、さまざまなレベルでのメッシュコンポーネントに関する洞察を得ることができます。Kiali は、リアルタイムで namespace のインタラクティブなグラフビューも提供します。

Kiali コンソールにアクセスするには、Red Hat OpenShift Service Mesh がインストールされ、Kiali がインストールおよび設定されている必要があります。

インストールプロセスにより、Kiali コンソールにアクセスするためのルートが作成されます。

Kiali コンソールの URL が分かっている場合は、直接アクセスできます。URL が分からない場合は、以下の指示を使用します。

管理者の手順

1. 管理者ロールで Red Hat OpenShift Service on AWS Web コンソールにログインします。
2. **Home → Projects** をクリックします。
3. **Projects** ページで、必要に応じてフィルターを使用してプロジェクトの名前を検索します。
4. プロジェクトの名前 (例: **info**) をクリックします。
5. **Project details** ページの **Launcher** セクションで、**Kiali** リンクをクリックします。
6. Red Hat OpenShift Service on AWS コンソールにアクセスするときに使用するものと同じユーザー名とパスワードを使用して Kiali コンソールにログインします。
初回の Kiali コンソールへのログイン時に、表示するパーミッションを持つ Service Mesh 内のすべての namespace を表示する **Overview** ページが表示されます。

コンソールのインストールを検証中で、namespace がまだメッシュに追加されていないと、**istio-system** 以外のデータは表示されない可能性があります。

開発者の手順

1. 開発者ロールで Red Hat OpenShift Service on AWS Web コンソールにログインします。
2. **Project** をクリックします。
3. 必要に応じて、**Project Details** ページで、フィルターを使用してプロジェクトの名前を検索します。
4. プロジェクトの名前 (例: **info**) をクリックします。
5. **Project** ページの **Launcher** セクションで、**Kiali** リンクをクリックします。
6. **Log In With OpenShift** をクリックします。

1.22.3.1.2. Jaeger コンソールへのアクセス

Jaeger コンソールにアクセスするには、Red Hat OpenShift Service Mesh がインストールされ、Red Hat OpenShift 分散トレーシング Platform (Jaeger) がインストールおよび設定されている必要があります。

インストールプロセスにより、Jaeger コンソールにアクセスするためのルートが作成されます。

Jaeger コンソールの URL が分かっている場合は、これに直接アクセスできます。URL が分からない場合は、以下の指示を使用します。



重要

Red Hat OpenShift Service Mesh 2.5 以降、Red Hat OpenShift 分散トレーシング Platform (Jaeger) および OpenShift Elasticsearch Operator は非推奨となり、今後のリリースで削除される予定です。Red Hat は、現在のリリースのライフサイクル中にこの機能のバグ修正とサポートを提供しますが、この機能は今後、機能拡張を受け取らず、削除されます。Red Hat OpenShift 分散トレーシング Platform (Jaeger) の代わりに、Red Hat OpenShift 分散トレーシング Platform を使用することもできます。

OpenShift コンソールからの手順

1. cluster-admin 権限を持つユーザーとして Red Hat OpenShift Service on AWS Web コンソールにログインします。(Red Hat OpenShift Dedicated を使用する場合は **dedicated-admin** ロールがあるアカウント。
2. **Networking** → **Routes** に移動します。
3. **Routes** ページで、**Namespace** メニューから Service Mesh コントロールプレーンプロジェクトを選択します (例: **istio-system**)。
Location 列には、各ルートのリンク先アドレスが表示されます。
4. 必要な場合は、フィルターを使用して **jaeger** ルートを検索します。ルートの **Location** をクリックしてコンソールを起動します。
5. **Log In With OpenShift** をクリックします。

Kiali コンソールからの手順

1. Kiali コンソールを起動します。
2. 左側のナビゲーションペインで **Distributed Tracing** をクリックします。

3. **Log In With OpenShift** をクリックします。

CLI からの手順

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

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

2. コマンドラインを使用してルートの詳細をクエリーするには、以下のコマンドを入力します。この例では、**istio-system** が Service Mesh コントロールプレーンの namespace です。

```
$ oc get route -n istio-system jaeger -o jsonpath='{.spec.host}'
```

3. ブラウザーを起動し、**https://<JAEGER_URL>** に移動します。ここで、**<JAEGER_URL>** は直前の手順で検出されたルートです。
4. Red Hat OpenShift Service on AWS 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ログインします。
5. Service Mesh にサービスを追加し、トレースを生成している場合は、フィルターと **Find Traces** ボタンを使用してトレースデータを検索します。
コンソールインストールを検証すると、表示するトレースデータはありません。

1.22.3.2. Service Mesh コントロールプレーンのトラブルシューティング

Service Mesh コントロールプレーンのデプロイ時に問題が発生した場合は、

- **ServiceMeshControlPlane** リソースがサービスおよび Operator とは別のプロジェクトにインストールされていることを確認します。このドキュメントでは **istio-system** プロジェクトをサンプルとして使用しますが、Operator およびサービスが含まれるプロジェクトから分離されている限り、コントロールプレーンを任意のプロジェクトにデプロイできます。
- **ServiceMeshControlPlane** および **Jaeger** カスタムリソースが同じプロジェクトにデプロイされていることを確認します。たとえば、両方の **istio-system** プロジェクトを使用します。

1.22.4. データプレーンのトラブルシューティング

データプレーン は、Service Mesh 内のサービス間の受信および送信ネットワーク通信をすべて傍受し、制御するインテリジェントプロキシのセットです。

Red Hat OpenShift Service Mesh は、アプリケーションの Pod 内のプロキシサイドカーに依存して、アプリケーションに Service Mesh 機能を提供します。

1.22.4.1. サイドカーインジェクションのトラブルシューティング

Red Hat OpenShift Service Mesh は、プロキシサイドカーコンテナを Pod に自動的に挿入させません。サイドカーインジェクションをオプトインする必要があります。

1.22.4.1.1. Istio サイドカーインジェクションのトラブルシューティング

アプリケーションのデプロイメントで自動インジェクションが有効になっているかどうかを確認します。Envoy プロキシの自動インジェクションが有効になっている場合

は、**spec.template.metadata.annotations** の下の **Deployment** リソースに **sidecar.istio.io/inject:"true"** アノテーションがなければなりません。

1.22.4.1.2. Jaeger エージェントのサイドカーインジェクションのトラブルシューティング

アプリケーションのデプロイメントで自動インジェクションが有効になっているかどうかを確認します。Jaeger エージェントの自動インジェクションが有効な場合は、**Deployment** リソースに **sidecar.jaegertracing.io/inject:"true"** アノテーションが必要です。

サイドカーインジェクションの詳細は、[自動インジェクションの有効化](#) を参照してください。

1.23. ENVOY プロキシのトラブルシューティング

Envoy プロキシは、Service Mesh 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 Service Mesh でテレメトリを収集し、報告します。Envoy は、同じ Pod の関連するサービスに対してサイドカーコンテナとしてデプロイされます。

1.23.1. Envoy アクセスログの有効化

Envoy アクセスログは、トラフィックの障害およびフローの診断に役立ち、エンドツーエンドのトラフィックフロー分析に役立ちます。

すべての istio-proxy コンテナのアクセスロギングを有効にするには、**ServiceMeshControlPlane** (SMCP) オブジェクトを編集してロギングの出力のファイル名を追加します。

手順

1. cluster-admin ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。以下のコマンドを入力します。次に、プロンプトが表示されたら、ユーザー名とパスワードを入力します。

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

2. Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) に切り替えます。

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

3. **ServiceMeshControlPlane** ファイルを編集します。

```
$ oc edit smcp <smcp_name>
```

4. 以下の例で示すように、**name** を使用してプロキシログのファイル名を指定します。**name** の値を指定しないと、ログエントリは書き込まれません。

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

```
accessLogging:
  file:
    name: /dev/stdout  #file name
```

1.23.2. サポート

このドキュメントで説明されている手順、または Red Hat OpenShift Service on AWS 全般で問題が発生した場合は、[Red Hat カスタマーポータル](#) にアクセスしてください。

カスタマーポータルでは、以下を行うことができます。

- Red Hat 製品に関するアークティクルおよびソリューションを対象とした Red Hat ナレッジベースの検索またはブラウズ。
- Red Hat サポートに対するサポートケースの送信。
- その他の製品ドキュメントへのアクセス。

クラスターの問題を特定するには、[OpenShift Cluster Manager](#) で Insights を使用できます。Insights により、問題の詳細と、利用可能な場合は問題の解決方法に関する情報が提供されます。

このドキュメントの改善への提案がある場合、またはエラーを見つけた場合は、最も関連性の高いドキュメントコンポーネントの [Jira Issue](#) を送信してください。セクション名や Red Hat OpenShift Service on AWS のバージョンなど、具体的な情報を指定してください。

1.23.2.1. Red Hat ナレッジベースについて

[Red Hat ナレッジベース](#) は、お客様が Red Hat の製品やテクノロジーを最大限に活用できるようにするための豊富なコンテンツを提供します。Red Hat ナレッジベースは、Red Hat 製品のインストール、設定、および使用に関する記事、製品ドキュメント、および動画で構成されています。さらに、既知の問題に対する解決策を検索でき、それぞれに根本原因の簡潔な説明と修復手順が記載されています。

1.23.2.2. Red Hat ナレッジベースの検索

Red Hat OpenShift Service on AWS の問題が発生した場合には、初期検索を実行して、Red Hat ナレッジベースにソリューションがすでに存在しているかどうかを確認できます。

前提条件

- Red Hat カスタマーポータルのアカウントがある。

手順

1. [Red Hat カスタマーポータル](#) にログインします。
2. **Search** をクリックします。
3. 検索フィールドに、問題に関連する次のようなキーワードと文字列を入力します。
 - Red Hat OpenShift Service on AWS コンポーネント (**etcd** など)
 - 関連する手順 (**installation** など)
 - 明示的な失敗に関連する警告、エラーメッセージ、およびその他の出力

4. **Enter** キーをクリックします。
5. オプション: **Red Hat OpenShift Service on AWS**製品フィルターを選択します。
6. オプション: **Documentation** コンテンツタイプフィルターを選択します。

1.23.2.3. サポートケースの送信

前提条件

- **dedicated-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- Red Hat OpenShift Cluster Manager にアクセスできる。

手順

1. Red Hat カスタマーポータルの [Customer Support ページ](#) にログインします。
2. **Get support** をクリックします。
3. **Customer Support** ページの **Cases** タブで、以下を行います。
 - a. オプション: 必要に応じて、事前に入力されたアカウントと所有者の詳細を変更します。
 - b. 問題に該当するカテゴリ (**Bug**、**Defect** など) を選択し、**Continue** をクリックします。
4. 以下の情報を入力します。
 - a. **Summary** フィールドには、問題の簡潔で説明的な概要と、確認されている現象および予想される動作の詳細情報を入力します。
 - b. **Product** ドロップダウンメニューから **Red Hat OpenShift Service on AWS**を選択します。
5. Red Hat ナレッジベースで推奨されるソリューション一覧を確認してください。この一覧に上げられているソリューションは、報告しようとしている問題に適用される可能性があります。提案されている記事が問題に対応していない場合は、**Continue** をクリックします。
6. 報告している問題に対する一致に基づいて推奨される Red Hat ナレッジベースソリューションの一覧が更新されることを確認してください。ケース作成プロセスでより多くの情報を提供すると、このリストの絞り込みが行われます。提案されている記事が問題に対応していない場合は、**Continue** をクリックします。
7. アカウント情報が予想通りに表示されていることを確認し、そうでない場合は適宜修正します。
8. 自動入力された Red Hat OpenShift Service on AWS クラスター ID が正しいことを確認します。正しくない場合は、クラスター ID を手動で取得します。
 - Red Hat OpenShift Service on AWS Web コンソールを使用して手動でクラスター ID を取得するには以下を実行します。
 - a. **Home** → **Overview** に移動します。
 - b. **Details** セクションの **Cluster ID** フィールドで値を見つけます。

- または、Red Hat OpenShift Service on AWS Web コンソールから新しいサポートケースを開き、クラスター ID を自動入力することもできます。
 - a. ツールバーから、(?)Help → Open Support Case に移動します。
 - b. Cluster ID 値が自動的に入力されます。
- OpenShift CLI (oc) を使用してクラスター ID を取得するには、以下のコマンドを実行します。

```
$ oc get clusterversion -o jsonpath='{.items[].spec.clusterID}'{"\n"}
```

9. プロンプトが表示されたら、以下の質問に回答し、**Continue** をクリックします。

- What are you experiencing? What are you expecting to happen?
- Define the value or impact to you or the business.
- Where are you experiencing this behavior? What environment?
- When does this behavior occur? Frequency? Repeatedly? At certain times?

10. 関連する診断データファイルをアップロードし、**Continue** をクリックします。

11. 関連するケース管理の詳細情報を入力し、**Continue** をクリックします。

12. ケースの詳細をプレビューし、**Submit** をクリックします。

1.24. SERVICE MESH コントロールプレーン設定の参照

デフォルトの **ServiceMeshControlPlane** (SMCP) リソースを変更するか、完全にカスタムの SMCP リソースを作成して Red Hat OpenShift Service Mesh をカスタマイズできます。このリファレンスセクションでは、SMCP リソースで利用可能な設定オプションを説明します。

1.24.1. Service Mesh コントロールプレーンのパラメーター

以下の表は、**ServiceMeshControlPlane** リソースのトップレベルのパラメーターを一覧表示しています。

表1.31 **ServiceMeshControlPlane** リソースパラメーター

名前	説明	型
apiVersion	APIVersion はオブジェクトのこの表現のバージョンスキーマを定義します。サーバーは認識されたスキーマを最新の内部値に変換し、認識されない値は拒否することがあります。 ServiceMeshControlPlane バージョン 2.0 の値は maistra.io/v2 です。	ServiceMeshControlPlane バージョン 2.0 の値は maistra.io/v2 です。

名前	説明	型
kind	kind はこのオブジェクトが表す REST リソースを表す文字列の値です。	ServiceMeshControlPlane で唯一有効な値は、 ServiceMeshControlPlane です。
metadata	この ServiceMeshControlPlane インスタンスに関するメタデータ。Service Mesh コントロールプレーンインストールの名前を指定して作業を追跡できます (basic など)。	string
spec	この ServiceMeshControlPlane の必要な状態の仕様です。これには、Service Mesh コントロールプレーンを構成するすべてのコンポーネントの設定オプションが含まれます。	詳細は、表 2 を参照してください。
status	この ServiceMeshControlPlane と、Service Mesh コントロールプレーンを構成するコンポーネントの現在のステータス。	詳細は、表 3 を参照してください。

以下の表は、**ServiceMeshControlPlane** リソースの仕様を一覧表示しています。これらのパラメーターを変更すると、Red Hat OpenShift Service Mesh コンポーネントが設定されます。

表1.32 **ServiceMeshControlPlane** リソース仕様

名前	説明	設定可能なパラメーター
addons	addons パラメーターを使用して、可視化やメトリクスストレージなど、コアの Service Mesh コントロールプレーンコンポーネント以外の追加機能を設定します。	3scale 、 grafana 、 jaeger 、 kiali 、および prometheus
cluster	cluster パラメーターは、クラスターの一般的な設定 (クラスター名、ネットワーク名、マルチクラスター、メッシュ拡張など) の設定を行います。	meshExpansion 、 multiCluster 、 name 、および network

名前	説明	設定可能なパラメーター
gateways	gateways パラメーターを使用して、メッシュの ingress および egress ゲートウェイを設定します。	enabled 、 additionalEgress 、 additionalIngress 、 egress 、 ingress 、および openshiftRoute
general	general パラメーターは、その他の場所には適合しない一般的な Service Mesh コントロールプレーンの設定を表します。	logging および validationMessages
policy	policy パラメーターを使用して、Service Mesh コントロールプレーンのポリシーチェックを設定します。ポリシーチェックを有効にするには、 spec.policy.enabled を true に設定します。	mixer remote 、または type 。 type は Istiod 、 Mixer または None に設定できます。
profiles	profiles パラメーターを使用して、デフォルト値に使用するために ServiceMeshControlPlane プロファイルを選択します。	default
proxy	proxy パラメーターを使用してサイドカーのデフォルト動作を設定します。	accessLogging 、 adminPort 、 concurrency 、および envoyMetricsService
runtime	runtime パラメーターを使用して、Service Mesh コントロールプレーンコンポーネントを設定します。	components 、および defaults
security	security パラメーターを使用すると、Service Mesh コントロールプレーンのセキュリティの各種機能を設定できます。	certificateAuthority 、 controlPlane 、 identity 、 dataPlane および trust
techPreview	techPreview パラメーターを使用すると、テクノロジープレビュー機能への早期アクセスが可能になります。	該当なし
telemetry	spec.mixer.telemetry.enabled が true に設定されている場合、telemetry は有効にされます。	mixer 、 remote 、および type 。 type は Istiod 、 Mixer または None に設定できます。

名前	説明	設定可能なパラメーター
tracing	tracing パラメーターを使用して、メッシュの分散トレースを有効にします。	sampling 、 type 。 type は Jaeger または None に設定できます。
version	version パラメーターは、インストールする Service Mesh コントロールプレーンの Maistra バージョンを指定します。空のバージョンで ServiceMeshControlPlane を作成する場合、受付 Webhook はバージョンを現行バージョンに設定します。空のバージョンの新規の ServiceMeshControlPlanes は v2.0 に設定されます。空のバージョンの既存の ServiceMeshControlPlanes はそれらの設定を保持します。	string

ControlPlaneStatus は Service Mesh の現在の状態を表します。

表1.33 **ServiceMeshControlPlane** リソース **ControlPlaneStatus**

名前	説明	型
annotations	annotations パラメーターは、通常は ServiceMeshControlPlane によってデプロイされるコンポーネントの数などの追加の余分なステータス情報を保存します。これらのステータスは、JSONPath 式でオブジェクトのカウントを許可しないコマンドラインツールの oc で使用されます。	設定不可

名前	説明	型
conditions	オブジェクトの現在の状態として観察される最新の状態を表します。 Reconcile は、Operator がデプロイされるコンポーネントの実際の状態の調整を ServiceMeshControlPlane リソースの設定を使用して完了したかどうかを示します。 Installed は、Service Mesh コントロールプレーンがインストールされているかどうかを示します。 Ready は、すべての Service Mesh コントロールプレーンコンポーネントの準備ができているかどうかを示します。	string
コンポーネント	デプロイされた各 Service Mesh コントロールプレーンコンポーネントのステータスを表示します。	string
appliedSpec	すべてのプロファイルが適用された後に生成される設定の仕様です。	ControlPlaneSpec
appliedValues	チャートの生成に使用される生成される values.yaml です。	ControlPlaneSpec
chartVersion	このリソースに対して最後に処理されたチャートのバージョンです。	string
observedGeneration	直近の調整時にコントローラーによって観察される生成です。ステータスの情報は、オブジェクトの特定の生成に関連するものです。 status.conditions は、 status.observedGeneration フィールドが metadata.generation に一致しない場合は最新の状態ではありません。	integer
operatorVersion	このリソースを最後に処理した Operator のバージョンです。	string
readiness	コンポーネントおよび所有リソースの準備状態 (readiness) のステータス	string

この例の **ServiceMeshControlPlane** の定義には、サポート対象のパラメーターがすべて含まれます。

ServiceMeshControlPlane リソースの例

```

apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
  proxy:
    runtime:
      container:
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 500m
            memory: 128Mi
  tracing:
    type: Jaeger
  gateways:
    ingress: # istio-ingressgateway
      service:
        type: ClusterIP
        ports:
          - name: status-port
            port: 15020
          - name: http2
            port: 80
            targetPort: 8080
          - name: https
            port: 443
            targetPort: 8443
        meshExpansionPorts: []
    egress: # istio-egressgateway
      service:
        type: ClusterIP
        ports:
          - name: status-port
            port: 15020
          - name: http2
            port: 80
            targetPort: 8080
          - name: https
            port: 443
            targetPort: 8443
  additionalIngress:
    some-other-ingress-gateway: {}
  additionalEgress:
    some-other-egress-gateway: {}

  policy:
    type: Mixer
    mixer: # only applies if policy.type: Mixer

```

```

enableChecks: true
failOpen: false

telemetry:
  type: Istiod # or Mixer
  mixer: # only applies if telemetry.type: Mixer, for v1 telemetry
  sessionAffinity: false
  batching:
    maxEntries: 100
    maxTime: 1s
  adapters:
    kubernetesenv: true
    stdio:
      enabled: true
      outputAsJSON: true
addons:
  grafana:
    enabled: true
    install:
      config:
        env: {}
        envSecrets: {}
      persistence:
        enabled: true
        storageClassName: ""
        accessMode: ReadWriteOnce
        capacity:
          requests:
            storage: 5Gi
      service:
        ingress:
          contextPath: /grafana
        tls:
          termination: reencrypt
  kiali:
    name: kiali
    enabled: true
    install: # install kiali CR if not present
    dashboard:
      viewOnly: false
      enableGrafana: true
      enableTracing: true
      enablePrometheus: true
    service:
      ingress:
        contextPath: /kiali
  jaeger:
    name: jaeger
    install:
      storage:
        type: Elasticsearch # or Memory
      memory:
        maxTraces: 100000
      elasticsearch:
        nodeCount: 3
        storage: {}

```

```

    redundancyPolicy: SingleRedundancy
    indexCleaner: {}
    ingress: {} # jaeger ingress configuration
runtime:
  components:
    pilot:
      deployment:
        replicas: 2
      pod:
        affinity: {}
      container:
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 500m
            memory: 128Mi
    grafana:
      deployment: {}
      pod: {}
    kiali:
      deployment: {}
      pod: {}

```

1.24.2. 仕様パラメーター

1.24.2.1. 一般的なパラメーター

以下の例は、**ServiceMeshControlPlane** オブジェクトの **spec.general** パラメーターと適切な値を持つ利用可能なパラメーターの説明を示しています。

一般的なパラメーターの例

```

apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  general:
    logging:
      componentLevels: {}
      # misc: error
    logAsJSON: false
    validationMessages: true

```

表1.34 Istio の一般的なパラメーター

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

パラメーター	説明	値	デフォルト値
<code>logging:</code>	Service Mesh コントロールプレーンコンポーネントのロギングを設定するために使用します。		該当なし
<code>logging: componentLevels:</code>	コンポーネントのロギングレベルを指定するために使用します。	使用できる値は、 debug 、 info 、 warn 、 error 、 fatal です。	該当なし
<code>logging: logAsJSON:</code>	JSON ロギングを有効または無効にします。	true/false	該当なし
<code>validationMessages:</code>	istio.io リソースの status フィールドへの検証メッセージを有効または無効にするのに使用します。これは、リソースで設定エラーを検出するのに役立ちます。	true/false	該当なし

1.24.2.2. プロファイルパラメーター

ServiceMeshControlPlane オブジェクトプロファイルを使用すると、再利用可能な設定を作成できます。**profile** 設定を設定しない場合は、Red Hat OpenShift Service Mesh は default プロファイルを使用します。

以下の例は、**ServiceMeshControlPlane** オブジェクトの **spec.profiles** パラメーターを示しています。

プロファイルパラメーターの例

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

プロファイルの作成に関する詳細は、[コントロールプレーンプロファイルの作成](#) を参照してください。

セキュリティ設定の詳細な例は、[Mutual Transport Layer Security \(mTLS\)](#) を参照してください。

1.24.2.3. トレースパラメーター

以下の例は、**ServiceMeshControlPlane** オブジェクトの **spec.tracing** パラメーターと適切な値を持つ利用可能なパラメーターの説明を示しています。

トレースパラメーターの例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
  tracing:
    sampling: 100
    type: Jaeger
```

表1.35 Istio トレースパラメーター

パラメーター	説明	値	デフォルト値
tracing: sampling:	サンプリングレートは、Envoy プロキシがトレースを生成する頻度を決定します。サンプリングレートを使用して、トレースシステムに報告される要求の割合を制御します。	0 から 10000 までの整数値で、インクリメントは 0.01% (0 から 100%) になります。たとえば、値を 10 に設定するとリクエストの 0.1% がサンプリングされ、値を 100 に設定するとリクエストの 1% がサンプリングされ、値を 500 に設定するとリクエストの 5% がサンプリングされ、 10000 に設定するとリクエストの 100% がサンプリングされます。	10000 (トレースの 100%)
tracing: type:	現在、サポートされるトレーサーの唯一のタイプは Jaeger です。Jaeger はデフォルトで有効になっています。トレースを無効にするには、 type パラメーターを None に設定します。	none 、 Jaeger	Jaeger

1.24.2.4. バージョンパラメーター

Red Hat OpenShift Service Mesh Operator は、さまざまなバージョンの **ServiceMeshControlPlane** のインストールをサポートしています。version パラメーターは、インストールする Service Mesh コントロールプレーンのバージョンを指定します。SMCP の作成時にバージョンパラメーターを指定しないと、Operator は値を最新バージョン (2.6) に設定します。既存の **ServiceMeshControlPlane** オブジェクトは、Operator のアップグレード中にバージョン設定を保持します。

1.24.2.5. 3scale の設定

以下の表では、**ServiceMeshControlPlane** リソースの 3scale Istio アダプターのパラメーターを説明しています。

3scale パラメーターの例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  addons:
    3Scale:
      enabled: false
      PARAM_THREESCALE_LISTEN_ADDR: 3333
      PARAM_THREESCALE_LOG_LEVEL: info
      PARAM_THREESCALE_LOG_JSON: true
      PARAM_THREESCALE_LOG_GRPC: false
      PARAM_THREESCALE_REPORT_METRICS: true
      PARAM_THREESCALE_METRICS_PORT: 8080
      PARAM_THREESCALE_CACHE_TTL_SECONDS: 300
      PARAM_THREESCALE_CACHE_REFRESH_SECONDS: 180
      PARAM_THREESCALE_CACHE_ENTRIES_MAX: 1000
      PARAM_THREESCALE_CACHE_REFRESH_RETRIES: 1
      PARAM_THREESCALE_ALLOW_INSECURE_CONN: false
      PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS: 10
      PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS: 60
      PARAM_USE_CACHED_BACKEND: false
      PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS: 15
      PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED: true
# ...
```

表1.36 3scale パラメーター

パラメーター	説明	値	デフォルト値
enabled	3scale アダプターを使用するかどうか	true/false	false
PARAM_THREESCALE_LISTEN_ADDR	gRPC サーバーのリ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

パラメーター	説明	値	デフォルト値
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
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

パラメーター	説明	値	デフォルト値
PARAM_USE_CACHE_BACKEND	true の場合は、認可要求のインメモリー apisonator キャッシュの作成を試行します。	true/false	false
PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS	バックエンドキャッシュが有効な場合は、3scale に対してキャッシュをフラッシュする間隔を秒単位で設定します。	時間 (秒単位)	15
PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED	バックエンドキャッシュが承認データを取得できない場合は常に、要求を拒否する (クローズする) か、許可する (オープンする) かどうか。	true/false	true

1.24.3. ステータスパラメーター

status パラメーターは、Service Mesh の現在の状態を記述します。この情報は Operator によって生成され、読み取り専用です。

表1.37 Istio ステータスパラメーター

名前	説明	型
observedGeneration	直近の調整時にコントローラーによって観察される生成です。ステータスの情報は、オブジェクトの特定の生成に関連するものです。 status.conditions は、 status.observedGeneration フィールドが metadata.generation に一致しない場合は最新の状態ではありません。	integer

名前	説明	型
annotations	annotations パラメーターは、通常は ServiceMeshControlPlane オブジェクトによってデプロイされるコンポーネントの数などの追加の余分なステータス情報を保存します。これらのステータスは、JSONPath 式でオブジェクトのカウントを許可しないコマンドラインツールの oc で使用されます。	設定不可
readiness	コンポーネントおよび所有リソースの readiness ステータスです。	string
operatorVersion	このリソースを最後に処理した Operator のバージョンです。	string
コンポーネント	デプロイされた各 Service Mesh コントロールプレーンコンポーネントのステータスを表示します。	string
appliedSpec	すべてのプロファイルが適用された後に生成される設定の仕様です。	ControlPlaneSpec
conditions	オブジェクトの現在の状態として観察される最新の状態を表します。 Reconciled は、Operator がデプロイされるコンポーネントの実際の状態の調整を ServiceMeshControlPlane リソースの設定を使用して完了したかどうかを示します。インストール済みは、Service Mesh コントロールプレーンがインストールされていることを示します。 Ready は、すべての Service Mesh コントロールプレーンコンポーネントの準備が整っていることを示します。	string
chartVersion	このリソースに対して最後に処理されたチャートのバージョンです。	string
appliedValues	チャートの生成に使用された、生成される values.yaml ファイル。	ControlPlaneSpec

1.24.4. 関連情報

- **ServiceMeshControlPlane** リソースで機能を設定する方法の詳細は、以下のリンクを参照してください。
 - [セキュリティ](#)
 - [トラフィック管理](#)
 - [メトリクスとトレース](#)

1.25. KIALI 設定リファレンス

Service Mesh Operator は **ServiceMeshControlPlane** を作成する際に、Kiali リソースも処理します。次に Kiali Operator は Kiali インスタンスの作成時にこのオブジェクトを使用します。

1.25.1. SMCP での Kiali 設定の指定

Kiali は、**ServiceMeshControlPlane** リソースの **addons** セクションで設定できます。Kiali はデフォルトで有効です。Kiali を無効にするには、**spec.addons.kiali.enabled** を **false** に設定します。

Kiali 設定は、以下の 2 つの方法のいずれかで指定できます。

- **spec.addons.kiali.install** の **ServiceMeshControlPlane** リソースで Kiali 設定を指定します。Kiali 設定の完全なリストが SMCP で利用できないため、このアプローチにはいくつかの制限があります。
- Kiali インスタンスを設定してデプロイし、Kiali リソースの名前を **ServiceMeshControlPlane** リソースの **spec.addons.kiali.name** の値として指定します。CR は、Service Mesh コントロールプレーンと同じ namespace (例: **istio-system**) に作成する必要があります。**name** の値に一致する Kiali リソースが存在する場合、コントロールプレーンは、そのコントロールプレーンで使用する対象の Kiali リソースを設定します。このアプローチにより、Kiali リソースで Kiali 設定を完全にカスタマイズできます。このアプローチでは、Kiali リソースのさまざまなフィールド (例: **accessible_namespaces** リスト)、および Grafana、Prometheus、およびトレースのエンドポイントが上書きされることに注意してください。

Kiali の SMCP パラメーターの例

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  addons:
    kiali:
      name: kiali
      enabled: true
      install:
        dashboard:
          viewOnly: false
          enableGrafana: true
          enableTracing: true
          enablePrometheus: true
        service:
          ingress:
            contextPath: /kiali
```

表1.38 ServiceMeshControlPlane Kiali パラメーター

パラメーター	説明	値	デフォルト値
spec: addons: kiali: name:	Kiali カスタムリソースの名前。 name の値に一致する Kiali CR が存在する場合、Service Mesh Operator はその CR をインストールに使用します。Kiali CR が存在しない場合は、Operator はこの名前と SMCP で指定された設定オプションを使用して Kiali CR を作成します。	string	kiali
kiali: enabled:	このパラメーターは、Kiali を有効または無効にします。Kiali はデフォルトで有効です。	true/false	true
kiali: install:	指定された Kiali リソースが存在しない場合は、Kiali リソースをインストールします。 addons.kiali.enabled が false に設定されている場合は、 install セクションは無視されます。		
kiali: install: dashboard:	Kiali に付属のダッシュボードの設定パラメーター。		
kiali: install: dashboard: viewOnly:	このパラメーターは、Kiali コンソールの表示専用 (view-only) モードを有効または無効にします。表示専用モードを有効にすると、ユーザーは Kiali コンソールを使用して Service Mesh を変更できなくなります。	true/false	false

パラメーター	説明	値	デフォルト値
kiali: install: dashboard: enableGrafana:	spec.addons.grafana 設定に基づいて設定された Grafana エンドポイント。	true/false	true
kiali: install: dashboard: enablePrometheus:	spec.addons.prometheus 設定に基づいて設定された Prometheus エンドポイント。	true/false	true
kiali: install: dashboard: enableTracing:	Jaeger カスタムリソース設定に基づいて設定されたトレースエンドポイント。	true/false	true
kiali: install: service:	Kiali インストールに関連付けられた Kubernetes サービスの設定パラメーター。		
kiali: install: service: metadata:	リソースに適用する追加のメタデータを指定するために使用します。	該当なし	該当なし
kiali: install: service: metadata: annotations:	コンポーネントのサービスに適用するアノテーションを追加で指定するために使用します。	string	該当なし
kiali: install: service: metadata: labels:	コンポーネントのサービスに適用するラベルを追加で指定するために使用します。	string	該当なし

パラメーター	説明	値	デフォルト値
<code>kiali:</code> <code>install:</code> <code>service:</code> <code>ingress:</code>	OpenShift Route を介してコンポーネントのサービスにアクセスする詳細を指定するために使用します。	該当なし	該当なし
<code>kiali:</code> <code>install:</code> <code>service:</code> <code>ingress:</code> <code>metadata:</code> <code>annotations:</code>	コンポーネントのサービス入力に適用する注アノテーションを追加で指定するために使用します。	string	該当なし
<code>kiali:</code> <code>install:</code> <code>service:</code> <code>ingress:</code> <code>metadata:</code> <code>labels:</code>	コンポーネントのサービス ingress に適用するラベルを追加で指定するために使用します。	string	該当なし
<code>kiali:</code> <code>install:</code> <code>service:</code> <code>ingress:</code> <code>enabled:</code>	コンポーネントに関連付けられたサービスの OpenShift ルートをカスタマイズするために使用します。	true/false	true
<code>kiali:</code> <code>install:</code> <code>service:</code> <code>ingress:</code> <code>contextPath:</code>	サービスへのコンテキストパスを指定するために使用します。	string	該当なし
<code>install:</code> <code>service:</code> <code>ingress:</code> <code>hosts:</code>	OpenShift ルートごとに単一のホスト名を指定するために使用します。空のホスト名は、ルートのデフォルトのホスト名を意味します。	string	該当なし

パラメーター	説明	値	デフォルト値
install: service: ingress: tls:	OpenShift ルートの TLS を設定するために使用します。		該当なし
kiali: install: service: nodePort:	コンポーネントのサービス Values 。 <component>.service.nodePort.port の nodePort を指定するために使用します	integer	該当なし

1.25.2. Kiali カスタムリソースでの Kiali 設定の指定

ServiceMeshControlPlane (SMCP) リソースではなく、Kiali カスタムリソース (CR) で Kiali を設定することにより、Kiali デプロイメントを完全にカスタマイズできます。この設定は SMCP の外部で指定されるため、"外部 Kiali" と呼ばれることもあります。



注記

ServiceMeshControlPlane と Kiali カスタムリソースを同じ namespace にデプロイする必要があります。たとえば、**istio-system** です。

Kiali インスタンスを設定してデプロイしてから、SMCP リソースの **spec.addons.kiali.name** の値として Kiali リソースの **name** を指定できます。**name** の値に一致する Kiali CR が存在する場合、Service Mesh コントロールプレーンは既存のインストールを使用します。この方法では、Kiali 設定を完全にカスタマイズできます。

1.26. JAEGER 設定リファレンス

Service Mesh Operator は **ServiceMeshControlPlane** リソースをデプロイする際に、分散トレースのリソースを作成することもできます。Service Mesh は分散トレースに Jaeger を使用します。



重要

- Jaeger は、FIPS 検証済みの暗号化モジュールを使用しません。
- Red Hat OpenShift Service Mesh 2.5 以降、Red Hat OpenShift 分散トレーシング Platform (Jaeger) は非推奨となり、今後のリリースで削除される予定です。Red Hat は、現在のリリースのライフサイクル中にこの機能のバグ修正とサポートを提供しますが、この機能は今後、機能拡張を受け取らず、削除されます。Red Hat OpenShift 分散トレーシング Platform (Jaeger) の代わりに、Red Hat OpenShift 分散トレーシング Platform を使用することもできます。

1.26.1. トレースの有効化および無効化

ServiceMeshControlPlane リソースでトレースタイプおよびサンプリングレートを指定して、分散トレースを有効にします。

デフォルトの all-in-one Jaeger パラメーター

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
  tracing:
    sampling: 100
    type: Jaeger
```

Red Hat OpenShift Service Mesh 2.6 では、トレースタイプ **Jaeger** は非推奨となり、デフォルトで無効になっています。

Red Hat OpenShift Service Mesh 2.5 以前では、トレースタイプ **Jaeger** がデフォルトで有効になっています。**Jaeger** トレースを無効にするには、**ServiceMeshControlPlane** リソースの **spec.tracing.type** パラメーターを **None** に設定します。

サンプリングレートは、Envoy プロキシがトレースを生成する頻度を決定します。サンプリングレートオプションを使用して、トレースシステムに報告される要求の割合を制御できます。この設定は、メッシュ内のトラフィックおよび収集するトレースデータ量に基づいて設定できます。**sampling** は 0.01% の増分を表すスケールされた整数として設定します。たとえば、値を **10** サンプル (0.1% トレース)、および **500** サンプル (5% トレース)、および **10000** サンプル (100% トレース) に設定します。



注記

SMCP サンプリング設定オプションは Envoy サンプリングレートを制御します。Jaeger トレースサンプリングレートを Jaeger カスタムリソースで設定します。

1.26.2. SMCP での Jaeger 設定の指定

Jaeger は、**ServiceMeshControlPlane** リソースの **addons** セクションで設定します。ただし、SMCP で設定可能な内容にはいくつかの制限があります。

SMCP が Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator に設定情報を渡すと、**allInOne**、**production**、または **streaming** の 3 つのデプロイメントストラテジーのいずれかがトリガーされます。

1.26.3. 分散トレースプラットフォームのデプロイ

分散トレーシングプラットフォーム (Jaeger) には、事前定義されたデプロイメントストラテジーがあります。Jaeger カスタムリソース (CR) ファイルでデプロイメントストラテジーを指定します。分散トレーシング Platform (Jaeger) のインスタンスを作成すると、Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator はこの設定ファイルを使用して、デプロイメントに必要なオブジェクトを作成します。

Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator は現在、次のデプロイメントストラテジーをサポートしています。

- **allInOne** (デフォルト): このストラテジーは、開発、テスト、およびデモを目的としたものであり、実稼働での使用を目的としたものではありません。主なバックエンドコンポーネントである Agent、Collector、および Query サービスはすべて、インメモリーストレージを使用するよ

うに (デフォルトで) 設定された単一の実行可能ファイルにパッケージ化されます。このデプロイメントストラテジーは、SMCP で設定できます。



注記

インメモリーストレージには永続性がありません。つまり、Jaeger インスタンスがシャットダウンするか、再起動するか、置き換えられると、トレースデータが失われます。各 Pod には独自のメモリーがあるため、インメモリーストレージはスケーリングできません。永続ストレージの場合は、デフォルトのストレージとして Elasticsearch を使用する **production** または **streaming** ストラテジーを使用する必要があります。

- **production**: production ストラテジーは、実稼働環境向けのストラテジーであり、トレースデータの長期の保存が重要となり、より拡張性および高可用性のあるアーキテクチャーも必要になります。そのため、バックエンドの各コンポーネントは別々にデプロイされます。エージェントは、インストルメント化されたアプリケーションのサイドカーとして挿入できます。Query および Collector サービスは、サポートされているストレージタイプ (現時点では Elasticsearch) で設定されます。これらの各コンポーネントの複数のインスタンスは、パフォーマンスと回復性を確保するために、必要に応じてプロビジョニングできます。このデプロイメントストラテジーを SMCP に設定できますが、完全にカスタマイズするには、Jaeger CR で設定を指定し、SMCP にリンクする必要があります。
- **streaming**: streaming ストラテジーは、Collector と Elasticsearch バックエンドストレージ間に配置されるストリーミング機能を提供することで、production ストラテジーを増強する目的で設計されています。これにより、負荷の高い状況でバックエンドストレージに加わる圧力を軽減し、他のトレース処理後の機能がストリーミングプラットフォーム ([AMQ Streams/ Kafka](#)) から直接リアルタイムのスパンデータを利用できるようにします。このデプロイメントストラテジーを SMCP で設定することはできません。Jaeger CR を設定し、SMCP へのリンクを設定する必要があります。



注記

streaming ストラテジーには、AMQ Streams 用の追加の Red Hat サブスクリプションが必要です。

1.26.3.1. デフォルトの分散トレーシングプラットフォーム (Jaeger) デプロイメント

Jaeger 設定オプションを指定しない場合、**ServiceMeshControlPlane** リソースはデフォルトで **allInOne** Jaeger デプロイメントストラテジーを使用します。デフォルトの **allInOne** デプロイメントストラテジーを使用する場合は、**spec.addons.jaeger.install.storage.type** を **Memory** に設定します。デフォルトを使用するか、**install** で追加設定オプションを許可できます。

コントロールプレーンのデフォルト Jaeger パラメーター (Memory)

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
  tracing:
    sampling: 10000
    type: Jaeger
  addons:
```

```
jaeger:
  name: jaeger
  install:
    storage:
      type: Memory
```

1.26.3.2. 実稼働環境向け分散トレーシングプラットフォーム (Jaeger) のデプロイメント (最小限)

production デプロイメントストラテジーのデフォルト設定を使用するには、**spec.addons.jaeger.install.storage.type** を **Elasticsearch** に設定し、**install** で追加設定オプションを指定します。SMCP は Elasticsearch リソースおよびイメージ名の設定のみをサポートすることに注意してください。

コントロールプレーンのデフォルト Jaeger パラメーター (Elasticsearch)

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
  tracing:
    sampling: 10000
    type: Jaeger
  addons:
    jaeger:
      name: jaeger #name of Jaeger CR
      install:
        storage:
          type: Elasticsearch
        ingress:
          enabled: true
      runtime:
        components:
          tracing.jaeger.elasticsearch: # only supports resources and image name
        container:
          resources: {}
```

1.26.3.3. 実稼働環境向け分散トレーシングプラットフォーム (Jaeger) のデプロイメント (完全カスタマイズ)

SMCP は最小限の Elasticsearch パラメーターのみをサポートします。実稼働環境を完全にカスタマイズし、すべての Elasticsearch 設定パラメーターにアクセスするには、Jaeger カスタムリソース (CR) を使用して Jaeger を設定します。

または、Jaeger インスタンスを作成および設定し、**spec.addons.jaeger.name** を Jaeger インスタンスの名前 (この例では **MyJaegerInstance**) に設定できます。

Jaeger production CR がリンクされたコントロールプレーン

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
```

```

name: basic
spec:
  version: v2.6
  tracing:
    sampling: 1000
    type: Jaeger
  addons:
    jaeger:
      name: MyJaegerInstance #name of Jaeger CR
      install:
        storage:
          type: Elasticsearch
        ingress:
          enabled: true

```

1.26.3.4. Jaeger デプロイメントのストリーミング

streaming デプロイメントストラテジーを使用するには、まず Jaeger インスタンスを作成および設定してから、**spec.addons.jaeger.name** を Jaeger インスタンスの名前 (この例では **MyJaegerInstance**) に設定します。

リンクされた Jaeger ストリーミング CR を使用したコントロールプレーン

```

apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.6
  tracing:
    sampling: 1000
    type: Jaeger
  addons:
    jaeger:
      name: MyJaegerInstance #name of Jaeger CR

```

1.26.4. Jaeger カスタムリソースでの Jaeger 設定の指定

ServiceMeshControlPlane (SMCP) リソースではなく Jaeger カスタムリソース (CR) に Jaeger を設定し、Jaeger デプロイメントを完全にカスタマイズできます。この設定は SMCP の外部に指定されているため、"外部 Jaeger" と呼ばれることもあります。



注記

SMCP と Jaeger CR を同じ namespace にデプロイする必要があります。たとえば、**istio-system** です。

スタンドアロンの Jaeger インスタンスを設定し、デプロイしてから、Jaeger リソースの **name** を、SMCP リソースの **spec.addons.jaeger.name** の値として指定できます。**name** の値に一致する Jaeger CR が存在する場合、Service Mesh コントロールプレーンは既存のインストールを使用します。この方法では、Jaeger 設定を完全にカスタマイズできます。

1.26.4.1. デプロイメントのベストプラクティス

- Red Hat OpenShift 分散トレーシング Platform インスタンス名は一意である必要があります。複数の Red Hat OpenShift 分散トレーシング Platform (Jaeger) インスタンスが必要で、サイドカー注入エージェントを使用している場合は、Red Hat OpenShift 分散トレーシング Platform (Jaeger) インスタンスに一意の名前を付け、注入アノテーションでトレースデータのレポート先となる Red Hat OpenShift 分散トレーシング Platform (Jaeger) インスタンス名を明示的に指定する必要があります。
- マルチテナント実装があり、テナントが名前空間によって分離されている場合は、各テナント名前空間に Red Hat OpenShift 分散トレーシング Platform (Jaeger) インスタンスをデプロイします。

1.26.4.2. Service Mesh の分散トレースセキュリティの設定

分散トレーシング Platform (Jaeger) は、デフォルトの認証に OAuth を使用します。ただし、Red Hat OpenShift Service Mesh は、Grafana、Kiali、分散トレーシング Platform (Jaeger) などの依存サービス間の通信を容易にするために、**htpasswd** と呼ばれるシークレットを使用します。**ServiceMeshControlPlane** で分散トレーシング Platform (Jaeger) を設定すると、Service Mesh は **htpasswd** を使用するようにセキュリティ設定を自動的に設定します。

Jaeger カスタムリソースで分散トレーシング Platform (Jaeger) 設定を指定する場合は、**htpasswd** 設定を手動で設定し、Kiali が通信できるように **htpasswd** シークレットが Jaeger インスタンスにマウントされていることを確認する必要があります。

1.26.4.2.1. Web コンソールからの Service Mesh の分散トレースセキュリティの設定

Jaeger リソースを変更して、Web コンソールで Service Mesh で使用するための分散トレーシング Platform (Jaeger) セキュリティを設定できます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。(Red Hat OpenShift Dedicated を使用する場合は **dedicated-admin** ロールがあるアカウント。
- Red Hat OpenShift Service Mesh Operator がインストールされている。
- クラスターにデプロイされた **ServiceMeshControlPlane**。
- Red Hat OpenShift Service on AWS Web コンソールにアクセスできる。

手順

1. **cluster-admin** ロールを持つユーザーとして Red Hat OpenShift Service on AWS 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、リストから **ServiceMeshControlPlane** リソースがデプロイされているプロジェクト (例: **istio-system**) を選択します。
4. **Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator** をクリックします。
5. **Operator** の詳細 ページで、**Jaeger** タブをクリックします。
6. Jaeger インスタンスの名前をクリックします。
7. Jaeger の詳細 ページで、**YAML** タブをクリックして設定を変更します。

8. 次の例に示すように、**Jaeger** カスタムリソースファイルを編集して、**htpasswd** 設定を追加します。

- **spec.ingress.openshift.htpasswdFile**
- **spec.volumes**
- **spec.volumeMounts**

htpasswd 設定を示す Jaeger リソースの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
spec:
  ingress:
    enabled: true
    openshift:
      htpasswdFile: /etc/proxy/htpasswd/auth
      sar: '{"namespace": "istio-system", "resource": "pods", "verb": "get"}'
    options: {}
    resources: {}
    security: oauth-proxy
  volumes:
    - name: secret-htpasswd
      secret:
        secretName: htpasswd
    - configMap:
        defaultMode: 420
        items:
          - key: ca-bundle.crt
            path: tls-ca-bundle.pem
          name: trusted-ca-bundle
          optional: true
          name: trusted-ca-bundle
  volumeMounts:
    - mountPath: /etc/proxy/htpasswd
      name: secret-htpasswd
    - mountPath: /etc/pki/ca-trust/extracted/pem/
      name: trusted-ca-bundle
      readOnly: true
# ...
```

9. **Save** をクリックします。

1.26.4.2.2. コマンドラインからの Service Mesh の分散トレースセキュリティの設定

OpenShift CLI (**oc**) を実行して、コマンドラインから Service Mesh で使用するために、Jaeger リソースを変更して分散トレーシング Platform (Jaeger) のセキュリティを設定できます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。(Red Hat OpenShift Dedicated を使用する場合は **dedicated-admin** ロールがあるアカウント。
- Red Hat OpenShift Service Mesh Operator がインストールされている。

- クラスターにデプロイされた **ServiceMeshControlPlane**。
- Red Hat OpenShift Service on AWS バージョンに一致する OpenShift CLI (**oc**) にアクセスできる。

手順

1. 以下のコマンドを実行して、**cluster-admin** ロールが割り当てられたユーザーとして OpenShift CLI (**oc**) にログインします。(Red Hat OpenShift Dedicated を使用する場合は **dedicated-admin** ロールがあるアカウント。

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

2. 次のコマンドを入力して、コントロールプレーンをインストールしたプロジェクト (**istio-system** など) に変更します。

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

3. 次のコマンドを実行して、Jaeger カスタムリソースファイルを編集します。

```
$ oc edit -n openshift-distributed-tracing -f jaeger.yaml
```

4. 次の例に示すように、**Jaeger** カスタムリソースファイルを編集して、**htpasswd** 設定を追加します。

- **spec.ingress.openshift.htpasswdFile**
- **spec.volumes**
- **spec.volumeMounts**

htpasswd 設定を示す Jaeger リソースの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
spec:
  ingress:
    enabled: true
    openshift:
      htpasswdFile: /etc/proxy/htpasswd/auth
      sar: '{"namespace": "istio-system", "resource": "pods", "verb": "get"}'
    options: {}
    resources: {}
    security: oauth-proxy
  volumes:
    - name: secret-htpasswd
      secret:
        secretName: htpasswd
    - configMap:
        defaultMode: 420
        items:
          - key: ca-bundle.crt
            path: tls-ca-bundle.pem
        name: trusted-ca-bundle
        optional: true
```

```

    name: trusted-ca-bundle
  volumeMounts:
  - mountPath: /etc/proxy/htpasswd
    name: secret-htpasswd
  - mountPath: /etc/pki/ca-trust/extracted/pem/
    name: trusted-ca-bundle
    readOnly: true

```

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

```
$ oc get pods -n openshift-distributed-tracing
```

1.26.4.3. 分散トレースのデフォルト設定オプション

Jaeger カスタムリソース (CR) は、分散トレーシング Platform (Jaeger) リソースを作成するときに使用するアーキテクチャーと設定を定義します。これらのパラメーターを変更して、分散トレーシング Platform (Jaeger) の実装をビジネスニーズに合わせてカスタマイズできます。

Jaeger CR の汎用 YAML の例

```

apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: name
spec:
  strategy: <deployment_strategy>
  allInOne:
    options: {}
    resources: {}
  agent:
    options: {}
    resources: {}
  collector:
    options: {}
    resources: {}
  sampling:
    options: {}
  storage:
    type:
    options: {}
  query:
    options: {}
    resources: {}
  ingester:
    options: {}
    resources: {}
  options: {}

```

表1.39 Jaeger パラメーター

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

パラメーター	説明	値	デフォルト値
apiVersion:	オブジェクトの作成時に使用する API バージョン。	jaegertracing.io/v1	jaegertracing.io/v1
kind:	作成する Kubernetes オブジェクトの種類を定義します。	jaeger	
metadata:	name 文字列、 UID 、およびオプションの namespace などのオブジェクトを一意に特定するのに役立つデータ。		Red Hat OpenShift Service on AWS は UID を自動的に生成し、オブジェクトが作成されるプロジェクトの名前で namespace を完了します。
name:	オブジェクトの名前。	分散トレーシング Platform (Jaeger) インスタンスの名前。	jaeger-all-in-one-inmemory
spec:	作成するオブジェクトの仕様。	分散トレーシング Platform (Jaeger) インスタンスのすべての設定パラメーターが含まれます。すべての Jaeger コンポーネントの共通定義が必要な場合、これは spec ノードで定義されます。定義が個々のコンポーネントに関連する場合は、 spec/<component> ノードに置かれます。	該当なし
strategy:	Jaeger デプロイメント戦略	allInOne 、 production 、または streaming	allInOne
allInOne:	allInOne イメージは Agent、Collector、Query、Ingester、および Jaeger UI を単一 Pod にデプロイするため、このデプロイメントの設定は、コンポーネント設定を allInOne パラメーターの下でネストする必要があります。		

パラメーター	説明	値	デフォルト値
agent:	Agent を定義する設定オプション。		
collector:	Jaeger Collector を定義する設定オプション。		
sampling:	トレース用のサンプリングストラテジーを定義する設定オプション。		
storage:	ストレージを定義する設定オプション。すべてのストレージ関連のオプションは、 allInOne または他のコンポーネントオプションではなく、 storage に配置される必要があります。		
query:	Query サービスを定義する設定オプション。		
ingester:	Ingestor サービスを定義する設定オプション。		

次の YAML の例は、デフォルト設定を使用して Red Hat OpenShift 分散トレーシング Platform (Jaeger) デプロイメントを作成するために必要な最小限のものです。

最小限必要な dist-tracing-all-in-one.yaml の例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-all-in-one-inmemory
```

1.26.4.4. Jaeger Collector 設定オプション

Jaeger Collector は、トレーサーによってキャプチャーされたスパンを受信し、**production** ストラテジーを使用する場合はそれらを永続 Elasticsearch ストレージに書き込み、**streaming** ストラテジーを使用する場合は AMQ Streams に書き込むコンポーネントです。

Collector はステートレスであるため、Jaeger Collector のインスタンスの多くは並行して実行できます。Elasticsearch クラスターの場所を除き、Collector では設定がほとんど必要ありません。

表1.40 Operator によって使用される Jaeger Collector パラメーターを定義するためのパラメーター

パラメーター	説明	値
<code>collector: replicas:</code>	作成する Collector レプリカの数 を指定します。	整数 (例: 5)。

表1.41 Collector に渡される設定パラメーター

パラメーター	説明	値
<code>spec: collector: options: {}</code>	Jaeger Collector を定義する設定 オプション。	
<code>options: collector: num-workers:</code>	キューからプルするワーカーの 数。	整数 (例: 50)。
<code>options: collector: queue-size:</code>	Collector キューのサイズ。	整数 (例: 2000)。
<code>options: kafka: producer: topic: jaeger-spans</code>	topic パラメーターは、Collector によってメッセージを生成するた めに使用され、Ingester によって メッセージを消費するために使用 される Kafka 設定を特定します。	プロデューサーのラベル。
<code>options: kafka: producer: brokers: my-cluster- kafka-brokers.kafka:9092</code>	メッセージを生成するために Collector によって使用される Kafka 設定を特定します。ブロー カーが指定されておらず、AMQ Streams 1.4.0+ がインストールさ れている場合、Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator が Kafka をセルフプロビジョニング します。	
<code>options: log-level:</code>	Collector のロギングレベル。	使用できる値 は、 debug 、 info 、 warn 、 erro r 、 fatal 、 panic です。

パラメーター	説明	値
<pre>options: otlp: enabled: true grpc: host-port: 4317 max-connection-age: 0s max-connection-age- grace: 0s max-message-size: 4194304 tls: enabled: false cert: /path/to/cert.crt cipher-suites: "TLS_AES_256_GCM_SHA 384,TLS_CHACHA20_POL Y1305_SHA256" client-ca: /path/to/cert.ca reload-interval: 0s min-version: 1.2 max-version: 1.3</pre>	OTLP/gRPC を受け入れるには、 otlp を明示的に有効にします。他はすべて任意のオプションです。	

パラメーター	説明	値
<pre> options: otlp: enabled: true http: cors: allowed-headers: [<header-name>[, <header- name>]*] allowed-origins: * host-port: 4318 max-connection-age: 0s max-connection-age- grace: 0s max-message-size: 4194304 read-timeout: 0s read-header-timeout: 2s idle-timeout: 0s tls: enabled: false cert: /path/to/cert.crt cipher-suites: "TLS_AES_256_GCM_SHA 384,TLS_CHACHA20_POL Y1305_SHA256" client-ca: /path/to/cert.ca reload-interval: 0s min-version: 1.2 max-version: 1.3 </pre>	OTLP/HTTP を受け入れるには、 otlp を明示的に有効にします。他はすべて任意のオプションです。	

1.26.4.5. 分散トレースのサンプリング設定オプション

Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator を使用すると、リモートサンプラーを使用するように設定されたトレーサーに提供されるサンプリングストラテジーを定義できます。

すべてのトレースが生成される間に、それらの一部のみがサンプリングされます。トレースをサンプリングすると、追加の処理や保存のためにトレースにマークが付けられます。



注記

これは、トレースがサンプリングの意思決定が行われる際に Envoy プロキシによって開始されている場合に関連がありません。Jaeger サンプリングの意思決定は、トレースがクライアントを使用してアプリケーションによって開始される場合にのみ関連します。

サービスがトレースコンテキストが含まれていない要求を受信すると、クライアントは新しいトレースを開始し、これにランダムなトレース ID を割り当て、現在インストールされているサンプリングストラテジーに基づいてサンプリングの意思決定を行います。サンプリングの意思決定はトレース内の後続のすべての要求に伝播され、他のサービスが再度サンプリングの意思決定を行わないようにします。

分散トレーシング Platform (Jaeger) ライブラリーは、次のサンプラーをサポートしています。

- **Probabilistic:** サンプラーは、**sampling.param** プロパティの値と等しいサンプリングの確率で、ランダムなサンプリングの意思決定を行います。たとえば、**sampling.param=0.1** を使用した場合は、約 10 のうち 1 トレースがサンプリングされます。
- **Rate Limiting:** サンプラーは、リーキーバケット (leaky bucket) レートリミッターを使用して、トレースが一定のレートでサンプリングされるようにします。たとえば、**sampling.param=2.0** を使用した場合は、1 秒あたり 2 トレースの割合で要求がサンプリングされます。

表1.42 Jaeger サンプリングのオプション

パラメーター	説明	値	デフォルト値
spec: sampling: options: {} default_strategy: service_strategy:	トレース用のサンプリングストラテジーを定義する設定オプション。		設定を指定しない場合、Collector はすべてのサービスの確率 0.001 (0.1%) のデフォルトの確率的なサンプリングポリシーを返します。
default_strategy: type: service_strategy: type:	使用するサンプリングストラテジー。上記の説明を参照してください。	有効な値は probabilistic 、および ratelimiting です。	probabilistic
default_strategy: param: service_strategy: param:	選択したサンプリングストラテジーのパラメーター	10 進値および整数値 (0、.1、1、10)	1

この例では、トレースインスタンスをサンプリングする確率が 50% の確率的なデフォルトサンプリングストラテジーを定義します。

確率的なサンプリングの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: with-sampling
spec:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 0.5
  service_strategies:
    - service: alpha
      type: probabilistic
```

```

param: 0.8
operation_strategies:
  - operation: op1
    type: probabilistic
    param: 0.2
  - operation: op2
    type: probabilistic
    param: 0.4
- service: beta
  type: ratelimiting
  param: 5

```

ユーザー指定の設定がない場合、分散トレーシング Platform (Jaeger) は次の設定を使用します。

デフォルトのサンプリング

```

spec:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 1

```

1.26.4.6. 分散トレースのストレージ設定オプション

spec.storage の下で Collector、Ingester、および Query サービスのストレージを設定します。これらの各コンポーネントの複数のインスタンスは、パフォーマンスと回復性を確保するために、必要に応じてプロビジョニングできます。

表1.43 Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator が分散トレーシングストレージを定義するために使用する一般的なストレージパラメーター

パラメーター	説明	値	デフォルト値
spec: storage: type:	デプロイメントに使用するストレージのタイプ。	memory または elasticsearch メモリーストレージは、Pod がシャットダウンした場合にデータが永続化されないため、開発、テスト、デモ、および概念検証用の環境にのみに適しています。実稼働環境では、分散トレーシング Platform (Jaeger) が永続ストレージ用の Elasticsearch をサポートします。	memory
storage: secretname:	シークレットの名前 (例: tracing-secret)。		該当なし

パラメーター	説明	値	デフォルト値
<code>storage: options: {}</code>	ストレージを定義する設定オプション。		

表1.44 Elasticsearch インデックスクリーナーのパラメーター

パラメーター	説明	値	デフォルト値
<code>storage: esIndexCleaner: enabled:</code>	Elasticsearch ストレージを使用する場合は、デフォルトでジョブが作成され、古いトレースをインデックスからクリーンアップします。このパラメーターは、インデックスクリーナージョブを有効または無効にします。	true/ false	true
<code>storage: esIndexCleaner: numberOfDays:</code>	インデックスの削除を待機する日数。	整数値	7
<code>storage: esIndexCleaner: schedule:</code>	Elasticsearch インデックスを消去する頻度に関するスケジュールを定義します。	cron 式	"55 23 * * *"

1.26.4.6.1. Elasticsearch インスタンスの自動プロビジョニング

Jaeger カスタムリソースをデプロイすると、Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator は OpenShift Elasticsearch Operator を使用して、カスタムリソースファイルのストレージセクションで指定された設定に基づいて Elasticsearch クラスターを作成します。Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator は、次の設定が設定されている場合、Elasticsearch をプロビジョニングします。

- **spec.storage:type** は **elasticsearch** に設定されている
- **spec.storage.elasticsearch.doNotProvision** は **false** に設定されている
- **spec.storage.options.es.server-urls** が定義されていない。つまり、OpenShift Elasticsearch Operator によってプロビジョニングされていない Elasticsearch インスタンスへの接続がない。

Elasticsearch をプロビジョニングする場合、Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator は、Elasticsearch カスタムリソース名を Jaeger カスタムリソースの **spec.storage.elasticsearch.name** の値に設定します。**spec.storage.elasticsearch.name** に値を指定しない場合、Operator は **elasticsearch** を使用します。

制約

- 名前空間ごとに、セルフプロビジョニングされた Elasticsearch インスタンスを備えた分散トレーシングプラットフォーム (Jaeger) を1つだけ持つことができます。Elasticsearch クラスターは、単一の分散トレーシング Platform (Jaeger) インスタンス専用となります。
- namespace ごとに1つの Elasticsearch のみを使用できます。



注記

OpenShift ロギングの一部として Elasticsearch がすでにインストールされている場合、Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator はインストールされた OpenShift Elasticsearch Operator を使用してストレージをプロビジョニングできます。

以下の設定パラメーターは、セルフプロビジョニングされた Elasticsearch インスタンス用です。セルフプロビジョニングされた Elasticsearch インスタンスとは、OpenShift Elasticsearch Operator を使用して Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator によって作成されたインスタンスです。セルフプロビジョニングされた Elasticsearch の設定オプションは、設定ファイルの **spec:storage:elasticsearch** の下で指定します。

表1.45 Elasticsearch リソース設定パラメーター

パラメーター	説明	値	デフォルト値
elasticsearch: properties: doNotProvision:	Elasticsearch インスタンスを Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator によってプロビジョニングするかどうかを指定するために使用します。	true/false	true
elasticsearch: properties: name:	Elasticsearch インスタンスの名前。Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator は、このパラメーターで指定された Elasticsearch インスタンスを使用して Elasticsearch に接続します。	string	elasticsearch
elasticsearch: nodeCount:	Elasticsearch ノードの数。高可用性を確保するには、少なくとも3つのノードを使用します。 “スプリットブレイン”の問題が生じる可能性があるため、2つのノードを使用しないでください。	整数値。例: 概念実証用 = 1、最小デプロイメント = 3	3

パラメーター	説明	値	デフォルト値
<code>elasticsearch: resources: requests: cpu:</code>	ご使用の環境設定に基づく、要求に対する中央処理単位の数。	コアまたはミリコアで指定されます (例: 200m、0.5、1)。例: 概念実証用 = 500m、最小デプロイメント = 1	1
<code>elasticsearch: resources: requests: memory:</code>	ご使用の環境設定に基づく、要求に使用できるメモリー。	バイト単位で指定します (例: 200Ki、50Mi、5Gi)。例: 概念実証用 = 1Gi、最小デプロイメント = 16Gi*	16Gi
<code>elasticsearch: resources: limits: cpu:</code>	ご使用の環境設定に基づく、中央処理単位数の制限。	コアまたはミリコアで指定されます (例: 200m、0.5、1)。例: 概念実証用 = 500m、最小デプロイメント = 1	
<code>elasticsearch: resources: limits: memory:</code>	ご使用の環境設定に基づく、利用可能なメモリー制限。	バイト単位で指定します (例: 200Ki、50Mi、5Gi)。例: 概念実証用 = 1Gi、最小デプロイメント = 16Gi*	
<code>elasticsearch: redundancyPolicy:</code>	データレプリケーションポリシーは、Elasticsearch シャードをクラスター内のデータノードにレプリケートする方法を定義します。指定されていない場合は、Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator がノード数に基づいて最適なレプリケーションを自動的に決定します。	ZeroRedundancy (レプリカシャードなし)、 SingleRedundancy (レプリカシャード1つ)、 MultipleRedundancy (各インデックスはデータノードの半分に分散される)、 FullRedundancy (各インデックスはクラスター内のすべてのデータノードに完全にレプリケートされます)	

パラメーター	説明	値	デフォルト値
elasticsearch: useCertManagement:	分散トレーシング Platform (Jaeger) が OpenShift Elasticsearch Operator の証明書管理機能を使用するかどうかを指定するために使用します。この機能は、Red Hat OpenShift Service on AWS 4.7 の {logging-title} 5.2 に追加されました。新しい Jaeger デプロイメントでは推奨される設定です。	true/false	true

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は、デフォルトで各 Pod に割り当てる設定を 16 Gi 未満にすることはできず、Pod ごとに最大 64 Gi を割り当てる必要があります。

実稼働ストレージの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 3
    resources:
      requests:
        cpu: 1
        memory: 16Gi
    limits:
      memory: 16Gi
```

永続ストレージを含むストレージの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 1
      storage: 1
      storageClassName: gp2
```

```

size: 5Gi
resources:
  requests:
    cpu: 200m
    memory: 4Gi
  limits:
    memory: 4Gi
  redundancyPolicy: ZeroRedundancy

```

- 1 永続ストレージの設定。この場合、AWS **gp2** のサイズは **5Gi** です。値が指定されていない場合、分散トレーシング Platform (Jaeger) は **emptyDir** を使用します。OpenShift Elasticsearch Operator は、分散トレーシング Platform (Jaeger) インスタンスで削除されない **PersistentVolumeClaim** と **PersistentVolume** をプロビジョニングします。同じ名前と名前空間を持つ分散トレーシング Platform (Jaeger) インスタンスを作成すると、同じボリュームをマウントできます。

1.26.4.6.2. 既存の Elasticsearch インスタンスへの接続

分散トレーシング Platform では、既存の Elasticsearch クラスターをストレージとして使用できます。既存の Elasticsearch クラスター (外部 Elasticsearch インスタンスとも呼ばれます) は、Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator または OpenShift Elasticsearch Operator によってインストールされなかったインスタンスです。

Jaeger カスタムリソースをデプロイするときに、次の設定が設定されている場合は、Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator は Elasticsearch をプロビジョニングしません。

- **spec.storage.elasticsearch.doNotProvision** が **true** に設定されている
- **spec.storage.options.es.server-urls** に値がある
- **spec.storage.elasticsearch.name** に値がある場合、または Elasticsearch インスタンス名が **elasticsearch** の場合。

Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator は、**spec.storage.elasticsearch.name** で指定された Elasticsearch インスタンスを使用して Elasticsearch に接続します。

制約

- Red Hat OpenShift Service on AWS ログ Elasticsearch インスタンスを分散トレーシング Platform (Jaeger) で共有または再利用することはできません。Elasticsearch クラスターは、単一の分散トレーシング Platform (Jaeger) インスタンス専用となります。

以下の設定パラメーターは、外部 Elasticsearch インスタンスとして知られる、既存の Elasticsearch インスタンス向けです。この場合は、カスタムリソースファイルの **spec:storage:options:es** で、Elasticsearch の設定オプションを指定します。

表1.46 汎用 ES 設定パラメーター

パラメーター	説明	値	デフォルト値
es: server-urls:	Elasticsearch インスタンスの URL。	Elasticsearch サーバーの完全修飾ドメイン名。	<a href="http://elasticsearch.<namespace>.svc:9200">http://elasticsearch.<namespace>.svc:9200
es: max-doc-count:	Elasticsearch クエリーから返す最大ドキュメント数。これは集約にも適用されます。 es.max-doc-count と es.max-num-spans の両方を設定する場合は、Elasticsearch は 2 つの内の小さい方の値を使用します。		10000
es: max-num-spans:	[非推奨 : 今後のリリースで削除されます。代わりに es.max-doc-count を使用してください。] Elasticsearch のクエリーごとに、1 度にフェッチするスパンの最大数。 es.max-num-spans と es.max-doc-count の両方を設定する場合、Elasticsearch は 2 つの内の小さい方の値を使用します。		10000
es: max-span-age:	Elasticsearch のスパンの最大ルックバック。		72h0m0s
es: sniffer:	Elasticsearch のスニファースタートアップ設定。クライアントはスニフリングプロセスを使用してすべてのノードを自動的に検索します。デフォルトでは無効になっています。	true/ false	false

パラメーター	説明	値	デフォルト値
es: sniffer-tls-enabled:	Elasticsearch クラスターに対してスニッフィングする際に TLS を有効にするためのオプション。クライアントはスニッフィングプロセスを使用してすべてのノードを自動的に検索します。デフォルトでは無効になっています。	true/ false	false
es: timeout:	クエリーに使用されるタイムアウト。ゼロに設定するとタイムアウトはありません。		0s
es: username:	Elasticsearch で必要なユーザー名。Basic 認証は、指定されている場合に CA も読み込みます。 es.password も参照してください。		
es: password:	Elasticsearch で必要なパスワード。 es.username も参照してください。		
es: version:	主要な Elasticsearch バージョン。指定されていない場合、値は Elasticsearch から自動検出されます。		0

表1.47 ES データレプリケーションパラメーター

パラメーター	説明	値	デフォルト値
es: num-replicas:	Elasticsearch のインデックスごとのレプリカ数。		1
es: num-shards:	Elasticsearch のインデックスごとのシャード数。		5

表1.48 ES インデックス設定パラメーター

パラメーター	説明	値	デフォルト値
<code>es: create-index-templates:</code>	true に設定されている場合は、アプリケーションの起動時にインデックステンプレートを自動的に作成します。テンプレートが手動でインストールされる場合は、 false に設定されます。	true/ false	true
<code>es: index-prefix:</code>	分散トレーシング Platform (Jaeger) インデックスのオプションの接頭辞。たとえば、これを "production" に設定すると、"production-tracing-*" という名前のインデックスが作成されます。		

表1.49 ES バルクプロセッサ設定パラメーター

パラメーター	説明	値	デフォルト値
<code>es: bulk: actions:</code>	バルクプロセッサがディスクへの更新のコミットを決定する前にキューに追加できる要求の数。		1000
<code>es: bulk: flush-interval:</code>	time.Duration: この後に、他のしきい値に関係なく一括要求がコミットされます。バルクプロセッサのフラッシュ間隔を無効にするには、これをゼロに設定します。		200ms
<code>es: bulk: size:</code>	バルクプロセッサがディスクへの更新をコミットするまでに一括要求が発生する可能性のあるバイト数。		5000000
<code>es: bulk: workers:</code>	一括要求を受信し、Elasticsearch にコミットできるワーカーの数。		1

表1.50 ES TLS 設定パラメーター

パラメーター	説明	値	デフォルト値
es: tls: ca:	リモートサーバーの検証に使用される TLS 認証局 (CA) ファイルへのパス。		デフォルトではシステムトラストストアを使用します。
es: tls: cert:	リモートサーバーに対するこのプロセスの特定に使用される TLS 証明書ファイルへのパス。		
es: tls: enabled:	リモートサーバーと通信する際に、トランスポート層セキュリティ (TLS) を有効にします。デフォルトでは無効になっています。	true/ false	false
es: tls: key:	リモートサーバーに対するこのプロセスの特定に使用される TLS 秘密鍵ファイルへのパス。		
es: tls: server-name:	リモートサーバーの証明書の予想される TLS サーバー名を上書きします。		
es: token-file:	ベアラートークンが含まれるファイルへのパス。このフラグは、指定されている場合は認証局 (CA) ファイルも読み込みます。		

表1.51 ES アーカイブ設定パラメーター

パラメーター	説明	値	デフォルト値
es-archive: bulk: actions:	バルクプロセッサがディスクへの更新のコミットを決定する前にキューに追加できる要求の数。		0

パラメーター	説明	値	デフォルト値
es-archive: bulk: flush-interval:	time.Duration: この後に、他のしきい値に関係なく一括要求がコミットされます。バルクプロセッサのフラッシュ間隔を無効にするには、これをゼロに設定します。		0s
es-archive: bulk: size:	バルクプロセッサがディスクへの更新をコミットするまでに一括要求が発生する可能性のあるバイト数。		0
es-archive: bulk: workers:	一括要求を受信し、Elasticsearch にコミットできるワーカーの数。		0
es-archive: create-index-templates:	true に設定されている場合は、アプリケーションの起動時にインデックステンプレートを自動的に作成します。テンプレートが手動でインストールされる場合は、 false に設定されます。	true/ false	false
es-archive: enabled:	追加ストレージを有効にします。	true/ false	false
es-archive: index-prefix:	分散トレーシング Platform (Jaeger) インデックスのオプションの接頭辞。たとえば、これを "production" に設定すると、"production-tracing-*" という名前のインデックスが作成されます。		
es-archive: max-doc-count:	Elasticsearch クエリーから返す最大ドキュメント数。これは集約にも適用されます。		0

パラメーター	説明	値	デフォルト値
es-archive: max-num-spans:	[非推奨 : 今後のリリースで削除されます。代わりに es-archive.max-doc-count を使用してください。] Elasticsearch のクエリーごとに、1度にフェッチするスパンの最大数。		0
es-archive: max-span-age:	Elasticsearch のスパンの最大ルックバック。		0s
es-archive: num-replicas:	Elasticsearch のインデックスごとのレプリカ数。		0
es-archive: num-shards:	Elasticsearch のインデックスごとのシャード数。		0
es-archive: password:	Elasticsearch で必要なパスワード。 es.username も参照してください。		
es-archive: server-urls:	Elasticsearch サーバーのコンマ区切りの一覧。完全修飾 URL(例: http://localhost:9200)として指定される必要があります。		
es-archive: sniffer:	Elasticsearch のスニファースettings。クライアントはスニフリングプロセスを使用してすべてのノードを自動的に検索します。デフォルトでは無効になっています。	true/ false	false

パラメーター	説明	値	デフォルト値
es-archive: sniffer-tls- enabled:	Elasticsearch クラスターに対してスニッフィングする際に TLS を有効にするためのオプション。クライアントはスニッフィングプロセスを使用してすべてのノードを自動的に検索します。デフォルトでは無効になっています。	true/ false	false
es-archive: timeout:	クエリーに使用されるタイムアウト。ゼロに設定するとタイムアウトはありません。		0s
es-archive: tls: ca:	リモートサーバーの検証に使用される TLS 認証局 (CA) ファイルへのパス。		デフォルトではシステムトラストストアを使用します。
es-archive: tls: cert:	リモートサーバーに対するこのプロセスの特定に使用される TLS 証明書ファイルへのパス。		
es-archive: tls: enabled:	リモートサーバーと通信する際に、トランスポート層セキュリティ (TLS) を有効にします。デフォルトでは無効になっています。	true/ false	false
es-archive: tls: key:	リモートサーバーに対するこのプロセスの特定に使用される TLS 秘密鍵ファイルへのパス。		
es-archive: tls: server-name:	リモートサーバーの証明書の予想される TLS サーバー名を上書きします。		

パラメーター	説明	値	デフォルト値
<code>es-archive: token-file:</code>	ベアラートークンが含まれるファイルへのパス。このフラグは、指定されている場合は認証局 (CA) ファイルも読み込みます。		
<code>es-archive: username:</code>	Elasticsearch で必要なユーザー名。Basic 認証は、指定されている場合に CA も読み込みます。 es-archive.password も参照してください。		
<code>es-archive: version:</code>	主要な Elasticsearch バージョン。指定されていない場合、値は Elasticsearch から自動検出されます。		0

ボリュームマウントを含むストレージの例

```

apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200
        index-prefix: my-prefix
        tls:
          ca: /es/certificates/ca.crt
      secretName: tracing-secret
  volumeMounts:
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
  volumes:
    - name: certificates
      secret:
        secretName: quickstart-es-http-certs-public

```

以下の例は、ボリュームからマウントされる TLS CA 証明書およびシークレットに保存されるユーザー/パスワードを使用して外部 Elasticsearch クラスターを使用する Jaeger CR を示しています。

外部 Elasticsearch の例:

```

apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200 ❶
        index-prefix: my-prefix
        tls: ❷
          ca: /es/certificates/ca.crt
        secretName: tracing-secret ❸
  volumeMounts: ❹
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
  volumes:
    - name: certificates
      secret:
        secretName: quickstart-es-http-certs-public

```

- ❶ デフォルト namespace で実行されている Elasticsearch サービスへの URL。
- ❷ TLS 設定。この場合は、CA 証明書のみを使用できますが、相互 TLS を使用する場合に es.tls.key および es.tls.cert を含めることもできます。
- ❸ 環境変数 ES_PASSWORD および ES_USERNAME を定義するシークレット。kubectl create secret generic tracing-secret --from-literal=ES_PASSWORD=changeme --from-literal=ES_USERNAME=elastic により作成されます
- ❹ すべてのストレージコンポーネントにマウントされるボリュームのマウントとボリューム。

1.26.4.7. クエリー設定オプション

Query とは、ストレージからトレースを取得し、ユーザーインターフェイスをホストしてそれらを表示するサービスです。

表1.52 Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator がクエリーを定義するために使用するパラメーター

パラメーター	説明	値	デフォルト値
spec: query: replicas:	作成する Query レプリカ カの数指定します。	整数 (例: 2)	

表1.53 Query に渡される設定パラメーター

パラメーター	説明	値	デフォルト値
spec: query: options: {}	Query サービスを定義する設定オプション。		
options: log-level:	Query のロギングレベル。	使用できる値は、 debug 、 info 、 warn 、 error 、 fatal 、 panic です。	
options: query: base-path:	すべての jaeger-query HTTP ルートのベースパスは、root 以外の値に設定できます。たとえば、 /jaeger ではすべての UI URL が /jaeger で開始するようになります。これは、リバースプロキシの背後で jaeger-query を実行する場合に役立ちます。	/<path>	

Query 設定の例

```

apiVersion: jaegertracing.io/v1
kind: "Jaeger"
metadata:
  name: "my-jaeger"
spec:
  strategy: allInOne
  allInOne:
    options:
      log-level: debug
      query:
        base-path: /jaeger

```

1.26.4.8. Ingestor 設定オプション

Ingestor は、Kafka トピックから読み取り、Elasticsearch ストレージバックエンドに書き込むサービスです。**allInOne** または **production** デプロイメントストラテジーを使用している場合は、Ingestor サービスを設定する必要はありません。

表1.54 Ingestor に渡される Jaeger パラメーター

パラメーター	説明	値
spec: ingester: options: {}	Ingester サービスを定義する設定オプション。	
options: deadlockInterval:	Ingester が終了するまでメッセージを待機する間隔 (秒単位または分単位) を指定します。システムの初期化中にメッセージが到着しない場合に Ingester が終了しないように、デッドロックの間隔はデフォルトで無効化 (0 に設定) されます。	分と秒 (例: 1m0s) デフォルト値は 0 です。
options: kafka: consumer: topic:	topic パラメーターは、コレクターによってメッセージを生成するために使用され、Ingester によってメッセージを消費するために使用される Kafka 設定を特定します。	コンシューマーのラベル例: jaeger-spans
options: kafka: consumer: brokers:	メッセージを消費するために Ingester によって使用される Kafka 設定を特定します。	ブローカーのラベル (例: my-cluster-kafka-brokers.kafka:9092)
options: log-level:	Ingester のロギングレベル。	使用できる値は、 debug 、 info 、 warn 、 error 、 fatal 、 dpanic 、 panic です。

ストリーミング Collector および Ingester の例

```

apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-streaming
spec:
  strategy: streaming
  collector:
    options:
      kafka:
        producer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
  ingester:
    options:
      kafka:

```

```

consumer:
  topic: jaeger-spans
  brokers: my-cluster-kafka-brokers.kafka:9092
ingester:
  deadlockInterval: 5
storage:
  type: elasticsearch
options:
  es:
    server-urls: http://elasticsearch:9200

```

1.27. SERVICE MESH のアンインストール

Red Hat OpenShift Service Mesh を既存の Red Hat OpenShift Service on AWS インスタンスからアンインストールし、そのリソースを削除するには、コントロールプレーンと Operator を削除してから、コマンドを実行してリソースを手動で削除する必要があります。

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

Service Mesh を既存の Red Hat OpenShift Service on AWS インスタンスからアンインストールするには、最初に Service Mesh コントロールプレーンと Operator を削除します。次に、コマンドを実行して残りのリソースを削除します。

1.27.1.1. Web コンソールを使用した Service Mesh コントロールプレーンの削除

Web コンソールを使用して Red Hat OpenShift Service Mesh コントロールプレーンを削除します。

手順

1. Red Hat OpenShift Service on AWS Web コンソールにログインします。
2. **Project** メニューをクリックし、Service Mesh コントロールプレーンをインストールしたプロジェクト (例: **istio-system**) を選択します。
3. **Operators** → **Installed Operators** に移動します。
4. **Provided APIs** の **Service Mesh Control Plane** をクリックします。
5. **ServiceMeshControlPlane** メニュー  をクリックします。
6. **Delete Service Mesh Control Plane** をクリックします。
7. 確認ダイアログウィンドウで **Delete** をクリックし、**ServiceMeshControlPlane** を削除します。

1.27.1.2. CLI を使用した Service Mesh コントロールプレーンの削除

CLI を使用して Red Hat OpenShift Service Mesh コントロールプレーンを削除します。この例では、**istio-system** は、コントロールプレーンプロジェクトです。

手順

1. Red Hat OpenShift Service on AWS CLI にログインします。

2. 次のコマンドを実行して、**ServiceMeshMemberRoll** リソースを削除します。

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

3. 以下のコマンドを実行して、インストールした **ServiceMeshControlPlane** の名前を取得します。

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

4. **<name_of_custom_resource>** を先のコマンドの出力に置き換え、以下のコマンドを実行してカスタムリソースを削除します。

```
$ oc delete smcp -n istio-system <name_of_custom_resource>
```

1.27.2. インストールされた Operator の削除

Red Hat OpenShift Service Mesh を正常に削除するには、Operator を削除する必要があります。Red Hat OpenShift Service Mesh Operator を削除した後、Kiali Operator、Red Hat OpenShift 分散トレーシング Platform (Jaeger) Operator、および OpenShift Elasticsearch Operator を削除する必要があります。

1.27.2.1. Operator の削除

以下の手順に従って、Red Hat OpenShift Service Mesh を設定する Operator を削除します。以下の Operator ごとに手順を繰り返します。

- Red Hat OpenShift Service Mesh
- Kiali
- Red Hat OpenShift 分散トレーシングプラットフォーム (Jaeger)
- OpenShift Elasticsearch

手順

1. Red Hat OpenShift Service on AWS 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、キーワードを **Filter by name** に入力して各 Operator を見つけます。次に、Operator 名をクリックします。
3. **Operator Details** ページで、**Actions** メニューから **Uninstall Operator** を選択します。プロンプトに従って各 Operator をアンインストールします。

1.27.3. Operator リソースのクリーンアップ

Red Hat OpenShift Service on AWS Web コンソールを使用して、Red Hat OpenShift Service Mesh Operator を削除した後に残ったリソースを手動で削除できます。

前提条件

- クラスター管理アクセスを持つアカウント。(Red Hat OpenShift Dedicated を使用する場合は **dedicated-admin** ロールがあるアカウント。)

- OpenShift CLI (**oc**) へのアクセスがある。

手順

1. Red Hat OpenShift Service on AWS CLI にクラスター管理者としてログインします。
2. 以下のコマンドを実行して、Operator のアンインストール後にリソースをクリーンアップします。サービスメッシュなしで分散トレーシング Platform (Jaeger) をスタンドアロンサービスとして引き続き使用する場合は、Jaeger リソースを削除しないでください。



注記

OpenShift Elasticsearch Operator はデフォルトで **openshift-operators-redhat** にインストールされます。他の Operator はデフォルトで **openshift-operators** namespace にインストールされます。Operator を別の namespace にインストールしている場合は、**openshift-operators** を Red Hat OpenShift Service Mesh Operator がインストールされていたプロジェクトの名前に置き換えます。

```
$ oc delete svc maistra-admission-controller -n openshift-operators
```

```
$ oc -n openshift-operators delete ds -lmaistra-version
```

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

```
$ oc delete clusterrole istio-view istio-edit
```

```
$ oc delete clusterrole jaegers.jaegertracing.io-v1-admin jaegers.jaegertracing.io-v1-crdview  
jaegers.jaegertracing.io-v1-edit jaegers.jaegertracing.io-v1-view
```

```
$ oc delete cm -n openshift-operators maistra-operator-cabundle
```

```
$ oc delete cm -n openshift-operators istio-cni-config istio-cni-config-v2-3
```

```
$ oc delete sa -n openshift-operators istio-cni
```