OpenShift Container Platform 4.11

서비스 메시

서비스 메시 설치, 사용법, 릴리스 정보

Last Updated: 2024-01-05

OpenShift Container Platform 4.11 서비스 메시

서비스 메시 설치, 사용법, 릴리스 정보

법적 공지

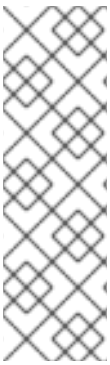
초록

이 문서에서는 OpenShift Container Platform에서 서비스 메시를 사용하는 방법에 대한 정보를 제공합니다.

차례

1장. 서비스 메시 2.X

1.1. OPENSHIFT SERVICE MESH 정보



참고

Red Hat OpenShift Service Mesh는 OpenShift Container Platform과 다른 주기로 출시되고 Red Hat OpenShift Service Mesh Operator가 **ServiceMeshControlPlane** 의 여러 버전 배포를 지원하므로 서비스 메시 문서는 제품의 마이너 버전에 대한 별도의 문서 세트를 유지 관리하지 않습니다. 현재 문서 세트는 특정 주제 또는 특정 기능에 대해 버전별 제한이 호출되지 않는 한 최신 버전의 Service Mesh에 적용됩니다.

Red Hat OpenShift Service Mesh 라이프 사이클 및 지원되는 플랫폼에 대한 자세한 내용은 플랫폼 라이프 사이클 정책을 참조하십시오.

1.1.1. Red Hat OpenShift Service Mesh 소개

Red Hat OpenShift Service Mesh는 애플리케이션에서 중앙 집중식 제어 지점을 생성하여 마이크로 서비스 아키텍처에서 다양한 문제에 대응합니다. 애플리케이션 코드를 변경하지 않고도 기존 분산 애플리케이션에 투명한 레이어를 추가합니다.

마이크로 서비스 아키텍처는 엔터프라이즈 애플리케이션의 작업을 모듈식 서비스로 분할하므로 확장 및 유지 관리를 더 쉽게 수행할 수 있습니다. 그러나 마이크로 서비스 아키텍처에 구축된 엔터프라이즈 애플리케이션이 크기와 복잡성이 증가함에 따라 마이크로 서비스 아키텍처의 이해 및 관리가 어려워집니다. 서비스 메시는 서비스 간 트래픽을 캡처하거나 차단하거나 다른 서비스에 대한 새 요청을 리디렉트 또는 생성하여 이러한 아키텍처의 문제에 대응할 수 있습니다.

오픈 소스 Istio project를 기반으로 하는 Service Mesh는 배포된 서비스 네트워크를 쉽게 구축할 수 있는 방법을 제공하여 검색, 로드 밸런싱, 서비스 간 인증, 실패 복구, 지표 및 모니터링을 지원합니다. 또한 서비스 메시는 A/B 테스트, 카나리아 릴리스, 액세스 제어, 엔드 투 엔드 인증을 비롯한 복잡한 운영 기능을 제공합니다.

1.1.2. 핵심 기능

Red Hat OpenShift Service Mesh는 서비스 네트워크 전반에서 여러 주요 기능을 균일하게 제공합니다.

- **트래픽 관리** - 서비스 간 트래픽 및 API 호출 흐름을 제어하고, 호출을 더 안정적으로 만들며, 불리한 조건에서도 네트워크를 보다 견고하게 만듭니다.
- **서비스 ID 및 보안** - 메시에서 확인 가능한 ID로 서비스를 제공하고 다양한 수준의 신뢰도를 갖춘 네트워크를 통해 전달될 때 서비스 트래픽을 보호할 수 있는 기능을 제공합니다.
- **정책 강화** - 서비스 간 상호 작용에 조직 정책을 적용하여 액세스 정책이 시행되고 리소스가 소비자 간에 공정하게 배포되도록 합니다. 애플리케이션 코드를 변경하는 것이 아니라 메시를 구성하여 정책 변경을 수행합니다.
- **Telemetry** - 서비스 간의 종속성과 트래픽 속성 및 흐름을 이해하여 문제를 신속하게 식별할 수 있는 기능을 제공합니다.

1.2. 서비스 메시 릴리스 노트

1.2.1. 보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 CTO Chris Wright의 메시지 를 참조하십시오.

1.2.2. 새로운 기능 및 개선 사항

이 릴리스에는 다음 구성 요소 및 개념과 관련된 개선 사항이 추가되었습니다.

1.2.2.1. Red Hat OpenShift Service Mesh 버전 2.4.4 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.11 이상 버전에서 지원됩니다.

1.2.2.1.1. Red Hat OpenShift Service Mesh 버전 2.4.4에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.16.7
Envoy Proxy	1.24.12
Jaeger	1.47.0
Kiali	1.65.10

1.2.2.2. Red Hat OpenShift Service Mesh 버전 2.4.3 새 기능

- Red Hat OpenShift Service Mesh Operator는 이제 ARM 기반 클러스터에서 기술 프리뷰 기능으로 사용할 수 있습니다.
- gRPC API를 사용하여 외부 인증 공급자를 구성하는 데 사용되는 **envoyExtAuthzGrpc** 필드가 추가되었습니다.
- CVE(Common Vulnerabilities and Exposures)가 해결되었습니다.
- 이 릴리스는 OpenShift Container Platform 4.10 및 최신 버전에서 지원됩니다.

1.2.2.2.1. Red Hat OpenShift Service Mesh 버전 2.4.3에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.16.7
Envoy Proxy	1.24.10
Jaeger	1.42.0
Kiali	1.65.8

1.2.2.2.2. Red Hat OpenShift Service Mesh operator to ARM 기반 클러스터

중요

Red Hat OpenShift Service Mesh operator to ARM 기반 클러스터는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

이번 릴리스에서는 ARM 기반 클러스터에서 Red Hat OpenShift Service Mesh Operator를 기술 프리뷰 기능으로 사용할 수 있습니다. 이미지는 Istio, Envoy, Prometheus, Kiali, Grafana에서 사용할 수 있습니다. Jaeger에서 이미지를 사용할 수 없으므로 Jaeger를 서비스 메시 애드온으로 비활성화해야 합니다.

1.2.2.2.3. 외부 인증 구성에 대한 원격 프로시저 호출(gRPC) API 지원

이번 개선된 기능에는 gRPC API를 사용하여 외부 권한 부여 공급자를 구성하기 위해 **envoyExtAuthzGrpc** 필드가 추가되었습니다.

1.2.2.3. Red Hat OpenShift Service Mesh 버전 2.4.2 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.10 이상 버전에서 지원됩니다.

1.2.2.3.1. Red Hat OpenShift Service Mesh 버전 2.4.2에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.16.7
Envoy Proxy	1.24.10
Jaeger	1.42.0
Kiali	1.65.7

1.2.2.4. Red Hat OpenShift Service Mesh 버전 2.4.1 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.10 이상 버전에서 지원됩니다.

1.2.2.4.1. Red Hat OpenShift Service Mesh 버전 2.4.1에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.16.5

구성 요소	버전
Envoy Proxy	1.24.8
Jaeger	1.42.0
Kiali	1.65.7

1.2.2.5. Red Hat OpenShift Service Mesh 버전 2.4 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.10 이상 버전에서 지원됩니다.

1.2.2.5.1. Red Hat OpenShift Service Mesh 버전 2.4에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.16.5
Envoy Proxy	1.24.8
Jaeger	1.42.0
Kiali	1.65.6

1.2.2.5.2. 클러스터 전체 배포

이번 개선된 기능에는 일반적으로 사용 가능한 클러스터 전체 배포 버전이 도입되었습니다. 클러스터 전체 배포에는 전체 클러스터의 리소스를 모니터링하는 서비스 메시 컨트롤 플레인이 포함되어 있습니다. 컨트롤 플레인은 모든 네임스페이스에서 단일 쿼리를 사용하여 메시 구성에 영향을 주는 각 Istio 또는 Kubernetes 리소스를 모니터링합니다. 클러스터 전체 배포에서 컨트롤 플레인이 수행하는 쿼리 수를 줄이면 성능이 향상됩니다.

1.2.2.5.3. 검색 선택기 지원

이번 개선된 기능에는 **meshConfig.discoverySelectors** 필드의 일반적으로 사용 가능한 버전이 도입되어 클러스터 전체 배포에서 서비스 메시 컨트롤 플레인에서 검색할 수 있는 서비스를 제한할 수 있습니다.

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

1.2.2.5.4. cert-manager istio-csr과의 통합

이번 업데이트를 통해 Red Hat OpenShift Service Mesh는 **cert-manager** 컨트롤러 및 **istio-csr** 에이전트와 통합됩니다. **cert-manager** 는 인증서 및 인증서 발행자를 Kubernetes 클러스터에서 리소스 유형으로 추가하고 해당 인증서를 획득, 갱신 및 사용하는 프로세스를 단순화합니다. **cert-manager** 는 Istio의 중간 CA 인증서를 제공하고 순환합니다. **istio-csr** 과의 통합을 통해 사용자는 Istio 프록시에서 인증서 요청을 **cert-manager** 에 위임할 수 있습니다. **ServiceMeshControlPlane** v2.4에서는 **cert-manager** 에서 **cacerts** 시크릿으로 제공하는 CA 인증서를 허용합니다.

참고

cert-manager 및 **istio-csr** 과의 통합은 IBM Power, IBM Z 및 IBM® LinuxONE에서 지원되지 않습니다.

1.2.2.5.5. 외부 권한 부여 시스템과 통합

이번 개선된 기능에는 **AuthorizationPolicy** 리소스의 **action: CUSTOM** 필드를 사용하여 Red Hat OpenShift Service Mesh를 외부 권한 부여 시스템과 통합하는 데 일반적으로 사용 가능한 방법이 도입되었습니다. **envoyExtAuthzHttp** 필드를 사용하여 액세스 제어를 외부 권한 부여 시스템에 위임합니다.

1.2.2.5.6. 외부 Prometheus 설치와 통합

이번 개선된 기능에는 일반적으로 사용 가능한 Prometheus 확장 공급자 버전이 도입되었습니다. **spec.meshConfig** 사양에서 **extensionProviders** 필드의 값을 **prometheus** 로 설정하여 OpenShift Container Platform 모니터링 스택 또는 사용자 정의 Prometheus 설치에 지표를 노출할 수 있습니다. Telemetry 오브젝트는 트래픽 지표를 수집하도록 Istio 프록시를 구성합니다. 서비스 메시는 Prometheus 지표에 대한 Telemetry API만 지원합니다.

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

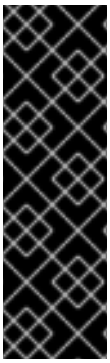
1.2.2.5.7. 단일 스택 IPv6 지원

이번 개선된 기능을 통해 단일 스택 IPv6 클러스터를 일반적으로 지원하여 광범위한 IP 주소에 대한 액세스를 제공합니다. 듀얼 스택 IPv4 또는 IPv6 클러스터는 지원되지 않습니다.

참고

IBM Power, IBM Z 및 IBM® LinuxONE에서는 단일 스택 IPv6 지원을 사용할 수 없습니다.

1.2.2.5.8. OpenShift Container Platform Gateway API 지원



중요

OpenShift Container Platform Gateway API 지원은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

이번 개선된 기능에는 OpenShift Container Platform Gateway API의 업데이트된 기술 프리뷰 버전이 도입되었습니다. 기본적으로 OpenShift Container Platform Gateway API는 비활성화되어 있습니다.

1.2.2.5.8.1. OpenShift Container Platform Gateway API 활성화

OpenShift Container Platform Gateway API를 활성화하려면 **ServiceMeshControlPlane** 리소스의 **techPreview.gatewayAPI** 사양에서 **활성화된** 필드의 값을 **true** 로 설정합니다.

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

이전에는 환경 변수를 사용하여 Gateway API를 활성화했습니다.

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

1.2.2.5.9. 인프라 노드에 컨트롤 플레인 배포

OpenShift 인프라 노드에서 서비스 메시 컨트롤 플레인 배포가 지원 및 문서화됩니다. 자세한 내용은 다음 설명서를 참조하십시오.

- 인프라 노드에서 실행되도록 모든 Service Mesh Control Plane 구성 요소 구성
- 인프라 노드에서 실행되도록 개별 Service Mesh Control Plane 구성 요소 구성

1.2.2.5.10. Istio 1.16 지원

Service Mesh 2.4는 Istio 1.16을 기반으로 하며 새로운 기능 및 제품 개선 사항을 제공합니다. 많은 Istio 1.16 기능이 지원되지만 다음 예외가 표시되어야 합니다.

- 사이드카의 HBONE 프로토콜은 지원되지 않는 실험 기능입니다.

- ARM64 아키텍처의 서비스 메시는 지원되지 않습니다.
- OpenTelemetry API는 기술 프리뷰 기능으로 유지됩니다.

1.2.2.6. Red Hat OpenShift Service Mesh 버전 2.3.8 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.11 이상 버전에서 지원됩니다.

1.2.2.6.1. Red Hat OpenShift Service Mesh 버전 2.3.8에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.14.5
Envoy Proxy	1.22.11
Jaeger	1.47.0
Kiali	1.57.13

1.2.2.7. Red Hat OpenShift Service Mesh 버전 2.3.7 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.10 이상 버전에서 지원됩니다.

1.2.2.7.1. Red Hat OpenShift Service Mesh 버전 2.3.7에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.14.6
Envoy Proxy	1.22.11
Jaeger	1.42.0
Kiali	1.57.11

1.2.2.8. Red Hat OpenShift Service Mesh 버전 2.3.6 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.10 이상 버전에서 지원됩니다.

1.2.2.8.1. Red Hat OpenShift Service Mesh 버전 2.3.6에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.14.5
Envoy Proxy	1.22.11
Jaeger	1.42.0
Kiali	1.57.10

1.2.2.9. Red Hat OpenShift Service Mesh 버전 2.3.5 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.10 이상 버전에서 지원됩니다.

1.2.2.9.1. Red Hat OpenShift Service Mesh 버전 2.3.5에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.14.5
Envoy Proxy	1.22.9
Jaeger	1.42.0
Kiali	1.57.10

1.2.2.10. Red Hat OpenShift Service Mesh 버전 2.3.4 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.10 이상 버전에서 지원됩니다.

1.2.2.10.1. Red Hat OpenShift Service Mesh 버전 2.3.4에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.14.6
Envoy Proxy	1.22.9
Jaeger	1.42.0
Kiali	1.57.9

1.2.2.11. Red Hat OpenShift Service Mesh 버전 2.3.3 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.11.1. Red Hat OpenShift Service Mesh 버전 2.3.3에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.14.5
Envoy Proxy	1.22.9
Jaeger	1.42.0
Kiali	1.57.7

1.2.2.12. Red Hat OpenShift Service Mesh 버전 2.3.2 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.12.1. Red Hat OpenShift Service Mesh 버전 2.3.2에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.14.5
Envoy Proxy	1.22.7
Jaeger	1.39
Kiali	1.57.6

1.2.2.13. Red Hat OpenShift Service Mesh 버전 2.3.1 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스에는 새로운 기능이 도입되고, CVE(Common Vulnerabilities and Exposures)에 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.13.1. Red Hat OpenShift Service Mesh 버전 2.3.1에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.14.5
Envoy Proxy	1.22.4
Jaeger	1.39

구성 요소	버전
Kiali	1.57.5

1.2.2.14. Red Hat OpenShift Service Mesh 버전 2.3 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스에는 새로운 기능이 도입되고, CVE(Common Vulnerabilities and Exposures)에 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.14.1. Red Hat OpenShift Service Mesh 버전 2.3에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.14.3
Envoy Proxy	1.22.4
Jaeger	1.38
Kiali	1.57.3

1.2.2.14.2. 새 CNI(Container Network Interface) DaemonSet 컨테이너 및 ConfigMap

openshift-operators 네임스페이스에는 새로운 istio CNI DaemonSet **istio-cni-node-v2-3** 및 새 **ConfigMap** 리소스 **istio-cni-config-v2-3** 이 포함됩니다.

Service Mesh Control Plane 2.3으로 업그레이드할 때 기존 **istio-cni-node** DaemonSet은 변경되지 않으며 새로운 **istio-cni-node-v2-3** DaemonSet이 생성됩니다.

이 이름 변경은 이전 릴리스 또는 이전 릴리스를 사용하여 배포된 Service Mesh Control Plane과 연결된 **istio-cni-node** CNI DaemonSet에는 영향을 미치지 않습니다.

1.2.2.14.3. 게이트웨이 삽입 지원

이번 릴리스에서는 게이트웨이 주입에 대한 일반적으로 사용 가능한 지원이 도입되었습니다. 게이트웨이 구성은 서비스 워크로드와 함께 실행되는 사이드카 Envoy 프록시 대신 메시의 에지에서 실행되는 독립 실행형 Envoy 프록시에 적용됩니다. 이를 통해 게이트웨이 옵션을 사용자 지정할 수 있습니다. 게이트웨이 삽입을 사용하는 경우 게이트웨이 프록시를 실행하려는 네임스페이스에 다음 리소스를 만들어야 합니다. **Service**,**Deployment**,**Role**, **RoleBinding**.

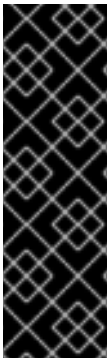
1.2.2.14.4. Istio 1.14 지원

서비스 메시 2.3은 새로운 기능 및 제품 개선 사항을 제공하는 Istio 1.14를 기반으로 합니다. 많은 Istio 1.14 기능이 지원되지만 다음 예외에 유의하십시오.

- 이미지 필드를 제외하고 ProxyConfig API가 지원됩니다.
- Telemetry API는 기술 프리뷰 기능입니다.

- SPIRE 런타임은 지원되는 기능이 아닙니다.

1.2.2.14.5. OpenShift Service Mesh 콘솔



중요

OpenShift Service Mesh Console은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

이번 릴리스에서는 Kiali 인터페이스를 OpenShift 웹 콘솔에 직접 통합하는 OpenShift Container Platform Service Mesh Console의 기술 프리뷰 버전이 도입되었습니다. 자세한 내용은 OpenShift Service Mesh Console 소개(기술 프리뷰)를 참조하십시오.

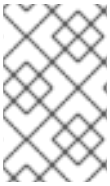
1.2.2.14.6. 클러스터 전체 배포

중요

클러스터 전체 배포는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

이번 릴리스에서는 클러스터 전체 배포를 기술 프리뷰 기능으로 도입했습니다. 클러스터 전체 배포에는 전체 클러스터의 리소스를 모니터링하는 Service Mesh Control Plane이 포함되어 있습니다. 컨트롤 플레인은 모든 네임스페이스에서 단일 쿼리를 사용하여 메시 구성에 영향을 미치는 각 Istio 또는 Kubernetes 리소스 종류를 모니터링합니다. 반면 다중 테넌트 접근 방식은 각 리소스 유형에 대해 네임스페이스당 쿼리를 사용합니다. 클러스터 전체 배포에서 컨트롤 플레인이 수행하는 쿼리 수를 줄이면 성능이 향상됩니다.



참고

이 클러스터 전체 배포 문서는 SMCP v2.3을 사용하여 배포된 컨트롤 플레인에만 적용됩니다. SMCP v2.3을 사용하여 생성된 클러스터 전체 배포는 SMCP v2.4를 사용하여 생성된 클러스터 전체 배포와 호환되지 않습니다.

1.2.2.14.6.1. 클러스터 전체 배포 구성

다음 예제 **ServiceMeshControlPlane** 오브젝트는 클러스터 전체 배포를 구성합니다.

클러스터 전체 배포를 위한 SMCP를 생성하려면 사용자가 **cluster-admin** ClusterRole에 속해야 합니다. SMCP가 클러스터 전체 배포용으로 구성된 경우 클러스터에서 유일한 SMCP여야 합니다. 컨트롤 플레인 모드를 다중 테넌트에서 클러스터 전체(또는 클러스터 전체에서 다중 테넌트로)로 변경할 수 없습니다. 다중 테넌트 컨트롤 플레인이 이미 있는 경우 이를 삭제하고 새 플레인을 생성합니다.

이 예에서는 클러스터 전체 배포를 위해 SMCP를 구성합니다.

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

1 Istiod를 사용하면 각 개별 네임스페이스를 모니터링하지 않고 클러스터 수준에서 리소스를 모니터링할 수 있습니다.

또한 클러스터 전체 배포를 위해 SMMR도 구성해야 합니다. 이 예에서는 클러스터 전체 배포에 대해 SMMR을 구성합니다.

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

1 이후에 생성하는 네임스페이스를 포함하여 모든 네임스페이스를 메시에 추가합니다. 다음 네임스페이스는 메시의 일부가 아닙니다. kube, openshift, kube-* 및 openshift-*.

1.2.2.15. Red Hat OpenShift Service Mesh 버전 2.2.11 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.11 이상 버전에서 지원됩니다.

1.2.2.15.1. Red Hat OpenShift Service Mesh 버전 2.2.11에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.47.0
Kiali	1.48.10

1.2.2.16. Red Hat OpenShift Service Mesh 버전 2.2.10 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.10 이상 버전에서 지원됩니다.

1.2.2.16.1. Red Hat OpenShift Service Mesh 버전 2.2.10에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.42.0
Kiali	1.48.8

1.2.2.17. Red Hat OpenShift Service Mesh 버전 2.2.9 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.10 이상 버전에서 지원됩니다.

1.2.2.17.1. Red Hat OpenShift Service Mesh 버전 2.2.9에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.42.0
Kiali	1.48.7

1.2.2.18. Red Hat OpenShift Service Mesh 버전 2.2.8 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.10 이상 버전에서 지원됩니다.

1.2.2.18.1. Red Hat OpenShift Service Mesh 버전 2.2.8에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.42.0
Kiali	1.48.7

1.2.2.19. Red Hat OpenShift Service Mesh 버전 2.2.7 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 대한 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.10 이상 버전에서 지원됩니다.

1.2.2.19.1. Red Hat OpenShift Service Mesh 버전 2.2.7에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.42.0
Kiali	1.48.6

1.2.2.20. Red Hat OpenShift Service Mesh 버전 2.2.6 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.20.1. Red Hat OpenShift Service Mesh 버전 2.2.6에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.39
Kiali	1.48.5

1.2.2.21. Red Hat OpenShift Service Mesh 버전 2.2.5 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.21.1. Red Hat OpenShift Service Mesh 버전 2.2.5에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.12.9
Envoy Proxy	1.20.8

구성 요소	버전
Jaeger	1.39
Kiali	1.48.3

1.2.2.22. Red Hat OpenShift Service Mesh 버전 2.2.4 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.22.1. Red Hat OpenShift Service Mesh 버전 2.2.4에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.36.14
Kiali	1.48.3

1.2.2.23. Red Hat OpenShift Service Mesh 버전 2.2.3 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures), 버그 수정 및 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.23.1. Red Hat OpenShift Service Mesh 버전 2.2.3에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.12.9
Envoy Proxy	1.20.8
Jaeger	1.36
Kiali	1.48.3

1.2.2.24. Red Hat OpenShift Service Mesh 버전 2.2.2 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures), 버그 수정 및 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.24.1. Red Hat OpenShift Service Mesh 버전 2.2.2에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.12.7
Envoy Proxy	1.20.6
Jaeger	1.36
Kiali	1.48.2-1

1.2.2.24.2. 경로 레이블 복사

이번 개선된 기능을 통해 주석 복사 외에도 OpenShift 경로의 특정 레이블을 복사할 수 있습니다. Red Hat OpenShift Service Mesh는 Istio Gateway 리소스에 있는 모든 레이블 및 주석(kubectl.kubernetes.io로 시작하는 주석 제외)을 관리형 OpenShift 경로 리소스에 복사합니다.

1.2.2.25. Red Hat OpenShift Service Mesh 버전 2.2.1 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures), 버그 수정 및 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.25.1. Red Hat OpenShift Service Mesh 버전 2.2.1에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.12.7
Envoy Proxy	1.20.6
Jaeger	1.34.1
Kiali	1.48.2-1

1.2.2.26. Red Hat OpenShift Service Mesh 2.2 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스에는 새로운 기능 및 개선 사항이 추가되었으며 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.26.1. Red Hat OpenShift Service Mesh 버전 2.2에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.12.7
Envoy Proxy	1.20.4
Jaeger	1.34.1

구성 요소	버전
Kiali	1.48.0.16

1.2.2.26.2. WasmPlugin API

이 릴리스에서는ECDHEs **mPlugin** API에 대한 지원이 추가되어 **ServiceMeshExtension** API를 더 이상 사용하지 않습니다.

1.2.2.26.3. ROSA 지원

이번 릴리스에서는 다중 클러스터 페더레이션을 포함하여 AWS(ROSA)의 Red Hat OpenShift에 대한 서비스 메시 지원을 도입했습니다.

1.2.2.26.4. Istio-node DaemonSet의 이름

이번 릴리스에서는 **istio-node** DaemonSet이 업스트림 Istio의 이름과 일치하도록 **istio-cni-node** 로 이름이 변경되었습니다.

1.2.2.26.5. Envoy 사이드카 네트워킹 변경

Istio 1.10은 기본적으로 **lo** 대신 **eth0** 을 사용하여 애플리케이션 컨테이너로 트래픽을 전송하도록 Envoy를 업데이트했습니다.

1.2.2.26.6. 서비스 메시 컨트롤 플레인 1.1

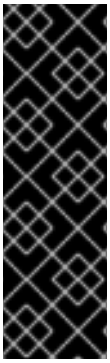
이번 릴리스에서는 모든 플랫폼의 서비스 메시 1.1을 기반으로 서비스 메시 컨트롤 플레인에 대한 지원 종료를 표시합니다.

1.2.2.26.7. Istio 1.12 지원

서비스 메시 2.2는 Istio 1.12를 기반으로 하며 새로운 기능 및 제품 개선 사항을 제공합니다. 많은 Istio 1.12 기능이 지원되지만 다음과 같은 지원되지 않는 기능을 고려해야 합니다.

- AuthPolicy Dry Run은 기술 프리뷰 기능입니다.
- gRPC 프록시 없는 서비스 메시는 기술 프리뷰 기능입니다.
- Telemetry API는 기술 프리뷰 기능입니다.
- 검색 선택기는 지원되는 기능이 아닙니다.
- 외부 컨트롤 플레인은 지원되지 않습니다.
- 게이트웨이 삽입은 지원되는 기능이 아닙니다.

1.2.2.26.8. Kubernetes 게이트웨이 API



중요

Kubernetes 게이트웨이 API는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

Kubernetes Gateway API는 기본적으로 비활성화되어 있는 기술 프리뷰 기능입니다. Kubernetes API 배포 컨트롤러가 비활성화되어 있는 경우 수신 게이트웨이를 수동으로 배포하고 생성된 게이트웨이 오브젝트에 연결해야 합니다.

Kubernetes API 배포 컨트롤러가 활성화된 경우 게이트웨이 오브젝트가 생성될 때 수신 게이트웨이가 자동으로 배포됩니다.

1.2.2.26.8.1. 게이트웨이 API CRD 설치

게이트웨이 API CRD는 기본적으로 OpenShift 클러스터에 사전 설치되지 않습니다. SMCP에서 게이트웨이 API 지원을 활성화하기 전에 CRD를 설치합니다.

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

1.2.2.26.8.2. Kubernetes 게이트웨이 API 활성화

기능을 활성화하려면 **ServiceMeshControlPlane** 에서 **Istiod** 컨테이너에 대해 다음 환경 변수를 설정합니다.

```
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"
```

게이트웨이 API 리스너에 대한 경로 연결 제한은 **SameNamespace** 또는 **All** 설정을 사용하여 수행할 수 있습니다. Istio는 **listeners.allowedRoutes.namespaces** 의 라벨 선택기 사용을 무시하고 기본 동작(**SameNamespace**)으로 되돌립니다.

1.2.2.26.8.3. 기존 게이트웨이를 게이트웨이 리소스에 수동으로 연결

Kubernetes API 배포 컨트롤러가 비활성화된 경우 수동으로 배포한 다음 수신 게이트웨이를 생성된 Gateway 리소스에 연결해야 합니다.

```
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: Gateway
metadata:
```

```
  name: gateway
spec:
  addresses:
  - value: ingress.istio-gateways.svc.cluster.local
    type: Hostname
```

1.2.2.27. Red Hat OpenShift Service Mesh 2.1.6 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.27.1. Red Hat OpenShift Service Mesh 버전 2.1.6에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.9.9
Envoy Proxy	1.17.5
Jaeger	1.36
Kiali	1.36.16

1.2.2.28. Red Hat OpenShift Service Mesh 2.1.5.2 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)에 버그 수정이 포함되어 있으며 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.28.1. Red Hat OpenShift Service Mesh 버전 2.1.5.2에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.9.9
Envoy Proxy	1.17.5
Jaeger	1.36
Kiali	1.24.17

1.2.2.29. Red Hat OpenShift Service Mesh 2.1.5.1 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures), 버그 수정 및 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.29.1. Red Hat OpenShift Service Mesh 버전 2.1.5.1에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.9.9
Envoy Proxy	1.17.5
Jaeger	1.36
Kiali	1.36.13

1.2.2.30. Red Hat OpenShift Service Mesh 2.1.5 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures), 버그 수정 및 OpenShift Container Platform 4.9 이상 버전에서 지원됩니다.

1.2.2.30.1. Red Hat OpenShift Service Mesh 버전 2.1.5에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.36
Kiali	1.36.12-1

1.2.2.31. Red Hat OpenShift Service Mesh 2.1.4 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

1.2.2.31.1. Red Hat OpenShift Service Mesh 버전 2.1.4에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.30.2
Kiali	1.36.12-1

1.2.2.32. Red Hat OpenShift Service Mesh 2.1.3 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

1.2.2.32.1. Red Hat OpenShift Service Mesh 버전 2.1.3에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.30.2
Kiali	1.36.10-2

1.2.2.33. Red Hat OpenShift Service Mesh 2.1.2.1 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

1.2.2.33.1. Red Hat OpenShift Service Mesh 버전 2.1.2.1에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.30.2
Kiali	1.36.9

1.2.2.34. Red Hat OpenShift Service Mesh 2.1.2 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

이번 릴리스에서는 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator가 기본적으로 **openshift-distributed-tracing** 네임스페이스에 설치됩니다. 이전에는 기본 설치가 **openshift-operator** 네임스페이스에 있었습니다.

1.2.2.34.1. Red Hat OpenShift Service Mesh 버전 2.1.2에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.9.9

구성 요소	버전
Envoy Proxy	1.17.1
Jaeger	1.30.1
Kiali	1.36.8

1.2.2.35. Red Hat OpenShift Service Mesh 2.1.1 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

이 릴리스에는 네트워크 정책의 자동 생성을 비활성화하는 기능도 추가되었습니다.

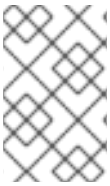
1.2.2.35.1. Red Hat OpenShift Service Mesh 버전 2.1.1에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.9.9
Envoy Proxy	1.17.1
Jaeger	1.24.1
Kiali	1.36.7

1.2.2.35.2. 네트워크 정책 비활성화

Red Hat OpenShift Service Mesh는 Service Mesh Control Plane 및 애플리케이션 네임스페이스에서 여러 **NetworkPolicies** 리소스를 자동으로 생성하고 관리합니다. 애플리케이션과 컨트롤 플레인이 서로 통신할 수 있도록 하기 위한 것입니다.

예를 들어 회사 보안 정책을 적용하는 등 **NetworkPolicies** 리소스의 자동 생성 및 관리를 비활성화하려면 이를 수행할 수 있습니다. **ServiceMeshControlPlane** 을 편집하여 **spec.security.manageNetworkPolicy** 설정을 **false**로 설정할 수 있습니다.



참고

spec.security.manageNetworkPolicy Red Hat OpenShift Service Mesh를 비활성화하면 **NetworkPolicy** 오브젝트가 생성되지 **않습니다**. 시스템 관리자는 네트워크를 관리하고 이러한 문제가 발생할 수 있는 문제를 해결합니다.

절차

1. OpenShift Container Platform 웹 콘솔에서 **Operator → 설치된 Operator**를 클릭합니다.
2. 프로젝트 메뉴에서 Service Mesh Control Plane을 설치한 프로젝트(예: **istio-system**)를 선택합니다.

3. Red Hat OpenShift Service Mesh Operator를 클릭합니다. **Istio Service Mesh Control Plane**열에서 **ServiceMeshControlPlane**의 이름을 클릭합니다(예: **basic-install**).

4. **ServiceMeshControlPlane 세부 정보 만들기** 페이지에서 **YAML**을 클릭하여 구성을 수정합니다.

5. 이 예와 같이 **ServiceMeshControlPlane** 필드 **spec.security.manageNetworkPolicy** 를 **false** 로 설정합니다.

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

6. **저장**을 클릭합니다.

1.2.2.36. 새로운 기능 및 개선 사항 Red Hat OpenShift Service Mesh 2.1

이번 Red Hat OpenShift Service Mesh 릴리스에는 OpenShift Container Platform 4.6 EUS, 4.7, 4.8, 4.9와 함께 Istio 1.9.8, Envoy 프록시 1.17.1, Jaeger 1.24.1 및 Kiali 1.36.5에 대한 지원이 추가되었습니다.

1.2.2.36.1. Red Hat OpenShift Service Mesh 버전 2.1에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.9.6
Envoy Proxy	1.17.1
Jaeger	1.24.1
Kiali	1.36.5

1.2.2.36.2. 서비스 메시 페더레이션

서비스 메시를 통합하기 위해 CRD(Custom Resource Definitions)가 추가되었습니다. 서비스 메시는 동일한 클러스터 또는 다른 OpenShift 클러스터 모두에서 결합될 수 있습니다. 다음과 같은 새 리소스는 다음과 같습니다.

- **ServiceMeshpeer** - 게이트웨이 구성, 루트 신뢰 인증서 구성 및 상태 필드를 포함하여 별도의 서비스 메시를 사용하여 페더레이션을 정의합니다. 한 쌍의 연합 메시에서 각 메시는 별도의 **ServiceMeshPeriod 리소스를** 정의합니다.

- **ExportedServiceMeshSet - 피어 메시에서 가져올 수 있는 지정된 ServiceMesh** peer에 사용할 수 있는 서비스를 정의합니다.

- **ImportedServiceSet** - 피어 메시에서 가져온 지정된 **ServiceMesh** peer에 대한 서비스를 정의합니다. 또한 이러한 서비스는 피어의 **ExportedServiceMeshSet** 리소스에서 사용할 수 있어야 합니다.

Service Mesh Federation은 AWS의 Red Hat OpenShift Service on AWS (ROSA), Azure Red Hat OpenShift (ARO) 또는 OpenShift Dedicated (OSD) 간에 지원되지 않습니다.

1.2.2.36.3. OVN-Kubernetes CNI(Container Network Interface) 사용 가능

OVN-Kubernetes CNI(Container Network Interface)는 이전에 Red Hat OpenShift Service Mesh 2.0.1에서 기술 프리뷰 기능으로 소개되었으며 이제 OpenShift Container Platform 4.7.32, OpenShift Container Platform 4.8.12 및 OpenShift Container Platform 4.9에서 사용할 수 있도록 Red Hat OpenShift Service Mesh 2.1 및 2.0.x에서 일반적으로 사용할 수 있습니다.

1.2.2.36.4. Service Mesh WebAssembly (WASM) Extensions

이제 2.0에서 기술 프리뷰로 처음 도입된 **ServiceMeshExtensions** CRD(Custom Resource Definitions CRD)를 일반적으로 사용할 수 있습니다. CRD를 사용하여 자체 플러그인을 빌드할 수 있지만 Red Hat은 생성하는 플러그인에 대한 지원을 제공하지 않습니다.

Mixer는 Service Mesh 2.1에서 완전히 제거되었습니다. Mixer가 활성화된 경우 Service Mesh 2.0.x 릴리스에서 2.1으로 업그레이드가 차단됩니다. Mixer 플러그인은 WebAssembly Extensions에 포트해야 합니다.

1.2.2.36.5. 3scale WebAssembly Adapter(WASM)

이제 Mixer가 공식적으로 제거되면 OpenShift Service Mesh 2.1에서 3scale mixer 어댑터를 지원하지 않습니다. Service Mesh 2.1으로 업그레이드하기 전에 Mixer 기반 3scale 어댑터 및 추가 Mixer 플러그인을 제거합니다. 그런 다음 **ServiceMeshExtension** 리소스를 사용하여 Service Mesh 2.1+로 새 3scale WebAssembly 어댑터를 수동으로 설치하고 구성합니다.

3scale 2.11은 **WebAssembly** 를 기반으로 업데이트된 서비스 메시 통합을 도입합니다.

1.2.2.36.6. Istio 1.9 지원

서비스 메시 2.1은 Istio 1.9를 기반으로 하며, 이로 인해 많은 새 기능과 제품 개선 사항이 제공됩니다. Istio 1.9 대부분의 기능이 지원되지만 다음 예외를 유의해야 합니다.

- 가상 머신 통합은 아직 지원되지 않습니다.
- Kubernetes Gateway API는 아직 지원되지 않습니다.
- WebAssembly HTTP 필터의 원격 가져오기 및 로드는 아직 지원되지 않습니다.
- Kubernetes CSR API를 사용한 사용자 정의 CA 통합은 아직 지원되지 않습니다.
- 트래픽 모니터링을 위한 요청 분류는 기술 프리뷰 기능입니다.
- 권한 부여 정책의 CUSTOM 작업을 통한 외부 권한 부여 시스템과의 통합은 기술 프리뷰 기능입니다.

1.2.2.36.7. Service Mesh Operator 성능 개선

Red Hat OpenShift Service Mesh가 모든 **ServiceMeshControlPlane** 조정이 끝나면 이전 리소스를 정리하는 데 사용하는 시간이 단축되었습니다. 이로 인해 **ServiceMeshControlPlane** 배포가 빨라지고 기존 SMCP에 변경 사항이 보다 신속하게 적용됩니다.

1.2.2.36.8. Kiali 업데이트

Kiali 1.36에는 다음과 같은 기능 및 향상된 기능이 포함되어 있습니다.

- 서비스 메시 문제 해결 기능
    - 컨트롤 플레인 및 게이트웨이 모니터링

    - 프록시 동기화 상태
    - Envoy 구성 보기
    - Envoy 프록시 및 애플리케이션 로그 간 연결 표시 통합 보기
- 페더레이션 서비스 메시 뷰를 지원하기 위한 네임스페이스 및 클러스터 박스
- 새로운 검증, 마법사 및 분산 추적 개선 사항

1.2.2.37. Red Hat OpenShift Service Mesh 2.0.11.1 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures), 버그 수정, OpenShift Container Platform 4.9 이상에서 지원됩니다.

1.2.2.37.1. Red Hat OpenShift Service Mesh 버전 2.0.11.1에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.6.14
Envoy Proxy	1.14.5
Jaeger	1.36
Kiali	1.24.17

1.2.2.38. Red Hat OpenShift Service Mesh 2.0.11 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures), 버그 수정, OpenShift Container Platform 4.9 이상에서 지원됩니다.

1.2.2.38.1. Red Hat OpenShift Service Mesh 버전 2.0.11에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.6.14
Envoy Proxy	1.14.5
Jaeger	1.36
Kiali	1.24.16-1

1.2.2.39. Red Hat OpenShift Service Mesh 2.0.10 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

1.2.2.39.1. Red Hat OpenShift Service Mesh 버전 2.0.10에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.6.14
Envoy Proxy	1.14.5
Jaeger	1.28.0
Kiali	1.24.16-1

1.2.2.40. Red Hat OpenShift Service Mesh 2.0.9 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

1.2.2.40.1. Red Hat OpenShift Service Mesh 버전 2.0.9에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.6.14
Envoy Proxy	1.14.5
Jaeger	1.24.1
Kiali	1.24.11

1.2.2.41. Red Hat OpenShift Service Mesh 2.0.8 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 버그 수정을 처리합니다.

1.2.2.42. Red Hat OpenShift Service Mesh 2.0.7.1 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)를 제공합니다.

1.2.2.42.1. Red Hat OpenShift Service Mesh가 URI 조각을 처리하는 방법 변경

Red Hat OpenShift Service Mesh에는 원격으로 악용할 수 있는 취약점인 CVE-2021-39156 이 포함되어 있습니다. 여기서 URI 경로의 # 문자로 시작하는 URL 섹션이 Istio URI 경로 기반 권한 부여 정책을 우회할 수 있습니다. 예를 들어 Istio 권한 부여 정책은 URI 경로 **/user/profile** 에 전송된 요청을 거부합니다. 취약한 버전에서 URI 경로 **/user/profile#section1** 이 있는 요청이 거부 정책 및 경로를 백엔드로 바이패스합니다(정규화된 URI **경로 /user/profile%23section1**)으로 인해 보안 문제가 발생할 수 있습니다.

DENY 작업 및 **operation.paths**.paths와 함께 권한 부여 정책을 사용하거나 ALLOW 작업 및 **operation.notPaths** 와 함께 권한 부여 정책을 사용하는 경우 이 취약점의 영향을 받습니다.

완화 기능을 사용하면 권한 부여 및 라우팅 전에 요청 URI의 조각 부분이 제거됩니다. 이렇게 하면 URI에 조각이 있는 요청이 내용 부분이 없는 URI를 기반으로 하는 권한 부여 정책을 우회하지 않습니다.

완화 방법의 새 동작을 옵트아웃하려면 URI의 fragment 섹션이 유지됩니다. URI 조각을 유지하도록 **ServiceMeshControlPlane** 을 구성할 수 있습니다.

주의

새 동작을 비활성화하면 위에서 설명한 대로 경로를 정규화하고 안전하지 않은 것으로 간주됩니다. URI 조각을 유지하기 전에 보안 정책에서 이 작업을 수행할 수 있어야 합니다.

ServiceMeshControlPlane 수정의 예

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

1.2.2.42.2. 권한 부여 정책에 필요한 업데이트

Istio는 호스트 이름 자체와 일치하는 모든 포트 모두에 대한 호스트 이름을 생성합니다. 예를 들어 "httpbin.foo" 호스트의 가상 서비스 또는 게이트웨이는 "httpbin.foo 및 httpbin.foo:*"과 일치하는 구성을 생성합니다. 그러나 정확한 권한 부여 정책은 **hosts** 또는 **notHosts** 필드에 지정된 정확한 문자열과만 일치합니다.

규칙에서 호스트 또는 notHosts 를 결정하는 데 정확한 문자열 비교를 사용하여 **AuthorizationPolicy** 리소스가 있는 경우 클러스터에 영향을 받습니다.

정확한 일치 대신 접두사를 사용하도록 권한 부여 정책 규칙을 업데이트해야 합니다. 예를 들어 첫 번째 **AuthorizationPolicy** 예제에서 **hosts: ["httpbin.com"]** 을 **hosts: ["httpbin.com:*"]** 로 바꿉니다.

접두사 일치를 사용하는 첫 번째 AuthorizationPolicy 예

```
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:*"]
```

접두사 일치를 사용하는 AuthorizationPolicy 예

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

1.2.2.43. Red Hat OpenShift Service Mesh 2.0.7 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

1.2.2.44. Red Hat OpenShift Dedicated 및 Microsoft Azure Red Hat OpenShift의 Red Hat OpenShift Service Mesh

Red Hat OpenShift Service Mesh는 이제 Red Hat OpenShift Dedicated 및 Microsoft Azure Red Hat OpenShift를 통해 지원됩니다.

1.2.2.45. Red Hat OpenShift Service Mesh 2.0.6 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

1.2.2.46. Red Hat OpenShift Service Mesh 2.0.5 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

1.2.2.47. Red Hat OpenShift Service Mesh 2.0.4 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

중요

CVE-2021-29492 및 CVE-2021-31920 문제를 해결하려면 수동 단계가 완료되어야 합니다.

1.2.2.47.1. CVE-2021-29492 및 CVE-2021-31920에서 필요한 수동 업데이트

Istio에는 경로 기반 권한 부여 규칙이 사용될 때 여러 슬래시 또는 이스케이프된 슬래시 문자(**%2F** 또는 **%5C**)가 있는 HTTP 요청 경로가 잠재적으로 Istio 권한 부여 정책을 우회할 수 있는 원격으로 악용 가능한 취약점이 포함되어 있습니다.

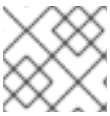
예를 들어 Istio 클러스터 관리자가 경로 **/admin**에 있는 요청을 거부하도록 권한 부여 DENY 정책을 정의한다고 가정합니다. **//admin** URL 경로에 전송된 요청이 권한 부여 정책에서 거부되지 않습니다.

RFC 3986에 따르면 여러 개의 슬래시가 있는 **//admin** 경로는 기술적으로 **/admin**과 다른 경로로 처리되어야 합니다. 그러나 일부 백엔드 서비스는 여러 슬래시를 단일 슬래시로 병합하여 URL 경로를 정규화하도록 선택합니다. 이로 인해 권한 부여 정책(**//admin**이 **/admin**과 일치하지 않음)을 우회할 수 있으며 사용자는 백엔드의 **/admin** 경로에 있는 리소스에 액세스할 수 있습니다. 결과적으로 이는 보안 문제로 나타날 수 있습니다.

ALLOW action + notPaths 필드 또는 **DENY action + paths field** 경로 필드 패턴을 사용하는 권한 부여 정책이 있는 경우 클러스터는 이 취약점의 영향을 받습니다. 이러한 패턴은 예기치 않은 정책 우회에 취약합니다.

다음과 같은 경우 클러스터는 이 취약점의 영향을 받지 않습니다.

- 권한 부여 정책이 없습니다.
- 권한 부여 정책은 **paths** 또는 **notPaths** 필드를 정의하지 않습니다.
- 권한 부여 정책은 **ALLOW action + paths** 필드 또는 **DENY action + notPaths** 필드 패턴을 사용합니다. 이러한 패턴은 정책 우회 대신 예기치 않은 거부를 유발할 수 있습니다. 이러한 경우 업그레이드는 선택 사항입니다.



참고

경로 정규화를 위한 Red Hat OpenShift Service Mesh 구성 위치는 Istio 구성과 다릅니다.

1.2.2.47.2. 경로 정규화 구성 업데이트

Istio 권한 부여 정책은 HTTP 요청의 URL 경로를 기반으로 할 수 있습니다. URI 정규화라고도 하는 경로 정규화는 들어오는 요청의 경로를 수정 및 표준화하여 정규화된 경로를 표준 방식으로 처리할 수 있도록 합니다. 구문적으로 경로 정규화 후에는 다른 경로가 동일할 수 있습니다.

Istio는 권한 부여 정책에 대해 평가하고 요청을 라우팅하기 전에 요청 경로에서 다음 정규화 체계를 지원합니다.

표 1.1. 정규화 체계

옵션	설명	예제	참고
NONE	정규화는 수행되지 않습니다. Envoy가 수신한 모든 항목은 정확히 그대로 모든 백엔드 서비스에 전달됩니다.	**../%2FA../b**는 권한 부여 정책에 의해 평가되고 서비스로 전송됩니다.	이 설정은 CVE-2021-31920에 취약합니다.

옵션	설명	예제	참고
BASE	현재 이는 Istio의 **기본** 설치에 사용되는 옵션입니다. 이로 인해 Envoy 프록시에 **normalize_path** 옵션을 적용하며, RFC 3986에 따라 백슬래시를 슬래시로 변환하는 추가 정규화를 따릅니다.	**/a/../b**는 **/b**로 정규화됩니다. **\da**는 **/da**로 정규화됩니다.	이 설정은 CVE-2021-31920에 취약합니다.
MERGE_SLASHES	*BASE* 정규화 후 슬래시가 병합됩니다.	**/a//b**는 **/a/b**로 정규화됩니다.	CVE-2021-31920을 완화하려면 이 설정으로 업데이트합니다.
DECODE_AND_MERGE_SLASHES	기본적으로 모든 트래픽을 허용할 때 가장 엄격한 설정입니다. 이 설정은 권한 부여 정책 경로를 철저하게 테스트해야 한다는 경고와 함께 권장됩니다. 백분율로 인코딩된 슬래시 및 백슬래시 문자(**%2F**, **%2f**, **%5C** 및 **%5c**)는 **MERGE_SLASHES** 정규화 전에 / 또는 \로 디코딩됩니다.	**/a%2fb**는 **/a/b**로 정규화됩니다.	CVE-2021-31920을 완화하려면 이 설정으로 업데이트합니다. 이 설정은 더 안전하지만 애플리케이션이 중단될 수도 있습니다. 프로덕션에 배포하기 전에 애플리케이션을 테스트합니다.

정규화 알고리즘은 다음 순서로 수행됩니다.

1. 백분율로 디코딩된 **%2F**, **%2f**, **%5C** 및 **%5c**.
2. Envoy의 **normalize_path** 옵션에 의해 구현된 RFC 3986 및 기타 정규화입니다.
3. 슬래시를 병합합니다.

주의

이러한 정규화 옵션은 HTTP 표준 및 일반적인 업계 관행의 권장 사항을 나타내지만 애플리케이션은 원하는 방식으로 URL을 해석할 수 있습니다. 거부 정책을 사용할 때 애플리케이션이 작동하는 방식을 이해해야 합니다.

1.2.2.47.3. 경로 정규화 구성 예

Envoy는 백엔드 서비스의 기대치와 일치하도록 요청 경로를 표준화하여 시스템 보안에 매우 중요합니다. 다음 예제는 시스템을 구성하기 위한 참조로 사용할 수 있습니다. 정규화된 URL 경로 또는 **NONE**이 선택된 경우 원래 URL 경로는 다음과 같습니다.

1. 권한 부여 정책을 확인하는 데 사용됩니다.
2. 백엔드 애플리케이션으로 전달됩니다.

표 1.2. 구성 예

애플리케이션 조건	선택...
프록시를 사용하여 정규화를 수행합니다.	**BASE,MERGE_SLASHES** 또는 **DECODE_AND_MERGE_SLASHES**
RFC 3986을 기반으로 요청 경로를 정규화하고 슬래시를 병합하지 않습니다.	**BASE**
RFC 3986을 기반으로 요청 경로를 정규화하고 슬래시를 병합하지만 백분율로 인코딩된 슬래시를 디코딩하지는 않습니다.	**MERGE_SLASHES**
RFC 3986을 기반으로 요청 경로를 표준화하고, 백분율로 인코딩된 슬래시를 디코딩하고, 슬래시를 병합합니다.	**DECODE_AND_MERGE_SLASHES**
프로세스는 RFC 3986과 호환되지 않는 방식으로 요청 경로를 처리합니다.	**NONE**

1.2.2.47.4. 경로 정규화를 위해 SMCP 구성

Red Hat OpenShift Service Mesh에 대한 경로 정규화를 구성하려면 **ServiceMeshControlPlane**에서 다음을 지정합니다. 시스템 설정을 결정하는 데 도움이 되도록 구성 예제를 사용합니다.

SMCP v2 pathNormalization

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

1.2.2.47.5. 케이스 정규화를 위한 설정

일부 환경에서는 대/소문자를 구분하지 않는 권한 부여 정책의 경로를 사용하는 것이 유용할 수 있습니다. 예를 들어 **https://myurl/get** 및 **https://myurl/GeT**을 동일한 방법으로 처리합니다. 이 경우 아래에 표시된 **EnvoyFilter**를 사용할 수 있습니다. 이 필터는 비교에 사용되는 경로와 애플리케이션에 제공되는 경로를 모두 변경합니다. 이 예제에서 **istio-system** 은 Service Mesh Control Plane 프로젝트의 이름입니다.

EnvoyFilter 를 파일에 저장하고 다음 명령을 실행합니다.

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

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
```

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

1.2.2.48. Red Hat OpenShift Service Mesh 2.0.3의 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

또한 이 릴리스에는 다음과 같은 새로운 기능이 있습니다.

- 지정된 Service Mesh Control Plane 네임스페이스에서 정보를 수집하는 옵션을 **must-gather** 데이터 수집 툴에 추가했습니다. 자세한 내용은 OSSM-351을 참조하십시오.
- 수백 개의 네임스페이스를 사용하여 Service Mesh Control Plane의 성능 향상

1.2.2.49. Red Hat OpenShift Service Mesh 2.0.2 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스에는 IBM Z 및 IBM Power Systems에 대한 지원이 추가되었습니다. 또한 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

1.2.2.50. Red Hat OpenShift Service Mesh 2.0.1 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

1.2.2.51. Red Hat OpenShift Service Mesh 2.0 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스에는 Istio 1.6.5, Jaeger 1.20.0, Kiali 1.24.2, 3scale Istio Adapter 2.0 및 OpenShift Container Platform 4.6에 대한 지원이 추가되었습니다.

또한 이 릴리스에는 다음과 같은 새로운 기능이 있습니다.

- Service Mesh Control Plane의 설치, 업그레이드 및 관리를 간소화합니다.

- Service Mesh Control Plane의 리소스 사용량과 시작 시간을 줄입니다.
- 네트워크를 통한 상호 컨트롤 플레인 통신을 줄임으로써 성능을 향상시킵니다.
  - Envoy의 SDS(Secret Discovery Service)에 대한 지원을 추가합니다. SDS는 Envoy 사이드 카 프록시에 시크릿을 전달하기 위한 보다 안전하고 효율적인 메커니즘입니다.
- 잘 알려진 보안 위험이 있는 Kubernetes Secrets를 사용할 필요가 없습니다.
- 새 인증서를 인식하기 위해 프록시를 다시 시작할 필요가 없으므로 인증서 순환 중에 성능이 향상됩니다.
  - WebAssembly 확장을 사용하여 구축된 Istio의 Telemetry v2 아키텍처에 대한 지원이 추가되었습니다. 이 새로운 아키텍처는 상당한 성능 향상을 가져왔습니다.
  - 서비스 메시 컨트롤 플레인을 보다 쉽게 관리할 수 있도록 간소화된 구성으로 ServiceMeshControlPlane 리소스를 v2로 업데이트합니다.
  - WebAssembly 확장을 기술 프리뷰 기능으로 도입합니다.

1.2.3. 기술 프리뷰

이 릴리스의 일부 기능은 현재 기술 프리뷰 단계에 있습니다. 이러한 실험적 기능은 프로덕션용이 아닙니다.

중요

기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

1.2.4. 사용되지 않거나 삭제된 기능

이전 릴리스에서 사용 가능하던 일부 기능이 더 이상 사용되지 않거나 삭제되었습니다.

더 이상 사용되지 않는 기능은 여전히 OpenShift Container Platform에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

제거된 기능이 더 이상 제품에 존재하지 않습니다.

1.2.4.1. Red Hat OpenShift Service Mesh 2.4에서 더 이상 사용되지 않거나 삭제된 기능

v2.1 **ServiceMeshControlPlane** 리소스는 더 이상 지원되지 않습니다. 고객은 **ServiceMeshControlPlane** 리소스의 이후 버전을 사용하도록 메시 배포를 업그레이드해야 합니다.

Istio OpenShift Routing (IOR)에 대한 지원은 더 이상 사용되지 않으며 향후 릴리스에서 제거됩니다.

Grafana에 대한 지원은 더 이상 사용되지 않으며 향후 릴리스에서 제거됩니다.

Red Hat OpenShift Service Mesh 2.3에서 더 이상 사용되지 않는 다음 암호화 제품군에 대한 지원은 클라이언트와 서버 측의 TLS 협상에 사용되는 기본 암호 목록에서 제거되었습니다. 이러한 암호화 제품군 중 하나가 필요한 서비스에 액세스해야 하는 애플리케이션은 프록시에서 TLS 연결을 시작할 때 연결할 수

없습니다.

- 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

1.2.4.2. Red Hat OpenShift Service Mesh 2.3에서 더 이상 사용되지 않거나 삭제된 기능

다음 암호화 제품군에 대한 지원은 더 이상 사용되지 않습니다. 향후 릴리스에서는 클라이언트와 서버 측의 TLS 협상에서 사용되는 기본 암호 목록에서 제거됩니다.

- 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 버전 2.2에서 더 이상 사용되지 않는 **ServiceMeshExtension** API는 Red Hat OpenShift Service Mesh 버전 2.3에서 제거되었습니다. **ServiceMeshExtension** API를 사용 중인 경우 WebAssembly 확장을 계속 사용하려면 VolumeSnapshot **smPlugin** API로 마이그레이션해야 합니다.

1.2.4.3. Red Hat OpenShift Service Mesh 2.2에서 더 이상 사용되지 않는 기능

ServiceMeshExtension API는 릴리스 2.2로 더 이상 사용되지 않으며 향후 릴리스에서 제거됩니다. **ServiceMeshExtension** API는 릴리스 2.2에서 계속 지원되지만 고객은 새 **wasmPlugin** API로 이동하기 시작해야 합니다.

1.2.4.4. Red Hat OpenShift Service Mesh 2.2에서 제거된 기능

이번 릴리스에서는 모든 플랫폼의 Service Mesh 1.1을 기반으로 하는 Service Mesh Control Plane에 대한 지원 종료를 표시합니다.

1.2.4.5. Red Hat OpenShift Service Mesh 2.1에서 제거된 기능

Service Mesh 2.1에서 Mixer 구성 요소가 제거되었습니다. 버그 수정 및 지원은 Service Mesh 2.0 라이프 사이클 종료를 통해 제공됩니다.

Mixer 플러그인이 활성화된 경우 Service Mesh 2.0.x 릴리스에서 2.1으로 업그레이드하지 않습니다. Mixer 플러그인은 WebAssembly Extensions에 포트해야 합니다.

1.2.4.6. Red Hat OpenShift Service Mesh 2.0에서 더 이상 사용되지 않는 기능

Mixer 구성 요소는 릴리스 2.0에서 더 이상 사용되지 않으며 릴리스 2.1에서 제거될 예정입니다. Mixer를 사용한 확장 구현은 릴리스 2.0에서 계속 지원되지만, 확장은 새로운 WebAssembly 메커니즘으로 마이그레이션되어야 합니다.

다음 리소스 유형은 Red Hat OpenShift Service Mesh 2.0에서 더 이상 지원되지 않습니다.

- **Policy**(authentication.istio.io/v1alpha1)은 더 이상 지원되지 않습니다. 정책 리소스의 특정 구성에 따라 동일한 효과를 달성하기 위해 여러 리소스를 구성해야 할 수 있습니다.
  - **RequestAuthentication**(security.istio.io/v1beta1) 사용
  - **PeerAuthentication**(security.istio.io/v1beta1) 사용
- **ServiceMeshPolicy**(maistra.io/v1)는 더 이상 지원되지 않습니다.
  - 위에서 언급한 것처럼 **RequestAuthentication** 또는 **PeerAuthentication** 을 사용하지만 Service Mesh Control Plane 네임스페이스에 배치합니다.
- **RbacConfig**(rbac.istio.io/v1alpha1)는 더 이상 지원되지 않습니다.
  - **RbacConfig**, **ServiceRole**, 및 **ServiceRoleBinding**을 포함하는 **AuthorizationPolicy**(security.istio.io/v1beta1)로 대체됩니다.
- **ServiceMeshRbacConfig**(maistra.io/v1)는 더 이상 지원되지 않습니다.
  - 위의 대로 **AuthorizationPolicy** 를 사용하지만 Service Mesh Control Plane 네임스페이스에 배치합니다.
- **ServiceRole**(rbac.istio.io/v1alpha1)은 더 이상 지원되지 않습니다.
- **ServiceRoleBinding**(rbac.istio.io/v1alpha1)은 더 이상 지원되지 않습니다.
- Kiali에서는 **login** 및 **LDAP** 전략이 더 이상 사용되지 않습니다. 향후 버전에서는 OpenID 공급자를 사용한 인증을 도입할 예정입니다.

1.2.5. 확인된 문제

이러한 제한 사항은 Red Hat OpenShift Service Mesh에 있습니다.

- Red Hat OpenShift Service Mesh는 아직 IPv6 를 완전히 지원하지 않습니다. 결과적으로 Red Hat OpenShift Service Mesh는 듀얼 스택 클러스터를 지원하지 않습니다.
- 그래프 레이아웃 - 애플리케이션 아키텍처 및 표시할 데이터(그래프 노드 및 상호 작용 수)에 따라 Kiali 그래프의 레이아웃이 다르게 렌더링됩니다. 모든 상황에 적합하게 렌더링되는 단일 레이아웃을 만드는 것이 불가능하지는 않지만 어렵기 때문에 Kiali는 다양한 레이아웃 옵션을 제공합니다. 다른 레이아웃을 선택하려면 **그래프 설정** 메뉴에서 다른 **레이아웃 스키마**를 선택할 수 있습니다.
- Kiali 콘솔에서 분산 추적 플랫폼(Jaeger) 및 Grafana와 같은 관련 서비스에 처음 액세스하는 경

우 인증서를 수락하고 OpenShift Container Platform 로그인 인증 정보를 사용하여 다시 인증해야 합니다. 이것은 프레임워크가 콘솔에 포함된 페이지를 표시하는 방법에 문제가 있기 때문입니다.

- Bookinfo 샘플 애플리케이션은 IBM Power, IBM Z 및 IBM® LinuxONE에 설치할 수 없습니다.
- WebAssembly 확장은 IBM Power, IBM Z 및 IBM® LinuxONE에서 지원되지 않습니다.
- LuaJIT는 IBM Power, IBM Z 및 IBM® LinuxONE에서 지원되지 않습니다.
- IBM Power, IBM Z 및 IBM® LinuxONE에서는 단일 스택 IPv6 지원을 사용할 수 없습니다.

1.2.5.1. 서비스 메시의 알려진 문제

이는 Red Hat OpenShift Service Mesh에서 알려진 문제입니다.

- OSSM-3890 다중 테넌트 메시 배포에서 Gateway API를 사용하도록 시도하면 다음과 유사한 오류 메시지가 생성됩니다.

```
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...
```

다중 테넌트 메시 배포에서 게이트웨이 API를 지원하려면 모든 게이트웨이 API CRD(Custom Resource Definition) 파일이 클러스터에 있어야 합니다.

다중 테넌트 메시 배포에서 CRD 검사가 비활성화되고 Istio는 클러스터에 있는 CRD를 검색할 수 없습니다. 결과적으로 Istio는 지원되는 모든 게이트웨이 API CRD를 확인하려고 하지만 일부 CRD가 없는 경우 오류가 생성됩니다.

서비스 메시 2.3.1 이상 버전은 **v1alpha2** 및 **v1beta1** CRD를 모두 지원합니다. 따라서 게이트웨이 API를 지원하기 위해 다중 테넌트 메시 배포에 두 CRD 버전이 모두 있어야 합니다.

해결방법: 다음 예에서 **kubectl get** 작업은 **v1alpha2** 및 **v1beta1** CRD를 설치합니다. URL에는 추가 **실험적인** 세그먼트가 포함되어 있으며 그에 따라 기존 스크립트를 업데이트합니다.

```
$ 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 이름이 **default** 인 SMCP 배포에 실패합니다. SMCP 오브젝트를 생성하고 해당 version 필드를 v2.3으로 설정하는 경우 오브젝트의 이름은 **기본값** 일 수 없습니다. 이름이 기본값인 경우 컨트롤 플레인은 배포에 실패하고 OpenShift에서 다음 메시지와 함께 **Warning** 이벤트를 생성합니다.

오류 처리 구성 요소 mesh-config: error: [mesh-config/templates/telemetryv2_1.6.yaml: 내부 오류가 발생했습니다. Webhook "rev.validation.istio.io"를 호출하지 못했습니다. 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, istiod-default.istio-system.svc, mesh-config/templates/enable-mesh-permissive.yaml

- OSSM-1655 Kiali 대시보드에 SMCP 에서 mTLS를 활성화한 후 오류가 표시됩니다.

  SMCP에서 spec.security.controlPlane.mtls 설정을 활성화하면 Kiali 콘솔에 다음과 같은 오류 메시지가 표시됩니다. No subsets defined.

- OSSM-1505 이 문제는 OpenShift Container Platform 4.11에서 ServiceMeshExtension 리소스를 사용하는 경우에만 발생합니다. OpenShift Container Platform 4.11에서 ServiceMeshExtension 를 사용하면 리소스가 준비되지 않습니다. oc describe ServiceMeshExtension 를 사용하여 문제를 검사하면 피벗: 함수에서 구현되지 않은 기능 전에 stderr: Error creating mount namespace 오류가 표시됩니다.

  해결방법: ServiceMeshExtension 가 Service Mesh 2.2에서 더 이상 사용되지 않습니다. ServiceMeshExtension 에서 WasmPlugin 리소스로 마이그레이션합니다. 자세한 내용은 ServiceMeshExtension 에서 wasmPlugin 리소스로 마이그레이션 을 참조하십시오.

- OSSM-1396 게이트웨이 리소스에 ServiceMeshControlPlane 을 업데이트할 때 다시 생성하는 대신 spec.externalIPs 설정이 포함된 경우 게이트웨이가 제거되고 다시 생성되지 않습니다.

- OSSM-1168 서비스 메시 리소스가 단일 YAML 파일로 생성되면 Envoy 프록시 사이드카가 Pod에 안정적으로 삽입되지 않습니다. SMCP, SMMR 및 Deployment 리소스가 개별적으로 생성되면 배포가 예상대로 작동합니다.

- OSSM-1115 spec.proxy API의 동시성 필드가 istio-proxy로 전달되지 않았습니다. concurrency 필드는 ProxyConfig 로 설정된 경우 작동합니다. concurrency 필드는 실행할 작업자 스레드 수를 지정합니다. 필드가 0 으로 설정되면 사용 가능한 작업자 스레드 수는 CPU 코어 수와 동일합니다. 필드가 설정되지 않은 경우 사용 가능한 작업자 스레드 수는 기본값인 2 입니다.

  다음 예에서 concurrency 필드는 0 으로 설정됩니다.

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

- OSSM-1052 서비스 메시 컨트롤 플레인에서 ingressgateway에 대해 서비스 ExternalIP 를 구성할 때 서비스가 생성되지 않습니다. SMCP의 스키마에 서비스 매개변수가 누락되어 있습니

다.

해결방법: SMCP 사양에서 게이트웨이 생성을 비활성화하고 서비스, 역할 및 RoleBinding을 포함하여 수동으로 게이트웨이 배포를 완전히 관리합니다.

- OSSM-882 이는 서비스 메시 2.1 및 이전 버전에 적용됩니다. namespace는 accessible_namespace 목록에 있지만 Kiali UI에는 표시되지 않습니다. 기본적으로 Kiali는 이러한 네임스페이스가 일반적으로 메시의 일부가 아닌 internal-use이므로 "kube"로 시작하는 네임스페이스를 표시하지 않습니다.

예를 들어 'akube-a'라는 네임스페이스를 생성하고 서비스 메시 멤버 롤에 추가하면 Kiali UI에 네임스페이스가 표시되지 않습니다. 정의된 제외 패턴의 경우 소프트웨어가 패턴으로 시작하거나 패턴을 포함하는 네임스페이스를 제외합니다.

해결방법: Kiali 사용자 정의 리소스 설정을 변경하여 설정(^)을 접두사로 지정합니다. 예를 들어 다음과 같습니다.

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

- MAISTRA-2692 Mixer를 제거한 경우 Service Mesh 2.0.x에서 정의된 사용자 정의 지표는 2.1에서 사용할 수 없습니다. 사용자 지정 지표는 EnvoyFilter 를 사용하여 구성할 수 있습니다. Red Hat은 명시적으로 문서화된 경우를 제외하고 EnvoyFilter 구성을 지원할 수 없습니다. 이는 기본 Envoy API와 긴밀하게 결합되므로 이전 버전과의 호환성을 유지할 수 없습니다.

- MAISTRA-2648 서비스 메시 확장은 현재 IBM Z에 배포된 메시와 호환되지 않습니다.

- MAISTRA-1959 *2.0으로 마이그레이션* mTLS가 활성화된 경우 Prometheus 스크래핑(spec.addons.prometheus.scrape 를 true로 설정)이 작동하지 않습니다. 또한 Kiali는 mTLS가 비활성화되면 관련 없는 그래프 데이터를 표시합니다.

이 문제는 프록시 구성에서 포트 15020을 제외하여 해결할 수 있습니다. 예를 들면 다음과 같습니다.

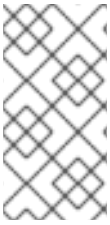
```
spec:
```

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

- MAISTRA-453 새 프로젝트를 생성하고 즉시 pod를 배포하면 사이드카 삽입이 발생하지 않습니다. pod가 생성되기 전에 Operator에서 maistra.io/member-of를 추가하지 못하므로 사이드카 삽입을 수행하려면 pod를 삭제하고 다시 생성해야 합니다.

- MAISTRA-158 동일한 호스트 이름을 참조하는 여러 게이트웨이를 적용하면 모든 게이트웨이가 작동을 중지합니다.

1.2.5.2. Kiali의 확인된 문제



참고

Kiali의 새로운 문제는 OpenShift Service Mesh 프로젝트에서 생성되어야 하며 Component가 Kiali로 설정되어야 합니다.

다음은 Kiali에서 알려진 문제입니다.

- KIALI-2206 처음으로 Kiali 콘솔에 액세스했을 때 Kiali에 대해 캐시된 브라우저 데이터가 없는 경우 Kiali 서비스 상세 정보 페이지의 Metrics 탭에 있는 'Grafana에서 보기' 링크가 잘못된 위치로 리디렉션됩니다. 이 문제가 발생하는 유일한 상황은 Kiali에 처음 액세스하는 경우입니다.

- KIALI-507 Kiali는 Internet Explorer 11을 지원하지 않습니다. 기본 프레임워크가 Internet Explorer를 지원하지 않기 때문입니다. Kiali 콘솔에 액세스하려면 Chrome, Edge, Firefox 또는 Safari 브라우저의 두 가지 최신 버전 중 하나를 사용하십시오.

1.2.6. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- OSSM-4851 이전 버전에서는 runAsGroup,runAsUser 또는 fsGroup 매개변수가 nil 였을 때 메시 내부의 네임스페이스에 새 Pod를 배포하는 Operator에서 오류가 발생했습니다. 이제 nil 값을 방지하기 위해 yaml 검증이 추가되었습니다.

- OSSM-3771 이전에는 SMCP(Service Mesh Control Plane)에 정의된 추가 수신 게이트웨이에 대해 OpenShift 경로를 비활성화할 수 없었습니다. 이제 각 additionalIngress 게이트웨이에 routeConfig 블록을 추가할 수 있으므로 각 게이트웨이에 대해 OpenShift 경로 생성을 활성화하거나 비활성화할 수 있습니다.

이전 릴리스에서 다음 문제가 해결되었습니다.

1.2.6.1. 서비스 메시의 수정된 문제

- OSSM-4197 이전에 'ServiceMeshControlPlane' 리소스의 v2.2 또는 v2.1을 배포한 경우 /etc/cni/multus/net.d/ 디렉터리가 생성되지 않았습니다. 결과적으로 istio-cni Pod가 준비되지 않았으며 istio-cni pod 로그에 다음 메시지가 포함되었습니다.

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

이제 'ServiceMeshControlPlane' 리소스의 v2.2 또는 v2.1을 배포하면 /etc/cni/multus/net.d/ 디렉터리가 생성되고 istio-cni Pod가 준비됩니다.

- OSSM-3993 이전 버전에서는 Kiali는 443 의 표준 HTTPS 포트에서 프록시를 통해 OpenShift OAuth만 지원했습니다. 이제 Kiali는 비표준 HTTPS 포트를 통해 OpenShift OAuth를 지원합니다. 포트를 활성화하려면 spec.server.web_port 필드를 Kiali CR에서 프록시의 비표준 HTTPS 포트로 설정해야 합니다.

- OSSM-3936 이전에는 injection_label_rev 및 injection_label_name 속성의 값이 하드 코딩되었습니다. 이로 인해 Kiali CRD(Custom Resource Definition)에서 사용자 정의 구성이 적용되지 않았습니다. 이제 속성 값이 하드 코딩되지 않습니다. spec.istio_labels 사양에서 injection_label_rev 및 injection_label_name 속성 값을 사용자 지정할 수 있습니다.

- OSSM-3644 이전의 페더레이션 egress-gateway는 네트워크 게이트웨이 끝점의 잘못된 업데이트를 수신하여 추가 엔드 포인트 항목을 발생시킵니다. 이제 페더레이션 게이트웨이가 서버 측에서 업데이트되어 올바른 네트워크 게이트웨이 엔드 포인트를 수신합니다.

- OSSM-3595 이전 버전에서는 SELinux가 유틸리티 iptables-restore 가 /tmp 디렉터리에서 파일을 여는 것을 허용하지 않았기 때문에 RHEL에서 istio-cni 플러그인이 실패하는 경우가 있습니다. 이제 SELinux는 파일을 사용하는 대신 stdin 입력 스트림을 통해 iptables-restore 를 전달합니다.

- OSSM-3586 이전 버전에서는 GCP(Google Cloud Platform) 메타데이터 서버를 사용할 수

없는 경우 Istio 프록시가 시작 속도가 느렸습니다. Istio 1.14.6으로 업그레이드할 때 메타데이터 서버를 사용할 수 없는 경우에도 Istio 프록시는 GCP에서 예상대로 시작됩니다.

- OSSM-3025 Istiod가 준비되지 않을 수 있습니다. 경우에 따라 메시에 여러 멤버 네임스페이스가 포함된 경우 Istiod Pod가 Istiod 내의 교착 상태로 인해 준비되지 않은 경우가 있었습니다. 교착 상태가 해결되어 이제 Pod가 예상대로 시작됩니다.

- OSSM-2493 기본 nodeSelector 및 SMCP 의 허용 오차 는 Kiali에 전달되지 않습니다. SMCP.spec.runtime.defaults 에 추가하는 nodeSelector 및 허용 오차 가 이제 Kiali 리소스에 전달됩니다.

- OSSM-2492 SMCP의 기본 허용 오차는 Jaeger로 전달되지 않습니다. SMCP.spec.runtime.defaults 에 추가하는 nodeSelector 및 허용 오차 가 이제 Jaeger 리소스에 전달됩니다.

- OSSM-2374 ServiceMeshMember 리소스 중 하나를 삭제한 경우 Service Mesh Operator가 ServiceMeshMemberRoll 을 삭제했습니다. 마지막 ServiceMeshMember 를 삭제할 때 이 동작이 예상되지만 삭제된 멤버 외에 멤버가 포함된 경우 Operator는 ServiceMeshMemberRoll 을 삭제하지 않아야 합니다. 이 문제는 해결되어 Operator는 마지막 ServiceMeshMember 리소스가 삭제될 때만 ServiceMeshMemberRoll을 삭제합니다.

- OSSM-2373 로그인 시 OAuth 메타데이터를 가져오려는 오류입니다. 클러스터 버전을 가져오려면 system:anonymous 계정을 사용합니다. 클러스터의 기본 번들 ClusterRoles 및 ClusterRoleBinding을 사용하면 익명 계정이 버전을 올바르게 가져올 수 있습니다. system:anonymous 계정이 클러스터 버전을 가져올 수 있는 권한이 손실되면 OpenShift 인증을 사용할 수 없게 됩니다.

  이 문제는 Kiali SA를 사용하여 클러스터 버전을 가져와 해결되었습니다. 이를 통해 클러스터의 보안을 개선할 수 있습니다.

- OSSM-2371 사용자가 워크로드 세부 정보의 로그 탭의 kebab 메뉴를 통해 프록시 로깅 수준을 변경할 수 있습니다. 이 문제는 Kiali가 "view-only"로 구성된 경우 "Set Proxy Log Level" 아래의 옵션을 비활성화하도록 수정되었습니다.

- OSSM-2344 Istiod 재시작으로 Kiali는 포트 전달 요청으로 CRI-O를 플러딩합니다. 이 문제는 Kiali가 Istiod에 연결할 수 없고 Kiali에서 istiod에 많은 수의 요청을 동시에 발행한 경우 발생했습니다. Kiali는 이제 istiod로 보내는 요청 수를 제한합니다.

- OSSM-2335 추적의chart 플롯 위에 마우스 포인터를 드래그하면 동시 백엔드 요청으로 인해 Kiali 콘솔이 응답하지 않는 경우가 있었습니다.

- OSSM-2221 이전 버전에서는 **ignore-namespace** 레이블이 기본적으로 네임스페이스에 적용되었기 때문에 **ServiceMeshControlPlane** 네임스페이스의 게이트웨이 삽입을 수행할 수 없었습니다.

  v2.4 컨트롤 플레인을 생성할 때 네임스페이스에 더 이상 **ignore-namespace** 레이블이 적용되고 게이트웨이 삽입이 가능합니다.

  다음 예에서 **oc label** 명령은 기존 배포의 네임스페이스에서 **ignore-namespace** 레이블을 제거합니다.

  ```
  $ oc label namespace <istio_system> maistra.io/ignore-namespace-
  ```

  위의 예에서 **<istio_system>**은 **ServiceMeshControlPlane** 네임스페이스의 이름을 나타냅니다.

- OSSM-2053 **Red Hat OpenShift Service Mesh Operator 2.2** 또는 **2.3**을 사용하는 경우 **SMMR** 컨트롤러가 **SMMR.status.configuredMembers** 에서 멤버 네임스페이스를 제거했습니다. 이로 인해 몇 분 동안 멤버 네임스페이스의 서비스를 사용할 수 없게 되었습니다.

  Red Hat OpenShift Service Mesh Operator 2.2 또는 **2.3**을 사용하면 **SMMR** 컨트롤러에서 더 이상 **SMMR.status.configuredMembers** 의 네임스페이스를 제거하지 않습니다. 대신 컨트롤러는 **SMMR.status.pendingMembers** 에 네임스페이스를 추가하여 해당 네임스페이스가 최신 상태가 되지 않음을 나타냅니다. 조정 중에 각 네임스페이스가 **SMCP**와 동기화되므로 네임스페이스는 **SMMR.status.pendingMembers** 에서 자동으로 제거됩니다.

- OSSM-1962 통합 컨트롤러에서 끝점**Slices** 를 사용합니다. 이제 페더레이션 컨트롤러에서 **EndpointSlices** 를 사용하므로 대규모 배포에서 확장성과 성능이 향상됩니다. **PILOT_USE_ENDPOINT_SLICE** 플래그는 기본적으로 활성화되어 있습니다. 플래그를 비활성화하면 페더레이션 배포 사용을 방지할 수 있습니다.

- OSSM-1668 새 필드 **spec.security.jwksResolverCA** 가 버전 **2.1 SMCP** 에 추가되었지만 **2.2.0** 및 **2.2.1** 릴리스에서는 누락되었습니다. 이 필드가 누락된 **Operator** 버전에 문제가 있는 **Operator** 버전에서 업그레이드할 때 **SMCP** 에서 **.spec.security.jwksResolverCA** 필드를 사용할 수 없었습니다.

- OSSM-1325 **istiod Pod**가 충돌하고 다음 오류 메시지: 치명적 오류: 동시 맵 반복 및 쓰기 매핑을 표시합니다.

- OSSM-1211 장애 조치를 위해 Federated 서비스 메시를 구성하면 예상대로 작동하지 않습니다.

  Istiod pilot 로그에는 다음과 같은 오류가 표시됩니다. envoy connection [C289] TLS 오류: 337047686:SSL routine:tls_process_server_certificate:certificate 검증 실패

- OSSM-1099 Kiali 콘솔에 Sorry라는 메시지가 표시되었습니다. 새로 고침을 시도하거나 다른 페이지로 이동합니다.

- SMCP에 정의된 OSSM-1074 Pod 주석은 Pod에 삽입되지 않습니다.

- OSSM-999 Kiali retention가 예상대로 작동하지 않았습니다. 대시보드 그래프에서 일정 시간이 회색이었습니다.

- OSSM-797 Kiali Operator Pod는 Operator를 설치하거나 업데이트하는 동안 CreateContainerConfigError 를 생성합니다.

- OSSM-722 kube 로 시작하는 네임스페이스는 Kiali에서 숨겨집니다.

- OSSM-569 Prometheus istio-proxy 컨테이너에 대한 CPU 메모리 제한은 없습니다. Prometheus istio-proxy 사이드카는 이제 spec.proxy.runtime.container 에 정의된 리소스 제한을 사용합니다.

- OSSM-535 는 SMCP에서 validationMessages를 지원합니다. Service Mesh Control Plane의 ValidationMessages 필드를 이제 True 로 설정할 수 있습니다. 이 명령은 리소스 상태에 대한 로그를 기록합니다. 이 로그는 문제를 해결할 때 유용할 수 있습니다.

- OSSM-449 VirtualService 및 Service로 인해 "도메인에 대한 고유한 값만 허용됩니다. 도메인 중복 항목이 허용됩니다."

- OSSM-419 이름이 유사한 네임스페이스는 서비스 메시 멤버 역할에 네임스페이스를 정의할 수 없는 경우에도 Kiali 네임스페이스 목록에 모두 표시됩니다.

- OSSM-296 Kiali 사용자 지정 리소스(CR)에 상태 구성을 추가할 때 Kiali configmap에 복제되지 않습니다.

- OSSM-291 Kiali 콘솔의 애플리케이션, 서비스 및 워크로드 페이지에서 '필터에서 레이블 삭제' 기능이 작동하지 않습니다.

- OSSM-289 Kiali 콘솔에는 'istio-ingressgateway' 및 'jaeger-query' 서비스에 대한 서비스 세부 정보 페이지에 표시되는 추적이 없습니다. 추적은 Jaeger에 있습니다.

- OSSM-287 Kiali 콘솔에는 그래프 서비스에 표시되는 추적이 없습니다.

- OSSM-285 Kiali 콘솔에 액세스하려고 할 때 "Error trying to get OAuth Metadata"와 같은 오류 메시지가 표시됩니다.

  해결방법: Kiali Pod를 다시 시작합니다.

- MAISTRA-2735 Red Hat OpenShift Service Mesh 버전 2.1에서 SMCP를 조정할 때 Service Mesh Operator가 삭제하는 리소스입니다. 이전에는 Operator에서 다음 라벨을 사용하여 리소스를 삭제했습니다.

  - maistra.io/owner
  - app.kubernetes.io/version

  이제 Operator에서 app.kubernetes.io/managed-by=maistra-istio-operator 레이블도 포함하지 않는 리소스를 무시합니다. 자체 리소스를 생성하는 경우 app.kubernetes.io/managed-by=maistra-istio-operator 레이블을 해당 리소스에 추가하지 않아야 합니다.

- MAISTRA-2687 Red Hat OpenShift Service Mesh 2.1 페더레이션 게이트웨이는 외부 인증서를 사용할 때 전체 인증서 체인을 전송하지 않습니다. Service Mesh 페더레이션 송신 게이트웨이는 클라이언트 인증서만 보냅니다. 페더레이션 수신 게이트웨이는 루트 인증서에 대해서만 알고 있으므로 페더레이션 가져오기 ConfigMap 에 루트 인증서를 추가하지 않으면 클라이언트 인증서를 확인할 수 없습니다.

- MAISTRA-2635 더 이상 사용되지 않는 Kubernetes API를 대체합니다. OpenShift Container Platform 4.8과 호환되도록 하기 위해 apiextensions.k8s.io/v1beta1 API는 Red Hat OpenShift Service Mesh 2.0.8에서 더 이상 사용되지 않습니다.

- MAISTRA-2631 nsenter 바이너리가 존재하지 않기 때문에 podman이 실패하므로 WASM 기능이 작동하지 않습니다. Red Hat OpenShift Service Mesh는 다음 오류 메시지를 생성합니다. Error: CNI 네트워크 플러그인 exec: "nsenter": 실행 파일은 $PATH 에서 찾을 수 없습니다. 컨테이너 이미지에 nsenter가 포함되어 있으며 WASM이 예상대로 작동합니다.

- MAISTRA-2534 istiod에서 JWT 규칙에 지정된 발급자에 대한 JWKS를 가져오기를 시도하면 발급자 서비스가 502로 응답했습니다. 이로 인해 프록시 컨테이너가 준비되지 않아 배포가 중단되었습니다. 커뮤니티 버그 수정이 Service Mesh 2.0.7 릴리스에 포함되어 있습니다.

- MAISTRA-2411 Operator가 ServiceMeshControlPlane에서 spec.gateways.additionalIngress를 사용하여 새 수신 게이트웨이를 생성하면 Operator는 기본 istio-ingressgateway에 대한 추가 수신 게이트웨이에 대한 NetworkPolicy를 생성하지 않습니다. 이로 인해 새 게이트웨이 경로에서 503 응답이 발생합니다.

  해결방법: <istio-system> 네임스페이스에서 NetworkPolicy 를 수동으로 생성합니다.

- MAISTRA-2401 CVE-2021-3586 servicemesh-operator: NetworkPolicy 리소스가 인그레스 리소스에 대해 포트를 잘못 지정했습니다. Red Hat OpenShift Service Mesh에 설치된 NetworkPolicy 리소스가 액세스할 수 있는 포트를 올바르게 지정하지 않았습니다. 이로 인해 모든 pod에서 이러한 리소스의 모든 포트에 액세스할 수 있었습니다. 다음 리소스에 적용되는 네트워크 정책은 영향을 받습니다.

  - Galley
  - Grafana
  - Istiod
  - Jaeger
  - Kiali
  - Prometheus

- Sidecar injector

- MAISTRA-2378 클러스터가 ovs-multitenant와 함께 OpenShift SDN을 사용하도록 구성되고 메시에 다수의 네임스페이스(200+)가 포함된 경우 OpenShift Container Platform 네트워킹 플러그인은 네임스페이스를 빠르게 구성할 수 없습니다. 서비스 메시의 시간이 초과되어 서비스 메시에서 네임스페이스가 지속적으로 드롭된 다음 다시 나열됩니다.

- MAISTRA-2370 listerInformer에서 tombstones를 처리합니다. 업데이트된 캐시 코드베이스는 네임스페이스 캐시에서 집계된 캐시로 이벤트를 변환할 때 tombstones를 처리하지 않아 go 루틴에서 패닉이 발생했습니다.

- MAISTRA-2117 Operator에 선택적 ConfigMap 마운트를 추가합니다. 이제 CSV에 smcp-templates ConfigMap 이 있는 경우 마운트되는 선택적 ConfigMap 볼륨 마운트가 포함됩니다. smcp-templates ConfigMap 이 없으면 마운트된 디렉터리가 비어 있습니다. ConfigMap 을 생성할 때 디렉터리는 ConfigMap 의 항목으로 채워지고 SMCP.spec.profiles 에서 참조할 수 있습니다. Service Mesh Operator를 다시 시작할 필요가 없습니다.

  수정된 CSV와 함께 2.0 Operator를 사용하여 smcp-templates ConfigMap을 마운트하면 Red Hat OpenShift Service Mesh 2.1으로 업그레이드할 수 있습니다. 업그레이드 후에는 CSV를 편집하지 않고도 기존 ConfigMap 및 포함된 프로필을 계속 사용할 수 있습니다. 이전에 ConfigMap을 다른 이름으로 사용한 고객은 업그레이드 후 ConfigMap의 이름을 변경하거나 CSV를 업데이트해야 합니다.

- MAISTRA-2010 AuthorizationPolicy는 request.regex.headers 필드를 지원하지 않습니다. validatingwebhook는 필드가 있는 모든 AuthorizationPolicy를 거부하며, 이를 비활성화한 경우에도 Pilot은 동일한 코드를 사용하여 유효성을 검사하려고 시도하지만 작동하지 않습니다.

- MAISTRA-1979 *2.0으로 마이그레이션* 변환 Webhook는 SMCP.status를 v2에서 v1로 변환할 때 다음과 같은 중요한 필드를 삭제합니다.

  - conditions
  - components
  - observedGeneration

- annotations

Operator를 2.0으로 업그레이드하면 리소스의 maistra.io/v1 버전을 사용하여 SMCP 상태를 판독하는 클라이언트 툴이 중단될 수 있습니다.

또한 oc get servicemeshcontrolplanes.v1.maistra.io를 실행할 때 READY 및 STATUS 열이 비어 있습니다.

- ServiceMeshExtensions에 대한 MAISTRA-1947 *기술 프리뷰* 업데이트는 적용되지 않습니다.

해결방법: ServiceMeshExtensions 를 제거하고 다시 생성합니다.

- MAISTRA-1983 *2.0으로 마이그레이션* 기존의 유효하지 않은 ServiceMeshControlPlane을 사용하여 2.0.0으로 업그레이드하면 쉽게 복구할 수 없습니다. ServiceMeshControlPlane 리소스의 유효하지 않은 항목으로 인해 복구할 수 없는 오류가 발생했습니다. 수정으로 오류를 복구할 수 있습니다. 유효하지 않은 리소스를 삭제하고 새 리소스로 교체하거나 리소스를 편집하여 오류를 수정할 수 있습니다. 리소스 편집에 대한 자세한 내용은 [Red Hat OpenShift Service Mesh 설치 구성]을 참조하십시오.

- MAISTRA-1502 버전 1.0.10에서 CVE가 수정되므로 Grafana의 홈 대시보드 메뉴에서는 Istio 대시보드 를 사용할 수 없습니다. Istio 대시보드에 액세스하려면 탐색 패널에서 대시보드 메뉴를 클릭하고 관리 탭을 선택합니다.

- MAISTRA-1399 Red Hat OpenShift Service Mesh는 더 이상 지원되지 않는 CNI 프로토콜을 설치할 수 없습니다. 지원되는 네트워크 구성이 변경되지 않았습니다.

- MAISTRA-1089 *2.0으로 마이그레이션* 비 컨트롤 플레인 네임스페이스에서 생성된 게이트웨이는 자동으로 삭제됩니다. SMCP 사양에서 게이트웨이 정의를 제거한 후 이러한 리소스를 수동으로 삭제해야 합니다.

- MAISTRA-858 Istio 1.1.x와 관련된 더 이상 사용하지 않는 옵션 및 구성을 설명하는 다음과 같은 Envoy 로그 메시지가 예상됩니다.

[2019-06-03 07:03:28.943][19][warning][misc] [external/envoy/source/common/protobuf/utility.cc:129] Using deprecated option 'envoy.api.v2.listener.Filter.config'. 이 구성은 곧 Envoy에서 삭제될 예정입니다.

- [2019-08-12 22:12:59.001][13][warning][misc] [external/envoy/source/common/protobuf/utility.cc:174] Using deprecated option 'envoy.api.v2.Listener.use_original_dst' from file lds.proto. 이 구성은 곧 Envoy에서 삭제될 예정입니다.

- MAISTRA-806 제거된 Istio Operator pod로 인해 메시 및 CNI가 배포되지 않습니다.

  해결방법: 제어 창을 배포하는 동안 istio-operator Pod가 제거되면 제거된 istio-operator Pod를 삭제합니다.

- MAISTRA-681 서비스 메시 컨트롤 플레인에 네임스페이스가 많은 경우 성능 문제가 발생할 수 있습니다.

- MAISTRA-193 citadel에 대해 상태 확인이 활성화되면 예기치 않은 콘솔 정보 메시지가 표시됩니다.

- Bugzilla 1821432 OpenShift Container Platform 사용자 정의 리소스 세부 정보 페이지의 토글 제어가 CR을 올바르게 업데이트하지 않습니다. OpenShift Container Platform 웹 콘솔의 SMCP(Service Mesh Control Plane) 개요 페이지의 UI 토글 제어가 리소스에서 잘못된 필드를 업데이트하는 경우가 있습니다. SMCP를 업데이트하려면 토글 제어를 클릭하는 대신 YAML 콘텐츠를 직접 편집하거나 명령줄에서 리소스를 업데이트합니다.

1.3. 서비스 메시 이해

Red Hat OpenShift Service Mesh는 서비스 메시에서 네트워크로 연결된 마이크로 서비스에 대해 동작 정보 및 운영 제어용 플랫폼을 제공합니다. Red Hat OpenShift Service Mesh를 사용하면 OpenShift Container Platform 환경에서 마이크로 서비스를 연결, 보호 및 모니터링할 수 있습니다.

1.3.1. 서비스 메시 이해

*서비스 메시*는 분산 마이크로 서비스 아키텍처에서 애플리케이션을 구성하는 마이크로 서비스 네트워크와 이러한 마이크로 서비스 간의 상호 작용입니다. 서비스 메시의 크기와 복잡성이 증가함에 따라 이를 이해하고 관리하는 것이 어려워질 수 있습니다.

오픈 소스 Istio 프로젝트를 기반으로 하는 Red Hat OpenShift Service Mesh는 서비스 코드를 변경할 필요 없이 기존 분산 애플리케이션에 투명 계층을 추가합니다. 마이크로 서비스 간의 모든 네트워크 통신을 차단하는 메시의 관련 서비스에 특수 사이드카 프록시를 배포하여 Red Hat OpenShift Service

Mesh 지원을 서비스에 추가합니다. 서비스 메시 컨트롤 플레인 기능을 사용하여 서비스 메시를 구성하고 관리합니다.

Red Hat OpenShift Service Mesh를 사용하면, 다음과 같은 기능을 제공하는 배포된 서비스 네트워크를 쉽게 생성할 수 있습니다.

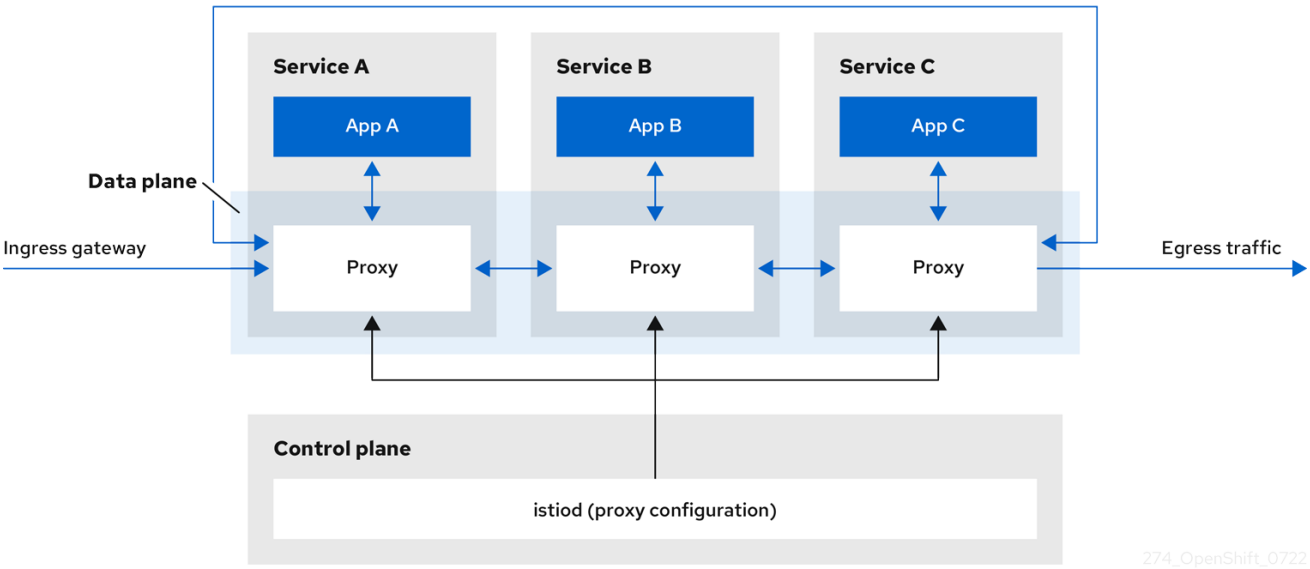
- 검색
- 로드 밸런싱
- 서비스 간 인증
- 장애 복구
- 지표
- 모니터링

Red Hat OpenShift Service Mesh는 다음과 같은 보다 복잡한 운영 기능을 제공합니다:

- **A/B** 테스트
- **Canary** 릴리스
- 액세스 제어
- 엔드 투 엔드 인증

1.3.2. 서비스 메시 아키텍처

서비스 메시 기술은 네트워크 통신 수준에서 작동합니다. 즉, 서비스 메시 구성 요소는 요청을 수정하거나, 리디렉션하거나, 다른 서비스에 새 요청을 생성하여 마이크로 서비스로 들어오고 나가는 트래픽을 캡처하거나 가로챕니다.



높은 수준에서 **Red Hat OpenShift Service Mesh**는 데이터 플레인과 컨트롤 플레인으로 구성됩니다.

데이터 플레인은 **pod**에서 애플리케이션 컨테이너와 함께 실행되며 서비스 메시의 마이크로 서비스 간 인바운드 및 아웃바운드 네트워크 통신을 가로채고 제어하는 지능형 프록시 집합입니다. 데이터 플레인은 인바운드(**ingress**) 및 아웃바운드(**egress**) 네트워크 트래픽을 가로채는 방식으로 구현됩니다. **Istio** 데이터 플레인은 **pod**의 사이드 애플리케이션 컨테이너와 함께 실행되는 **Envoy** 컨테이너로 구성됩니다. **Envoy** 컨테이너는 **pod** 내외의 모든 네트워크 통신을 제어하는 프록시 역할을 합니다.

- **Envoy** 프록시는 데이터 플레인 트래픽과 상호 작용하는 유일한 **Istio** 구성 요소입니다. 서비스 간에 들어오는 모든(**ingress**) 및 발신(**egress**) 네트워크 트래픽은 프록시를 통해 이동합니다. 또한 **Envoy** 프록시는 메시 내에서 서비스 트래픽과 관련된 모든 메트릭을 수집합니다. **Envoy** 프록시는 서비스와 동일한 **Pod**에서 실행되는 사이드카로 배포됩니다. **Envoy** 프록시는 메시 게이트웨이를 구현하는 데도 사용됩니다.
  - 사이드카 프록시는 워크로드 인스턴스에 대한 인바운드 및 아웃바운드 통신을 관리합니다.
  - 게이트웨이는 들어오거나 나가는 **HTTP/TCP** 연결을 수신하는 로드 밸런서 장치로 작동하는 프록시입니다. 게이트웨이 구성은 서비스 워크로드와 함께 실행되는 사이드카 **Envoy** 프록시가 아닌, 메시의 에지에서 실행되는 독립 실행형 **Envoy** 프록시에 적용됩니다. 게이트웨이를 사용하여 메시에 대한 인바운드 및 아웃바운드 트래픽을 관리하여 메시에 들어오거나 나가려는 트래픽을 지정할 수 있습니다.
    -

Ingress-gateway - Ingress 컨트롤러라고도 하는 **Ingress** 게이트웨이는 서비스 메시를 입력하는 트래픽을 수신하고 제어하는 전용 **Envoy** 프록시입니다. **Ingress** 게이트웨이를 사용하면 모니터링 및 경로 규칙과 같은 기능을 클러스터로 들어오는 트래픽에 적용할 수 있습니다.

- **egress-gateway -** 송신 컨트롤러라고도 하는 **Egress** 게이트웨이는 서비스 메시를 나가는 트래픽을 관리하는 전용 **Envoy** 프록시입니다. **Egress Gateway**를 사용하면 모니터링 및 경로 규칙과 같은 기능이 메시를 종료하는 트래픽에 적용할 수 있습니다.

컨트롤 플레인은 데이터 플레인을 구성하는 프록시를 관리하고 구성합니다. 구성에 대한 권한 있는 소스이며 액세스 제어 및 사용 정책을 관리하고 서비스 메시의 프록시에서 메트릭을 수집합니다.

- **Istio** 컨트롤 플레인은 이전의 여러 컨트롤 플레인 구성 요소(**Citadel, Galley, Pilot**)를 단일 바이너리로 통합하는 **Istiod**로 구성됩니다. **Istiod**는 서비스 검색, 구성 및 인증서 관리를 제공합니다. 고급 라우팅 규칙을 **Envoy** 구성으로 변환하고 런타임 시 사이드카로 전달합니다.
  - **Istiod**는 **CA**(인증 기관) 역할을 하며 데이터 플레인에서 보안 **mTLS** 통신을 지원하는 인증서를 생성할 수 있습니다. 이를 위해 외부 **CA**를 사용할 수도 있습니다.
  - **Istiod**는 **OpenShift** 클러스터에 배포된 워크로드에 사이드카 프록시 컨테이너를 삽입하는 역할을 합니다.

Red Hat OpenShift Service Mesh는 **istio-operator**를 사용하여 컨트롤 플레인 설치를 관리합니다. ***Operator***는 **OpenShift** 클러스터에서 공통 활동을 구현하고 자동화할 수 있는 소프트웨어입니다. 컨트롤러 역할을 하여 클러스터에서 원하는 오브젝트 상태(이 경우 **Red Hat OpenShift Service Mesh** 설치)를 설정하거나 변경할 수 있습니다.

또한 **Red Hat OpenShift Service Mesh**는 다음 **Istio** 추가 기능도 제품의 일부로 번들로 제공합니다.

- **Kiali - Kiali**는 **Red Hat OpenShift Service Mesh**의 관리 콘솔입니다. 대시보드, 관찰 기능, 강력한 구성 및 유효성 검사 기능을 제공합니다. 트래픽 토폴로지를 유추하고 서비스 메시의 구조를 표시하고 메시의 상태를 표시합니다. **Kiali**는 자세한 메트릭, 강력한 검증, **Grafana**에 대한 액세스, 분산 추적 플랫폼(**Jaeger**)과의 강력한 통합을 제공합니다.
- **Prometheus - Red Hat OpenShift Service Mesh**는 **Prometheus**를 사용하여 서비스의 원격 분석 정보를 저장합니다. **Kiali**는 **Prometheus**를 사용하여 메트릭, 상태 및 메시 토폴로지를 가져옵니다.

- Jaeger - Red Hat OpenShift Service Mesh는 분산 추적 플랫폼(Jaeger)을 지원합니다. Jaeger는 여러 서비스 간에 단일 요청과 관련된 추적을 중앙 집중화하고 표시하는 오픈소스 추적 기능 서버입니다. 분산 추적 플랫폼(Jaeger)을 사용하면 마이크로서비스 기반 분산 시스템을 모니터링하고 문제를 해결할 수 있습니다.

- Elasticsearch - Elasticsearch는 오픈 소스, 분산형 JSON 기반 검색 및 분석 엔진입니다. 분산 추적 플랫폼(Jaeger)은 영구 스토리지에 Elasticsearch를 사용합니다.

- Grafana - Grafana는 메시 관리자에게 Istio 데이터에 대한 고급 쿼리 및 메트릭 분석 및 대시보드를 제공합니다. 선택적으로 Grafana를 사용하여 서비스 메시 메트릭을 분석할 수 있습니다.

다음 Istio 통합은 Red Hat OpenShift Service Mesh에서 지원됩니다.

- 3scale - Istio는 Red Hat 3scale API Management 솔루션과의 선택적 통합을 제공합니다. 2.1 이전 버전의 경우 이 통합은 3scale Istio 어댑터를 통해 달성되었습니다. 버전 2.1 이상의 경우 3scale 통합은 WebAssembly 모듈을 통해 이루어집니다.

3scale 어댑터 설치 방법에 대한 자세한 내용은 3scale Istio 어댑터 설명서를 참조하십시오.

1.3.3. Kiali 이해

Kiali는 서비스 메시의 마이크로 서비스와 해당 연결 방법을 표시하여 서비스 메시를 시각화할 수 있습니다.

1.3.3.1. Kiali 개요

Kiali는 OpenShift Container Platform에서 실행 중인 서비스 메시에 대한 관찰 기능을 제공합니다. Kiali는 Istio 서비스 메시를 정의하고 검증하며 관찰하는 데 도움이 됩니다. 이를 통해 토폴로지를 유추하여 서비스 메시의 구조를 이해하고 서비스 메시의 상태에 대한 정보를 제공할 수 있습니다.

Kiali는 회로 차단기, 요청 속도, 대기 시간, 트래픽 흐름 그래프와 같은 기능에 대한 가시성을 제공하는 네임스페이스의 대화형 그래프 보기를 실시간으로 제공합니다. Kiali는 애플리케이션에서 서비스 및 워크로드에 이르기까지 다양한 수준의 구성 요소에 대한 통찰력을 제공하며, 선택한 그래프 노드 또는 에지에서 상황별 정보에 대한 상호 작용과 차트를 표시할 수 있습니다. Kiali는 게이트웨이, 대상 규칙, 가상 서비스, 메시 정책 등과 같은 Istio 구성의 유효성을 확인하는 기능도 제공합니다. Kiali는 자세한 지표를 제공하며 고급 쿼리에 기본 Grafana 통합이 가능합니다. Jaeger를 Kiali 콘솔에 통합하면 분산 추적이 제공됩니다.

Kiali는 기본적으로 Red Hat OpenShift Service Mesh의 일부로 설치됩니다.

1.3.3.2. Kiali 아키텍처

Kiali는 오픈 소스 Kiali 프로젝트를 기반으로 합니다. Kiali는 Kiali 애플리케이션과 Kiali 콘솔이라는 두 가지 구성 요소로 구성됩니다.

- Kiali 애플리케이션(백엔드) - 이 구성 요소는 컨테이너 애플리케이션 플랫폼에서 실행되고 서비스 메시 구성 요소와 통신하며, 데이터를 검색 및 처리하고, 이 데이터를 콘솔에 노출합니다. Kiali 애플리케이션에는 스토리지가 필요하지 않습니다. 클러스터에 애플리케이션을 배포할 때 구성은 ConfigMaps 및 시크릿에 설정됩니다.

- Kiali 콘솔(프론트엔드) - Kiali 콘솔은 웹 애플리케이션입니다. Kiali 애플리케이션은 Kiali 콘솔을 제공하며 이를 사용자에게 표시하기 위해 데이터의 백엔드를 쿼리합니다.

또한 Kiali는 컨테이너 애플리케이션 플랫폼과 Istio에서 제공하는 외부 서비스 및 구성 요소에 따라 달라집니다.

- Red Hat Service Mesh(Istio) - Istio는 Kiali 요구 사항입니다. Istio는 서비스 메시를 제공하고 제어하는 구성 요소입니다. Kiali와 Istio를 별도로 설치할 수 있지만 Kiali는 Istio에 따라 달라지며 Istio가 존재하지 않는 경우 작동하지 않습니다. Kiali는 Prometheus 및 클러스터 API를 통해 노출되는 Istio 데이터와 구성을 검색해야 합니다.

- Prometheus - 전용 Prometheus 인스턴스는 Red Hat OpenShift Service Mesh 설치의 일부로 포함되어 있습니다. Istio Telemetry가 활성화되면 지표 데이터가 Prometheus에 저장됩니다. Kiali는 이 Prometheus 데이터를 사용하여 메시 토폴로지 확인, 지표 표시, 상태 계산, 가능한 문제 표시 등의 작업을 수행합니다. Kiali는 Prometheus와 직접 통신하고 Istio Telemetry에서 사용하는 데이터 스키마를 가정합니다. Prometheus는 Istio 종속성 및 Kiali에 대한 하드 종속성이며, 대부분의 Kiali 기능은 Prometheus없이 작동하지 않습니다.

- 클러스터 API - Kiali는 서비스 메시 구성을 가져와 해결하기 위해 OpenShift Container Platform(클러스터 API)의 API를 사용합니다. Kiali는 클러스터 API를 쿼리하여 네임스페이스, 서비스, 배포, pod 및 기타 엔티티에 대한 정의를 검색합니다. 또한 Kiali는 다른 클러스터 엔티티 간의 관계를 해결하기 위해 쿼리를 만듭니다. 클러스터 API는 가상 서비스, 대상 규칙, 경로 규칙, 게이트웨이, 할당량 등과 같은 Istio 구성을 검색하도록 쿼리합니다.

- Jaeger - Jaeger는 선택 사항이지만 Red Hat OpenShift Service Mesh의 일부로 설치됩니다. 기본 Red Hat OpenShift Service Mesh 설치의 일부로 분산 추적 플랫폼(Jaeger)을 설치할 때 Kiali 콘솔에는 분산 추적 데이터를 표시하는 탭이 포함됩니다. Istio의 분산 추적 기능을 비활

성화하면 추적 데이터를 사용할 수 없습니다. 또한 사용자가 추적 데이터를 보기 위해 **Service Mesh Control Plane**이 설치된 네임스페이스에 대한 액세스 권한이 있어야 합니다.

- **Grafana - Grafana**는 선택 사항이지만 **Red Hat OpenShift Service Mesh**의 일부로 설치됩니다. 사용 가능한 경우, **Kiali**의 지표 페이지에 사용자를 **Grafana**의 동일한 지표로 안내하는 링크가 표시됩니다. 사용자가 **Grafana** 대시보드에 대한 링크를 보고 **Grafana** 데이터를 보려면 **Service Mesh Control Plane**이 설치된 네임스페이스에 대한 액세스 권한이 있어야 합니다.

1.3.3.3. Kiali 기능

Kiali 콘솔은 **Red Hat Service Mesh**와 통합되어 다음 기능을 제공합니다.

- 상태 **-** 애플리케이션, 서비스 또는 워크로드에 대한 문제를 빠르게 식별합니다.
- 토폴로지 **-** 애플리케이션, 서비스 또는 워크로드가 **Kiali** 그래프를 통해 통신하는 방식을 시각화합니다.
- 지표 **-** 사전 정의된 지표 대시 보드를 통해 **Go, Node.js. Quarkus, Spring Boot, Thorntail, Vert.x**에 대한 서비스 메시 및 애플리케이션 성능을 차트로 작성할 수 있습니다. 또한 사용자 정의 대시보드를 생성할 수도 있습니다.
- 추적 **- Jaeger**와의 통합을 통해 애플리케이션을 구성하는 다양한 마이크로 서비스를 통해 요청 경로를 따를 수 있습니다.
- 검증 **-** 가장 일반적인 **Istio** 오브젝트에 대한 고급 검증(대상 규칙, 서비스 항목, 가상 서비스 등)을 수행합니다.
- 구성 **-** 마법사를 사용하거나 **Kiali** 콘솔의 **YAML** 편집기에서 직접 **Istio** 라우팅 구성을 생성, 업데이트 및 삭제할 수 있는 옵션입니다.

1.3.4. 분산 추적 이해

사용자가 애플리케이션에서 작업을 수행할 때마다 응답을 생성하기 위해 참여하도록 다양한 서비스를 필요로 할 수 있는 아키텍처에 의해 요청이 실행됩니다. 이 요청의 경로는 분산 트랜잭션입니다. 분산 추적 플랫폼(**Jaeger**)을 사용하면 애플리케이션을 구성하는 다양한 마이크로 서비스를 통한 요청 경로를 따르는 분산 추적을 수행할 수 있습니다.

분산 추적은 분산 트랜잭션에 있는 전체 이벤트 체인을 이해하기 위해 일반적으로 다양한 프로세스 또는 호스트에서 실행되는 다양한 작업 단위에 대한 정보를 결합하는 데 사용되는 기술입니다. 분산 추적을 통해 개발자는 대규모 서비스 지향 아키텍처에서 호출 흐름을 시각화할 수 있습니다. 직렬화, 병렬 처리 및 대기 시간 소스를 이해하는 데 유용할 수 있습니다.

분산 추적 플랫폼(Jaeger)은 마이크로 서비스 스택 전체에 걸쳐 개별 요청 실행을 기록하고 이를 추적으로 제공합니다. 추적은 시스템을 통한 데이터/실행 경로입니다. 엔드 투 엔드 추적은 하나 이상의 범위로 구성됩니다.

기간은 작업 이름, 작업의 시작 시간 및 기간이 있는 논리적 작업 단위를 나타냅니다. 기간은 중첩되어 인과 관계를 모델링하도록 주문될 수 있습니다.

1.3.4.1. 분산 추적 개요

서비스 소유자는 분산 추적을 사용하여 서비스 아키텍처에 대한 통찰력을 수집하기 위해 서비스를 계측할 수 있습니다. **Red Hat OpenShift** 분산 추적 플랫폼을 사용하여 최신 클라우드 네이티브, 마이크로서비스 기반 애플리케이션의 구성 요소 간의 상호 작용을 모니터링, 네트워크 프로파일링 및 문제 해결에 사용할 수 있습니다.

분산 추적 플랫폼을 사용하면 다음 기능을 수행할 수 있습니다.

- 분산 트랜잭션 모니터링
- 성능 및 대기 시간 최적화
- 근본 원인 분석 수행

분산 추적 플랫폼은 다음 세 가지 구성 요소로 구성됩니다.

- **Red Hat OpenShift distributed tracing platform(Jaeger)** 은 오픈 소스 **Jaeger** 프로젝트를 기반으로 합니다.
- **Red Hat OpenShift distributed tracing platform(Tempo)** 은 오픈 소스 **Grafana Tempo** 프로젝트를 기반으로 합니다.

- Red Hat OpenShift distributed tracing 데이터 수집 은 오픈 소스 OpenTelemetry 프로젝트를 기반으로 합니다.

1.3.4.2. Red Hat OpenShift distributed tracing 플랫폼 아키텍처

Red Hat OpenShift distributed tracing 플랫폼은 함께 작동하여 추적 데이터를 수집, 저장 및 표시하는 여러 구성 요소로 구성됩니다.

- Red Hat OpenShift distributed tracing platform(Jaeger) - 이 구성 요소는 오픈 소스 Jaeger 프로젝트를 기반으로 합니다.
  - 클라이언트 (Jaeger 클라이언트, Tracer, Reporter, 조정된 애플리케이션, 클라이언트 라이브러리)- 분산 추적 플랫폼(Jaeger) 클라이언트는 OpenTracing API의 언어별 구현입니다. 수동으로 또는 이미 OpenTracing과 통합된 Camel(Fuse), Spring Boot(RHOAR), MicroProfile(RHOAR/T©tail), Wildfly(EAP) 등의 다양한 기존 오픈 소스 프레임워크를 사용하여 분산 추적에 대해 애플리케이션을 조정하는 데 사용할 수 있습니다.
  - 에이전트 (Jaeger agent, Server Queue, Processor Workers) - 분산 추적 플랫폼(Jaeger) 에이전트는 UDP(User Datagram Protocol)를 통해 전송되는 기간을 수신 대기하는 네트워크 데몬입니다. 에이전트는 조정된 애플리케이션과 동일한 호스트에 배치되어야 합니다. 일반적으로 Kubernetes와 같은 컨테이너 환경에서 사이드카를 보유하여 수행됩니다.
  - Jaeger 수집기 (Collector, Queue, Workers) - Jaeger 수집기는 기간을 수신하여 처리를 위해 내부 큐에 배치합니다. 이를 통해 Jaeger 수집기는 기간이 스토리지로 이동할 때까지 기다리는 대신 클라이언트/에이전트로 즉시 돌아갈 수 있습니다.
  - 스토리지(데이터 저장소) - 수집기에는 영구 스토리지 백엔드가 필요합니다. Red Hat OpenShift distributed tracing platform(Jaeger)에는 스토리지 범위를 위한 플러그형 메커니즘이 있습니다. 이 릴리스에서 지원되는 유일한 스토리지는 Elasticsearch입니다.
  - 쿼리(쿼리 서비스) - 쿼리는 스토리지에서 추적을 검색하는 서비스입니다.
  - Ingester (Ingester Service) - Red Hat OpenShift distributed tracing 플랫폼은 수집기와 실제 Elasticsearch 백업 스토리지 간의 버퍼로 Apache Kafka를 사용할 수 있습니다. Ingester는 Kafka에서 데이터를 읽고 Elasticsearch 스토리지 백엔드에 쓰는 서비스입니다.
  - Jaeger 콘솔 - Red Hat OpenShift distributed tracing Platform (Jaeger) 사용자 인

터페이스를 사용하면 분산 추적 데이터를 시각화할 수 있습니다. 검색 페이지에서 추적을 찾고 개별 추적을 구성하는 기간의 세부 사항을 확인할 수 있습니다.

- **Red Hat OpenShift distributed tracing platform(Tempo) -** 이 구성 요소는 오픈 소스 Grafana Tempo 프로젝트를 기반으로 합니다.
  - 게이트웨이는 인증, 권한 부여 및 요청을 배포 또는 쿼리 프런트 엔드 서비스로 처리합니다. **Gateway - The Gateway handles authentication, authorization, and forwarding requests to the Cryostat or Query front-end service.**
  - 배포자 **-** 배포자는 **Jaeger, OpenTelemetry** 및 **Zipkin**을 비롯한 여러 형식으로 기간을 허용합니다. 이 경로는 **traceID** 를 해시하고 분산 일관된 해시 링을 사용하여 **Ingesters**로 구성됩니다.
  - **Ingester - Ingester**는 추적을 블록으로 배치하고, **bloom** 필터와 인덱스를 생성한 다음 모두 백엔드로 플러시합니다.
  - 쿼리 프런트 엔드 **-** 쿼리 프런트 엔드는 들어오는 쿼리에 대한 검색 공간을 분할합니다.**Query Frontend is responsible for sharding the search space for an incoming query.** 그런 다음 검색 쿼리가 **Queriers**로 전송됩니다. 쿼리 프런트 엔드 배포는 **Tempo** 쿼리 사이드카를 통해 **Jaeger UI**를 노출합니다.
  - **Querier - Querier**는 **Ingesters** 또는 백엔드 스토리지에서 요청된 추적 **ID**를 찾을 책임이 있습니다. 매개변수에 따라 **Ingesters**를 쿼리하고 백엔드에서 오브젝트 스토리지의 검색 블록으로 **Ingesters**를 가져올 수 있습니다.
  - **Cryo stator -** 백엔드 스토리지로 또는 백엔드 스토리지에서 차단하여 총 블록 수를 줄입니다.
- **Red Hat OpenShift** 분산 추적 데이터 수집 **-** 이 구성 요소는 오픈 소스 OpenTelemetry 프로젝트를 기반으로 합니다.
  - **OpenTelemetry Collector - OpenTelemetry** 수집기는 원격 분석 데이터를 수신, 처리 및 내보낼 수 있는 공급업체와 무관한 방법입니다. **OpenTelemetry** 수집기는 하나 이상의 오픈 소스 또는 상용 백엔드로 보내는 **Jaeger** 및 **Prometheus**와 같은 오픈 소스 관찰성 데이터 형식을 지원합니다. 수집기는 기본 위치 계측 라이브러리로 **Telemetry** 데이터를 내보냅니다.

1.3.4.3. Red Hat OpenShift distributed tracing 플랫폼 기능

Red Hat OpenShift distributed tracing 플랫폼은 다음과 같은 기능을 제공합니다.

- Kiali와의 통합 - 올바르게 구성된 경우 Kiali 콘솔에서 분산 추적 플랫폼 데이터를 볼 수 있습니다.
- 높은 확장성 - 분산 추적 플랫폼 백엔드는 단일 장애 지점이 없으며 비즈니스 요구 사항에 따라 확장할 수 있도록 설계되었습니다.
- 분산 컨텍스트 전파 - 다른 구성 요소의 데이터를 함께 연결하여 완전한 엔드 투 엔드 추적을 만들 수 있습니다.
- Zipkin과의 역호환성 - Red Hat OpenShift distributed tracing 플랫폼에는 Zipkin의 드롭인 대체 기능으로 사용할 수 있는 API가 있지만 Red Hat은 이 릴리스에서 Zipkin 호환성을 지원하지 않습니다.

1.3.5. 다음 단계

- OpenShift Container Platform 환경에 Red Hat OpenShift Service Mesh를 설치할 준비를 합니다.

1.4. 서비스 메시 배포 모델

Red Hat OpenShift Service Mesh는 비즈니스 요구 사항에 가장 적합하도록 다양한 방식으로 결합할 수 있는 여러 가지 배포 모델을 지원합니다.

Istio에서 테넌트는 배포된 워크로드 집합에 대한 공통 액세스 및 권한을 공유하는 사용자 그룹입니다. 테넌트를 사용하여 서로 다른 팀 간에 격리 수준을 제공할 수 있습니다. NetworkPolicies,AuthorizationPolicies, istio.io 또는 서비스 리소스에서 exportTo 주석을 사용하여 다른 테넌트에 대한 액세스를 분리할 수 있습니다.

1.4.1. cluster-Wide (Single Tenant) 메시 배포 모델

클러스터 전체 배포에는 전체 클러스터의 리소스를 모니터링하는 Service Mesh Control Plane이 포함되어 있습니다. 컨트롤 플레인이 모든 네임스페이스에서 단일 쿼리를 사용하여 Istio 및 Kubernetes 리소스를 모니터링한다는 점에서 전체 클러스터의 모니터링 리소스는 Istio 기능과 유사합니다. 결과적으로 클러스터 전체 배포는 API 서버로 전송되는 요청 수를 줄입니다.

Istio와 유사하게 클러스터 전체 메시에는 기본적으로 istio-injection=enabled namespace 라벨이 있는 네임스페이스가 포함됩니다. ServiceMeshMemberRoll 리소스의 spec.labelSelectors 필드를 수정하여 이 레이블을 변경할 수 있습니다.

1.4.2. 다중 테넌트 배포 모델

Red Hat OpenShift Service Mesh는 기본적으로 멀티 테넌시용으로 구성된 ServiceMeshControlPlane 을 설치합니다. Red Hat OpenShift Service Mesh는 다중 테넌트 Operator 를 사용하여 Service Mesh Control Plane 라이프사이클을 관리합니다. 메시 내에서 네임스페이스는 테넌시에 사용됩니다.

Red Hat OpenShift Service Mesh는 ServiceMeshControlPlane 리소스를 사용하여 기본적으로 리소스가 포함된 네임스페이스로 범위가 제한된 메시 설치를 관리합니다. ServiceMeshMemberRoll 및 ServiceMeshMember 리소스를 사용하여 추가 네임스페이스를 메시에 포함합니다. 네임스페이스는 단일 메시에만 포함될 수 있으며 단일 OpenShift 클러스터에 여러 메시를 설치할 수 있습니다.

일반적인 서비스 메시 배포에서는 단일 서비스 메시 컨트롤 플레인을 사용하여 메시의 서비스 간 통신을 구성합니다. Red Hat OpenShift Service Mesh는 테넌트당 하나의 컨트롤 플레인과 하나의 메시가 있고 클러스터 내에 여러 개의 독립적인 컨트롤 플레인이 있는 "소프트 멀티 테넌시"를 지원합니다. 다중 테넌트 배포는 서비스 메시에 액세스하고 다른 컨트롤 플레인 인스턴스에서 서비스 메시를 격리할 수 있는 프로젝트를 지정합니다.

클러스터 관리자는 모든 Istio 컨트롤 플레인에서 제어 및 가시성을 가져오고, 테넌트 관리자는 특정 서비스 메시, Kiali, Jaeger 인스턴스만 제어합니다.

팀에 지정된 네임스페이스 또는 네임스페이스 집합에만 워크로드를 배포할 수 있는 권한을 부여할 수 있습니다. 서비스 메시 관리자가 mesh-user 역할을 부여하는 경우 사용자는 ServiceMeshMember 리소스를 생성하여 ServiceMeshMemberRoll 에 네임스페이스를 추가할 수 있습니다.

1.4.3. Multimesh 또는 연합 배포 모델

페더레이션 은 별도의 관리 도메인에서 관리하는 별도의 메시 간에 서비스 및 워크로드를 공유할 수 있는 배포 모델입니다.

Istio 다중 클러스터 모델을 사용하려면 개별 메시가 상주하는 모든 Kubernetes API 서버에 대한 메시와 원격 액세스 간에 높은 수준의 신뢰가 필요합니다. Red Hat OpenShift Service Mesh 페더레이션은 메시 간 *최소* 신뢰를 가정하는 Service Mesh의 다중 클러스터 구현에 대한 의견 접근 방식을 취합니다.

통합 메시 는 단일 메시로 작동하는 메시 그룹입니다. 각 메시의 서비스는 고유한 서비스(예: 다른 메시

에서 서비스를 가져와 서비스 추가)가 될 수 있으며, 메시 전체에 동일한 서비스에 추가 워크로드를 제공하여 고가용성을 제공하거나 두 가지의 조합을 제공할 수 있습니다. 페더레이션 메시에 가입된 모든 메시는 개별적으로 관리되며, 페더레이션으로 내보내지 않고 다른 메시에서 가져온 서비스를 명시적으로 구성해야 합니다. 인증서 생성, 메트릭 및 추적 수집과 같은 지원 기능은 각 메시에서 로컬로 유지됩니다.

1.5. 서비스 메시 및 ISTIO 차이점

Red Hat OpenShift Service Mesh는 OpenShift Container Platform에 배포할 때 추가 기능을 제공하거나, 차이점을 처리하기 위한 Istio 설치와는 다릅니다.

1.5.1. Istio와 Red Hat OpenShift Service Mesh 간의 차이점

다음 기능은 서비스 메시와 Istio에서 다릅니다.

1.5.1.1. 명령줄 도구

Red Hat OpenShift Service Mesh의 명령줄 도구는 oc입니다. Red Hat OpenShift Service Mesh는 istioctl을 지원하지 않습니다.

1.5.1.2. 설치 및 업그레이드

Red Hat OpenShift Service Mesh는 Istio 설치 프로필을 지원하지 않습니다.

Red Hat OpenShift Service Mesh는 서비스 메시의 카나리아 업그레이드를 지원하지 않습니다.

1.5.1.3. 자동 삽입

업스트림 Istio 커뮤니티 설치는 레이블을 지정한 프로젝트 내의 pod에 사이드카를 자동으로 삽입합니다.

Red Hat OpenShift Service Mesh는 사이드카를 Pod에 자동으로 삽입하지 않지만 프로젝트에 레이블을 지정하지 않고 주석을 사용하여 삽입해야 합니다. 이 방법은 더 적은 권한이 필요하며 빌더 Pod와 같은 다른 OpenShift Container Platform 기능과 충돌하지 않습니다. 자동 삽입을 활성화하려면 *자동 사이드카 삽입 섹션에 설명된 대로* ***sidecar.istio.io/inject*** 레이블 또는 주석을 지정합니다.

표 1.3. 사이드카 삽입 라벨 및 주석 설정

	업스트림 Istio	Red Hat OpenShift Service Mesh
네임스페이스 라벨	"활성화" 및 "비활성화" 지원	"비활성화" 지원
Pod 라벨	"true" 및 "false" 지원	"true" 및 "false" 지원
Pod 주석	"false"만 지원	"true" 및 "false" 지원

1.5.1.4. Istio 역할 기반 액세스 제어 기능

역할 기반 액세스 제어(RBAC)는 서비스에 대한 액세스를 제어하는 데 사용할 수 있는 메커니즘을 제공합니다. 사용자 이름별로, 또는 속성 집합을 지정하여 제목을 식별하고 그에 따라 액세스 제어를 적용할 수 있습니다.

업스트림 Istio 커뮤니티 설치에는 정확한 헤더 일치를 수행하거나, 헤더에서 와일드카드를 일치시키거나, 특정 접두사 또는 접미사가 포함된 헤더를 확인하는 옵션이 포함되어 있습니다.

Red Hat OpenShift Service Mesh는 정규식을 사용하여 요청 헤더를 일치시키는 기능을 확장합니다. 정규식이 있는 request.regex.headers의 속성 키를 지정합니다.

요청 헤더와 일치하는 업스트림 Istio 커뮤니티 예

```
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 Mesh는 BoringSSL을 OpenSSL로 대체합니다. OpenSSL은 SSL(Secure Sockets Layer) 및 TLS(Transport Layer Security) 프로토콜의 오픈 소스 구현이 포함된 소프트웨어 라이브러리입니다. Red Hat OpenShift Service Mesh 프록시 바이너리는 기본 Red Hat Enterprise Linux 운영 체제에서 OpenSSL 라이브러리(libssl 및 libcrypto)를 동적으로 연결합니다.

1.5.1.6. 외부 워크로드

Red Hat OpenShift Service Mesh는 베어 메탈 서버에서 OpenShift 외부에서 실행되는 가상 머신과 같은 외부 워크로드를 지원하지 않습니다.

1.5.1.7. 가상 머신 지원

OpenShift Virtualization을 사용하여 OpenShift에 가상 머신을 배포할 수 있습니다. 그런 다음 mTLS 또는 AuthorizationPolicy와 같은 메시 정책을 메시의 일부인 다른 Pod와 마찬가지로 이러한 가상 머신에 적용할 수 있습니다.

1.5.1.8. 구성 요소 수정

- *maistra-version* 레이블이 모든 리소스에 추가되었습니다.
- 모든 Ingress 리소스가 OpenShift Route 리소스로 변환되었습니다.
- Grafana, distributed tracing(Jaeger) 및 Kiali는 기본적으로 활성화되어 OpenShift 경로를 통해 노출됩니다.
- Godebug가 모든 템플릿에서 제거됨
- istio-multi ServiceAccount과 ClusterRoleBinding, istio-reader ClusterRole이 제거되었습니다.

1.5.1.9. Envoy 필터

Red Hat OpenShift Service Mesh는 명시적으로 문서화된 경우를 제외하고 EnvoyFilter 구성을 지원하지 않습니다. 기본 Envoy API와의 긴밀한 결합으로 인해 이전 버전과의 호환성을 유지할 수 없습니다. EnvoyFilter 패치는 Istio에서 생성한 Envoy 구성의 형식에 매우 민감합니다. Istio에서 생성한 구성이 변경되면 EnvoyFilter 의 애플리케이션을 중단할 수 있습니다.

1.5.1.10. Envoy 서비스

Red Hat OpenShift Service Mesh는 QUIC 기반 서비스를 지원하지 않습니다.

1.5.1.11. Istio CNI(Container Network Interface) 플러그인

Red Hat OpenShift Service Mesh에는 CNI 플러그인이 포함되어 있으며, 이를 통해 애플리케이션 Pod 네트워킹을 구성할 수 있는 대체 방법을 제공합니다. CNI 플러그인은 승격된 권한으로 SCC(보안 컨텍스트 제약 조건)에 서비스 계정 및 프로젝트 액세스 권한을 부여할 필요가 없도록 init-container 네트워크 구성을 대체합니다.

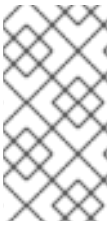
1.5.1.12. 글로벌 mTLS 설정

Red Hat OpenShift Service Mesh는 메시 내에서 상호 TLS 인증(mTLS)을 활성화 또는 비활성화하는 PeerAuthentication 리소스를 생성합니다.

1.5.1.13. 게이트웨이

Red Hat OpenShift Service Mesh는 기본적으로 수신 및 송신 게이트웨이를 설치합니다. 다음 설정을 사용하여 ServiceMeshControlPlane (SMCP) 리소스에서 게이트웨이 설치를 비활성화할 수 있습니다.

- spec.gateways.enabled=false 를 사용하여 수신 및 송신 게이트웨이를 모두 비활성화합니다.
- spec.gateways.ingress.enabled=false 를 사용하여 수신 게이트웨이를 비활성화합니다.
- spec.gateways.egress.enabled=false 를 사용하여 송신 게이트웨이를 비활성화합니다.



참고

Operator는 기본 게이트웨이에 주석을 달아 Red Hat OpenShift Service Mesh Operator에 의해 생성되고 관리됨을 나타냅니다.

1.5.1.14. 다중 클러스터 구성

멀티 클러스터 구성에 대한 Red Hat OpenShift Service Mesh 지원은 여러 클러스터에서 서비스 메시 통합으로 제한됩니다.

1.5.1.15. 사용자 정의 인증서 서명 요청(CSR)

Kubernetes CA(인증 기관)를 통해 CSR을 처리하도록 Red Hat OpenShift Service Mesh를 구성할 수 없습니다.

1.5.1.16. Istio 게이트웨이 경로

Istio 게이트웨이의 OpenShift 경로는 Red Hat OpenShift Service Mesh에서 자동으로 관리됩니다. Istio 게이트웨이가 서비스 메시 내부에서 생성, 업데이트 또는 삭제될 때마다 OpenShift 경로가 생성, 업데이트 또는 삭제됩니다.

IOR(Istio OpenShift Routing)이라는 Red Hat OpenShift Service Mesh Control Plane 구성 요소는 게이트웨이 경로를 동기화합니다. 자세한 내용은 자동 경로 생성을 참조하십시오.

1.5.1.16.1. catch-all 도메인

catch-all 도메인("*")은 지원되지 않습니다. 게이트웨이 정의에서 이 도메인이 발견되면 Red Hat OpenShift Service Mesh는 경로를 *생성하지만* 기본 호스트 이름을 만들기 위해 OpenShift에 의존합니다. 즉, 새로 생성된 경로는 catch-all ("*") 경로가 *아니며*, 대신 r<route-name>[-<project>].<suffix> 형식의 호스트 이름이 있습니다. 기본 호스트 이름이 작동하는 방식과 cluster-admin이 이를 사용자 정의할 수 있는 방법에 대한 자세한 내용은 OpenShift Container Platform 설명서를 참조하십시오. Red Hat OpenShift Dedicated를 사용하는 경우 Red Hat OpenShift Dedicated에서 dedicated-admin 역할을 참조하십시오.

1.5.1.16.2. 하위 도메인

하위 도메인(예: "*.domain.com")이 지원됩니다. 그러나 이 기능은 OpenShift Container Platform에서 기본적으로 활성화되어 있지 않습니다. 즉, Red Hat OpenShift Service Mesh는 하위 도메인이 있는 경로를 *생성하지만* OpenShift Container Platform이 이것을 활성화하도록 구성된 경우에만 적용됩니다.

1.5.1.16.3. TLS(Transport layer security)

TLS(Transport Layer Security)가 지원됩니다. 즉, 게이트웨이에 tls 섹션이 포함된 경우 OpenShift 경로는 TLS를 지원하도록 구성됩니다.

추가 리소스

- 자동 경로 생성

1.5.2. 다중 테넌트 설치

업스트림 Istio는 하나의 테넌트 접근법을 사용하지만 Red Hat OpenShift Service Mesh는 클러스터 내에서 여러 개의 독립적인 컨트롤 플레인을 지원합니다. Red Hat OpenShift Service Mesh는 다중 테넌트 연산자를 사용하여 컨트롤 플레인 라이프사이클을 관리합니다.

Red Hat OpenShift Service Mesh는 기본적으로 다중 테넌트 컨트롤 플레인을 설치합니다. 서비스 메시에 액세스할 수 있는 프로젝트를 지정하고 다른 컨트롤 플레인 인스턴스에서 서비스 메시를 분리합니다.

1.5.2.1. 멀티 테넌시 대 클러스터 전체 설치

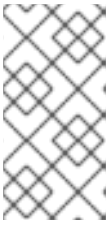
다중 테넌트 설치와 클러스터 전체 설치의 주요 차이점은 istod에서 사용하는 권한 범위입니다. 구성 요소는 더 이상 클러스터 범위의 역할 기반 액세스 제어(RBAC) 리소스 ClusterRoleBinding을 사용하지 않습니다.

ServiceMeshMemberRoll members 목록에 있는 모든 프로젝트는 컨트롤 플레인 배포와 관련된 각 서비스 계정에 대해 RoleBinding을 가지며, 각 컨트롤 플레인 배포는 해당하는 멤버 프로젝트만 감시합니다. 각 멤버 프로젝트에는 maistra.io/member-of 레이블이 추가됩니다. 여기서 member-of 값은 컨트롤 플레인 설치가 포함된 프로젝트입니다.

Red Hat OpenShift Service Mesh는 각 멤버 프로젝트를 구성하여 자체, 컨트롤 플레인 및 기타 멤버 프로젝트 간의 네트워크 액세스를 보장합니다. 정확한 구성은 OpenShift Container Platform 소프트웨어 정의 네트워킹(SDN)이 구성된 방법에 따라 다릅니다. 자세한 내용은 OpenShift SDN 정보를 참조하십시오.

OpenShift Container Platform 클러스터가 SDN 플러그인을 사용하도록 구성된 경우:

- NetworkPolicy: Red Hat OpenShift Service Mesh는 각 멤버 프로젝트에서 NetworkPolicy 리소스를 생성하여 다른 멤버 및 컨트롤 플레인에서 모든 pod로 수신할 수 있습니다. Service Mesh에서 멤버를 제거하면 이 NetworkPolicy 리소스는 프로젝트에서 삭제됩니다.



참고

또한 멤버 프로젝트 전용 수신으로 제한합니다. 멤버 외 프로젝트에서 수신이 필요한 경우 해당 트래픽을 허용하기 위해 NetworkPolicy를 생성해야 합니다.

-

다중 테넌트: **Red Hat OpenShift Service Mesh**는 각 멤버 프로젝트의 **NetNamespace**를 컨트롤 플레인 프로젝트의 **NetNamespace**에 결합합니다(**oc adm pod-network join-projects --to control-plane-project member-project**). 서비스 메시에서 멤버를 제거하면 해당 **NetNamespace**가 컨트롤 플레인과 분리됩니다(**oc adm pod-network isolate-projects member-project** 실행과 동일).

- 서브넷: 추가 구성이 수행되지 않습니다.

1.5.2.2. 클러스터 범위 리소스

업스트림 **Istio**에는 의존하는 두 개의 클러스터 범위 리소스가 있습니다. **MeshPolicy** 및 **ClusterRbacConfig** 이는 다중 테넌트 클러스터와 호환되지 않으며 아래에 설명된 대로 교체되었습니다.

- ***ServiceMeshPolicy***는 컨트롤 플레인 전체의 인증 정책 구성을 위해 **MeshPolicy**를 대체합니다. 이는 컨트롤 플레인과 동일한 프로젝트에서 생성되어야 합니다.

- ***ServicemeshRbacConfig***는 컨트롤 플레인 전체 역할 기반 액세스 제어 구성을 위해 **ClusterRbacConfig** 를 대체합니다. 이는 컨트롤 플레인과 동일한 프로젝트에서 생성되어야 합니다.

1.5.3. Kiali 및 서비스 메시

OpenShift Container Platform의 서비스 메시를 통해 **Kiali**를 설치하는 것은 여러 가지 면에서 커뮤니티 **Kiali** 설치와 다릅니다. 이러한 수정은 **OpenShift Container Platform**에 배포할 때 문제를 해결하거나, 추가 기능을 제공하거나, 차이점을 처리하기 위해 필요한 경우가 있습니다.

- **Kiali**는 기본적으로 활성화되어 있습니다.

- **Ingress**는 기본적으로 활성화되어 있습니다.

- **Kiali ConfigMap**이 업데이트되었습니다.

- **Kiali**의 **ClusterRole** 설정이 업데이트되었습니다.

- 서비스 메시 또는 **Kiali Operator**가 변경 사항을 덮어쓸 수 있으므로 **ConfigMap**을 편집하지 마십시오. **Kiali Operator**가 관리하는 파일에는 **kiali.io/** 레이블 또는 주석이 있습니다. **Operator**

파일을 업데이트하려면 **cluster-admin** 권한이 있는 사용자로 제한해야 합니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **Operator** 파일을 업데이트하려면 **dedicated-admin** 권한이 있는 사용자로 제한해야 합니다.

1.5.4. 분산 추적 및 서비스 메시

OpenShift Container Platform의 **Service Mesh**를 사용하여 분산 추적 플랫폼(**Jaeger**)을 설치하는 것은 여러 가지 면에서 커뮤니티 **Jaeger** 설치와 다릅니다. 이러한 수정은 **OpenShift Container Platform**에 배포할 때 문제를 해결하거나, 추가 기능을 제공하거나, 차이점을 처리하기 위해 필요한 경우가 있습니다.

- 분산 추적은 기본적으로 서비스 메시에 대해 활성화되어 있습니다.
- **Ingress**는 기본적으로 서비스 메시에 대해 활성화되어 있습니다.
- **Zipkin** 포트 이름의 이름이 **jaeger-collector-zipkin(http)**으로 변경되었습니다.
- **Jaeger**는 **production** 또는 **streaming** 배포 옵션을 선택할 때 기본적으로 스토리지에 **Elasticsearch**를 사용합니다.
- **Istio** 커뮤니티 버전은 일반적인 **"tracing"** 경로를 제공합니다. **Red Hat OpenShift Service Mesh**는 **Red Hat OpenShift distributed tracing Platform (Jaeger) Operator**가 설치하고 이미 **OAuth**에 의해 보호되는 **"jaeger"** 경로를 사용합니다.
- **Red Hat OpenShift Service Mesh**는 **Envoy** 프록시에 사이드카를 사용하며 **Jaeger** 또한 **Jaeger** 에이전트에 사이드카를 사용합니다. 이 두 가지 사이드카는 별도로 구성되어 있으며 서로 혼동해서는 안 됩니다. 프록시 사이드카는 **Pod**의 수신 및 송신 트래픽과 관련된 기간을 생성합니다. 에이전트 사이드카는 응용 프로그램에서 발송되는 기간을 수신하여 **Jaeger** 수집기로 보냅니다.

1.6. SERVICE MESH 설치 준비

Red Hat OpenShift Service Mesh를 설치하려면 먼저 **OpenShift Container Platform**을 구독하고 지원되는 구성에 **OpenShift Container Platform**을 설치해야 합니다.

1.6.1. 사전 요구 사항

-

Red Hat 계정에 유효한 OpenShift Container Platform 서브스크립션을 유지하십시오. 서브스크립션이 없는 경우 영업 담당자에게 자세한 내용을 문의하십시오.

- OpenShift Container Platform 4.11 개요 를 검토합니다.
- OpenShift Container Platform 4.11을 설치합니다. 제한된 네트워크에 Red Hat OpenShift Service Mesh를 설치하는 경우 선택한 OpenShift Container Platform 인프라에 대한 지침을 따르십시오.
  - AWS에 OpenShift Container Platform 4.11 설치
  - 사용자 프로비저닝 AWS에 OpenShift Container Platform 4.11 설치
  - 베어 메탈에 OpenShift Container Platform 4.11 설치
  - vSphere에 OpenShift Container Platform 4.11 설치
  - IBM Z 및 LinuxONE에 OpenShift Container Platform 4.11 설치
  - IBM Power에 OpenShift Container Platform 4.11 설치
- OpenShift Container Platform 버전과 일치하는 OpenShift Container Platform 명령줄 유틸리티(oc 클라이언트 도구) 버전을 설치하고 해당 경로에 추가합니다.
  - OpenShift Container Platform 4.11을 사용하는 경우 OpenShift CLI 정보를 참조하십시오.

Red Hat OpenShift Service Mesh 라이프사이클 및 지원되는 플랫폼에 대한 자세한 내용은 지원 정책을 참조하십시오.

1.6.2. 지원되는 구성

Red Hat OpenShift Service Mesh의 현재 릴리스에서는 다음 구성이 지원됩니다.

1.6.2.1. 지원되는 플랫폼

Red Hat OpenShift Service Mesh Operator는 ServiceMeshControlPlane 리소스의 여러 버전을 지원합니다. 버전 2.4 Service Mesh Control Plane은 다음 플랫폼 버전에서 지원됩니다.

- Red Hat OpenShift Container Platform 버전 4.10 이상
- Red Hat OpenShift Dedicated 버전 4.
- Azure Red Hat OpenShift (ARO) 버전 4.
- Red Hat OpenShift Service on AWS(ROSA).

1.6.2.2. 지원되지 않는 로깅 구성

명시적으로 지원되지 않는 경우는 다음과 같습니다.

- OpenShift Online은 Red Hat OpenShift Service Mesh에서 지원되지 않습니다.
- Red Hat OpenShift Service Mesh는 Service Mesh가 실행 중인 클러스터 외부에서 마이크로 서비스 관리를 지원하지 않습니다.

1.6.2.3. 지원되는 네트워크 구성

Red Hat OpenShift Service Mesh는 다음과 같은 네트워크 구성을 지원합니다.

- OpenShift-SDN
- OVN-Kubernetes는 지원되는 모든 OpenShift Container Platform 버전에서 사용할 수 있습니다.

- OpenShift Container Platform에서 인증되었으며 Service Mesh 적합성 테스트를 통과한 타사 CNI(Container Network Interface) 플러그인입니다. 자세한 내용은 인증된 OpenShift CNI 플러그인을 참조하십시오.

1.6.2.4. Service Mesh에 지원되는 구성

- 이번 Red Hat OpenShift Service Mesh 릴리스는 OpenShift Container Platform x86_64, IBM Z 및 IBM Power에서만 사용할 수 있습니다.
  - IBM Z는 OpenShift Container Platform 4.10 이상에서만 지원됩니다.
  - IBM Power는 OpenShift Container Platform 4.10 이상에서만 지원됩니다.
- 모든 Service Mesh 구성 요소가 단일 OpenShift Container Platform 클러스터에 포함된 구성입니다.
- 가상 머신과 같은 외부 서비스를 통합하지 않는 구성입니다.
- Red Hat OpenShift Service Mesh는 명시적으로 문서화된 경우를 제외하고 EnvoyFilter 구성을 지원하지 않습니다.

1.6.2.5. Kiali에 대해 지원되는 구성

- Kiali 콘솔은 Google Chrome, Microsoft Edge, Mozilla Firefox 또는 Apple Safari 브라우저의 두 가지 최신 릴리스에서만 지원됩니다.
- openshift 인증 전략은 Kiali가 Red Hat OpenShift Service Mesh(OSSM)와 함께 배포될 때 지원되는 유일한 인증 구성입니다. openshift 전략은 OpenShift Container Platform의 개별 RBAC(역할 기반 액세스 제어) 역할을 기반으로 액세스를 제어합니다.

1.6.2.6. 분산 추적에 지원되는 구성

- 사이드카로서의 Jaeger 에이전트는 Jaeger에 대해 지원되는 유일한 구성입니다. 다중 테넌트 설치 또는 OpenShift Dedicated에서는 데몬 세트로 Jaeger가 지원되지 않습니다.

1.6.2.7. 지원되는 WebAssembly 모듈

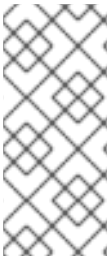
- 3scale WebAssembly는 제공된 유일한 WebAssembly 모듈입니다. 사용자 정의 WebAssembly 모듈을 생성할 수 있습니다.

1.6.3. 다음 단계

- OpenShift Container Platform 환경에 Red Hat OpenShift Service Mesh를 설치합니다.

1.7. OPERATOR 설치

Red Hat OpenShift Service Mesh를 설치하려면 먼저 OpenShift Container Platform에 필요한 Operator를 설치한 다음 ServiceMeshControlPlane 리소스를 생성하여 컨트롤 플레인을 배포합니다.



참고

이 기본 설치는 기본 OpenShift 설정을 기반으로 구성되며 프로덕션용으로 설계되지 않습니다. 이 기본 설치를 사용하여 설치를 확인한 다음 특정 환경에 대한 서비스 메시를 구성합니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh 설치 준비 프로세스를 읽어 보십시오.
- cluster-admin 역할이 있는 계정. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.

다음 단계에서는 OpenShift Container Platform에 Red Hat OpenShift Service Mesh의 기본 인스턴스를 설치하는 방법을 보여줍니다.

1.7.1. Operator 개요

Red Hat OpenShift Service Mesh에는 다음과 같은 네 가지 Operator가 필요합니다.

- OpenShift Elasticsearch - (선택 사항)은 분산 추적 플랫폼(Jaeger)을 사용하여 추적 및 로깅을 위한 데이터베이스 스토리지를 제공합니다. 오픈 소스 Elasticsearch 프로젝트를 기반으로

합니다.

- **Red Hat OpenShift distributed tracing platform(Jaeger) -** 분산 추적을 제공하여 복잡한 분산 시스템의 트랜잭션을 모니터링하고 문제를 해결합니다. 오픈 소스 **Jaeger** 프로젝트를 기반으로 합니다.

- **Red Hat**에서 제공하는 **Kiali Operator -** 서비스 메시에 대한 관찰 기능을 제공합니다. 단일 콘솔에서 구성을 보고, 트래픽을 모니터링하고, 추적을 분석할 수 있습니다. 오픈 소스 **Kiali** 프로젝트를 기반으로 합니다.

- **Red Hat OpenShift Service Mesh -** 애플리케이션을 구성하는 마이크로 서비스를 연결, 보안, 제어 및 관찰할 수 있습니다. **Service Mesh Operator**는 **Service Mesh** 구성 요소의 배포, 업데이트 및 삭제를 관리하는 **ServiceMeshControlPlane** 리소스를 정의하고 모니터링합니다. 오픈소스 **Istio** 프로젝트를 기반으로 합니다.

주의

Operator의 커뮤니티 버전은 설치하지 마십시오. 커뮤니티 **Operator**는 지원되지 않습니다.

1.7.2. Operator 설치

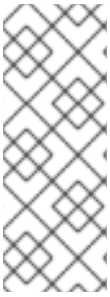
Red Hat OpenShift Service Mesh를 설치하려면 다음 **Operator**를 이 순서대로 설치합니다. 각 **Operator**에 대한 절차를 반복합니다.

- **OpenShift Elasticsearch**

- **Red Hat OpenShift distributed tracing platform (Jaeger)**

- **Red Hat**에서 제공하는 **Kiali Operator**

- **Red Hat OpenShift Service Mesh**



참고

이미 OpenShift Elasticsearch Operator를 OpenShift 로깅의 일부로 설치한 경우 OpenShift Elasticsearch Operator를 다시 설치할 필요가 없습니다. Red Hat OpenShift distributed tracing Platform (Jaeger) Operator는 설치된 OpenShift Elasticsearch Operator를 사용하여 Elasticsearch 인스턴스를 생성합니다.

프로세스

1. cluster-admin 역할의 사용자로 OpenShift Container Platform 웹 콘솔에 로그인합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.

2. OpenShift Container Platform 웹 콘솔에서 Operator → OperatorHub를 클릭합니다.

3. Operator 이름을 필터 상자에 입력하고 Operator의 Red Hat 버전을 선택합니다. Operator의 커뮤니티 버전은 지원되지 않습니다.

4. 설치를 클릭합니다.

5. 각 Operator의 Operator 설치 페이지에서 기본 설정을 수락합니다.

6. 설치를 클릭합니다. 목록에서 다음 Operator에 대한 단계를 반복하기 전에 Operator가 설치될 때까지 기다립니다.

   - OpenShift Elasticsearch Operator는 openshift-operators-redhat 네임스페이스에 설치되며 클러스터의 모든 네임스페이스에서 사용할 수 있습니다.
   - Red Hat OpenShift distributed tracing platform(Jaeger)은 openshift-distributed-tracing 네임스페이스에 설치되고 클러스터의 모든 네임스페이스에서 사용할 수 있습니다.
   - Red Hat 및 Red Hat OpenShift Service Mesh Operator에서 제공하는 Kiali Operator는 openshift-operators 네임스페이스에 설치되고 클러스터의 모든 네임스페이스에서 사용할 수 있습니다.

7. 4개의 Operator를 모두 설치한 후 Operators → 설치된 Operators를 클릭하여 Operator가

설치되었는지 확인합니다.

1.7.3. 인프라 노드에서 실행되도록 Service Mesh Operator 구성

이 작업은 Service Mesh Operator가 인프라 노드에서 실행되는 경우에만 수행해야 합니다.

작업자 노드에서 Operator가 실행되면 이 작업을 건너뜁니다.

사전 요구 사항

- Service Mesh Operator가 설치되어 있어야 합니다.
- 배포를 구성하는 노드 중 하나는 인프라 노드여야 합니다. 자세한 내용은 "인프라 머신 세트 생성"을 참조하십시오.

절차

1. 네임스페이스에 설치된 Operator를 나열합니다.

```
$ oc -n openshift-operators get subscriptions
```

2. Service Mesh Operator 서브스크립션 리소스를 편집하여 Operator를 실행해야 하는 위치를 지정합니다.

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

1 <name >은 Subscription 리소스의 이름을 나타냅니다. Subscription 리소스의 기본 이름은 servicemeshoperator 입니다.

3. Subscription 리소스에서 nodeSelector 및 tolerations 를 spec.config 에 추가합니다.

```
spec:
  config:
    nodeSelector: 1
      node-role.kubernetes.io/infra: ""
```

```
tolerations: 2
- effect: NoSchedule
  key: node-role.kubernetes.io/infra
  value: reserved
- effect: NoExecute
  key: node-role.kubernetes.io/infra
  value: reserved
```

1 Operator Pod가 인프라 노드에만 예약되도록 합니다.

2 인프라 노드에서 Pod를 허용하도록 합니다.

1.7.4. Service Mesh Operator가 인프라 노드에서 실행 중인지 확인

절차

- Operator Pod와 연결된 노드가 인프라 노드인지 확인합니다.

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

1.7.5. 다음 단계

- Red Hat OpenShift Service Mesh Operator는 Service Mesh Control Plane을 배포할 때까지 Service Mesh CRD(사용자 정의 리소스 정의)를 생성하지 않습니다. ServiceMeshControlPlane 리소스를 사용하여 Service Mesh 구성 요소를 설치하고 구성할 수 있습니다. 자세한 내용은 ServiceMeshControlPlane 생성을 참조하십시오.

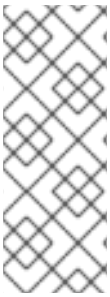
1.8. SERVICEMESHCONTROLPLANE 생성

1.8.1. About ServiceMeshControlPlane

컨트롤 플레인에는 Istiod, Ingress 및 Egress 게이트웨이 및 Kiali 및 Jaeger와 같은 기타 구성 요소가 포함됩니다. 컨트롤 플레인은 Service Mesh Operator 및 데이터 플레인 애플리케이션 및 서비스와 다른 별도의 네임스페이스에 배포해야 합니다. OpenShift Container Platform 웹 콘솔 또는 oc 클라이언트 툴을 사용하여 명령줄에서 SMCP(ServiceMeshControlPlane)의 기본 설치를 배포할 수 있습니다.

참고

이 기본 설치는 기본 **OpenShift Container Platform** 설정을 기반으로 구성되며 프로덕션용으로 설계되지 않습니다. 이 기본 설치를 사용하여 설치를 확인한 다음 환경에 대한 **ServiceMeshControlPlane** 설정을 구성합니다.



참고

ROSA(Red Hat OpenShift Service on AWS)는 리소스를 생성할 수 있는 위치에 대한 추가 제한을 배치하므로 기본 배포가 작동하지 않습니다. **ROSA** 환경에 **SMCP**를 배포하기 전에 추가 요구 사항은 **AWS**의 **Red Hat OpenShift Service**에 서비스 메시 설치를 참조하십시오.

참고

Service Mesh 문서는 **istio-system**을 예제 프로젝트로 사용하지만, 모든 프로젝트에 서비스 메시를 배포할 수 있습니다.

1.8.1.1. 웹 콘솔에서 Service Mesh Control Plane 배포

웹 콘솔을 사용하여 기본 **ServiceMeshControlPlane**을 배포할 수 있습니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 프로젝트의 이름입니다.

사전 요구 사항

- **Red Hat OpenShift Service Mesh Operator**가 설치되어 있어야 합니다.
- **cluster-admin** 역할이 있는 계정.

절차

1. **cluster-admin** 역할의 사용자로 **OpenShift Container Platform** 웹 콘솔에 로그인합니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 계정이 있어야 합니다.
2. **istio-system**이라는 프로젝트를 생성합니다.
   a. 홈 → 프로젝트로 이동합니다.

b. 프로젝트 만들기를 클릭합니다.

c. 이름 필드에 **istio-system**을 입력합니다. **ServiceMeshControlPlane** 리소스는 마이크로 서비스 및 **Operator**와 별도로 프로젝트에 설치해야 합니다.

이 단계에서는 **istio-system** 을 예로 사용하지만 서비스가 포함된 프로젝트와 별도로 모든 프로젝트에 **Service Mesh Control Plane**을 배포할 수 있습니다.

d. 생성을 클릭합니다.

3. **Operators** → 설치된 **Operator**로 이동합니다.

4. **Red Hat OpenShift Service Mesh Operator**를 클릭한 다음 **Istio Service Mesh Control Plane**을 클릭합니다.

5. **Istio Service Mesh Control Plane** 탭에서 **ServiceMeshControlPlane** 생성을 클릭합니다.

6. **ServiceMeshControlPlane** 생성 페이지에서 기본 **Service Mesh Control Plane** 버전을 수락하여 제품의 최신 버전에서 사용할 수 있는 기능을 활용합니다. 컨트롤 플레인의 버전에 따라 **Operator** 버전에 관계없이 사용 가능한 기능을 결정합니다.

a. 생성을 클릭합니다. **Operator**는 구성 매개변수를 기반으로 **pods**, 서비스 및 **Service Mesh Control Plane** 구성 요소를 생성합니다. 나중에 **ServiceMeshControlPlane** 설정을 구성할 수 있습니다.

7. 컨트롤 플레인이 올바르게 설치되었는지 확인하려면 **Istio Service Mesh Control Plane** 탭을 클릭합니다.

a. 새 컨트롤 플레인의 이름을 클릭합니다.

b. 리소스 탭을 클릭하여 **Operator**가 생성 및 구성된 **Red Hat OpenShift Service Mesh Control Plane** 리소스를 확인합니다.

1.8.1.2. CLI를 사용하여 Service Mesh Control Plane 배포

명령줄에서 기본 ServiceMeshControlPlane을 배포할 수 있습니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh Operator가 설치되어 있어야 합니다.
- OpenShift CLI(oc)에 액세스합니다.

절차

1. cluster-admin 역할의 사용자로 OpenShift Container Platform CLI에 로그인합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.

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

2. istio-system이라는 프로젝트를 생성합니다.

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

3. 다음 예제를 사용하여 istio-installation.yaml이라는 ServiceMeshControlPlane 파일을 생성합니다. Service Mesh Control Plane의 버전에 따라 Operator 버전에 관계없이 사용 가능한 기능을 결정합니다.

버전 2.4 istio-installation.yaml 예

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.4
  tracing:
    type: Jaeger
    sampling: 10000
  addons:
    jaeger:
      name: jaeger
```

```
    install:
      storage:
        type: Memory
  kiali:
    enabled: true
    name: kiali
  grafana:
    enabled: true
```

4. 다음 명령을 실행하여 서비스 메시 컨트롤 플레인을 배포합니다. 여기서 < istio_installation.yaml >에는 파일에 대한 전체 경로가 포함됩니다.

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

5. Pod 배포 진행 상황을 조사하려면 다음 명령을 실행합니다.

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

출력은 다음과 유사합니다.

```
NAME                                READY   STATUS    RESTARTS   AGE
grafana-b4d59bd7-mrgbr              2/2     Running   0          65m
istio-egressgateway-678dc97b4c-wrjkp   1/1     Running   0          108s
istio-ingressgateway-b45c9d54d-4qg6n   1/1     Running   0          108s
istiod-basic-55d78bbbcd-j5556       1/1     Running   0          108s
jaeger-67c75bd6dc-jv6k6             2/2     Running   0          65m
kiali-6476c7656c-x5msp              1/1     Running   0          43m
prometheus-58954b8d6b-m5std         2/2     Running   0          66m
```

1.8.1.3. CLI를 사용하여 SMCP 설치 검증

명령줄에서 ServiceMeshControlPlane 생성을 검증할 수 있습니다.

절차

1. cluster-admin 역할의 사용자로 OpenShift Container Platform CLI에 로그인합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.

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

2. 다음 명령을 실행하여 서비스 메시 컨트롤 플레인 설치를 확인합니다. 여기서 **istio-system**은 **Service Mesh Control Plane**을 설치한 네임스페이스입니다.

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

STATUS 열이 **ComponentsReady**인 경우 설치가 성공적으로 완료되었습니다.

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

1.8.2. 컨트롤 플레인 구성 요소 및 인프라 노드 정보

인프라 노드는 다음 두 가지 기본 목적으로 인프라 워크로드를 분리하는 방법을 제공합니다.

- 서브스크립션 수에 대한 청구 비용 발생 방지
- 인프라 워크로드의 유지 관리 및 관리를 분리하려면 다음을 수행합니다.

인프라 노드에서 실행되도록 일부 또는 모든 **Service Mesh Control Plane** 구성 요소를 구성할 수 있습니다.

1.8.2.1. 웹 콘솔을 사용하여 인프라 노드에서 실행되도록 모든 컨트롤 플레인 구성 요소 구성

Service Mesh Control Plane에서 배포한 모든 구성 요소가 인프라 노드에서 실행되는 경우 이 작업을 수행합니다. 이러한 배포된 구성 요소에는 **Istiod, Ingress Gateway, Egress Gateway** 및 **Prometheus, Grafana** 및 **Distributed Tracing**과 같은 선택적 애플리케이션이 포함됩니다.

컨트롤 플레인이 작업자 노드에서 실행되면 이 작업을 건너뜁니다.

사전 요구 사항

- **Red Hat OpenShift Service Mesh Operator**가 설치되어 있습니다.
- **cluster-admin** 역할의 사용자로 로그인되어 있습니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 사용자로 로그인되어 있습니다.

절차

1. OpenShift Container Platform 웹 콘솔에 로그인합니다.

2. Operators → 설치된 Operator로 이동합니다.

3. Red Hat OpenShift Service Mesh Operator를 클릭한 다음 Istio Service Mesh Control Plane 을 클릭합니다.

4. 컨트롤 플레인 리소스의 이름을 클릭합니다. 예를 들어 기본.

5. YAML 을 클릭합니다.

6. 다음 예와 같이 ServiceMeshControlPlane 리소스의 spec.runtime.defaults.pod 사양에 nodeSelector 및 tolerations 필드를 추가합니다.

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

1 ServiceMeshControlPlane Pod가 인프라 노드에만 예약되어 있는지 확인합니다.

2 실행을 위해 인프라 노드에서 Pod를 수락하는지 확인합니다.

7. 저장을 클릭합니다.

8. 새로 고침을 클릭합니다.

1.8.2.2. 웹 콘솔을 사용하여 인프라 노드에서 실행되도록 개별 컨트롤 플레인 구성 요소 구성

Service Mesh Control Plane에서 배포한 개별 구성 요소가 인프라 노드에서 실행되는 경우 이 작업을 수행합니다. 이러한 배포된 구성 요소에는 Istiod, Ingress 게이트웨이, Egress 게이트웨이가 포함됩니다.

컨트롤 플레인이 작업자 노드에서 실행되면 이 작업을 건너뜁니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh Operator가 설치되어 있습니다.
- cluster-admin 역할의 사용자로 로그인되어 있습니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 사용자로 로그인되어 있습니다.

절차

1. OpenShift Container Platform 웹 콘솔에 로그인합니다.
2. Operators → 설치된 Operator로 이동합니다.
3. Red Hat OpenShift Service Mesh Operator를 클릭한 다음 Istio Service Mesh Control Plane 을 클릭합니다.
4. 컨트롤 플레인 리소스의 이름을 클릭합니다. 예를 들어 기본.
5. YAML 을 클릭합니다.
6. 다음 예와 같이 ServiceMeshControlPlane 리소스의 spec.runtime.components.pilot.pod 사양에 nodeSelector 및 tolerations 필드를 추가합니다.

```
spec:
```

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

1 Istiod Pod가 인프라 노드에만 예약되도록 합니다.

2 실행을 위해 인프라 노드에서 Pod를 수락하는지 확인합니다.

7. 다음 예와 같이 ServiceMeshControlPlane 리소스의 spec.gateways.ingress.runtime.pod 및 spec.gateways.egress.runtime.pod 사양에 nodeSelector 및 tolerations 필드를 추가합니다.

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

```
- effect: NoExecute
  key: node-role.kubernetes.io/infra
  value: reserved
```

1 3

게이트웨이 Pod가 인프라 노드에만 예약되도록 합니다.

2 4

실행을 위해 인프라 노드에서 Pod를 수락하는지 확인합니다.

8. 저장을 클릭합니다.

9. 새로 고침을 클릭합니다.

1.8.2.3. CLI를 사용하여 인프라 노드에서 실행되도록 모든 컨트롤 플레인 구성 요소 구성

Service Mesh Control Plane에서 배포한 모든 구성 요소가 인프라 노드에서 실행되는 경우 이 작업을 수행합니다. 이러한 배포된 구성 요소에는 Istiod, Ingress Gateway, Egress Gateway 및 Prometheus, Grafana 및 Distributed Tracing과 같은 선택적 애플리케이션이 포함됩니다.

컨트롤 플레인이 작업자 노드에서 실행되면 이 작업을 건너뜁니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh Operator가 설치되어 있습니다.
- cluster-admin 역할의 사용자로 로그인되어 있습니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 사용자로 로그인되어 있습니다.

절차

1. ServiceMeshControlPlane 리소스를 YAML 파일로 엽니다.

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

1

<name >은 ServiceMeshControlPlane 리소스의 이름을 나타냅니다.

2. 인프라 노드에서 ServiceMeshControlPlane 에서 배포한 모든 Service Mesh 구성 요소를 실행하려면 ServiceMeshControlPlane 리소스의 spec.runtime.defaults.pod 사양에 nodeSelector 및 tolerations 필드를 추가합니다.

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

1 SMCP Pod가 인프라 노드에만 예약되도록 합니다.

2 인프라 노드에서 Pod를 허용하도록 합니다.

1.8.2.4. CLI를 사용하여 인프라 노드에서 실행되도록 개별 컨트롤 플레인 구성 요소 구성

Service Mesh Control Plane에서 배포한 개별 구성 요소가 인프라 노드에서 실행되는 경우 이 작업을 수행합니다. 이러한 배포된 구성 요소에는 Istiod, Ingress 게이트웨이, Egress 게이트웨이가 포함됩니다.

컨트롤 플레인이 작업자 노드에서 실행되면 이 작업을 건너뜁니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh Operator가 설치되어 있습니다.
- cluster-admin 역할의 사용자로 로그인되어 있습니다. Red Hat OpenShift Dedicated를 사

용하는 경우 **dedicated-admin** 역할의 사용자로 로그인되어 있습니다.

절차

1. **ServiceMeshControlPlane** 리소스를 **YAML** 파일로 엽니다.

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

1 **<name >**은 **ServiceMeshControlPlane** 리소스의 이름을 나타냅니다.

2. 인프라 노드에서 **Istiod** 구성 요소를 실행하려면 **ServiceMeshControlPlane** 리소스의 **spec.runtime.components.pilot.pod** 사양에 **nodeSelector** 및 **tolerations** 필드를 추가합니다.

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

1 **Istiod Pod**가 인프라 노드에만 예약되도록 합니다.

2 인프라 노드에서 **Pod**를 허용하도록 합니다.

3. 인프라 노드에서 **Ingress** 및 **Egress Gateways**를 실행하려면 **ServiceMeshControlPlane** 리소스의 **spec.gateways.ingress.runtime.pod** 사양과 **spec.gateways.egress.runtime.pod** 사양에 **nodeSelector** 및 **tolerations** 필드를 추가합니다.

```
spec:
```

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

1 3 게이트웨이 Pod가 인프라 노드에만 예약되도록 합니다.

2 4 인프라 노드에서 Pod를 허용하도록 합니다.

1.8.2.5. 인프라 노드에서 Service Mesh Control Plane이 실행 중인지 확인

절차

- Istiod, Ingress Gateway 및 Egress Gateway Pod와 연결된 노드가 인프라 노드인지 확인합니다.

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

1.8.3. 컨트롤 플레인 및 클러스터 전체 배포 정보

클러스터 전체 배포에는 전체 클러스터의 리소스를 모니터링하는 Service Mesh Control Plane이 포함되어 있습니다. 컨트롤 플레인이 모든 네임스페이스에서 단일 쿼리를 사용하여 Istio 및 Kubernetes 리

소스를 모니터링한다는 점에서 전체 클러스터의 모니터링 리소스는 Istio 기능과 유사합니다. 결과적으로 클러스터 전체 배포는 API 서버로 전송되는 요청 수를 줄입니다.

OpenShift Container Platform 웹 콘솔 또는 CLI를 사용하여 클러스터 전체 배포에 대해 Service Mesh Control Plane을 구성할 수 있습니다.

1.8.3.1. 웹 콘솔을 사용하여 클러스터 전체 배포를 위한 컨트롤 플레인 구성

OpenShift Container Platform 웹 콘솔을 사용하여 클러스터 전체 배포에 대한 ServiceMeshControlPlane 리소스를 구성할 수 있습니다. 이 예제에서 istio-system 은 Service Mesh Control Plane 프로젝트의 이름입니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh Operator가 설치되어 있습니다.
- cluster-admin 역할의 계정을 사용하거나 dedicated-admin 역할과 함께 Red Hat OpenShift Dedicated를 사용하는 경우 로그인되어 있습니다.

절차

1. istio-system이라는 프로젝트를 생성합니다.
   a. 홈 → 프로젝트로 이동합니다.
   b. 프로젝트 만들기를 클릭합니다.
   c. 이름 필드에 istio-system을 입력합니다. ServiceMeshControlPlane 리소스는 마이크로 서비스 및 Operator와 별도로 프로젝트에 설치해야 합니다.

      이러한 단계에서는 istio-system 을 예제로 사용합니다. 서비스가 포함된 프로젝트와 별도로 Service Mesh Control Plane을 모든 프로젝트에 배포할 수 있습니다.
   d. 생성을 클릭합니다.
2.

2. Operators → 설치된 Operator로 이동합니다.

3. Red Hat OpenShift Service Mesh Operator를 클릭한 다음 Istio Service Mesh Control Plane을 클릭합니다.

4. Istio Service Mesh Control Plane 탭에서 ServiceMeshControlPlane 생성을 클릭합니다.

5. YAML 보기를 클릭합니다. Service Mesh Control Plane의 버전에 따라 Operator 버전에 관계없이 사용 가능한 기능을 결정합니다.

6. YAML 파일의 spec.mode 필드를 수정하여 ClusterWide 를 지정합니다.

버전 2.4 istio-installation.yaml 예

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.4
  mode: ClusterWide
```

7. 생성을 클릭합니다. Operator는 구성 매개변수를 기반으로 pods, 서비스 및 Service Mesh Control Plane 구성 요소를 생성합니다. 또한 기본 구성의 일부로 없는 경우 Operator는 ServiceMeshMemberRoll 을 생성합니다.

8. 컨트롤 플레인이 올바르게 설치되었는지 확인하려면 Istio Service Mesh Control Plane 탭을 클릭합니다.

   a. 새 ServiceMeshControlPlane 오브젝트의 이름을 클릭합니다.

   b. 리소스 탭을 클릭하여 Operator가 생성 및 구성한 Red Hat OpenShift Service Mesh

Control Plane 리소스를 확인합니다.

이 모듈은 다음 assembly에 포함되어 있습니다. * service_mesh/v2x/ossm-create-smcp.adoc
:_mod-docs-content-type: PROCEDURE

1.8.3.2. CLI를 사용하여 클러스터 전체 배포를 위한 컨트롤 플레인 구성

CLI를 사용하여 클러스터 전체 배포에 대한 ServiceMeshControlPlane 리소스를 구성할 수 있습니다. 이 예에서 istio-system 은 Service Mesh Control Plane 네임스페이스의 이름입니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh Operator가 설치되어 있습니다.
- OpenShift CLI(oc)에 액세스할 수 있습니다.

절차

1. cluster-admin 역할의 사용자로 OpenShift Container Platform CLI에 로그인합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.

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

2. istio-system이라는 프로젝트를 생성합니다.

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

3. 다음 예제를 사용하여 istio-installation.yaml이라는 ServiceMeshControlPlane 파일을 생성합니다.

버전 2.4 istio-installation.yaml 예

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

```
spec:
  version: v2.4
  mode: ClusterWide
```

4. 다음 명령을 실행하여 서비스 메시 컨트롤 플레인을 배포합니다. 여기서 < istio_installation.yaml >에는 파일에 대한 전체 경로가 포함됩니다.

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

5. Pod 배포의 진행 상황을 모니터링하려면 다음 명령을 실행합니다.

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

다음 예와 유사한 출력이 표시됩니다.

출력 예

```
NAME                            READY   STATUS    RESTARTS   AGE
grafana-b4d59bd7-mrgbr             2/2     Running   0          65m
istio-egressgateway-678dc97b4c-wrjkp   1/1     Running   0          108s
istio-ingressgateway-b45c9d54d-4qg6n   1/1     Running   0          108s
istiod-basic-55d78bbbcd-j5556         1/1     Running   0          108s
jaeger-67c75bd6dc-jv6k6              2/2     Running   0          65m
kiali-6476c7656c-x5msp               1/1     Running   0          43m
prometheus-58954b8d6b-m5std          2/2     Running   0          66m
```

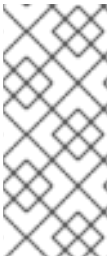
이 모듈은 다음 assembly에 포함되어 있습니다. * service_mesh/v2x/ossm-create-smcp.adoc

1.8.3.3. 클러스터 전체 메시의 멤버 롤 사용자 정의

클러스터 전체 모드에서는 ServiceMeshControlPlane 리소스를 생성할 때 ServiceMeshMemberRoll 리소스도 생성됩니다. ServiceMeshMemberRoll 리소스를 생성한 후 수정할 수 있습니다. 리소스를 수정한 후 Service Mesh Operator에서 더 이상 변경하지 않습니다. OpenShift

Container Platform 웹 콘솔을 사용하여 **ServiceMeshMemberRoll** 리소스를 수정하는 경우 수정을 덮어쓰도록 프롬프트를 수락합니다.

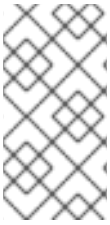
또는 **ServiceMeshControlPlane** 리소스를 배포하기 전에 **ServiceMeshMemberRoll** 리소스를 생성할 수 있습니다. **ServiceMeshControlPlane** 리소스를 생성할 때 **Service Mesh Operator**는 **ServiceMeshMemberRoll** 을 수정하지 않습니다.



참고

ServiceMeshMemberRoll 리소스 이름은 **default** 로 지정해야 하며 **ServiceMeshControlPlane** 리소스와 동일한 프로젝트 네임스페이스에서 생성해야 합니다.

메시에 네임스페이스를 추가하는 방법은 다음 두 가지가 있습니다. **spec.members** 목록에 해당 이름을 지정하여 네임스페이스를 추가하거나 레이블을 기반으로 네임스페이스를 포함하거나 제외하도록 네임스페이스를 구성할 수 있습니다.



참고

ServiceMeshMemberRoll 리소스에 멤버를 지정하는 방법에 관계없이 각 네임스페이스에서 **ServiceMeshMember** 리소스를 생성하여 메시에 멤버를 추가할 수도 있습니다.

1.8.4. Kiali를 사용하여 SMCP 설치 검증

Kiali 콘솔을 사용하여 서비스 메시 설치를 검증할 수 있습니다. **Kiali** 콘솔은 서비스 메시 구성 요소가 올바르게 배포 및 구성되는 여러 가지 방법을 제공합니다.

절차

1. **OpenShift Container Platform** 웹 콘솔에 **cluster-admin** 권한이 있는 사용자로 로그인합니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 계정이 있어야 합니다.

2. 네트워킹 → 경로로 이동합니다.

3. 경로 페이지의 네임스페이스 메뉴에서 **Service Mesh Control Plane** 프로젝트(예: **istio-system**)를 선택합니다.

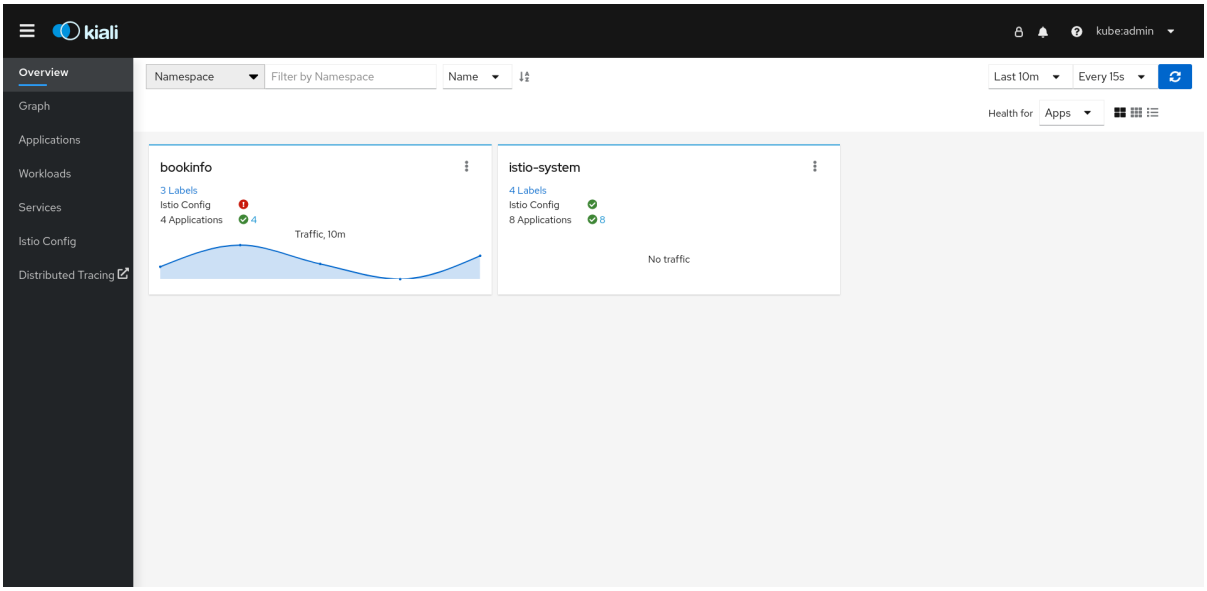
위치 열은 각 경로에 대한 연결된 주소를 표시합니다.

4. 필요한 경우 필터를 사용하여 **Kiali** 콘솔의 경로를 찾습니다. 경로 위치를 클릭하여 콘솔을 시작합니다.

5. **OpenShift**로 로그인 을 클릭합니다.

Kiali 콘솔에 처음 로그인하면 볼 수 있는 권한이 있는 서비스 메시에 모든 네임스페이스가 표시되는 개요 페이지가 표시됩니다. 개요 페이지에 여러 네임스페이스가 표시되면 **Kiali**에서 상태 또는 검증 문제가 있는 네임스페이스를 먼저 표시합니다.

그림 **1.1. Kiali** 개요 페이지



각 네임스페이스의 타일은 레이블 수, **Istio Config** 상태, 애플리케이션 상태 수, 네임스페이스의 트래픽 수 를 표시합니다. 콘솔 설치의 유효성을 검사하고 네임스페이스가 메시에 아직 추가되지 않은 경우 **istio-system** 이외의 데이터가 표시되지 않을 수 있습니다.

6. **Kiali**에는 특히 **Service Mesh Control Plane**이 설치된 네임스페이스에 대한 **4**개의 대시보드가 있습니다. 이러한 대시보드를 보려면 컨트롤 플레인 네임스페이스의 타일에서 옵션 메뉴

를 클릭하고(예: **istio-system**) 다음 옵션 중 하나를 선택합니다.

- **Istio** 메시 대시보드

- **Istio** 컨트롤 플레인 대시보드
- **Istio** 성능 대시보드
- **Istio wsm Exetension** 대시보드

그림 1.2. Grafana Istio 컨트롤 플레인 대시보드

Kiali는 **Grafana** 홈 페이지에서 사용할 수 있는 두 개의 추가 **Grafana** 대시보드도 설치합니다.

- **Istio** 워크로드 대시보드
- **Istio** 서비스 대시보드

7. **Service Mesh Control Plane** 노드를 보려면 그래프 페이지를 클릭하고 메뉴에서 **ServiceMeshControlPlane** 을 설치한 네임스페이스 (예: **istio-system**)를 선택합니다.

a. 필요한 경우 **Display idle nodes** 를 클릭합니다.

b. **Graph** 페이지에 대한 자세한 내용을 보려면 그래프 둘러보기 링크를 클릭합니다.

c.

메시 토폴로지를 보려면 네임스페이스 메뉴에서 서비스 메시 멤버 롤에서 하나 이상의 추가 네임스페이스를 선택합니다.

8. **istio-system** 네임스페이스에서 애플리케이션 목록을 보려면 애플리케이션 페이지를 클릭합니다. **Kiali**는 애플리케이션의 상태를 표시합니다.

   a. 정보 아이콘 위에 마우스를 가져가면 **Details** 열에 언급된 추가 정보를 볼 수 있습니다.

9. **istio-system** 네임스페이스에서 워크로드 목록을 보려면 워크로드 페이지를 클릭합니다. **Kiali**는 워크로드의 상태를 표시합니다.

   a. 정보 아이콘 위에 마우스를 가져가면 **Details** 열에 언급된 추가 정보를 볼 수 있습니다.

10. **istio-system** 네임스페이스에서 서비스 목록을 보려면 서비스 페이지를 클릭합니다. **Kiali**는 서비스 및 구성의 상태를 표시합니다.

    a. 정보 아이콘 위에 마우스를 가져가면 **Details** 열에 언급된 추가 정보를 볼 수 있습니다.

11. **istio-system** 네임스페이스에서 **Istio Configuration** 오브젝트 목록을 보려면 **Istio Config** 페이지를 클릭합니다. **Kiali**는 구성 상태를 표시합니다.

    a. 구성 오류가 있는 경우 행을 클릭하고 **Kiali**를 클릭하면 오류가 강조 표시된 상태에서 구성 파일을 엽니다.

1.8.5. Red Hat OpenShift Service on AWS(ROSA)에 설치

버전 **2.2**부터 **Red Hat OpenShift Service Mesh**는 **AWS(ROSA)**의 **Red Hat OpenShift Service**에 설치를 지원합니다. 이 섹션에서는 이 플랫폼에 서비스 메시를 설치할 때 추가 요구 사항을 설명합니다.

1.8.5.1. 설치 위치

Red Hat OpenShift Service Mesh를 설치하고 **ServiceMeshControlPlane** 을 생성할 때 새 네임스페이스(예: **istio-system**)를 생성해야 합니다.

1.8.5.2. 필수 Service Mesh Control Plane 구성

ServiceMeshControlPlane 파일의 기본 구성은 ROSA 클러스터에서 작동하지 않습니다. AWS의 Red Hat OpenShift Service에 설치할 때 기본 SMCP를 수정하고 spec.security.identity.type=Thirdparty 를 설정해야 합니다.

ROSA의 ServiceMeshControlPlane 리소스 예

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.4
  security:
    identity:
      type: ThirdParty  #required setting for ROSA
  tracing:
    type: Jaeger
    sampling: 10000
  policy:
    type: Istiod
  addons:
    grafana:
      enabled: true
    jaeger:
      install:
        storage:
          type: Memory
    kiali:
      enabled: true
    prometheus:
      enabled: true
  telemetry:
    type: Istiod
```

1.8.5.3. Kiali 구성에 제한 사항

AWS의 Red Hat OpenShift Service에는 리소스를 생성할 수 있는 추가 제한이 있으며 Red Hat 관리 네임스페이스에 Kiali 리소스를 생성할 수 없습니다.

즉, ROSA 클러스터에서 spec.deployment.accessible_namespaces 에 대한 다음과 같은 공통 설정을 허용하지 않습니다.

- ['**'] (모든 네임스페이스)
- default
- codeready-*
- openshift-*
- redhat-*

검증 오류 메시지는 모든 제한된 네임스페이스의 전체 목록을 제공합니다.

ROSA의 Kiali 리소스 예

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

1.8.6. 추가 리소스

Red Hat OpenShift Service Mesh는 클러스터 내에서 여러 개의 독립적인 컨트롤 플레인을 지원합니다. ServiceMeshControlPlane 프로필을 사용하여 재사용 가능한 구성을 생성할 수 있습니다. 자세한 내용은 컨트롤 플레인 프로필 생성을 참조하십시오.

1.8.7. 다음 단계

- 애플리케이션을 사용할 수 있도록 서비스 메시에 프로젝트를 추가합니다. 자세한 내용은 서비스 메시에 서비스 추가를 참조하십시오.

1.9. 서비스 메시에 서비스 추가

프로젝트에는 서비스가 포함되어 있지만 프로젝트를 서비스 메시에 추가하는 경우에만 서비스를 사용할 수 있습니다.

1.9.1. 서비스 메시에 프로젝트 추가 정보

Operator를 설치하고 ServiceMeshControlPlane 리소스를 생성한 후 서비스 메시에 하나 이상의 프로젝트를 추가합니다.

참고

OpenShift Container Platform에서 프로젝트는 기본적으로 프로젝트에서 사용할 수 있는 사용자 ID 범위와 같은 추가 주석이 있는 Kubernetes 네임스페이스입니다. 일반적으로 OpenShift Container Platform 웹 콘솔은 프로젝트라는 용어를 사용하며 CLI는 네임스페이스라는 용어를 사용하지만, 용어는 기본적으로 중요합니다.

OpenShift Container Platform 웹 콘솔 또는 CLI를 사용하여 기존 서비스 메시에 프로젝트를 추가할 수 있습니다. 서비스 메시에 프로젝트를 추가하는 방법은 다음 세 가지가 있습니다.

- ServiceMeshMemberRoll 리소스에서 프로젝트 이름을 지정합니다.
- ServiceMeshMemberRoll 리소스의 spec.labelSelectors 필드에 레이블 선택기 구성.
- 프로젝트에서 ServiceMeshMember 리소스 생성.

첫 번째 방법을 사용하는 경우 ServiceMeshMemberRoll 리소스를 생성해야 합니다.

1.9.2. Red Hat OpenShift Service Mesh 멤버 롤 생성

ServiceMeshMemberRoll 은 Service Mesh Control Plane에 속한 프로젝트를 나열합니다. ServiceMeshMemberRoll에 나열된 프로젝트만 컨트롤 플레인의 영향을 받습니다. 특정 컨트롤 플레인 배포의 멤버 롤에 추가할 때까지 프로젝트는 서비스 메시에 속하지 않습니다.

ServiceMeshControlPlane과 동일한 프로젝트에서 default 라는 ServiceMeshMemberRoll 리소스를 생성해야 합니다. (예: istio-system)

1.9.2.1. 웹 콘솔에서 멤버 롤 생성

웹 콘솔에서 서비스 메시 멤버 롤에 하나 이상의 프로젝트를 추가할 수 있습니다. 이 예제에서 istio-system 은 Service Mesh Control Plane 프로젝트의 이름입니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh Operator 설치 및 검증.
- 서비스 메시에 추가할 기존 프로젝트 목록.

프로세스

1. OpenShift Container Platform 웹 콘솔에 로그인합니다.
2. 메시에 대한 서비스가 아직 없거나 처음부터 시작하려는 경우 애플리케이션에 대한 프로젝트를 생성합니다. Service Mesh Control Plane을 설치한 프로젝트와 달라야 합니다.
   a. 홈 → 프로젝트로 이동합니다.
   b. 이름 필드에 이름을 입력합니다.
   c. 생성을 클릭합니다.
3. Operators → 설치된 Operator로 이동합니다.
4. 프로젝트 메뉴를 클릭하고 목록에서 ServiceMeshControlPlane 리소스가 배포되는 프로젝

트를 선택합니다(예: istio-system).

5. Red Hat OpenShift Service Mesh Operator를 클릭합니다.

6. Istio Service Mesh 멤버 목록 탭을 클릭합니다.

7. ServiceMeshMemberRoll 만들기를 클릭합니다.

8. Members를 클릭한 다음 Value 필드에 프로젝트 이름을 입력합니다. 여러 프로젝트를 추가할 수 있지만 프로젝트는 하나의 ServiceMeshMemberRoll 리소스에만 속할 수 있습니다.

9. 생성을 클릭합니다.

1.9.2.2. CLI에서 멤버 롤 생성

명령줄의 ServiceMeshMemberRoll에 프로젝트를 추가할 수 있습니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh Operator 설치 및 검증.
- 서비스 메시에 추가할 프로젝트 목록.
- OpenShift CLI(oc)에 액세스합니다.

프로세스

1. OpenShift Container Platform CLI에 로그인합니다.

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

2. 메시에 대한 서비스가 아직 없거나 처음부터 시작하려는 경우 애플리케이션에 대한 프로젝트를 생성합니다. Service Mesh Control Plane을 설치한 프로젝트와 달라야 합니다.

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

3. 프로젝트를 멤버로 추가하려면 다음 예제 **YAML**을 수정합니다. 여러 프로젝트를 추가할 수 있지만 프로젝트는 하나의 **ServiceMeshMemberRoll** 리소스에만 속할 수 있습니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 프로젝트의 이름입니다.

servicemeshmemberroll-default.yaml 예

```
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. 다음 명령을 실행하여 **istio-system** 네임스페이스에 **ServiceMeshMemberRoll** 리소스를 업로드하고 만듭니다.

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

5. 다음 명령을 실행하여 **ServiceMeshMemberRoll**이 성공적으로 생성되었는지 확인합니다.

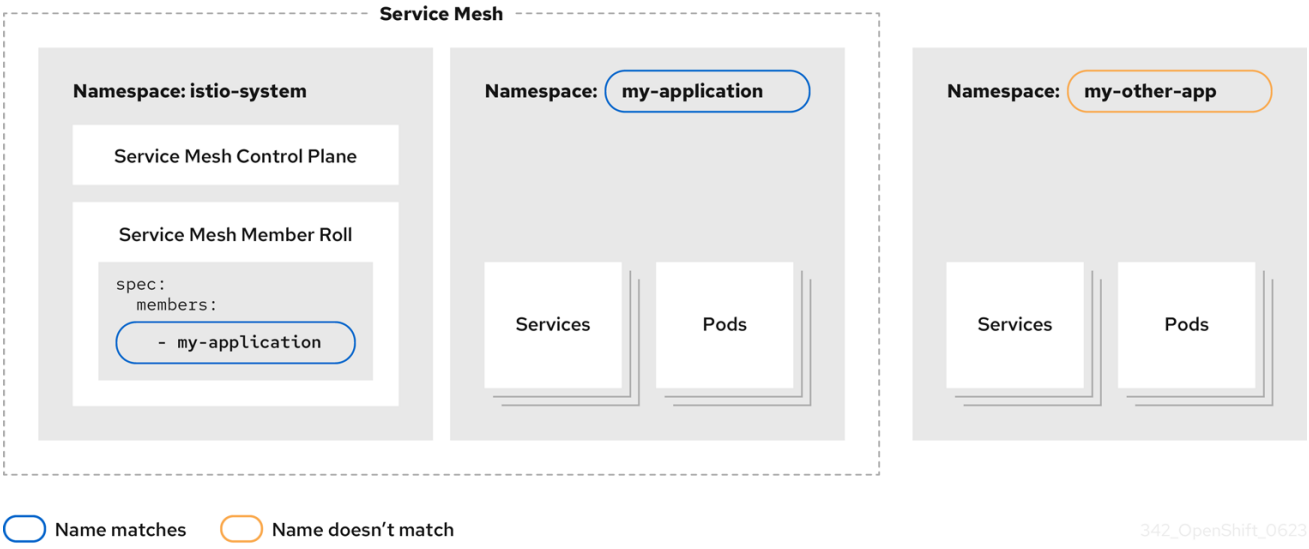
```
$ oc get smmr -n istio-system default
```

STATUS 열이 **Configured**인 경우 설치가 성공적으로 완료된 것입니다.

1.9.3. ServiceMeshMemberRoll 리소스를 사용하여 프로젝트 추가 정보

ServiceMeshMemberRoll 리소스를 사용하는 것이 서비스 메시에 프로젝트를 추가하는 가장 간단한 방법입니다. 프로젝트를 추가하려면 **ServiceMeshMemberRoll** 리소스의 **spec.members** 필드에 프로젝

트 이름을 지정합니다. **ServiceMeshMemberRoll** 리소스는 **ServiceMeshControlPlane** 리소스에서 제어하는 프로젝트를 지정합니다.



참고

이 방법을 사용하여 프로젝트를 추가하려면 사용자가 추가되는 프로젝트에서 **update servicemeshmemberrolls** 및 **update pod** 권한을 보유해야 합니다.

- 서비스 메시에 추가할 애플리케이션, 워크로드 또는 서비스가 이미 있는 경우 다음을 참조하십시오.
  - 웹 콘솔과 **ServiceMeshMemberRoll** 리소스를 사용하여 메시에서 프로젝트 추가 또는 제거
  - **CLI**와 **ServiceMeshMemberRoll** 리소스를 사용하여 메시에서 프로젝트 추가 또는 제거
- 또는 **Bookinfo**라는 샘플 애플리케이션을 설치하고 **ServiceMeshMemberRoll** 리소스에 추가하려면 **Bookinfo** 예제 애플리케이션 튜토리얼을 참조하십시오.

1.9.3.1. 웹 콘솔과 ServiceMeshMemberRoll 리소스를 사용하여 메시에서 프로젝트 추가 또는 제거

OpenShift Container Platform 웹 콘솔과 **ServiceMeshMemberRoll** 리소스를 사용하여 메시에서 프로젝트를 추가하거나 제거할 수 있습니다. 여러 프로젝트를 추가할 수 있지만 프로젝트는 하나의 메시에만 속할 수 있습니다.

해당 ServiceMeshControlPlane 리소스가 삭제되면 ServiceMeshMemberRoll 리소스가 삭제됩니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh Operator 설치 및 검증.
- 기존 ServiceMeshMemberRoll 리소스.
- ServiceMeshMemberRoll 리소스를 사용한 프로젝트의 이름입니다.
- 메시에서 추가하거나 삭제하려는 프로젝트의 이름입니다.

프로세스

1. OpenShift Container Platform 웹 콘솔에 로그인합니다.
2. Operators → 설치된 Operator로 이동합니다.
3. 프로젝트 메뉴를 클릭하고 목록에서 ServiceMeshControlPlane 리소스가 배포된 프로젝트를 선택합니다. 예: istio-system.
4. Red Hat OpenShift Service Mesh Operator를 클릭합니다.
5. Istio Service Mesh 멤버 목록 탭을 클릭합니다.
6. default 링크를 클릭합니다.
7. YAML 탭을 클릭합니다.
8. YAML을 수정하여 프로젝트를 멤버로 추가합니다(또는 기존 멤버를 제거하기 위해 삭제).

여러 프로젝트를 추가할 수 있지만 프로젝트는 하나의 **ServiceMeshMemberRoll** 리소스에만 속할 수 있습니다.

servicemeshmemberroll-default.yaml 예

```
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. 저장을 클릭합니다.

10. 새로 고침을 클릭합니다.

1.9.3.2. CLI와 ServiceMeshMemberRoll 리소스를 사용하여 메시에서 프로젝트 추가 또는 제거

CLI와 함께 **ServiceMeshMemberRoll** 리소스를 사용하여 하나 이상의 프로젝트를 메시에 추가할 수 있습니다. 여러 프로젝트를 추가할 수 있지만 프로젝트는 하나의 메시에만 속할 수 있습니다.

해당 **ServiceMeshControlPlane** 리소스가 삭제되면 **ServiceMeshMemberRoll** 리소스가 삭제됩니다.

사전 요구 사항

- **Red Hat OpenShift Service Mesh Operator** 설치 및 검증.
- 기존 **ServiceMeshMemberRoll** 리소스.

- ServiceMeshMemberRoll 리소스를 사용한 프로젝트의 이름입니다.
- 메시에서 추가하거나 삭제하려는 프로젝트의 이름입니다.
- OpenShift CLI(oc)에 액세스합니다.

프로세스

1. OpenShift Container Platform CLI에 로그인합니다.
2. ServiceMeshMemberRoll 리소스를 편집합니다.

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

3. YAML을 수정하여 프로젝트를 멤버로 추가하거나 제거합니다. 여러 프로젝트를 추가할 수 있지만 프로젝트는 하나의 ServiceMeshMemberRoll 리소스에만 속할 수 있습니다.

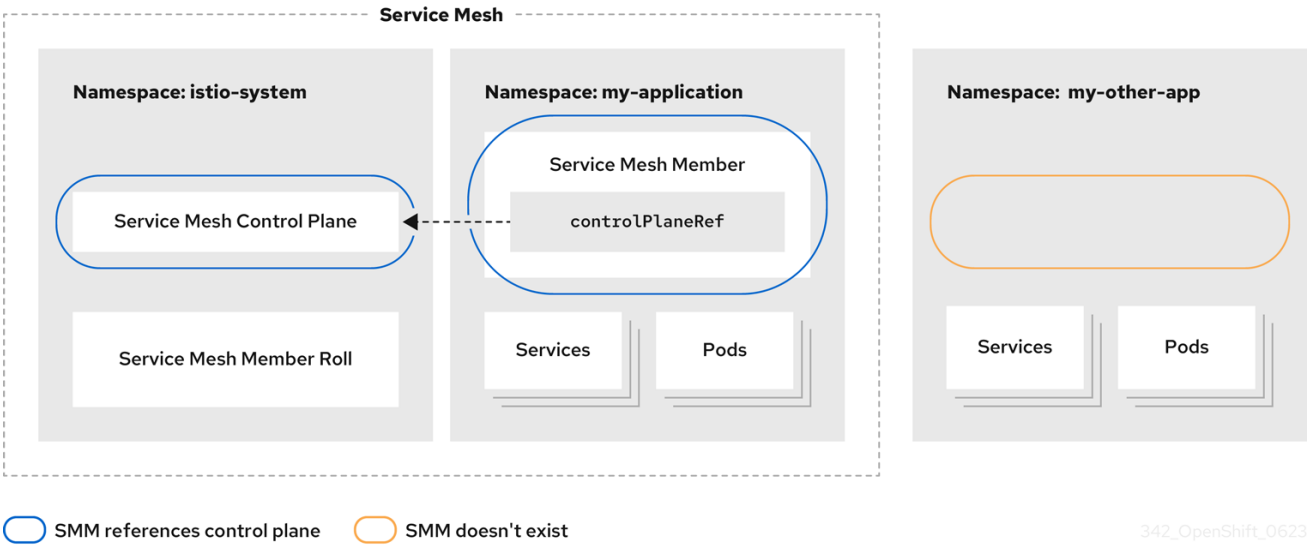
servicemeshmemberroll-default.yaml 예

```
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. 파일을 저장하고 편집기를 종료합니다.

1.9.4. ServiceMeshMember 리소스를 사용하여 프로젝트 추가 정보

ServiceMeshMember 리소스는 **ServiceMeshMemberRoll** 리소스를 수정하지 않고 서비스 메시에 프로젝트를 추가하는 방법을 제공합니다. 프로젝트를 추가하려면 서비스 메시에 추가할 프로젝트에서 **ServiceMeshMember** 리소스를 생성합니다. **Service Mesh Operator**가 **ServiceMeshMember** 오브젝트를 처리하면 프로젝트가 **ServiceMeshMemberRoll** 리소스의 **status.members** 목록에 표시됩니다. 그런 다음 프로젝트에 상주하는 서비스를 메시에서 사용할 수 있습니다.



메시 관리자는 **ServiceMeshMember** 리소스의 **ServiceMeshControlPlane** 리소스를 참조할 수 있는 각 메시 사용자 권한을 부여해야 합니다. 이 권한을 사용하면 해당 사용자에게 서비스 메시 프로젝트 또는 **ServiceMeshMemberRoll** 리소스에 대한 직접 액세스 권한이 없는 경우에도 메시에 프로젝트를 추가할 수 있습니다. 자세한 내용은 **Red Hat OpenShift Service Mesh** 멤버 생성을 참조하십시오.

1.9.4.1. 웹 콘솔과 ServiceMeshMember 리소스를 사용하여 메시에 프로젝트 추가

ServiceMeshMember 리소스를 **OpenShift Container Platform** 웹 콘솔과 함께 사용하여 메시에 하나 이상의 프로젝트를 추가할 수 있습니다.

사전 요구 사항

- **Red Hat OpenShift Service Mesh Operator**가 설치되어 있습니다.
- **ServiceMeshControlPlane** 리소스의 이름과 리소스가 속하는 프로젝트의 이름을 알고 있습니다.
- 메시에 추가할 프로젝트의 이름을 알고 있습니다.
-

서비스 메시 관리자는 서비스 메시에 대한 액세스 권한을 명시적으로 부여해야 합니다. 관리자는 **RoleBinding** 또는 **ClusterRoleBinding** 을 사용하여 **mesh-user Role** 에 할당하여 메시에 액세스할 수 있는 권한을 사용자에게 부여할 수 있습니다. 자세한 내용은 ***Red Hat OpenShift Service Mesh*** *멤버 생성을* 참조하십시오.

프로세스

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. **Operators** → 설치된 **Operator**로 이동합니다.
3. 프로젝트 메뉴를 클릭하고 드롭다운 목록에서 메시에 추가할 프로젝트를 선택합니다. 예를 들면 **istio-system**입니다.
4. **Red Hat OpenShift Service Mesh Operator**를 클릭합니다.
5. **Istio Service Mesh** 멤버 탭을 클릭합니다.
6. **ServiceMeshMember** 만들기를 클릭합니다.
7. **ServiceMeshMember** 의 기본 이름을 수락합니다.
8. **ControlPlaneRef** 를 확장하려면 클릭합니다.
9. 네임스페이스 필드에서 **ServiceMeshControlPlane** 리소스가 속하는 프로젝트를 선택합니다. 예를 들면 **istio-system**입니다.
10. 이름 필드에 이 네임스페이스가 속한 **ServiceMeshControlPlane** 리소스의 이름을 입력합니다. 예**: basic.**
11. 생성을 클릭합니다.
12. **ServiceMeshMember** 리소스가 생성되고 프로젝트가 메시에 추가되었는지 확인합니다. 리

소스 이름을 클릭합니다(예: 기본값). 화면 끝에 표시된 **Conditions** 섹션을 확인합니다. **Reconciled** 및 **Ready** 조건의 상태가 **True** 인지 확인합니다. **Status** 가 **False** 인 경우 자세한 내용은 **Reason** 및 **Message** 열을 참조하십시오.

1.9.4.2. CLI와 ServiceMeshMember 리소스를 사용하여 메시에 프로젝트 추가

CLI와 함께 **ServiceMeshMember** 리소스를 사용하여 메시에 하나 이상의 프로젝트를 추가할 수 있습니다.

사전 요구 사항

- **Red Hat OpenShift Service Mesh Operator**가 설치되어 있습니다.
- **ServiceMeshControlPlane** 리소스의 이름과 해당 리소스의 이름을 알고 있습니다.
- 메시에 추가할 프로젝트의 이름을 알고 있습니다.
- 서비스 메시 관리자는 서비스 메시에 대한 액세스 권한을 명시적으로 부여해야 합니다. 관리자는 **RoleBinding** 또는 **ClusterRoleBinding** 을 사용하여 **mesh-user Role** 에 할당하여 메시에 액세스할 수 있는 권한을 사용자에게 부여할 수 있습니다. 자세한 내용은 ***Red Hat OpenShift Service Mesh*** *멤버 생성을* 참조하십시오.

프로세스

1. **OpenShift Container Platform CLI**에 로그인합니다.
2. **ServiceMeshMember** 매니페스트에 대한 **YAML** 파일을 생성합니다. 매니페스트는 **istio-system** 네임스페이스에 배포된 **ServiceMeshControlPlane** 리소스에서 생성한 서비스 메시에 **my-application** 프로젝트를 추가합니다.

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

3. YAML 파일을 적용하여 ServiceMeshMember 리소스를 생성합니다.

```
$ oc apply -f <file-name>
```

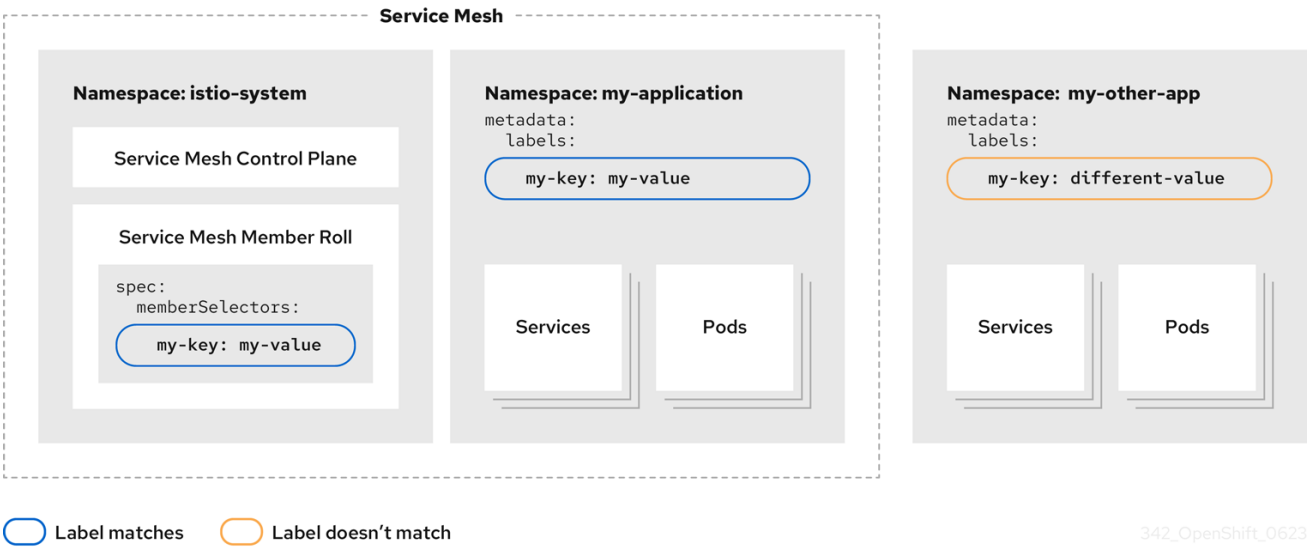
4. ServiceMeshMember 리소스를 생성한 후 네임스페이스가 메시의 일부인지 확인합니다. 다음 명령을 실행할 때 값 True 가 READY 열에 표시되는지 확인합니다.

```
$ oc get smm default -n my-application
```

또는 ServiceMeshMemberRoll 리소스에 액세스할 수 있는 경우, ServiceMeshMemberRoll 리소스의 status.members 및 status.configuredMembers 필드에 my-application 네임스페이스가 표시되는지 확인할 수도 있습니다.

1.9.5. 라벨 선택기를 사용하여 프로젝트 추가 정보

클러스터 전체 배포의 경우 라벨 선택기를 사용하여 메시에 프로젝트를 추가할 수 있습니다. ServiceMeshMemberRoll 리소스에 지정된 라벨 선택기를 사용하면 Service Mesh Operator에서 네임스페이스를 네임스페이스 레이블을 기반으로 또는 메시에 추가하거나 제거할 수 있습니다. 단일 라벨 선택기를 지정하는 데 사용할 수 있는 다른 표준 OpenShift Container Platform 리소스와 달리 ServiceMeshMemberRoll 리소스를 사용하여 여러 라벨 선택기를 지정할 수 있습니다.



네임스페이스의 라벨이 ServiceMeshMemberRoll 리소스에 지정된 선택기와 일치하는 경우 해당 네임스페이스가 메시에 포함됩니다.

참고

OpenShift Container Platform에서 프로젝트는 기본적으로 프로젝트에서 사용할 수 있는 사용자 **ID** 범위와 같은 추가 주석이 있는 **Kubernetes** 네임스페이스입니다. 일반적으로 **OpenShift Container Platform** 웹 콘솔은 *프로젝트* 라는 용어를 사용하고 **CLI**는 *네임스페이스* 를 사용하지만 용어는 본질적으로 무관합니다.

1.9.5.1. 웹 콘솔의 라벨 선택기를 사용하여 메시에 프로젝트 추가

레이블 선택기를 사용하여 **OpenShift Container Platform** 웹 콘솔을 사용하여 서비스 메시에 프로젝트를 추가할 수 있습니다.

사전 요구 사항

- **Red Hat OpenShift Service Mesh Operator**가 설치되어 있습니다.
- 배포에 기존 **ServiceMeshMemberRoll** 리소스가 있습니다.
- **cluster-admin** 역할의 사용자로 로그인되어 있습니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 사용자로 로그인되어 있습니다.

프로세스

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. **Operators** → 설치된 **Operator**로 이동합니다.
3. 프로젝트 메뉴를 클릭하고 드롭다운 목록에서 **ServiceMeshMemberRoll** 리소스가 배포된 프로젝트를 선택합니다. 예: **istio-system.**
4. **Red Hat OpenShift Service Mesh Operator**를 클릭합니다.
5. **Istio Service Mesh** 멤버 목록 탭을 클릭합니다.

6. **ServiceMeshMember Roll** 을 클릭합니다.

7. **ServiceMeshMemberRoll** 의 기본 이름을 수락합니다.

8. 라벨 필드에 키-값 쌍을 입력하여 서비스 메시에 포함할 네임스페이스를 식별하는 레이블을 정의합니다. 프로젝트 네임스페이스에 선택기에 의해 지정된 라벨이 있는 경우 프로젝트 네임스페이스가 서비스 메시에 포함됩니다. 두 레이블을 모두 포함할 필요는 없습니다.

   예를 들어 **mykey=myvalue** 를 입력하면 메시의 일부로 이 라벨이 있는 모든 네임스페이스가 포함됩니다. 선택기가 일치하는 항목을 식별하면 프로젝트 네임스페이스가 서비스 메시에 추가됩니다.

   myotherkey=myothervalue 를 입력하면 메시의 일부로 이 라벨이 있는 모든 네임스페이스가 포함됩니다. 선택기가 일치하는 항목을 식별하면 프로젝트 네임스페이스가 서비스 메시에 추가됩니다.

9. 생성을 클릭합니다.

1.9.5.2. CLI의 라벨 선택기를 사용하여 메시에 프로젝트 추가

레이블 선택기를 사용하여 **CLI**를 사용하여 서비스 메시에 프로젝트를 추가할 수 있습니다.

사전 요구 사항

- **Red Hat OpenShift Service Mesh Operator**가 설치되어 있습니다.
- 배포에 기존 **ServiceMeshMemberRoll** 리소스가 있습니다.
- **cluster-admin** 역할의 사용자로 로그인되어 있습니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 사용자로 로그인되어 있습니다.

절차

1. **OpenShift Container Platform CLI**에 로그인합니다.

2. **ServiceMeshMemberRoll** 리소스를 편집합니다.

```
$ oc edit smmr -n <controlplane_project>
```

이전 예제에서는 **< controlplane_project>**를 예로 사용합니다. 서비스가 포함된 프로젝트와 별도로 **Service Mesh Control Plane**을 모든 프로젝트에 배포할 수 있습니다.

3. **ServiceMeshMemberRoll** 리소스의 **spec.memberSelectors** 필드에 네임스페이스 레이블 선택기를 포함하도록 **YAML** 파일을 수정합니다.

참고

matchLabels 필드를 사용하는 대신 선택기에서 **matchExpressions** 필드를 사용할 수도 있습니다.

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
  namespace: istio-system
spec:
  memberSelectors: 1
  - matchLabels: 2
      mykey: myvalue 3
  - matchLabels: 4
      myotherkey: myothervalue 5
```

1

서비스 메시에 포함된 프로젝트 네임스페이스를 식별하는 데 사용되는 라벨 선택기가 포함되어 있습니다. 프로젝트 네임스페이스에 선택기에 의해 지정된 라벨이 있는 경우 프로젝트 네임스페이스가 서비스 메시에 포함됩니다. 프로젝트 네임스페이스는 두 레이블을 모두 포함할 필요가 없습니다.

2 3

mykey=myvalue 레이블이 있는 모든 네임스페이스를 지정합니다. 선택기가 일치하는 항목을 식별하면 프로젝트 네임스페이스가 서비스 메시에 추가됩니다.

4 5

myotherkey=myothervalue 레이블이 있는 모든 네임스페이스를 지정합니다. 선택기가 일치하는 항목을 식별하면 프로젝트 네임스페이스가 서비스 메시에 추가됩니다.

1.9.6. Bookinfo 예제 애플리케이션

Bookinfo 예제 애플리케이션에서는 OpenShift Container Platform에서 Red Hat OpenShift Service Mesh 2.4.4 설치를 테스트할 수 있습니다.

Bookinfo 애플리케이션은 온라인 서점의 단일 카탈로그 항목과 유사하게 한 권의 책에 대한 정보를 표시합니다. 애플리케이션은 도서 설명, 도서 세부 정보(ISBN, 페이지 수, 기타 정보), 도서 리뷰가 설명된 페이지를 표시합니다.

Bookinfo 애플리케이션은 이러한 마이크로 서비스로 구성됩니다.

- productpage 마이크로 서비스는 details 및 reviews 마이크로 서비스를 호출하여 페이지를 채웁니다.
- details 마이크로 서비스에는 도서 정보가 포함되어 있습니다.
- reviews 마이크로 서비스에는 도서 리뷰가 포함되어 있습니다. 이를 ratings 마이크로 서비스라고도 합니다.
- ratings 마이크로 서비스에는 도서 리뷰와 함께 제공되는 도서 순위 정보가 포함되어 있습니다.

리뷰 마이크로 서비스의 세 가지 버전이 있습니다.

- 버전 v1에서는 ratings 서비스를 호출하지 않습니다.
- 버전 v2는 ratings 서비스를 호출하고 각 평가를 1~5개의 검정별로 표시합니다.
- 버전 v3은 ratings 서비스를 호출하고 각 평가를 1~5개의 빨강별로 표시합니다.

1.9.6.1. Bookinfo 애플리케이션 설치

이 튜토리얼에서는 프로젝트를 생성하고, Bookinfo 애플리케이션을 해당 프로젝트에 배포하고, 서비

스 메시에서 실행 중인 애플리케이션을 확인하여 샘플 애플리케이션을 생성하는 방법을 안내합니다.

사전 요구 사항

- OpenShift Container Platform 4.1 이상이 설치되었습니다.
- Red Hat OpenShift Service Mesh 2.4.4가 설치되었습니다.
- OpenShift CLI(oc)에 액세스합니다.
- cluster-admin 역할이 있는 계정.

참고

Bookinfo 샘플 애플리케이션은 IBM Z 및 IBM Power Systems에 설치할 수 없습니다.

참고

이 섹션의 명령은 Service Mesh Control Plane 프로젝트가 istio-system 이라고 가정합니다. 컨트롤 플레인을 다른 네임스페이스에 설치한 경우 실행하기 전에 각 명령을 편집합니다.

프로세스

1. OpenShift Container Platform 웹 콘솔에 cluster-admin 권한이 있는 사용자로 로그인합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.
2. 홈 → 프로젝트를 클릭합니다.
3. 프로젝트 만들기를 클릭합니다.
4. Project Name 으로 info 를 입력하고 Display Name 을 입력하고 Description 을 입력한 다음 Create 를 클릭합니다.

- 또는 CLI에서 이 명령을 실행하여 info 프로젝트를 생성할 수도 있습니다.

```
$ oc new-project info
```

5. Operators → 설치된 Operators를 클릭합니다.

6. 프로젝트 메뉴를 클릭하고 Service Mesh Control Plane 네임스페이스를 사용합니다. 이 예제에서는 istio-system을 사용합니다.

7. Red Hat OpenShift Service Mesh Operator를 클릭합니다.

8. Istio Service Mesh 멤버 목록 탭을 클릭합니다.

   a. 이미 Istio Service Mesh 멤버 롤을 생성한 경우, 이름을 클릭한 다음 YAML 탭을 클릭하여 YAML 편집기를 엽니다.

   b. ServiceMeshMemberRoll을 생성하지 않은 경우 ServiceMeshMemberRoll 생성을 클릭합니다.

9. Members를 클릭한 다음 Value 필드에 프로젝트 이름을 입력합니다.

10. 생성을 클릭하여 업데이트된 서비스 메시 멤버 롤을 저장합니다.

    a. 또는 다음 예제를 YAML 파일에 저장합니다.

Bookinfo ServiceMeshMemberRoll example servicemeshmemberroll-default.yaml

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
```

```
spec:
  members:
  - info
```

b. 다음 명령을 실행하여 해당 파일을 업로드하고 **istio-system** 네임스페이스에 **ServiceMeshMemberRoll** 리소스를 만듭니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 프로젝트의 이름입니다.

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

11. 다음 명령을 실행하여 **ServiceMeshMemberRoll**이 성공적으로 생성되었는지 확인합니다.

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

STATUS 열이 **Configured**인 경우 설치가 성공적으로 완료된 것입니다.

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

12. **CLI**에서 **bookinfo.yaml** 파일을 적용하여 ***'info'*** 프로젝트에 **Bookinfo** 애플리케이션을 배포합니다.

```
$ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.4/samples/bookinfo/platform/kube/bookinfo.yaml
```

출력은 다음과 유사합니다.

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

```
service/productpage created
serviceaccount/info-productpage created
deployment.apps/productpage-v1 created
```

13. info-gateway.yaml 파일을 적용하여 수신 게이트웨이를 생성합니다.

```
$ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.4/samples/bookinfo/networking/bookinfo-gateway.yaml
```

출력은 다음과 유사합니다.

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

14. GATEWAY_URL 매개변수 값을 설정합니다.

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

1.9.6.2. 기본 대상 규칙 추가

Bookinfo 애플리케이션을 사용하기 전에 먼저 기본 대상 규칙을 추가해야 합니다. 상호 TLS(Transport layer security) 인증을 활성화했는지 여부에 따라 사전 구성된 YAML 파일이 두 개 있습니다.

프로세스

1. 대상 규칙을 추가하려면 다음 명령 중 하나를 실행합니다.

- 상호 TLS를 활성화하지 않은 경우:

```
$ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.4/samples/bookinfo/networking/destination-rule-all.yaml
```

- 상호 TLS를 활성화한 경우:

```
$ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.4/samples/bookinfo/networking/destination-rule-all-mtls.yaml
```

출력은 다음과 유사합니다.

```
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 설치 확인

샘플 Bookinfo 애플리케이션이 성공적으로 배포되었는지 확인하려면 다음 단계를 수행합니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh가 설치되어 있어야 합니다.
- Bookinfo 샘플 애플리케이션을 설치하는 단계를 완료합니다.

CLI의 프로시저

1. OpenShift Container Platform CLI에 로그인합니다.
2. 다음 명령으로 모든 pod가 준비되었는지 확인합니다.

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

모든 pod의 상태는 Running이어야 합니다. 출력은 다음과 유사합니다.

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

3. 다음 명령을 실행하여 제품 페이지의 URL을 검색합니다.

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

4.

웹 브라우저에 출력을 복사하여 붙여넣어 **Bookinfo** 제품 페이지가 배포되었는지 확인합니다.

Kiali 웹 콘솔의 절차

1.

Kiali 웹 콘솔의 주소를 가져옵니다.

a.

OpenShift Container Platform 웹 콘솔에 **cluster-admin** 권한이 있는 사용자로 로그인합니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 계정이 있어야 합니다.

b.

네트워킹 → 경로로 이동합니다.

c.

경로 페이지의 네임스페이스 메뉴에서 **Service Mesh Control Plane** 프로젝트(예: **istio-system**)를 선택합니다.

위치 열은 각 경로에 대한 연결된 주소를 표시합니다.

d.

Kiali의 위치 열에서 링크를 클릭합니다.

e.

OpenShift로 로그인 을 클릭합니다. **Kiali** 개요 화면에 각 프로젝트 네임스페이스에 대한 타일이 표시됩니다.

2.

Kiali에서 그래프 를 클릭합니다.

3.

네임스페이스 목록에서 **info**를 선택하고 그래프 유형 목록에서 앱 그래프 를 선택합니다.

4.

디스플레이 메뉴에서 유휴 노드 표시를 클릭합니다.

이렇게 하면 정의되었지만 요청 수신 또는 전송되지 않은 노드가 표시됩니다. 애플리케이션이 올바르게 정의되었지만 요청 트래픽이 보고되지 않았는지 확인할 수 있습니다.

- **Duration** 메뉴를 사용하여 오래된 트래픽이 캡처되도록 기간을 늘립니다.
- **Refresh Rate** 메뉴를 사용하여 트래픽을 더 자주 새로 고치거나 전혀 새로 고침하지 않습니다.

5. 서비스,워크로드 또는 **Istio Config** 를 클릭하여 정보 구성 요소의 목록 보기를 확인하고 정상인지 확인합니다.

1.9.6.4. Bookinfo 애플리케이션 제거

다음 단계에 따라 **Bookinfo** 애플리케이션을 제거하십시오.

사전 요구 사항

- **OpenShift Container Platform 4.1** 이상이 설치되었습니다.
- **Red Hat OpenShift Service Mesh 2.4.4**가 설치되었습니다.
- **OpenShift CLI(oc)**에 액세스합니다.

1.9.6.4.1. Bookinfo 프로젝트 삭제

절차

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. 홈 → 프로젝트를 클릭합니다.
3.

정보 메뉴

를 클릭한 다음 프로젝트 삭제 를 클릭합니다.

4. 확인 대화 상자에 info 를 입력한 다음 삭제 를 클릭합니다.

- 또는 CLI를 사용하여 이 명령을 실행하여 info 프로젝트를 생성할 수도 있습니다.

```
$ oc delete project info
```

1.9.6.4.2. 서비스 메시 멤버 롤에서 Bookinfo 프로젝트를 제거

절차

1. OpenShift Container Platform 웹 콘솔에 로그인합니다.

2. Operators → 설치된 Operators를 클릭합니다.

3. 프로젝트 메뉴를 클릭하고 목록에서 istio-system 을 선택합니다.

4. Red Hat OpenShift Service Mesh Operator에 대해 제공된 APIS에서 Istio Service Mesh 멤버 롤 링크를 클릭합니다.

5. ServiceMeshMemberRoll 메뉴

를 클릭하고 서비스 메시 멤버 롤 편집 을 선택합니다.

6. 기본 서비스 메시 멤버 롤 YAML을 편집하고 멤버 목록에서 정보를 제거합니다.

- 또는 CLI를 사용하여 이 명령을 실행하여 ServiceMeshMemberRoll 에서 info 프로젝트를 제거할 수 있습니다. 이 예제에서 istio-system 은 Service Mesh Control Plane 프로젝트의 이름입니다.

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

7. 저장을 클릭하여 서비스 메시 멤버 롤을 업데이트합니다.

1.9.7. 다음 단계

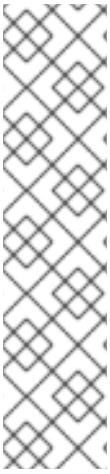
- 설치 프로세스를 계속 진행하려면 사이드카 삽입을 활성화해야 합니다.

1.10. 사이드카 삽입 활성화

메시에 서비스가 포함된 네임스페이스를 추가한 후 다음 단계는 애플리케이션의 **Deployment** 리소스에서 자동 사이드카 삽입을 활성화하는 것입니다. 각 배포에 대해 자동 사이드카 삽입을 활성화해야 합니다.

Bookinfo 샘플 애플리케이션을 설치한 경우 애플리케이션이 배포되고 사이드카가 설치 절차의 일부로 삽입되었습니다. 자체 프로젝트 및 서비스를 사용하는 경우 **OpenShift Container Platform**에 애플리케이션을 배포합니다.

자세한 내용은 **OpenShift Container Platform** 문서, 배포 및 **DeploymentConfig** 오브젝트 이해를 참조하십시오.



참고

Pod의 애플리케이션 컨테이너보다 먼저 실행되는 특수 컨테이너인 **Init Container**에서 시작한 트래픽은 기본적으로 서비스 메시 외부에서 이동할 수 없습니다. 메시 외부에서 네트워크 트래픽 연결을 설정해야 하는 모든 작업 **Init Container**가 수행됩니다.

Init Container를 서비스에 연결하는 방법에 대한 자세한 내용은 **Service Mesh** 사이드카가 삽입된 **Pod**의 **CrashLoopBackOff**의 **Red Hat Knowledgebase** 솔루션 **initContainer**를참조하십시오.

1.10.1. 사전 요구 사항

- 메시에 배포된 서비스(예: **Bookinfo** 샘플 애플리케이션)

- 배포 리소스 파일.

1.10.2. 자동 사이드카 삽입 활성화

애플리케이션을 배포할 때 배포 오브젝트에서 **spec.template.metadata.annotations** 에서 **true**로 주석 **sidecar.istio.io/inject** 를 **true** 로 구성하여 삽입을 선택해야 합니다. 이 설정을 통해 사이드카 삽입이 **OpenShift Container Platform** 에코시스템 내 여러 프레임 워크에서 사용되는 **builder pod**와 같은 다른 **OpenShift Container Platform** 기능을 방해하지 않도록 할 수 있습니다.

사전 요구 사항

- 서비스 메시의 일부인 네임스페이스와 자동 사이드카 삽입이 필요한 배포를 식별합니다.

절차

1. 배포를 찾으려면 **oc get** 명령을 사용합니다.

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

예를 들어 **info** 네임스페이스에서 **'ratings-v1'** 마이크로 서비스에 대한 배포 파일을 보려면 다음 명령을 사용하여 **YAML** 형식의 리소스를 확인합니다.

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

2. 편집기에서 애플리케이션의 배포 구성 **YAML** 파일을 엽니다.

3. 다음 예와 같이 **spec.template.metadata.annotations.sidecar.istio/inject** 를 **Deployment YAML**에 추가하고 **sidecar.istio.io/inject** 를 **true** 로 설정합니다.

info deployment-ratings-v1.yaml의 스니펫 예

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

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

4. 배포 구성 파일을 저장합니다.

5. 앱이 포함된 프로젝트에 파일을 다시 추가합니다.

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

이 예제에서 **info** 는 **ratings-v1 app** 및 **deployment-ratings-v1.yaml** 이 포함된 프로젝트의 이름입니다.

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

6. 리소스가 업로드되었는지 확인하려면 다음 명령을 실행합니다.

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

예를 들면 다음과 같습니다.

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

1.10.3. 사이드카 삽입 검증

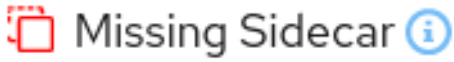
Kiali 콘솔은 애플리케이션, 서비스 및 워크로드에 사이드카 프록시가 있는지 여부를 확인하는 여러 가지 방법을 제공합니다.

그림 1.3. 누락된 사이드카 배지

그래프 페이지에는 다음 그래프에서 **Missing Sidecar** 를 나타내는 노드 배지가 표시됩니다.

- 앱 그래프
- 버전이 지정된 앱 그래프
- 워크로드 그래프

그림 1.4. 누락된 사이드카 아이콘



애플리케이션 페이지에는 사이드카가 없는 네임스페이스의 애플리케이션에 대한 세부 정보 열에 있는 **Missing Sidecar** 아이콘이 표시됩니다.

워크로드 페이지에는 사이드카가 없는 네임스페이스의 애플리케이션에 대한 세부 정보 열에 **Missing Sidecar** 아이콘이 표시됩니다.

서비스 페이지에는 사이드카가 없는 네임스페이스의 애플리케이션에 대한 세부 정보 열에 있는 **Missing Sidecar** 아이콘이 표시됩니다. 서비스의 여러 버전이 있는 경우 서비스 세부 정보 페이지를 사용하여 **Missing Sidecar** 아이콘을 확인합니다.

워크로드 세부 정보 페이지에는 애플리케이션과 프록시 로그 를 보고 연관시킬 수 있는 특수한 통합 로

그 탭이 있습니다. 애플리케이션 워크로드에 대한 사이드카 삽입의 유효성을 검사하는 다른 방법으로 Envoy 로그를 볼 수 있습니다.

또한 워크로드 세부 정보 페이지에는 Envoy 프록시인 모든 워크로드에 대한 Envoy 탭이나 Envoy 프록시와 함께 삽입되어 있습니다. 이 탭에는 클러스터,Listeners,Routes,Bootstrap,Config 및 Metrics 에 대한 하위 탭이 포함된 기본 제공 Envoy 대시보드가 표시됩니다.

Envoy 액세스 로그 활성화에 대한 자세한 내용은 문제 해결 섹션을 참조하십시오.

Envoy 로그를 보는 방법에 대한 자세한 내용은 Kiali 콘솔에서 로그보기를 참조하십시오.

1.10.4. 주석을 통해 프록시 환경 변수 설정

Envoy 사이드카 프록시에 대한 구성은 ServiceMeshControlPlane 에서 관리합니다.

injection-template.yaml 파일의 배포에 Pod 주석을 추가하여 애플리케이션의 사이드카 프록시의 환경 변수를 설정할 수 있습니다. 환경 변수가 사이드카에 삽입됩니다.

예: injection-template.yaml

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

주의

고유한 사용자 정의 리소스를 생성할 때 maistra.io/ 레이블 및 주석을 포함하지 않아야 합니다. 이러한 라벨 및 주석은 Operator에서 리소스를 생성하고 관리함을 나타냅니다. 자체 리소스를 생성할 때 Operator 생성 리소스에서 콘텐츠를 복사하는 경우 maistra.io/ 로 시작하는 레이블 또는 주석을 포함하지 마십시오. 이러한 라벨 또는 주석을 포함하는 리소스는 다음 조정 중에 Operator에 의해 덮어쓰거나 삭제됩니다.

1.10.5. 사이드카 프록시 업데이트

사이드카 프록시의 구성을 업데이트하려면 애플리케이션 관리자가 애플리케이션 Pod를 다시 시작해야 합니다.

배포 시 자동 사이드카 삽입을 사용하는 경우 주석을 추가하거나 수정하여 배포에서 pod 템플릿을 업데이트할 수 있습니다. 다음 명령을 실행하여 pod를 다시 배포합니다.

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

배포에서 자동 사이드카 삽입을 사용하지 않는 경우 배포 또는 Pod에 지정된 사이드카 컨테이너 이미지를 수정하여 사이드카를 수동으로 업데이트한 다음 Pod를 다시 시작해야 합니다.

1.10.6. 다음 단계

환경에 맞게 Red Hat OpenShift Service Mesh 기능을 구성합니다.

- 보안
- 트래픽 관리
- 메트릭, 로그 및 추적

1.11. 서비스 메시 업그레이드

Red Hat OpenShift Service Mesh의 최신 기능에 액세스하려면 현재 버전 2.4.4로 업그레이드합니다.

1.11.1. 버전 이해

Red Hat은 제품 릴리스에 의미 체계 버전 버전을 사용합니다. 의미 체계 버전 지정은 X.Y.Z 형식의 3 구성 요소 번호입니다.

- X는 주요 버전을 나타냅니다. 주요 릴리스는 일반적으로 아키텍처 변경, API 변경, 스키마 변경, 유사한 주요 업데이트 등 일부 종류의 변경 사항을 나타냅니다.
- Y는 마이너 버전을 나타냅니다. 마이너 릴리스에는 이전 버전과의 호환성을 유지하면서 새로운 기능과 기능이 포함되어 있습니다.
- Z는 Patch 버전 (z-stream 릴리스라고도 함)을 나타냅니다. 패치 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 릴리스 버그 수정을 해결하는 데 사용됩니다. 새로운 기능 및 기능은 일반적으로 패치 릴리스의 일부로 출시되지 않습니다.

1.11.1.1. 버전 관리로 서비스 메시 업그레이드에 영향을 미치는 방법

수행 중인 업데이트 버전에 따라 업그레이드 프로세스가 다릅니다.

- 패치 업데이트 - 패치 업그레이드는 OLM(Operator Lifecycle Manager)에 의해 관리됩니다. Operator를 업데이트할 때 자동으로 수행됩니다.
- 마이너 업그레이드 - 마이너 업그레이드에서는 둘 다 최신 Red Hat OpenShift Service Mesh Operator 버전으로 업데이트하고 ServiceMeshControlPlane 리소스에서 spec.version 값을 수동으로 수정해야 합니다.
- 주요 업그레이드 - 주요 업그레이드에서는 둘 다 최신 Red Hat OpenShift Service Mesh Operator 버전으로 업데이트하고 ServiceMeshControlPlane 리소스에서 spec.version 값을 수동으로 수정해야 합니다. 주요 업그레이드에는 이전 버전과 호환되지 않는 변경 사항이 포함될 수 있으므로 추가 수동 변경이 필요할 수 있습니다.

1.11.1.2. 서비스 메시 버전 이해

시스템에 배포한 Red Hat OpenShift Service Mesh 버전을 이해하려면 각 구성 요소 버전이 어떻게

관리되는지 이해해야 합니다.

- Operator 버전 - 최신 Operator 버전은 2.4.4입니다. Operator 버전 번호는 현재 설치된 Operator의 버전만 나타냅니다. Red Hat OpenShift Service Mesh Operator는 Service Mesh Control Plane의 여러 버전을 지원하므로 Operator 버전이 배포된 ServiceMeshControlPlane 리소스의 버전을 결정하지 않습니다.

중요

최신 Operator 버전으로 업그레이드하면 패치 업데이트가 자동으로 적용되지만 서비스 메시 컨트롤 플레인을 최신 마이너 버전으로 자동 업그레이드하지는 않습니다.

- ServiceMeshControlPlane 버전 - ServiceMeshControlPlane 버전은 사용 중인 Red Hat OpenShift Service Mesh 버전을 결정합니다. ServiceMeshControlPlane 리소스의 spec.version 필드 값은 Red Hat OpenShift Service Mesh를 설치하고 배포하는 데 사용되는 아키텍처 및 구성 설정을 제어합니다. Service Mesh Control Plane을 생성할 때 다음 두 가지 방법 중 하나로 버전을 설정할 수 있습니다.
  - 양식 보기에서 구성하려면 컨트롤 플레인 버전 메뉴에서 버전을 선택합니다.
  - YAML 보기에서 구성하려면 YAML 파일에서 spec.version 값을 설정합니다.

OLM(Operator Lifecycle Manager)은 서비스 메시 컨트롤 플레인 업그레이드를 관리하지 않으므로 SMCP를 수동으로 업그레이드하지 않으면 Operator 및 ServiceMeshControlPlane (SMCP)의 버전 번호가 일치하지 않을 수 있습니다.

1.11.2. 업그레이드 고려 사항

maistra.io/ 레이블 또는 주석은 사용자가 생성한 사용자 정의 리소스에 사용하지 않아야 하며 Red Hat OpenShift Service Mesh Operator에서 리소스를 관리해야 함을 나타냅니다.

주의

업그레이드 중에 Operator는 파일을 삭제하거나 교체하는 등 Operator에서 리소스를 관리함을 나타내는 다음 라벨 또는 주석을 포함하는 리소스에 대한 변경을 수행합니다.

업그레이드 전에 다음 라벨 또는 주석을 포함하는 사용자가 생성한 사용자 정의 리소스를 확인합니다.

- maistra.io/ AND app.kubernetes.io/managed-by 레이블 maistra-istio-operator (Red Hat OpenShift Service Mesh)
- Kiali.io/ (Kiali)
- Jaegertracing.io/ (Red Hat OpenShift distributed tracing platform (Jaeger))
- logging.openshift.io/ (Red Hat Elasticsearch)

업그레이드하기 전에 사용자가 생성한 사용자 정의 리소스에서 Operator가 관리됨을 나타내는 라벨 또는 주석을 확인합니다. Operator에서 관리하지 않으려는 사용자 정의 리소스에서 레이블 또는 주석을 제거합니다.

버전 2.0으로 업그레이드할 때 Operator는 SMCP와 동일한 네임스페이스에 이러한 라벨이 있는 리소스만 삭제합니다.

버전 2.1으로 업그레이드할 때 Operator는 모든 네임스페이스에서 이러한 라벨을 사용하여 리소스를 삭제합니다.

1.11.2.1. 업그레이드에 영향을 줄 수 있는 알려진 문제

업그레이드에 영향을 미칠 수 있는 알려진 문제는 다음과 같습니다.

- Operator를 업그레이드할 때 Jaeger 또는 Kiali의 사용자 정의 구성이 되돌릴 수 있습니다. Operator를 업그레이드하기 전에 Service Mesh 프로덕션 배포에서 Jaeger 또는 Kiali 오브젝트에 대한 사용자 정의 구성 설정을 기록하여 다시 생성할 수 있습니다.

- Red Hat OpenShift Service Mesh는 명시적으로 문서화된 경우를 제외하고 EnvoyFilter 구성 사용을 지원하지 않습니다. 이는 기본 Envoy API와 긴밀하게 결합되므로 이전 버전과의 호환성을 유지할 수 없습니다. Envoy Filters를 사용하며 Istio에서 생성된 구성이 ServiceMeshControlPlane 을 업그레이드하여 도입된 Envoy의 마지막 버전으로 인해 변경된 경우 구현되었을 수 있는 EnvoyFilter 를 중단할 가능성이 있습니다.

- OSSM-1505 ServiceMeshExtension 는 OpenShift Container Platform 버전 4.11에서 작동하지 않습니다. ServiceMeshExtension 는 Red Hat OpenShift Service Mesh 2.2에서 더 이상 사용되지 않기 때문에 알려진 문제는 수정되지 않으며, 와s mPluging으로 확장을 마이그레이션해야 합니다.

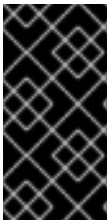
- OSSM-1396 게이트웨이 리소스에 ServiceMeshControlPlane 을 업데이트할 때 다시 생성하는 대신 spec.externalIPs 설정이 포함된 경우 게이트웨이가 제거되고 다시 생성되지 않습니다.

- OSSM-1052 서비스 메시 컨트롤 플레인에서 ingressgateway에 대해 서비스 ExternalIP 를 구성할 때 서비스가 생성되지 않습니다. SMCP의 스키마에 서비스 매개변수가 누락되어 있습니다.

  해결방법: SMCP 사양에서 게이트웨이 생성을 비활성화하고 서비스, 역할 및 RoleBinding 을 포함하여 수동으로 게이트웨이 배포를 완전히 관리합니다.

1.11.3. Operator 업그레이드

서비스 메시를 최신 보안 수정, 버그 수정 및 소프트웨어 업데이트로 패치하려면 Operator를 업데이트해야 합니다. Operator를 업그레이드하여 패치 업데이트를 시작합니다.



중요

Operator 버전이 서비스 메시 버전을 결정하지 않습니다. 배포된 Service Mesh Control Plane의 버전에 따라 서비스 메시 버전이 결정됩니다.

Red Hat OpenShift Service Mesh Operator는 Service Mesh Control Plane의 여러 버전을 지원하므로 Red Hat OpenShift Service Mesh Operator를 업데이트해도 배포된 ServiceMeshControlPlane

의 **spec.version** 값이 업데이트 *되지* 않습니다. **spec.version** 값은 두 자리 숫자(예: **2.2**)이며 패치 업데이트(예: **2.2.1**)는 **SMCP** 버전 값에 반영되지 않습니다.

OLM(Operator Lifecycle Manager)은 클러스터에서 **Operator**의 설치, 업그레이드, **RBAC**(역할 기반 액세스 제어)를 제어합니다. **OLM**은 **OpenShift Container Platform**에서 기본적으로 실행됩니다. 사용 가능한 **Operator** 및 설치된 **Operator**의 업그레이드에 대한 **OLM** 쿼리입니다.

Operator 업그레이드를 수행해야 하는지 여부는 설치 시 선택한 설정에 따라 다릅니다. 각 **Operator**를 설치하면 업데이트 채널 과 승인 전략을 선택했습니다. 이 두 설정의 조합은 **Operator** 업데이트 시기와 방법을 결정합니다.

표 1.4. 업데이트 채널 및 승인 전략의 상호 작용

	버전 지정된 채널	"stable" 또는 "Preview" 채널
자동	해당 버전의 마이너 릴리스 및 패치 릴리스에 대해서만 Operator를 자동으로 업데이트합니다. 는 다음 주요 버전 (즉, 버전 2.0에서 3.0으로) 자동으로 업데이트되지 않습니다. 다음 주요 버전으로 업데이트하는 데 필요한 Operator 서브스크립션을 수동으로 변경해야 합니다.	모든 주요, 마이너 및 패치 릴리스에 대해 Operator를 자동으로 업데이트합니다.
수동	지정된 버전에 대한 마이너 및 패치 릴리스에 필요한 수동 업데이트입니다. 다음 주요 버전으로 업데이트하는 데 필요한 Operator 서브스크립션을 수동으로 변경해야 합니다.	모든 주요, 마이너 및 패치 릴리스에 필요한 수동 업데이트입니다.

Red Hat OpenShift Service Mesh Operator를 업데이트하면 **OLM(Operator Lifecycle Manager)**이 이전 **Operator Pod**를 제거하고 새 **Pod**를 시작합니다. 새 **Operator Pod**가 시작되면 조정 프로세스에서 **ServiceMeshControlPlane (SMCP)**을 확인하고 **Service Mesh Control Plane** 구성 요소에 사용 가능한 업데이트된 컨테이너 이미지가 있는 경우 해당 **Service Mesh Control Plane Pod**를 새 컨테이너 이미지를 사용하는 것으로 교체합니다.

Kiali 및 **Red Hat OpenShift distributed tracing Platform(Jaeger) Operator**를 업그레이드할 때 **OLM** 조정 프로세스는 클러스터를 검사하고 관리 인스턴스를 새 **Operator**의 버전으로 업그레이드합니다. 예를 들어 **Red Hat OpenShift distributed tracing Platform(Jaeger) Operator**를 버전 **1.30.2**에서 버전 **1.34.1**로 업데이트하는 경우 **Operator**는 분산 추적 플랫폼(**Jaeger**)의 인스턴스를 실행하고 **1.34.1**로 업그레이드합니다.

Red Hat OpenShift Service Mesh의 특정 패치 버전을 유지하려면 자동 업데이트를 비활성화하고 해당 특정 버전의 **Operator**에 남아 있어야 합니다.

Operator 업그레이드에 대한 자세한 내용은 Operator Lifecycle Manager 설명서를 참조하십시오.

1.11.4. 컨트롤 플레인 업그레이드

마이너 릴리스 및 주요 릴리스를 위해 컨트롤 플레인을 수동으로 업데이트해야 합니다. 커뮤니티 Istio 프로젝트는 카나리아 업그레이드를 권장합니다. Red Hat OpenShift Service Mesh는 인플레이스 업그레이드만 지원합니다. Red Hat OpenShift Service Mesh를 사용하려면 각 마이너 릴리스에서 다음 마이너 릴리스로 업그레이드해야 합니다. 예를 들어 버전 2.0에서 버전 2.1으로 업그레이드한 다음 버전 2.2로 업그레이드해야 합니다. Red Hat OpenShift Service Mesh 2.0에서 2.2로 직접 업데이트할 수 없습니다.

서비스 메시 컨트롤 플레인을 업그레이드할 때 모든 Operator 관리 리소스(예: 게이트웨이)도 업그레이드됩니다.

동일한 클러스터에 여러 버전의 컨트롤 플레인을 배포할 수 있지만 Red Hat OpenShift Service Mesh는 서비스 메시의 카나리아 업그레이드를 지원하지 않습니다. 즉, spec.version 에 대해 다양한 SCMP 리소스가 있을 수 있지만 동일한 메시를 관리할 수는 없습니다.

확장 기능에 대한 자세한 내용은 ServiceMeshExtension에서wasmPlugin 리소스로 마이그레이션을 참조하십시오.

1.11.4.1. 버전 2.3에서 버전 2.4로 업그레이드

Service Mesh Control Plane을 버전 2.3에서 2.4로 업그레이드하면 다음과 같은 동작 변경 사항이 추가되었습니다.

- Istio OpenShift Routing (IOR)에 대한 지원은 더 이상 사용되지 않습니다. IOR 기능은 여전히 활성화되어 있지만 향후 릴리스에서 제거될 예정입니다.
- 다음 암호화 제품군은 더 이상 지원되지 않으며 클라이언트 및 서버 측 TLS 협상에 사용되는 암호 목록에서 제거되었습니다.
  - 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

이러한 암호화 제품군 중 하나를 사용하는 서비스에 액세스해야 하는 애플리케이션은 프록시가 TLS 연결을 시작할 때 연결할 수 없습니다.

1.11.4.2. 버전 2.2에서 버전 2.3으로 변경 사항 업그레이드

Service Mesh Control Plane을 버전 2.2에서 2.3으로 업그레이드하면 다음과 같은 동작 변경이 발생합니다.

- 이 릴리스에는 kubectls mPlugin API 를 사용해야 합니다. 2.2에서 더 이상 사용되지 않는 ServiceMeshExtension API에 대한 지원이 제거되었습니다. ServiceMeshExtension API를 사용하는 동안 업그레이드를 시도하면 업그레이드가 실패합니다.

1.11.4.3. 버전 2.1에서 버전 2.2로 업그레이드 변경

Service Mesh Control Plane을 버전 2.1에서 2.2로 업그레이드하면 다음과 같은 동작이 변경되었습니다.

- istio-node DaemonSet은 업스트림 Istio의 이름과 일치하도록 istio-cni-node 로 변경됩니다.
- Istio 1.10은 기본적으로 lo 대신 eth0 을 사용하여 애플리케이션 컨테이너로 트래픽을 전송하도록 Envoy를 업데이트했습니다.

- 이 릴리스에서는ECDHEs mPlugin API에 대한 지원이 추가되어 ServiceMeshExtension API를 더 이상 사용하지 않습니다.

1.11.4.4. 버전 2.0에서 버전 2.1으로의 업그레이드 변경

Service Mesh Control Plane을 버전 2.0에서 2.1으로 업그레이드하면 다음과 같은 아키텍처 및 동작 변경 사항이 도입되었습니다.

아키텍처 변경

Red Hat OpenShift Service Mesh 2.1에서 Mixer가 완전히 제거되었습니다. Mixer가 활성화된 경우 Red Hat OpenShift Service Mesh 2.0.x 릴리스에서 2.1으로 업그레이드가 차단됩니다.

v2.0에서 v2.1로 업그레이드할 때 다음 메시지가 표시되면 .spec.version 필드를 업데이트하기 전에 기존 Control Plane 사양의 기존 Mixer 유형을 Istiod 유형으로 업데이트합니다.

```
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]”
```

예를 들어 다음과 같습니다.

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  policy:
    type: Istiod
  telemetry:
    type: Istiod
  version: v2.4
```

동작 변경

- AuthorizationPolicy 업데이트:
  - PROXY 프로토콜에서는 ipBlocks 및 notIpBlocks 를 사용하여 원격 IP 주소를 지정하는 경우 대신 remoteIpBlocks 및 notRemoteIpBlocks 를 사용하도록 구성을 업데이트합니

다.

- 중첩된 JSON 웹 토큰(JWT)에 대한 지원이 추가되었습니다.

- EnvoyFilter 변경 사항 중단>
  - typed_config를 사용해야 합니다.
  - XDS v2는 더 이상 지원되지 않습니다.
  - 더 이상 사용되지 않는 필터 이름

- 이전 버전의 프록시는 최신 프록시에서 1xx 또는 204 상태 코드를 수신할 때 503 상태 코드를 보고할 수 있습니다.

1.11.4.5. Service Mesh Control Plane 업그레이드

Red Hat OpenShift Service Mesh를 업그레이드하려면 Red Hat OpenShift Service Mesh ServiceMeshControlPlane v2 리소스의 version 필드를 업데이트해야 합니다. 그런 다음 구성 및 적용되면 애플리케이션 Pod를 다시 시작하여 각 사이드카 프록시와 해당 구성을 업데이트합니다.

사전 요구 사항

- OpenShift Container Platform 4.9 이상에서 실행 중입니다.
- 최신 Red Hat OpenShift Service Mesh Operator가 있습니다.

프로세스

1. ServiceMeshControlPlane 리소스가 포함된 프로젝트로 전환합니다. 이 예제에서 istio-system 은 Service Mesh Control Plane 프로젝트의 이름입니다.

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

2. v2 ServiceMeshControlPlane 리소스 구성을 확인하여 유효한지 확인합니다.

a. 다음 명령을 실행하여 ServiceMeshControlPlane 리소스를 v2 리소스로 확인합니다.

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

작은 정보

Service Mesh Control Plane 구성을 백업합니다.

3. .spec.version 필드를 업데이트하고 구성을 적용합니다.

예를 들어 다음과 같습니다.

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.4
```

또는 명령줄을 사용하는 대신 웹 콘솔을 사용하여 Service Mesh Control Plane을 편집할 수도 있습니다. OpenShift Container Platform 웹 콘솔에서 프로젝트를 클릭하고 방금 입력한 프로젝트 이름을 선택합니다.

a. Operators → 설치된 Operator를 클릭합니다.

b. ServiceMeshControlPlane 인스턴스를 찾습니다.

c. 이전 예제와 같이 YAML 보기를 선택하고 YAML 파일의 텍스트를 업데이트합니다.

d. 저장을 클릭합니다.

1.11.4.6. Red Hat OpenShift Service Mesh를 버전 1.1에서 버전 2.0으로 마이그레이션

버전 1.1에서 2.0으로 업그레이드하려면 워크로드와 애플리케이션을 새 버전을 실행하는 Red Hat OpenShift Service Mesh의 새 인스턴스로 마이그레이션하는 수동 단계가 필요합니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh 2.0으로 업그레이드하려면 OpenShift Container Platform 4.7로 업그레이드해야 합니다.
- Red Hat OpenShift Service Mesh 버전 2.0 operator가 있어야 합니다. 자동 업그레이드 경로를 선택한 경우 Operator는 최신 정보를 자동으로 다운로드합니다. 하지만 Red Hat OpenShift Service Mesh 버전 2.0에서 기능을 사용하려면 몇 가지 단계를 거쳐야 합니다.

1.11.4.6.1. Red Hat OpenShift Service Mesh 업그레이드

Red Hat OpenShift Service Mesh를 업그레이드하려면 새 네임스페이스에 Red Hat OpenShift Service Mesh ServiceMeshControlPlane v2 리소스 인스턴스를 생성해야 합니다. 구성되면 이전 메시에서 새로운 서비스 메시로 마이크로 서비스 애플리케이션과 워크로드를 이동하십시오.

프로세스

1. v1 ServiceMeshControlPlane 리소스 구성을 점검하여 유효한지 확인합니다.
   a. 다음 명령을 실행하여 ServiceMeshControlPlane 리소스를 v2 리소스로 확인합니다.

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

   b. 유효하지 않은 필드에 대한 정보는 출력의 spec.techPreview.errored.message 필드를 확인합니다.
   c. v1 리소스에 유효하지 않은 필드가 있으면 리소스가 조정되지 않고 v2 리소스로 편집할 수 없습니다. v2 필드에 대한 모든 업데이트는 원래 v1 설정으로 덮어씁니다. 유효하지 않은 필드를 수정하려면 리소스의 v1 버전을 교체, 패치 또는 편집할 수 있습니다. 또한 수정하지 않고 리소스를 삭제할 수도 있습니다. 리소스가 수정된 후 조정할 수 있으며, v2 버전의 리소스를 수정하거나 볼 수 있습니다.
   d. 파일을 편집하여 리소스를 수정하려면 oc get를 사용하여 리소스를 검색하고, 로컬로 텍스트 파일을 편집한 다음, 편집한 파일로 리소스를 교체합니다.

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

e. 패치를 사용하여 리소스를 수정하려면 **oc patch**를 사용합니다.

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

f. 명령줄 도구로 리소스를 수정하려면 **oc edit**를 사용합니다.

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

2. **Service Mesh Control Plane** 구성을 백업합니다. **ServiceMeshControlPlane** 리소스가 포함된 프로젝트로 전환합니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 프로젝트의 이름입니다.

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

3. 다음 명령을 입력하여 현재 구성을 검색할 수 있습니다. **<smcp_name>**은 **ServiceMeshControlPlane** 리소스의 메타데이터에 지정됩니다(예: **basic-install** 또는 **full-install**).

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

4. **ServiceMeshControlPlane**을 구성에 대한 정보를 시작점으로 포함하는 **v2** 컨트롤 플레인 버전으로 변환합니다.

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

5. 프로젝트를 생성합니다. **OpenShift Container Platform** 콘솔 프로젝트 메뉴에서 **New Project**를 클릭하고 프로젝트 이름(예: **istio-system-upgrade**)을 입력합니다. 또는 **CLI**에서 이 명령을 실행할 수 있습니다.

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

6. **v2 ServiceMeshControlPlane**의 **metadata.namespace** 필드를 새 프로젝트 이름으로 업데이트합니다. 이 예제에서는 **istio-system-upgrade**를 사용합니다.

7. 1.1에서 2.0으로 version 필드를 업데이트하거나 v2 ServiceMeshControlPlane에서 제거합니다.

8. 새 네임스페이스에서 ServiceMeshControlPlane을 생성합니다. 명령줄에서 다음 명령을 실행하여 검색한 ServiceMeshControlPlane의 v2 버전을 사용하여 컨트롤 플레인을 배포합니다. 이 예제에서 '<smcp_name.v2>'를 파일 경로로 바꿉니다.

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

또는 콘솔을 사용하여 서비스 메시 컨트롤 플레인을 생성할 수 있습니다. OpenShift Container Platform 웹 콘솔에서 프로젝트를 클릭합니다. 그런 다음, 방금 입력한 프로젝트 이름을 선택합니다.

a. Operators → 설치된 Operator를 클릭합니다.

b. ServiceMeshControlPlane 만들기를 클릭합니다.

c. YAML 보기를 선택하고, 검색한 YAML 파일의 텍스트를 필드에 붙여넣습니다. apiVersion 필드가 maistra .io/v2로 설정되어 있는지 확인하고 새 네임스페이스를 사용하도록 metadata.namespace 필드를 수정합니다(예: istio-system-upgrade).

d. 생성을 클릭합니다.

1.11.4.6.2. 2.0 ServiceMeshControlPlane 구성

Red Hat OpenShift Service Mesh 버전 2.0에서 ServiceMeshControlPlane 리소스가 변경되었습니다. ServiceMeshControlPlane 리소스의 v2 버전을 생성한 후 새 기능을 활용하고 배포에 적합하게 수정합니다. ServiceMeshControlPlane 리소스를 수정할 때 Red Hat OpenShift Service Mesh 2.0의 사양 및 동작에 대해 다음과 같은 변경 사항을 고려하십시오. 또한 사용하는 기능에 대한 새로운 정보는 Red Hat OpenShift Service Mesh 2.0 제품 문서를 참조하십시오. v2 리소스는 Red Hat OpenShift Service Mesh 2.0 설치에 사용해야 합니다.

1.11.4.6.2.1. 아키텍처 변경

이전 버전에서 사용하는 아키텍처 단위는 Istiod로 교체되었습니다. 2.0에서 서비스 메시 컨트롤 플레인 구성 요소 Mixer, Pilot, Citadel, Galley, 사이드카 인젝터 기능이 단일 구성 요소인 Istiod로 결합되었습니다.

Mixer는 더 이상 컨트롤 플레인 구성 요소로 지원되지 않지만, Mixer 정책 및 Telemetry 플러그인은 이제 Istiod의 WASM 확장을 통해 지원됩니다. 레거시 Mixer 플러그인을 통합해야 하는 경우 정책 및 Telemetry에 대해 Mixer를 활성화할 수 있습니다.

SDS(Secret Discovery Service)는 Istiod에서 직접 사이드카에 인증서와 키를 배포하는 데 사용됩니다. Red Hat OpenShift Service Mesh 버전 1.1에서는 Citadel에 의해 시크릿이 생성되었으며, 이는 프록시가 클라이언트 인증서와 키를 검색하는 데 사용되었습니다.

1.11.4.6.2.2. 주석 변경

v2.0에서는 다음과 같은 주석이 더 이상 지원되지 않습니다. 이러한 주석 중 하나를 사용하는 경우 v2.0 Service Mesh Control Plane으로 이동하기 전에 워크로드를 업데이트해야 합니다.

- sidecar.maistra.io/proxyCPULimit은 sidecar.istio.io/proxyCPULimit로 교체되었습니다. 워크로드에서 sidecar.maistra.io 주석을 사용 중인 경우 대신 동등한 sidecar.istio.io를 사용하도록 해당 워크로드를 수정해야 합니다.
- sidecar.maistra.io/proxyMemoryLimit가 sidecar.istio.io/proxyMemoryLimit로 교체됨
- sidecar.istio.io/discoveryAddress는 더 이상 지원되지 않습니다. 또한 기본 검색 주소는 pilot.<control_plane_namespace>.svc:15010(또는 mtls가 활성화된 경우 포트 15011)에서 istiod-<smcp_name>.<control_plane_namespace>.svc:15012로 이동했습니다.
- 상태 포트는 더 이상 구성할 수 없으며 15021로 하드 코딩됩니다. * 사용자 정의 상태 포트를 정의하는 경우 (예: status.sidecar.istio.io/port) 워크로드를 v2.0 Service Mesh 컨트롤 플레인으로 이동하기 전에 재정의를 제거해야 합니다. 상태 포트를 0 으로 설정하여 준비 상태 점검을 비활성화할 수 있습니다.
- Kubernetes 시크릿 리소스는 더 이상 사이드카에 대한 클라이언트 인증서를 배포하는 데 사용되지 않습니다. 인증서는 이제 Istiod의 SDS 서비스를 통해 배포됩니다. 마운트된 보안을 사용하는 경우 v2.0 Service Mesh Control Plane의 워크로드에 더 오래 사용할 수 있습니다.

1.11.4.6.2.3. 동작 변경

Red Hat OpenShift Service Mesh 2.0의 일부 기능은 이전 버전과 다르게 작동합니다.

- 게이트웨이의 준비 상태 포트는 **15020**에서 **15021**로 이동했습니다.

- 대상 호스트 가시성에는 **VirtualService** 및 **ServiceEntry** 리소스가 포함됩니다. 사이드카 리소스를 통해 적용된 모든 제한을 포함합니다.

- 자동 상호 **TLS**는 기본적으로 활성화되어 있습니다. 프록시 간 통신은 글로벌 **PeerAuthentication** 정책에 관계없이 **mTLS**를 사용하도록 자동 구성됩니다.

- 보안 연결은 **spec.security.controlPlane.mtls** 설정에 관계없이 프록시가 **Service Mesh Control Plane**과 통신할 때 항상 사용됩니다. **spec.security.controlPlane.mtls** 설정은 **Mixer Telemetry** 또는 정책에 대한 연결을 구성할 때만 사용됩니다.

1.11.4.6.2.4. 지원되지 않는 리소스에 대한 마이그레이션 세부 정보

정책(authentication.istio.io/v1alpha1)

v2.0 Service Mesh Control Planes, PeerAuthentication 및 **RequestAuthentication**과 함께 사용하려면 정책 리소스를 새 리소스 유형으로 마이그레이션해야 합니다. 정책 리소스의 특정 구성에 따라 동일한 효과를 달성하기 위해 여러 리소스를 구성해야 할 수 있습니다.

상호 **TLS**

상호 **TLS** 적용은 **security.istio.io/v1beta1 PeerAuthentication** 리소스를 사용하여 수행됩니다. 레거시 **spec.peers.mtls.mode** 필드는 새로운 리소스의 **spec.mtls.mode** 필드에 직접 매핑됩니다. 선택 기준이 **spec.targets[x].name**의 서비스 이름 지정에서 **spec.selector.matchLabels**의 레이블 선택기로 변경되었습니다. **PeerAuthentication**에서 레이블은 대상 목록에 이름이 지정된 서비스의 선택기와 일치해야 합니다. 모든 포트별 설정은 **spec.portLevelMtls**에 매핑되어야 합니다.

인증

spec.origins에 지정된 추가 인증 방법은 **security.istio.io/v1beta1 RequestAuthentication** 리소스에 매핑되어야 합니다. **spec.selector.matchLabels**는 **PeerAuthentication**의 동일한 필드와 유사하게 구성되어야 합니다. **spec.origins.jwt** 항목의 **JWT** 주체와 관련된 구성은 **spec.rules** 항목의 유사한 필드에 매핑됩니다.

- 정책에 지정된 **spec.origins[x].jwt.triggerRules**는 하나 이상의 **security.istio.io/v1beta1 AuthorizationPolicy** 리소스에 매핑되어야 합니다. **spec.selector.labels**는 **RequestAuthentication**의 동일한 필드와 유사하게 구성되어야 합니다.

- **spec.origins[x].jwt.triggerRules.excludedPaths.excludedPaths** 는 **spec.action**이 **ALLOW**로 설정된 **AuthorizationPolicy**에 매핑되고, **spec.rules[x].to.operation.path** 항목이 제

외된 경로와 일치해야 합니다.

- spec.origins[x].jwt.triggerRules.includedPaths는 spec.action이 ALLOW로 설정된 별도의 AuthorizationPolicy에 매핑되고, spec.rules[x].to.operation.path 항목이 제외된 경로와 일치하며, spec.rules.[x].from.source.requestPrincipals 항목이 정책 리소스의 specified spec.origins[x].jwt.issuer와 일치해야 합니다

ServiceMeshPolicy(maistra.io/v1)

ServiceMeshPolicy는 v1 리소스의 spec.istio.global.mtls.enabled 또는 v2 리소스 설정의 spec.security.dataPlane.mtls 를 통해 Service Mesh Control Plane에 대해 자동으로 구성되었습니다. v2 컨트롤 플레인의 경우 설치 중에 기능적으로 동일한 PeerAuthentication 리소스가 생성됩니다. 이 기능은 Red Hat OpenShift Service Mesh 버전 2.0에서 더 이상 사용되지 않습니다.

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

이러한 리소스는 security.istio.io/v1beta1 AuthorizationPolicy 리소스로 교체되었습니다.

RbacConfig 동작을 모방하려면 RbacConfig에 지정된 spec.mode에 따라 설정이 달라지는 기본 AuthorizationPolicy를 작성해야 합니다.

- spec.mode가 OFF로 설정되면 AuthorizationPolicy가 요청에 적용되지 않는 한 기본 정책은 ALLOW이므로 리소스가 필요하지 않습니다.

- spec.mode가 ON으로 설정된 경우 spec: {}를 설정합니다. 메시의 모든 서비스에 대해 AuthorizationPolicy 정책을 생성해야 합니다.

- spec.mode가 ON_WITH_INCLUSION으로 설정되며, 포함된 각각의 네임스페이스에 spec: {}을 사용하여 AuthorizationPolicy를 생성해야 합니다. 개별 서비스 포함은 AuthorizationPolicy에서 지원되지 않습니다. 그러나 서비스의 워크로드에 적용되는 AuthorizationPolicy가 생성되면 명시적으로 허용되지 않는 다른 모든 요청이 거부됩니다.

- spec.mode가 ON_WITH_EXCLUSION으로 설정된 경우 AuthorizationPolicy에서 지원되지 않습니다. 글로벌 DENY 정책을 생성할 수 있지만, 네임스페이스 또는 워크로드에 적용할 수 있는 허용된 정책이 없기 때문에 메시의 모든 워크로드에 대해 AuthorizationPolicy를 생성해야 합니다.

AuthorizationPolicy에는 ServiceRoleBinding이 제공하는 기능과 유사한 구성이 적용되는 선택기와 ServiceRole이 제공하는 기능과 유사하며 적용되어야 하는 규칙에 대한 구성이 모두 포함됩니다.

ServiceMeshRbacConfig (maistra.io/v1)

이 리소스는 **Service Mesh Control Plane**의 네임스페이스에서 빈 **spec.selector**가 있는 **security.istio.io/v1beta1 AuthorizationPolicy** 리소스를 사용하여 교체됩니다. 이 정책은 메시의 모든 워크로드에 적용되는 기본 권한 부여 정책이 됩니다. 특정 마이그레이션 세부 사항은 위의 **RbacConfig**를 참조하십시오.

1.11.4.6.2.5. Mixer 플러그인

Mixer 구성 요소는 버전 **2.0**에서 기본적으로 비활성화되어 있습니다. 워크로드에 **Mixer** 플러그인을 사용하는 경우 **Mixer** 구성 요소를 포함하도록 버전 **2.0 ServiceMeshControlPlane**을 구성해야 합니다.

Mixer 정책 구성 요소를 활성화하려면 **ServiceMeshControlPlane**에 다음 스니펫을 추가합니다.

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

Mixer telemetry 구성 요소를 활성화하려면 **ServiceMeshControlPlane**에 다음 스니펫을 추가합니다.

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

또한 레거시 **mixer** 플러그인은 **WASM**으로 마이그레이션하고 새로운 **ServiceMeshExtension(maistra.io/v1alpha1)** 사용자 정의 리소스를 사용하여 통합할 수 있습니다.

업스트림 **Istio** 배포에 포함된 내장 **WASM** 필터는 **Red Hat OpenShift Service Mesh 2.0**에서 사용할 수 없습니다.

1.11.4.6.2.6. 상호 TLS 변경

워크로드별 **PeerAuthentication** 정책과 함께 **mTLS**를 사용할 때 워크로드 정책이 네임스페이스/글로벌 정책과 다를 경우 트래픽을 허용하려면 상응하는 **DestinationRule**이 필요합니다.

자동 **mTLS**는 기본적으로 활성화되어 있지만 **ServiceMeshControlPlane** 리소스에서 **spec.security.dataPlane.automtls**를 **false**로 설정하여 비활성화할 수 있습니다. 자동 **mTLS**를 비활성

화할 때 서비스 간 적절한 통신을 위해 **DestinationRules**가 필요할 수 있습니다. 예를 들어, 하나의 네임스페이스에 대해 **PeerAuthentication**을 **STRICT**으로 설정하면 **DestinationRule**이 네임스페이스의 서비스에 **TLS** 모드를 구성하지 않는 한 다른 네임스페이스의 서비스에 액세스하지 못할 수 있습니다.

mTLS에 대한 자세한 내용은 **mTLS(mutual Transport Layer Security)**활성화를 참조하십시오.

1.11.4.6.2.6.1. 기타 mTLS 예

mTLS 비활성화: 정보 샘플 애플리케이션의 **productpage** 서비스의 경우 **Red Hat OpenShift Service Mesh v1.1**에 대해 다음과 같은 방식으로 정책 리소스가 구성되었습니다.

정책 리소스 예

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

mTLS 비활성화: 정보 샘플 애플리케이션의 **productpage** 서비스의 경우 다음 예제를 사용하여 **Red Hat OpenShift Service Mesh v2.0**에 대한 **PeerAuthentication** 리소스를 구성합니다.

PeerAuthentication 리소스 예

```
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
```

mTLS 활성화: 정보 샘플 애플리케이션에서 **productpage** 서비스에 대한 **JWT** 인증의 경우, **Red Hat OpenShift Service Mesh v1.1**에 대해 다음과 같은 방식으로 정책 리소스가 구성되었습니다.

정책 리소스 예

```
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
```

mTLS 활성화: 정보 샘플 애플리케이션에서 **productpage** 서비스에 대한 **JWT** 인증의 경우 다음 예제를 사용하여 **Red Hat OpenShift Service Mesh v2.0**에 대한 **PeerAuthentication** 리소스를 구성합니다.

PeerAuthentication 리소스 예

```
#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
```

1.11.4.6.3. 설정 레시피

이러한 구성 레시피를 사용하여 다음 항목을 구성할 수 있습니다.

1.11.4.6.3.1. 데이터 플레인의 상호 TLS

데이터 플레인 통신에 대한 상호 TLS는 ServiceMeshControlPlane 리소스의 spec.security.dataPlane.mtls를 통해 구성되며, 기본적으로 false입니다.

1.11.4.6.3.2. 사용자 정의 서명 키

Istiod는 서비스 프록시에서 사용하는 클라이언트 인증서 및 개인 키를 관리합니다. 기본적으로 Istiod는 서명에 자체 서명된 인증서를 사용하지만 사용자 정의 인증서와 개인 키를 구성할 수 있습니다. 서명 키를 구성하는 방법에 대한 자세한 내용은 외부 인증 기관 키 및 인증서 추가를 참조하십시오.

1.11.4.6.3.3. 추적

추적 기능은 spec.tracing에서 구성됩니다. 현재 지원되는 유일한 추적기 유형은 Jaeger입니다. 샘플링은 0.01% 증분을 나타내는 스케일링된 정수입니다(예: 1은 0.01%, 10000은 100%). 추적 구현 및 샘플링 비율을 지정할 수 있습니다.

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

Jaeger는 ServiceMeshControlPlane 리소스의 애드온 섹션에서 구성됩니다.

```
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 설치는 install 필드로 사용자 지정할 수 있습니다. 리소스 제한과 같은 컨테이너 구성은 spec.runtime.components.jaeger 관련 필드에 구성됩니다. spec.addons.jaeger.name 값과 일치하는 Jaeger 리소스가 있으면 기존 설치를 사용하도록 서비스 메시 컨트롤 플레인이 구성됩니다. 기존 Jaeger 리소스를 사용하여 Jaeger 설치를 완전히 사용자 지정할 수 있습니다.

1.11.4.6.3.4. 시각화

Kiali 및 Grafana는 ServiceMeshControlPlane 리소스의 애드온 섹션에서 구성됩니다.

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

Grafana 및 Kiali 설치는 각각의 install 필드를 통해 사용자 지정할 수 있습니다. 리소스 제한과 같은 컨테이너 사용자 정의는 spec.runtime.components.kiali 및 spec.runtime.components.grafana에서 구성됩니다. name 값과 일치하는 기존 Kiali 리소스가 있는 경우 Service Mesh Control Plane은 컨트롤 플레인과 함께 사용할 Kiali 리소스를 구성합니다. Kiali 리소스의 일부 필드(예: accessible_namespaces 목록과 Grafana, Prometheus, 추적에 대한 끝점)는 재정의됩니다. 기존 리소스를 사용하여 Kiali 설치를 완전히 사용자 지정할 수 있습니다.

1.11.4.6.3.5. 리소스 사용률 및 스케줄링

리소스는 spec.runtime.<component>에서 구성됩니다. 다음과 같은 구성 요소 이름이 지원됩니다.

구성 요소	설명	지원되는 버전
보안	Citadel 컨테이너	v1.0/1.1
galley	Galley 컨테이너	v1.0/1.1
pilot	Pilot/Istiod 컨테이너	v1.0/1.1/2.0
mixer	Istio-telemetry 및 istio-policy 컨테이너	v1.0/1.1
mixer.policy	Istio-policy 컨테이너	v2.0
mixer.telemetry	Istio-telemetry 컨테이너	v2.0
global.oauthproxy	다양한 애드온과 함께 사용되는 oauth-proxy 컨테이너	v1.0/1.1/2.0
sidecarInjectorWebhook	사이드카 인젝터 webhook 컨테이너	v1.0/1.1
tracing.jaeger	일반 Jaeger 컨테이너 - 일부 설정은 적용할 수 없습니다. Service Mesh Control Plane 구성에서 기존 Jaeger 리소스를 지정하면 Jaeger 설치에 대한 완전한 사용자 정의가 지원됩니다.	v1.0/1.1/2.0
tracing.jaeger.agent	Jaeger 에이전트와 관련된 설정	v1.0/1.1/2.0
tracing.jaeger.allInOne	Jaeger allInOne과 관련된 설정	v1.0/1.1/2.0
tracing.jaeger.collector	Jaeger 수집기와 관련된 설정	v1.0/1.1/2.0
tracing.jaeger.elasticsearch	Jaeger elasticsearch 배포와 관련된 설정	v1.0/1.1/2.0
tracing.jaeger.query	Jaeger 쿼리와 관련된 설정	v1.0/1.1/2.0
prometheus	prometheus 컨테이너	v1.0/1.1/2.0

구성 요소	설명	지원되는 버전
kiali	Kiali 컨테이너 - Service Mesh Control Plane 구성에 기존 Kiali 리소스를 지정하면 Kiali 설치에 대한 완전한 사용자 정의가 지원됩니다.	v1.0/1.1/2.0
grafana	Grafana 컨테이너	v1.0/1.1/2.0
3scale	3scale 컨테이너	v1.0/1.1/2.0
wasmExtensions.cacher	WASM 확장 cacher 컨테이너	v2.0 - 기술 프리뷰

일부 구성 요소는 리소스 제한 및 스케줄링을 지원합니다. 자세한 내용은 성능 및 확장성을 참조하십시오.

1.11.4.6.4. 애플리케이션 및 워크로드를 마이그레이션하기 위한 다음 단계

애플리케이션 워크로드를 새 메시로 이동하고 이전 인스턴스를 제거하여 업그레이드를 완료합니다.

1.11.5. 데이터 플레인 업그레이드

컨트롤 플레인을 업그레이드한 후에도 데이터 플레인이 계속 작동합니다. 그러나 **Envoy** 프록시 및 프록시 구성에 대한 변경 사항을 적용하려면 애플리케이션 포드 및 워크로드를 다시 시작해야 합니다.

1.11.5.1. 애플리케이션 및 워크로드 업데이트

마이그레이션을 완료하려면 메시의 모든 애플리케이션 **Pod**를 재시작하여 **Envoy** 사이드카 프록시 및 해당 구성을 업그레이드합니다.

배포 롤링 업데이트를 수행하려면 다음 명령을 사용합니다.

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

메시를 구성하는 모든 애플리케이션에 대해 롤링 업데이트를 수행해야 합니다.

1.12. 사용자 및 프로파일 관리

1.12.1. Red Hat OpenShift Service Mesh 멤버 생성

ServiceMeshMember 리소스는 **Red Hat OpenShift Service Mesh** 관리자가 서비스 메시에 프로젝트를 추가할 수 있는 권한을 위임할 수 있는 방법을 제공합니다. 해당 사용자가 서비스 메시 프로젝트 또는 구성원 목록에 직접 액세스할 수 없는 경우에도 마찬가지입니다. 프로젝트 관리자가 프로젝트에서 **ServiceMeshMember** 리소스를 생성할 수 있는 권한을 자동으로 부여하는 동안 서비스 메시 관리자가 서비스 메시에 대한 액세스 권한을 명시적으로 부여할 때까지 **ServiceMeshControlPlane**를 가리킬 수 없습니다. 관리자는 **mesh-user** 사용자 역할을 부여하여 메시에 액세스할 수 있는 권한을 사용자에게 부여할 수 있습니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 프로젝트의 이름입니다.

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

관리자는 **Service Mesh Control Plane** 프로젝트에서 **meshuser** 역할 바인딩을 수정하여 액세스 권한이 부여된 사용자 및 그룹을 지정할 수 있습니다. **ServiceMeshMember** 는 이를 참조하는 **Service Mesh Control Plane** 프로젝트 내의 **ServiceMeshMemberRoll** 에 프로젝트를 추가합니다.

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

관리자가 **ServiceMeshControlPlane** 리소스를 생성한 후 **mesh-users** 역할 바인딩이 자동으로 생성됩니다. 관리자는 다음 명령을 사용하여 사용자에게 역할을 추가할 수 있습니다.

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

관리자가 **ServiceMeshControlPlane** 리소스를 생성하기 전에 **mesh-user** 역할 바인딩을 생성할 수도 있습니다. 예를 들어 관리자는 **ServiceMeshControlPlane** 리소스와 동일한 **oc apply** 작업으로 이를 생성할 수 있습니다.

이 예제에서는 **alice**에 대한 역할 바인딩이 추가되었습니다.

```
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 Control Plane 프로필 생성

ServiceMeshControlPlane 프로필을 사용하여 재사용 가능한 구성을 생성할 수 있습니다. 개별 사용자는 생성한 프로필을 자체 구성으로 확장할 수 있습니다. 프로필은 다른 프로필의 구성 정보를 상속할 수도 있습니다. 예를 들어, 회계 팀에 대한 계정 컨트롤 플레인과 마케팅 팀에 대한 마케팅 컨트롤 플레인을 생성할 수 있습니다. 개발 템플릿과 프로덕션 템플릿을 생성하는 경우 마케팅 팀과 회계 팀의 구성원은 팀별 사용자 지정을 통해 개발 및 프로덕션 프로필을 확장할 수 있습니다.

ServiceMeshControlPlane 과 동일한 구문을 따르는 Service Mesh Control Plane 프로필을 구성하면, 사용자는 계층적으로 설정을 상속합니다. Operator는 Red Hat OpenShift Service Mesh의 기본 설정이 포함된 default 프로필과 함께 제공됩니다.

1.12.2.1. ConfigMap 생성

사용자 지정 프로필을 추가하려면 openshift-operators 프로젝트에서 smcp-templates 라는 ConfigMap 을 생성해야 합니다. Operator 컨테이너는 ConfigMap 을 자동으로 마운트합니다.

사전 요구 사항

- Service Mesh Operator 설치 및 검증.
- cluster-admin 역할이 있는 계정. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.
- Operator 배포 위치.
- OpenShift CLI(oc)에 액세스합니다.

절차

1. OpenShift Container Platform CLI에 cluster-admin로 로그인합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.

2. CLI에서 이 명령을 실행하여 openshift-operators 프로젝트에서 smcp-templates라는 ConfigMap을 생성하고 <profiles-directory>를 로컬 디스크의 ServiceMeshControlPlane 파일의 위치로 교체합니다.

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

3. ServiceMeshControlPlane에서 profiles 매개변수를 사용하여 하나 이상의 템플릿을 지정할 수 있습니다.

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

1.12.2.2. 올바른 네트워크 정책 설정

서비스 메시는 서비스 메시 컨트롤 플레인과 멤버 네임스페이스에서 네트워크 정책을 생성하여 트래픽을 허용합니다. 배포하기 전에 다음 조건을 고려하여 OpenShift Container Platform 경로를 통해 이전에 노출된 서비스 메시의 서비스를 확인하십시오.

- Istio가 제대로 작동하려면 서비스 메시로 들어오는 트래픽이 항상 Ingress-gateway를 통과해야 합니다.
- 서비스 메시에 없는 별도의 네임스페이스에서 서비스 메시 외부에 서비스를 배포합니다.
- 서비스 메시 등록 네임스페이스에 배포해야 하는 메시 외 서비스는 해당 배포 maistra.io/expose-route: "true"에 레이블을 지정하여 OpenShift Container Platform 경로가 여전히 작동하도록 해야 합니다.

1.13. 보안

서비스 메시 애플리케이션이 복잡한 마이크로 서비스를 사용하여 구성된 경우 Red Hat OpenShift Service Mesh를 사용하여 해당 서비스 간 통신 보안을 사용자 지정할 수 있습니다. 서비스 메시의 트래픽 관리 기능과 함께 OpenShift Container Platform의 인프라는 애플리케이션의 복잡성을 관리하고 마이크로서비스를 보호하는데 도움이 됩니다.

시작하기 전

프로젝트가 있는 경우 ServiceMeshMemberRoll 리소스에 프로젝트를 추가합니다.

프로젝트가 없는 경우 Bookinfo 샘플 애플리케이션을 설치하고 ServiceMeshMemberRoll 리소스에 추가합니다. 샘플 애플리케이션은 보안 개념을 설명하는 데 도움이 됩니다.

1.13.1. mTLS(mutual Transport Layer Security) 정보

mTLS(mutual Transport Layer Security)은 두 당사자가 서로 인증할 수 있도록 하는 프로토콜입니다. 일부 프로토콜(IKE, SSH)에서는 기본 인증 모드이며, 다른 프로토콜(TLS)에서는 선택적입니다. 애플리케이션 또는 서비스 코드를 변경하지 않고 mTLS를 사용할 수 있습니다. TLS는 서비스 메시 인프라와 두 사이드카 프록시 사이에서 전적으로 처리됩니다.

기본적으로 Red Hat OpenShift Service Mesh의 mTLS가 활성화되고 허용 모드로 설정됩니다. 여기서 서비스 메시의 사이드카는 일반 텍스트 트래픽과 mTLS를 사용하여 암호화된 연결을 모두 허용합니다. 메시의 서비스가 메시 외부 서비스와 통신하는 경우 엄격한 mTLS가 해당 서비스 간의 통신을 중단할 수 있습니다. 워크로드를 서비스 메시로 마이그레이션하는 동안 허용 모드를 사용합니다. 그러면 메시, 네임스페이스 또는 애플리케이션 전반에서 엄격한 mTLS를 활성화할 수 있습니다.

서비스 메시 컨트롤 플레인 수준에서 메시 전체에 mTLS를 활성화하면 애플리케이션 및 워크로드를 다시 작성하지 않고도 서비스 메시의 모든 트래픽을 보호할 수 있습니다. ServiceMeshControlPlane 리소스의 데이터 플레인 수준에서 메시의 네임스페이스를 보호할 수 있습니다. 트래픽 암호화 연결을 사용자 지정하려면 PeerAuthentication 및 DestinationRule 리소스를 사용하여 애플리케이션 수준에서 네임스페이스를 구성합니다.

1.13.1.1. 서비스 메시에서 엄격한 mTLS 활성화

워크로드가 외부 서비스와 통신하지 않으면 통신 중단 없이 메시 전체에서 mTLS를 빠르게 활성화할 수 있습니다. ServiceMeshControlPlane 리소스에서 spec.security.dataPlane.mtls를 true로 설정하여 활성화할 수 있습니다. Operator는 필요한 리소스를 생성합니다.

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  version: v2.4
  security:
    dataPlane:
      mtls: true
```

또한 OpenShift Container Platform 웹 콘솔을 사용하여 mTLS를 활성화할 수 있습니다.

절차

1. 웹 콘솔에 로그인합니다.

2. 프로젝트 메뉴를 클릭하고 **Service Mesh Control Plane**을 설치한 프로젝트(예: **istio-system**)를 선택합니다.

3. **Operators** → 설치된 **Operators**를 클릭합니다.

4. 제공된 **API**에서 **Service Mesh Control Plane**을 클릭합니다.

5. **ServiceMeshControlPlane** 리소스의 이름(예: **production**)을 클릭합니다.

6. 세부 정보 페이지에서 데이터 플레인 보안의 보안 섹션에서 토글을 클릭합니다.

1.13.1.1.1. 특정 서비스의 수신 연결에 대해 사이드카 구성

정책을 생성하여 개별 서비스 또는 네임스페이스에 대해 **mTLS**를 구성할 수도 있습니다.

절차

1. 다음 예제를 사용하여 **YAML** 파일을 생성합니다.

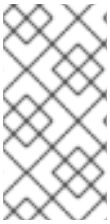
PeerAuthentication 정책 예 **policy.yaml**

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

a. <namespace>를 서비스가 있는 네임스페이스로 바꿉니다.

2. 다음 명령을 실행하여 서비스가 있는 네임스페이스에 리소스를 생성합니다. 방금 생성한 정책 리소스의 namespace 필드와 일치해야 합니다.

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



참고

자동 mTLS를 사용하지 않고 PeerAuthentication을 STRICT으로 설정하는 경우 서비스에 대한 DestinationRule 리소스를 생성해야 합니다.

1.13.1.1.2. 발신 연결에 대한 사이드카 구성

메시에서 다른 서비스로 요청을 보낼 때 mTLS를 사용하도록 서비스 메시를 구성하는 대상 규칙을 생성합니다.

프로세스

1. 다음 예제를 사용하여 YAML 파일을 생성합니다.

DestinationRule 예제 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>를 서비스가 있는 네임스페이스로 바꿉니다.

2. 다음 명령을 실행하여 서비스가 있는 네임스페이스에 리소스를 생성합니다. 방금 생성한 **DestinationRule** 리소스의 **namespace** 필드와 일치해야 합니다.

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

1.13.1.1.3. 최소 및 최대 프로토콜 버전 설정

사용자 환경에 서비스 메시의 암호화된 트래픽에 대한 특정 요구 사항이 있는 경우 **ServiceMeshControlPlane** 리소스에 **spec.security.controlPlane.tls.minProtocolVersion** 또는 **spec.security.controlPlane.tls.maxProtocolVersion**을 설정하여 허용되는 암호화 기능을 제어할 수 있습니다. **Service Mesh Control Plane** 리소스에 구성된 해당 값은 **TLS**를 통해 안전하게 통신할 때 메시 구성 요소에서 사용하는 최소 및 최대 **TLS** 버전을 정의합니다.

기본값은 **TLS_AUTO**이며 **TLS** 버전을 지정하지 않습니다.

표 **1.5.** 유효한 값

값	설명
TLS_AUTO	default
TLSv1_0	TLS 버전 1.0
TLSv1_1	TLS 버전 1.1
TLSv1_2	TLS 버전 1.2
TLSv1_3	TLS 버전 1.3

프로세스

1. 웹 콘솔에 로그인합니다.
2. 프로젝트 메뉴를 클릭하고 **Service Mesh Control Plane**을 설치한 프로젝트(예: **istio-system**)를 선택합니다.
3. **Operators** → 설치된 **Operators**를 클릭합니다.
4. 제공된 **API**에서 **Service Mesh Control Plane**을 클릭합니다.

5. **ServiceMeshControlPlane** 리소스의 이름(예: **production**)을 클릭합니다.

6. **YAML** 탭을 클릭합니다.

7. **YAML** 편집기에 다음 코드 조각을 삽입합니다. **minProtocolVersion**의 값을 **TLS** 버전 값으로 바꿉니다. 이 예에서 최소 **TLS** 버전은 **TLSv1_2**로 설정됩니다.

ServiceMeshControlPlane 스니펫

```
kind: ServiceMeshControlPlane
spec:
  security:
    controlPlane:
      tls:
        minProtocolVersion: TLSv1_2
```

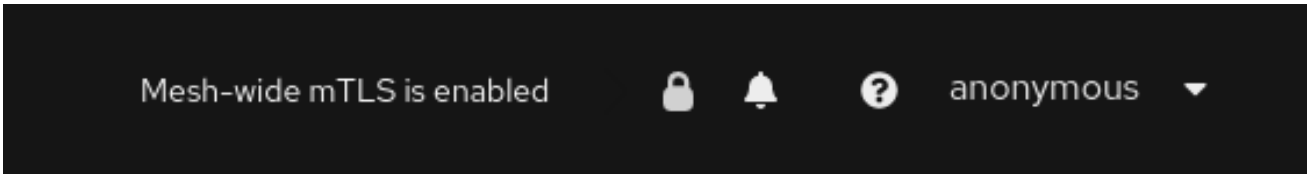
8. 저장을 클릭합니다.

9. 새로 고침을 클릭하여 변경 사항이 올바르게 업데이트되었는지 확인합니다.

1.13.1.2. Kiali를 사용하여 암호화 검증

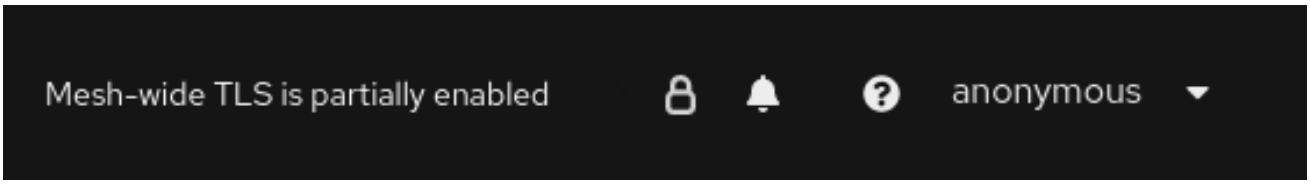
Kiali 콘솔은 애플리케이션, 서비스 및 워크로드에 **mTLS** 암호화가 활성화되어 있는지 여부를 확인하는 여러 가지 방법을 제공합니다.

그림 **1.5.** 마스트 헤드 아이콘 메시 전체 **mTLS** 활성화



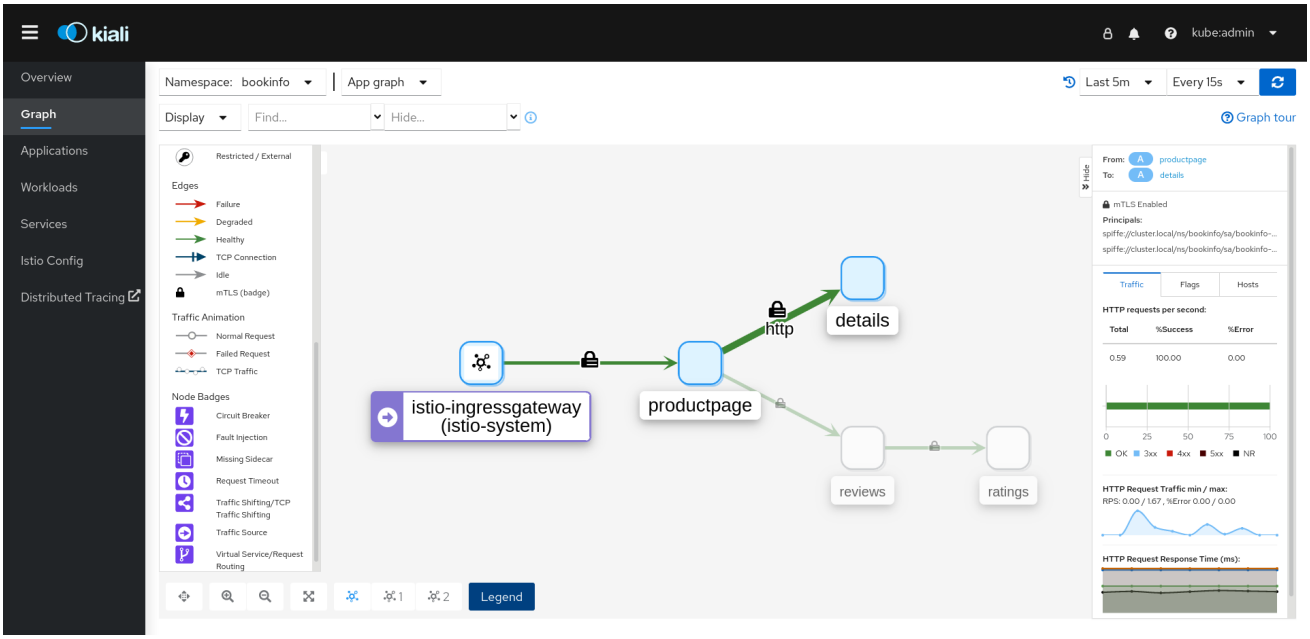
마스트 헤드 오른쪽에 있는 **Kiali**는 메시가 전체 서비스 메시에 대해 **mTLS**를 엄격하게 활성화한 경우 잠금 아이콘을 표시합니다. 이는 메시의 모든 통신이 **mTLS**를 사용한다는 것을 의미합니다.

그림 1.6. masthead 아이콘 메시 전체 mTLS가 부분적으로 활성화됨



메시가 PERMISSIVE 모드로 구성되거나 메시 전체 mTLS 구성에 오류가 있는 경우 Kiali는 hollow 잠금 아이콘을 표시합니다.

그림 1.7. 보안 배지



그래프 페이지에는 mTLS가 활성화되었음을 나타내기 위해 그래프 에지에 보안 배지를 표시하는 옵션이 있습니다. 그래프에서 보안 배지를 활성화하려면 표시 메뉴에서 Show badges 아래에서 보안 확인란을 선택합니다. 에지에 잠금 아이콘이 표시되면 mTLS가 활성화된 요청이 하나 이상 있음을 의미합니다. mTLS 및 비mTLS 요청이 모두 있는 경우 side-panel은 mTLS를 사용하는 요청의 백분율을 표시합니다.

애플리케이션 세부 정보 개요 페이지에는 mTLS가 활성화된 하나 이상의 요청이 있는 그래프 에지에 보안 아이콘이 표시됩니다.

워크로드 세부 정보 개요 페이지에는 mTLS가 활성화된 요청이 하나 이상 있는 그래프 에지에 보안 아이콘이 표시됩니다.

서비스 세부 정보 개요 페이지에는 mTLS가 활성화된 요청이 하나 이상 있는 그래프 에지에 보안 아이콘이 표시됩니다. 또한 Kiali는 mTLS에 대해 구성된 포트 옆에 네트워크 섹션에 잠금 아이콘을 표시합니다.

1.13.2. 역할 기반 액세스 제어(RBAC) 구성

RBAC(역할 기반 액세스 제어) 오브젝트에 따라 사용자 또는 서비스가 프로젝트 내에서 지정된 작업을 수행할 수 있는지가 결정됩니다. 메시의 워크로드에 대해 메시, 네임스페이스, 워크로드 전체 액세스 제어를 정의할 수 있습니다.

RBAC를 구성하려면 액세스를 구성하는 네임스페이스에 AuthorizationPolicy 리소스를 생성합니다. 메시 전체 액세스를 구성하는 경우 Service Mesh Control Plane을 설치한 프로젝트를 사용합니다(예: istio-system).

예를 들어 RBAC를 사용하면 다음과 같은 정책을 생성할 수 있습니다.

- 프로젝트 내 통신을 구성합니다.
- 기본 네임스페이스의 모든 워크로드에 대한 전체 액세스를 허용하거나 거부합니다.
- 수신 게이트웨이 액세스를 허용 또는 거부합니다.
- 액세스 하려면 토큰이 필요합니다.

권한 부여 정책에는 선택기, 작업 및 규칙 목록이 포함됩니다.

- selector 필드는 정책의 대상을 지정합니다.
- action 필드는 요청을 허용하거나 거부할지 여부를 지정합니다.
- rules 필드는 작업을 트리거할 시기를 지정합니다.
  - from 필드는 요청 원본에 대한 제약 조건을 지정합니다.
  - to 필드는 요청 대상 및 매개변수에 대한 제약 조건을 지정합니다.

- when 필드는 규칙을 적용하기 위한 추가 조건을 지정합니다.

프로세스

1. AuthorizationPolicy 리소스를 생성합니다. 다음 예제는 IP 주소가 수신 게이트웨이에 액세스하는 것을 거부하도록 ingress-policy AuthorizationPolicy를 업데이트하는 리소스를 보여줍니다.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ingress-policy
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: DENY
  rules:
  - from:
    - source:
      ipBlocks: ["1.2.3.4"]
```

2. 리소스를 작성한 후 다음 명령어를 실행하여 네임스페이스에 리소스를 만듭니다. 네임스페이스는 AuthorizationPolicy 리소스의 metadata.namespace 필드와 일치해야 합니다.

```
$ oc create -n istio-system -f <filename>
```

다음 단계

다른 일반적인 구성에 대해서는 다음 예제를 고려하십시오.

1.13.2.1. 프로젝트 내 통신 구성

AuthorizationPolicy 를 사용하여 메시의 메시 또는 서비스와 통신하는 트래픽을 허용하거나 거부하도록 Service Mesh Control Plane을 구성할 수 있습니다.

1.13.2.1.1. 네임스페이스 외부 서비스에 대한 액세스 제한

다음 AuthorizationPolicy 리소스 예제를 사용하여 info 네임스페이스에 없는 모든 소스의 요청을 거부할 수 있습니다.

```
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. 권한 부여 모두 허용 및 권한 부여 모두 거부(기본) 정책 만들기

다음 예제에서는 **info** 네임스페이스의 모든 워크로드에 대한 전체 액세스 권한을 허용하는 모든 권한 부여 정책을 보여줍니다.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-all
  namespace: info
spec:
  action: ALLOW
  rules:
  - {}
```

다음 예제에서는 **info** 네임스페이스의 모든 워크로드에 대한 액세스를 거부하는 정책을 보여줍니다.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: info
spec:
  {}
```

1.13.2.2. 수신 게이트웨이에 대한 액세스 허용 또는 거부

IP 주소를 기반으로 허용 또는 거부 목록을 추가하도록 권한 부여 정책을 설정할 수 있습니다.

```
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 웹 토큰으로 액세스 제한

JSON 웹 토큰(JWT)으로 메시에 액세스하는 항목을 제한할 수 있습니다. 인증 후 사용자 또는 서비스는 해당 토큰과 연결된 경로, 서비스에 액세스할 수 있습니다.

워크로드에서 지원하는 인증 방법을 정의하는 RequestAuthentication 리소스를 생성합니다. 다음 예제에서는 http://localhost:8080/auth/realms/master에서 발행한 JWT를 수락합니다.

```
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"
```

그런 다음, 동일한 네임스페이스에 AuthorizationPolicy 리소스를 생성하여, 사용자가 생성한 RequestAuthentication 리소스와 함께 작업할 수 있습니다. 다음 예제에서는 httpbin 워크로드에 요청을 보낼 때 Authorization 헤더에 JWT가 있어야 합니다.

```
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. 암호화 제품군 및 ECDH 곡선 구성

암호화 제품군 및 ECDH(Elliptic-curve Diffie–Hellman) 곡선은 서비스 메시를 보호하는 데 도움이 될 수 있습니다. ServiceMeshControlPlane 리소스에서 spec.security.controlplane.tls.cipherSuites 및 ECDH 곡선을 사용하여 쉼표로 구분된 암호화 제품군 목록을 정의할 수 있습니다. 이러한 속성 중 하나가 비어 있으면 기본값이 사용됩니다.

서비스 메시에서 TLS 1.2 또는 이전 버전을 사용하는 경우 cipherSuites 설정이 적용됩니다. TLS 1.3을 사용할 때는 효과가 없습니다.

우선순위에 따라 암호화 제품군을 쉼표로 구분된 목록으로 설정합니다. 예를 들어 ecdhCurves: CurveP256, CurveP384는 CurveP256을 CurveP384보다 높은 우선순위로 설정합니다.

참고

암호화 제품군을 구성할 때 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 또는 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256을 포함해야 합니다. HTTP/2 지원에는 이러한 암호화 제품군 중 하나 이상이 필요합니다.

지원되는 암호화 제품군은 다음과 같습니다.

- 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**

지원되는 ECDH 곡선은 다음과 같습니다.

- CurveP256
- CurveP384
- CurveP521
- X25519

1.13.4. 외부 인증 기관 키 및 인증서 추가

기본적으로 Red Hat OpenShift Service Mesh는 자체 서명된 루트 인증서와 키를 생성하고 이를 사용하여 워크로드 인증서에 서명합니다. 사용자 정의 인증서 및 키를 사용하여 사용자 정의 루트 인증서로 워크로드 인증서에 서명할 수도 있습니다. 이 작업은 인증서와 키를 서비스 메시에 연결하는 예제를 보여줍니다.

사전 요구 사항

- 인증서를 구성하려면 상호 TLS가 활성화된 Red Hat OpenShift Service Mesh를 설치합니다.
- 이 예에서는 Maistra 리포지토리 의 인증서를 사용합니다. 프로덕션의 경우 인증 기관의 자체 인증서를 사용합니다.
- 이러한 지침으로 결과를 확인하려면 Bookinfo 샘플 애플리케이션을 배포합니다.
- 인증서를 확인하려면 OpenSSL이 필요합니다.

1.13.4.1. 기존 인증서 및 키 추가

기존 서명(CA) 인증서 및 키를 사용하려면 CA 인증서, 키, 루트 인증서가 포함된 신뢰 파일 체인을 생성해야 합니다. 해당 인증서 각각에 대해 다음과 같은 정확한 파일 이름을 사용해야 합니다. CA 인증서를 ca-cert.pem, 키는 ca-key.pem이라고 합니다. ca-cert.pem을 서명하는 루트 인증서는 root-cert.pem이

라고 합니다. 워크로드에서 중개 인증서를 사용하는 경우 **cert-chain.pem** 파일에 인증서를 지정해야 합니다.

1. Maistra 리포지토리에서 로컬로 예제 인증서를 저장하고 **&lt ;path& gt;**를 인증서 경로로 바꿉니다.

2. 입력 파일 **ca-cert.pem,ca-key.pem,root-cert.pem** 및 **cert-chain.pem** 을 포함하는 **cacert** 라는 시크릿을 생성합니다.

```
$ 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** 리소스에서 **spec.security.dataPlane.mtls true** 를 **true** 로 설정하고 다음 예와 같이 **certificateAuthority** 필드를 구성합니다. 기본 **rootCADir**는 **/etc/cacerts**입니다. 키와 인증서가 기본 위치에 마운트된 경우 **privateKey**를 설정할 필요가 없습니다. 서비스 메시는 **secret-mount** 파일에서 인증서와 키를 읽습니다.

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  security:
    dataPlane:
      mtls: true
    certificateAuthority:
      type: Istiod
      istiod:
        type: PrivateKey
        privateKey:
          rootCADir: /etc/cacerts
```

4. **cacert** 시크릿을 생성/변경/삭제한 후 **Service Mesh Control Plane istiod** 및 게이트웨이 **pod**를 다시 시작해야 변경 사항이 적용됩니다. 다음 명령을 사용하여 **Pod**를 재시작합니다.

```
$ oc -n istio-system delete pods -l 'app in (istiod,istio-ingressgateway, istio-
egressgateway)'
```

Operator는 삭제된 후 **Pod**를 자동으로 다시 생성합니다.

5. 사이드카 프록시가 시크릿 변경 사항을 선택하도록 **info** 애플리케이션 **pod**를 다시 시작합니다. 다음 명령을 사용하여 **Pod**를 재시작합니다.

```
$ oc -n info delete pods --all
```

출력은 다음과 유사합니다.

```
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. Pod가 생성되고 다음 명령을 사용하여 준비되었는지 확인합니다.

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

1.13.4.2. 인증서 확인

Bookinfo 샘플 애플리케이션을 사용하여 CA에 연결된 인증서로 워크로드 인증서에 서명하는지 확인합니다. 이 프로세스에서는 openssl 을 시스템에 설치해야 합니다.

1. 정보 워크로드에서 인증서를 추출하려면 다음 명령을 사용합니다.

```
$ 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
```

명령을 실행한 후 작업 디렉터리에 proxy-cert-1.pem,proxy-cert-2.pem 및 proxy-cert-3.pem 의 3개의 파일이 있어야 합니다.

2. 루트 인증서가 관리자가 지정한 것과 동일한지 확인합니다. <path>를 인증서 경로로 교체합니다.

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

터미널 창에서 다음 구문을 실행합니다.

```
$ openssl x509 -in ./proxy-cert-3.pem -text -noout > /tmp/pod-root-cert.crt.txt
```

터미널 창에서 다음 구문을 실행하여 인증서를 비교합니다.

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

다음 결과가 표시됩니다. **Files /tmp/root-cert.crt.txt** 및 **/tmp/pod-root-cert.crt.txt**는 동일합니다.

3. **CA** 인증서가 관리자가 지정한 것과 동일한지 확인합니다. **<path>**를 인증서 경로로 교체합니다.

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

터미널 창에서 다음 구문을 실행합니다.

```
$ openssl x509 -in ./proxy-cert-2.pem -text -noout > /tmp/pod-cert-chain-ca.crt.txt
```

터미널 창에서 다음 구문을 실행하여 인증서를 비교합니다.

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

다음 결과가 표시됩니다. **Files /tmp/ca-cert.crt.txt** 및 **/tmp/pod-cert-chain-ca.crt.txt**는 동일합니다.

4. 루트 인증서에서 워크로드 인증서로의 인증서 체인을 확인합니다. **<path>**를 인증서 경로로 교체합니다.

```
$ openssl verify -CAfile <(cat <path>/ca-cert.pem <path>/root-cert.pem) ./proxy-cert-1.pem
```

다음과 같은 결과가 표시됩니다. **./proxy-cert-1.pem: OK**

1.13.4.3. 인증서 제거

추가한 인증서를 제거하려면 다음 단계를 따르십시오.

1. 시크릿 **cacerts**를 제거합니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 프로젝트의 이름입니다.

```
$ oc delete secret cacerts -n istio-system
```

2. **ServiceMeshControlPlane** 리소스에서 자체 서명된 루트 인증서로 서비스 메시를 재배포합니다.

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

1.13.5. cert-manager 및 istio-csr와 서비스 메시 통합 정보

cert-manager 툴은 **Kubernetes**에서 **X.509** 인증서 관리를 위한 솔루션입니다. **Vault, Google Cloud Certificate Authority Service, Let's Encrypt** 및 기타 공급자와 같은 개인 또는 공개 키 인프라(**PKI**)와 애플리케이션을 통합하는 통합 **API**를 제공합니다.

cert-manager 툴은 만료되기 전에 구성된 시간에 인증서를 갱신하여 인증서가 유효하고 최신 상태인지 확인합니다.

Istio 사용자의 경우 **cert-manager**는 **Istio** 프록시에서 **CSR**(인증서 서명 요청)을 처리하는 **CA**(인증 기관) 서버인 **istio-csr** 와의 통합을 제공합니다. 그런 다음 서버는 **CSR**을 구성된 **CA** 서버로 전달하는 **cert-manager**에 서명을 위임합니다.

참고

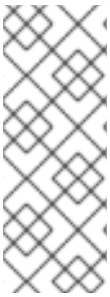
Red Hat은 **istio-csr** 및 **cert-manager**와의 통합을 지원합니다. **Red Hat**은 **istio-csr** 또는 커뮤니티 **cert-manager** 구성 요소에 대한 직접 지원을 제공하지 않습니다. 여기에 표시된 커뮤니티 **cert-manager**를 사용하는 것은 데모 목적으로만 사용됩니다.

사전 요구 사항

- **cert-manager** 버전 중 하나:

- cert-manager Operator for Red Hat OpenShift 1.10 이상
- 커뮤니티 cert-manager Operator 1.11 이상
- cert-manager 1.11 이상

- OpenShift Service Mesh Operator 2.4 이상
- Istio-csr 0.6.0 이상



참고

jetstack/cert-manager- istio-csr Helm 차트를 사용하여 istio-csr 서버를 설치할 때 모든 네임스페이스에 구성 맵을 생성하지 않으려면 istio-csr .yaml 파일에서 다음 설정을 사용합니다. app.controller.configmapNamespaceSelector: "maistra.io/member-namespace >"

1.13.5.1. cert-manager 설치

cert-manager 툴을 설치하여 TLS 인증서의 라이프사이클을 관리하고 유효하고 최신 상태인지 확인할 수 있습니다. 환경에서 Istio를 실행하는 경우 Istio 프록시에서 CSR(인증서 서명 요청)을 처리하는 istio-csr 인증 기관(CA) 서버를 설치할 수도 있습니다. istio-csr CA는 구성된 CA에 위임하는 cert-manager 툴에 서명을 위임합니다.

절차

1. 루트 클러스터 발행자를 생성합니다.

```
$ oc apply -f cluster-issuer.yaml
```

```
$ oc apply -n istio-system -f istio-ca.yaml
```

예: cluster-issuer.yaml

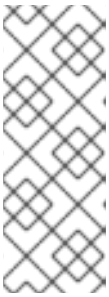
```
apiVersion: cert-manager.io/v1
kind: Issuer
```

```
  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
```

예: istio-ca.yaml

```
  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
```



참고

root-ca 는 클러스터 발행자이므로 selfsigned-root-issuer 발행자 및 root-ca 인증서의 네임스페이스가 cert-manager 이므로 cert-manager는 자체 네임스페이스에서 참조된 시크릿을 찾습니다. Red Hat OpenShift용 cert-manager Operator의 경우 자체 네임스페이스는 cert-manager입니다.

2. istio-csr 설치 :

```
$ helm install istio-csr jetstack/cert-manager-istio-csr \
    -n istio-system \
    -f deploy/examples/cert-manager/istio-csr/istio-csr.yaml
```

예: istio-csr.yaml

```
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
```

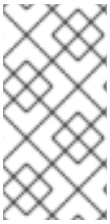
3. SMCP를 배포합니다.

```
$ oc apply -f mesh.yaml -n istio-system
```

예: 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
```



참고

security.identity.type: security.certificateAuthority.type: cert-manager 가 구성된 경우 타사 를 설정해야 합니다.

검증

샘플 **httpbin** 서비스 및 **sleep** 앱을 사용하여 수신 게이트웨이의 **mTLS** 트래픽을 확인하고 **cert-manager** 툴이 설치되었는지 확인합니다.

1. **HTTP** 및 절전 앱을 배포합니다.

```
$ 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** 에서 **httpbin** 서비스에 액세스할 수 있는지 확인합니다.

```
$ 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"
```

출력 예:

```
200
```

3. 수신 게이트웨이에서 **httpbin** 서비스로 **mTLS** 트래픽을 확인합니다.

```
$ oc apply -n <namespace> -f https://raw.githubusercontent.com/maistra/istio/maistra-2.4/samples/httpbin/httpbin-gateway.yaml
```

4. **istio-ingressgateway** 경로를 가져옵니다.

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

5. 수신 게이트웨이에서 **httpbin** 서비스로의 **mTLS** 트래픽을 확인합니다.

```
$ curl -s -I http://$INGRESS_HOST/headers -o /dev/null -w "%{http_code}" -s
```

1.13.6. 추가 리소스

OpenShift Container Platform 용 **cert-manager Operator**를 설치하는 방법에 대한 자세한 내용은 **cert-manager Operator for Red Hat OpenShift** 설치를 참조하십시오.

1.14. 서비스 메시의 트래픽 관리

Red Hat OpenShift Service Mesh를 사용하면 서비스 간 트래픽 흐름과 **API** 호출을 제어할 수 있습니다. 서비스 메시의 일부 서비스는 메시 내에서 통신해야 하며 다른 서비스는 숨겨야 할 수 있습니다. 트래픽을 관리하여 특정 백엔드 서비스를 숨기고, 서비스를 노출하거나, 테스트 또는 버전 관리 배포를 생성하거나, 서비스 세트에 보안 계층을 추가할 수 있습니다.

1.14.1. 게이트웨이 사용

게이트웨이를 사용하여 메시에 대한 인바운드 및 아웃바운드 트래픽을 관리하여 메시에 들어가거나 나가려는 트래픽을 지정할 수 있습니다. 게이트웨이 구성은 서비스 워크로드와 함께 실행되는 사이드카 **Envoy** 프록시가 아닌, 메시의 에지에서 실행되는 독립 실행형 **Envoy** 프록시에 적용됩니다.

Kubernetes Ingress API와 같이 시스템으로 들어오는 트래픽을 제어하는 다른 메커니즘과 달리 **Red Hat OpenShift Service Mesh** 게이트웨이는 트래픽 라우팅의 모든 기능과 유연성을 사용합니다.

Red Hat OpenShift Service Mesh 게이트웨이 리소스는 Red Hat OpenShift Service Mesh TLS 설정을 노출하고 구성하기 위해 포트와 같은 계층 4-6 로드 밸런싱 속성을 사용할 수 있습니다. 애플리케이션 계층 트래픽 라우팅(L7)을 동일한 API 리소스에 추가하는 대신, 일반 Red Hat OpenShift Service Mesh 가상 서비스를 게이트웨이에 바인딩하고 서비스 메시의 다른 데이터 플레인 트래픽처럼 게이트웨이 트래픽을 관리할 수 있습니다.

게이트웨이는 주로 수신 트래픽을 관리하는 데 사용되지만 송신 게이트웨이를 구성할 수도 있습니다. 송신 게이트웨이를 사용하면 메시를 나가는 트래픽에 대해 전용 종료 노드를 구성할 수 있습니다. 이를 통해 외부 네트워크에 대한 액세스 권한이 있는 서비스를 제한하여 서비스 메시에 보안 제어를 추가할 수 있습니다. 게이트웨이를 사용하여 전적으로 내부 프록시를 구성할 수도 있습니다.

게이트웨이 예제

게이트웨이 리소스는 들어오거나 나가는 HTTP/TCP 연결을 수신하는 메시의 에지에서 작동하는 로드 밸런서를 설명합니다. 사양은 노출되는 포트 세트, 사용할 프로토콜 유형, 로드 밸런서에 대한 SNI 구성 등을 설명합니다.

다음 예제는 외부 HTTPS 수신 트래픽에 대해 샘플 게이트웨이 구성을 보여줍니다.

```
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
```

이 게이트웨이 구성으로 ext-host.example.com의 HTTPS 트래픽을 포트 443의 메시로 허용할 수 있지만 트래픽에 라우팅을 지정하지 않습니다.

라우팅을 지정하고 게이트웨이가 의도한 대로 작동하려면 게이트웨이도 가상 서비스에 바인딩해야 합니다. 다음 예와 같이 가상 서비스의 게이트웨이 필드를 사용하여 이 작업을 수행합니다.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: virtual-svc
spec:
  hosts:
  - ext-host.example.com
  gateways:
    - ext-host-gwy
```

그러면 외부 트래픽에 대한 라우팅 규칙으로 가상 서비스를 구성할 수 있습니다.

1.14.1.1. 게이트웨이 삽입 활성화

게이트웨이 구성은 서비스 워크로드와 함께 실행되는 사이드카 **Envoy** 프록시가 아닌 메시의 에지에서 실행되는 독립 실행형 **Envoy** 프록시에 적용됩니다. 게이트웨이는 **Envoy** 프록시이므로 사이드카를 삽입할 수 있는 방식과 유사하게 게이트웨이를 자동으로 삽입하도록 서비스 메시를 구성할 수 있습니다.

게이트웨이에 대한 자동 삽입을 사용하여 **ServiceMeshControlPlane** 리소스와 독립적으로 게이트웨이를 배포 및 관리하고 사용자 애플리케이션으로 게이트웨이를 관리할 수 있습니다. 게이트웨이 배포에 **autoinjection**을 사용하면 개발자에게 게이트웨이 배포를 완전히 제어하고 작업을 단순화할 수 있습니다. 새 업그레이드를 사용할 수 있거나 구성이 변경되면 게이트웨이 **Pod**를 다시 시작하여 업데이트합니다. 이렇게 하면 게이트웨이 배포를 운영 중인 사이드카와 동일하게 작동합니다.

참고

ECDHE은 **ServiceMeshControlPlane** 네임스페이스에 대해 기본적으로 비활성화되어 있습니다(예: **istio-system** 네임스페이스). 보안 모범 사례로 컨트롤 플레인과 다른 네임스페이스에 게이트웨이를 배포합니다.

1.14.1.2. 자동 게이트웨이 삽입 배포

게이트웨이를 배포할 때 **gateway** 배포 오브젝트에 삽입 레이블 또는 주석을 추가하여 삽입을 선택해야 합니다. 다음 예제에서는 게이트웨이를 배포합니다.

사전 요구 사항

- 네임스페이스는 **ServiceMeshMemberRoll** 에서 정의하거나 **ServiceMeshMember** 리소스를 생성하여 메시의 멤버여야 합니다.

절차

1. Istio 수신 게이트웨이의 고유 레이블을 설정합니다. 게이트웨이가 워크로드를 선택할 수 있도록 하려면 이 설정이 필요합니다. 이 예에서는 ingressgateway 를 게이트웨이 이름으로 사용합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: istio-ingressgateway
  namespace: istio-ingress
spec:
  type: ClusterIP
  selector:
    istio: ingressgateway
  ports:
  - name: http2
    port: 80
    targetPort: 8080
  - name: https
    port: 443
    targetPort: 8443
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: istio-ingressgateway
  namespace: istio-ingress
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  template:
    metadata:
      annotations:
        inject.istio.io/templates: gateway
      labels:
        istio: ingressgateway
        sidecar.istio.io/inject: "true" 1
    spec:
      containers:
      - name: istio-proxy
        image: auto 2
```

1 sidecar.istio.io/inject 필드를 "true" 로 설정하여 게이트웨이 삽입을 활성화합니다.

2 Pod가 시작될 때마다 이미지가 자동으로 업데이트되도록 image 필드를 auto 로 설정합니다.

2. TLS에 대한 자격 증명을 읽을 수 있도록 역할을 설정합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: istio-ingressgateway-sds
  namespace: istio-ingress
rules:
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["get", "watch", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: istio-ingressgateway-sds
  namespace: istio-ingress
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: istio-ingressgateway-sds
subjects:
- kind: ServiceAccount
  name: default
```

3. 클러스터 외부에서 새 게이트웨이에 대한 액세스 권한을 부여합니다. 이 액세스 권한은 spec.security.manageNetworkPolicy 가 true 로 설정될 때마다 필요합니다.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: gatewayingress
  namespace: istio-ingress
spec:
  podSelector:
    matchLabels:
      istio: ingressgateway
  ingress:
    - {}
  policyTypes:
  - Ingress
```

4. 수신 트래픽이 증가하면 Pod를 자동으로 스케일링합니다. 이 예에서는 최소 복제본을 2 로 설정하고 최대 복제본을 5 개로 설정합니다. 또한 사용률이 80%에 도달하면 다른 복제본을 생성합니다.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
```

```
  labels:
    istio: ingressgateway
    release: istio
  name: ingressgatewayhpa
  namespace: istio-ingress
spec:
  maxReplicas: 5
  metrics:
  - resource:
      name: cpu
      target:
        averageUtilization: 80
        type: Utilization
    type: Resource
  minReplicas: 2
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: istio-ingressgateway
```

5. 노드에서 실행 중이어야 하는 최소 **Pod** 수를 지정합니다. 이 예에서는 새 노드에서 **Pod**를 다시 시작하면 하나의 복제본이 실행됩니다.

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  labels:
    istio: ingressgateway
    release: istio
  name: ingressgatewaypdb
  namespace: istio-ingress
spec:
  minAvailable: 1
  selector:
    matchLabels:
      istio: ingressgateway
```

1.14.1.3. Ingress 트래픽 관리

Red Hat OpenShift Service Mesh에서 **Ingress Gateway**는 모니터링, 보안 및 라우팅 규칙과 같은 기능을 클러스터에 들어오는 트래픽에 적용할 수 있도록 합니다. 서비스 메시 게이트웨이를 사용하여 서비스 메시 외부에서 서비스를 노출합니다.

1.14.1.3.1. Ingress IP 및 포트 확인

Ingress 구성은 환경에서 외부 로드 밸런서를 지원하는지 여부에 따라 달라집니다. 외부 로드 밸런서는 클러스터의 **Ingress IP** 및 포트에 설정됩니다. 클러스터의 **IP** 및 포트가 외부 로드 밸런서에 구성되어 있는지 확인하려면 다음 명령을 실행합니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 프로젝트의 이름입니다.

```
$ oc get svc istio-ingressgateway -n istio-system
```

해당 명령은 네임스페이스에 있는 각 항목의 **NAME, TYPE, CLUSTER-IP, EXTERNAL-IP, PORT(S), AGE**를 반환합니다.

EXTERNAL-IP 값이 설정되면 해당 환경에 **Ingress** 게이트웨이에 사용할 수 있는 외부 로드 밸런서가 있습니다.

EXTERNAL-IP 값이 **<none>** 또는 영구적으로 **<pending>**인 경우, 해당 환경은 **Ingress** 게이트웨이에 외부 로드 밸런서를 제공하지 않습니다. 서비스의 노드 포트를 사용하여 게이트웨이에 액세스할 수 있습니다.

1.14.1.3.1.1. 로드 밸런서를 사용하여 Ingress 포트 확인

환경에 외부 로드 밸런서가 있는 경우 다음 지침을 따릅니다.

절차

1. 다음 명령을 실행하여 **Ingress IP** 및 포트를 설정합니다. 이 명령은 터미널에서 변수를 설정합니다.

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

2. 다음 명령을 실행하여 **Ingress** 포트를 설정합니다.

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

3. 다음 명령을 실행하여 보안 **Ingress** 포트를 설정합니다.

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

4. 다음 명령을 실행하여 **TCP Ingress** 포트를 설정합니다.

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

참고

일부 환경에서는 IP 주소 대신 호스트 이름을 사용하여 로드 밸런서가 노출될 수 있습니다. 이 경우 Ingress 게이트웨이의 EXTERNAL-IP 값은 IP 주소가 아닙니다. 대신 호스트 이름이며 이전 명령은 INGRESS_HOST 환경 변수를 설정하지 못합니다.

이 경우 다음 명령을 사용하여 INGRESS_HOST 값을 수정합니다.

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

1.14.1.3.1.2. 로드 밸런서 없이 Ingress 포트 확인

환경에 외부 로드 밸런서가 없는 경우 Ingress 포트를 확인하고 대신 노드 포트를 사용합니다.

절차

1. Ingress 포트를 설정합니다.

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

2. 다음 명령을 실행하여 보안 Ingress 포트를 설정합니다.

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

3. 다음 명령을 실행하여 TCP Ingress 포트를 설정합니다.

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

1.14.1.4. 수신 게이트웨이 구성

Ingress 게이트웨이는 들어오는 HTTP/TCP 연결을 수신하는 메시의 에지에서 작동하는 로드 밸런서입니다. 노출된 포트와 프로토콜을 구성하지만 트래픽 라우팅 구성은 포함하지 않습니다. Ingress 트래픽의 트래픽 라우팅은 내부 서비스 요청과 동일한 방식으로 라우팅 규칙으로 구성됩니다.

다음 단계에서는 게이트웨이를 만들고 **Bookinfo** 샘플 애플리케이션에서 서비스를 **/productpage** 및 **/login.** 경로의 외부 트래픽에 노출하도록 **VirtualService**를 구성하는 방법을 보여줍니다.

절차

1. 트래픽을 수락하기 위해 게이트웨이를 만듭니다.

a. **YAML** 파일을 생성한 후 다음 **YAML**을 이 파일에 복사합니다.

게이트웨이 예제 **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** 파일을 적용합니다.

```
$ oc apply -f gateway.yaml
```

2. **VirtualService** 오브젝트를 생성하여 호스트 헤더를 다시 작성합니다.

a. **YAML** 파일을 생성한 후 다음 **YAML**을 이 파일에 복사합니다.

가상 서비스 예

```
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 파일을 적용합니다.

```
$ oc apply -f vs.yaml
```

3. 게이트웨이 및 VirtualService가 올바르게 설정되었는지 확인합니다.

a. 게이트웨이 URL을 설정합니다.

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

b. 포트 번호를 설정합니다. 이 예제에서 istio-system 은 Service Mesh Control Plane 프로젝트의 이름입니다.

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

c.

명시적으로 노출된 페이지를 테스트합니다.

```
curl -s -I "$GATEWAY_URL/productpage"
```

예상 결과는 200입니다.

1.14.2. 자동 경로 이해

게이트웨이의 OpenShift 경로는 Service Mesh에서 자동으로 관리됩니다. Istio 게이트웨이가 서비스 메시 내부에서 생성, 업데이트 또는 삭제될 때마다 OpenShift 경로가 생성, 업데이트 또는 삭제됩니다.

1.14.2.1. 하위 도메인이 있는 경로

Red Hat OpenShift Service Mesh 는 하위 도메인으로 경로를 생성하지만 이를 활성화하려면 OpenShift Container Platform을 구성해야 합니다. 하위 도메인(예: *.domain.com)은 지원되지만 기본적으로는 지원되지 않습니다. 와일드카드 호스트 게이트웨이를 구성하기 전에 OpenShift Container Platform 와일드카드 정책을 구성합니다.

자세한 내용은 와일드카드 경로 사용을 참조하십시오.

1.14.2.2. 하위 도메인 경로 생성

다음 예제에서는 Bookinfo 샘플 애플리케이션에 게이트웨이를 생성하여 하위 도메인 경로를 생성합니다.

```
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 리소스는 다음 OpenShift 경로를 생성합니다. 다음 명령을 사용하여 경로가 생성되었는지 확인할 수 있습니다. 이 예제에서 istio-system 은 Service Mesh Control Plane 프로젝트의 이름입니다.

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

예상 출력

```
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
```

게이트웨이를 삭제하면 Red Hat OpenShift Service Mesh가 경로를 삭제합니다. 그러나 수동으로 생성한 경로는 Red Hat OpenShift Service Mesh에 의해 수정되지 않습니다.

1.14.2.3. 경로 라벨 및 주석

OpenShift 경로에 특정 레이블 또는 주석이 필요한 경우가 있습니다. 예를 들어 OpenShift 경로의 일부 고급 기능은 특수 주석을 사용하여 관리됩니다. 다음 "추가 리소스" 섹션의 "경로별 주석"을 참조하십시오.

이 사용 사례 및 기타 사용 사례에서 Red Hat OpenShift Service Mesh는 Istio 게이트웨이 리소스에 있는 모든 레이블 및 주석(kubectl.kubernetes.io로 시작하는 주석 제외)을 관리형 OpenShift 경로 리소스로 복사합니다.

Service Mesh에서 생성한 OpenShift 경로에 특정 레이블 또는 주석이 필요한 경우 Istio 게이트웨이 리소스에서 생성하고 Service Mesh에서 관리하는 OpenShift 경로 리소스에 복사됩니다.

추가 리소스

- 경로별 주석.

1.14.2.4. 자동 경로 생성 비활성화

기본적으로 ServiceMeshControlPlane 리소스는 Istio 게이트웨이 리소스를 OpenShift 경로와 자동으로 동기화합니다. 자동 경로 생성을 비활성화하면 특별한 경우가 있거나 경로를 수동으로 제어하려는

경우 보다 유연하게 경로를 제어할 수 있습니다.

1.14.2.4.1. 특정 사례에 대한 자동 경로 생성 비활성화

특정 Istio 게이트웨이의 OpenShift 경로 자동 관리를 비활성화하려면 주석 maistra.io/manageRoute: false 를 게이트웨이 메타데이터 정의에 추가해야 합니다. Red Hat OpenShift Service Mesh는 다른 Istio 게이트웨이를 자동으로 관리하는 동안 이 주석이 있는 Istio 게이트웨이를 무시합니다.

1.14.2.4.2. 모든 사례에 대한 자동 경로 생성 비활성화

메시의 모든 게이트웨이에 대한 OpenShift 경로 자동 관리를 비활성화할 수 있습니다.

ServiceMeshControlPlane 필드 gateways.openshiftRoute.enabled 를 false 로 설정하여 Istio 게이트웨이와 OpenShift 경로 간의 통합을 비활성화합니다. 예를 들어, 다음 리소스 스니펫을 참조하십시오.

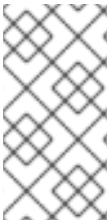
```
apiVersion: maistra.io/v1alpha1
kind: ServiceMeshControlPlane
metadata:
  namespace: istio-system
spec:
  gateways:
    openshiftRoute:
      enabled: false
```

1.14.3. 서비스 항목 이해

서비스 항목은 Red Hat OpenShift Service Mesh가 내부적으로 관리하는 서비스 레지스트리에 항목을 추가합니다. 서비스 항목을 추가한 후 Envoy 프록시는 메시의 서비스인 것처럼 서비스에 트래픽을 보냅니다. 서비스 항목을 사용하면 다음을 수행할 수 있습니다.

- 서비스 메시 외부에서 실행되는 서비스의 트래픽을 관리합니다.
- 웹에서 소비된 API 또는 레거시 인프라의 서비스에 대한 트래픽과 같은 외부 대상의 트래픽을 리디렉션 및 전달합니다.
- 외부 대상에 대한 재시도, 시간 초과 및 오류 삽입 정책을 정의합니다.

- VM(가상 머신)에서 메시에 VM을 추가하여 메시 서비스를 실행합니다.



참고

Kubernetes에서 다중 클러스터 Red Hat OpenShift Service Mesh 메시를 구성하기 위해 다른 클러스터의 서비스를 메시에 추가합니다.

서비스 항목 예

다음 예제는 Red Hat OpenShift Service Mesh 서비스 레지스트리에 ext-resource 외부 종속성을 추가하는 mesh-external 서비스 항목입니다.

```
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 필드를 사용하여 외부 리소스를 지정합니다. 완전히 한정하거나 와일드카드 접두사 도메인 이름을 사용할 수 있습니다.

메시의 다른 서비스에 대한 트래픽을 구성하는 것과 동일한 방식으로 서비스 항목에 대한 트래픽을 제어하도록 가상 서비스 및 대상 규칙을 구성할 수 있습니다. 예를 들어 다음 대상 규칙은 서비스 항목을 사용하여 구성된 ext-svc.example.com 외부 서비스에 대한 연결을 보호하기 위해 상호 TLS를 사용하도록 트래픽 경로를 구성합니다.

```
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. VirtualService 사용

가상 서비스가 있는 **Red Hat OpenShift Service Mesh**를 통해 여러 버전의 마이크로 서비스로 요청을 동적으로 라우팅할 수 있습니다. 가상 서비스를 사용하면 다음을 수행할 수 있습니다.

- 단일 가상 서비스를 통해 여러 애플리케이션 서비스를 처리합니다. 예를 들어 메시에서 **Kubernetes**를 사용하는 경우 특정 네임스페이스의 모든 서비스를 처리하도록 가상 서비스를 구성할 수 있습니다. 가상 서비스를 사용하면 모놀리식 애플리케이션을 원활한 소비자 환경을 통해 별도의 마이크로 서비스로 구성된 서비스로 전환할 수 있습니다.
- 게이트웨이와 결합하여 트래픽 규칙을 구성하고 수신 및 송신 트래픽을 제어합니다.

1.14.4.1. VirtualService 구성

요청은 가상 서비스를 통해 서비스 메시 내의 서비스로 라우팅됩니다. 각 가상 서비스는 순서대로 평가되는 라우팅 규칙 세트로 구성됩니다. **Red Hat OpenShift Service Mesh**는 가상 서비스에 대해 주어진 각 요청을 메시 내의 실제 특정 대상에 연결합니다.

가상 서비스가 없으면 **Red Hat OpenShift Service Mesh**는 모든 서비스 인스턴스 간에 최소 요청 부하 분산을 사용하여 트래픽을 배포합니다. 가상 서비스에서는 하나 이상의 호스트 이름에 대한 트래픽 동작을 지정할 수 있습니다. 가상 서비스의 라우팅 규칙은 가상 서비스에 대한 트래픽을 적절한 대상으로 전송하는 방법을 **Red Hat OpenShift Service Mesh**에 알립니다. 경로 대상은 동일한 서비스 또는 완전히 다른 서비스 버전일 수 있습니다.

절차

1. 다음 예제를 사용하여 **YAML** 파일을 생성하여 애플리케이션에 연결하는 사용자에 따라 **Bookinfo** 샘플 애플리케이션 서비스의 다양한 버전으로 요청을 라우팅합니다.

예: VirtualService.yaml

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

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

2. 다음 명령을 실행하여 **VirtualService.yaml**을 적용합니다. 여기서 **VirtualService.yaml**은 파일 경로입니다.

```
$ oc apply -f <VirtualService.yaml>
```

1.14.4.2. VirtualService 구성 참조

매개변수	설명
`spec:` `  hosts:`	**hosts** 필드에는 라우팅 규칙이 적용되는 가상 서비스의 대상 주소가 나열됩니다. 이는 서비스에 요청을 보내는 데 사용되는 주소입니다. 가상 서비스 호스트 이름은 IP 주소, DNS 이름 또는 정규화된 도메인 이름으로 확인되는 짧은 이름일 수 있습니다.

`spec:`
`  http:`
`  - match:`
	http 섹션에는 호스트 필드에 지정된 대상으로 전송된 HTTP/1.1, HTTP2, gRPC 트래픽을 라우팅하기 위한 일치 조건 및 작업을 설명하는 가상 서비스의 라우팅 규칙이 포함됩니다. 라우팅 규칙은 트래픽을 이동할 대상과 지정된 일치 조건으로 구성됩니다. 예제의 첫 번째 라우팅 규칙에는 일치 필드로 시작하는 조건이 있습니다. 이 예제에서 이 라우팅은 사용자 **jason**의 모든 요청에 적용됩니다. **headers**, **end-user**, **exact** 필드를 추가하여 적절한 요청을 선택합니다.

매개변수	설명
spec: http: - match: - destination:	경로 섹션의 **destination** 필드는 이 조건과 일치하는 트래픽에 대한 실제 대상을 지정합니다. 가상 서비스의 호스트와 달리 대상 호스트는 Red Hat OpenShift Service Mesh 서비스 레지스트리에 있는 실제 대상이어야 합니다. 프록시가 있는 메시 서비스 또는 서비스 항목을 사용하여 추가된 비 메시 서비스일 수 있습니다. 이 예제에서 호스트 이름은 Kubernetes 서비스 이름입니다.

1.14.5. 대상 규칙 이해

대상 규칙은 가상 서비스 라우팅 규칙이 평가된 후에 적용되므로 트래픽의 실제 대상에 적용됩니다. 가상 서비스는 트래픽을 대상으로 라우팅합니다. 대상 규칙은 해당 대상의 트래픽에 발생하는 요소를 설정합니다.

기본적으로 **Red Hat OpenShift Service Mesh**는 최소 요청 부하 분산 정책을 사용합니다. 여기서 활성 연결 수가 가장 적은 풀의 서비스 인스턴스에서 요청을 수신합니다. 또한 **Red Hat OpenShift Service Mesh**는 특정 서비스 또는 서비스 하위 집합에 대한 요청의 대상 규칙에 지정할 수 있는 다음과 같은 모델을 지원합니다.

- **Random:** 요청은 풀의 인스턴스에 무작위로 전달됩니다.
- **Weighted:** 요청은 구체적인 비율에 따라 풀의 인스턴스로 전달됩니다.
- **Least requests:** 요청은 요청 수가 가장 적은 인스턴스로 전달됩니다.

대상 규칙 예

다음 예제 대상 규칙은 서로 다른 로드 밸런싱 정책을 사용하여 **my-svc** 대상 서비스에 대해 세 가지 다른 하위 집합을 구성합니다.

```
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. 네트워크 정책 이해

Red Hat OpenShift Service Mesh는 **Service Mesh Control Plane** 및 애플리케이션 네임스페이스에서 여러 **NetworkPolicies** 리소스를 자동으로 생성하고 관리합니다. 애플리케이션과 컨트롤 플레인이 서로 통신할 수 있도록 하기 위한 것입니다.

예를 들어 **SDN** 플러그인을 사용하도록 **OpenShift Container Platform** 클러스터를 구성한 경우 **Red Hat OpenShift Service Mesh**는 각 멤버 프로젝트에서 **NetworkPolicy** 리소스를 생성합니다. 이를 통해 다른 메시 멤버 및 컨트롤 플레인의 메시에 있는 모든 **Pod**에 수신이 가능합니다. 또한 멤버 프로젝트 전용 수신으로 제한합니다. 멤버 외 프로젝트에서 수신이 필요한 경우 해당 트래픽을 허용하기 위해 **NetworkPolicy**를 생성해야 합니다. **Service Mesh**에서 네임스페이스를 제거하면 이 **NetworkPolicy** 리소스는 프로젝트에서 삭제됩니다.

1.14.6.1. 자동 NetworkPolicy 생성 비활성화

예를 들어 회사 보안 정책을 시행하거나 메시의 **Pod**에 직접 액세스할 수 있도록 **NetworkPolicy** 리소스의 자동 생성 및 관리를 비활성화하려면 다음을 수행할 수 있습니다. **ServiceMeshControlPlane** 을 편집하고 **spec.security.manageNetworkPolicy** 를 **false** 로 설정할 수 있습니다.

참고

spec.security.manageNetworkPolicy Red Hat OpenShift Service Mesh를 비활성화하면 **NetworkPolicy** 오브젝트가 생성되지 않습니다. 시스템 관리자는 네트워크를 관리하고 이러한 문제가 발생할 수 있는 문제를 해결합니다.

사전 요구 사항

- **Red Hat OpenShift Service Mesh Operator** 버전 **2.1.1** 이상이 설치되었습니다.

- ServiceMeshControlPlane 리소스는 버전 2.1 이상으로 업데이트되었습니다.

절차

1. OpenShift Container Platform 웹 콘솔에서 Operator → 설치된 Operator를 클릭합니다.
2. 프로젝트 메뉴에서 Service Mesh Control Plane을 설치한 프로젝트 (예: istio-system)를 선택합니다.
3. Red Hat OpenShift Service Mesh Operator를 클릭합니다. Istio Service Mesh Control Plane 열에서 ServiceMeshControlPlane의 이름을 클릭합니다(예: basic-install).
4. ServiceMeshControlPlane 세부 정보 만들기 페이지에서 YAML을 클릭하여 구성을 수정합니다.
5. 이 예와 같이 ServiceMeshControlPlane 필드 spec.security.manageNetworkPolicy 를 false 로 설정합니다.

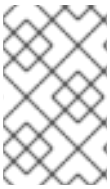
```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
spec:
  security:
    manageNetworkPolicy: false
```

6. 저장을 클릭합니다.

1.14.7. 트래픽 관리를 위한 사이드카 구성

기본적으로 Red Hat OpenShift Service Mesh는 연결된 워크로드의 모든 포트에서 트래픽을 허용하고 트래픽을 전달할 때 메시의 모든 워크로드에 도달할 수 있도록 모든 Envoy 프록시를 구성합니다. 사이드카 구성을 사용하여 다음을 수행할 수 있습니다.

- Envoy 프록시가 수락하는 포트와 프로토콜 집합을 미세 조정합니다.
- Envoy 프록시가 도달할 수 있는 서비스 집합을 제한합니다.



참고

서비스 메시의 성능을 최적화하려면 **Envoy** 프록시 구성을 제한하는 것이 좋습니다.

Bookinfo 샘플 애플리케이션에서 모든 서비스가 동일한 네임스페이스 및 컨트롤 플레인에서 실행되는 다른 서비스에 도달할 수 있도록 사이드카를 구성합니다. 이 사이드카 구성은 **Red Hat OpenShift Service Mesh** 정책 및 원격 분석 기능을 사용하는 데 필요합니다.

절차

1. 다음 예제를 사용하여 **YAML** 파일을 생성하여 특정 네임스페이스의 모든 워크로드에 사이드카 구성을 적용하도록 지정합니다. 그렇지 않으면 **workloadSelector**를 사용하여 특정 워크로드를 선택합니다.

예제 **sidecar.yaml**

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

2. 다음 명령을 실행하여 **sidecar.yaml**을 적용합니다. 여기서 **sidecar.yaml**은 파일의 경로입니다.

```
$ oc apply -f sidecar.yaml
```

3. 다음 명령을 실행하여 사이드카가 성공적으로 생성되었는지 확인합니다.

```
$ oc get sidecar
```

1.14.8. 라우팅 튜토리얼

이 안내서는 **Bookinfo** 샘플 애플리케이션을 참조하여 예제 애플리케이션에 라우팅 예제를 제공합니다. **Bookinfo** 애플리케이션을 설치하여 이러한 라우팅 예제가 작동하는 방법을 확인합니다.

1.14.8.1. Bookinfo 라우팅 튜토리얼

Service Mesh Bookinfo 샘플 애플리케이션은 각각 여러 가지 버전이 있는 네 개의 마이크로 서비스로 구성됩니다. **Bookinfo** 샘플 애플리케이션을 설치한 후에는 **reviews** 마이크로 서비스의 세 가지 버전이 동시에 실행됩니다.

브라우저에서 **Bookinfo** 앱 **/product** 페이지에 액세스하여 여러 번 새로 고침하면 북 리뷰 출력에 별점이 포함된 경우도 있고 그렇지 않은 경우도 있습니다. 라우팅할 명시적인 기본 서비스 버전이 없으면 서비스 메시는 사용 가능한 모든 버전으로 차례대로 요청을 라우팅합니다.

이 튜토리얼은 모든 트래픽을 마이크로 서비스의 **v1**(버전 **1**)으로 라우팅하는 규칙을 적용하는 데 도움이 됩니다. 나중에 **HTTP** 요청 헤더의 값을 기반으로 트래픽을 라우팅하는 규칙을 적용할 수 있습니다.

사전 요구 사항

- 다음 예제에서 작동하도록 **Bookinfo** 샘플 애플리케이션을 배포하십시오.

1.14.8.2. 가상 서비스 적용

다음 절차에서 가상 서비스는 마이크로 서비스의 기본 버전을 설정하는 가상 서비스를 적용하여 모든 트래픽을 각 마이크로 서비스의 **v1**로 라우팅합니다.

절차

1. 가상 서비스를 적용합니다.

```
$ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.4/samples/info/networking/virtual-service-all-v1.yaml
```

2. 가상 서비스를 적용했는지 확인하려면 다음 명령을 사용하여 정의된 경로를 표시합니다.

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

이 명령은 **kind: VirtualService**의 리소스를 **YAML** 형식으로 반환합니다.

reviews 서비스 버전 **1**을 포함하여 서비스 메시를 **Bookinfo** 마이크로 서비스 **v1** 버전으로 라우팅하도록 구성했습니다.

1.14.8.3. 새 경로 구성 테스트

Bookinfo 앱의 **/productpage**를 다시 새로 고침하여 새 구성을 테스트합니다.

절차

1. **GATEWAY_URL** 매개변수 값을 설정합니다. 이 변수를 사용하여 나중에 **Bookinfo** 제품 페이지의 **URL**을 찾을 수 있습니다. 이 예제에서 컨트롤 플레인 프로젝트는 **istio-system**입니다.

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

2. 다음 명령을 실행하여 제품 페이지의 **URL**을 검색합니다.

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

3. 브라우저에서 **Bookinfo** 사이트를 엽니다.

페이지의 리뷰 부분은 새로 고침 횟수와 관계없이 별점 없이 표시됩니다. 이는 리뷰 서비스의 모든 트래픽을 **reviews:v1** 버전으로 라우팅하도록 서비스 메시를 구성했기 때문이며, 이 서비스 버전은 별점 서비스에 액세스할 수 없습니다.

이제 서비스 메시가 트래픽을 하나의 서비스 버전으로 라우팅합니다.

1.14.8.4. 사용자 ID 기반 경로

특정 사용자의 모든 트래픽이 특정 서비스 버전으로 라우팅되도록 경로 구성을 변경합니다. 이 경우 **jason**이라는 사용자의 모든 트래픽은 서비스 **reviews:v2**로 라우팅됩니다.

서비스 메시에는 사용자 **ID**에 대한 특별한 기본 이해가 없습니다. 이 예제는 **productpage** 서비스가 모든 아웃바운드 **HTTP** 요청에 대한 사용자 정의 **end-user** 헤더를 검토 서비스에 추가한다는 사실에 의

해 활성화됩니다.

절차

1. 다음 명령을 실행하여 **Bookinfo** 샘플 애플리케이션에서 사용자 기반 라우팅을 활성화하도록 설정합니다.

```
$ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.4/samples/info/networking/virtual-service-reviews-test-v2.yaml
```

2. 다음 명령을 실행하여 규칙이 생성되었는지 확인합니다. 이 명령은 **kind: VirtualService**의 모든 리소스를 **YAML** 형식으로 반환합니다.

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

3. **Bookinfo** 앱의 **/productpage**에서 암호없이 **jason**으로 로그인합니다.

4. 브라우저를 새로 고침합니다. 별점은 각 리뷰 옆에 표시됩니다.

5. 다른 사용자로 로그인합니다(원하는 이름 선택). 브라우저를 새로 고침합니다. 이제 별이 사라졌습니다. **Jason**을 제외한 모든 사용자에 대해 트래픽이 **reviews:v1**으로 라우팅됩니다.

사용자 **ID**를 기반으로 트래픽을 라우팅하도록 **Bookinfo** 샘플 애플리케이션을 성공적으로 구성했습니다.

1.15. 메트릭, 로그 및 추적

메시에 애플리케이션을 추가한 후 애플리케이션을 통해 데이터 흐름을 확인할 수 있습니다. 자체 애플리케이션이 설치되어 있지 않은 경우 **Bookinfo** 샘플 애플리케이션을 설치하여 **Red Hat OpenShift Service Mesh**에서 관찰 기능이 작동하는 방식을 확인할 수 있습니다.

1.15.1. 콘솔 주소 검색

Red Hat OpenShift Service Mesh는 서비스 메시 데이터를 볼 수 있는 다음 콘솔을 제공합니다.

- **Kiali 콘솔 - Kiali는 Red Hat OpenShift Service Mesh의 관리 콘솔입니다.**

- **Jaeger 콘솔 - Jaeger는 Red Hat OpenShift 분산 추적 플랫폼의 관리 콘솔입니다.**

- **Grafana 콘솔 - Grafana는 메시 관리자에게 Istio 데이터에 대한 고급 쿼리 및 지표 분석 및 대시보드를 제공합니다. 선택적으로 Grafana를 사용하여 서비스 메시 메트릭을 분석할 수 있습니다.**

- **Prometheus 콘솔 - Red Hat OpenShift Service Mesh는 Prometheus를 사용하여 서비스의 Telemetry 정보를 저장합니다.**

Service Mesh Control Plane을 설치하면 설치된 각 구성 요소에 대한 경로가 자동으로 생성됩니다. 경로 주소가 있으면 Kiali, Jaeger, Prometheus 또는 Grafana 콘솔에 액세스하여 서비스 메시 데이터를 보고 관리할 수 있습니다.

사전 요구 사항

- **구성 요소가 활성화 및 설치되어 있어야 합니다. 예를 들어 분산 추적을 설치하지 않은 경우 Jaeger 콘솔에 액세스할 수 없습니다.**

OpenShift 콘솔의 절차

1. **OpenShift Container Platform 웹 콘솔에 cluster-admin 권한이 있는 사용자로 로그인합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.**

2. **네트워킹 → 경로로 이동합니다.**

3. **경로 페이지의 네임스페이스 메뉴에서 Service Mesh Control Plane 프로젝트(예: istio-system)를 선택합니다.**

   위치 열은 각 경로에 대한 연결된 주소를 표시합니다.

4. **필요한 경우 필터를 사용하여 액세스하려는 경로가 있는 구성 요소 콘솔을 찾습니다. 경로 위치를 클릭하여 콘솔을 시작합니다.**

5. **OpenShift**로 로그인 을 클릭합니다.

CLI의 프로세스

1. **cluster-admin** 역할의 사용자로 **OpenShift Container Platform CLI**에 로그인합니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 계정이 있어야 합니다.

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

2. **Service Mesh Control Plane** 프로젝트로 전환합니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 프로젝트입니다. 다음 명령을 실행합니다.

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

3. 다양한 **Red Hat OpenShift Service Mesh** 콘솔의 경로를 가져오려면 다음 명령을 실행합니다.

```
$ oc get routes
```

이 명령은 **Kiali, Jaeger, Prometheus** 및 **Grafana** 웹 콘솔의 **URL**과 서비스 메시의 기타 경로를 반환합니다. 출력은 다음과 유사합니다.

```
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** 열에서 액세스할 콘솔의 **URL**을 브라우저로 복사하여 콘솔을 엽니다.

5. **OpenShift**로 로그인 을 클릭합니다.

1.15.2. Kiali 콘솔에 액세스

Kiali 콘솔에서 애플리케이션의 토폴로지, 상태 및 지표를 볼 수 있습니다. 서비스에 문제가 발생하면 Kiali 콘솔을 사용하여 서비스를 통해 데이터 흐름을 볼 수 있습니다. 추상 애플리케이션, 서비스 및 워크로드를 포함하여 다양한 수준에서 메시 구성 요소에 대한 인사이트를 볼 수 있습니다. Kiali는 네임스페이스에 대한 대화형 그래프 보기도 실시간으로 제공합니다.

Kiali 콘솔에 액세스하려면 Red Hat OpenShift Service Mesh가 설치 및 구성되어 있어야 합니다.

설치 프로세스는 Kiali 콘솔에 액세스하기 위한 경로를 생성합니다.

Kiali 콘솔의 URL을 알고 있는 경우 직접 액세스할 수 있습니다. URL을 모르는 경우 다음 지침을 사용하십시오.

관리자의 절차

1. 관리자 역할을 사용하여 OpenShift Container Platform 웹 콘솔에 로그인합니다.
2. 홈 → 프로젝트를 클릭합니다.
3. 필요한 경우 프로젝트 페이지에서 필터를 사용하여 프로젝트 이름을 찾습니다.
4. 프로젝트 이름을 클릭합니다(예: info).
5. 프로젝트 세부 정보 페이지의 시작 관리자 섹션에서 Kiali 링크를 클릭합니다.
6. OpenShift Container Platform 콘솔에 액세스하는 데 사용하는 것과 동일한 사용자 이름 및 암호를 사용하여 Kiali 콘솔에 로그인합니다.

   Kiali 콘솔에 처음 로그인하면 볼 수 있는 권한이 있는 서비스 메시에 모든 네임스페이스가 표시되는 개요 페이지가 표시됩니다.

   콘솔 설치의 유효성을 검사하고 네임스페이스가 메시에 아직 추가되지 않은 경우 istio-system 이외의 데이터가 표시되지 않을 수 있습니다.

개발자용 절차

1. 개발자 역할을 사용하여 **OpenShift Container Platform** 웹 콘솔에 로그인합니다.

2. 프로젝트를 클릭합니다.

3. 필요한 경우 프로젝트 세부 정보 페이지에서 필터를 사용하여 프로젝트 이름을 찾습니다.

4. 프로젝트 이름을 클릭합니다(예: **info**).

5. 프로젝트 페이지의 시작 관리자 섹션에서 **Kiali** 링크를 클릭합니다.

6. **OpenShift**로 로그인 을 클릭합니다.

1.15.3. Kiali 콘솔에서 서비스 메시 데이터 보기

Kiali Graph는 메시 트래픽의 강력한 시각화를 제공합니다. 토폴로지는 실시간 요청 트래픽을 **Istio** 구성 정보와 결합하여 서비스 메시의 동작에 대한 즉각적인 통찰력을 제공하므로 문제를 신속하게 파악할 수 있습니다. 여러 그래프 유형을 사용하면 높은 수준의 서비스 토폴로지, 하위 수준 워크로드 토폴로지 또는 애플리케이션 수준 토폴로지로 트래픽을 시각화할 수 있습니다.

몇 가지의 그래프를 선택할 수 있습니다.

- 앱 그래프는 동일한 레이블이 있는 애플리케이션에 대한 집계 워크로드를 보여줍니다.

- 서비스 그래프는 메시의 각 서비스에 대한 노드를 표시되지만 그래프에서 모든 애플리케이션과 워크로드는 제외됩니다. 높은 수준의 보기를 제공하며 정의된 서비스에 대한 모든 트래픽을 집계합니다.

- 버전이 지정된 앱 그래프는 애플리케이션의 각 버전에 대한 노드를 보여줍니다. 모든 애플리케이션 버전이 함께 그룹화됩니다.

- 워크로드 그래프는 서비스 메시의 각 워크로드에 대한 노드를 표시합니다. 이 그래프는 애플리케이션 및 버전 레이블을 사용할 필요가 없습니다. 애플리케이션에서 버전 레이블을 사용하지

않는 경우 이 그래프를 사용하십시오.

그래프 노드는 다양한 정보로 장식되어 가상 서비스 및 서비스 항목과 같은 다양한 경로 라우팅 옵션과 오류 삽입 및 회로 차단기와 같은 특수 구성을 가리킵니다. **mTLS** 문제, 대기 시간 문제, 오류 트래픽 등을 확인할 수 있습니다. 그래프는 구성 가능하며, 트래픽 애니메이션을 표시할 수 있으며 강력한 찾기 및 숨기기 기능을 제공합니다.

범례 버튼을 클릭하여 그래프에 표시되는 모양, 색상, 화살표 및 배지에 대한 정보를 봅니다.

지표 요약을 보려면 그래프에서 노드 또는 에지를 선택하여 요약 세부 정보 패널에 지표 세부 정보를 표시합니다.

1.15.3.1. Kiali에서 그래프 레이아웃 변경

Kiali 그래프의 레이아웃은 애플리케이션 아키텍처 및 표시할 데이터에 따라 다르게 렌더링될 수 있습니다. 예를 들어 그래프 노드 수와 상호 작용은 **Kiali** 그래프가 렌더링되는 방법을 결정할 수 있습니다. 모든 상황에 적합하게 렌더링되는 단일 레이아웃을 생성할 수 없기 때문에 **Kiali**는 다양한 레이아웃을 선택할 수 있습니다.

사전 요구 사항

- 자체 애플리케이션이 설치되어 있지 않은 경우 **Bookinfo** 샘플 애플리케이션을 설치합니다. 그런 다음 다음 명령을 여러 번 입력하여 **Bookinfo** 애플리케이션에 대한 트래픽을 생성합니다.

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

이 명령은 애플리케이션의 **productpage** 마이크로 서비스에 액세스하는 사용자를 시뮬레이션합니다.

절차

1. **Kiali** 콘솔을 시작합니다.
2. **OpenShift**로 로그인 을 클릭합니다.
3. **Kiali** 콘솔에서 그래프 를 클릭하여 네임스페이스 그래프를 확인합니다.

4. 네임스페이스 메뉴에서 애플리케이션 네임스페이스를 선택합니다(예: **info**).

5. 다른 그래프 레이아웃을 선택하려면 다음 중 하나 또는 둘 다 수행합니다.

- 그래프 상단에 있는 메뉴에서 다른 그래프 데이터 그룹화를 선택합니다.
  - 앱 그래프
  - 서비스 그래프
  - 버전이 지정된 앱 그래프(기본값)
  - 워크로드 그래프
- 그래프 하단의 범인에서 다른 그래프 레이아웃을 선택합니다.
  - 레이아웃 기본 **dagre**
  - 레이아웃 **1** 공동 기능
  - 레이아웃 **2 cola**

1.15.3.2. Kiali 콘솔에서 로그 보기

Kiali 콘솔에서 워크로드 로그를 볼 수 있습니다. 워크로드 세부 정보 페이지에는 애플리케이션 및 프록시 로그 를 모두 표시하는 통합 로그 보기가 표시되는 로그 탭이 포함되어 있습니다. **Kiali**에 로그 표시를 원하는 빈도를 선택할 수 있습니다.

Kiali에 표시된 로그의 로깅 수준을 변경하려면 워크로드 또는 프록시의 로깅 구성을 변경합니다.

사전 요구 사항

- 서비스 메시가 설치 및 구성되어 있습니다.
- Kiali가 설치 및 구성되어 있습니다.
- Kiali 콘솔의 주소입니다.
- 메시에 추가된 애플리케이션 또는 Bookinfo 샘플 애플리케이션.

절차

1. Kiali 콘솔을 시작합니다.
2. OpenShift로 로그인 을 클릭합니다.

   Kiali 개요 페이지에는 볼 권한이 있는 메시에 추가된 네임스페이스가 표시됩니다.
3. 워크로드 를 클릭합니다.
4. 워크로드 페이지의 네임스페이스 메뉴에서 프로젝트를 선택합니다.
5. 필요한 경우 필터를 사용하여 보려는 로그를 찾습니다. 워크로드 이름을 클릭합니다. 예를 들어 ratings-v1 을 클릭합니다.
6. 워크로드 세부 정보 페이지에서 로그 탭을 클릭하여 워크로드 로그를 확인합니다.

작은 정보

로그 항목이 표시되지 않으면 시간 범위 또는 새로 고침 간격을 조정해야 할 수 있습니다.

1.15.3.3. Kiali 콘솔에서 지표 보기

Kiali 콘솔에서 애플리케이션, 워크로드 및 서비스에 대한 인바운드 및 아웃바운드 지표를 볼 수 있습

니다. 세부 정보 페이지에는 다음 탭이 포함됩니다.

- 인바운드 애플리케이션 메트릭
- 아웃바운드 애플리케이션 메트릭
- 인바운드 워크로드 지표
- 아웃바운드 워크로드 지표
- 인바운드 서비스 메트릭

이러한 탭에는 관련 애플리케이션, 워크로드 또는 서비스 수준에 맞는 사전 정의된 지표 대시보드가 표시됩니다. 애플리케이션 및 워크로드 세부 정보 보기에는 볼륨, 기간, 크기 또는 TCP 트래픽과 같은 요청 및 응답 지표가 표시됩니다. 서비스 세부 정보 보기에는 인바운드 트래픽에 대한 요청 및 응답 메트릭만 표시됩니다.

Kiali를 사용하면 차트 크기를 선택하여 차트를 사용자 지정할 수 있습니다. Kiali는 소스 또는 대상 프록시 메트릭에서 보고한 메트릭을 표시할 수도 있습니다. 또한 Kiali는 메트릭에 대한 추적 범위를 오버레이할 수 있습니다.

사전 요구 사항

- 서비스 메시가 설치 및 구성되어 있습니다.
- Kiali가 설치 및 구성되어 있습니다.
- Kiali 콘솔의 주소입니다.
- (선택 사항) 분산 추적이 설치 및 구성되어 있습니다.

절차

1. **Kiali** 콘솔을 시작합니다.

2. **OpenShift**로 로그인 을 클릭합니다.

   Kiali 개요 페이지에는 볼 권한이 있는 메시에 추가된 네임스페이스가 표시됩니다.

3. 애플리케이션,워크로드 또는 서비스를 클릭합니다.

4. 애플리케이션,워크로드, 또는 서비스 페이지의 네임스페이스 메뉴에서 프로젝트를 선택합니다.

5. 필요한 경우 필터를 사용하여 확인하려는 로그의 애플리케이션, 워크로드 또는 서비스를 찾습니다. 이름을 클릭합니다.

6. 애플리케이션 세부 정보,워크로드 세부 정보 또는 서비스 세부 정보 페이지에서 인바운드 지표 또는 아웃 바운드 지표 탭을 클릭하여 지표를 확인합니다.

1.15.4. 분산 추적

분산 추적은 애플리케이션에서 서비스 호출의 경로를 추적하여 애플리케이션에서 개별 서비스의 성능을 추적하는 프로세스입니다. 사용자가 애플리케이션에서 작업을 수행할 때마다 여러 서비스가 상호 작용해야 응답을 생성할 수 있는 요청이 실행됩니다. 이 요청의 경로는 분산 트랜잭션이라고 합니다.

Red Hat OpenShift Service Mesh는 **Red Hat OpenShift distributed tracing** 플랫폼을 사용하여 개발자가 마이크로 서비스 애플리케이션에서 호출 흐름을 볼 수 있도록 합니다.

1.15.4.1. 기존 분산 추적 인스턴스 연결

OpenShift Container Platform에 기존 **Red Hat OpenShift distributed tracing Platform(Jaeger)** 인스턴스가 이미 있는 경우 분산 추적 플랫폼에 해당 인스턴스를 사용하도록 **ServiceMeshControlPlane** 리소스를 구성할 수 있습니다.

사전 요구 사항

- Red Hat OpenShift distributed tracing 플랫폼 인스턴스가 설치 및 구성되어 있습니다.

절차

1. OpenShift Container Platform 웹 콘솔에서 Operator → 설치된 Operator를 클릭합니다.

2. 프로젝트 메뉴를 클릭하고 Service Mesh Control Plane을 설치한 프로젝트(예: istio-system)를 선택합니다.

3. Red Hat OpenShift Service Mesh Operator를 클릭합니다. Istio Service Mesh Control Plane 열에서 ServiceMeshControlPlane 리소스의 이름을 클릭합니다. (예: basic)

4. 분산 추적 플랫폼(Jaeger) 인스턴스의 이름을 ServiceMeshControlPlane 에 추가합니다.

   a. YAML 탭을 클릭합니다.

   b. ServiceMeshControlPlane 리소스의 spec.addons.jaeger.name 에 분산 추적 플랫폼(Jaeger) 인스턴스의 이름을 추가합니다. 다음 예에서 distr-tracing-production 은 분산 추적 플랫폼(Jaeger) 인스턴스의 이름입니다.

   분산 추적 구성 예

```
spec:
  addons:
    jaeger:
      name: distr-tracing-production
```

   c. 저장을 클릭합니다.

5. 다시 로드를 클릭하여 ServiceMeshControlPlane 리소스가 올바르게 구성되었는지 확인합니다.

1.15.4.2. 샘플링 속도 조정

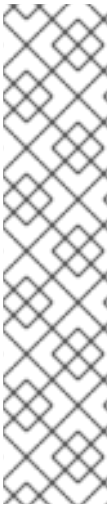
추적은 서비스 메시의 서비스 간 실행 경로입니다. 추적은 하나 이상의 범위로 구성됩니다. 범위는 이름, 시작 시간 및 기간이 있는 논리적 작업 단위입니다. 샘플링 속도는 추적이 유지되는 빈도를 결정합니다.

Envoy 프록시 샘플링 속도는 기본적으로 서비스 메시의 추적의 100%를 샘플링하도록 설정됩니다. 샘플링 속도가 높으면 클러스터 리소스와 성능이 소모되지만 문제를 디버깅할 때 유용합니다. 프로덕션에 Red Hat OpenShift Service Mesh를 배포하기 전에 값을 더 적은 비율의 추적으로 설정합니다. 예를 들어 spec.tracing.sampling 을 100 으로 설정하여 추적의 1%를 샘플링합니다.

Envoy 프록시 샘플링 속도를 0.01% 증분을 나타내는 스케일링된 정수로 구성합니다.

기본 설치에서 spec.tracing.sampling은 추적의 100%를 샘플링하는 10000으로 설정됩니다. 예를 들어 다음과 같습니다.

- 값을 10으로 설정하면 추적의 0.1%를 샘플링합니다.
- 값을 500으로 설정하면 추적의 5%가 샘플링됩니다.



참고

Envoy 프록시 샘플링 속도는 서비스 메시에서 사용할 수 있는 애플리케이션에 적용되며 Envoy 프록시를 사용합니다. 이 샘플링 비율은 Envoy 프록시가 수집하고 추적하는 데이터의 양을 결정합니다.

Jaeger 원격 샘플링 속도는 서비스 메시 외부에 있는 애플리케이션에 적용되며 데이터베이스와 같은 Envoy 프록시를 사용하지 않습니다. 이 샘플링 속도는 분산 추적 시스템에서 수집하고 저장하는 데이터의 양을 결정합니다. 자세한 내용은 분산 추적 구성 옵션을 참조하십시오.

절차

1. OpenShift Container Platform 웹 콘솔에서 Operator → 설치된 Operator를 클릭합니다.
2. 프로젝트 메뉴를 클릭하고 컨트롤 플레인을 설치한 프로젝트(예: istio-system)를 선택합니다.

3. Red Hat OpenShift Service Mesh Operator를 클릭합니다. Istio Service Mesh Control Plane 열에서 ServiceMeshControlPlane 리소스의 이름을 클릭합니다. (예: basic)

4. 샘플링 속도를 조정하려면 spec.tracing.sampling에 대해 다른 값을 설정합니다.

   a. YAML 탭을 클릭합니다.

   b. ServiceMeshControlPlane 리소스에서 spec.tracing.sampling의 값을 설정합니다. 다음 예에서는 100으로 설정합니다.

   Jaeger 샘플링 예

```
spec:
  tracing:
    sampling: 100
```

   c. 저장을 클릭합니다.

5. 다시 로드를 클릭하여 ServiceMeshControlPlane 리소스가 올바르게 구성되었는지 확인합니다.

1.15.5. Jaeger 콘솔에 액세스

Jaeger 콘솔에 액세스하려면 Red Hat OpenShift Service Mesh가 설치되어 있어야 합니다. Red Hat OpenShift distributed tracing Platform (Jaeger)이 설치되어 구성되어 있어야 합니다.

설치 프로세스는 Jaeger 콘솔에 액세스하기 위한 경로를 생성합니다.

Jaeger 콘솔의 URL을 알고 있는 경우 직접 액세스할 수 있습니다. URL을 모르는 경우 다음 지침을 사용하십시오.

OpenShift 콘솔의 절차

1. **OpenShift Container Platform** 웹 콘솔에 **cluster-admin** 권한이 있는 사용자로 로그인합니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 계정이 있어야 합니다.

2. 네트워킹 → 경로로 이동합니다.

3. 경로 페이지의 네임스페이스 메뉴에서 **Service Mesh Control Plane** 프로젝트(예: **istio-system**)를 선택합니다.

   위치 열은 각 경로에 대한 연결된 주소를 표시합니다.

4. 필요한 경우 필터를 사용하여 **jaeger** 경로를 찾습니다. 경로 위치를 클릭하여 콘솔을 시작합니다.

5. **OpenShift**로 로그인 을 클릭합니다.

Kiali 콘솔의 절차

1. **Kiali** 콘솔을 시작합니다.

2. 왼쪽 탐색 창에서 **Distributed Tracing** 을 클릭합니다.

3. **OpenShift**로 로그인 을 클릭합니다.

CLI의 프로세스

1. **cluster-admin** 역할의 사용자로 **OpenShift Container Platform CLI**에 로그인합니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 계정이 있어야 합니다.

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

2.

명령줄을 사용하여 경로의 세부 정보를 쿼리하려면 다음 명령을 입력합니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 네임스페이스입니다.

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

3. 브라우저를 시작하고 **https://<JAEGER_URL >**로 이동합니다. 여기서 **< JAEGER_URL >**은 이전 단계에서 검색한 경로입니다.

4. **OpenShift Container Platform** 콘솔에 액세스하는 데 사용하는 것과 동일한 사용자 이름 및 암호를 사용하여 로그인합니다.

5. 서비스 메시에 서비스를 추가하고 생성된 추적이 있는 경우 필터를 사용하고 추적 찾기 버튼을 사용하여 추적 데이터를 검색할 수 있습니다.

콘솔 설치의 유효성을 확인하는 경우 표시할 추적 데이터가 없습니다.

Jaeger 구성에 대한 자세한 내용은 분산 추적 설명서를 참조하십시오.

1.15.6. Grafana 콘솔에 액세스

Grafana는 서비스 메시 지표를 보고 쿼리하고 분석하는 데 사용할 수 있는 분석 도구입니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 네임스페이스입니다. **Grafana**에 액세스하려면 다음을 수행합니다.

절차

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.

2. 프로젝트 메뉴를 클릭하고 **Service Mesh Control Plane**을 설치한 프로젝트(예: **istio-system**)를 선택합니다.

3. 경로를 클릭합니다.

4. **Grafana** 행의 위치 열에서 링크를 클릭합니다.

5. **OpenShift Container Platform** 인증 정보를 사용하여 **Grafana** 콘솔에 로그인합니다.

1.15.7. Prometheus 콘솔에 액세스

Prometheus는 마이크로 서비스에 대한 다차원 데이터를 수집하는 데 사용할 수 있는 모니터링 및 경고 툴입니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 네임스페이스입니다.

절차

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. 프로젝트 메뉴를 클릭하고 **Service Mesh Control Plane**을 설치한 프로젝트(예: **istio-system**)를 선택합니다.
3. 경로를 클릭합니다.
4. **Prometheus** 행의 위치 열에서 링크를 클릭합니다.
5. **OpenShift Container Platform** 인증 정보를 사용하여 **Prometheus** 콘솔에 로그인합니다.

1.15.8. 사용자 워크로드 모니터링과 통합

기본적으로 **Red Hat OpenShift Service Mesh(OSSM)**는 메시에서 메트릭을 수집하기 위한 **Prometheus** 전용 인스턴스와 함께 **SMCP(Service Mesh Control Plane)**를 설치합니다. 그러나 프로덕션 시스템에는 사용자 정의 프로젝트에 대한 **OpenShift Container Platform** 모니터링과 같은 고급 모니터링 시스템이 필요합니다.

다음 단계에서는 서비스 메시를 사용자 워크로드 모니터링과 통합하는 방법을 보여줍니다.

사전 요구 사항

- 사용자 워크로드 모니터링이 활성화되어 있습니다.

- Red Hat OpenShift Service Mesh Operator 2.4가 설치되어 있습니다.
- Kiali Operator 1.65가 설치되어 있습니다.

절차

1. 다음 명령을 실행하여 Kiali의 Thanos에 대한 토큰을 생성합니다.

   a. 다음 명령을 실행하여 SECRET 환경 변수를 설정합니다.

   ```
   $ SECRET=`oc get secret -n openshift-user-workload-monitoring |
    grep  prometheus-user-workload-token | head -n 1 | awk '{print $1 }'`
   ```

   b. 다음 명령을 실행하여 TOKEN 환경 변수를 설정합니다.

   ```
   $ TOKEN=`oc get secret $SECRET -n openshift-user-workload-monitoring -o
   jsonpath='{.data.token}' | base64 -d`
   ```

   c. 다음 명령을 실행하여 Kiali의 Thanos에 대한 토큰을 생성합니다.

   ```
   $ oc create secret generic thanos-querier-web-token -n istio-system --from-
   literal=token=$TOKEN
   ```

2. user-workload 모니터링을 위해 Kiali를 구성합니다.

```
apiVersion: kiali.io/v1alpha1
kind: Kiali
metadata:
 name: kiali-user-workload-monitoring
 namespace: istio-system
spec:
 external_services:
  istio:
   url_service_version: 'http://istiod-basic.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
```

3. 외부 Prometheus에 대한 SMCP를 구성합니다.

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  addons:
    prometheus:
      enabled: false 1
    grafana:
      enabled: false 2
    kiali:
      name: kiali-user-workload-monitoring
  meshConfig:
    extensionProviders:
    - name: prometheus
      prometheus: {}
```

1 OSSM에서 제공하는 기본 Prometheus 인스턴스를 비활성화합니다.

2

4. 모니터링 네임스페이스에서 수신 트래픽을 허용하도록 사용자 정의 네트워크 정책을 적용합니다.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: user-workload-access
  namespace: info 1
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: monitoring
  podSelector: {}
  policyTypes:
  - Ingress
```

1

사용자 정의 네트워크 정책은 모든 네임스페이스에 적용해야 합니다.

5. Istio 프록시에서 트래픽 지표를 활성화하려면 Telemetry 오브젝트를 적용합니다.

```
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: enable-prometheus-metrics
  namespace: istio-system 1
spec:
  selector: 2
    matchLabels:
      app: info
  metrics:
  - providers:
    - name: prometheus
```

1 컨트롤 플레인 네임스페이스에서 생성된 Telemetry 오브젝트는 메시의 모든 워크로드에 적용됩니다. Telemetry를 하나의 네임스페이스에 적용하려면 대상 네임스페이스에 오브젝트를 생성합니다.

2

6. ServiceMonitor 오브젝트를 적용하여 Istio 컨트롤 플레인을 모니터링합니다.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: istiod-monitor
  namespace: istio-system 1
spec:
  targetLabels:
  - app
  selector:
    matchLabels:
      istio: pilot
  endpoints:
  - port: http-monitoring
    interval: 30s
    relabelings:
```

```
  - action: replace
    replacement: "basic-istio-system" 2
    targetLabel: mesh_id
```

1

OpenShift Container Platform 모니터링은 ServiceMonitor 및 PodMonitor 오브젝트의 namespaceSelector 사양을 무시하므로 컨트롤 플레인 네임스페이스를 포함하여 모든 메시 네임스페이스에 PodMonitor 오브젝트를 적용해야 합니다.

2

"basic-istio-system" 문자열은 SMCP 이름과 해당 네임스페이스의 조합이지만 클러스터의 사용자 워크로드 모니터링을 사용하는 모든 메시에 고유한 레이블을 사용할 수 있습니다. 2단계에서 구성된 Kiali 리소스의 spec.prometheus.query_scope 는 이 값과 일치해야 합니다.

참고

user-workload 모니터링을 사용하는 메시가 하나뿐인 경우 Kiali 리소스의 mesh_id 재지정 및 spec.prometheus.query_scope 필드 모두 선택 사항입니다 (메쉬_id 레이블이 제거된 경우 여기에 제공된 query_scope 필드를 제거해야 함).

user-workload 모니터링을 사용하여 여러 메시가 있을 수 있는 경우 Kiali 리소스의 mesh_id 레이블 재지정과 spec.prometheus.query_scope 필드 모두 Kiali 리소스의 메트릭만 표시해야 합니다. Kiali가 배포되지 않는 경우 mesh_id 재레이블링을 계속 적용하는 것이 좋습니다. 따라서 다른 메시와 메트릭을 구분할 수 있습니다.

7. PodMonitor 오브젝트를 적용하여 Istio 프록시에서 지표를 수집합니다.

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: istio-proxies-monitor
  namespace: istio-system 1
spec:
  selector:
    matchExpressions:
    - key: istio-prometheus-ignore
      operator: DoesNotExist
  podMetricsEndpoints:
  - path: /stats/prometheus
    interval: 30s
    relabelings:
    - action: keep
```

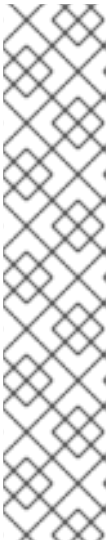
```
      sourceLabels: [__meta_kubernetes_pod_container_name]
      regex: "istio-proxy"
    - action: keep
      sourceLabels:
[__meta_kubernetes_pod_annotationpresent_prometheus_io_scrape]
    - action: replace
      regex: (\d+);(([A-Fa-f0-9]{1,4}::?){1,7}[A-Fa-f0-9]{1,4})
      replacement: '[$2]:$1'
      sourceLabels: [__meta_kubernetes_pod_annotation_prometheus_io_port,
      __meta_kubernetes_pod_ip]
      targetLabel: __address__
    - action: replace
      regex: (\d+);((([0-9]+?)(\.|$)){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" (2)
      targetLabel: mesh_id
```

(1)

OpenShift Container Platform 모니터링은 **ServiceMonitor** 및 **PodMonitor** 오브젝트의 **namespaceSelector** 사양을 무시하므로 컨트롤 플레인 네임스페이스를 포함하여 모든 메시 네임스페이스에 **PodMonitor** 오브젝트를 적용해야 합니다.

(2)

"basic-istio-system" 문자열은 **SMCP** 이름과 해당 네임스페이스의 조합이지만 클러스터의 사용자 워크로드 모니터링을 사용하는 모든 메시에 고유한 레이블을 사용할 수 있습니다. 2단계에서 구성된 **Kiali** 리소스의 **spec.prometheus.query_scope** 는 이 값과 일치해야 합니다.



참고

user-workload 모니터링을 사용하는 메시가 하나뿐인 경우 **Kiali** 리소스의 **mesh_id** 재지정 및 **spec.prometheus.query_scope** 필드 모두 선택 사항입니다 (메쉬_id 레이블이 제거된 경우 여기에 제공된 **query_scope** 필드를 제거해야 함).

user-workload 모니터링을 사용하여 여러 메시가 있을 수 있는 경우 **Kiali** 리소스의 **mesh_id** 레이블 재지정과 **spec.prometheus.query_scope** 필드 모두 **Kiali** 리소스의 메트릭만 표시해야 합니다. **Kiali**가 배포되지 않는 경우 **mesh_id** 재레이블링을 계속 적용하는 것이 좋습니다. 따라서 다른 메시와 메트릭을 구분할 수 있습니다.

8. **OpenShift Container Platform** 웹 콘솔을 열고 메트릭이 표시되는지 확인합니다.

1.15.9. 추가 리소스

- 사용자 정의 프로젝트 모니터링 활성화

1.16. 성능 및 확장

기본 **ServiceMeshControlPlane** 설정은 프로덕션용이 아닙니다. 이 설정은 리소스가 매우 제한된 환경인 기본 **OpenShift Container Platform** 설치 시 성공적으로 설치되도록 설계되었습니다. **SMCP** 설치에 성공했는지 확인한 후 **SMCP** 내에 정의된 설정을 사용자 환경에 맞게 수정해야 합니다.

1.16.1. 컴퓨팅 리소스에 제한 설정

기본적으로 **spec.proxy** 설정은 **cpu: 10m** 및 **memory: 128M**입니다. **Pilot**을 사용하는 경우 **spec.runtime.components.pilot**의 기본값이 동일합니다.

다음 예의 설정은 초당 **1,000**개의 서비스 및 **1,000**개의 요청을 기반으로 합니다. **ServiceMeshControlPlane**에서 **cpu** 및 **memory** 값을 변경할 수 있습니다.

절차

1. **OpenShift Container Platform** 웹 콘솔에서 **Operator** → 설치된 **Operator**를 클릭합니다.

2. 프로젝트 메뉴를 클릭하고 **Service Mesh Control Plane**을 설치한 프로젝트(예: **istio-**

system)를 선택합니다.

3. Red Hat OpenShift Service Mesh Operator를 클릭합니다. Istio Service Mesh Control Plane 열에서 ServiceMeshControlPlane의 이름을 클릭합니다(예: basic).

4. 독립형 Jaeger 인스턴스의 이름을 ServiceMeshControlPlane에 추가합니다.

   a. YAML 탭을 클릭합니다.

   b. ServiceMeshControlPlane 리소스의 spec.proxy.runtime.container.resources.requests.cpu 및 spec.proxy.runtime.container.resources.requests.memory 값을 설정합니다.

   버전 2.4 ServiceMeshControlPlane 예

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  version: v2.4
  proxy:
    runtime:
      container:
        resources:
          requests:
            cpu: 600m
            memory: 50Mi
          limits: {}

  runtime:
    components:
      pilot:
        container:
          resources:
            requests:
              cpu: 1000m
              memory: 1.6Gi
            limits: {}
```

c. 저장을 클릭합니다.

5. 다시 로드를 클릭하여 **ServiceMeshControlPlane** 리소스가 올바르게 구성되었는지 확인합니다.

1.16.2. 로드 테스트 결과

업스트림 **Istio** 커뮤니티 로드 테스트 메시는 초당 **70,000**개의 메시 전체 요청이 있는 **1000**개의 서비스와 **2000**개의 사이드카로 구성됩니다. **Istio 1.12.3**을 사용하여 테스트를 실행하면 다음 결과가 생성됩니다.

- **Envoy** 프록시는 프록시를 통과하는 초당 **1000**개 요청마다 **0.35 vCPU** 및 **40MB** 메모리 를 사용합니다.
- **Istiod**는 **1**개의 **vCPU** 및 **1.5GB** 메모리를 사용합니다.
- **Envoy** 프록시는 **2.65ms** 를 **90**번째 백분위 대기 시간에 추가합니다.
- 레거시 **istio-telemetry** 서비스(기본적으로 **Service Mesh 2.0**에서 비활성화됨)는 **Mixer**를 사용하는 배포에 대해 초당 **1,000** 개의 메시 전체 요청마다 **0.6 vCPU**를 사용합니다. 데이터 플레인 구성 요소인 **Envoy** 프록시는 시스템을 통과하는 데이터를 처리합니다. **Service Mesh Control Plane** 구성 요소 **Istiod**는 데이터 플레인을 구성합니다. 데이터 플레인과 컨트롤 플레인에는 별도의 성능 문제가 있습니다.

1.16.2.1. 서비스 메시 컨트롤 플레인 성능

Istiod는 사용자가 승인한 구성 파일 및 시스템의 현재 상태를 기반으로 사이드카 프록시를 구성합니다. **Kubernetes** 환경에서 **CRD(Custom Resource Definitions)**와 배포는 시스템의 구성 및 상태를 구성합니다. 게이트웨이 및 가상 서비스와 같은 **Istio** 구성 오브젝트는 사용자 인증된 구성을 제공합니다. 프록시에 대한 구성을 생성하기 위해 **Istiod**는 **Kubernetes** 환경과 사용자 인증된 구성에서 결합된 구성 및 시스템 상태를 처리합니다.

서비스 메시 컨트롤 플레인은 수천 개의 서비스를 지원하며 유사한 수의 사용자 인증된 가상 서비스 및 기타 구성 오브젝트를 사용하여 수천 개의 **Pod**에 분산됩니다. **Istiod**의 **CPU** 및 메모리 요구 사항은 구성 수와 가능한 시스템 상태에 따라 확장됩니다. **CPU** 사용량은 다음과 같은 요인에 따라 확장됩니다.

- 배포 변경 비율.
- 구성 변경 비율.
- Istiod에 연결된 프록시 수.

그러나 이 부분은 기본적으로 수평 확장할 수 있습니다.

1.16.2.2. 데이터 플레인 성능

데이터 플레인 성능은 여러 요인에 따라 달라집니다. 예를 들면 다음과 같습니다.

- 클라이언트 연결 수
- 대상 요청 속도
- 요청 크기 및 응답 크기
- 프록시 작업자 스레드 수
- 프로토콜
- CPU 코어 수
- 프록시 필터, 특히 telemetry v2 관련 필터의 수 및 유형입니다.

대기 시간, 처리량, 프록시 CPU 및 메모리 사용은 이러한 요인의 기능으로 측정됩니다.

1.16.2.2.1. CPU 및 메모리 사용

사이드카 프록시는 데이터 경로에서 추가 작업을 수행하므로 CPU와 메모리를 사용합니다. Istio 1.12.3부터 프록시는 초당 1000개 요청마다 약 0.5 vCPU를 사용합니다.

프록시의 메모리 사용은 프록시가 보유하고 있는 총 구성 상태에 따라 달라집니다. 다수의 리스너, 클러스터 및 경로를 통해 메모리 사용량을 늘릴 수 있습니다.

프록시는 일반적으로 통과된 데이터를 버퍼링하지 않기 때문에 요청 속도는 메모리 소비에 영향을 미치지 않습니다.

1.16.2.2.2. 추가 대기 시간

Istio가 데이터 경로에 사이드카 프록시를 삽입하기 때문에 대기 시간이 중요합니다. Istio는 인증 필터, telemetry 필터 및 메타데이터 교환 필터를 프록시에 추가합니다. 모든 추가 필터는 프록시 내부의 경로 길이를 추가하고 대기 시간에 영향을 미칩니다.

Envoy 프록시는 응답이 클라이언트에 전송된 후 원시 telemetry 데이터를 수집합니다. 요청을 위해 원시 Telemetry를 수집하는 데 소요되는 시간은 해당 요청을 완료하는 데 걸리는 총 시간에 기여하지 않습니다. 그러나 작업자가 요청을 처리하느라 바쁘기 때문에 작업자는 즉시 다음 요청 처리를 시작하지 않습니다. 이 프로세스는 다음 요청의 대기열 대기 시간에 추가되며 평균 및 테일 대기 시간에 영향을 미칩니다. 실제 정확한 대기 시간은 트래픽 패턴에 따라 달라집니다.

메시 내부에서 요청은 클라이언트 측 프록시를 통과한 다음, 서버 측 프록시를 통과합니다. Istio 1.12.3(즉, telemetry v2가 있는 Istio)의 기본 구성에서 두 프록시는 기준 데이터 플레인 대기 시간을 통해 각각 약 1.7ms 및 2.7ms를 90번째 및 99번째 백분위 대기 시간에 추가합니다.

1.17. 프로덕션을 위한 서비스 메시 구성

기본 설치에서 프로덕션으로 이동할 준비가 되면 프로덕션 요구 사항을 충족하도록 컨트롤 플레인, 추적 및 보안 인증서를 구성해야 합니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh를 설치 및 구성합니다.
- 스테이징 환경에서 구성을 테스트합니다.

1.17.1. 프로덕션을 위한 ServiceMeshControlPlane 리소스 구성

Service Mesh를 테스트하기 위해 기본 ServiceMeshControlPlane 리소스를 설치한 경우 프로덕션에서 Red Hat OpenShift Service Mesh를 사용하기 전에 프로덕션 사양으로 구성해야 합니다.

기존 ServiceMeshControlPlane 리소스의 metadata.name 필드를 변경할 수 없습니다. 프로덕션 배포의 경우 기본 템플릿을 사용자 지정해야 합니다.

절차

1. 프로덕션을 위해 분산 추적 플랫폼(Jaeger)을 구성합니다.

a. Elasticsearch에 spec.addons.jaeger.install.storage.type를 설정하여 production 배포 전략을 사용하기 위해 ServiceMeshControlPlane 리소스를 편집하고 install에서 추가 구성 옵션을 지정합니다. Jaeger 인스턴스를 생성 및 구성하고 spec.addons.jaeger.name을 Jaeger 인스턴스의 이름으로 설정할 수 있습니다.

Elasticsearch를 포함한 기본 Jaeger 매개변수

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.4
  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.

프로덕션을 위한 샘플링 속도를 구성합니다. 자세한 내용은 성능 및 확장성 섹션을 참조하십시오.

2. 외부 인증 기관에서 보안 인증서를 설치하여 보안 인증서가 프로덕션에 준비되었는지 확인합니다. 자세한 내용은 보안 섹션을 참조하십시오.

3. 결과를 확인합니다. 다음 명령을 입력하여 **ServiceMeshControlPlane** 리소스가 올바르게 업데이트되었는지 확인합니다. 이 예에서 **basic**은 **ServiceMeshControlPlane** 리소스의 이름입니다.

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

1.17.2. 추가 리소스

- 성능을 위해 서비스 메시 조정에 대한 자세한 내용은 성능 및 확장성을 참조하십시오.

1.18. 서비스 메시 연결

*페더레이션*은 별도의 관리 도메인에서 관리하는 별도의 메시 간에 서비스 및 워크로드를 공유할 수 있는 배포 모델입니다.

1.18.1. 페더레이션 개요

페더레이션은 별도의 메시 간에 서비스를 연결할 수 있는 일련의 기능이며 여러 별도의 관리 도메인에서 인증, 권한 부여 및 트래픽 관리와 같은 서비스 메시 기능을 사용할 수 있습니다.

페더레이션 메시를 구현하면 여러 **OpenShift** 클러스터에서 실행되는 단일 서비스 메시를 실행, 관리 및 관찰할 수 있습니다. **Red Hat OpenShift Service Mesh** 페더레이션은 메시 간 *최소* 신뢰를 가정하는 **Service Mesh**의 다중 클러스터 구현에 대한 의견 접근 방식을 취합니다.

서비스 메시 페더레이션에서는 각 메시가 개별적으로 관리되고 자체 관리자가 있는 것으로 가정합니다. 기본 동작은 통신이 허용되지 않으며 메시 간에 정보가 공유되지 않는다는 것입니다. 메시 간 정보 공유는 명시적 옵트인(**opt-in**) 기반입니다. 공유를 위해 구성되지 않은 경우 페더레이션 메시에서 공유되지 않습니다. 인증서 생성, 메트릭 및 추적 수집과 같은 지원 기능은 각 메시에서 로컬로 유지됩니다.

각 서비스 메시에서 **ServiceMeshControlPlane** 을 구성하여 구체적으로 페더레이션을 위한 수신 및 송신 게이트웨이를 생성하고 메시에 대한 신뢰 도메인을 지정합니다.

페더레이션은 또한 추가 연합 파일을 생성하는 것과 관련이 있습니다. 다음 리소스는 둘 이상의 메시 간 연합을 구성하는 데 사용됩니다.

- **ServiceMeshPeriod** 리소스 는 한 쌍의 서비스 메시 간 페더레이션을 선언합니다.
- **ExportedServiceSet** 리소스는 피어 메시에서 하나 이상의 서비스를 사용할 수 있음을 선언합니다.
- **ImportedServiceSet** 리소스는 피어 메시에서 내보낸 서비스를 메시로 가져올 서비스를 선언합니다.

1.18.2. 페더레이션 기능

메시에 가입하기 위한 **Red Hat OpenShift Service Mesh** 통합 접근법의 기능은 다음과 같습니다.

- 각 메시에 대한 공통 루트 인증서를 지원합니다.
- 각 메시에 대해 다양한 루트 인증서를 지원합니다.
- 메시 관리자는 **Federated** 메시 외부의 메시에 대한 인증서 체인, 서비스 검색 끝점, 신뢰 도메인 등을 수동으로 구성해야 합니다.
- 메시 간에 공유할 서비스만 내보내거나 가져옵니다.
  - 기본적으로 배포된 워크로드에 대한 정보는 통합의 다른 메시와 공유하지 않습니다. 서비스를 내보내 다른 메시에 표시하고 자체 메시 외부의 워크로드의 요청을 허용할 수 있습니다.
  - 내보낸 서비스를 다른 메시로 가져올 수 있으므로 해당 메시의 워크로드가 가져온 서비스에 요청을 보낼 수 있습니다.
- 항상 메시 간 통신을 암호화합니다.

- 로컬로 배포된 워크로드 및 다른 메시에 배포된 워크로드 간 로드 밸런싱 구성을 지원합니다.

메시가 다른 메시에 결합되면 다음을 수행할 수 있습니다.

- 연합된 메시에 대한 신뢰 세부 정보를 제공합니다.
- 연합된 메시에 대한 신뢰 세부 정보를 검색합니다.
- 내보낸 자체 서비스에 대한 연합 메시에 정보를 제공합니다.
- 연합 메시에서 내보낸 서비스에 대한 정보를 검색합니다.

1.18.3. 페더레이션 보안

Red Hat OpenShift Service Mesh 페더레이션은 메시 간 최소 신뢰를 가정하는 **Service Mesh**의 다중 클러스터 구현에 대한 의견 접근 방식을 취합니다. 데이터 보안은 연합 기능의 일부로 구축됩니다.

- 각 메시는 고유한 관리를 통해 고유한 테넌트로 간주됩니다.
- 페더레이션에서 각 메시에 대해 고유한 신뢰 도메인을 생성합니다.
- 연합 메시 간 트래픽은 **mTLS(mutual Transport Layer Security)**을 사용하여 자동으로 암호화됩니다.
- **Kiali** 그래프는 가져온 메시 및 서비스만 표시합니다. 메시로 가져오지 않은 다른 메시 또는 서비스는 볼 수 없습니다.

1.18.4. 페더레이션 제한

메시에 가입하기 위한 **Red Hat OpenShift Service Mesh** 통합 접근법에는 다음과 같은 제한 사항이 있습니다.

- OpenShift Dedicated에서는 메시의 페더레이션이 지원되지 않습니다.

1.18.5. 페더레이션 사전 요구 사항

메시에 가입하기 위한 Red Hat OpenShift Service Mesh 통합 접근법에는 다음과 같은 사전 요구 사항이 있습니다.

- 두 개 이상의 OpenShift Container Platform 4.6 이상의 클러스터.
- 페더레이션은 Red Hat OpenShift Service Mesh 2.1 이상에서 도입되었습니다. 페더레이션할 각 메시에 Red Hat OpenShift Service Mesh 2.1 이상의 Operator가 설치되어 있어야 합니다.
- 연결하려는 각 메시에 버전 2.1 이상 ServiceMeshControlPlane 이 배포되어야 합니다.
- 원시 TLS 트래픽을 지원하려면 페더레이션 게이트웨이와 연결된 서비스를 지원하는 로드 밸런서를 구성해야 합니다. 페더레이션 트래픽은 서비스 트래픽을 위한 검색 및 원시 암호화된 TCP 를 위한 HTTPS로 구성됩니다.
- 다른 메시에 노출하려는 서비스를 내보내서 가져올 수 있기 전에 배포해야 합니다. 그러나 이것은 엄격한 요구 사항이 아닙니다. export/import에 대해 아직 존재하지 않는 서비스 이름을 지정할 수 있습니다. ExportedServiceSet 및 ImportedServiceSet 에서 이름이 지정된 서비스를 배포하면 내보내기/가져오기에 자동으로 사용할 수 있습니다.

1.18.6. 메시 통합 계획

메시 연합 구성을 시작하기 전에 일정 시간이 걸리므로 구현을 계획해야 합니다.

- 얼마나 많은 메시가 연합에 참여할 계획입니까? 제한된 수의 메시 (두 개 또는 세 개)로 시작해야 할 수도 있습니다.
- 각 메시에 사용할 이름 지정 규칙은 무엇입니까? 미리 정의된 이름 지정 규칙이 있으면 구성 및 문제 해결에 도움이 됩니다. 이 문서의 예제에서는 각 메시에 대해 다른 색상을 사용합니다. 각 메시와 다음 페더레이션 리소스를 누가 소유하고 관리하는지 결정하는 데 도움이 되는 이름 지정 규칙을 결정해야 합니다.

- 클러스터 이름
- 클러스터 네트워크 이름
- 메시 이름 및 네임스페이스
- 페더레이션 수신 게이트웨이
- 페더레이션 송신 게이트웨이
- 보안 신뢰 도메인

참고

페더레이션의 각 메시에는 고유한 신뢰 도메인이 있어야 합니다.

- 각 메시의 어떤 서비스가 연합 메시로 내보낼 계획입니까? 각 서비스를 개별적으로 내보내거나 레이블을 지정하거나 와일드카드를 사용할 수 있습니다.
  - 서비스 네임스페이스에 별칭을 사용하고 싶으신가요?
  - 내보낸 서비스에 별칭을 사용하고 싶으신가요?
- 각 메시에서 가져올 계획인 내보낸 서비스는 무엇입니까? 각 메시는 필요한 서비스만 가져옵니다.
  - 가져온 서비스에 별칭을 사용하고 싶으신가요?

1.18.7. 클러스터 전체에서 메시 통합

다른 클러스터에서 실행 중인 **OpenShift Service Mesh**의 인스턴스 하나를 연결하기 위해 동일한 클

러스터에 배포된 두 메시를 연결할 때와 매우 다르지 않습니다. 그러나 하나의 메시의 수신 게이트웨이는 다른 메시에서 연결할 수 있어야합니다. 클러스터가 이러한 유형의 서비스를 지원하는 경우 게이트웨이 서비스를 LoadBalancer 서비스로 구성하는 것입니다.

OSI 모델의 계층4에서 작동하는 로드 밸런서를 통해 서비스를 노출해야 합니다.

1.18.7.1. 베어 메탈에서 실행되는 클러스터에 페더레이션 인그레스 노출

클러스터가 베어 메탈에서 실행되고 LoadBalancer 서비스를 완전히 지원하는 경우 수신 게이트웨이 서비스 오브젝트의 .status.loadBalancer.ingress.ip 필드에 있는 IP 주소는 ServiceMesh -02- 오브젝트의 .spec.remote.addresses 필드에 있는 항목 중 하나로 지정해야 합니다.

클러스터가 LoadBalancer 서비스를 지원하지 않는 경우 NodePort 서비스를 사용하면 다른 메시를 실행하는 클러스터에서 노드에 액세스할 수 있는 경우 옵션이 될 수 있습니다. ServiceMesh peer 개체에서 .spec.remote.addresses 필드에 노드의 IP 주소와 .spec.remote.discoveryPort 및 .spec.remote.servicePort 필드에 서비스의 노드 포트를 지정합니다.

1.18.7.2. IBM Power 및 IBM Z에서 실행되는 클러스터에 페더레이션 수신 노출

클러스터가 IBM Power 또는 IBM Z 인프라에서 실행되고 LoadBalancer 서비스를 완전히 지원하는 경우 수신 게이트웨이 서비스 오브젝트의 .status.loadBalancer.ingress.ip 필드에 있는 IP 주소는 ServiceMesh Period 오브젝트의 .spec.remote.addresses 필드에 있는 항목 중 하나로 지정해야 합니다.

클러스터가 LoadBalancer 서비스를 지원하지 않는 경우 NodePort 서비스를 사용하면 다른 메시를 실행하는 클러스터에서 노드에 액세스할 수 있는 경우 옵션이 될 수 있습니다. ServiceMesh peer 개체에서 .spec.remote.addresses 필드에 노드의 IP 주소와 .spec.remote.discoveryPort 및 .spec.remote.servicePort 필드에 서비스의 노드 포트를 지정합니다.

1.18.7.3. AWS(Amazon Web Services)에 페더레이션 수신 노출

기본적으로 AWS에서 실행되는 클러스터의 LoadBalancer 서비스는 L4 로드 밸런싱을 지원하지 않습니다. Red Hat OpenShift Service Mesh 통합이 올바르게 작동하려면 수신 게이트웨이 서비스에 다음 주석을 추가해야 합니다.

service.beta.kubernetes.io/aws-load-balancer-type: nlb

수신 게이트웨이 서비스 오브젝트의 .status.loadBalancer.ingress.hostname 필드에 있는 정규화된 도메인 이름은 ServiceMesh 피어 오브젝트의 .spec.remote.addresses 필드에 있는 항목 중 하나로 지

정해야 합니다.

1.18.7.4. Azure에서 페더레이션 인그레스 노출

Microsoft Azure에서 메시 페더레이션이 제대로 작동하기 위해 서비스 유형을 LoadBalancer 로 설정하는 것만으로 충분합니다.

수신 게이트웨이 서비스 오브젝트의 .status.loadBalancer.ingress.ip 필드에 있는 IP 주소는 ServiceMesh Period 오브젝트의 .spec.remote.addresses 필드에 있는 항목 중 하나로 지정해야 합니다.

1.18.7.5. GCP(Google Cloud Platform)에 페더레이션 수신 노출

Google Cloud Platform에서 서비스 유형을 LoadBalancer 로 설정하는 것만으로 메시 통합이 올바르게 작동하기에 충분합니다.

수신 게이트웨이 서비스 오브젝트의 .status.loadBalancer.ingress.ip 필드에 있는 IP 주소는 ServiceMesh Period 오브젝트의 .spec.remote.addresses 필드에 있는 항목 중 하나로 지정해야 합니다.

1.18.8. 페더레이션 구현 체크리스트

서비스 메시에는 다음 활동이 포함됩니다.

❑ 통합하려는 클러스터 간 네트워킹을 구성합니다.

❑ 원시 TLS 트래픽을 지원하기 위해 페더레이션 게이트웨이와 연결된 서비스를 지원하는 로드 밸런서를 구성합니다.

dpdk 각 클러스터에 Red Hat OpenShift Service Mesh 버전 2.1 이상 Operator를 설치합니다.

jenkinsfile 버전 2.1 이상 ServiceMeshControlPlane 을 각 클러스터에 배포합니다.

통합하려는 각 메시에 대해 SMCP를 구성합니다.

연결하려는 각 메시에 대해 페더레이션 송신 게이트웨이를 만듭니다.

연결하려는 각 메시에 대해 페더레이션 수신 게이트웨이를 만듭니다.

고유한 트러스트 도메인을 구성합니다.**Configure a unique trust domain.**

❑ 각 메시 쌍에 대해 **ServiceMesh peer** 리소스를 생성하여 두 개 이상의 메시를 통합합니다.

dpdk ExportedServiceSet 리소스를 생성하여 하나의 메시에서 피어 메시로 서비스를 사용할 수 있도록 하여 서비스를 내보냅니다.

dpdk는 메시 피어에서 공유하는 서비스를 가져오기 위해 **ImportedServiceSet** 리소스를 만들어 서비스를 가져옵니다.

1.18.9. 페더레이션을 위한 서비스 메시 컨트롤 플레인 구성

메시를 통합하기 전에 메시 통합을 위해 **ServiceMeshControlPlane** 을 구성해야 합니다. 연합의 멤버인 모든 메시가 동일하고 각 메시는 독립적으로 관리되므로 페더레이션에 참여할 *각* 메시마다 **SMCP**를 구성해야 합니다.

다음 예에서 **red-mesh** 의 관리자는 **green-mesh** 및 **blue-mesh** 와 함께 페더레이션에 대해 **SMCP**를 구성하고 있습니다.

red-mesh에 대한 샘플 **SMCP**

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: red-mesh
  namespace: red-mesh-system
spec:
  version: v2.4
  runtime:
    defaults:
      container:
        imagePullPolicy: Always
```

```
gateways:
  additionalEgress:
    egress-green-mesh:
      enabled: true
      requestedNetworkView:
      - green-network
      routerMode: sni-dnat
      service:
        metadata:
          labels:
            federation.maistra.io/egress-for: egress-green-mesh
        ports:
        - port: 15443
          name: tls
        - port: 8188
          name: http-discovery  #note HTTP here
    egress-blue-mesh:
      enabled: true
      requestedNetworkView:
      - blue-network
      routerMode: sni-dnat
      service:
        metadata:
          labels:
            federation.maistra.io/egress-for: egress-blue-mesh
        ports:
        - port: 15443
          name: tls
        - port: 8188
          name: http-discovery  #note HTTP here
  additionalIngress:
    ingress-green-mesh:
      enabled: true
      routerMode: sni-dnat
      service:
        type: LoadBalancer
        metadata:
          labels:
            federation.maistra.io/ingress-for: ingress-green-mesh
        ports:
        - port: 15443
          name: tls
        - port: 8188
          name: https-discovery  #note HTTPS here
    ingress-blue-mesh:
      enabled: true
      routerMode: sni-dnat
      service:
        type: LoadBalancer
        metadata:
          labels:
            federation.maistra.io/ingress-for: ingress-blue-mesh
        ports:
        - port: 15443
          name: tls
        - port: 8188
```

```
        name: https-discovery  #note HTTPS here
security:
  trust:
    domain: red-mesh.local
```

표 1.6. ServiceMeshControlPlane 페더레이션 구성 매개변수

매개변수	설명	값	기본값

`spec:`
`  cluster:`
`  name:`
	클러스터의 이름입니다. 클러스터 이름을 지정할 필요는 없지만 문제 해결에 유용합니다.	문자열	해당 없음

`spec:`
`  cluster:`
`  network:`
	클러스터 네트워크의 이름입니다. 네트워크 이름을 지정할 필요는 없지만 구성 및 문제 해결에 유용합니다.	문자열	해당 없음

1.18.9.1. 페더레이션 게이트웨이 이해

게이트웨이를 사용하여 메시에 대한 인바운드 및 아웃바운드 트래픽을 관리하여 메시에 들어가거나 나가려는 트래픽을 지정할 수 있습니다.

수신 및 송신 게이트웨이를 사용하여 서비스 메시(North-South 트래픽)를 입력하고 나가는 트래픽을 관리합니다. 연합 메시를 생성할 때 추가적인 수신/egress 게이트웨이를 생성하여 연합 메시 간 서비스 검색, 통합 메시 간 통신, 서비스 메시 간 트래픽 흐름(동부 트래픽)을 관리합니다.

메시 간 충돌 이름 지정을 방지하려면 각 메시에 대해 별도의 송신 및 수신 게이트웨이를 생성해야 합니다. 예를 들어 **red-mesh** 에는 **green-mesh** 및 **blue-mesh** 로 이동하는 트래픽에 대해 별도의 송신 게이트웨이가 있습니다.

표 1.7. 페더레이션 게이트웨이 매개변수

매개변수	설명	값	기본값
spec: gateways: additionalEgress: <egressName>:	페더레이션에서 *각* 메시 피어에 대한 추가 송신 게이트웨이를 정의합니다.		
spec: gateways: additionalEgress: <egressName>: enabled:	이 매개변수는 페더레이션 송신을 활성화 또는 비활성화합니다.	**true**/**false**	**true**
spec: gateways: additionalEgress: <egressName>: requestedNetworkView:	내보낸 서비스와 연결된 네트워크입니다.	메시의 SMCP에서 **spec.cluster.network** 값으로 설정합니다. 그렇지 않으면 <ServiceMeshpeer-name>-network를 사용합니다. 예를 들어, 해당 메시 **의 ServiceMesh** peer 리소스의 이름이 **west** 인 경우 네트워크 이름은 **west-network** 입니다.	
spec: gateways: additionalEgress: <egressName>: routerMode:	게이트웨이에서 사용할 라우터 모드입니다.	**sni-dnat**	

매개변수	설명	값	기본값
spec: gateways: additionalEgress: <egressName>: service: metadata: labels: federation.maistra.io/egress-for:	연결된 트래픽이 클러스터의 기본 시스템 게이트웨이를 통과하지 않도록 게이트웨이에 대한 고유 레이블을 지정합니다.		
spec: gateways: additionalEgress: <egressName>: service: ports:	**포트: 및 name:** 를 지정하는 데 사용되는 TLS 및 서비스 검색에 사용됩니다. 페더레이션 트래픽은 서비스 트래픽을 위해 원시 암호화된 TCP로 구성됩니다.	포트 **15443** 은 연합 내의 다른 메시에 TLS 서비스 요청을 보내는 데 필요합니다. 포트 **8188** 은 페더레이션의 다른 메시에 서비스 검색 요청을 보내는 데 필요합니다.	
spec: gateways: additionalIngress:	페더레이션에서 *각* 메시 피어에 대한 추가 수신 게이트웨이 게이트웨이를 정의합니다.		
spec: gateways: additionalIgress: <ingressName>: enabled:	이 매개변수는 페더레이션 인그레스를 활성화 또는 비활성화합니다.	**true/false**	**true**
spec: gateways: additionalIngress: <ingressName>: routerMode:	게이트웨이에서 사용할 라우터 모드입니다.	**sni-dnat**	

매개변수	설명	값	기본값
spec: gateways: additionalIngress: <ingressName>: service: type:	수신 게이트웨이 서비스는 OSI 모델의 계층 4에서 작동하고 공개적으로 사용 가능한 로드 밸런서를 통해 노출되어야 합니다.	**LoadBalancer**	
spec: gateways: additionalIngress: <ingressName>: service: type:	클러스터가 **LoadBalancer** 서비스를 지원하지 않으면 **NodePort** 서비스를 통해 수신 게이트웨이 서비스가 노출될 수 있습니다.	**NodePort**	
spec: gateways: additionalIngress: <ingressName>: service: metadata: labels: federation.maistra.io/ingress-for:	연결된 트래픽이 클러스터의 기본 시스템 게이트웨이를 통과하지 않도록 게이트웨이에 대한 고유 레이블을 지정합니다.		
spec: gateways: additionalIngress: <ingressName>: service: ports:	**포트: 및 name:** 를 지정하는 데 사용되는 TLS 및 서비스 검색에 사용됩니다. 페더레이션 트래픽은 서비스 트래픽을 위해 원시 암호화된 TCP로 구성됩니다. 페더레이션 트래픽은 검색을 위한 HTTPS로 구성됩니다.	포트 **15443** 은 다른 메시에 대한 TLS 서비스 요청을 수신하는 데 필요합니다. 포트 **8188** 은 페더레이션의 다른 메시에 대한 서비스 검색 요청을 수신하는 데 필요합니다.	

매개변수	설명	값	기본값
spec: gateways: additionalIngress: <ingressName>: service: ports: nodePort:	**nodePort:** 클러스터가 **LoadBalancer** 서비스를 지원하지 않는 경우	지정된 경우 TLS 및 서비스 검색에는 **포트** : 및 **name** 외에도 필요합니다. **NodePort: 30000-32767** 범위에 있어야 합니다.	

다음 예에서 관리자는 **NodePort** 서비스를 사용하여 **green-mesh** 로 통합에 **SMCP**를 구성하고 있습니다.

NodePort의 샘플 **SMCP**

```
gateways:
  additionalIngress:
   ingress-green-mesh:
    enabled: true
    routerMode: sni-dnat
    service:
     type: NodePort
     metadata:
      labels:
       federation.maistra.io/ingress-for: ingress-green-mesh
     ports:
     - port: 15443
      nodePort: 30510
      name: tls
     - port: 8188
      nodePort: 32359
      name: https-discovery
```

1.18.9.2. 페더레이션 신뢰 도메인 매개변수 이해

페더레이션의 각 메시에는 고유한 신뢰 도메인이 있어야 합니다. 이 값은 **ServiceMeshPeer** 리소스에서 메시 통합을 구성할 때 사용됩니다.

```
kind: ServiceMeshControlPlane
metadata:
  name: red-mesh
  namespace: red-mesh-system
spec:
  security:
    trust:
      domain: red-mesh.local
```

표 1.8. 페더레이션 보안 매개변수

매개변수	설명	값	기본값

`spec:`
`  security:`
`  trust:`
`  domain:`
	메시에 대한 신뢰 도메인의 고유 이름을 지정하는 데 사용됩니다. 도메인은 연합 내의 모든 메시에 대해 고유해야 합니다.	**<mesh-name>.local**	해당 없음

콘솔의 프로세스

OpenShift Container Platform 웹 콘솔을 사용하여 ServiceMeshControlPlane을 편집하려면 다음 절차를 따르십시오. 이 예에서는 red-mesh 를 예제로 사용합니다.

1. **cluster-admin 역할의 사용자로 OpenShift Container Platform 웹 콘솔에 로그인합니다.**
2. **Operators → 설치된 Operator로 이동합니다.**
3. **프로젝트 메뉴를 클릭하고 Service Mesh Control Plane을 설치한 프로젝트를 선택합니다. 예를 들면 red-mesh-system 입니다.**
4. **Red Hat OpenShift Service Mesh Operator를 클릭합니다.**
5. **Istio Service Mesh Control Plane 탭에서 ServiceMeshControlPlane 의 이름을 클릭합니다(예: red-mesh).**
6. **ServiceMeshControlPlane 세부 정보 만들기 페이지에서 YAML을 클릭하여 구성을 수정합니다.**
7.

ServiceMeshControlPlane 을 수정하여 페더레이션 인그레스 및 송신 게이트웨이를 추가하고 신뢰 도메인을 지정합니다.

8. 저장을 클릭합니다.

CLI의 프로세스

다음 절차에 따라 명령줄로 **ServiceMeshControlPlane**을 생성하거나 편집합니다. 이 예에서는 **red-mesh** 를 예제로 사용합니다.

1. **cluster-admin** 역할의 사용자로 **OpenShift Container Platform CLI**에 로그인합니다. 다음 명령을 입력합니다. 메시지가 표시되면 사용자 이름과 암호를 입력합니다.

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

2. **Service Mesh Control Plane**을 설치한 프로젝트로 변경합니다(예: **red-mesh-system**).

```
$ oc project red-mesh-system
```

3. **ServiceMeshControlPlane** 파일을 편집하여 페더레이션 인그레스 및 송신 게이트웨이를 추가하고 신뢰 도메인을 지정합니다.

4. 다음 명령을 실행하여 **Service Mesh Control Plane**을 편집합니다. 여기서 **red-mesh-system** 은 시스템 네임스페이스이고 **red-mesh** 는 **ServiceMeshControlPlane** 오브젝트의 이름입니다.

```
$ oc edit -n red-mesh-system smcp red-mesh
```

5. 다음 명령을 입력합니다. 여기서 **red-mesh-system** 은 시스템 네임스페이스이며, **Service Mesh Control Plane** 설치 상태를 확인합니다.

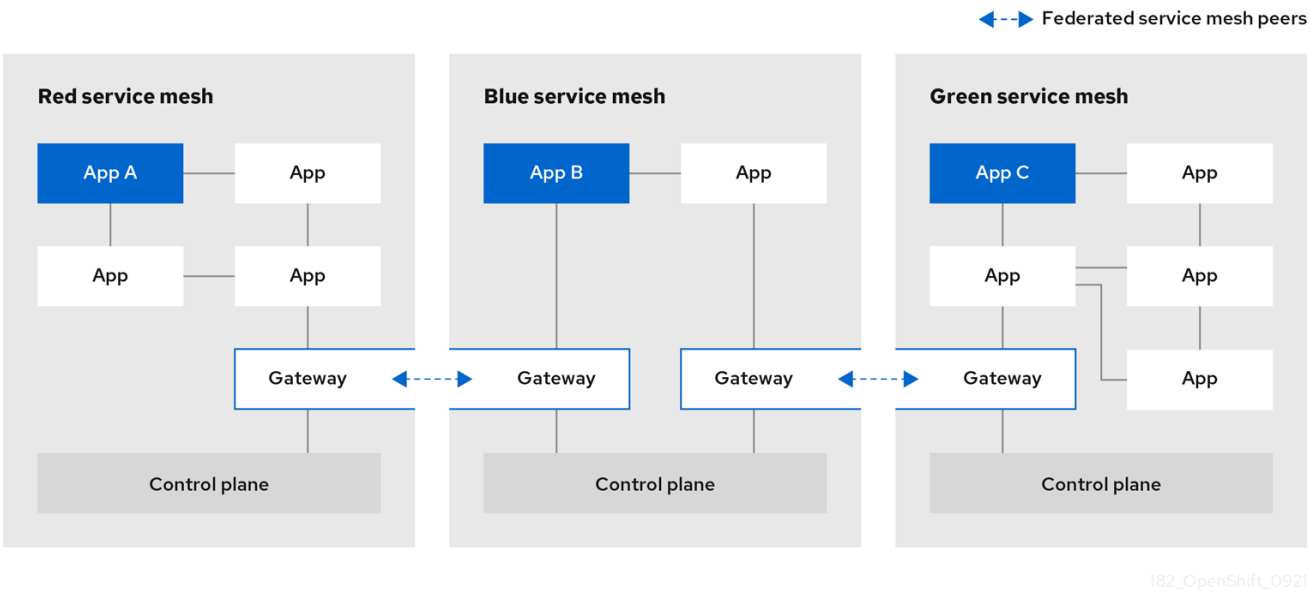
```
$ oc get smcp -n red-mesh-system
```

READY 열에 모든 구성 요소가 준비되었음을 나타내는 설치가 성공적으로 완료되었습니다.

```
NAME      READY   STATUS            PROFILES     VERSION   AGE
red-mesh   10/10   ComponentsReady   ["default"]   2.1.0     4m25s
```

1.18.10. 페더레이션 메시 결합

ServiceMeshPeer 리소스를 생성하여 두 메시 간 페더레이션을 선언합니다. **ServiceMeshPeer** 리소스는 두 메시 간의 페더레이션을 정의하고, 이를 사용하여 피어 메시에 대한 검색, 피어 메시에 대한 액세스, 다른 메시의 클라이언트의 유효성을 확인하는 데 사용되는 인증서를 구성합니다.



메시는 일대일로 통합되므로 각 피어 쌍에는 다른 서비스 메시에 대한 통합 연결을 지정하는 **ServiceMesh peer** 리소스 쌍이 필요합니다. 예를 들어 **red** 와 **green** 이라는 두 개의 메시를 통합하려면 **ServiceMeshPeer** 파일이 필요합니다.

1. **red-mesh-system**에서 녹색 메시에 대한 **ServiceMesh peer**를 만듭니다.
2. **green-mesh-system**에서 빨간색 메시에 대한 **ServiceMesh peer**를 만듭니다.

빨간색,블루, 녹색 이라는 세 개의 메시를 태그하려면 6개의 **ServiceMeshPeriod** 파일이 필요합니다.

1. **red-mesh-system**에서 녹색 메시에 대한 **ServiceMesh peer**를 만듭니다.
2. **red-mesh-system**에서 **Blue** 메시에 대한 **ServiceMesh peer**를 만듭니다.
3. **green-mesh-system**에서 빨간색 메시에 대한 **ServiceMesh peer**를 만듭니다.

4. green-mesh-system에서 Blue 메시에 대한 ServiceMesh peer를 만듭니다.

5. blue-mesh-system에서 빨간색 메시에 대한 ServiceMesh peer를 만듭니다.

6. blue-mesh-system에서 녹색 메시에 대한 ServiceMesh peer를 만듭니다.

ServiceMeshPeriod 리소스의 구성은 다음과 같습니다.

- 검색 및 서비스 요청에 사용되는 다른 메시의 수신 게이트웨이의 주소입니다.
- 지정된 피어 메시와 상호 작용하는 데 사용되는 로컬 수신 및 송신 게이트웨이의 이름입니다.
- 이 메시에 요청을 보낼 때 다른 메시에서 사용하는 클라이언트 ID입니다.
- 다른 메시에서 사용하는 신뢰 도메인입니다.
- 다른 메시에서 사용하는 신뢰 도메인에서 클라이언트 인증서의 유효성을 확인하는 데 사용되는 루트 인증서가 포함된 ConfigMap 의 이름입니다.

다음 예에서 red-mesh 의 관리자는 green-mesh 로 페더레이션을 구성하고 있습니다.

red-mesh에 대한 ServiceMeshpeer 리소스의 예

```
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
```

표 1.9. ServiceMeshPeriod 구성 매개변수

매개변수	설명	값
metadata: name:	이 리소스가 페더레이션을 구성하는 피어 메시의 이름입니다.	문자열
metadata: namespace:	Service Mesh Control Plane이 설치된 이 메시의 시스템 네임스페이스입니다.	문자열
spec: remote: addresses:	이 메시에서 요청을 처리하는 피어 메시의 수신 게이트웨이의 공용 주소 목록입니다.	
spec: remote: discoveryPort:	검색 요청을 처리하는 주소가 있는 포트입니다.	기본값은 8188입니다.
spec: remote: servicePort:	주소가 서비스 요청을 처리하는 포트입니다.	기본값은 15443입니다.
spec: gateways: ingress: name:	피어 메시에서 수신한 요청을 서비스하는 이 메시의 수신 수신 이름입니다. 예: **ingress-green-mesh**.	

매개변수	설명	값
spec: gateways: egress: name:	피어 메시로 전송되는 요청을 서비스하는 이 메시의 송신 이름입니다. 예를 들어 **egress-green-mesh**.	
spec: security: trustDomain:	피어 메시에서 사용하는 신뢰 도메인입니다.	<peerMeshName>.local
spec: security: clientID:	이 메시로 호출할 때 피어 메시에서 사용하는 클라이언트 ID입니다.	<peerMeshTrustDomain>/ns/<peerMeshSystem>/sa/<peerMeshEgressGatewayName>-service-account
spec: security: certificateChain: kind: ConfigMap name:	피어 메시에 의해 제공되는 클라이언트 및 서버 인증서의 유효성을 검사하는 데 사용되는 루트 인증서가 포함된 종류의(예: ConfigMap) 및 리소스 이름입니다. 인증서가 포함된 구성 맵 항목의 키는 **root-cert.pem** 이어야 합니다.	kind: ConfigMap 이름: <peerMesh>-ca-root-cert

1.18.10.1. ServiceMeshpeer 리소스 생성

사전 요구 사항

- **두 개 이상의 OpenShift Container Platform 4.6 이상의 클러스터.**
- **클러스터가 이미 네트워크되어 있어야 합니다.**
- **페더레이션 게이트웨이와 연결된 서비스를 지원하는 로드 밸런서는 원시 TLS 트래픽을 지원하도록 구성해야 합니다.**
- **각 클러스터에는 배포된 페더레이션을 지원하도록 버전 2.1 이상의 ServiceMeshControlPlane 이 구성되어 있어야 합니다.**
- **cluster-admin 역할이 있는 계정.**

CLI의 프로세스

다음 절차에 따라 명령줄에서 **ServiceMesh peer** 리소스를 만듭니다. 이 예에서는 **red-mesh** 에서 **green-mesh** 에 대한 피어 리소스를 생성하는 방법을 보여줍니다.

1. **cluster-admin** 역할의 사용자로 **OpenShift Container Platform CLI**에 로그인합니다. 다음 명령을 입력합니다. 메시지가 표시되면 사용자 이름과 암호를 입력합니다.

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

2. 컨트롤 플레인을 설치한 프로젝트(예: **red-mesh-system**)로 변경합니다.

```
$ oc project red-mesh-system
```

3. 통합하려는 두 메시에 대해 다음 예제를 기반으로 **ServiceMesh Period** 파일을 만듭니다.

red-mesh에 대한 **ServiceMeshPeer** 리소스 예 **- green-mesh**

```
kind: ServiceMeshPeer
apiVersion: federation.maistra.io/v1
metadata:
  name: green-mesh
  namespace: red-mesh-system
spec:
  remote:
    addresses:
    - ingress-red-mesh.green-mesh-system.apps.domain.com
  gateways:
    ingress:
      name: ingress-green-mesh
    egress:
      name: egress-green-mesh
  security:
    trustDomain: green-mesh.local
    clientID: green-mesh.local/ns/green-mesh-system/sa/egress-red-mesh-service-
account
    certificateChain:
      kind: ConfigMap
      name: green-mesh-ca-root-cert
```

4.

다음 명령을 실행하여 리소스를 배포합니다. 여기서 **red-mesh-system** 은 시스템 네임스페이스이고 **servicemeshpeer.yaml** 에는 편집한 파일에 대한 전체 경로가 포함됩니다.

```
$ oc create -n red-mesh-system -f servicemeshpeer.yaml
```

5.

빨간색 메시와 녹색 메시 간 연결이 설정되었는지 확인하려면 **red-mesh-system** 네임스페이스에서 **green-mesh ServiceMesh peer**의 상태를 검사합니다.

```
$ oc -n red-mesh-system get servicemeshpeer green-mesh -o yaml
```

red-mesh와 **green-mesh** 간의 **ServiceMeshpeer** 연결 예

```
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 필드에는 피어 메시의 **istiod**(이 예에서는 녹색 메시)가 현재 메시(이 예에서는 빨간색 메시)에서 **istiod**에 연결되어 있음을 보여줍니다.

status.discoveryStatus.active.watch 필드에는 현재 메시의 **istiod**가 피어 메시의 **istiod**에 연결되어 있는 것으로 표시됩니다.

green-mesh-system 에서 **red-mesh** 피어라는 **servicemeshpeer** 를 선택하면 녹색 메시의 관점에서 동일한 두 연결에 대한 정보를 확인할 수 있습니다.

두 메시 간 연결이 설정되지 않으면 **ServiceMeshPeriod** 상태는 **status.discoveryStatus.inactive** 필드에 이를 나타냅니다.

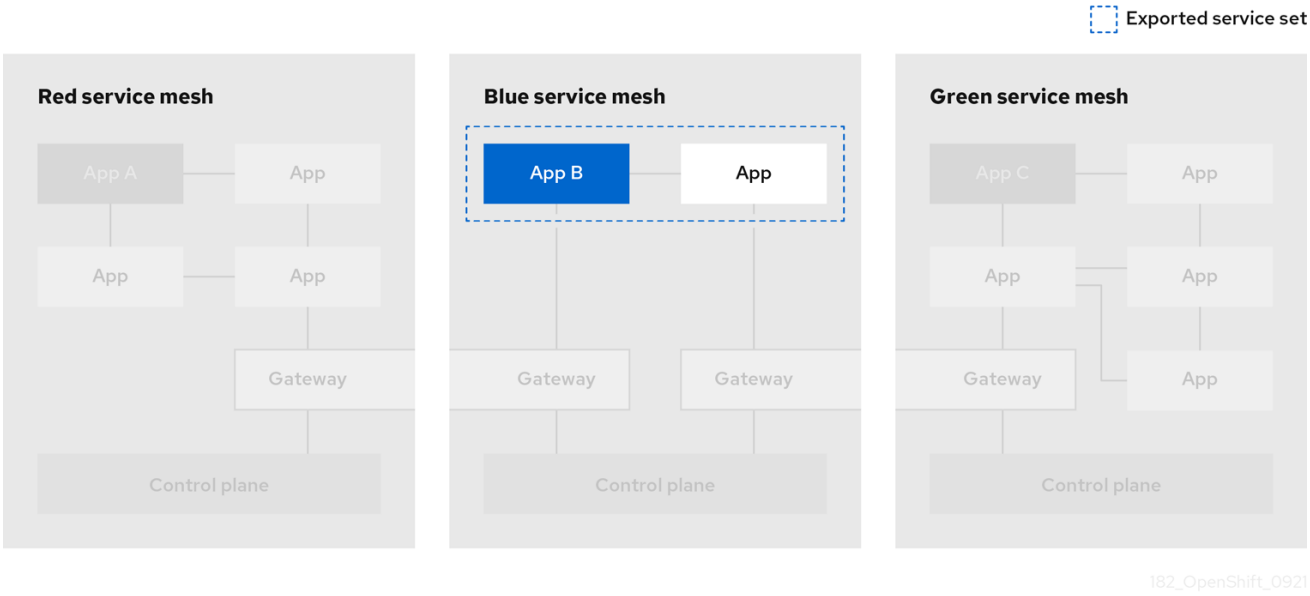
연결 시도가 실패한 이유에 대한 자세한 내용은 **Istiod** 로그, 피어에 대한 송신 트래픽을 처리하는 송신 게이트웨이의 액세스 로그 및 피어 메시의 현재 메시에 대한 수신 트래픽을 처리하는 수신 게이트웨이를 검사합니다.

예를 들어 빨간색 메시가 녹색 메시에 연결할 수 없는 경우 다음 로그를 확인합니다.

- **red-mesh-system**의 **Istiod-red-mesh**
- **red-mesh-system**의 **egress-Green-mesh**
- **green-mesh-system**의 **ingress-red-mesh**

1.18.11. 연합된 메시에서 서비스 내보내기

서비스를 내보내면 메시에서 하나 이상의 해당 서비스를 연합 메시의 다른 멤버와 공유할 수 있습니다.



ExportedServiceSet 리소스를 사용하여 페더레이션 메시에서 다른 피어에 사용할 수 있도록 하는 하나의 메시에서 서비스를 선언합니다. 각 서비스를 피어와 공유할 수 있도록 명시적으로 선언해야 합니다.

- 네임스페이스 또는 이름으로 서비스를 선택할 수 있습니다.
- 와일드카드를 사용하여 서비스를 선택할 수 있습니다. 예를 들어 네임스페이스의 모든 서비스를 내보낼 수 있습니다.
- 별칭을 사용하여 서비스를 내보낼 수 있습니다. 예를 들어 **foo/bar** 서비스를 **custom-ns/bar**로 내보낼 수 있습니다.
- 메시의 시스템 네임스페이스에 표시되는 서비스만 내보낼 수 있습니다. 예를 들어 **networking.istio.io/exportTo** 레이블이 '.'로 설정된 다른 네임스페이스의 서비스는 내보내기 후보가 아닙니다.
- 내보낸 서비스의 경우 대상 서비스는 원래 요청자가 아닌 수신 게이트웨이의 트래픽만 확인합니다(즉, 다른 메시의 송신 게이트웨이 또는 워크로드 또는 요청 발생)의 클라이언트 **ID**를 볼 수 없습니다.

다음 예제는 **red-mesh** 가 **green-mesh** 로 내보내는 서비스의 예입니다.

ExportedServiceSet 리소스의 예

```
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
```

표 1.10. ExportedServiceSet 매개변수

매개변수	설명	값
`metadata:` `  name:`	이 서비스를 노출하는 ServiceMeshpeer의 이름입니다.	**ServiceMeshPeer** 리소스의 메시 **이름** 값과 일치해야 합니다.
`metadata:` `  namespace:`	이 리소스를 포함하는 프로젝트/네임스페이스의 이름(슬래쉬의 시스템 네임스페이스여야 함) .	

`spec:`
`  exportRules:`
`  - type:`
	이 서비스의 내보내기를 관리할 규칙 유형입니다. 서비스에 대해 발견된 첫 번째 일치 규칙은 내보내기에 사용됩니다.	**NameSelector**, **LabelSelector**

`spec:`
`  exportRules:`
`  - type: NameSelector`
`  nameSelector:`
`  namespace:`
`  name:`
	NameSelector 규칙을 만들려면 서비스의 **네임스페이스** 와 **Service** 리소스에 정의된 서비스 **이름을** 지정합니다.	

`spec:`
`  exportRules:`
`  - type: NameSelector`
`  nameSelector:`
`  alias:`
`  namespace:`
`  name:`
	서비스에 대한 **네임스페이스** 와 **이름을** 지정한 후 서비스의 별칭을 사용하는 **NameSelector** 규칙을 만들려면 **네임스페이스** 및 서비스 **이름에** 사용할 별칭을 지정합니다.	

매개변수	설명	값
spec: exportRules: - type: LabelSelector labelSelector: namespace: <exportingMesh> selector: matchLabels: <labelKey>: <labelValue>	**LabelSelector** 규칙을 만들려면 서비스의 **네임스페이스** 를 지정하고 **Service** 리소스에 정의된 **라벨** 을 지정합니다. 위의 예에서 레이블은 **export-service** 입니다.	
spec: exportRules: - type: LabelSelector labelSelector: namespace: <exportingMesh> selector: matchLabels: <labelKey>: <labelValue> aliases: - namespace: name: alias: namespace: name:	서비스에 별칭을 사용하는 **LabelSelector** 규칙을 만들려면 **선택기** 를 지정한 후 서비스의 **이름** 또는 **네임스페이스에** 사용할 별칭을 지정합니다. 위의 예에서 네임스페이스 별칭은 일치하는 모든 서비스에 대한 **info** 입니다.	

red-mesh에 있는 모든 네임스페이스에서 blue-mesh로 이름이 "ratings"인 서비스를 내보냅니다.

```
kind: ExportedServiceSet
apiVersion: federation.maistra.io/v1
metadata:
  name: blue-mesh
  namespace: red-mesh-system
spec:
  exportRules:
  - type: NameSelector
    nameSelector:
      namespace: "*"
      name: ratings
```

west-data-center 네임스페이스에서 **green-mesh**로 모든 서비스를 내보냅니다.

```
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: "*"
```

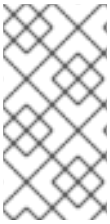
1.18.11.1. ExportedServiceSet 생성

메시 피어에 사용할 수 있는 서비스를 명시적으로 선언하기 위해 **ExportedServiceSet** 리소스를 생성합니다.

서비스는 **<export-name>.<export-namespace>.svc.<ServiceMeshPeer.name>-exports.local** 로 내보내지 고 대상 서비스로 자동으로 라우팅됩니다. 내보낸 서비스가 메시에서 알려진 이름입니다. 수신 게이트웨이가 이 이름으로 향하는 요청을 수신하면 내보낸 실제 서비스로 라우팅됩니다. 예를 들어, 이름이 **ratings.red-mesh-info** 라는 서비스가 **ratings.bookinfo.bookinfo**로 **green-mesh -info**로 내보낸 경우, 서비스는 이름 **ratings.bookinfo.svc.green-mesh-exports.local** 로 내보내지고 해당 호스트 이름에 대한 수신 게이트웨이에서 수신하는 트래픽은 **ratings.red-mesh-bookinfo** 서비스로 라우팅됩니다.

사전 요구 사항

- 클러스터와 **ServiceMeshControlPlane** 은 메시 통합에 대해 구성되어 있습니다.
- **cluster-admin** 역할이 있는 계정.



참고

아직 존재하지 않는 경우에도 내보내기를 위해 서비스를 구성할 수 있습니다. **ExportedServiceSet**에 지정된 값과 일치하는 서비스가 배포되면 자동으로 내보내집니다.

CLI의 프로세스

다음 절차에 따라 명령줄에서 **ExportedServiceSet** 을 만듭니다.

1. **cluster-admin** 역할의 사용자로 **OpenShift Container Platform CLI**에 로그인합니다. 다음 명령을 입력합니다. 메시지가 표시되면 사용자 이름과 암호를 입력합니다.

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

2. **Service Mesh Control Plane**을 설치한 프로젝트(예: **red-mesh-system**)로 변경합니다.

```
$ oc project red-mesh-system
```

3. **red-mesh** 가 서비스를 **green-mesh** 로 내보내는 다음 예제를 기반으로 **ExportedServiceSet** 파일을 만듭니다.

red-mesh에서 **green-mesh**로 **ExportedServiceSet** 리소스의 예

```
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. 다음 명령을 실행하여 **red-mesh-system** 네임스페이스에 **ExportedServiceSet** 리소스를 업로드하고 만듭니다.

```
$ oc create -n <ControlPlaneNamespace> -f <ExportedServiceSet.yaml>
```

예를 들어 다음과 같습니다.

```
$ oc create -n red-mesh-system -f export-to-green-mesh.yaml
```

5. 페더레이션 메시의 각 메시 피어에 필요한 추가 **ExportedServiceSets** 를 만듭니다.

6. **red-mesh** 에서 내보낸 서비스를 검증하여 **green-mesh** 와 공유하려면 다음 명령을 실행합니다.

```
$ oc get exportedserviceset <PeerMeshExportedTo> -o yaml
```

예를 들어 다음과 같습니다.

```
$ oc get exportedserviceset green-mesh -o yaml
```

7. 다음 명령을 실행하여 **green-mesh**와 공유할 **red-mesh** 내보내기를 서비스의 유효성을 검사합니다.

```
$ oc get exportedserviceset <PeerMeshExportedTo> -o yaml
```

예를 들어 다음과 같습니다.

```
$ oc -n red-mesh-system get exportedserviceset green-mesh -o yaml
```

녹색 메시와 공유되는 빨간색 메시에서 내보낸 서비스의 유효성을 검사하는 예입니다.

```
status:
  exportedServices:
```

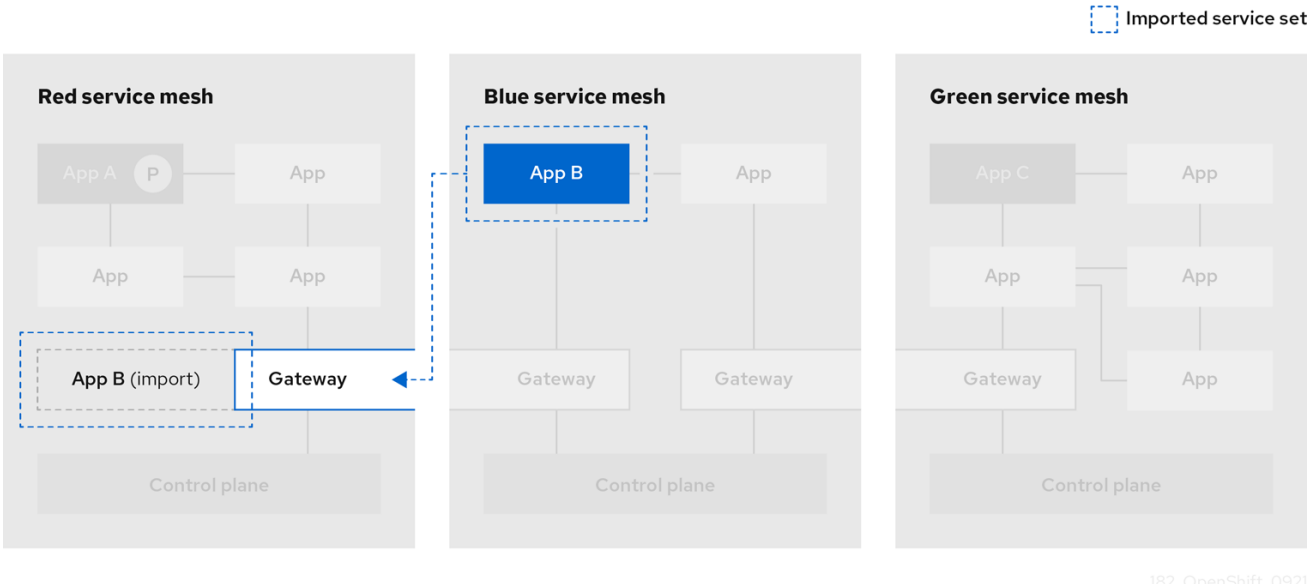
```
- exportedName: red-ratings.info.svc.green-mesh-exports.local
  localService:
    hostname: ratings.red-mesh-info.svc.cluster.local
    name: ratings
    namespace: red-mesh-info
- exportedName: reviews.red-mesh-info.svc.green-mesh-exports.local
  localService:
    hostname: reviews.red-mesh-info.svc.cluster.local
    name: reviews
    namespace: red-mesh-info
```

status.exportedServices 어레이는 현재 내보낸 서비스를 나열합니다(해당 서비스는 **ExportedServiceSet** 오브젝트의 내보내기 규칙과 일치함). 배열의 각 항목은 내보낸 서비스의 이름과 내보낸 로컬 서비스에 대한 세부 정보를 나타냅니다.

내보낼 것으로 예상되는 서비스가 누락된 경우 **Service** 오브젝트가 있는지, 해당 이름 또는 레이블이 **ExportedServiceSet** 오브젝트에 정의된 **exportRules** 와 일치하고 **Service** 오브젝트의 네임스페이스가 **ServiceMeshMemberRoll** 또는 **ServiceMeshMember** 오브젝트를 사용하여 서비스 메시의 멤버로 구성되어 있는지 확인합니다.

1.18.12. 통합 메시로 서비스 가져오기

서비스를 가져오면 서비스 메시 내에서 다른 메시에서 내보낸 서비스를 명시적으로 지정할 수 있습니다.



ImportedServiceSet 리소스를 사용하여 가져올 서비스를 선택합니다. 메시 피어 및 명시적으로 가져

온 서비스만 메시에서 사용할 수 있습니다. 명시적으로 가져오지 않은 서비스는 메시 내에서 사용할 수 없습니다.

- 네임스페이스 또는 이름으로 서비스를 선택할 수 있습니다.
- 예를 들어 와일드카드를 사용하여 서비스를 선택하여 네임스페이스로 내보낸 모든 서비스를 가져올 수 있습니다.
- 메시에 글로벌할 수 있는 라벨 선택기를 사용하여 내보내기 서비스를 선택하거나 특정 멤버 네임스페이스로 범위를 지정할 수 있습니다.
- 별칭을 사용하여 서비스를 가져올 수 있습니다. 예를 들어 **custom-ns/bar** 서비스를 **other-mesh/bar** 로 가져올 수 있습니다.
- 정규화된 도메인 이름에 대해 가져온 서비스의 **name.namespace** 에 추가할 사용자 지정 도메인 접미사를 지정할 수 있습니다(예: **bar.other-mesh.imported.local**).

다음 예제는 **green-mesh** 에서 **red-mesh** 에 의해 내보낸 서비스를 가져오는 데 사용됩니다.

ImportedServiceSet의 예

```
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
```

표 1.11. ImportedServiceSet 매개변수

매개변수	설명	값
metadata: name:	연결된 메시에 서비스를 내보낸 ServiceMeshPeriod의 이름입니다.	
metadata: namespace:	ServiceMeshpeer 리소스(mesh 시스템 네임스페이스)가 포함된 네임스페이스의 이름입니다.	
spec: importRules: - type:	서비스의 가져오기를 제어하는 규칙 유형입니다. 서비스에 대해 발견된 첫 번째 일치 규칙이 가져오기에 사용됩니다.	**NameSelector**
spec: importRules: - type: NameSelector nameSelector: namespace: name:	**NameSelector** 규칙을 만들려면 **네임스페이스** 와 내보낸 서비스의 **이름을** 지정합니다.	
spec: importRules: - type: NameSelector importAsLocal:	로컬 서비스를 사용하여 원격 엔드포인트를 집계하려면 **true** 로 설정합니다. **true** 인 경우 서비스는 **<name>.<namespace>.svc.cluster.local로 가져옵니다.**	**true/false**
spec: importRules: - type: NameSelector nameSelector: namespace: name: alias: namespace: name:	서비스에 대한 **네임스페이스** 와 **이름을** 지정한 후 서비스의 별칭을 사용하는 **NameSelector** 규칙을 만들려면 **네임스페이스** 및 서비스 **이름에** 사용할 별칭을 지정합니다.	

red-mesh에서 "info/ratings" 서비스를 blue-mesh로 가져옵니다.

```
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-mesh의 **west-data-center** 네임스페이스에서 **green-mesh**로 모든 서비스를 가져옵니다. 이러한 서비스는 **<name>.west-data-center.svc.red-mesh-imports.local**로 액세스할 수 있습니다.

```
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 생성

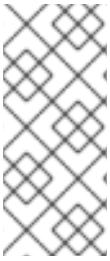
ImportedServiceSet 리소스를 생성하여 메시로 가져올 서비스를 명시적으로 선언합니다.

서비스는 이름이 **<exported-name>. .<exported-namespace> svc. <ServiceMeshPeer.name>.remote** 로 가져와서 송신 게이트웨이 네임스페이스 내에서만 표시되고 내보낸 서비스의 호스트 이름과 연결됩니다. 이 서비스는 **importAsLocal**가 **true**으로 설정되어 있는 경우 (**domainSuffix**이 **svc.cluster.local**인 경우)가 아니면 기본적으로 **domainSuffix**이 **svc. <ServiceMeshPeer.name>-imports.local**인 **<export-name>.<export-namespace>.<domainSuffix>**로 로컬에서 사용할 수 있습니다. **importAsLocal** 이 **false** 로 설정되면 가져오기 규칙의 도메인 접미사가 적

용됩니다. 로컬 가져오기를 메시의 다른 서비스와 마찬가지로 처리할 수 있습니다. 내보낸 서비스의 원격 이름으로 리디렉션되는 송신 게이트웨이를 통해 자동으로 라우팅됩니다.

사전 요구 사항

- 클러스터와 **ServiceMeshControlPlane** 은 메시 통합에 대해 구성되어 있습니다.
- **cluster-admin** 역할이 있는 계정.



참고

아직 내보내지 않은 경우에도 가져오기 위해 서비스를 구성할 수 있습니다. **ImportedServiceSet**에 지정된 값과 일치하는 서비스가 배포되고 내보내면 자동으로 가져옵니다.

CLI의 프로세스

다음 절차에 따라 명령줄에서 **ImportedServiceSet** 을 생성합니다.

1. **cluster-admin** 역할의 사용자로 **OpenShift Container Platform CLI**에 로그인합니다. 다음 명령을 입력합니다. 메시지가 표시되면 사용자 이름과 암호를 입력합니다.

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

2. **Service Mesh Control Plane**을 설치한 프로젝트(예: **green-mesh-system**)로 변경합니다.

```
$ oc project green-mesh-system
```

3. 다음 예제를 기반으로 **ImportedServiceSet** 파일을 만듭니다. 여기서 **green-mesh** 가 이전에 **red-mesh** 에서 내보낸 서비스를 가져옵니다.

red-mesh에서 **green-mesh**로 **ImportedServiceSet** 리소스의 예

```
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. 다음 명령을 실행하여 **green-mesh-system** 네임스페이스에 **ImportedServiceSet** 리소스를 업로드하고 만듭니다.

```
$ oc create -n <ControlPlaneNamespace> -f <ImportedServiceSet.yaml>
```

예를 들어 다음과 같습니다.

```
$ oc create -n green-mesh-system -f import-from-red-mesh.yaml
```

5. 페더레이션 메시의 각 메시 피어에 필요한 대로 추가로 **ImportedServiceSet** 리소스를 만듭니다.

6. **green-mesh** 에 가져온 서비스를 확인하려면 다음 명령을 실행합니다.

```
$ oc get importedserviceset <PeerMeshImportedInto> -o yaml
```

예를 들어 다음과 같습니다.

```
$ oc get importedserviceset green-mesh -o yaml
```

7. 다음 명령을 실행하여 메시로 가져온 서비스의 유효성을 검사합니다.

```
$ oc get importedserviceset <PeerMeshImportedInto> -o yaml
```

예를 들어, 빨간색 메시에서 내보낸 서비스가 **'green-mesh- system** 네임스페이스에서 가져온 **serviceset/red-mesh'** 오브젝트의 **status** 섹션을 사용하여 녹색 메시로 가져왔는지 확인합니다.

```
$ 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: ""
```

위 예제에서는 **localService** 아래에 있는 채워진 필드에 표시된 대로 **ratings** 서비스만 가져옵니다. **reviews** 서비스는 가져오기용으로 사용할 수 있지만 **ImportedServiceSet** 오브젝트의 **importRules** 와 일치하지 않기 때문에 현재 가져온 것은 아닙니다.

1.18.13. 장애 조치를 위한 페더레이션 메시 구성

장애 조치(**failover**)는 자동 및 원활하게 안정적인 백업 시스템(예: 다른 서버)으로 전환할 수 있습니다. 페더레이션 메시의 경우 다른 메시의 서비스에 장애 조치하도록 하나의 메시에서 서비스를 구성할 수 있습니다.

ImportedServiceSet 리소스에서 **importAsLocal** 및 **locality** 설정을 설정한 다음 **ImportedServiceSet** 에 지정된 지역으로 서비스에 대한 장애 조치를 구성하는 **DestinationRule** 을 구성하여 페일오버를 구성합니다.

사전 요구 사항

- 두 개 이상의 **OpenShift Container Platform 4.6** 이상의 클러스터가 이미 네트워크 연결 및 통합되었습니다.
- 페더레이션 메시의 각 메시 피어에 대해 이미 생성된 **ExportedServiceSet** 리소스입니다.

- 통합 메시의 각 메시 피어에 대해 이미 **ImportedServiceSet** 리소스가 생성되었습니다.
- **cluster-admin** 역할이 있는 계정.

1.18.13.1. 장애 조치(failover)를 위해 ImportedServiceSet 구성

관리자는 현지화된 부하 분산을 통해 트래픽이 시작된 위치와 종료될 위치를 기준으로 엔드포인트의 트래픽 배포를 제어할 수 있습니다. 이러한 지역에는 **{region}/{zone}/{sub-zone}** 형식의 지역 계층을 지정하는 임의의 레이블을 사용하여 지정됩니다.

이 섹션의 예제에서 **green-mesh** 는 **us-east** 지역에 있으며 **red-mesh** 는 **us-west** 지역에 있습니다.

red-mesh에서 **green-mesh**로 **ImportedServiceSet** 리소스의 예

```
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
```

표 1.12. ImportedServiceLocality 필드 테이블

이름	설명	유형
지역:	가져온 서비스가 있는 리전입니다.	string

이름	설명	유형
서브 존:	가져온 서비스가 있는 하위 영역입니다. I Subzone은 지정되며 영역도 지정해야 합니다.	string
영역:	가져온 서비스가 있는 영역입니다. Zone을 지정하는 경우 지역도 지정해야 합니다.	string

절차

1. **cluster-admin 역할의 사용자로 OpenShift Container Platform CLI에 로그인하고 다음 명령을 입력합니다.**

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

2. **Service Mesh Control Plane을 설치한 프로젝트로 변경하고 다음 명령을 입력합니다.**

```
$ oc project <smcp-system>
```

예를 들면 green-mesh-system 입니다.

```
$ oc project green-mesh-system
```

3. **ImportedServiceSet 파일을 편집합니다. 여기서 < ImportedServiceSet.yaml >에는 편집하려는 파일에 대한 전체 경로가 포함되어 있으며 다음 명령을 입력합니다.**

```
$ oc edit -n <smcp-system> -f <ImportedServiceSet.yaml>
```

예를 들어 이전 ImportedServiceSet 예제에 표시된 대로 빨간색-mesh-system에서 green-mesh-system으로 가져오는 파일을 수정하려면 다음을 수행합니다.

```
$ oc edit -n green-mesh-system -f import-from-red-mesh.yaml
```

4. **파일을 수정합니다.**

   a. **spec.importRules.importAsLocal 을 true 로 설정합니다.**

b. spec.locality 를 지역 , 영역 또는 하위 영역으로 설정합니다.

c. 변경 사항을 저장하십시오.

1.18.13.2. 장애 조치(failover)를 위해 DestinationRule 구성

다음을 구성하는 DestinationRule 리소스를 만듭니다.

- 서비스에 대한 이상값 탐지입니다. 장애 조치가 제대로 작동하려면 이 작업이 필요합니다. 특히, 서비스 엔드포인트가 비정상인 시기를 확인하도록 사이드카 프록시를 구성하여 결국 다음 로컬에 장애 조치를 트리거합니다.
- 리전 간 페일오버 정책. 이렇게 하면 영역 경계 이외의 장애 조치(failover)가 예측될 수 있습니다.This ensures that failover beyond a region boundary will behave predictably.

절차

1. cluster-admin 역할의 사용자로 OpenShift Container Platform CLI에 로그인합니다. 다음 명령을 입력합니다. 메시지가 표시되면 사용자 이름과 암호를 입력합니다.

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

2. Service Mesh Control Plane을 설치한 프로젝트로 변경합니다.

```
$ oc project <smcp-system>
```

예를 들면 green-mesh-system 입니다.

```
$ oc project green-mesh-system
```

3. green-mesh가 사용할 수 없는 다음 예제를 기반으로 DestinationRule 파일을 만듭니다. 이 파일은 us-east 리전의 green-mesh에서 us-west 으로 트래픽을 라우팅해야 합니다.

DestinationRule예

```
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** 을 배포합니다. 여기서 **< DestinationRule >**에는 파일의 전체 경로가 포함되어 다음 명령을 입력합니다.

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

예를 들어 다음과 같습니다.

```
$ oc create -n info -f green-mesh-us-west-DestinationRule.yaml
```

1.18.14. 연합 메시에서 서비스 제거

연합 메시에서 서비스를 제거해야 하는 경우(예: 사용되지 않거나 다른 서비스로 교체된 경우).

1.18.14.1. 단일 메시에서 서비스를 제거하려면 다음을 수행합니다.

더 이상 서비스에 액세스할 필요가 없는 메시 피어의 **ImportedServiceSet** 리소스에서 서비스 항목을 제거합니다.

1.18.14.2. 전체 연합 메시에서 서비스를 제거하려면

서비스를 소유한 메시의 **ExportedServiceSet** 리소스에서 서비스 항목을 제거합니다.

1.18.15. 연합된 메시에서 메시 제거

페더레이션에서 메시를 제거해야 하는 경우 그렇게 할 수 있습니다.

1. 삭제된 메시의 **ServiceMeshControlPlane** 리소스를 편집하여 피어 메시에 대한 모든 연합 수신 게이트웨이를 제거합니다.

2. 제거된 메시가 연결된 각 메시 피어에 대해:

   a. 두 메시를 연결하는 **ServiceMesh peer** 리소스를 제거합니다.

   b. 피어 메시의 **ServiceMeshControlPlane** 리소스를 편집하여 제거된 메시를 제공하는 송신 게이트웨이를 제거합니다.

1.19. 확장

WebAssembly 확장을 사용하여 **Red Hat OpenShift Service Mesh** 프록시에 새 기능을 직접 추가할 수 있습니다. 이를 통해 애플리케이션에서 더 일반적인 기능을 이동하고 **WebAssembly** 바이트 코드로 컴파일하는 단일 언어로 구현할 수 있습니다.

참고

WebAssembly 확장은 **IBM Z** 및 **IBM Power Systems**에서 지원되지 않습니다.

1.19.1. WebAssembly 모듈 개요

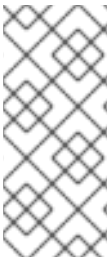
WebAssembly 모듈은 프록시를 포함한 여러 플랫폼에서 실행될 수 있으며 광범위한 언어 지원, 빠른 실행 및 샌드박스 기반 보안 모델을 제공합니다.

Red Hat OpenShift Service Mesh 확장은 **Envoy HTTP** 필터 이며 다양한 기능을 제공합니다.

- 요청 및 응답의 본문과 헤더를 조작합니다.

- 인증 또는 정책 검사와 같이 요청 경로에 없는 서비스에 대한 대역 외 **HTTP** 요청

- 필터가 서로 통신할 수 있는 사이드 채널 데이터 스토리지 및 큐입니다.



참고

새 **WebAssembly** 확장을 생성할 때**ECDHEs mPlugin API**를 사용합니다. **ServiceMeshExtension API**는 **Red Hat OpenShift Service Mesh** 버전 **2.2**에서 더 이상 사용되지 않으며 **Red Hat OpenShift Service Mesh** 버전 **2.3**에서 제거되었습니다.

Red Hat OpenShift Service Mesh 확장을 작성하는 데는 다음 두 가지가 있습니다.

1. **proxy-wasm API** 를 노출하는 **SDK**를 사용하여 확장 기능을 작성하고 **WebAssembly** 모듈로 컴파일해야 합니다.

2. 그런 다음 모듈을 컨테이너로 패키징해야 합니다.

지원되는 언어

WebAssembly 바이트 코드에 컴파일된 모든 언어를 사용하여 **Red Hat OpenShift Service Mesh** 확장을 작성할 수 있지만, 다음 언어에는 **proxy-wasm API**를 공개하는 기존 **SDK**가 있어 직접 사용할 수 있습니다.

표 **1.13.** 지원되는 언어

언어	유지 관리자	리포지터리
AssemblyScript	solo.io	solo-io/proxy-runtime
C++	proxy-wasm 팀(Istio 커뮤니티)	proxy-wasm/proxy-wasm-cpp-sdk
Go	tetrate.io	tetratelabs/proxy-wasm-go-sdk
Rust	proxy-wasm 팀(Istio 커뮤니티)	proxy-wasm/proxy-wasm-rust-sdk

1.19.2. ExsmPlugin 컨테이너 형식

Istio는wasm 플러그인 메커니즘에서 OCI(Open Container Initiative) 이미지를 지원합니다. wasm 플러그인을 컨테이너 이미지로 배포할 수 있으며 spec.url 필드를 사용하여 컨테이너 레지스트리 위치를 참조할 수 있습니다. 예를 들어 quay.io/my-username/my-plugin:latest.

WASM 모듈에 대한 각 실행 환경(runtime)에는 런타임별 구성 매개 변수가 있을 수 있으므로 WASM 이미지는 다음 두 개의 계층으로 구성될 수 있습니다.

- plugin.wasm (필수) - 콘텐츠 계층. 이 계층은 런타임을 통해 로드할 WebAssembly 모듈의 바이트 코드가 포함된 .wasm 바이너리로 구성됩니다. 이 파일의 이름을 plugin.wasm 로 지정해야 합니다.

- runtime-config.json (선택 사항) - 구성 계층. 이 계층은 대상 런타임의 모듈에 대한 메타데이터를 설명하는 JSON 형식의 문자열로 구성됩니다. 구성 계층에는 대상 런타임에 따라 추가 데이터가 포함될 수도 있습니다. 예를 들어 WASM Envoy Filter 구성에는 필터에서 사용할 수 있는 root_id가 포함되어 있습니다.

1.19.3. WasmPlugin API 참조

WasmPlugins API는 Istio 프록시에서 WebAssembly 필터를 통해 제공하는 기능을 확장하는 메커니즘을 제공합니다.

여러 WasmPlugins를 배포할 수 있습니다. 단계 및 우선 순위 설정은 Envoy의 필터 체인의 일부로 실행 순서(Envoy의 필터 체인의 일부로)를 결정하여 사용자 제공wasmPlugins와 Istio의 내부 필터 간 복잡한 상호 작용을 구성할 수 있습니다.

다음 예제에서 인증 필터는 OpenID 흐름을 구현하고 Authorization 헤더를 JSON 웹 토큰(JWT)으로 채웁니다. Istio 인증은 이 토큰을 사용하여 수신 게이트웨이에 배포합니다. ExsmPlugin 파일은 프록시 사이드카 파일 시스템에 있습니다. 필드 URL을 확인합니다.

```
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
```

다음은 동일한 예입니다. 그러나 이번에는 파일 시스템의 파일 대신 **OCI(Open Container Initiative)** 이미지가 사용됩니다. **URL ,imagePullPolicy , imagePullSecret** 필드를 기록해 둡니다.

```
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
```

표 1.14. 와smPlugin 필드 참조

필드	유형	설명	필수 항목
spec.selector	WorkloadSelector	이 플러그인 구성을 적용해야 하는 특정 Pod/VM 세트를 선택하는 데 사용되는 기준입니다. 생략하면 이 구성이 동일한 네임스페이스의 모든 워크로드 인스턴스에 적용됩니다. 구성 루트 네임스페이스에 Extras **mPlugin** 필드가 있는 경우 모든 네임스페이스의 적용 가능한 모든 워크로드에 적용됩니다.	아니요

필드	유형	설명	필수 항목
spec.url	string	Exsm 모듈 또는 OCI 컨테이너의 URL입니다. 스키마가 없는 경우 기본값은 **oci://** 로, OCI 이미지를 참조합니다. 다른 유효한 체계는 프록시 컨테이너 내에 로컬로 존재하는 .wasm 모듈 파일을 참조하기 위한 **file://** 와 원격으로 호스팅되는 .wasm 모듈 파일의 경우 **http[s]://** 입니다.	아니요
spec.sha256	string	wasm 모듈 또는 OCI 컨테이너를 확인하는 데 사용할 SHA256 체크섬입니다. **url** 필드에서 SHA256을 이미 참조하는 경우(**@sha256:** 표기법 사용) 이 필드의 값과 일치해야 합니다. 태그에서 OCI 이미지를 참조하고 이 필드가 설정된 경우 가져오기 후 이 필드의 콘텐츠에 대해 체크섬이 확인됩니다.	아니요
spec.imagePullPolicy	PullPolicy	OCI 이미지를 가져올 때 적용할 가져오기 동작입니다. SHA 대신 태그에서 이미지를 참조하는 경우에만 관련이 있습니다. **url** 필드에서 OCI 이미지를 참조하고 **latest** 태그가 **Always** 값이 Always 가 기본값인 경우 미러링 K8s 동작을 제외하고 기본값은 **IfNotPresent** 입니다. **url** 필드가 **file://** 또는 **http[s]://** 를 직접 사용하는 경우 설정이 무시됩니다.	아니요

필드	유형	설명	필수 항목
spec.imagePullSecret	string	OCI 이미지 가져오기에 사용할 자격 증명. 이미지를 가져올 때 레지스트리에 대한 인증을 위한 풀 시크릿이 포함된 **wasmPlugin** 오브젝트와 동일한 네임스페이스에 시크릿의 이름입니다.	아니요
spec.phase	PluginPhase	필터 체인에서 이 **WasmPlugin** 개체가 삽입되는 위치를 결정합니다.	아니요
spec.priority	**int64**	동일한 **단계** 값이 있는 **WasmPlugins** 개체의 순서를 결정합니다. 동일한 단계의 동일한 워크로드에 여러 개의 fe **smPlugins** 개체가 적용되는 경우 우선 순위와 내림차순으로 적용됩니다. **우선순위** 필드가 설정되지 않은 경우 또는 동일한 값을 가진 두 개의 **wasmPlugins** 개체 **가 있는 경우, 순서는 월 mPlugins** 오브젝트의 이름과 네임 스페이스에서 결정됩니다. 기본값은 **0** 입니다.	아니요
spec.pluginName	string	Envoy 구성에서 사용되는 플러그인 이름입니다. 일부 Requiresm 모듈에는 이 값을 실행하도록 하려면 이 값이 필요할 수 있습니다.	아니요
spec.pluginConfig	struct	플러그인으로 전달할 구성입니다.	아니요
spec.pluginConfig.verificationKey	string	서명된 OCI 이미지 또는 wasm 모듈의 서명을 확인하는 데 사용되는 공개 키입니다. PEM 형식으로 제공해야 합니다.	아니요

WorkloadSelector 개체는 필터를 프록시에 적용할 수 있는지 확인하는 데 사용되는 기준을 지정합니

다. 일치하는 기준에는 프록시와 연결된 메타데이터, pod/VM에 연결된 라벨과 같은 워크로드 인스턴스 정보 또는 초기 핸드셰이크 중 Istio에 제공하는 기타 정보가 포함됩니다. 여러 조건이 지정된 경우 워크로드 인스턴스를 선택하기 위해 모든 조건을 일치해야 합니다. 현재는 라벨 기반 선택 메커니즘만 지원됩니다.

표 1.15. WorkloadSelector

필드	유형	설명	필수 항목
matchLabels	map<string, string>	정책을 적용해야 하는 특정 Pod/VM 세트를 나타내는 하나 이상의 레이블입니다. 레이블 검색 범위는 리소스가 있는 구성 네임스페이스로 제한됩니다.	있음

PullPolicy 오브젝트는 OCI 이미지를 가져올 때 적용할 풀 동작을 지정합니다.

표 1.16. PullPolicy

값	설명
<empty>	기본값은 latest 태그가 있는 OCI 이미지를 제외하고 IfNotPresent 값으로 기본값은 Always 여야 합니다.
IfNotPresent	기존 버전의 이미지를 이전에 가져온 경우 이 버전이 사용됩니다. 이미지가 로컬에 없는 경우 최신 버전을 가져옵니다.
Always	이 플러그인을 적용할 때 항상 최신 버전의 이미지를 가져옵니다.

구조체 는 동적으로 입력된 값에 매핑되는 필드로 구성된 구조화된 데이터 값을 나타냅니다.Represents a structured data value, consisting of fields which map to dynamically typed values. 일부 언어에서는 Struct가 네이티브 표현에 의해 지원될 수 있습니다. 예를 들어 JavaScript와 같은 스크립팅 언어에서 구조체는 개체로 표시됩니다.

표 1.17. struct

필드	유형	설명
필드	map<string, Value>	동적으로 입력된 값의 맵입니다.

PluginPhase 는 플러그인이 삽입되는 필터 체인의 단계를 지정합니다.

표 1.18. PluginPhase

필드	설명
<empty>	컨트롤 플레인은 플러그인을 삽입할 위치를 결정합니다. 이는 일반적으로 라우터 바로 앞에 필터 체인의 끝에 있습니다. 플러그인이 다른 플러그인과 독립적 인 경우 PluginPhase를 지정하지 마십시오.
AUTHN	Istio 인증 필터 전에 플러그인을 삽입합니다.
AUTHZ	Istio 권한 부여 필터 전에 및 Istio 인증 필터 후에 플러그인을 삽입합니다.
STATS	Istio 통계 필터 전에 및 Istio 권한 부여 필터 후에 플러그인을 삽입합니다.

1.19.3.1. 월 mPlugin 리소스 배포

wasmPlugin 리소스를 사용하여 Red Hat OpenShift Service Mesh 확장을 활성화할 수 있습니다. 이 예제에서 istio-system 은 Service Mesh Control Plane 프로젝트의 이름입니다. 다음 예제에서는 사용자를 인증하기 위해 OpenID Connect 흐름을 수행하는 openid-connect 필터를 생성합니다.

절차

1. **다음 예제 리소스를 만듭니다.**

plugin.yaml 예

```
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. 다음 명령을 사용하여 **plugin.yaml** 파일을 적용합니다.

```
$ oc apply -f plugin.yaml
```

1.19.4. ServiceMeshExtension 컨테이너 형식

컨테이너 이미지를 유효한 확장 이미지로 만들려면 컨테이너 파일 시스템의 루트에 **WebAssembly** 모듈의 바이트 코드가 포함된 **.wasm** 파일과 **manifest.yaml** 파일이 있어야 합니다.

참고

새 **WebAssembly** 확장을 생성할 때**ECDHEs mPlugin API**를 사용합니다. **ServiceMeshExtension API**는 **Red Hat OpenShift Service Mesh** 버전 **2.2**에서 더 이상 사용되지 않으며 **Red Hat OpenShift Service Mesh** 버전 **2.3**에서 제거되었습니다.

manifest.yaml

```
schemaVersion: 1

name: <your-extension>
description: <description>
version: 1.0.0
phase: PreAuthZ
priority: 100
module: extension.wasm
```

표 1.19. manifest.yml에 대한 필드 참조

필드	설명	필수 항목

필드	설명	필수 항목
schemaVersion	매니페스트 스키마 버전 지정에 사용됩니다. 현재 가능한 값은 **1**입니다.	이 필드는 필수 항목입니다.
name	해당 확장의 이름입니다.	이 필드는 메타데이터일 뿐이며 현재 사용되지 않습니다.
description	해당 확장의 설명입니다.	이 필드는 메타데이터일 뿐이며 현재 사용되지 않습니다.
version	해당 확장의 버전입니다.	이 필드는 메타데이터일 뿐이며 현재 사용되지 않습니다.
phase	해당 확장의 기본 실행 단계입니다.	이 필드는 필수 항목입니다.
priority	해당 확장의 기본 우선순위입니다.	이 필드는 필수 항목입니다.
module	컨테이너 파일 시스템의 루트에서 WebAssembly 모듈에 대한 상대적 경로입니다.	이 필드는 필수 항목입니다.

1.19.5. ServiceMeshExtension 참조

ServiceMeshExtension API는 WebAssembly 필터를 통해 Istio 프록시에서 제공하는 기능을 확장하는 메커니즘을 제공합니다. WebAssembly 확장을 작성하는 데는 두 가지 부분이 있습니다.

1. **proxy-wasm API를 노출하는 SDK를 사용하여 확장 기능을 작성하고 WebAssembly 모듈로 컴파일합니다.**
2. **컨테이너에 패키징합니다.**

참고

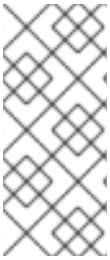
새 WebAssembly 확장을 생성할 때ECDHEs mPlugin API를 사용합니다. Red Hat OpenShift Service Mesh 버전 2.2에서 더 이상 사용되지 않는 ServiceMeshExtension API는 Red Hat OpenShift Service Mesh 버전 2.3에서 제거되었습니다.

표 1.20. ServiceMeshExtension 필드 참조

필드	설명
metadata.namespace	**ServiceMeshExtension** 소스의 **metadata.namespace** 필드에는 특별한 의미가 있습니다. 컨트롤 플레인 네임스페이스와 동일한 경우 확장은 해당 **workloadSelector** 값과 일치하는 서비스 메시의 모든 워크로드에 적용됩니다. 다른 메시 네임스페이스에 배포하면 동일한 네임스페이스의 워크로드에만 적용됩니다.
spec.workloadSelector	**spec.workloadSelector** 필드는 Istio 게이트웨이 리소스의 **spec.selector** 필드와 동일한 의미가 있습니다. Pod 레이블을 기반으로 하는 워크로드와 일치합니다. **workloadSelector** 값을 지정하지 않으면 네임스페이스의 모든 워크로드에 확장이 적용됩니다.
spec.config	이 필드는 배포 중인 확장 기능에 따라 의미 체계와 함께 확장으로 전달되는 구조화된 필드입니다.
spec.image	확장자가 있는 이미지를 가리키는 컨테이너 이미지 URI입니다.
spec.phase	단계는 인증, 권한 부여, 지표 생성과 같은 기존 Istio 기능과 관련하여 필터 체인에서 확장이 삽입되는 위치를 결정합니다. 유효한 값: PreAuthN, PostAuthN, PreAuthZ, PostAuthZ, PreStats, PostStats. 이 필드의 기본값은 확장의 **manifest.yaml** 파일에 설정된 값이지만 사용자가 덮어쓸 수 있습니다.
spec.priority	동일한 **spec.phase** 값이 있는 여러 확장이 동일한 워크로드 인스턴스에 적용되는 경우 **spec.priority** 값에 따라 실행 순서가 결정됩니다. 우선순위가 높은 확장이 먼저 실행됩니다. 이를 통해 상호 의존적인 확장을 허용합니다. 이 필드의 기본값은 확장의 **manifest.yaml** 파일에 설정된 값이지만 사용자가 덮어쓸 수 있습니다.

1.19.5.1. ServiceMeshExtension 리소스 배포

ServiceMeshExtension 리소스를 사용하여 Red Hat OpenShift Service Mesh 확장을 활성화할 수 있습니다. 이 예제에서 istio-system 은 Service Mesh Control Plane 프로젝트의 이름입니다.



참고

새 WebAssembly 확장을 생성할 때ECDHEs mPlugin API를 사용합니다. ServiceMeshExtension API는 Red Hat OpenShift Service Mesh 버전 2.2에서 더 이상 사용되지 않으며 Red Hat OpenShift Service Mesh 버전 2.3에서 제거되었습니다.

Rust SDK를 사용하여 빌드된 전체 예제는 header-append-filter를 참조하십시오. 이는 확장의 config 필드에서 가져온 이름과 값을 사용하여 HTTP 응답에 하나 이상의 헤더를 추가하는 간단한 필터입니다. 아래 코드 조각의 샘플 구성을 참조하십시오.

절차

1. 다음 예제 리소스를 만듭니다.

ServiceMeshExtension 리소스 extensions.yaml의 예

```
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. 다음 명령을 사용하여 extensions.yaml 파일을 적용합니다.

```
$ oc apply -f <extension>.yaml
```

1.19.6. ServiceMeshExtension 에서 Juls mPlugin 리소스로 마이그레이션

Red Hat OpenShift Service Mesh 버전 2.2에서 더 이상 사용되지 않는 ServiceMeshExtension API는 Red Hat OpenShift Service Mesh 버전 2.3에서 제거되었습니다. ServiceMeshExtension API를 사용 중인 경우 WebAssembly 확장을 계속 사용하려면 VolumeSnapshot smPlugin API로 마이그레이션해야 합니다.

API는 매우 비슷합니다. 마이그레이션은 다음 두 단계로 구성됩니다.

1. 플러그인 파일 이름 변경 및 모듈 패키징 업데이트.

2. 업데이트된 컨테이너 이미지를 참조하는 **wasmPlugin** 리소스 생성.

1.19.6.1. API 변경

새로운 **WasmPlugin API**는 **ServiceMeshExtension** 와 유사하지만 특히 필드 이름에서 몇 가지 차이점이 있습니다.

표 **1.21. ServiceMeshExtensions** 와 **WasmPlugin**간의 필드 변경

ServiceMeshExtension	WasmPlugin
spec.config	**spec.pluginConfig**
spec.workloadSelector	**spec.selector**
spec.image	**spec.url**
spec.phase 유효한 값: PreAuthN, PostAuthN, PreAuthZ, PostAuthZ, PreStats, PostStats	**spec.phase** 유효한 값: <empty>, AUTHN, AUTHZ, protocolTS

다음은 **ServiceMeshExtension** 리소스를 와**s mPlugin** 리소스로 변환할 수 있는 방법 의 예입니다.

ServiceMeshExtension 리소스

```
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
```

위의 **ServiceMeshExtension**에 해당하는 새로운 **WasmPlugin** 리소스

```
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. 컨테이너 이미지 형식 변경

새로운 **ExtrasmPlugin** 컨테이너 이미지 형식은 **ServiceMeshExtensions** 와 유사하지만 다음과 같은 차이점이 있습니다.

- **ServiceMeshExtension** 컨테이너 형식은 컨테이너 파일 시스템의 루트 디렉터리에 **manifest.yaml** 이라는 메타데이터 파일이 필요했습니다. **Ex smPlugin** 컨테이너 형식에는 **manifest.yaml** 파일이 필요하지 않습니다.
- 이전에는 파일 이름이 있을 수 있는 **.wasm** 파일(실제 플러그인)의 이름은 **plugin.wasm** 이어야 하며 컨테이너 파일 시스템의 루트 디렉터리에 있어야 합니다.

1.19.6.3. WasmPlugin 리소스로 마이그레이션

ServiceMeshExtension API에서**ECDHEs mPlugin API**로 **WebAssembly** 확장을 업그레이드하려면 플러그인 파일의 이름을 바꿉니다.

사전 요구 사항

- ServiceMeshControlPlane 버전이 2.2 이상으로 업그레이드됩니다.

절차

1. 컨테이너 이미지를 업데이트합니다. 플러그인이 이미 컨테이너 내부 /plugin.wasm 에 있는 경우 다음 단계로 건너뜁니다. 그렇지 않은 경우:

   a. 플러그인 파일 이름이 plugin.wasm 인지 확인합니다. 확장 파일의 이름을 plugin.wasm 로 지정해야 합니다.

   b. 플러그인 파일이 루트(/) 디렉터리에 있는지 확인합니다. 확장 파일을 컨테이너 파일 시스템의 루트에 저장해야 합니다.

   c. 컨테이너 이미지를 다시 빌드하여 컨테이너 레지스트리로 푸시합니다.

2. ServiceMeshExtension 리소스를 제거하고 사용자가 빌드한 새 컨테이너 이미지를 참조하는wasmPlugin 리소스를 생성합니다.

1.20. 3SCALE WEBASSEMBLY 모듈 사용

참고

threescale-wasm-auth 모듈은 3scale API Management 2.11 이상과 Red Hat OpenShift Service Mesh 2.1.0 이후의 통합에서 실행됩니다.

threescale-wasm-auth 모듈은*ABI*(애플리케이션 바이너리 인터페이스)라는 인터페이스 집합을 사용하는 WebAssembly 모듈입니다. 이는 *Proxy-WASM* 사양에 따라 ABI를 구현하는 모든 소프트웨어를 구동하기 때문에 3scale에 대해 HTTP 요청을 승인할 수 있습니다.

ABI 사양으로, Proxy-WASM은 *host* 라는 소프트웨어와 이름이 지정된 다른 *모듈*,*프로그램* 또는 *확장 기능* 간의 상호 작용을 정의합니다. 호스트는 모듈에서 작업을 수행하는 데 사용하는 서비스 집합을 노출하고 이 경우 프록시 요청을 처리합니다.

호스트 환경은 소프트웨어 (이 경우 HTTP 프록시)와 상호 작용하는 WebAssembly 가상 머신으로 구성됩니다.

이 모듈은 가상 시스템에서 실행되는 명령과 **Proxy-WASM**에 의해 지정된 **ABI**를 제외하고 외부 세계와 분리하여 실행됩니다. 이는 확장 지점을 소프트웨어에 제공하는 안전한 방법입니다. 확장 기능은 가상 머신 및 호스트와 잘 정의된 방식으로만 상호 작용할 수 있습니다. 상호 작용은 컴퓨팅 모델과 프록시가 보유해야 하는 외부 세계와의 연결을 제공합니다.

1.20.1. 호환성

threescale-wasm-auth 모듈은 ***Proxy-WASM ABI*** 사양의 모든 구현과 완전히 호환되도록 설계되었습니다. 그러나 이 시점에서 **Envoy** 역방향 프록시에서 작동하도록 철저하게 테스트되었습니다.

1.20.2. 독립 실행형 모듈로 사용

자체 포함 설계 때문에 서비스 메시와 독립적으로 **Proxy-WASM** 프록시와 **3scale Istio** 어댑터 배포에서 작동하도록 이 모듈을 구성할 수 있습니다.

1.20.3. 사전 요구 사항

- 이 모듈은 **3scale 2.11** 이상이 필요한 **OpenID 연결(OIDC)** 을 사용하도록 서비스를 구성하는 경우를 제외하고 지원되는 모든 **3scale** 릴리스와 함께 작동합니다.

1.20.4. threescale-wasm-auth 모듈 구성

OpenShift Container Platform의 클러스터 관리자는 **3scale-wasm-auth** 모듈을 구성하여 **ABI**(애플리케이션 바이너리 인터페이스)를 통해 **HTTP** 요청을 **3scale API Management**에 인증할 수 있습니다. **ABI**는 호스트와 모듈 간의 상호 작용을 정의하고, 호스트 서비스를 노출하며, 모듈을 사용하여 프록시 요청을 처리할 수 있습니다.

1.20.4.1. wassmPlugin API 확장

서비스 메시는 프록시-**WASM** 확장을 지정하고 사이드카 프록시(pans **mPlugin** 이라고도 함)에 지정하는 사용자 정의 리소스 정의를 제공합니다. 서비스 메시는 **3scale**로 **HTTP API** 관리가 필요한 워크로드 세트에 이 사용자 정의 리소스를 적용합니다.

자세한 내용은 사용자 정의 리소스 정의를 참조하십시오.

참고

WebAssembly 확장 구성은 현재 수동 프로세스입니다. **3scale** 시스템에서 서비스 구성 가져오기 지원은 향후 릴리스에서 사용할 수 있습니다.

사전 요구 사항

- 이 모듈을 적용할 **Service Mesh** 배포에서 **Kubernetes** 워크로드 및 네임스페이스를 식별합니다.
- **3scale** 테넌트 계정이 있어야 합니다. 일치하는 서비스 및 관련 애플리케이션 및 메트릭이 포함된 **SaaS** 또는 **3scale 2.11 온-프레미스에서** 참조하십시오.
- **info** 네임스페이스의 **< product_page>** 마이크로 서비스에 모듈을 적용하면 **Bookinfo** 샘플 애플리케이션 을 참조하십시오.
  - 다음 예제는 **threescale-wasm-auth** 모듈에 대한 사용자 정의 리소스의 **YAML** 형식입니다. 이 예에서는 업스트림 **Maistra** 버전의 **Service Mesh,ECDHEs mPlugin API**를 참조합니다. 모듈이 적용할 애플리케이션 세트를 식별하는 선택기 와 함께 **threescale-wasm-auth** 모듈이 배포되는 네임스페이스를 선언해야 합니다.

```
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 네임스페이스.

2 선택기.

- spec.pluginConfig 필드는 모듈 구성에 따라 다르며 이전 예제에서 채워지지 않습니다. 대신 예제에서는 < yaml_configuration> 자리 표시자 값을 사용합니다. 이 사용자 정의 리소스 예제 형식을 사용할 수 있습니다.

  - spec.pluginConfig 필드는 애플리케이션에 따라 다릅니다. 다른 모든 필드는 이 사용자 정의 리소스의 여러 인스턴스에 걸쳐 유지됩니다. 예를 들면 다음과 같습니다.

    - url: 최신 버전의 모듈이 배포될 때만 변경됩니다.

    - 단계: 이 모듈은 동일한 상태를 유지합니다. 이 모듈은 프록시에서 모든 로컬 승인을 완료한 후 OIDC(OpenID Connect) 토큰 검증과 같이 호출해야 하므로 이 모듈을 호출해야 합니다.

- spec.pluginConfig 및 기타 사용자 정의 리소스에 모듈 구성이 있는 후 oc apply 명령과 함께 적용합니다.

```
$ oc apply -f threescale-wasm-auth-info.yaml
```

추가 리소스

- ServiceMeshExtension 에서ECDHEs mPlugin 리소스로 마이그레이션

- 사용자 정의 리소스

1.20.5. 3scale 외부 ServiceEntry 오브젝트 적용

threescale-wasm-auth 모듈에서 3scale에 대한 요청을 승인하려면 모듈이 3scale 서비스에 액세스할 수 있어야 합니다. HTTPS 프로토콜을 사용하도록 외부 ServiceEntry 오브젝트와 TLS 구성에 해당 DestinationRule 오브젝트를 적용하여 Red Hat OpenShift Service Mesh 내에서 이 작업을 수행할 수 있습니다.

CR(사용자 정의 리소스)은 서비스 관리 API 및 계정 관리 API의 백엔드 및 시스템 구성 요소에 대해 서비스 메시에서 3scale Hosted(SaaS)로 보안 액세스를 위한 서비스 항목 및 대상 규칙을 설정합니다. Service Management API는 각 요청의 권한 부여 상태에 대한 쿼리를 수신합니다. 계정 관리 API는 서비스에 대한 API 관리 구성 설정을 제공합니다.

절차

1. **3scale 호스팅 백엔드에 대해 다음 외부 ServiceEntry CR 및 관련 DestinationRule CR을 클러스터에 적용합니다.**

   a. **ServiceEntry CR을 service-entry-threescale-saas-backend.yml 이라는 파일에 추가합니다.**

   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 CR을 destination-rule-threescale-saas-backend.yml 이라는 파일에 추가합니다.**

   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.

다음 명령을 실행하여 **3scale** 호스팅 백엔드의 외부 **ServiceEntry CR**을 클러스터에 적용하고 저장합니다.

```
$ oc apply -f service-entry-threescale-saas-backend.yml
```

d.

다음 명령을 실행하여 **3scale Hosted** 백엔드의 외부 **DestinationRule CR**을 클러스터에 적용하고 저장합니다.

```
$ oc apply -f destination-rule-threescale-saas-backend.yml
```

2.

3scale Hosted 시스템에 다음 외부 **ServiceEntry CR** 및 관련 **DestinationRule CR**을 클러스터에 적용합니다.

a.

ServiceEntry CR을 **service-entry-threescale-saas-system.yml** 이라는 파일에 추가합니다.

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 CR을 **destination-rule-threescale-saas-system.yml** 이라는 파일에 추가합니다.

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. 다음 명령을 실행하여 **3scale Hosted** 시스템의 외부 **ServiceEntry CR**을 클러스터에 적용하고 저장합니다.

```
$ oc apply -f service-entry-threescale-saas-system.yml
```

d. 다음 명령을 실행하여 **3scale Hosted** 시스템의 외부 **DestinationRule CR**을 클러스터에 적용하고 저장합니다.

```
$ oc apply -f <destination-rule-threescale-saas-system.yml>
```

또는 **in-mesh 3scale** 서비스를 배포할 수 있습니다. **in-mesh 3scale** 서비스를 배포하려면 **3scale**을 배포하고 배포에 연결하여 **CR**의 서비스 위치를 변경합니다.

추가 리소스

- 서비스 항목 및 대상 규칙 문서

1.20.6. 3scale WebAssembly 모듈 구성

ECDHE smPlugin 사용자 정의 리소스 사양은 **Proxy-WASM** 모듈이 읽을 수 있는 구성을 제공합니다.

사양은 호스트에 포함되어 있으며 **Proxy-WASM** 모듈에서 읽습니다. 일반적으로 구성은 모듈에서 구문 분석하기 위한 **JSON** 파일 형식이지만**ECDHEs mPlugin** 리소스는 **spec** 값을 **YAML**로 해석하고 모듈

에서 사용할 수 있도록 JSON으로 변환할 수 있습니다.

독립 실행형 모드에서 프록시-WASM 모듈을 사용하는 경우 JSON 형식을 사용하여 구성을 작성해야 합니다. JSON 형식을 사용하는 것은 호스트 구성 파일(예: Envoy) 내에서 필요한 경우 이스케이프 및 quoting을 사용하는 것을 의미합니다. Requires mPlugin 리소스와 함께 WebAssembly 모듈을 사용하면 구성이 YAML 형식으로 되어 있습니다. 이 경우 유효하지 않은 구성으로 인해 모듈에서 사이드카의 로깅 스트림에 대한 JSON 표현을 기반으로 진단 정보를 표시하게 됩니다.

중요

EnvoyFilter 사용자 정의 리소스는 일부 3scale Istio 어댑터 또는 서비스 메시 릴리스에서 사용할 수 있지만 지원되는 API가 아닙니다. EnvoyFilter 사용자 정의 리소스를 사용하는 것은 권장되지 않습니다. EnvoyFilter 사용자 정의 리소스 대신ECDHEs mPlugin API를 사용합니다. EnvoyFilter 사용자 지정 리소스를 사용해야 하는 경우 사양을 JSON 형식으로 지정해야 합니다.

1.20.6.1. 3scale WebAssembly 모듈 구성

3scale WebAssembly 모듈 구성의 아키텍처는 3scale 계정 및 권한 부여 서비스와 처리할 서비스 목록에 따라 다릅니다.

사전 요구 사항

사전 요구 사항은 모든 경우에서 최소 필수 필드 집합입니다.

- 3scale 계정 및 권한 부여 서비스의 경우 backend-listener URL입니다.
- 처리할 서비스 목록: 서비스 ID와 하나 이상의 인증 정보는 방법 및 찾을 위치를 찾습니다.
- userkey, appkey,appid 및 OpenID Connect (OIDC) 패턴을 다루는 예제를 찾을 수 있습니다.
- WebAssembly 모듈은 정적 구성에서 지정한 설정을 사용합니다. 예를 들어, 모듈에 매핑 규칙 구성을 추가하는 경우 3scale 관리 포털에 이러한 매핑 규칙이 없는 경우에도 항상 적용됩니다. pasts mPlugin 리소스의 나머지 부분은 spec.pluginConfig YAML 항목과 관련이 있습니다.

1.20.6.2. 3scale WebAssembly 모듈 api 오브젝트

3scale WebAssembly 모듈의 **API** 최상위 문자열은 모듈에서 사용할 구성 버전을 정의합니다.

참고

존재하지 않거나 지원되지 않는 **api** 오브젝트 버전은 **3scale WebAssembly** 모듈을 작동 불가능하게 렌더링합니다.

API 최상위 문자열 예

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
  namespace: <info>
spec:
  pluginConfig:
    api: v1
...
```

api 항목은 구성의 나머지 값을 정의합니다. 허용되는 유일한 값은 **v1** 입니다. 현재 구성과의 호환성을 중단하거나 **v1** 을 사용하는 모듈이 처리할 수 없는 더 많은 논리가 필요한 새로운 설정에서는 다른 값이 필요합니다.

1.20.6.3. 3scale WebAssembly 모듈 시스템 오브젝트

시스템 최상위 오브젝트는 특정 계정에 대한 **3scale** 계정 관리 **API**에 액세스하는 방법을 지정합니다. **upstream** 필드는 오브젝트의 가장 중요한 부분입니다. 시스템 오브젝트는 선택 사항이지만 **3scale**의 *시스템* 구성 요소에 대한 연결을 제공하지 않으려면 **3scale WebAssembly** 모듈에 완전히 고정 구성을 제공하지 않는 것이 좋습니다.

시스템 오브젝트 외에 정적 구성 오브젝트를 제공하는 경우 정적 구성이 항상 우선합니다.

```
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
  ...
```

표 1.22. 시스템 오브젝트 필드

이름	설명	필수 항목
name	3scale 서비스의 식별자로, 현재 다른 위치에서 참조되지 않습니다.	선택 사항
upstream	연결할 네트워크 호스트에 대한 세부 정보입니다. **업스트림** 은 시스템이라는 3scale 계정 관리 API 호스트를 나타냅니다.	있음
token	읽기 권한이 있는 3scale 개인 액세스 토큰입니다.	있음
ttl	새 변경 사항을 가져오기 전에 이 호스트에서 검색한 구성을 유효한 것으로 간주하는 최소 시간(초)입니다. 기본값은 600초(10분)입니다. **참고:** 최대 용량은 없지만 모듈은 일반적으로 이 TTL이 경과한 후 적절한 시간 내에 모든 구성을 가져옵니다.	선택 사항

1.20.6.4. 3scale WebAssembly 모듈 업스트림 오브젝트

업스트림 오브젝트는 프록시에서 호출을 수행할 수 있는 외부 호스트를 설명합니다.

```
apiVersion: maistra.io/v1
upstream:
  name: outbound|443||multitenant.3scale.net
  url: "https://myaccount-admin.3scale.net/"
  timeout: 5000
...
```

표 1.23. 업스트림 오브젝트 필드

이름	설명	필수 항목

이름	설명	필수 항목
name	**이름은 무료 형식 식별자가 아닙니다.Name** is not a free-form identifier. 프록시 구성에 정의된 대로 외부 호스트의 식별자입니다. 독립 실행형 **Envoy** 구성의 경우 다른 프록시에서 **업스트림** 이라고도 하는 클러스터 의 이름에 매핑됩니다. **참고:** 서비스 메시 및 3scale Istio 어댑터 컨트롤 플레인은 여러 필드의 구분자로 수직 표시줄(|)을 사용하여 형식에 따라 이름을 구성하기 때문에 이 필드의 값입니다. 이 통합을 위해 항상 format: **outbound|<port>|| <hostname> 을 사용합니다**.	있음
url	설명된 서비스에 액세스하기 위한 전체 URL입니다. 스키마에서 제외하지 않는 한 TCP 포트를 포함해야 합니다.	있음
Timeout	응답 시간을 초과하는 이 서비스에 대한 연결이 오류로 간주되도록 시간 초과(밀리초)입니다. 기본값은 1000초입니다.	선택 사항

1.20.6.5. 3scale WebAssembly 모듈 백엔드 오브젝트

backend 최상위 오브젝트는 HTTP 요청을 승인 및 보고하기 위해 3scale Service Management API에 액세스하는 방법을 지정합니다. 이 서비스는 3scale의 *백엔드* 구성 요소에서 제공합니다.

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
    backend:
      name: backend
      upstream: <object>
    ...
```

표 1.24. 백엔드 오브젝트 필드

이름	설명	필수 항목
name	3scale 백엔드의 식별자로, 현재 다른 위치에서 참조되지 않습니다.	선택 사항
upstream	연결할 네트워크 호스트에 대한 세부 정보입니다. 이는 알려진 3scale 계정 관리 API 호스트를 참조해야 합니다.	네, 필요합니다. 가장 중요하고 필요한 필드입니다.

1.20.6.6. 3scale WebAssembly 모듈 서비스 오브젝트

최상위 개체는 모듈의 이 특정 인스턴스에서 처리하는 서비스 식별자를 지정합니다.

계정에는 여러 서비스가 있으므로 처리할 서비스를 지정해야 합니다. 나머지 구성에서는 서비스 구성 방법을 확인할 수 있습니다.

services 필드는 필수입니다. 유용하게 사용할 수 있는 하나 이상의 서비스를 포함해야 하는 배열입니다.

```
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 배열의 각 요소는 **3scale** 서비스를 나타냅니다.

표 1.25. Service 오브젝트 필드

이름	설명	필수 항목
ID	이 3scale 서비스의 식별자로, 현재 다른 위치에서 참조되지 않습니다.	있음
token	이 **토큰** 은 시스템에서 서비스의 프록시 구성에서 확인하거나 다음 **curl** 명령을 사용하여 시스템에서 해당 토큰을 검색할 수 있습니다. **curl://<system_host>/admin/api/services/<service_id>/proxy/configs/production/latest.json?access_token=<access_token>" | jq '.proxy_config.content.backend_authentication_value**	선택 사항
권한 부여	문자열 배열, 각 문자열은 일치시킬 *URL 의 권한을* 나타냅니다. 이 문자열은 별표(*), 더하기 기호(+), 물음표(??) 일치자를 지원하는 글러 패턴을 허용합니다.	있음
credentials	검색할 자격 증명을 정의하는 오브젝트이며 어디에서 찾을 수 있습니다.	있음
mapping_rules	매핑 규칙 및 3scale 메서드를 나타내는 개체의 배열입니다.	선택 사항

1.20.6.7. 3scale WebAssembly 모듈 인증 정보 오브젝트

credentials 오브젝트는 서비스 오브젝트의 구성 요소입니다. Credentials(자격 증명)는 조회할 자격 증명 종류 및 이 작업을 수행하는 단계를 지정합니다.

모든 필드는 선택 사항이지만 하나 이상의 user_key 또는 app_id 를 지정해야 합니다. 각 인증 정보를 지정하는 순서는 모듈에 의해 미리 설정되므로 관련이 없습니다. 각 인증 정보의 인스턴스 하나만 지정합니다.

```
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>
  ...
```

표 1.26. 인증 정보 오브젝트 필드

이름	설명	필수 항목
user_key	3scale 사용자 키를 정의하는 조회 쿼리의 배열입니다. 사용자 키는 일반적으로 API 키라고 합니다.	선택 사항
app_id	3scale 애플리케이션 식별자를 정의하는 조회 쿼리의 배열입니다. 애플리케이션 식별자는 3scale 또는 Red Hat Single Sign-On(RH-SSO) 또는 OpenID Connect(OIDC)와 같은 ID 공급자를 사용하여 제공됩니다. 여기에 지정된 조회 쿼리의 해결 방법은 성공하고 두 개의 값으로 확인되면 **app_id** 와 **app_key** 를 설정합니다.	선택 사항
app_key	3scale 애플리케이션 키를 정의하는 조회 쿼리의 배열입니다. 확인된 **app_id** 가 없는 애플리케이션 키는 쓸모가 없으므로 **app_id** 가 지정된 경우에만 이 필드를 지정합니다.	선택 사항

1.20.6.8. 3scale WebAssembly 모듈 조회 쿼리

조회 쿼리 오브젝트는 **credentials** 오브젝트의 필드의 일부입니다. 지정된 인증 정보 필드를 찾아서 처리하는 방법을 지정합니다. 평가되면 성공적인 확인은 하나 이상의 값을 찾을 수 있음을 의미합니다. 해결 실패는 값을 찾을 수 없음을 의미합니다.

조회 쿼리 배열은 단기 또는 관계를 설명합니다. 쿼리 중 하나를 성공적으로 해결하면 나머지 쿼리의 평가가 중지되고 값 또는 값을 지정된 **Credential-type**에 할당합니다. 배열의 각 쿼리는 서로 독립적입니다.

조회 쿼리 는 여러 소스 유형 중 하나일 수 있는 단일 필드인 소스 오브젝트로 구성됩니다. 다음 예제를 참조하십시오.

```
apiVersion: extensions.istio.io/v1alpha1
```

```
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
    services:
    - credentials:
        user_key:
          - <source_type>: <object>
          - <source_type>: <object>
          ...
        app_id:
          - <source_type>: <object>
          ...
        app_key:
          - <source_type>: <object>
          ...
    ...
```

1.20.6.9. 3scale WebAssembly 모듈 소스 오브젝트

소스 오브젝트는 **credentials** 오브젝트 필드 내에 있는 소스 배열의 일부로 존재합니다. **source-type** 이라고 하는 오브젝트 필드 이름은 다음 중 하나입니다.

- **headers:** 조회 쿼리는 **HTTP** 요청 헤더를 입력으로 수신합니다.
- **QUERY_STRING:** 조회 쿼리 는 **URL** 쿼리 문자열 매개변수를 입력으로 수신합니다.
- **filter:** 조회 쿼리 는 필터 메타데이터를 입력으로 수신합니다.

모든 **source-type** 오브젝트에는 다음 두 개 이상의 필드가 있습니다.

표 **1.27. Source-type** 오브젝트 필드

이름	설명	필수 항목
keys	문자열 배열, 각 **키는** 입력 데이터에 있는 항목을 나타냅니다.	있음

이름	설명	필수 항목
ops	**키** 항목을 수행하는 **작업** 배열입니다. 배열은 작업이 입력을 수신하고 다음 작업에서 출력을 생성하는 파이프라인입니다. 출력을 제공하지 못하는 **작업으로** 인해 **조회 쿼리**가 실패로 확인됩니다. 작업의 파이프라인 순서에 따라 평가 순서가 결정됩니다.	선택 사항

필터 필드 이름에는 데이터를 조회하는 데 사용하는 메타데이터에 경로를 표시하는 데 필요한 경로 항목이 있습니다.

키가 입력 데이터와 일치하면 나머지 키도 평가되지 않으며 소스 확인 알고리즘은 지정된 작업 (ops) (있는 경우)을 실행하도록 건너뜁니다. ops 를 지정하지 않으면 일치하는 키 의 결과 값(있는 경우)이 반환됩니다.

작업을 통해 첫 번째 단계에서 키 를 조회한 후 보유한 입력에 대한 특정 조건 및 변환을 지정할 수 있습니다. 속성을 변환, 디코딩 및 어설션해야 할 때 작업을 사용하지만 모든 요구 사항을 처리하기 위해 성숙한 언어를 제공하지 않으며 *Turing-completeness* 가 부족합니다.

스택에 작업 출력이 저장되었습니다. 평가 시 인증 정보에서 사용하는 값 수에 따라 스택 하단의 값 또는 값을 할당하는 방식으로 조회 쿼리 가 완료됩니다.

1.20.6.10. 3scale WebAssembly 모듈 작업 오브젝트

특정 소스 유형에 속하는 ops 배열의 각 요소는 값에 변환을 적용하거나 테스트를 수행하는 작업 오브젝트입니다. 이러한 개체에 사용할 필드 이름은 작업 자체의 이름이고, 모든 값은 작업 자체의 매개 변수이며, 이 매개 변수는 작업 개체의 매개 변수이며, 예를 들어 필드 및 값, 목록 또는 문자열로 구조 개체일 수 있습니다.

대부분의 작업은 하나 이상의 입력에 참여하고 하나 이상의 출력을 생성합니다. 입력을 사용하거나 출력을 생성할 때 값 스택을 사용하여 작업합니다. 작업에서 사용하는 각 값은 값 스택에서 채워지고 처음에는 소스 일치로 채워집니다. 해당 값으로 출력되는 값은 스택으로 푸시됩니다. 다른 작업에서는 특정 속성을 어설션하는 이외의 출력을 사용하거나 생성하지 않지만 값 스택을 검사합니다.

참고

해결이 완료되면 값을 **app_id,app_key** 또는 **user_key** 로 할당하는 등 다음 단계에서 선택한 값은 스택의 하단 값에서 가져옵니다.

몇 가지 작업 범주가 있습니다.

- **decode:**이는 다른 형식을 얻기 위해 디코딩하여 입력 값을 변환합니다.
- **string** 값은 입력으로 사용하고 변환 및 검사를 수행합니다.
- **stack:** 이 입력에서 값 집합을 가져와서 스택의 특정 위치에 대한 여러 스택 변환 및 선택을 수행합니다.
- **Check:**이는 부작용의 무료 방식으로 작업 세트에 대한 어설션 속성을 나타냅니다.
- **Control:** 평가 흐름을 수정할 수 있는 작업을 수행합니다.
- **Format:** 입력 값의 형식별 구조를 구문 분석하고 값을 찾습니다.

모든 작업은 이름 식별자에 의해 문자열로 지정됩니다.**All operations are specified by the name identifiers as strings.**

추가 리소스

- 사용 가능한 작업

1.20.6.11. 3scale WebAssembly 모듈 mapping_rules 오브젝트

mapping_rules 오브젝트는 서비스 오브젝트의 일부입니다. 이는 **REST** 경로 패턴 세트 및 관련 **3scale** 메트릭 및 패턴 일치 시 사용할 수를 지정합니다.

시스템 최상위 오브젝트에 동적 구성이 제공되지 않는 경우 값이 필요합니다. 오브젝트가 시스템 최상

위 수준 항목 외에 제공되는 경우 **mapping_rules** 오브젝트가 먼저 평가됩니다.

mapping_rules 는 배열 오브젝트입니다. 해당 배열의 각 요소는 **mapping_rule** 오브젝트입니다. 들어오는 요청에 대한 평가 일치 매핑 규칙은 ***APIManager*** 에 대한 권한 부여 및 보고를 위한 **3scale** 메서드 세트를 제공합니다. 여러 일치 규칙이 동일한 메서드 를 참조하는 경우 **3scale**로 호출할 때 **deltas** 의 합계가 있습니다. 예를 들어, 두 규칙이 **deltas** 가 **1**과 **3**인 경우 ***Hits*** 보고를 **3scale**로 사용하여 **Hits** 메서드를 두 번 늘리는 경우 **delta** 는 **4**입니다.

1.20.6.12. 3scale WebAssembly 모듈 mapping_rule 오브젝트

mapping_rule 오브젝트는 **mapping_rules** 오브젝트에서 배열의 일부입니다.

mapping_rule 개체 필드는 다음 정보를 지정합니다.

- 일치시킬 ***HTTP 요청 메서드*** 입니다.
- 경로와 일치하는 패턴입니다.
- 보고할 양과 함께 보고할 **3scale** 방법입니다. 필드를 지정하는 순서에 따라 평가 순서가 결정됩니다.

표 1.28. mapping_rule 오브젝트 필드

이름	설명	필수 항목
method	동사라고도 하는 HTTP 요청 메서드를 나타내는 문자열을 지정합니다. 허용되는 값은 허용되는 HTTP 메서드 이름 중 대소문자를 구분하지 않습니다. 모든 항목의 특수 값은 모든 메서드와 일치합니다.	있음
패턴	HTTP 요청의 URI 경로 구성 요소와 일치하는 패턴입니다. 이 패턴은 3scale에 의해 문서화된 것과 동일한 구문을 따릅니다. **{this}** 와 같은 중괄호 사이의 모든 문자 시퀀스를 사용하여 와일드카드(* 문자 사용)를 허용합니다.	있음

이름	설명	필수 항목
사용법	**사용** 오브젝트 목록입니다. 규칙이 일치하면 **deltas** 가 있는 모든 메서드가 권한 부여 및 보고를 위해 3scale로 전송되는 방법 목록에 추가됩니다. 다음 필수 필드를 사용하여 **usages** 오브젝트를 포함합니다. ● **name**: 보고할 **메서드** 시스템 이름입니다. ● **delta**: 이 방법을 최대로 늘릴 수 있습니다.	있음
last	이 규칙의 성공적인 일치 여부를 통해 더 많은 매핑 규칙의 평가를 중지해야 합니다.	선택적 부울입니다. 기본값은 **false**입니다.

다음 예제는 3scale의 메서드 간 기존 계층과 독립적입니다. 즉, 3scale 측에서 실행되는 모든 것은 이에 영향을 미치지 않습니다. 예를 들어 *Hits* 지표는 모두 상위 항목일 수 있으므로 인증된 요청에 보고된 모든 메서드의 합계로 인해 4개의 히트를 저장하고 3scale Authrep API 엔드포인트를 호출합니다.

아래 예제에서는 모든 규칙과 일치하는 경로에 대한 GET 요청을 사용합니다.

mapping_rules GET 요청 예

```
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
  ...
```

모듈은 다음과 같이 사용 데이터를 사용하여 **3scale**에 수행하는 모든 사용량 을 요청에 추가됩니다.

- 적중: 1
- 제품: 2
- 영업: 1

1.20.7. 인증 정보 사용 사례에 대한 3scale WebAssembly 모듈 예

서비스에 대한 요청의 자격 증명을 얻기 위해 구성 단계를 적용하는 데 대부분의 시간을 사용합니다.

다음은 특정 사용 사례에 맞게 수정할 수 있는 자격 증명 예제입니다.

여러 소스 개체를 자체 조회 쿼리 로 지정하는 경우 둘 중 하나가 성공적으로 해결될 때까지 순서대로 평가되지만 모두 결합할 수 있습니다.

1.20.7.1. 쿼리 문자열 매개변수의 API 키(user_key)

다음 예제에서는 동일한 이름의 쿼리 문자열 매개변수 또는 헤더에서 **user_key** 를 조회합니다.

-

```
credentials:
 user_key:
  - query_string:
      keys:
       - user_key
  - header:
      keys:
       - user_key
```

1.20.7.2. 애플리케이션 ID 및 키

다음 예제에서는 쿼리 또는 헤더에서 **app_key** 및 **app_id** 자격 증명을 조회합니다.

```
credentials:
 app_id:
  - header:
      keys:
       - app_id
  - query_string:
      keys:
       - app_id
 app_key:
  - header:
      keys:
       - app_key
  - query_string:
      keys:
       - app_key
```

1.20.7.3. 권한 부여 헤더

요청에는 권한 부여 헤더에 **app_id** 및 **app_key** 가 포함됩니다. 끝에 출력된 값이 하나 이상 있는 경우 **app_key** 를 할당할 수 있습니다.

여기에서 해상도는 끝에 하나 또는 두 개의 출력이 있는 경우 **app_key** 를 할당합니다.

authorization 헤더는 권한 부여 유형의 값을 지정하고 해당 값은 **Base64** 로 인코딩됩니다. 즉, 값을 공백 문자로 분할하고 두 번째 출력을 가져온 다음 콜론(**:**)을 구분자로 사용하여 다시 분할할 수 있습니다. 예를 들어 **app_id:app_key** 형식을 사용하는 경우 헤더는 인증 정보에 대한 다음 예제와 유사합니다.

```
aladdin:opensesame:  Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
```

다음 예와 같이 소문자 헤더 필드 이름을 사용해야 합니다.

```
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
```

이전 예제 사용 사례에서 권한 부여의 헤더를 확인합니다.

1. 문자열 값을 가져와 공백으로 분할하고 인증 정보 **-type**과 인증 정보자체의 두 개 이상의 값이 생성되었는지 확인한 다음 **Credential -type**을 삭제합니다.

2. 그런 다음 필요한 데이터가 포함된 두 번째 값을 디코딩하고 콜론(:) 문자를 사용하여 **app_id** 첫 번째를 포함한 작업 스택을 찾은 다음 **app_key** 가 존재하는 경우 해당 값을 분할합니다.

   a. **app_key** 가 권한 부여 헤더에 없으면 특정 소스(예: 이 경우 키 **app_key**)가 있는 헤더가 확인됩니다.

3. 자격 증명에 조건을 추가하려면 기본 권한 부여를 허용합니다. 여기서 **app_id** 는 **aladdin** 또는 **admin** 이거나 **app_id** 는 최소 **8**자 이상입니다.

4. **app_key** 는 다음 예와 같이 값을 포함해야 하며 최소 **64**자여야 합니다.

```
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. 권한 부여 헤더 값을 선택한 후 유형을 맨 위에 배치하도록 스택을 재검토하여 기본 인증 정보-type을 가져옵니다.

6. 여기에서 일치하는 글을 실행합니다. 유효성이 검사되고 인증 정보가 디코딩되고 분할되면 스택 하단에서 app_id 및 top의 app_key 가 발생할 수 있습니다.

7. 테스트 실행: 스택에 두 개의 값이 있는 경우 app_key 가 획득되었음을 의미합니다.

   a. app_id 와 app_key 를 포함하여 문자열 길이가 1에서 63 사이인지 확인합니다. 키의 길이가 0이면 키를 놓고 키가 없는 것처럼 계속합니다. app_id 만 있고 app_key 가 없는 경우 누락된 다른 분기는 성공적인 테스트 및 평가가 계속됨을 나타냅니다.

마지막 작업 assert 는 부작용이 스택에 해당하지 않음을 나타냅니다. 그런 다음 스택을 수정할 수 있

습니다.

1. 맨 위에 app_id 를 갖도록 스택을 되돌립니다.

   a. app_key 가 있는지 여부에 관계없이 스택을 되돌리면 app_id 가 맨 위에 있는지 확인합니다.

2. 및 을 사용하여 테스트에서 스택의 콘텐츠를 유지합니다.

   다음 옵션 중 하나를 사용하십시오.

   - app_id 의 문자열 길이는 8입니다.
   - app_id 가 aladdin 또는 admin 과 일치하는지 확인합니다.

1.20.7.4. OpenID Connect (OIDC) 사용 사례

서비스 메시 및 3scale Istio 어댑터의 경우 다음 예에 표시된 대로 RequestAuthentication 을 배포하여 자체 워크로드 데이터 및 jwtRules 를 작성해야합니다.

```
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 을 적용하면 JWT 토큰을 검증하기 위해 네이티브 플러그인으로 Envoy 를 구성합니다. 프록시는 모듈을 실행하기 전에 모든 항목을 검증하므로 실패한 모든 요청은 3scale WebAssembly 모듈에 만들지 않습니다.

JWT 토큰이 검증되면 프록시는 해당 콘텐츠를 플러그인의 특정 구성에 종속된 항목으로 내부 메타데이터 오브젝트에 저장합니다. 이 사용 사례에서는 알 수 없는 키 이름이 포함된 단일 항목으로 구조 오브젝트를 조회할 수 있는 기능을 제공합니다.

OIDC의 3scale app_id 는 OAuth client_id 와 일치합니다. 이는 JWT 토큰의 azp 또는 aud 필드에서 확인할 수 있습니다.

Envoy의 네이티브 JWT 인증 필터에서 app_id 필드를 가져오려면 다음 예제를 참조하십시오.

```
credentials:
 app_id:
  - filter:
      path:
       - envoy.filters.http.jwt_authn
       - "0"
      keys:
       - azp
       - aud
      ops:
       - take:
           head: 1
```

이 예제에서는 모듈에서 filter 소스 유형을 사용하여 Envoy-specific JWT 인증 네이티브 플러그인에서 오브젝트에 대한 필터 메타데이터를 조회하도록 지시합니다. 이 플러그인에는 단일 항목이 있고 사전 구성된 이름이 있는 구조 오브젝트의 일부로 JWT 토큰이 포함됩니다. 0 을 사용하여 단일 항목에만 액세스하도록 지정합니다.

결과 값은 다음 두 필드를 해결할 구조입니다.

- azp: app_id 가 발견된 값입니다.
- AUD: 이 정보를 찾을 수 있는 값입니다.

이 작업을 수행하면 할당에 대해 하나의 값만 유지됩니다.

1.20.7.5. 헤더에서 JWT 토큰 가져오기

일부 설정에는 유효성 검사 토큰이 JSON 형식의 헤더를 통해 이 모듈에 도달하는 JWT 토큰에 대한 검증 프로세스가 있을 수 있습니다.

app_id 를 가져오려면 다음 예제를 참조하십시오.

```
credentials:
  app_id:
    - header:
        keys:
          - x-jwt-payload
        ops:
          - base64_urlsafe
          - json:
            - keys:
              - azp
              - aud
          - take:
              head: 1
```

1.20.8. 3scale WebAssembly 모듈 최소 작업 구성

다음은 3scale WebAssembly 모듈 최소 작업 구성의 예입니다. 이 값을 복사하여 붙여넣고 편집하여 사용자 구성으로 작업할 수 있습니다.

```
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 어댑터 사용

3scale Istio Adapter는 Red Hat OpenShift Service Mesh 내에서 실행되는 서비스에 레이블을 지정하고 해당 서비스를 3scale API 관리 솔루션과 통합할 수 있는 선택적 어댑터입니다. Red Hat OpenShift Service Mesh에는 필요하지 않습니다.

중요

Red Hat OpenShift Service Mesh 버전 2.0 이상에서만 3scale Istio 어댑터를 사용할 수 있습니다. Mixer 구성 요소는 릴리스 2.0에서 더 이상 사용되지 않으며 릴리스 2.1에서 제거되었습니다. Red Hat OpenShift Service Mesh 버전 2.1.0 이상의 경우 3scale WebAssembly 모듈을 사용해야 합니다.

3scale Istio 어댑터로 3scale 백엔드 캐시를 활성화하려면 Mixer 정책 및 Mixer Telemetry도 활성화해야 합니다. Red Hat OpenShift Service Mesh Control Plane 배포를 참조하십시오.

1.21.1. Red Hat OpenShift Service Mesh와 3scale 어댑터 통합

이러한 예제를 사용하여 3scale Istio 어댑터로 서비스에 대한 요청을 구성할 수 있습니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh 버전 2.x
- 작업 중인 3scale 계정(SaaS 또는 3scale 2.9 on-Premises)
- 백엔드 캐시를 활성화하려면 3scale 2.9 이상 필요
- Red Hat OpenShift Service Mesh 사전 요구 사항
- Mixer 정책 적용이 활성화되었는지 확인합니다. Mixer 정책 시행 업데이트 섹션에서는 현재 Mixer 정책 시행 상태를 확인하고 정책 시행을 활성화하는 지침을 제공합니다.
- mixer 플러그인을 사용하는 경우 Mixer 정책 및 Telemetry를 활성화해야 합니다.
  - 업그레이드 시 SMCP(Service Mesh Control Plane)를 올바르게 구성해야 합니다.

참고

3scale Istio Adapter를 구성하려면 사용자 정의 리소스 파일에 어댑터 매개변수를 추가하는 방법에 대한 Red Hat OpenShift Service Mesh 사용자 정의 리소스를 참조하십시오.

참고

특히 kind: handler 리소스에 주의하십시오. 3scale 계정 인증 정보로 업데이트해야 합니다. 선택적으로 service_id를 처리기에 추가할 수 있지만 3scale 계정의 하나의 서비스에만 처리기를 렌더링하므로 이전 버전과의 호환성을 위해서만 유지됩니다. service_id를 처리기에 추가하는 경우 다른 서비스에 3scale을 활성화하려면 다른 service_ids로 더 많은 처리기를 생성해야 합니다.

아래 단계에 따라 3scale 계정당 단일 처리기를 사용합니다.

절차

1. 3scale 계정에 대한 처리기를 생성하고 계정 인증 정보를 지정합니다. 서비스 식별자를 생략합니다.

```
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"
```

필요한 경우, 3scale 구성에서 제공하는 URL을 재정의하기 위해 *params* 섹션에 backend_url 필드를 제공할 수 있습니다. 어댑터가 3scale 온프레미스 인스턴스와 동일한 클러스터에서 실행되고 내부 클러스터 DNS를 사용하려는 경우 유용할 수 있습니다.

2. 다음과 같이 3scale 계정에 속하는 서비스의 배포 리소스를 편집하거나 패치합니다.

   a. 유효한 service_id에 해당하는 값을 사용하여 "service-mesh.3scale.net/service-id" 레이블을 추가합니다.

   b. 1단계에서 *처리기 리소스의 이름*이 값이 되도록 "service-mesh.3scale.net/credentials" 레이블을 추가합니다.

3. 더 많은 서비스를 추가하려는 경우 2단계를 수행하여 3scale 계정 인증 정보 및 서비스 식별자에 연결합니다.

4. 3scale 구성으로 규칙 구성을 수정하여 3scale 처리기에 규칙을 전송합니다.

규칙 구성 예

-

```
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 사용자 정의 리소스 생성

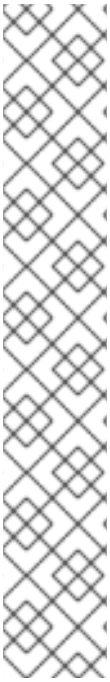
어댑터에는 handler, instance, rule 사용자 정의 리소스를 생성할 수 있는 도구가 포함되어 있습니다.

표 1.29. 사용법

옵션	설명	필수 항목	기본값
-h, --help	사용 가능한 옵션에 대한 도움말 출력 생성	아니요	
--name	이 URL의 고유 이름, 토큰 쌍	예	
-n, --namespace	템플릿을 생성할 네임스페이스	아니요	istio-system
-t, --token	3scale 액세스 토큰	예	
-u, --url	3scale 관리자 포털 URL	예	
--backend-url	3scale 백엔드 URL. 설정하면 시스템 설정에서 읽은 값을 재정의합니다.	아니요	
-s, --service	3scale API/서비스 ID	아니요	
--auth	지정을 위한 3scale 인증 패턴(1=API Key, 2=App Id/App Key, 3=OIDC)	아니요	하이브리드
-o, --output	생성된 매니페스트를 저장할 파일	아니요	표준 출력

옵션	설명	필수 항목	기본값
--version	CLI 버전을 출력하고 즉시 종료합니다.	아니요	

1.21.1.1.1. URL 예제에서 템플릿 생성



참고

- 배포된 어댑터의 매니페스트를 생성하며 **3scale** 어댑터 컨테이너 이미지에서 **oc exec**를 통해 다음 명령을 실행합니다.
- **3scale-config-gen** 명령을 사용하여 **YAML** 구문 및 들여쓰기 오류를 방지할 수 있습니다.
- 주석을 사용하는 경우 **--service**를 생략할 수 있습니다.
- 이 명령은 **oc exec**를 통해 컨테이너 이미지 내에서 호출해야 합니다.

절차

- **3scale-config-gen** 명령을 사용하여 토큰, **URL** 쌍을 단일 처리기로 여러 서비스에서 공유할 수 있도록 템플릿 파일을 자동 생성합니다.

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

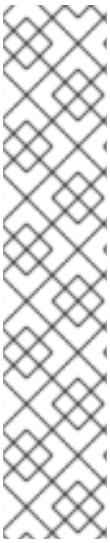
- 다음 예제에서는 처리기에 포함된 서비스 **ID**로 템플릿을 생성합니다.

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

추가 리소스

- 토큰.

1.21.1.2. 배포된 어댑터에서 매니페스트 생성



참고

- NAME은 3scale로 관리 중인 서비스와 식별하는 데 사용하는 식별자입니다.
- CREDENTIALS_NAME 참조는 규칙 구성의 match 섹션에 해당하는 식별자입니다. CLI 툴을 사용하는 경우 NAME 식별자로 자동 설정됩니다.
- 해당 값은 특정할 필요가 없습니다. 레이블 값은 규칙의 내용과 일치해야 합니다. 자세한 정보는 어댑터를 통한 서비스 트래픽 라우팅을 참조하십시오.

1. 이 명령을 실행하여 istio-system 네임스페이스의 배포된 어댑터에서 매니페스트를 생성합니다.

```
$ 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. 터미널에 샘플 출력이 생성됩니다. 필요한 경우 이러한 샘플을 편집하고 oc create 명령을 사용하여 오브젝트를 생성합니다.

3. 요청이 어댑터에 도달하면 어댑터는 서비스가 3scale의 API에 매핑되는 방식을 알아야 합니다. 다음 두 가지 방법으로 이러한 정보를 제공할 수 있습니다.

   a. 워크로드에 레이블 지정(권장)

   b. 처리기를 service_id로 하드 코딩

4. 필요한 주석으로 워크로드를 업데이트합니다.

참고

처리기에 아직 포함되지 않은 경우, 이 예제에 제공된 서비스 **ID**만 업데이트해야 합니다. 처리기의 설정이 우선합니다.

```
$ 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. 어댑터를 통한 서비스 트래픽 라우팅

3scale 어댑터를 통해 서비스 트래픽을 유도하려면 다음 단계를 따르십시오.

사전 요구 사항

- **3scale** 관리자의 인증 정보 및 서비스 **ID**

절차

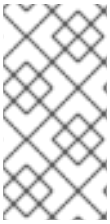
1. **kind: rule** 리소스의 구성에서 이전에 생성한 **destination.labels["service-mesh.3scale.net/credentials"] == "threescale"** 규칙과 일치합니다.

2. 서비스를 통합하기 위해 대상 워크로드 배포에서 위의 레이블을 **PodTemplateSpec**에 추가합니다. **threescale** 값은 생성된 처리기의 이름을 나타냅니다. 이 처리기에서는 **3scale**를 호출하는 데 필요한 액세스 토큰을 저장합니다.

3. **destination.labels["service-mesh.3scale.net/service-id"] == "replace-me"** 레이블을 워크로드에 추가하여 요청 시 인스턴스를 통해 서비스 **ID**를 어댑터에 전달합니다.

1.21.2. 3scale로 통합 설정 구성

3scale 통합 설정을 구성하려면 다음 절차를 따르십시오.



참고

3scale SaaS 고객의 경우, Red Hat OpenShift Service Mesh는 조Early Access 프로그램의 일부로 활성화됩니다.

절차

1. [your_API_name] → 통합으로 이동합니다.
2. 설정을 클릭합니다.
3. *배포*에서 Istio 옵션을 선택합니다.
   - *인증*에서 API Key (user_key) 옵션은 기본적으로 선택됩니다.
4. 제품 업데이트를 클릭하여 선택 사항을 저장합니다.
5. 설정을 클릭합니다.
6. 구성 업데이트를 클릭합니다.

1.21.3. 캐싱 동작

3scale System API의 응답은 기본적으로 어댑터 내에서 캐시됩니다. 항목이 cacheTTLSeconds 값보다 오래되면 캐시에서 제거됩니다. 또한 기본적으로 캐시된 항목의 자동 새로 고침은 cacheRefreshSeconds 값에 따라 만료되기 몇 초 전에 시도됩니다. 이 값을 cacheTTLSeconds 값보다 높게 설정하여 자동 새로 고침을 비활성화할 수 있습니다.

cacheEntriesMax를 양수가 아닌 값으로 설정하여 캐싱을 완전히 비활성화할 수 있습니다.

새로 고침 프로세스를 사용하면 호스트가 연결할 수 없는 캐시된 값은 만료가 지나면 제거되기 전에 다시 시도됩니다.

1.21.4. 요청 인증

이 릴리스에서는 다음 인증 방법을 지원합니다.

- 표준 **API** 키: 식별자와 시크릿 토큰으로 작동하는 임의의 단일 문자열 또는 해시입니다.
- 애플리케이션 식별자 및 키 쌍: 변경 불가능한 식별자 및 변경 가능한 시크릿 키 문자열입니다.
- **OpenID** 인증 방법: **JSON** 웹 토큰에서 구문 분석된 클라이언트 **ID** 문자열입니다.

1.21.4.1. 인증 패턴 적용

인증 동작을 구성하려면 다음 인증 방법 예제에 설명된 인스턴스 사용자 정의 리소스를 수정합니다. 다음에서 인증 자격 증명을 허용할 수 있습니다.

- 요청 헤더
- 요청 매개변수
- 요청 헤더 및 쿼리 매개변수 둘 다

참고

헤더에서 값을 지정하는 경우 소문자여야 합니다. 예를 들어 **User-Key**로 헤더를 보내려면 구성에서 **request.headers["user-key"]**로 참조되어야 합니다.

1.21.4.1.1. API 키 인증 방법

서비스 메시는 **subject** 사용자 정의 리소스 매개변수의 **user** 옵션에 지정된 대로 쿼리 매개변수 및 요청 헤더에서 **API** 키를 찾습니다. 사용자 정의 리소스 파일에 지정된 순서로 값을 확인합니다. 원하지 않는 옵션을 생략하여 **API** 키 검색을 쿼리 매개변수 또는 요청 헤더로 제한할 수 있습니다.

이 예에서 서비스 메시는 **user_key** 쿼리 매개변수에서 **API** 키를 찾습니다. **API** 키가 쿼리 매개변수에 없으면 서비스 메시가 **user-key** 헤더를 확인합니다.

API 키 인증 방법 예

```
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"
```

어댑터가 다른 쿼리 매개변수 또는 요청 헤더를 검사하도록 하려면 이름을 적절하게 변경합니다. 예를 들어 "key"라는 쿼리 매개변수에서 API 키를 확인하려면 request.query_params["user_key"]을 request.query_params["key"]로 변경합니다.

1.21.4.1.2. 애플리케이션 ID 및 애플리케이션 키 쌍 인증 방법

서비스 메시는 subject 사용자 정의 리소스 매개변수의 properties 옵션에 지정된 대로 쿼리 매개변수 및 요청 헤더에서 애플리케이션 ID와 애플리케이션 키를 찾습니다. 애플리케이션 키는 선택 사항입니다. 사용자 정의 리소스 파일에 지정된 순서로 값을 확인합니다. 원하지 않는 옵션을 제외하여 자격 증명 검색을 쿼리 매개변수 또는 요청 헤더로 제한할 수 있습니다.

이 예에서 서비스 메시는 쿼리 매개변수의 애플리케이션 ID 및 애플리케이션 키를 먼저 찾고 필요한 경우 요청 헤더로 이동합니다.

애플리케이션 ID 및 애플리케이션 키 쌍 인증 방법 예

```
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"
```

어댑터가 다른 쿼리 매개변수 또는 요청 헤더를 검사하도록 하려면 이름을 적절하게 변경합니다. 예를 들어, identification라는 쿼리 매개변수의 애플리케이션 ID를 확인하려면 request.query_params["app_id"]을 request.query_params["identification"]로 변경합니다.

1.21.4.1.3. OpenID 인증 방법

*OIDC(OpenID Connect) 인증 방법*을 사용하려면 subject 필드에서 properties 값을 사용하여 client_id 또는 필요한 경우 app_key로 설정할 수 있습니다.

이전에 설명된 방법을 사용하여 이 오브젝트를 조작할 수 있습니다. 아래 설정 예에서 클라이언트 식별자(애플리케이션 ID)는 *azp* 레이블 아래에 있는 JSON 웹 토큰(JWT)에서 구문 분석됩니다. 필요에 따라 수정할 수 있습니다.

OpenID 인증 방법 예

```
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"] | ""
```

이 통합이 올바르게 작동하려면 클라이언트가 ID 공급자(IdP)에서 생성되도록 OIDC를 3scale에서 수행해야 합니다. 해당 서비스와 동일한 네임스페이스에서 보호하려는 서비스에 대해 요청 권한 부여를 생성해야 합니다. JWT는 요청의 Authorization 헤더로 전달됩니다.

아래에 정의된 샘플 RequestAuthentication에서 issuer, jwksUri, selector를 적절하게 대체합니다.

OpenID 정책 예

```
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. 하이브리드 인증 방법

특정 인증 방법을 적용하지 않도록 선택하고, 두 방법에 대해 유효한 자격 증명을 수락할 수 있습니다. API 키와 애플리케이션 ID/애플리케이션 키 쌍이 모두 제공되면 서비스 메시는 API 키를 사용합니다.

이 예제에서 서비스 메시는 쿼리 매개변수에서 API 키를 확인한 다음 요청 헤더를 확인합니다. API 키가 없는 경우 쿼리 매개변수에서 애플리케이션 ID와 키를 확인한 다음 요청 헤더를 확인합니다.

하이브리드 인증 방법 예

```
apiVersion: "config.istio.io/v1alpha2"
kind: instance
metadata:
  name: threescale-authorization
spec:
```

```
template: authorization
params:
 subject:
  user: request.query_params["user_key"] | request.headers["user-key"] |
  properties:
   app_id: request.query_params["app_id"] | request.headers["app-id"] | ""
   app_key: request.query_params["app_key"] | request.headers["app-key"] | ""
   client_id: request.auth.claims["azp"] | ""
 action:
  path: request.url_path
  method: request.method | "get"
  service: destination.labels["service-mesh.3scale.net/service-id"] | ""
```

1.21.5. 3scale Adapter 지표

기본적으로 어댑터는 /metrics 끝점의 포트 8080에서 공개되는 다양한 Prometheus 지표를 보고합니다. 이러한 지표는 어댑터와 3scale 간의 상호 작용 수행 방식에 대한 인사이트를 제공합니다. 이 서비스는 Prometheus에서 자동으로 검색 및 스크랩하도록 레이블이 지정됩니다.

참고

Service Mesh 1.x의 이전 릴리스 이후 3scale Istio 어댑터 지표에 호환되지 않는 변경 사항이 있습니다.

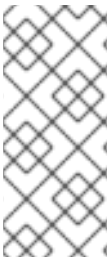
Prometheus에서 지표는 백엔드 캐시에 대해 한 번의 추가로 이름이 변경되었으므로, Service Mesh 2.0부터 다음 지표가 존재합니다.

표 1.30. Prometheus 지표

지표	유형	설명
threescale_latency	히스토그램	어댑터와 3scale 간의 대기 시간을 요청합니다.
threescale_http_total	카운터	3scale 백엔드에 대한 요청의 HTTP 상태 응답 코드입니다.
threescale_system_cache_hits	카운터	구성 캐시에서 가져온 3scale 시스템에 대한 총 요청 수입니다.
threescale_backend_cache_hits	카운터	백엔드 캐시에서 가져온 3scale 백엔드에 대한 총 요청 수입니다.

1.21.6. 3scale 백엔드 캐시

3scale 백엔드 캐시는 3scale Service Management API의 클라이언트에 대한 권한 부여 및 보고 캐시를 제공합니다. 이 캐시는 어댑터에 내장되어 관리자가 절충을 기꺼이 받아들인다는 가정하에 특정 상황에서 응답 대기 시간을 줄일 수 있습니다.



참고

3scale 백엔드 캐시는 기본적으로 비활성화되어 있습니다. 3scale 백엔드 캐시 기능은 낮은 대기 시간과 프로세서 및 메모리의 더 많은 리소스 소비를 위해 마지막 플러시를 수행한 후 속도 제한 및 잠재적 손실에서 부정확하게 사용됩니다.

1.21.6.1. 백엔드 캐시 활성화의 이점

다음은 백엔드 캐시 활성화의 몇 가지 이점입니다.

- 3scale Istio Adapter에서 관리하는 서비스에 액세스하는 동안 대기 시간이 길면 백엔드 캐시를 활성화합니다.
- 백엔드 캐시를 활성화하면 어댑터가 3scale API 관리자의 요청 승인을 계속 확인하지 못하므로 대기 시간이 줄어듭니다.
  - 이렇게 하면 3scale Istio Adapter가 3scale API 관리자에게 권한 부여를 요청하기 전에 저장 및 재사용할 수 있도록 3scale 권한 부여의 메모리 내 캐시가 생성됩니다. 권한 부여는 승인 또는 거부에 소요되는 시간이 훨씬 짧습니다.
- 백엔드 캐싱은 3scale Istio Adapter를 실행하는 서비스 메시와 다른 지리적 위치에서 3scale API 관리자를 호스팅할 때 유용합니다.
  - 일반적으로 3scale Hosted(SaaS) 플랫폼과 관련이 있지만 사용자가 다른 지리적 위치나 다른 가용성 영역에 있는 다른 클러스터에서 3scale API 관리자를 호스팅하거나 네트워크 오버헤드가 3scale API 관리자의 눈에 띄는 경우도 있습니다.

1.21.6.2. 대기 시간을 줄이기 위한 절충

다음은 대기 시간을 더 줄이기 위한 절충안입니다.

- 플러시가 발생할 때마다 **3scale** 어댑터의 권한 부여 상태 업데이트.
  - 즉, 두 개 이상의 어댑터 인스턴스가 플러시 간격 간에 부정확함을 발생시킵니다.
  - 제한을 초과하고 비정상적인 동작을 유발하는 요청이 너무 빈번하게 허용될 가능성이 커져, 각 요청을 처리하는 어댑터에 따라 어떤 요청은 통과하고, 어떤 요청은 통과하지 않습니다.
- 데이터를 플러시하고 권한 부여 정보를 업데이트할 수 없는 어댑터 캐시는 **API** 관리자에게 해당 정보를 보고하지 않고 종료되거나 중단될 수 있습니다.
- **API** 관리자에 연결할 때 네트워크 연결 문제로 인해 어댑터 캐시가 요청 허용/거부 여부를 결정할 수 없는 경우 페일오픈 또는 페일클로즈 정책이 적용됩니다.
- 일반적으로 어댑터를 부팅한 직후나 연결이 장기간 끊긴 뒤에 캐시 누락이 발생하면, **API** 관리자에게 쿼리하기 위해 대기 시간이 늘어납니다.
- 어댑터 캐시는 활성화된 캐시가 없을 때보다 컴퓨팅 권한 부여에 훨씬 더 많은 작업을 수행해야 프로세서 리소스에 부담이 됩니다.
- 메모리 요구 사항은 캐시로 관리되는 제한, 애플리케이션 및 서비스 조합에 비례하여 증가합니다.

1.21.6.3. 백엔드 캐시 구성 설정

다음 포인트는 백엔드 캐시 구성 설정을 설명합니다.

- **3scale** 구성 옵션에서 백엔드 캐시를 구성하는 설정을 찾습니다.
- 마지막 **3**가지 설정에서 백엔드 캐시 활성화를 제어합니다.
  - **PARAM_USE_CACHE_BACKEND -** 백엔드 캐시를 활성화하려면 **true**로 설정합니다.

- PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS - 캐시 데이터를 API 관리자에 플러시하려는 연속 시도 사이의 시간(초)을 설정합니다.

- PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED - 캐시 데이터가 충분하지 않고 3scale API 관리자에 연결할 수 없는 경우 서비스 요청을 허용/열기 또는 거부/종료 여부를 설정합니다.

1.21.7. 3scale Istio Adapter APIcast 에뮬레이션

3scale Istio Adapter는 다음과 같은 상태가 발생하는 경우 APIcast를 수행합니다.

- 요청이 정의된 매핑 규칙과 일치할 수 없는 경우 반환되는 HTTP 코드는 404 Not Found입니다. 이전에는 403 Forbidden이었습니다.

- 제한을 초과하여 요청이 거부되면 반환된 HTTP 코드는 429 Too Many Requests입니다. 이전에는 403 Forbidden이었습니다.

- CLI를 통해 기본 템플릿을 생성할 때 헤더에 대시 대신 밑줄을 사용합니다(예: user-key 대신 user_key).

1.21.8. 3scale Istio 어댑터 검증

3scale Istio 어댑터가 예상대로 작동하는지 확인해야 할 수 있습니다. 어댑터가 작동하지 않는 경우 다음 단계를 사용하여 문제를 해결합니다.

절차

1. Service Mesh Control Plane 네임스페이스에서 *3scale-adapter* Pod가 실행 중인지 확인합니다.

```
$ oc get pods -n <istio-system>
```

2. *3scale-adapter* Pod에서 버전과 같이 자체 부팅에 대한 정보를 출력했는지 확인합니다.

```
$ oc logs <istio-system>
```

3. **3scale** 어댑터 통합으로 보호되는 서비스에 대한 요청을 수행할 때 항상 올바른 인증 정보가 없는 요청을 시도하여 실패하는지 확인합니다. **3scale** 어댑터 로그를 확인하여 추가 정보를 수집합니다.

추가 리소스

- Pod 및 컨테이너 로그 검사

1.21.9. 3scale Istio 어댑터 문제 해결 체크리스트

3scale Istio 어댑터를 설치하는 관리자는 통합이 제대로 작동하지 않을 수 있는 여러 시나리오가 있습니다. 다음 목록을 사용하여 설치 문제를 해결합니다.

- **YAML** 들여쓰기가 잘못되었습니다.
- **YAML** 섹션이 누락되었습니다.
- **YAML** 변경 사항을 클러스터에 적용하는 것을 잊어버렸습니다.
- **service-mesh.3scale.net/credentials** 키를 사용하여 서비스 워크로드의 레이블을 지정하는 것을 잊어버렸습니다.
- 서비스 워크로드에 **service_id**가 포함되지 않은 처리기를 사용할 때 **service-mesh.3scale.net/service-id**로 레이블을 지정하여 계정별로 재사용할 수 있도록 하는 것을 잊었습니다.
- ***Rule*** 사용자 지정 리소스는 잘못된 처리기 또는 인스턴스 사용자 지정 리소스를 가리키거나 참조에 해당 네임스페이스 접미사가 없습니다.
- ***Rule*** 사용자 정의 리소스 **match** 섹션은 구성 중인 서비스와 일치하지 않거나 현재 실행 중이거나 존재하지 않는 대상 워크로드를 가리킵니다.
- 처리기의 **3scale** 관리 포털의 잘못된 액세스 토큰 또는 **URL**입니다.

- *인스턴스* 사용자 정의 리소스의 **params/subject/properties** 섹션은 쿼리 매개 변수, 헤더 및 권한 부여 클레임과 같은 잘못된 위치를 지정하거나 매개 변수 이름이 테스트에 사용되는 요청과 일치하지 않기 때문에 **app_id, app_key,** 또는 **client_id**에 대한 올바른 매개 변수를 나열하지 못합니다.

- 구성 생성기가 실제로 어댑터 컨테이너 이미지에 있고 **oc exec** 호출 해야 한다는 사실을 인식하지 못한 채 구성 생성기를 사용하지 못했습니다 .

1.22. 서비스 메시 문제 해결

이 섹션에서는 **Red Hat OpenShift Service Mesh**에서 일반적인 문제를 식별하고 해결하는 방법을 설명합니다. **OpenShift Container Platform**에 **Red Hat OpenShift Service Mesh**를 배포할 때 다음 섹션을 사용하여 문제를 해결하고 디버깅할 수 있습니다.

1.22.1. 서비스 메시 버전 이해

시스템에 배포한 **Red Hat OpenShift Service Mesh** 버전을 이해하려면 각 구성 요소 버전이 어떻게 관리되는지 이해해야 합니다.

- **Operator** 버전 **-** 최신 **Operator** 버전은 **2.4.4**입니다. **Operator** 버전 번호는 현재 설치된 **Operator**의 버전만 나타냅니다. **Red Hat OpenShift Service Mesh Operator**는 **Service Mesh Control Plane**의 여러 버전을 지원하므로 **Operator** 버전이 배포된 **ServiceMeshControlPlane** 리소스의 버전을 결정하지 않습니다.

중요

최신 **Operator** 버전으로 업그레이드하면 패치 업데이트가 자동으로 적용되지만 서비스 메시 컨트롤 플레인을 최신 마이너 버전으로 자동 업그레이드하지는 않습니다.

- **ServiceMeshControlPlane** 버전 **-** **ServiceMeshControlPlane** 버전은 사용 중인 **Red Hat OpenShift Service Mesh** 버전을 결정합니다. **ServiceMeshControlPlane** 리소스의 **spec.version** 필드 값은 **Red Hat OpenShift Service Mesh**를 설치하고 배포하는 데 사용되는 아키텍처 및 구성 설정을 제어합니다. **Service Mesh Control Plane**을 생성할 때 다음 두 가지 방법 중 하나로 버전을 설정할 수 있습니다.

  - 양식 보기에서 구성하려면 컨트롤 플레인 버전 메뉴에서 버전을 선택합니다.

- YAML 보기에서 구성하려면 YAML 파일에서 spec.version 값을 설정합니다.

OLM(Operator Lifecycle Manager)은 서비스 메시 컨트롤 플레인 업그레이드를 관리하지 않으므로 SMCP를 수동으로 업그레이드하지 않으면 Operator 및 ServiceMeshControlPlane (SMCP)의 버전 번호가 일치하지 않을 수 있습니다.

1.22.2. Operator 설치 문제 해결

이 섹션의 정보 외에도 다음 주제를 검토하십시오.

- Operator란 무엇인가?
- Operator 라이프사이클 관리 개념.
- OpenShift Operator 문제 해결 섹션.
- OpenShift 설치 문제 해결 섹션.

1.22.2.1. Operator 설치 검증

Red Hat OpenShift Service Mesh Operator를 설치할 때 OpenShift는 Operator 설치의 일부로 다음 오브젝트를 자동으로 생성합니다.

- 구성 맵
- 사용자 정의 리소스 정의
- Deployments
- pods

- 복제본 세트
- 역할
- 역할 바인딩
- **secrets**
- 서비스 계정
- **services**

OpenShift Container Platform 콘솔에서

OpenShift Container Platform 콘솔을 사용하여 **Operator Pod**를 사용할 수 있고 실행 중인지 확인할 수 있습니다.

1. 워크로드 → **Pod**로 이동합니다.
2. **openshift-operators** 네임스페이스를 선택합니다.
3. 다음 **pod**가 존재하고 **running** 상태인지 확인합니다.
   - **istio-operator**
   - **jaeger-operator**
   - **Kiali-operator**
4. **openshift-operators-redhat** 네임스페이스를 선택합니다.

5. elasticsearch-operator Pod가 존재하고 실행 중 상태가 있는지 확인합니다.

명령줄에서

1. 다음 명령을 사용하여 Operator pod를 사용할 수 있고 openshift-operators 네임스페이스에서 실행 중인지 확인합니다.

```
$ oc get pods -n openshift-operators
```

출력 예

```
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. 다음 명령을 사용하여 Elasticsearch Operator를 확인합니다.

```
$ oc get pods -n openshift-operators-redhat
```

출력 예

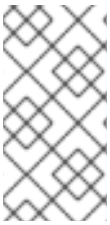
```
NAME                                      READY   STATUS    RESTARTS   AGE
elasticsearch-operator-d4f59b968-796vq    1/1     Running   0          15s
```

1.22.2.2. 서비스 메시 Operator 문제 해결

Operator 문제가 발생하는 경우

- Operator 서브스크립션 상태를 확인합니다.

- 지원되는 Red Hat 버전 대신 커뮤니티 버전의 Operator를 설치하지 않았는지 확인합니다.
- Red Hat OpenShift Service Mesh를 설치할 cluster-admin 역할이 있는지 확인합니다.
- 문제가 Operator 설치와 관련된 경우 Operator Pod 로그에서 오류가 있는지 확인합니다.



참고

OpenShift 콘솔을 통해서만 Operator를 설치할 수 있으며 명령줄에서 OperatorHub에 액세스할 수 없습니다.

1.22.2.2.1. Operator Pod 로그 보기

oc logs 명령을 사용하여 Operator 로그를 볼 수 있습니다. Red Hat은 지원 케이스 해결을 위해 로그를 요청할 수 있습니다.

절차

- Operator Pod 로그를 보려면 명령을 입력합니다.

```
$ oc logs -n openshift-operators <podName>
```

예를 들면 다음과 같습니다.

```
$ oc logs -n openshift-operators istio-operator-bb49787db-zgr87
```

1.22.3. 컨트롤 플레인 문제 해결

Service Mesh *Control Plane*은 *이전의 여러 컨트롤 플레인* 구성 요소(Citadel, Galley, Pilot)를 단일 바이너리로 통합하는 Istiod로 구성됩니다. ServiceMeshControlPlane 을 배포하면 아키텍처 항목에 설명된 대로 Red Hat OpenShift Service Mesh를 구성하는 다른 구성 요소도 생성됩니다.

1.22.3.1. Service Mesh Control Plane 설치 검증

Service Mesh Control Plane을 생성할 때 Service Mesh Operator는 ServiceMeshControlPlane 리소스 파일에 지정한 매개변수를 사용하여 다음을 수행합니다.

- Istio 구성 요소를 생성하고 다음 pod를 배포합니다.
  - istiod
  - istio-ingressgateway
  - istio-egressgateway
  - grafana
  - prometheus
- Kiali Operator를 호출하여 SMCP 또는 Kiali 사용자 정의 리소스의 구성을 기반으로 Kaili 배포를 생성합니다.

참고

Service Mesh Operator가 아닌 Kiali Operator에서 Kiali 구성 요소를 봅니다.

- Red Hat OpenShift distributed tracing Platform (Jaeger) Operator를 호출하여 SMCP 또는 Jaeger 사용자 정의 리소스의 구성을 기반으로 분산 추적 플랫폼(Jaeger) 구성 요소를 생성합니다.

참고

Service Mesh Operator가 아닌 Red Hat Elasticsearch Operator에서 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 및 Elasticsearch 구성 요소를 통해 Jaeger 구성 요소를 봅니다.

OpenShift Container Platform 콘솔에서

OpenShift Container Platform 웹 콘솔에서 Service Mesh Control Plane 설치를 확인할 수 있습니다.

1. **Operators → 설치된 Operator로 이동합니다.**

2. **< istio-system> 네임스페이스 를 선택합니다.**

3. **Red Hat OpenShift Service Mesh Operator를 선택합니다.**

   a. **Istio Service Mesh Control Plane 탭을 클릭합니다.**

   b. **컨트롤 플레인의 이름을 클릭합니다(예: 기본).**

   c. **배포에서 생성한 리소스를 보려면 리소스 탭을 클릭합니다. 예를 들어 필터를 사용하여 보기를 좁힐 수 있습니다. 예를 들어 모든 Pod 의 상태가 running 인지 확인할 수 있습니다.**

   d. **SMCP 상태가 문제를 나타내는 경우 YAML 파일에서 status: output을 확인하여 자세한 내용을 확인합니다.**

   e. **Operator → 설치된 Operator 로 다시 이동합니다.**

4. **OpenShift Elasticsearch Operator를 선택합니다.**

   a. **Elasticsearch 탭을 클릭합니다.**

   b. **배포 이름을 클릭합니다(예: elasticsearch).**

   c. **배포에서 생성된 리소스를 보려면 리소스 탭을 클릭합니다. .**

   d. **Status (상태) 열에 문제가 있는 경우 자세한 내용은 YAML 탭에서 status: 출력을 확인합니다.**

e. **Operator → 설치된 Operator** 로 다시 이동합니다.

5. **Red Hat OpenShift distributed tracing Platform (Jaeger) Operator**를 선택합니다.

a. **Jaeger** 탭을 클릭합니다.

b. 배포 이름을 클릭합니다(예: **jaeger**).

c. 배포에서 생성한 리소스를 보려면 리소스 탭을 클릭합니다.

d. **Status** 열이 문제를 나타내는 경우 자세한 내용은 **YAML** 탭에서 **status:** 출력을 확인합니다.

e. **Operators → 설치된 Operator**로 이동합니다.

6. **Kiali Operator**를 선택합니다.

a. **Istio Service Mesh Control Plane** 탭을 클릭합니다.

b. 배포 이름을 클릭합니다(예: **kiali**).

c. 배포에서 생성한 리소스를 보려면 리소스 탭을 클릭합니다.

d. **Status** (상태) 열에 문제가 있는 경우 자세한 내용은 **YAML** 탭에서 **status:** 출력을 확인합니다.

명령줄에서

1. 다음 명령을 실행하여 **Service Mesh Control Plane Pod**가 사용 가능하고 실행 중인지 확인합니다. 여기서 **istio-system** 은 **SMCP**를 설치한 네임스페이스입니다.

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

출력 예

```
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. 다음 명령을 사용하여 **Service Mesh Control Plane** 배포의 상태를 확인합니다. **istio-system** 을 **SMCP**를 배포한 네임스페이스로 교체합니다.

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

STATUS 열이 **ComponentsReady** 이면 설치가 성공적으로 완료되었습니다.

출력 예

```
NAME    READY   STATUS           PROFILES     VERSION   AGE
basic   10/10   ComponentsReady  ["default"]  2.1.3     4m2s
```

Service Mesh Control Plane을 수정하고 재배포한 경우 상태가 **UpdateSuccessful** 여야 합니다.

출력 예

```
NAME          READY    STATUS            TEMPLATE   VERSION   AGE
basic-install  10/10    UpdateSuccessful  default    v1.1      3d16h
```

3. SMCP 상태가 ComponentsReady 이외의 항목을 표시하는 경우 SCMP 리소스의 status: output을 SCMP 리소스에서 표시합니다.

```
$ oc describe smcp <smcp-name> -n <controlplane-namespace>
```

출력 예

```
$ oc describe smcp basic -n istio-system
```

4. 다음 명령을 사용하여 Jaeger 배포의 상태를 확인합니다. 여기서 istio-system 은 SMCP를 배포한 네임스페이스입니다.

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

출력 예

```
NAME   STATUS   VERSION   STRATEGY   STORAGE   AGE
jaeger   Running   1.30.0   allinone   memory   15m
```

5. 다음 명령을 사용하여 Kiali 배포의 상태를 확인합니다. 여기서 istio-system 은 SMCP를 배포한 네임스페이스입니다.

```
$ oc get kiali -n <istio-system>
```

출력 예

```
NAME   AGE
kiali   15m
```

-

1.22.3.1.1. Kiali 콘솔에 액세스

Kiali 콘솔에서 애플리케이션의 토폴로지, 상태 및 지표를 볼 수 있습니다. 서비스에 문제가 발생하면 Kiali 콘솔을 사용하여 서비스를 통해 데이터 흐름을 볼 수 있습니다. 추상 애플리케이션, 서비스 및 워크로드를 포함하여 다양한 수준에서 메시 구성 요소에 대한 인사이트를 볼 수 있습니다. Kiali는 네임스페이스에 대한 대화형 그래프 보기도 실시간으로 제공합니다.

Kiali 콘솔에 액세스하려면 Red Hat OpenShift Service Mesh가 설치 및 구성되어 있어야 합니다.

설치 프로세스는 Kiali 콘솔에 액세스하기 위한 경로를 생성합니다.

Kiali 콘솔의 URL을 알고 있는 경우 직접 액세스할 수 있습니다. URL을 모르는 경우 다음 지침을 사용하십시오.

관리자의 절차

1. 관리자 역할을 사용하여 OpenShift Container Platform 웹 콘솔에 로그인합니다.
2. 홈 → 프로젝트를 클릭합니다.
3. 필요한 경우 프로젝트 페이지에서 필터를 사용하여 프로젝트 이름을 찾습니다.
4. 프로젝트 이름을 클릭합니다(예: info).
5. 프로젝트 세부 정보 페이지의 시작 관리자 섹션에서 Kiali 링크를 클릭합니다.
6. OpenShift Container Platform 콘솔에 액세스하는 데 사용하는 것과 동일한 사용자 이름 및 암호를 사용하여 Kiali 콘솔에 로그인합니다.

   Kiali 콘솔에 처음 로그인하면 볼 수 있는 권한이 있는 서비스 메시에 모든 네임스페이스가 표시되는 개요 페이지가 표시됩니다.

콘솔 설치의 유효성을 검사하고 네임스페이스가 메시에 아직 추가되지 않은 경우 **istio-system** 이외의 데이터가 표시되지 않을 수 있습니다.

개발자용 절차

1. 개발자 역할을 사용하여 **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. 프로젝트를 클릭합니다.
3. 필요한 경우 프로젝트 세부 정보 페이지에서 필터를 사용하여 프로젝트 이름을 찾습니다.
4. 프로젝트 이름을 클릭합니다(예: **info**).
5. 프로젝트 페이지의 시작 관리자 섹션에서 **Kiali** 링크를 클릭합니다.
6. **OpenShift**로 로그인 을 클릭합니다.

1.22.3.1.2. Jaeger 콘솔에 액세스

Jaeger 콘솔에 액세스하려면 **Red Hat OpenShift Service Mesh**가 설치되어 있어야 합니다. **Red Hat OpenShift distributed tracing Platform (Jaeger)**이 설치되어 구성되어 있어야 합니다.

설치 프로세스는 **Jaeger** 콘솔에 액세스하기 위한 경로를 생성합니다.

Jaeger 콘솔의 **URL**을 알고 있는 경우 직접 액세스할 수 있습니다. **URL**을 모르는 경우 다음 지침을 사용하십시오.

OpenShift 콘솔의 절차

1. **OpenShift Container Platform** 웹 콘솔에 **cluster-admin** 권한이 있는 사용자로 로그인합니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 계정이 있어야 합니다.

2. 네트워킹 → 경로로 이동합니다.

3. 경로 페이지의 네임스페이스 메뉴에서 **Service Mesh Control Plane** 프로젝트(예: **istio-system**)를 선택합니다.

   위치 열은 각 경로에 대한 연결된 주소를 표시합니다.

4. 필요한 경우 필터를 사용하여 **jaeger** 경로를 찾습니다. 경로 위치를 클릭하여 콘솔을 시작합니다.

5. **OpenShift**로 로그인 을 클릭합니다.

Kiali 콘솔의 절차

1. **Kiali** 콘솔을 시작합니다.

2. 왼쪽 탐색 창에서 **Distributed Tracing** 을 클릭합니다.

3. **OpenShift**로 로그인 을 클릭합니다.

CLI의 프로세스

1. **cluster-admin** 역할의 사용자로 **OpenShift Container Platform CLI**에 로그인합니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 계정이 있어야 합니다.

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

2. 명령줄을 사용하여 경로의 세부 정보를 쿼리하려면 다음 명령을 입력합니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 네임스페이스입니다.

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

3. 브라우저를 시작하고 **https://<JAEGER_URL >**로 이동합니다. 여기서 **< JAEGER_URL >** 은 이전 단계에서 검색한 경로입니다.

4. **OpenShift Container Platform** 콘솔에 액세스하는 데 사용하는 것과 동일한 사용자 이름 및 암호를 사용하여 로그인합니다.

5. 서비스 메시에 서비스를 추가하고 생성된 추적이 있는 경우 필터를 사용하고 추적 찾기 버튼을 사용하여 추적 데이터를 검색할 수 있습니다.

   콘솔 설치의 유효성을 확인하는 경우 표시할 추적 데이터가 없습니다.

1.22.3.2. Service Mesh Control Plane 문제 해결

Service Mesh Control Plane을 배포하는 동안 문제가 발생하는 경우

- **ServiceMeshControlPlane** 리소스가 서비스 및 **Operator**와 별도의 프로젝트에 설치되어 있는지 확인합니다. 이 문서에서는 **istio-system** 프로젝트를 예로 사용하지만 **Operator** 및 서비스가 포함된 프로젝트와 별도로 모든 프로젝트에 컨트롤 플레인을 배포할 수 있습니다.

- **ServiceMeshControlPlane** 및 **Jaeger** 사용자 정의 리소스가 동일한 프로젝트에 배포되었는지 확인합니다. 예를 들어 둘 다에 대해 **istio-system** 프로젝트를 사용합니다.

1.22.4. 데이터 플레인 문제 해결

데이터 플레인 은 서비스 메시의 서비스 간 인바운드 및 아웃바운드 네트워크 통신을 가로채고 제어하는 지능형 프록시 집합입니다.

Red Hat OpenShift Service Mesh는 애플리케이션 **pod** 내의 프록시 사이드카를 사용하여 애플리케이션에 서비스 메시 기능을 제공합니다.

1.22.4.1. 사이드카 삽입 문제 해결

Red Hat OpenShift Service Mesh는 **Pod**에 프록시 사이드카를 자동으로 삽입하지 않습니다. 사이드카 삽입을 선택해야 합니다.

1.22.4.1.1. Istio 사이드카 삽입 문제 해결

애플리케이션의 배포에서 자동 삽입이 활성화되어 있는지 확인합니다. **Envoy** 프록시에 대한 자동 삽

입이 활성화된 경우 spec.template.metadata.annotations 아래의 Deployment 리소스에 sidecar.istio.io/inject:"true" 주석이 있어야 합니다.

1.22.4.1.2. Jaeger 에이전트 사이드카 삽입 문제 해결

애플리케이션의 배포에서 자동 삽입이 활성화되어 있는지 확인합니다. Jaeger 에이전트에 대한 자동 삽입이 활성화된 경우 Deployment 리소스에 sidecar.jaegertracing.io/inject:"true" 주석이 있어야 합니다.

사이드카 삽입에 대한 자세한 내용은 자동 삽입 활성화를참조하십시오.

1.23. ENVOY 프록시 문제 해결

Envoy 프록시는 서비스 메시의 모든 서비스에 대한 인바운드 및 아웃바운드 트래픽을 가로채기합니다. 또한 Envoy는 서비스 메시에서 Telemetry를 수집하고 보고합니다. Envoy는 동일한 pod에서 관련 서비스에 사이드카로 배포됩니다.

1.23.1. Envoy 액세스 로그 활성화

Envoy 액세스 로그는 트래픽 오류 및 흐름을 진단하는 데 유용하며 엔드 투 엔드 트래픽 흐름 분석을 지원합니다.

모든 istio-proxy 컨테이너에 대한 액세스 로깅을 활성화하려면 ServiceMeshControlPlane (SMCP) 오브젝트를 편집하여 로깅 출력의 파일 이름을 추가합니다.

프로세스

1. cluster-admin 역할의 사용자로 OpenShift Container Platform CLI에 로그인합니다. 다음 명령을 입력합니다. 메시지가 표시되면 사용자 이름과 암호를 입력합니다.

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

2. Service Mesh Control Plane을 설치한 프로젝트(예: istio-system)로 변경합니다.

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

3. ServiceMeshControlPlane 파일을 편집합니다.

```
$ oc edit smcp <smcp_name>
```

4. 다음 예제에 표시된 대로 **name** 을 사용하여 프록시 로그의 파일 이름을 지정합니다. **name** 의 값을 지정하지 않으면 로그 항목이 기록되지 않습니다.

```
spec:
  proxy:
    accessLogging:
      file:
        name: /dev/stdout     #file name
```

Pod 문제 해결에 대한 자세한 내용은 **Pod** 문제 해결을 참조하십시오. https://access.redhat.com/documentation/en-us/openshift_container_platform/4.11/html-single/support/#investigating-pod-issues-1

1.23.2. 지원 요청

이 문서에 설명된 절차 또는 일반적으로 **OpenShift Container Platform**에 문제가 발생하는 경우 **Red Hat** 고객 포털에 액세스하십시오. 고객 포털에서 다음을 수행할 수 있습니다.

- **Red Hat** 제품과 관련된 기사 및 솔루션에 대한 **Red Hat** 지식베이스를 검색하거나 살펴볼 수 있습니다.
- **Red Hat** 지원에 대한 지원 케이스 제출할 수 있습니다.
- 다른 제품 설명서에 액세스 가능합니다.

클러스터 문제를 식별하기 위해 **OpenShift Cluster Manager Hybrid Cloud Console** 에서 **Insights** 를 사용할 수 있습니다. **Insights**는 문제에 대한 세부 정보 및 문제 해결 방법에 대한 정보를 제공합니다.

이 문서를 개선하기 위한 제안이 있거나 오류를 발견한 경우 가장 관련 있는 문서 구성 요소에 대한 **Jira** 문제를 제출하십시오. 섹션 이름 및 **OpenShift Container Platform** 버전과 같은 구체적인 정보를 제공합니다.

1.23.2.1. Red Hat 지식베이스 정보

Red Hat 지식베이스는 **Red Hat**의 제품과 기술을 최대한 활용할 수 있도록 풍부한 콘텐츠를 제공합니다. **Red Hat** 지식베이스는 **Red Hat** 제품 설치, 설정 및 사용에 대한 기사, 제품 문서 및 동영상으로 구성

되어 있습니다. 또한 알려진 문제에 대한 솔루션을 검색할 수 있으며, 간결한 근본 원인 설명 및 해결 단계를 제공합니다.

1.23.2.2. Red Hat 지식베이스 검색

OpenShift Container Platform 문제가 발생한 경우 초기 검색을 수행하여 솔루션이 이미 Red Hat Knowledgebase 내에 존재하는지 확인할 수 있습니다.

사전 요구 사항

- Red Hat 고객 포털 계정이 있어야 합니다.

프로세스

1. Red Hat 고객 포털에 로그인합니다.
2. Search를 클릭합니다
3. 검색 필드에서 다음을 포함하여 문제와 관련된 키워드 및 문자열을 입력합니다.
   - OpenShift Container Platform 구성 요소 (etcd 등)
   - 관련 절차 (예: installation 등)
   - 명시적 실패와 관련된 경고, 오류 메시지 및 기타 출력
4. Enter 키를 클릭합니다.
5. 선택 사항: OpenShift Container Platform 제품 필터를 선택합니다.
6. 선택 사항: 문서 콘텐츠 유형 필터를 선택합니다.

1.23.2.3. 서비스 메시 데이터 수집 정보

oc adm must-gather CLI 명령을 사용하면 Red Hat OpenShift Service Mesh와 연관된 기능 및 오브젝트를 포함하여 클러스터에 대한 정보를 수집할 수 있습니다.

사전 요구 사항

- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- OpenShift Container Platform CLI(oc)가 설치되어 있어야 합니다.

프로세스

1. must-gather을 사용하여 Red Hat OpenShift Service Mesh 데이터를 수집하려면 Red Hat OpenShift Service Mesh 이미지를 지정해야 합니다.

```
$ oc adm must-gather --image=registry.redhat.io/openshift-service-mesh/istio-must-gather-rhel8:2.4
```

2. must-gather 로 특정 Service Mesh Control Plane 네임스페이스에 대한 Red Hat OpenShift Service Mesh 데이터를 수집하려면 Red Hat OpenShift Service Mesh 이미지 및 네임스페이스를 지정해야 합니다. 이 예에서는 수집 후 < namespace >를 istio-system 과 같은 서비스 메시 컨트롤 플레인 네임스페이스로 교체합니다.

```
$ oc adm must-gather --image=registry.redhat.io/openshift-service-mesh/istio-must-gather-rhel8:2.4 gather <namespace>
```

이렇게 하면 다음 항목이 포함된 로컬 디렉터리가 생성됩니다.

- Istio Operator 네임스페이스 및 해당 하위 오브젝트
- 모든 컨트롤 플레인 네임스페이스 및 해당 하위 오브젝트
- 모든 네임스페이스 및 해당 하위 오브젝트는 모든 서비스 메시에 속하는 오브젝트
- 모든 Istio CRD(사용자 정의 리소스 정의)

- 지정된 네임스페이스의 VirtualServices와 같은 모든 Istio CRD 오브젝트
- 모든 Istio Webhook

즉각 지원을 받을 수 있도록 OpenShift Container Platform 및 Red Hat OpenShift Service Mesh 둘 다에 대한 진단 정보를 제공하십시오.

1.23.2.4. 지원 케이스 제출

사전 요구 사항

- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- OpenShift CLI(oc)가 설치되어 있습니다.
- Red Hat 고객 포털 계정이 있어야 합니다.
- Red Hat 표준 또는 프리미엄 서브스크립션이 있습니다.

프로세스

1. Red Hat 고객 포털의 고객 지원 페이지에 로그인합니다.
2. 지원 받기를 클릭합니다.
3. 고객 지원 페이지의 케이스 탭에서 다음을 수행합니다.
   a. 선택 사항: 필요한 경우 미리 채워진 계정과 소유자를 변경합니다.
   b. 문제에 대한 적절한 카테고리(예: Bug 또는 Defect)를 선택하고 Continue 를 클릭합니다.

4. 다음 정보를 입력합니다.

   a. 요약 필드에 간결하지만 설명적인 문제 요약을 입력하고 경험되는 증상에 대한 자세한 내용과 기대치를 입력합니다.

   b. 제품 드롭다운 메뉴에서 **OpenShift Container Platform** 을 선택합니다.

   c. 버전 드롭다운에서 **4.11** 을 선택합니다.

5. 보고되는 문제와 관련이 있을 수 있는 권장 **Red Hat** 지식베이스 솔루션 목록을 확인합니다. 제안된 문서로 문제가 해결되지 않으면 **Continue**을 클릭합니다.

6. 보고되는 문제와 관련있는 제안된 **Red Hat** 지식베이스 솔루션 목록을 확인하십시오. 케이스 작성 과정에서 더 많은 정보를 제공하면 목록이 구체화됩니다. 제안된 문서로 문제가 해결되지 않으면 **Continue**을 클릭합니다.

7. 제시된 계정 정보가 정확한지 확인하고 필요한 경우 적절하게 수정합니다.

8. 자동 입력된 **OpenShift Container Platform** 클러스터 **ID**가 올바른지 확인합니다. 그렇지 않은 경우 클러스터 **ID**를 수동으로 가져옵니다.

   - **OpenShift Container Platform** 웹 콘솔을 사용하여 클러스터 **ID**를 수동으로 가져오려면 다음을 수행합니다.

     a. 홈 → 개요 로 이동합니다.

     b. **Details** 섹션의 **Cluster ID** 필드에서 값을 찾습니다.

   - 또는 **OpenShift Container Platform** 웹 콘솔을 통해 새 지원 케이스를 열고 클러스터 **ID**를 자동으로 입력할 수 있습니다.

     a. 툴바에서 **(?) Help → Open Support Case**로 이동합니다.

b.
Cluster ID 값이 자동으로 입력됩니다.

- **OpenShift CLI (oc)**를 사용하여 클러스터 **ID**를 얻으려면 다음 명령을 실행합니다.

```
$ oc get clusterversion -o jsonpath='{.items[].spec.clusterID}{"\n"}'
```

9. 프롬프트가 표시되면 다음 질문을 입력한 후 **Continue**를 클릭합니다.

- 이 문제가 어디에서 발생했습니까? 어떤 시스템 환경을 사용하고 있습니까?
- 이 동작이 언제 발생했습니까? 발생 빈도는 어떻게 됩니까? 반복적으로 발생합니까? 특정 시간에만 발생합니까?
- 이 문제의 발생 기간 및 비즈니스에 미치는 영향에 대한 정보를 제공해주십시오.

10. 관련 진단 데이터 파일을 업로드하고 **Continue**를 클릭합니다. **oc adm must-gather** 명령을 사용하여 수집된 데이터와 해당 명령으로 수집되지 않은 특정 문제와 관련된 데이터를 제공하는 것이 좋습니다

11. 관련 케이스 관리 세부 정보를 입력하고 **Continue**를 클릭합니다.

12. 케이스 세부 정보를 미리보고 **Submit**을 클릭합니다.

1.24. 서비스 메시 컨트롤 플레인 구성 참조

기본 **ServiceMeshControlPlane (SMCP)** 리소스를 수정하거나 완전히 사용자 정의 **SMCP** 리소스를 생성하여 **Red Hat OpenShift Service Mesh**를 사용자 지정할 수 있습니다. 이 참조 섹션에는 **SMCP** 리소스에 사용할 수 있는 구성 옵션이 설명되어 있습니다.

1.24.1. 서비스 메시 컨트롤 플레인 매개변수

다음 표에는 **ServiceMeshControlPlane** 리소스의 최상위 매개변수가 나열되어 있습니다.

표 1.31. ServiceMeshControlPlane 리소스 매개변수

이름	설명	유형
apiVersion	APIVersion은 버전이 지정된 이 오브젝트 표현의 스키마를 정의합니다. 서버는 인식된 스키마를 최신 내부 값으로 변환해야 하며, 인식되지 않는 값을 거부할 수 있습니다. **ServiceMeshControlPlane** 버전 2.0의 값은 **maistra.io/v2**입니다.	**ServiceMeshControlPlane** 버전 2.0의 값은 **maistra.io/v2**입니다.
kind	kind는 이 오브젝트가 나타내는 REST 리소스를 나타내는 문자열 값입니다.	**ServiceMeshControlPlane**은 ServiceMeshControlPlane의 유일한 유효한 값입니다.
metadata	이 **ServiceMeshControlPlane** 인스턴스에 대한 메타데이터입니다. Service Mesh Control Plane 설치의 이름을 지정하여 작업을 추적할 수 있습니다(예: **basic**).	string
spec	이 **ServiceMeshControlPlane**의 원하는 상태 사양입니다. 여기에는 Service Mesh Control Plane을 구성하는 모든 구성 요소에 대한 구성 옵션이 포함됩니다.	자세한 내용은 표 2를 참조하십시오.
status	이 **ServiceMeshControlPlane**의 현재 상태 및 Service Mesh 컨트롤 플레인을 구성하는 구성 요소입니다.	자세한 내용은 표 3을 참조하십시오.

다음 표에는 ServiceMeshControlPlane 리소스의 사양이 나열되어 있습니다. 이러한 매개변수를 변경하면 Red Hat OpenShift Service Mesh 구성 요소가 구성됩니다.

표 1.32. ServiceMeshControlPlane 리소스 사양

이름	설명	구성 가능한 매개변수
addons	**addons** 매개변수는 시각화 또는 메트릭 스토리지와 같은 핵심 Service Mesh Control Plane 구성 요소 이외의 추가 기능을 구성합니다.	**3scale**, **grafana**, **jaeger**, **kiali**, **prometheus**.

이름	설명	구성 가능한 매개변수
cluster	**cluster** 매개변수는 클러스터의 일반 구성(예: 클러스터 이름, 네트워크 이름, 다중 클러스터, 메시 확장 등)을 설정합니다.	**meshExpansion**, **multiCluster**, **name**, **network**
gateways	**gateways** 매개변수를 사용하여 메시에 대한 수신 및 송신 게이트웨이를 구성합니다.	**enabled**, **additionalEgress**, **additionalIngress**, **egress**, **ingress**, **openshiftRoute**
general	**general** 매개변수는 다른 곳에 맞지 않는 일반 Service Mesh Control Plane 구성을 나타냅니다.	**logging**, **validationMessages**
policy	**policy** 매개변수를 사용하여 Service Mesh Control Plane에 대한 정책 검사를 구성합니다. **spec.policy.enabled**를 **true**로 설정하여 정책 검사를 활성화할 수 있습니다.	**mixer remote** 또는 **type**. **type**은 **Istiod**, **Mixer** 또는 **None**으로 설정할 수 있습니다.
profiles	**profiles** 매개변수를 사용하여 기본값에 적용할 **ServiceMeshControlPlane** 프로필을 선택합니다.	**default**
proxy	**proxy** 매개변수를 사용하여 사이드카의 기본 동작을 설정합니다.	**accessLogging**, **adminPort**, **concurrency**, **envoyMetricsService**
runtime	**runtime** 매개변수를 사용하여 Service Mesh Control Plane 구성 요소를 구성합니다.	**components**, **defaults**
보안	**security** 매개변수를 사용하면 Service Mesh Control Plane에 대한 보안 측면을 구성할 수 있습니다.	**certificateAuthority**, **controlPlane**, **identity**, **dataPlane**, **trust**
techPreview	**techPreview** 매개변수를 사용하면 기술 프리뷰에 있는 기능에 조기 액세스할 수 있습니다.	해당 없음
telemetry	**spec.mixer.telemetry.enabled**를 **true**로 설정되면 Telemetry가 활성화됩니다.	**mixer**, **remote**, **type**. **type**은 **Istiod**, **Mixer** 또는 **None**으로 설정할 수 있습니다.
tracing	**tracing** 매개변수를 사용하여 메시의 분산 추적을 활성화합니다.	**sampling**, **type**. **type**은 **Jaeger** 또는 **None**으로 설정할 수 있습니다.

이름	설명	구성 가능한 매개변수
version	**version** 매개변수를 사용하여 설치할 Service Mesh Control Plane의 Maistra 버전을 지정합니다. 비어 있는 버전으로 **ServiceMeshControlPlane**을 생성할 때 승인 Webhook는 버전을 현재 버전으로 설정합니다. 빈 버전이 있는 새로운 **ServiceMeshControlPlanes**가 **v2.0**으로 설정됩니다. 빈 버전이 있는 기존 **ServiceMeshControlPlanes**는 설정을 유지합니다.	string

ControlPlaneStatus는 서비스 메시의 현재 상태를 나타냅니다.

표 1.33. ServiceMeshControlPlane 리소스 ControlPlaneStatus

이름	설명	유형
annotations	**annotations** 매개변수는 **ServiceMeshControlPlane**에서 배포한 구성 요소 수와 같이 일반적으로 중복되는 상태 정보를 저장합니다. 이러한 상태는 아직 JSONPath 표현식에서 오브젝트를 셀 수 없는 **oc** 명령줄 도구에서 사용됩니다.	구성 불가능
conditions	오브젝트의 현재 상태에 대해 사용 가능한 최신 관찰을 나타냅니다. **Reconciled**는 Operator가 **ServiceMeshControlPlane** 리소스의 구성을 사용하여 배포된 구성 요소의 실제 상태를 조정했는지 여부를 나타냅니다. **Installed** 는 Service Mesh Control Plane이 설치되었는지를 나타냅니다. **Ready** 는 모든 Service Mesh Control Plane 구성 요소가 준비되었는지를 나타냅니다.	string
components	배포된 각 Service Mesh Control Plane 구성 요소의 상태를 표시합니다.	string
appliedSpec	모든 프로필이 적용된 후 구성 옵션의 결과 사양입니다.	**ControlPlaneSpec**

이름	설명	유형
appliedValues	차트를 생성하는 데 사용되는 결과 values.yaml입니다.	**ControlPlaneSpec**
chartVersion	이 리소스를 위해 마지막으로 처리된 차트의 버전입니다.	string
observedGeneration	가장 최근 조정 중에 컨트롤러가 관찰한 생성입니다. 상태의 정보는 이 특정 오브젝트 생성과 관련이 있습니다. **status.conditions**는 **status.observedGeneration** 필드가 **metadata.generation**과 일치하지 않는 경우 최신 상태가 아닙니다.	정수
operatorVersion	이 리소스를 마지막으로 처리하는 Operator의 버전입니다.	string
readiness	구성 요소 및 소유 리소스의 준비 상태입니다.	string

이 예제 ServiceMeshControlPlane 정의에는 지원되는 모든 매개변수가 포함되어 있습니다.

ServiceMeshControlPlane 리소스 예

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.4
  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. 사양 매개변수

1.24.2.1. 일반 매개변수

다음 예제는 **ServiceMeshControlPlane** 오브젝트의 **spec. authorization** 매개변수와 적절한 값과 함께 사용 가능한 매개변수에 대한 설명을 보여줍니다.

일반 매개변수 예

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  general:
    logging:
      componentLevels: {}
        # misc: error
      logAsJSON: false
    validationMessages: true
```

표 1.34. Istio 일반 매개변수

매개변수	설명	값	기본값
logging:	을 사용하여 Service Mesh Control Plane 구성 요소에 대한 로깅을 구성합니다.		해당 없음
logging: componentLevels:	을 사용하여 구성 요소 로깅 수준을 지정합니다.	가능한 값: **trace**, **debug**, **info**, **warning**, **error**, **fatal**, **panic**.	해당 없음
logging: logAsJSON:	을 사용하여 JSON 로깅을 활성화하거나 비활성화합니다.	**true**/**false**	해당 없음

매개변수	설명	값	기본값
validationMessages:	을 사용하여 istio.io 리소스의 상태 필드에 대한 유효성 검사 메시지를 활성화하거나 비활성화합니다. 이는 리소스의 구성 오류를 탐지하는 데 유용할 수 있습니다.	**true/false**	해당 없음

1.24.2.2. profiles 매개변수

ServiceMeshControlPlane 오브젝트 프로필을 사용하여 재사용 가능한 구성을 생성할 수 있습니다. 프로필 설정을 구성하지 않으면 **Red Hat OpenShift Service Mesh**가 기본 프로필을 사용합니다.

다음 예제는 **ServiceMeshControlPlane** 오브젝트의 **spec.profiles** 매개변수를 보여줍니다.

프로필 매개변수 예

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

프로필 생성에 대한 자세한 내용은 컨트롤 플레인 프로필 생성을 참조하십시오.

보안 구성의 자세한 내용은 mTLS(mutual Transport Layer Security) 을 참조하십시오.

1.24.2.3. techPreview 매개변수

spec.techPreview 매개변수를 사용하면 기술 프리뷰에 있는 기능에 조기 액세스할 수 있습니다.

중요

기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

1.24.2.4. 매개변수 추적

다음 예제에서는 **ServiceMeshControlPlane** 오브젝트의 **spec.tracing** 매개변수와 적절한 값과 함께 사용 가능한 매개변수에 대한 설명을 보여줍니다.

추적 매개변수 예

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.4
  tracing:
    sampling: 100
    type: Jaeger
```

표 1.35. Istio 추적 매개변수

매개변수	설명	값	기본값
tracing: sampling:	샘플링 비율은 Envoy 프록시가 추적을 생성하는 빈도를 결정합니다. 샘플링 속도를 사용하여 추적 시스템에 보고되는 요청의 백분율을 제어합니다.	0에서 10000 사이의 정수 값은 0.01%(0 ~ 100%)를 나타냅니다. 예를 들어 값을 **10** 으로 설정하면 요청의 0.1%를 **100** 으로 설정하면 값이 **500** 으로 설정된 요청의 1%를 샘플링하고, **10000** 으로 설정하면 요청의 100%를 샘플링합니다.	**10000** (100%의 추적)

매개변수	설명	값	기본값
tracing: type:	현재 지원되는 유일한 추적 유형은 **Jaeger**입니다. Jaeger는 기본적으로 활성화되어 있습니다. 추적을 비활성화하려면 **type** 매개 변수를 **None**으로 설정합니다.	**없음,Jaeger**	**Jaeger**

1.24.2.5. 버전 매개변수

Red Hat OpenShift Service Mesh Operator는 ServiceMeshControlPlane 의 다양한 버전의 설치를 지원합니다. version 매개변수를 사용하여 설치할 Service Mesh Control Plane의 버전을 지정합니다. SMCP를 생성할 때 version 매개변수를 지정하지 않으면 Operator에서 값을 최신 버전 (2.4)로 설정합니다. 기존 ServiceMeshControlPlane 오브젝트는 Operator 업그레이드 중에 버전 설정을 유지합니다.

1.24.2.6. 3scale 구성

다음 표는 **ServiceMeshControlPlane** 리소스의 **3scale Istio** 어댑터에 대한 매개변수를 설명합니다

3scale 매개변수 예

```
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
```

표 1.36. 3scale 매개변수

매개변수	설명	값	기본값
enabled	3scale 어댑터 사용 여부	**true/false**	**false**
PARAM_THREESCALE_LISTEN_ADDR	gRPC 서버의 수신 주소를 설정	유효한 포트 번호	**3333**
PARAM_THREESCALE_LOG_LEVEL	최소 로그 출력 수준을 설정합니다.	**debug**, **info**, **warn**, **error** 또는 **none**	**info**
PARAM_THREESCALE_LOG_JSON	로그 형식이 JSON인지 여부를 제어	**true/false**	**true**
PARAM_THREESCALE_LOG_GRPC	로그에 gRPC 정보가 포함되었는지 여부를 제어	**true/false**	**true**
PARAM_THREESCALE_REPORT_METRICS	3scale 시스템 및 백엔드 지표가 수집되어 Prometheus에 보고되는지 제어	**true/false**	**true**
PARAM_THREESCALE_METRICS_PORT	3scale **/metrics** 끝점을 스크랩할 수 있는 포트를 설정	유효한 포트 번호	**8080**
PARAM_THREESCALE_CACHE_TTL_SECONDS	캐시에서 만료된 항목을 제거하기 전에 대기하는 시간(초)	시간(초)	**300**
PARAM_THREESCALE_CACHE_REFRESH_SECONDS	캐시 요소를 새로 고침하려고 할 때 만료되기 전 시간	시간(초)	**180**
PARAM_THREESCALE_CACHE_ENTRIES_MAX	언제든지 캐시에 저장할 수 있는 항목의 최대 수. 캐싱을 비활성화하려면 **0**으로 설정합니다.	유효한 번호	**1000**
PARAM_THREESCALE_CACHE_REFRESH_RETRIES	캐시 업데이트 루프 중에 연결할 수 없는 호스트가 재시도되는 횟수	유효한 번호	**1**

매개변수	설명	값	기본값
PARAM_THREESCALE_ALLOW_INSECURE_CONN	**3scale** API를 호출할 때 인증서 확인을 건너뛸 수 있습니다. 이 설정 사용은 권장되지 않습니다.	**true/false**	**false**
PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS	3scale System 및 백엔드에 대한 요청을 종료하기 전 대기하는 시간(초)을 설정합니다.	시간(초)	**10**
PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS	연결이 닫히기 전에 연결할 수 있는 최대 시간(초) (+/-10% jitter)을 설정합니다.	시간(초)	60
PARAM_USE_CACHE_BACKEND	true인 경우, 권한 부여 요청에 대해 메모리 내 apisonator 캐시를 생성합니다.	**true/false**	**false**
PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS	백엔드 캐시가 활성화된 경우 3scale에 대해 캐시를 플러싱하는 간격(초)을 설정합니다.	시간(초)	15
PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED	백엔드 캐시가 권한 부여 데이터를 검색할 수 없을 때마다 요청을 거부(닫기)할지, 허용할지(열기) 여부	**true/false**	**true**

1.24.3. status 매개변수

status 매개변수는 서비스 메시의 현재 상태를 설명합니다. 이 정보는 Operator에서 생성하며 읽기 전용입니다.

표 1.37. Istio 상태 매개변수

이름	설명	유형

이름	설명	유형
observedGeneration	가장 최근 조정 중에 컨트롤러가 관찰한 생성입니다. 상태의 정보는 이 특정 오브젝트 생성과 관련이 있습니다. **status.conditions**는 **status.observedGeneration** 필드가 **metadata.generation**과 일치하지 않는 경우 최신 상태가 아닙니다.	integer
annotations	**annotations** 매개변수는 **ServiceMeshControlPlane** 오브젝트에서 배포한 구성 요소 수와 같이 일반적으로 중복된 상태 정보를 저장합니다. 이러한 상태는 아직 JSONPath 표현식에서 오브젝트를 셀 수 없는 **oc** 명령줄 도구에서 사용됩니다.	구성 불가능
readiness	구성 요소 및 소유 리소스의 준비 상태.	string
operatorVersion	이 리소스를 마지막으로 처리하는 Operator의 버전입니다.	string
components	배포된 각 Service Mesh Control Plane 구성 요소의 상태를 표시합니다.	string
appliedSpec	모든 프로필이 적용된 후 구성 옵션의 결과 사양입니다.	**ControlPlaneSpec**
conditions	오브젝트의 현재 상태에 대해 사용 가능한 최신 관찰을 나타냅니다. **reconciled** 는 Operator가 **ServiceMeshControlPlane** 리소스의 구성을 사용하여 배포된 구성 요소의 실제 상태를 조정 완료했음을 나타냅니다. **Installed** 는 Service Mesh Control Plane이 설치되었음을 나타냅니다. **Ready** 는 모든 Service Mesh Control Plane 구성 요소가 준비되었음을 나타냅니다.	string
chartVersion	이 리소스를 위해 마지막으로 처리된 차트의 버전입니다.	string
appliedValues	차트를 생성하는 데 사용된 결과 **values.yaml** 파일입니다.	**ControlPlaneSpec**

1.24.4. 추가 리소스

- ServiceMeshControlPlane 리소스에서 기능을 구성하는 방법에 대한 자세한 내용은 다음 링크를 참조하십시오.
  - 보안
  - 트래픽 관리
  - 메트릭 및 추적

1.25. KIALI 구성 참조

Service Mesh Operator에서 ServiceMeshControlPlane을 생성할 때 Kiali 리소스도 처리합니다. 그런 다음 Kiali Operator는 Kiali 인스턴스를 생성할 때 이 오브젝트를 사용합니다.

1.25.1. SMCP에서 Kiali 구성 지정

ServiceMeshControlPlane 리소스의 addons 섹션에서 Kiali를 구성할 수 있습니다. Kiali는 기본적으로 활성화되어 있습니다. Kiali를 비활성화하려면 spec.addons.kiali.enabled 를 false 로 설정합니다.

다음 두 가지 방법 중 하나로 Kiali 구성을 지정할 수 있습니다.

- spec.addons.kiali.install 의 ServiceMeshControlPlane 리소스에서 Kiali 구성을 지정합니다. 이 방법에는 SMCP에서 전체 Kiali 구성 목록을 사용할 수 없으므로 몇 가지 제한 사항이 있습니다.
- Kiali 인스턴스를 구성하고 배포하고 ServiceMeshControlPlane 리소스의 spec.addons.kiali.name 값으로 Kiali 리소스의 이름을 지정합니다. Service Mesh Control Plane과 동일한 네임스페이스에 CR을 생성해야 합니다(예: istio-system). name 값과 일치하는 Kiali 리소스가 있으면 컨트롤 플레인에서 컨트롤 플레인과 함께 사용할 Kiali 리소스를 구성합니다. 이 방법을 사용하면 Kiali 리소스에서 Kiali 구성을 완전히 사용자 지정할 수 있습니다. 이 방법을 사용하면 Kiali 리소스의 다양한 필드를 Service Mesh Operator, 특히 accessible_namespaces 목록 및 Grafana, Prometheus 및 추적의 끝점에서 덮어씁니다.

Kiali에 대한 SMCP 매개변수 예

```
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
```

표 1.38. ServiceMeshControlPlane Kiali 매개변수

매개변수	설명	값	기본값

`spec:`
`  addons:`
`  kiali:`
`  name:`
	Kiali 사용자 정의 리소스의 이름입니다. **name** 값과 일치하는 Kiali CR이 있으면 Service Mesh Operator는 설치에 해당 CR을 사용합니다. Kiali CR이 없는 경우 Operator는 이 **이름과** SMCP에 지정된 구성 옵션을 사용하여 하나를 생성합니다.	string	**kiali**

`kiali:` `  enabled:`	이 매개변수는 Kiali를 활성화 또는 비활성화합니다. Kiali는 기본적으로 활성화되어 있습니다.	**true/false**	**true**
`kiali:` `  install:`	이름이 Kiali 리소스가 없는 경우 Kiali 리소스를 설치합니다. **addons.kiali.enabled** 가 **false** 로 설정된 경우 **설치** 섹션이 무시됩니다.		

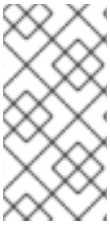
매개변수	설명	값	기본값
kiali: install: dashboard:	Kiali와 함께 제공되는 대시보드에 대한 구성 매개변수입니다.		
kiali: install: dashboard: viewOnly:	이 매개변수는 Kiali 콘솔에 대한 보기 전용 모드를 활성화하거나 비활성화합니다. 보기 전용 모드가 활성화되면 Kiali 콘솔을 사용하여 서비스 메시를 변경할 수 없습니다.	**true/false**	**false**
kiali: install: dashboard: enableGrafana:	**spec.addons.grafana** 구성을 기반으로 구성된 Grafana 엔드포인트입니다.	**true/false**	**true**
kiali: install: dashboard: enablePrometheus:	**spec.addons.prometheus** 구성을 기반으로 구성된 Prometheus 끝점입니다.	**true/false**	**true**
kiali: install: dashboard: enableTracing:	Jaeger 사용자 정의 리소스 구성에 따라 구성된 추적 끝점입니다.	**true/false**	**true**
kiali: install: service:	Kiali 설치와 관련된 Kubernetes 서비스의 구성 매개변수입니다.		
kiali: install: service: metadata:	를 사용하여 리소스에 적용할 추가 메타데이터를 지정합니다.	해당 없음	해당 없음

매개변수	설명	값	기본값
kiali: install: service: metadata: annotations:	을 사용하여 구성 요소 서비스에 적용할 추가 주석을 지정합니다.	string	해당 없음
kiali: install: service: metadata: labels:	을 사용하여 구성 요소의 서비스에 적용할 추가 라벨을 지정합니다.	string	해당 없음
kiali: install: service: ingress:	을 사용하여 OpenShift 경로를 통해 구성 요소 서비스에 액세스하는 데 필요한 세부 정보를 지정합니다.	해당 없음	해당 없음
kiali: install: service: ingress: metadata: annotations:	을 사용하여 구성 요소의 서비스 수신에 적용할 추가 주석을 지정합니다.	string	해당 없음
kiali: install: service: ingress: metadata: labels:	을 사용하여 구성 요소의 서비스 인그레스에 적용할 추가 라벨을 지정합니다.	string	해당 없음
kiali: install: service: ingress: enabled:	을 사용하여 구성 요소와 연결된 서비스의 OpenShift 경로를 사용자 지정합니다.	**true/false**	**true**

매개변수	설명	값	기본값
kiali: install: service: ingress: contextPath:	을 사용하여 서비스의 컨텍스트 경로를 지정합니다.	string	해당 없음
install: service: ingress: hosts:	을 사용하여 OpenShift 경로당 단일 호스트 이름을 지정합니다. 비어 있는 호스트 이름은 경로의 기본 호스트 이름을 의미합니다.	string	해당 없음
install: service: ingress: tls:	을 사용하여 OpenShift 경로의 TLS를 구성합니다.		해당 없음
kiali: install: service: nodePort:	을 사용하여 구성 요소의 서비스 **Values.<component>.service. nodePort.port**에 대한 **nodePort**를 지정합니다.	integer	해당 없음

1.25.2. Kiali 사용자 정의 리소스에서 Kiali 구성 지정

ServiceMeshControlPlane (SMCP) 리소스가 아닌 **Kiali CR(**사용자 정의 리소스**)**에서 **Kiali** 배포를 완전히 사용자 지정할 수 있습니다. 구성이 **SMCP** 외부에 지정되므로 이 구성을 **"**외부 **Kiali"**라고 합니다.



참고

동일한 네임스페이스에 **ServiceMeshControlPlane** 및 **Kiali** 사용자 정의 리소스를 배포해야 합니다. 예를 들면 **istio-system**입니다.

Kiali 인스턴스를 구성하고 배포한 다음 **SMCP** 리소스의 **spec.addons.kiali. name** 값으로 **Kiali** 리소스의 이름을 지정할 수 있습니다. **name** 값과 일치하는 **Kiali CR**이 있으면 **Service Mesh Control Plane**에서 기존 설치를 사용합니다. 이 방법을 사용하면 **Kiali** 구성을 완전히 사용자 지정할 수 있습니다.

1.26. JAEGER 설정 참조

Service Mesh Operator가 ServiceMeshControlPlane 리소스를 생성할 때 분산 추적을 위한 리소스도 배포할 수 있습니다. 서비스 메시는 분산 추적을 위해 Jaeger를 사용합니다.

중요

Jaeger는 FIPS 검증 암호화 모듈을 사용하지 않습니다.

1.26.1. 추적 활성화 및 비활성화

ServiceMeshControlPlane 리소스에 추적 유형 및 샘플링 비율을 지정하여 분산 추적을 활성화합니다.

기본 all-in-one Jaeger 매개변수

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.4
  tracing:
    sampling: 100
    type: Jaeger
```

현재 지원되는 유일한 추적 유형은 Jaeger입니다.

Jaeger는 기본적으로 활성화되어 있습니다. 추적을 비활성화하려면 type을 None으로 설정합니다.

샘플링 비율은 Envoy 프록시가 추적을 생성하는 빈도를 결정합니다. 샘플링 비율 옵션을 사용하여 추적 시스템에 보고되는 요청의 백분율을 제어할 수 있습니다. 메시의 트래픽 및 수집하려는 추적 데이터 양을 기반으로 이 설정을 구성할 수 있습니다. 0.01% 증분을 나타내는 스케일링된 정수로 sampling을 구성합니다. 예를 들어, 값을 10로 설정하면 추적의 0.1%를 샘플링하고, 500으로 설정하면 추적의 5%를 샘플링하며, 10000으로 설정하면 추적의 100%를 샘플링합니다.

참고

SMCP 샘플링 구성 옵션은 Envoy 샘플링 비율을 제어합니다. Jaeger 사용자 정의 리소스에서 Jaeger 추적 샘플링 비율을 구성합니다.

1.26.2. SMCP에서 Jaeger 설정 지정

ServiceMeshControlPlane 리소스의 addons 섹션에서 Jaeger를 구성합니다. 그러나 SMCP에서 구성할 수 있는 몇 가지 제한 사항이 있습니다.

SMCP가 구성 정보를 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator에 전달할 때 allInOne,production 또는 streaming 의 세 가지 배포 전략 중 하나를 트리거합니다.

1.26.3. 분산 추적 플랫폼 배포

분산 추적 플랫폼(Jaeger)에는 사전 정의된 배포 전략이 있습니다. Jaeger 사용자 정의 리소스 (CR) 파일에 배포 전략을 지정합니다. 분산 추적 플랫폼(Jaeger)의 인스턴스를 생성할 때 Red Hat OpenShift distributed tracing Platform(Jaeger) Operator는 이 구성 파일을 사용하여 배포에 필요한 오브젝트를 생성합니다.

Red Hat OpenShift distributed tracing Platform (Jaeger) Operator는 현재 다음과 같은 배포 전략을 지원합니다.

- allInOne(기본값) - 이 전략은 개발, 테스트 및 데모 목적으로 설계되었으며 이는 프로덕션 사용을 목적으로 하지 않습니다. 기본 백엔드 구성 요소인 에이전트, 수집기 및 쿼리 서비스는 모두 메모리 내 스토리지를 사용하도록 (기본적으로) 구성된 단일 실행 파일로 패키지됩니다. SMCP에서 이 배포 전략을 구성할 수 있습니다.

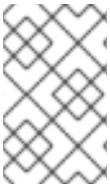
  참고

  메모리 내 스토리지는 영구적이지 않습니다. 즉, Jaeger 인스턴스가 종료, 재시작 또는 교체되면 추적 데이터가 손실됩니다. 각 Pod에 자체 메모리가 있으므로 메모리 내 스토리지를 확장할 수 없습니다. 영구 스토리지의 경우 Elasticsearch를 기본 스토리지로 사용하는 production 또는 streaming 전략을 사용해야 합니다.

- 프로덕션 - 프로덕션 전략은 장기적인 추적 데이터 저장과 더 확장 가능하고 가용성이 높은 아키텍처가 필요한 프로덕션 환경을 위한 것입니다. 따라서 각 백엔드 구성 요소는 별도로 배포됩니다. 에이전트는 조정된 애플리케이션에서 사이드카로 삽입될 수 있습니다. 쿼리 및 수집기 서비스는 지원되는 스토리지 유형(현재 Elasticsearch)으로 구성됩니다. 이러한 각 구성 요소의 여러

인스턴스는 성능 및 복원에 필요한 대로 프로비저닝할 수 있습니다. SMCP에서 이 배포 전략을 구성할 수 있지만 완전히 사용자 정의하려면 Jaeger CR에 구성을 지정하고 SMCP에 링크를 연결해야 합니다.

- 스트리밍 - 스트리밍 전략은 Collector와 Elasticsearch 백엔드 스토리지 간에 적용되는 스트리밍 기능을 제공하여 프로덕션 전략을 보강하도록 설계되었습니다. 이를 통해 높은 로드 상황에서 백엔드 스토리지의 부담을 줄이고 다른 추적 후 처리 기능을 통해 스트리밍 플랫폼 (AMQ Streams/ Kafka)에서 직접 실시간 데이터를 가져올 수 있습니다. SMCP에서 이 배포 전략을 구성할 수 없습니다. Jaeger CR을 구성하고 이를 SMCP에 연결해야 합니다.



참고

스트리밍 전략에는 AMQ Streams에 대한 추가 Red Hat 서브스크립션이 필요합니다.

1.26.3.1. 기본 분산 추적 플랫폼(Jaeger) 배포

Jaeger 구성 옵션을 지정하지 않으면 ServiceMeshControlPlane 리소스는 기본적으로 allInOne Jaeger 배포 전략을 사용합니다. 기본 allInOne 배포 전략을 사용하려면 spec.addons.jaeger.install.storage.type을 Memory로 설정합니다. 기본값을 허용하거나 install에서 추가 구성 옵션을 지정할 수 있습니다.

컨트롤 플레인 기본 Jaeger 매개변수 (메모리)

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.4
  tracing:
    sampling: 10000
    type: Jaeger
  addons:
    jaeger:
      name: jaeger
      install:
        storage:
          type: Memory
```

1.26.3.2. 프로덕션 분산 추적 플랫폼(Jaeger) 배포(최소)

production 배포 전략의 기본 설정을 사용하려면 spec.addons.jaeger.install.storage.type을 Elasticsearch로 설정하고 install에서 추가 구성 옵션을 지정합니다. SMCP는 Elasticsearch 리소스 및 이미지 이름 설정만 지원한다는 점에 유의하십시오.

컨트롤 플레인 기본 Jaeger 매개변수(Elasticsearch)

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.4
  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. 프로덕션 분산 추적 플랫폼(Jaeger) 배포(완전 사용자 지정)

SMCP는 최소한의 Elasticsearch 매개변수만 지원합니다. 프로덕션 환경을 완전히 사용자 지정하고 모든 Elasticsearch 구성 매개변수에 액세스하려면 Jaeger 사용자 정의 리소스(CR)를 사용하여 Jaeger를 구성합니다.

Jaeger 인스턴스를 생성 및 구성하고 spec.addons.jaeger.name 을 Jaeger 인스턴스의 이름으로 설정합니다(예: MyJaegerInstance).

연결된 Jaeger 프로덕션 CR이 있는 컨트롤 플레인

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.4
  tracing:
    sampling: 1000
    type: Jaeger
  addons:
    jaeger:
      name: MyJaegerInstance #name of Jaeger CR
      install:
        storage:
          type: Elasticsearch
        ingress:
          enabled: true
```

1.26.3.4. 스트리밍 Jaeger 배포

streaming 배포 전략을 사용하려면 먼저 Jaeger 인스턴스를 생성 및 구성한 다음 spec.addons.jaeger.name 을 Jaeger 인스턴스의 이름으로 설정합니다(이 예에서는 MyJaegerInstance).

연결된 Jaeger 스트리밍 CR이 있는 컨트롤 플레인

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
spec:
  version: v2.4
  tracing:
    sampling: 1000
    type: Jaeger
  addons:
    jaeger:
      name: MyJaegerInstance  #name of Jaeger CR
```

1.26.4. Jaeger 사용자 정의 리소스에서 Jaeger 구성 지정

ServiceMeshControlPlane (SMCP) 리소스가 아닌 Jaeger CR(사용자 정의 리소스)에서 Jaeger를 구성하여 Jaeger 배포를 완전히 사용자 지정할 수 있습니다. 구성이 SMCP 외부에 지정되므로 이 구성을 "외부 Jaeger"라고도 합니다.

참고

동일한 네임스페이스에 SMCP 및 Jaeger CR을 배포해야 합니다. 예를 들면 istio-system입니다.

독립형 Jaeger 인스턴스를 구성하고 배포한 다음 SMCP 리소스의 spec.addons.jaeger.name 값으로 Jaeger 리소스의 name을 지정할 수 있습니다. name 값과 일치하는 Jaeger CR이 있으면 Service Mesh Control Plane에서 기존 설치를 사용합니다. 이 방법을 사용하면 Jaeger 설정을 완전히 사용자 지정할 수 있습니다.

1.26.4.1. 배포 모범 사례

- Red Hat OpenShift distributed tracing 플랫폼 인스턴스 이름은 고유해야 합니다. Red Hat OpenShift distributed tracing platform (Jaeger) 인스턴스를 여러 개 보유하고 있고 사이드카 삽입 에이전트를 사용하고자 하는 경우 Red Hat OpenShift distributed tracing platform(Jaeger) 인스턴스에 고유한 이름이 있어야 하며 injection 주석은 추적 데이터를 보고해야 하는 Red Hat OpenShift 분산 추적 플랫폼(Jaeger) 인스턴스 이름을 명시적으로 지정해야 합니다.

- 다중 테넌트 구현과 테넌트가 네임스페이스로 구분된 경우 Red Hat OpenShift distributed tracing Platform(Jaeger) 인스턴스를 각 테넌트 네임스페이스에 배포합니다.

영구 스토리지 구성에 대한 자세한 내용은 영구 스토리지 이해 및 선택한 스토리지 옵션에 대한 적절한 구성 항목을 참조하십시오.

1.26.4.2. 서비스 메시에 대한 분산 추적 보안 구성

분산 추적 플랫폼(Jaeger)은 기본 인증에 OAuth를 사용합니다. 그러나 Red Hat OpenShift Service Mesh는 htpasswd 라는 시크릿을 사용하여 Grafana, Kiali 및 분산 추적 플랫폼(Jaeger)과 같은 종속 서비스 간의 통신을 용이하게 합니다. ServiceMeshControlPlane 에서 분산 추적 플랫폼(Jaeger)을 구성할 때 서비스 메시는 htpasswd 를 사용하도록 보안 설정을 자동으로 구성합니다.

Jaeger 사용자 정의 리소스에서 분산 추적 플랫폼(Jaeger) 구성을 지정하는 경우 htpasswd 설정을 수동으로 구성하고 Kiali와 통신할 수 있도록 htpasswd 시크릿이 Jaeger 인스턴스에 마운트되었는지 확인해야 합니다.

1.26.4.2.1. 웹 콘솔에서 서비스 메시에 대한 분산 추적 보안 구성

웹 콘솔에서 Service Mesh와 함께 사용할 Jaeger 리소스를 수정하여 분산 추적 플랫폼(Jaeger) 보안을 구성할 수 있습니다.

사전 요구 사항

- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.
- Red Hat OpenShift Service Mesh Operator가 설치되어 있어야 합니다.
- ServiceMeshControlPlane 이 클러스터에 배포됩니다.
- OpenShift Container Platform 웹 콘솔에 액세스할 수 있습니다.

절차

1. cluster-admin 역할의 사용자로 OpenShift Container Platform 웹 콘솔에 로그인합니다.
2. Operators → 설치된 Operators로 이동합니다.
3. 프로젝트 메뉴를 클릭하고 목록에서 ServiceMeshControlPlane 리소스가 배포되는 프로젝트를 선택합니다(예: istio-system).
4. Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 를 클릭합니다.
5. Operator 세부 정보 페이지에서 Jaeger 탭을 클릭합니다.
6. Jaeger 인스턴스의 이름을 클릭합니다.
7. Jaeger 세부 정보 페이지에서 YAML 탭을 클릭하여 구성을 수정합니다.

8. **Jaeger 사용자 정의 리소스 파일을 편집하여 다음 예와 같이 htpasswd 구성을 추가합니다.**

- **spec.ingress.openshift.htpasswdFile**
- **spec.volumes**
- **spec.volumeMounts**

htpasswd 구성을 표시하는 Jaeger 리소스의 예

```
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.

9.
저장을 클릭합니다.

1.26.4.2.2. 명령줄에서 서비스 메시에 대한 분산 추적 보안 구성

OpenShift CLI(oc)를 실행하여 명령줄에서 **Service Mesh**와 함께 사용할 **Jaeger** 리소스를 수정하여 분산 추적 플랫폼**(Jaeger)** 보안을 구성할 수 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 계정이 있어야 합니다.
- **Red Hat OpenShift Service Mesh Operator**가 설치되어 있어야 합니다.
- **ServiceMeshControlPlane** 이 클러스터에 배포됩니다.
- **OpenShift Container Platform** 버전과 일치하는 **OpenShift CLI(oc)**에 액세스할 수 있습니다.

절차

1. 다음 명령을 실행하여 **cluster-admin** 역할의 사용자로 **OpenShift CLI(oc)**에 로그인합니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 계정이 있어야 합니다.

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

2. 다음 명령을 입력하여 컨트롤 플레인을 설치한 프로젝트(예: **istio-system**)로 변경합니다.

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

3. 다음 명령을 실행하여 **Jaeger** 사용자 정의 리소스 파일을 편집합니다. 여기서 **jaeger.yaml** 은 **Jaeger** 사용자 정의 리소스의 이름입니다.

```
$ oc edit -n tracing-system -f jaeger.yaml
```

4

4. Jaeger 사용자 정의 리소스 파일을 편집하여 다음 예와 같이 htpasswd 구성을 추가합니다.

- spec.ingress.openshift.htpasswdFile
- spec.volumes
- spec.volumeMounts

htpasswd 구성을 표시하는 Jaeger 리소스의 예

```
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. 다음 명령을 실행하여 변경 사항을 적용합니다. 여기서 <jaeger.yaml>은 Jaeger 사용자 정

의 리소스의 이름입니다.

```
$ oc apply -n tracing-system -f <jaeger.yaml>
```

6. 다음 명령을 실행하여 Pod 배포의 진행 상황을 확인합니다.

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

1.26.4.3. 분산 추적 기본 구성 옵션

Jaeger CR(사용자 정의 리소스)은 분산 추적 플랫폼(Jaeger) 리소스를 생성할 때 사용할 아키텍처 및 설정을 정의합니다. 이러한 매개변수를 수정하여 비즈니스 요구에 맞게 분산 추적 플랫폼(Jaeger) 구현을 사용자 지정할 수 있습니다.

Jaeger CR의 일반 YAML 예

```
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: {}
```

표 1.39. Jaeger 매개변수

매개변수	설명	값	기본값
apiVersion:	오브젝트를 생성할 때 사용할 API 버전입니다.	**jaegertracing.io/v1**	**jaegertracing.io/v1**
kind:	생성할 Kubernetes 오브젝트를 정의합니다.	**jaeger**	
metadata:	**name** 문자열, **UID** 및 선택적 **namespace** 를 포함하여 오브젝트를 고유하게 식별할 수 있는 데이터입니다.		OpenShift Container Platform은 **UID**를 자동으로 생성하고 오브젝트가 생성된 프로젝트의 이름으로 **namespace**를 완료합니다.
name:	오브젝트의 이름입니다.	분산 추적 플랫폼 (Jaeger) 인스턴스의 이름입니다.	**jaeger-all-in-one-inmemory**
spec:	생성할 오브젝트의 사양입니다.	분산 추적 플랫폼 (Jaeger) 인스턴스에 대한 모든 구성 매개변수를 포함합니다. 모든 Jaeger 구성 요소에 대한 공통 정의가 필요한 경우 **사양** 노드 아래에 정의됩니다. 정의가 개별 구성 요소와 관련된 경우 **spec/<component> 노드 아래에** 배치됩니다.	해당 없음
strategy:	Jaeger 배포 전략	**allInOne**, **production** 또는 **streaming**	**allInOne**
allInOne:	**allInOne** 이미지는 에이전트, 수집기, 쿼리, Ingester 및 Jaeger UI를 단일 Pod에 배포하므로 이 배포에 대한 구성은 **allInOne** 매개변수의 구성을 중첩해야 합니다.		
agent:	에이전트를 정의하는 구성 옵션입니다.		
collector:	Jaeger 수집기를 정의하는 구성 옵션입니다.		

매개변수	설명	값	기본값
sampling:	추적을 위한 샘플링 전략을 정의하는 구성 옵션입니다.		
storage:	스토리지를 정의하는 구성 옵션입니다. 모든 스토리지 관련 옵션은 **allInOne** 또는 기타 구성 요소 옵션 아래에 있는 대신 **스토리지** 아래에 배치해야 합니다.		
query:	쿼리 서비스를 정의하는 구성 옵션입니다.		
ingester:	Ingester 서비스를 정의하는 구성 옵션입니다.		

다음 예제 YAML은 기본 설정을 사용하여 Red Hat OpenShift distributed tracing platform(Jaeger) 배포를 생성하는 데 필요한 최소값입니다.

예: 필요한 최소 dist-tracing-all-in-one.yaml

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-all-in-one-inmemory
```

1.26.4.4. Jaeger 수집기 구성 옵션

Jaeger 수집기는 추적기에서 캡처한 기간을 수신하여 프로덕션 전략을 사용할 때 영구 Elasticsearch 스토리지에 작성하거나 streaming 전략을 사용할 때 AMQ Streams에 기록하는 구성 요소입니다.

수집기는 상태 비저장이므로 Jaeger 수집기의 많은 인스턴스가 병렬로 실행될 수 있습니다. 수집기는 Elasticsearch 클러스터의 위치를 제외하고 거의 구성이 필요하지 않습니다.

표 1.40. Operator에서 Jaeger Collector를 정의하는 데 사용하는 매개변수

매개변수	설명	값
collector: replicas:	생성할 수집기 복제본 수를 지정합니다.	정수(예: **5**)

표 1.41. 수집기에 전달되는 구성 매개변수

매개변수	설명	값
spec: collector: options: {}	Jaeger 수집기를 정의하는 구성 옵션입니다.	
options: collector: num-workers:	큐에서 가져온 작업자 수입니다.	정수(예: **50**)
options: collector: queue-size:	수집기 큐의 크기입니다.	정수(예: **2000**)
options: kafka: producer: topic: jaeger-spans	**topic** 매개변수는 메시지를 생성하기 위해 수집기에서 사용하는 Kafka 구성과 메시지를 사용하는 Ingester를 식별합니다.	생산자의 레이블입니다.
options: kafka: producer: brokers: my-cluster-kafka-brokers.kafka:9092	메시지를 생성하기 위해 수집기에서 사용하는 Kafka 구성을 식별합니다. 브로커를 지정하지 않고 AMQ Streams 1.4.0 이상이 설치된 경우 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator는 Kafka를 자체 프로비저닝합니다.	
options: log-level:	수집기의 로깅 수준입니다.	가능한 값: **debug**,**info**,**warn**,**error**,**fatal**,**panic**.

1.26.4.5. 분산 추적 샘플링 구성 옵션

Red Hat OpenShift distributed tracing platform(Jaeger) Operator는 원격 샘플러를 사용하도록 구성된 추적기에 제공할 샘플링 전략을 정의하는 데 사용할 수 있습니다.

모든 추적이 생성되지만 소수만 샘플링됩니다. 추적 샘플링은 추가 처리 및 스토리지의 추적을 나타냅니다.

참고

샘플링 결정이 내려지기 때문에 **Envoy** 프록시에서 추적을 시작한 경우 이는 관련이 없습니다. **Jaeger** 샘플링 결정은 클라이언트를 사용하여 애플리케이션에서 추적을 시작할 때만 관련이 있습니다.

서비스에서 추적 컨텍스트가 없는 요청을 수신하면 클라이언트는 새 추적을 시작하고 임의 추적 **ID**를 할당하고 현재 설치된 샘플링 전략에 따라 샘플링 결정을 내립니다. 샘플링 결정은 추적의 모든 후속 요청으로 전파되어 다른 서비스가 샘플링 결정을 다시 수행하지 않습니다.

분산 추적 플랫폼**(Jaeger)** 라이브러리는 다음 샘플을 지원합니다.

- 확률론 **-** 샘플러는 샘플링**(sampling.param)** 속성의 값과 동일한 샘플링의 확률로 임의 샘플링 결정을 내립니다. 예를 들어 **sampling.param=0.1** 샘플은 **10**개의 추적에서 약 **1**개를 사용합니다.

- 속도 제한 **-** 샘플러는 누수된 버킷 속도 제한기를 사용하여 추적을 특정한 일정 속도로 샘플링합니다. 예를 들어 **sampling.param=2.0** 을 사용하면 초당 **2**개의 추적 속도가 포함된 요청을 샘플링합니다.

표 **1.42. Jaeger** 샘플링 옵션

매개변수	설명	값	기본값
spec: sampling: options: {} default_strategy: service_strategy:	추적을 위한 샘플링 전략을 정의하는 구성 옵션입니다.		구성을 제공하지 않으면 수집기는 모든 서비스에 대해 0.001(0.1%)의 기본 확률적 샘플링 정책을 반환합니다.

매개변수	설명	값	기본값
default_strategy: type: service_strategy: type:	사용할 샘플링 전략입니다. 위의 설명을 참조하십시오.	유효한 값은 **probabilistic** 및 **ratelimiting**입니다.	**probabilistic**
default_strategy: param: service_strategy: param:	선택한 샘플링 전략에 대한 매개변수입니다.	10진수 및 정수 값(0, .1, 1, 10)	1

이 예에서는 추적 인스턴스가 샘플링될 가능성이 50%인 비율로 확률적인 기본 샘플링 전략을 정의합니다.

확률 샘플링 예

```
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
```

사용자가 제공하는 구성이 없는 경우 분산 추적 플랫폼(Jaeger)은 다음 설정을 사용합니다.

기본 샘플링

```
spec:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 1
```

1.26.4.6. 분산 추적 스토리지 구성 옵션

spec:storage에서 Collector, Ingester 및 쿼리 서비스에 대한 스토리지를 구성합니다. 이러한 각 구성 요소의 여러 인스턴스는 성능 및 복원에 필요한 대로 프로비저닝할 수 있습니다.

표 1.43. Red Hat OpenShift distributed tracing Platform (Jaeger) Operator에서 분산 추적 스토리지를 정의하는 데 사용하는 일반 스토리지 매개변수

매개변수	설명	값	기본값

spec:
  storage:
  type:
	배포에 사용할 스토리지 유형입니다.	**memory** 또는 **elasticsearch**. 메모리 스토리지는 Pod가 종료되면 데이터가 유지되지 않으므로 개념 환경의 개발, 테스트, 시연 및 검증에만 적합합니다. 프로덕션 환경의 경우 distributed tracing platform(Jaeger)은 영구 스토리지를 위해 Elasticsearch를 지원합니다.	**memory**

storage: secretname:	시크릿 이름(예: **tracing-secret**)입니다.		해당 없음

매개변수	설명	값	기본값
storage: options: {}	스토리지를 정의하는 구성 옵션입니다.		

표 1.44. Elasticsearch 인덱스 정리 매개변수

매개변수	설명	값	기본값
storage: esIndexCleaner: enabled:	Elasticsearch 스토리지를 사용하는 경우 기본적으로 인덱스에서 오래된 추적을 정리하는 작업이 생성됩니다. 이 매개변수는 인덱스 정리 작업을 활성화하거나 비활성화합니다.	**true/ false**	**true**
storage: esIndexCleaner: numberOfDays:	인덱스를 삭제하기 전에 대기하는 날의 수입니다.	정수 값	**7**
storage: esIndexCleaner: schedule:	Elasticsearch 인덱스를 정리하는 빈도에 대한 일정을 정의합니다.	Cron 표현식	"55 23 * * *"

1.26.4.6.1. Elasticsearch 인스턴스 자동 프로비저닝

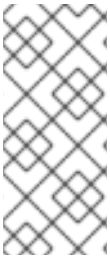
Jaeger 사용자 정의 리소스를 배포할 때 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator는 OpenShift Elasticsearch Operator를 사용하여 사용자 정의 리소스 파일의 스토리지 섹션에 제공된 구성을 기반으로 Elasticsearch 클러스터를 생성합니다. 다음 구성이 설정된 경우 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator는 Elasticsearch를 프로비저닝합니다.

- **spec.storage:type 이 elasticsearch로 설정됩니다.**
- **spec.storage.elasticsearch.doNotProvision 을 false로 설정**
- **spec.storage.options.es.server-urls 가 정의되지 않습니다. 즉 Red Hat Elasticsearch Operator에서 프로비저닝하지 않은 Elasticsearch 인스턴스에 대한 연결이 없습니다.**

Elasticsearch를 프로비저닝할 때 **Red Hat OpenShift distributed tracing Platform (Jaeger) Operator**는 **Jaeger** 사용자 정의 리소스에서 **Elasticsearch** 사용자 정의 리소스 이름을 **spec.storage.elasticsearch.name** 값으로 설정합니다. **spec.storage.elasticsearch.name** 에 대한 값을 지정하지 않으면 **Operator**는 **elasticsearch** 을 사용합니다.

제한 사항

- 네임스페이스당 자체 프로비저닝 **Elasticsearch** 인스턴스가 있는 하나의 분산 추적 플랫폼 **(Jaeger)**만 있을 수 있습니다. **Elasticsearch** 클러스터는 단일 분산 추적 플랫폼**(Jaeger)** 인스턴스 전용입니다.
- 네임스페이스당 **Elasticsearch**가 하나만 있을 수 있습니다.



참고

Elasticsearch를 **OpenShift Logging**의 일부로 이미 설치한 경우 **Red Hat OpenShift distributed tracing Platform (Jaeger) Operator**에서 설치된 **OpenShift Elasticsearch Operator**를 사용하여 스토리지를 프로비저닝할 수 있습니다.

다음 구성 매개변수는 **OpenShift Elasticsearch Operator**를 사용하여 **Red Hat OpenShift distributed tracing Platform(Jaeger) Operator**가 생성한 인스턴스인 *자체 프로비저닝* **Elasticsearch** 인스턴스에 대한 것입니다. 구성 파일의 **spec:storage:elasticsearch**에서 자체 프로비저닝 **Elasticsearch**에 대한 구성 옵션을 지정합니다.

표 1.45. Elasticsearch 리소스 구성 매개변수

매개변수	설명	값	기본값
elasticsearch: properties: doNotProvision:	을 사용하여 Red Hat OpenShift distributed tracing Platform(Jaeger) Operator에서 Elasticsearch 인스턴스를 프로비저닝해야 하는지 여부를 지정합니다.	**true/false**	**true**

매개변수	설명	값	기본값
elasticsearch: properties: name:	Elasticsearch 인스턴스의 이름입니다. Red Hat OpenShift distributed tracing platform(Jaeger) Operator는 이 매개변수에 지정된 Elasticsearch 인스턴스를 사용하여 Elasticsearch에 연결합니다.	string	**elasticsearch**
elasticsearch: nodeCount:	Elasticsearch 노드 수입니다. 고가용성의 경우 최소 3개의 노드를 사용합니다. "스플릿 브레인" 문제가 발생할 수 있으므로 2개의 노드를 사용하지 마십시오.	정수 값입니다. 예를 들면 개념 증명 = 1, 최소 배포 =3입니다.	3
elasticsearch: resources: requests: cpu:	사용자 환경 구성에 따른 요청에 대한 중앙 처리 단위 수입니다.	코어 또는 밀리코어(예: 200m, 0.5, 1)에 지정되어 있습니다. 예를 들면 개념 증명 = 500m, 최소 배포 =1입니다.	1
elasticsearch: resources: requests: memory:	환경 구성에 따른 요청에 사용 가능한 메모리입니다.	바이트 단위로 지정됩니다(예: 200Ki, 50Mi, 5Gi). 예를 들면 개념 증명 = 1Gi, 최소 배포 = 16Gi*입니다.	16Gi
elasticsearch: resources: limits: cpu:	사용자 환경 구성에 따른 중앙 처리 장치 수에 대한 제한입니다.	코어 또는 밀리코어(예: 200m, 0.5, 1)에 지정되어 있습니다. 예를 들면 개념 증명 = 500m, 최소 배포 =1입니다.	
elasticsearch: resources: limits: memory:	사용자 환경 구성에 따라 사용 가능한 메모리 제한입니다.	바이트 단위로 지정됩니다(예: 200Ki, 50Mi, 5Gi). 예를 들면 개념 증명 = 1Gi, 최소 배포 = 16Gi*입니다.	

매개변수	설명	값	기본값
elasticsearch: redundancyPolicy:	데이터 복제 정책은 Elasticsearch shard가 클러스터의 데이터 노드에 복제되는 방법을 정의합니다. 지정하지 않으면 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator가 노드 수에 따라 가장 적절한 복제를 자동으로 결정합니다.	**ZeroRedundancy**(replica shard 없음), **SingleRedundancy**(하나의 replica shard), **MultipleRedundancy**(각 인덱스가 데이터 노드의 반을 넘어 분산됨), **FullRedundancy**(각 인덱스가 클러스터의 모든 데이터 노드에 전체적으로 복제됨).	
elasticsearch: useCertManagement:	을 사용하여 Red Hat Elasticsearch Operator의 인증서 관리 기능을 사용해야 하는지 여부를 지정합니다. 이 기능은 OpenShift Container Platform 4.7에서 Red Hat OpenShift 5.2에 대한 로깅 하위 시스템에 추가되었으며 새로운 Jaeger 배포의 기본 설정입니다.	**true/false**	**true**
	*각 Elasticsearch 노드는 더 낮은 메모리 설정으로 작동할 수 있지만 프로덕션 배포에는 권장되지 않습니다. 프로덕션 용도의 경우 기본적으로 각 Pod에 할당된 16Gi 미만이 있어야 하지만 Pod당 최대 64Gi까지 할당할 수도 있습니다.		

프로덕션 스토리지 예

```
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
```

영구 스토리지가 있는 스토리지 예:

```
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

영구 스토리지 구성. 이 경우 **AWS gp2에 5Gi** 크기가 있습니다. 값을 지정하지 않으면 분산 추적 플랫폼(Jaeger)에서 **emptyDir** 을 사용합니다. **OpenShift Elasticsearch Operator**는 분산 추적 플랫폼(Jaeger) 인스턴스에서 제거되지 않는 **PersistentVolumeClaim** 및 **PersistentVolume** 을 프로비저닝합니다. 동일한 이름과 네임스페이스로 분산 추적 플랫폼(Jaeger) 인스턴스를 생성하면 동일한 볼륨을 마운트할 수 있습니다.

1.26.4.6.2. 기존 Elasticsearch 인스턴스에 연결

분산 추적 플랫폼이 있는 스토리지에 기존 **Elasticsearch** 클러스터를 사용할 수 있습니다. *외부* **Elasticsearch** 인스턴스라고도 하는 기존 **Elasticsearch** 클러스터는 **Red Hat OpenShift distributed tracing Platform (Jaeger) Operator** 또는 **Red Hat Elasticsearch Operator**에서 설치하지 않은 인스턴스입니다.

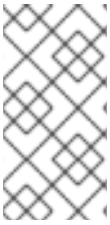
Jaeger 사용자 정의 리소스를 배포할 때 다음 구성이 설정된 경우 **Red Hat OpenShift distributed tracing Platform (Jaeger) Operator**는 **Elasticsearch**를 프로비저닝하지 않습니다.

- spec.storage.elasticsearch.doNotProvision set to true
- spec.storage.options.es.server-urls 의 값이 있습니다.
- spec.storage.elasticsearch.name 에 값이 있거나 Elasticsearch 인스턴스 이름이 elasticsearch 인 경우

Red Hat OpenShift distributed tracing platform(Jaeger) Operator는 spec.storage.elasticsearch.name 에 지정된 Elasticsearch 인스턴스를 사용하여 Elasticsearch에 연결합니다.

제한 사항

- 분산 추적 플랫폼(Jaeger)과 OpenShift Container Platform 로깅 Elasticsearch 인스턴스를 공유하거나 재사용할 수 없습니다. Elasticsearch 클러스터는 단일 분산 추적 플랫폼(Jaeger) 인스턴스 전용입니다.



참고

Red Hat은 외부 Elasticsearch 인스턴스를 지원하지 않습니다. Customer Portal에서 테스트된 통합 매트릭스를 검토할 수 있습니다.

다음 구성 매개변수는 *외부* Elasticsearch 인스턴스 라고도 하는 기존 Elasticsearch 인스턴스에 대한 것입니다. 이 경우 spec:storage:options:es 사용자 지정 리소스 파일에서 Elasticsearch에 대한 구성 옵션을 지정합니다.

표 1.46. 일반 ES 구성 매개변수

매개변수	설명	값	기본값
es: server-urls:	Elasticsearch 인스턴스의 URL입니다.	Elasticsearch 서버의 정규화된 도메인 이름입니다.	http://elasticsearch.<namespace>.svc:9200

매개변수	설명	값	기본값
es: max-doc-count:	Elasticsearch 쿼리에서 반환하는 최대 문서 수입니다. 이는 집계에도 적용됩니다. **es.max-doc-count** 및 **es.max-num-spans**를 모두 설정하면 Elasticsearch에서 이 둘 중 작은 값을 사용합니다.		10000
es: max-num-spans:	[**더 이상 사용되지 않음** - 향후 릴리스에서 제거되며 대신 **es.max-doc-count**를 사용합니다.] Elasticsearch에서 쿼리 당 한 번에 가져올 최대 기간 수입니다. **es.max-num-spans** 및 **es.max-doc-count**를 모두 설정하면 Elasticsearch는 이 둘 중 작은 값을 사용합니다.		10000
es: max-span-age:	Elasticsearch에서 기간에 대한 최대 조회 수입니다.		72h0m0s
es: sniffer:	Elasticsearch의 스니퍼 구성입니다. 클라이언트는 스니핑 프로세스를 사용하여 모든 노드를 자동으로 찾습니다. 기본적으로 비활성되어 있습니다.	**true**/ **false**	**false**
es: sniffer-tls-enabled:	Elasticsearch 클러스터를 스니핑할 때 TLS를 활성화하는 옵션입니다. 클라이언트는 스니핑 프로세스를 사용하여 모든 노드를 자동으로 찾습니다. 기본적으로 비활성화되어 있습니다.	**true**/ **false**	**false**
es: timeout:	쿼리에 사용되는 시간 제한입니다. 0으로 설정하면 시간 제한이 없습니다.		0s

매개변수	설명	값	기본값
es: username:	Elasticsearch에 필요한 사용자 이름입니다. 기본 인증은 지정된 경우 CA도 로드합니다. **es.password**도 참조하십시오.		
es: password:	Elasticsearch에 필요한 암호입니다. **es.username**도 참조하십시오.		
es: version:	주요 Elasticsearch 버전입니다. 지정하지 않으면 Elasticsearch에서 값을 자동으로 탐지합니다.		0

표 1.47. ES 데이터 복제 매개변수

매개변수	설명	값	기본값
es: num-replicas:	Elasticsearch의 인덱스당 복제본 수입니다.		1
es: num-shards:	Elasticsearch의 인덱스당 shard 수입니다.		5

표 1.48. ES 인덱스 구성 매개변수

매개변수	설명	값	기본값
es: create-index-templates:	**true**로 설정할 때 애플리케이션 시작 시 인덱스 템플릿을 자동으로 생성합니다. 템플릿이 수동으로 설치되면 **false**로 설정합니다.	**true**/ **false**	**true**

매개변수	설명	값	기본값
es: index-prefix:	분산 추적 플랫폼 (Jaeger) 인덱스의 선택적 접두사입니다. 예를 들어 이 값을 "production"으로 설정하면 "production-tracing-*"라는 인덱스가 생성됩니다.		

표 1.49. ES 일괄 프로세서 구성 매개변수

매개변수	설명	값	기본값
es: bulk: actions:	대규모 프로세서가 디스크에 업데이트를 커밋하기 전에 큐에 추가할 수 있는 요청 수입니다.		1000
es: bulk: flush-interval:	다른 임계값에 관계없이 대규모 요청이 커밋된 후 **time.Duration**입니다. 대규모 프로세서 플러시 간격을 비활성화하려면 이를 0으로 설정합니다.		200ms
es: bulk: size:	대규모 프로세서가 디스크에 업데이트를 커밋하기 전에 대규모 요청이 수행할 수 있는 바이트 수입니다.		5000000
es: bulk: workers:	Elasticsearch에 대규모 요청을 수신하고 커밋할 수 있는 작업자 수입니다.		1

표 1.50. ES TLS 구성 매개변수

매개변수	설명	값	기본값
es: tls: ca:	원격 서버를 확인하는 데 사용되는 TLS 인증 기관 (CA) 파일의 경로입니다.		기본적으로 시스템 신뢰 저장소를 사용합니다.

매개변수	설명	값	기본값

es:
  tls:
  cert:
	이 프로세스를 원격 서버로 식별하는 데 사용되는 TLS 인증서 파일의 경로입니다.		

es:
  tls:
  enabled:
	원격 서버에 연결할 때 TLS(Transport Layer Security)를 활성화합니다. 기본적으로 비활성되어 있습니다.	**true/ false**	**false**

es:
  tls:
  key:
	이 프로세스를 원격 서버로 식별하는 데 사용되는 TLS 개인 키 파일의 경로입니다.		

es:
  tls:
  server-name:
	원격 서버의 인증서에서 예상 TLS 서버 이름을 재정의합니다.		

es: token-file:	전달자 토큰이 포함된 파일의 경로입니다. 이 플래그는 지정된 경우 CA(인증 기관) 파일도 로드합니다.		

표 1.51. ES 아카이브 구성 매개변수

매개변수	설명	값	기본값

es-archive:
  bulk:
  actions:
	대규모 프로세서가 디스크에 업데이트를 커밋하기 전에 큐에 추가할 수 있는 요청 수입니다.		0

es-archive:
  bulk:
  flush-interval:
	다른 임계값에 관계없이 대규모 요청이 커밋된 후 **time.Duration**입니다. 대규모 프로세서 플러시 간격을 비활성화하려면 이를 0으로 설정합니다.		0s

매개변수	설명	값	기본값
es-archive: bulk: size:	대규모 프로세서가 디스크에 업데이트를 커밋하기 전에 대규모 요청이 수행할 수 있는 바이트 수입니다.		0
es-archive: bulk: workers:	Elasticsearch에 대규모 요청을 수신하고 커밋할 수 있는 작업자 수입니다.		0
es-archive: create-index-templates:	**true**로 설정할 때 애플리케이션 시작 시 인덱스 템플릿을 자동으로 생성합니다. 템플릿이 수동으로 설치되면 **false**로 설정합니다.	**true/ false**	**false**
es-archive: enabled:	추가 스토리지를 활성화합니다.	**true/ false**	**false**
es-archive: index-prefix:	분산 추적 플랫폼(Jaeger) 인덱스의 선택적 접두사입니다. 예를 들어 이 값을 "production"으로 설정하면 "production-tracing-*"라는 인덱스가 생성됩니다.		
es-archive: max-doc-count:	Elasticsearch 쿼리에서 반환하는 최대 문서 수입니다. 이는 집계에도 적용됩니다.		0
es-archive: max-num-spans:	[**더 이상 사용되지 않음** - 향후 릴리스에서 제거되며 대신 **es-archive.max-doc-count**를 사용합니다.] Elasticsearch에서 쿼리 당 한 번에 가져올 최대 기간 수입니다.		0
es-archive: max-span-age:	Elasticsearch에서 기간에 대한 최대 조회 수입니다.		0s

매개변수	설명	값	기본값
es-archive: num-replicas:	Elasticsearch의 인덱스 당 복제본 수입니다.		0
es-archive: num-shards:	Elasticsearch의 인덱스 당 shard 수입니다.		0
es-archive: password:	Elasticsearch에 필요한 암호입니다. **es.username**도 참조하십시오.		
es-archive: server-urls:	Elasticsearch 서버의 쉼표로 구분된 목록입니다. 정규화된 URL로 지정해야 합니다(예: http://localhost:9200).		
es-archive: sniffer:	Elasticsearch의 스니퍼 구성입니다. 클라이언트는 스니핑 프로세스를 사용하여 모든 노드를 자동으로 찾습니다. 기본적으로 비활성되어 있습니다.	**true**/ **false**	**false**
es-archive: sniffer-tls-enabled:	Elasticsearch 클러스터를 스니핑할 때 TLS를 활성화하는 옵션입니다. 클라이언트는 스니핑 프로세스를 사용하여 모든 노드를 자동으로 찾습니다. 기본적으로 비활성되어 있습니다.	**true**/ **false**	**false**
es-archive: timeout:	쿼리에 사용되는 시간 제한입니다. 0으로 설정하면 시간 제한이 없습니다.		0s
es-archive: tls: ca:	원격 서버를 확인하는 데 사용되는 TLS 인증 기관(CA) 파일의 경로입니다.		기본적으로 시스템 신뢰 저장소를 사용합니다.

매개변수	설명	값	기본값
es-archive: tls: cert:	이 프로세스를 원격 서버로 식별하는 데 사용되는 TLS 인증서 파일의 경로입니다.		
es-archive: tls: enabled:	원격 서버에 연결할 때 TLS(Transport Layer Security)를 활성화합니다. 기본적으로 비활성되어 있습니다.	**true/ false**	**false**
es-archive: tls: key:	이 프로세스를 원격 서버로 식별하는 데 사용되는 TLS 개인 키 파일의 경로입니다.		
es-archive: tls: server-name:	원격 서버의 인증서에서 예상 TLS 서버 이름을 재정의합니다.		
es-archive: token-file:	전달자 토큰이 포함된 파일의 경로입니다. 이 플래그는 지정된 경우 CA(인증 기관) 파일도 로드합니다.		
es-archive: username:	Elasticsearch에 필요한 사용자 이름입니다. 기본 인증은 지정된 경우 CA도 로드합니다. **es-archive.password**도 참조하십시오.		
es-archive: version:	주요 Elasticsearch 버전입니다. 지정하지 않으면 Elasticsearch에서 값을 자동으로 탐지합니다.		0

볼륨 마운트가 있는 스토리지 예

```
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
```

다음 예는 시크릿에 저장된 볼륨 및 사용자/암호에서 마운트된 **TLS CA** 인증서가 포함된 외부 **Elasticsearch** 클러스터를 사용하는 **Jaeger CR**을 보여줍니다.

외부 **Elasticsearch** 예

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200 (1)
        index-prefix: my-prefix
        tls: (2)
          ca: /es/certificates/ca.crt
    secretName: tracing-secret (3)
  volumeMounts: (4)
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
```

```
volumes:
 - name: certificates
   secret:
     secretName: quickstart-es-http-certs-public
```

1

기본 네임스페이스에서 실행되는 **Elasticsearch** 서비스에 대한 **URL**입니다.

2

TLS 구성입니다. 이 경우 **CA** 인증서만 해당하지만 상호 **TLS**를 사용하는 경우 **es.tls.key** 및 **es.tls.cert**를 포함할 수 있습니다.

3

환경 변수 **ES_PASSWORD** 및 **ES_USERNAME**을 정의하는 시크릿입니다. **kubectl create secret generic tracing-secret --from-literal=ES_PASSWORD=changeme --from-literal=ES_USERNAME=elastic**에 의해 생성됨

4

모든 스토리지 구성 요소에 마운트되는 볼륨 마운트 및 볼륨입니다.

1.26.4.7. Elasticsearch로 인증서 관리

Red Hat Elasticsearch Operator를 사용하여 인증서를 생성하고 관리할 수 있습니다. **Red Hat Elasticsearch Operator**를 사용하여 인증서를 관리하면 여러 **Jaeger** 수집기와 함께 단일 **Elasticsearch** 클러스터를 사용할 수 있습니다.

중요

Elasticsearch로 인증서 관리는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(**SLA**)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

버전 2.4부터 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator는 Elasticsearch 사용자 정의 리소스에서 다음 주석을 사용하여 인증서 생성을 Red Hat Elasticsearch Operator에 위임합니다.

- logging.openshift.io/elasticsearch-cert-management: "true"
- logging.openshift.io/elasticsearch-cert.jaeger-<shared-es-node-name>: "user.jaeger"
- logging.openshift.io/elasticsearch-cert.curator-<shared-es-node-name>: "system.logging.curator"

여기서 & lt;shared-es-node-name& gt;은 Elasticsearch 노드의 이름입니다. 예를 들어 custom-es 라는 Elasticsearch 노드를 생성하는 경우 사용자 정의 리소스는 다음 예와 같을 수 있습니다.

주석을 표시하는 Elasticsearch CR의 예

```
apiVersion: logging.openshift.io/v1
kind: Elasticsearch
metadata:
  annotations:
    logging.openshift.io/elasticsearch-cert-management: "true"
    logging.openshift.io/elasticsearch-cert.jaeger-custom-es: "user.jaeger"
    logging.openshift.io/elasticsearch-cert.curator-custom-es: "system.logging.curator"
  name: custom-es
spec:
  managementState: Managed
  nodeSpec:
    resources:
      limits:
        memory: 16Gi
      requests:
        cpu: 1
        memory: 16Gi
  nodes:
    - nodeCount: 3
      proxyResources: {}
      resources: {}
      roles:
        - master
        - client
        - data
      storage: {}
  redundancyPolicy: ZeroRedundancy
```

사전 요구 사항

- OpenShift Container Platform 4.7
- Red Hat OpenShift 5.2의 로깅 하위 시스템
- Elasticsearch 노드와 Jaeger 인스턴스는 동일한 네임스페이스에 배포해야 합니다. 예를 들면 tracing-system 입니다.

Jaeger 사용자 정의 리소스에서 spec.storage.elasticsearch.useCertManagement 를 true 로 설정하여 인증서 관리를 활성화합니다.

useCertManagement를 보여주는 예

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      name: custom-es
      doNotProvision: true
      useCertManagement: true
```

Red Hat OpenShift distributed tracing platform(Jaeger) Operator는 Elasticsearch 사용자 정의 리소스 이름을 Elasticsearch 사용자 정의 리소스에서 Jaeger 사용자 정의 리소스의 값으로 설정합니다.

인증서는 Red Hat Elasticsearch Operator에 의해 프로비저닝되며 Red Hat OpenShift distributed tracing platform (Jaeger) Operator가 인증서를 삽입합니다.

OpenShift Container Platform을 사용하여 **Elasticsearch**를 구성하는 방법에 대한 자세한 내용은 로그 저장소 구성 또는 분산 추적 구성 및 배포를 참조하십시오.

1.26.4.8. 쿼리 구성 옵션

쿼리는 스토리지에서 추적을 검색하고 사용자 인터페이스에서 표시하도록 호스팅하는 서비스입니다.

표 1.52. Red Hat OpenShift distributed tracing Platform (Jaeger) Operator에서 쿼리를 정의하는 데 사용하는 매개변수

매개변수	설명	값	기본값
spec: query: replicas:	생성할 쿼리 복제본 수를 지정합니다.	예: 정수 **2**)	

표 1.53. 쿼리에 전달된 구성 매개변수

매개변수	설명	값	기본값
spec: query: options: {}	쿼리 서비스를 정의하는 구성 옵션입니다.		
options: log-level:	쿼리의 로깅 수준입니다.	가능한 값: **debug**,**info**,**warn**,**error**,**fatal**,**panic**.	
options: query: base-path:	모든 jaeger-query HTTP 경로의 기본 경로는 root 값이 아닌 값으로 설정할 수 있습니다(예: **/jaeger**는 모든 UI URL을 **/jaeger**로 시작합니다). 이는 리버스 프록시 뒤에서 jaeger-query를 실행할 때 유용할 수 있습니다.	/<path>	

샘플 쿼리 구성

```
apiVersion: jaegertracing.io/v1
```

```
kind: "Jaeger"
metadata:
  name: "my-jaeger"
spec:
  strategy: allInOne
  allInOne:
    options:
      log-level: debug
      query:
        base-path: /jaeger
```

1.26.4.9. Ingester 구성 옵션

Ingester는 Kafka 항목에서 읽고 Elasticsearch 스토리지 백엔드에 쓰는 서비스입니다. allInOne 또는 production 배포 전략을 사용하는 경우 Ingester 서비스를 구성할 필요가 없습니다.

표 1.54. Ingester에 전달된 Jaeger 매개변수

매개변수	설명	값

`spec:`
`  ingester:`
`  options: {}`
	Ingester 서비스를 정의하는 구성 옵션입니다.	

`options:` `  deadlockInterval:`	Ingester가 종료되기 전에 메시지를 기다려야 하는 간격(초 또는 분)을 지정합니다. 교착 상태 간격은 시스템 초기화 중에 메시지가 없을 때 Ingester를 종료하지 않도록 기본적으로 (set to **0**) 비활성화됩니다.	분 및 초(예: **1m0s**)입니다. 기본값은 **0**입니다.

`options:`
`  kafka:`
`  consumer:`
`  topic:`
	topic 매개변수는 메시지를 생성하는 수집기와 메시지를 사용하는 Ingester에서 사용하는 Kafka 구성을 식별합니다.	소비자의 레이블입니다. 예를 들면 **jaeger-spans**입니다.

`options:`
`  kafka:`
`  consumer:`
`  brokers:`
	메시지를 사용하려면 Ingester에서 사용하는 Kafka 구성을 식별합니다.	브로커의 레이블은 예를 들면 **my-cluster-kafka-brokers.kafka:9092**입니다.

매개변수	설명	값
options: log-level:	Ingester의 로깅 수준입니다.	가능한 값: **debug**,**info**,**warn**,**error**,**fatal**,**dpanic**,**panic**.

스트리밍 수집기 및 Ingester 예

```
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. 서비스 메시 설치 제거

기존 OpenShift Container Platform 인스턴스에서 Red Hat OpenShift Service Mesh를 설치 제거하고 해당 리소스를 제거하려면 컨트롤 플레인을 삭제하고 Operator를 삭제하고 명령을 실행하여 일부 리소스를 수동으로 제거해야 합니다.

1.27.1. Red Hat OpenShift Service Mesh Control Plane 제거

기존 OpenShift Container Platform 인스턴스에서 Service Mesh를 설치 제거하려면 먼저 Service Mesh Control Plane과 Operator를 삭제합니다. 그런 다음 명령을 실행하여 남은 리소스를 제거합니다.

1.27.1.1. 웹 콘솔을 사용하여 Service Mesh Control Plane 제거

웹 콘솔을 사용하여 Red Hat OpenShift Service Mesh Control Plane을 제거할 수 있습니다.

절차

1. OpenShift Container Platform 웹 콘솔에 로그인합니다.
2. 프로젝트 메뉴를 클릭하고 Service Mesh Control Plane을 설치한 프로젝트(예: istio-system)를 선택합니다.
3. Operators → 설치된 Operator로 이동합니다.
4. 제공된 API에서 Service Mesh Control Plane을 클릭합니다.
5. ServiceMeshControlPlane 메뉴
   
   를 클릭합니다.
6. Service Mesh Control Plane 삭제를 클릭합니다.
7. 확인 대화 상자에서 삭제를 클릭하여 ServiceMeshControlPlane을 삭제합니다.

1.27.1.2. CLI를 사용하여 Service Mesh Control Plane 제거

CLI를 사용하여 Red Hat OpenShift Service Mesh Control Plane을 제거할 수 있습니다. 이 예제에서 istio-system은 컨트롤 플레인 프로젝트의 이름입니다.

절차

1

I. OpenShift Container Platform CLI에 로그인합니다.

2. 다음 명령을 실행하여 ServiceMeshMemberRoll 리소스를 삭제합니다.

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

3. 이 명령을 실행하여 설치된 ServiceMeshControlPlane의 이름을 검색합니다.

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

4. <name_of_custom_resource>을 이전 명령의 출력으로 바꾸고, 이 명령을 실행하여 사용자 정의 리소스를 삭제합니다.

```
$ oc delete smcp -n istio-system <name_of_custom_resource>
```

1.27.2. 설치된 Operator 제거

Red Hat OpenShift Service Mesh를 성공적으로 제거하려면 Operator를 제거해야 합니다. Red Hat OpenShift Service Mesh Operator를 제거한 후 Kiali Operator, Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 및 OpenShift Elasticsearch Operator를 제거해야 합니다.

1.27.2.1. Operator 제거

Red Hat OpenShift Service Mesh를 구성하는 Operator를 제거하려면 다음 절차를 따르십시오. 다음 각 Operator에 대해 단계를 반복합니다.

- Red Hat OpenShift Service Mesh
- Kiali
- Red Hat OpenShift distributed tracing platform (Jaeger)
- OpenShift Elasticsearch

절차

1. OpenShift Container Platform 웹 콘솔에 로그인합니다.

2. Operator → 설치된 Operator 페이지에서 스크롤하거나 이름별 필터링에 키워드를 입력하여 각 Operator를 찾습니다. 그런 다음 Operator 이름을 클릭합니다.

3. Operator 상세 정보 페이지의 작업 메뉴에서 Operator 제거를 선택합니다. 프롬프트에 따라 각 Operator를 제거합니다.

1.27.3. Operator 리소스 정리

OpenShift Container Platform 웹 콘솔을 사용하여 Red Hat OpenShift Service Mesh Operator를 제거한 후 남은 리소스를 수동으로 제거할 수 있습니다.

사전 요구 사항

- 클러스터 관리 권한이 있는 계정. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.

- OpenShift CLI(oc)에 액세스합니다.

절차

1. OpenShift Container Platform CLI에 클러스터 관리자로 로그인합니다.

2. Operator를 제거한 후 다음 명령을 실행하여 리소스를 정리합니다. 서비스 메시 없이 독립형 서비스로 분산 추적 플랫폼(Jaeger)을 계속 사용하려면 Jaeger 리소스를 삭제하지 마십시오.

참고

OpenShift Elasticsearch Operator는 기본적으로 openshift-operators-redhat 에 설치됩니다. 다른 Operator는 기본적으로 openshift-operators 네임스페이스에 설치됩니다. 다른 네임스페이스에 Operators를 설치한 경우openshift-operators를 Red Hat OpenShift Service Mesh Operator가 설치된 프로젝트의 이름으로 교체합니다.

```
$ oc delete validatingwebhookconfiguration/openshift-operators.servicemesh-resources.maistra.io
```

```
$ oc delete mutatingwebhookconfiguration/openshift-operators.servicemesh-resources.maistra.io
```

```
$ oc delete svc maistra-admission-controller -n openshift-operators
```

```
$ oc -n openshift-operators delete ds -lmaistra-version
```

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

```
$ oc delete clusterrole istio-view istio-edit
```

```
$ oc delete clusterrole jaegers.jaegertracing.io-v1-admin jaegers.jaegertracing.io-v1-crdview jaegers.jaegertracing.io-v1-edit jaegers.jaegertracing.io-v1-view
```

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

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

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

```
$ oc delete crds jaegers.jaegertracing.io
```

```
$ oc delete cm -n openshift-operators maistra-operator-cabundle
```

```
$ oc delete cm -n openshift-operators istio-cni-config istio-cni-config-v2-3
```

```
$ oc delete sa -n openshift-operators istio-cni
```

2장. SERVICE MESH 1.X

2.1. 서비스 메시 릴리스 노트

주의

더 이상 지원되지 않는 **Red Hat OpenShift Service Mesh** 릴리스에 대한 문서를 보고 있습니다.

서비스 메시 버전 **1.0** 및 **1.1** 컨트롤 플레인은 더 이상 지원되지 않습니다. 서비스 메시 컨트롤 플레인 업그레이드에 대한 자세한 내용은 서비스 메시 업그레이드를 참조하십시오.

특정 **Red Hat OpenShift Service Mesh** 릴리스의 지원 상태에 대한 자세한 내용은 제품 라이프사이클 페이지를 참조하십시오.

2.1.1. 보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(**master**), 슬레이브(**slave**), 블랙리스트(**blacklist**), 화이트리스트(**whitelist**) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 **CTO Chris Wright**의 메시지를 참조하십시오.

2.1.2. Red Hat OpenShift Service Mesh 소개

Red Hat OpenShift Service Mesh는 애플리케이션에서 중앙 집중식 제어 지점을 생성하여 마이크로서비스 아키텍처에서 다양한 문제에 대응합니다. 애플리케이션 코드를 변경하지 않고도 기존 분산 애플리케이션에 투명한 레이어를 추가합니다.

마이크로 서비스 아키텍처는 엔터프라이즈 애플리케이션의 작업을 모듈식 서비스로 분할하므로 확장 및 유지 관리를 더 쉽게 수행할 수 있습니다. 그러나 마이크로 서비스 아키텍처에 구축된 엔터프라이즈 애플리케이션이 크기와 복잡성이 증가함에 따라 마이크로 서비스 아키텍처의 이해 및 관리가 어려워집니다. 서비스 메시는 서비스 간 트래픽을 캡처하거나 차단하거나 다른 서비스에 대한 새 요청을 리디렉트 또는 생성하여 이러한 아키텍처의 문제에 대응할 수 있습니다.

오픈 소스 **Istio project**를 기반으로 하는 **Service Mesh**는 배포된 서비스 네트워크를 쉽게 구축할 수

있는 방법을 제공하여 검색, 로드 밸런싱, 서비스 간 인증, 실패 복구, 지표 및 모니터링을 지원합니다. 또한 서비스 메시는 A/B 테스트, 카나리아 릴리스, 액세스 제어, 엔드 투 엔드 인증을 비롯한 복잡한 운영 기능을 제공합니다.

2.1.3. 지원 요청

이 문서에 설명된 절차 또는 일반적으로 OpenShift Container Platform에 문제가 발생하는 경우 Red Hat 고객 포털에 액세스하십시오. 고객 포털에서 다음을 수행할 수 있습니다.

- Red Hat 제품과 관련된 기사 및 솔루션에 대한 Red Hat 지식베이스를 검색하거나 살펴볼 수 있습니다.
- Red Hat 지원에 대한 지원 케이스 제출할 수 있습니다.
- 다른 제품 설명서에 액세스 가능합니다.

클러스터 문제를 식별하기 위해 OpenShift Cluster Manager Hybrid Cloud Console 에서 Insights를 사용할 수 있습니다. Insights는 문제에 대한 세부 정보 및 문제 해결 방법에 대한 정보를 제공합니다.

이 문서를 개선하기 위한 제안이 있거나 오류를 발견한 경우 가장 관련 있는 문서 구성 요소에 대한 Jira 문제를 제출하십시오. 섹션 이름 및 OpenShift Container Platform 버전과 같은 구체적인 정보를 제공합니다.

지원 사례를 여는 경우 클러스터에 대한 디버깅 정보를 Red Hat 지원에 제공하면 도움이 됩니다.

must-gather 도구를 사용하면 가상 머신 및 Red Hat OpenShift Service Mesh 관련 기타 데이터를 포함하여 OpenShift Container Platform 클러스터에 대한 진단 정보를 수집할 수 있습니다.

즉각 지원을 받을 수 있도록 OpenShift Container Platform 및 Red Hat OpenShift Service Mesh 둘 다에 대한 진단 정보를 제공하십시오.

2.1.3.1. must-gather 툴 정보

oc adm must-gather CLI 명령은 다음과 같은 문제를 디버깅하는 데 필요할 가능성이 높은 클러스터에서 정보를 수집합니다.

- 리소스 정의
- 서비스 로그

기본적으로 **oc adm must-gather** 명령은 기본 플러그인 이미지를 사용하고 **./must-gather.local** 에 씁니다.

또는 다음 섹션에 설명된 대로 적절한 인수로 명령을 실행하여 특정 정보를 수집할 수 있습니다.

- 하나 이상의 특정 기능과 관련된 데이터를 수집하려면 다음 섹션에 나열된 대로 이미지와 함께 **--image** 인수를 사용합니다.

  예를 들어 다음과 같습니다.

  ```
  $ oc adm must-gather  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.11.0
  ```

- 감사 로그를 수집하려면 다음 섹션에 설명된 대로 **-- /usr/bin/gather_audit_logs** 인수를 사용합니다.

  예를 들어 다음과 같습니다.

  ```
  $ oc adm must-gather -- /usr/bin/gather_audit_logs
  ```

  참고

  감사 로그는 파일 크기를 줄이기 위해 기본 정보 세트의 일부로 수집되지 않습니다.

oc adm must-gather 을 실행하면 클러스터의 새 프로젝트에 임의의 이름이 있는 새 **Pod**가 생성됩니다. 해당 **Pod**에 대한 데이터가 수집되어 **must-gather.local**로 시작하는 새 디렉터리에 저장됩니다. 이 디렉터리는 현재 작업 중인 디렉터리에 생성되어 있습니다.

예를 들어 다음과 같습니다.

```
NAMESPACE                 NAME              READY   STATUS     RESTARTS     AGE
...
openshift-must-gather-5drcj    must-gather-bklx4    2/2     Running    0          72s
openshift-must-gather-5drcj    must-gather-s8sdh    2/2     Running    0          72s
...
```

2.1.3.2. 사전 요구 사항

- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- OpenShift Container Platform CLI(oc)가 설치되어 있어야 합니다.

2.1.3.3. 서비스 메시 데이터 수집 정보

oc adm must-gather CLI 명령을 사용하면 Red Hat OpenShift Service Mesh와 연관된 기능 및 오브젝트를 포함하여 클러스터에 대한 정보를 수집할 수 있습니다.

사전 요구 사항

- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- OpenShift Container Platform CLI(oc)가 설치되어 있어야 합니다.

절차

1. must-gather을 사용하여 Red Hat OpenShift Service Mesh 데이터를 수집하려면 Red Hat OpenShift Service Mesh 이미지를 지정해야 합니다.

```
$ oc adm must-gather --image=registry.redhat.io/openshift-service-mesh/istio-must-gather-rhel8:2.4
```

2. must-gather 로 특정 Service Mesh Control Plane 네임스페이스에 대한 Red Hat OpenShift Service Mesh 데이터를 수집하려면 Red Hat OpenShift Service Mesh 이미지 및 네임스페이스를 지정해야 합니다. 이 예에서는 수집 후 < namespace >를 istio-system 과 같은 서비스 메시 컨트롤 플레인 네임스페이스로 교체합니다.

```
$ oc adm must-gather --image=registry.redhat.io/openshift-service-mesh/istio-must-gather-rhel8:2.4 gather <namespace>
```

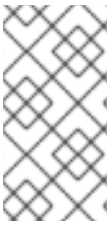
이렇게 하면 다음 항목이 포함된 로컬 디렉터리가 생성됩니다.

- **Istio Operator** 네임스페이스 및 해당 하위 오브젝트
- 모든 컨트롤 플레인 네임스페이스 및 해당 하위 오브젝트
- 모든 네임스페이스 및 해당 하위 오브젝트는 모든 서비스 메시에 속하는 오브젝트
- 모든 **Istio CRD**(사용자 정의 리소스 정의)
- 지정된 네임스페이스의 **VirtualServices**와 같은 모든 **Istio CRD** 오브젝트
- 모든 **Istio Webhook**

2.1.4. Red Hat OpenShift Service Mesh 지원 구성

다음은 **Red Hat OpenShift Service Mesh**에 지원되는 구성입니다.

- **OpenShift Container Platform** 버전 **4.6** 이상



참고

OpenShift Online 및 **Red Hat OpenShift Dedicated**는 **Red Hat OpenShift Service Mesh**에서 지원되지 않습니다.

- 배포가 통합되지 않은 단일 **OpenShift Container Platform** 클러스터에 포함되어야 합니다.
- 이번 **Red Hat OpenShift Service Mesh** 릴리스는 **OpenShift Container Platform x86_64**에서만 사용 가능합니다.
- 이 릴리스에서는 모든 **Service Mesh** 구성 요소가 작동하는 **OpenShift Container Platform**

클러스터에 포함된 구성만 지원합니다. 클러스터 외부에 있거나 멀티 클러스터 시나리오에 있는 마이크로 서비스 관리는 지원하지 않습니다.

- 이 릴리스는 가상 머신과 같은 외부 서비스를 통합하지 않는 구성만 지원합니다.

Red Hat OpenShift Service Mesh 라이프사이클 및 지원되는 구성에 대한 자세한 내용은 지원 정책을 참조하십시오.

2.1.4.1. Red Hat OpenShift Service Mesh에서 Kiali에 지원되는 구성

- **Kiali Observation Console**은 **Chrome, Edge, Firefox** 또는 **Safari** 브라우저의 두 가지 최신 버전에서만 지원됩니다.

2.1.4.2. 지원되는 Mixer 어댑터

- 이 릴리스에서는 다음 **Mixer** 어댑터만 지원합니다.
  - **3scale Istio** 어댑터

2.1.5. 새로운 기능

Red Hat OpenShift Service Mesh는 서비스 네트워크 전반에서 여러 주요 기능을 균일하게 제공합니다.

- 트래픽 관리 - 서비스 간 트래픽 및 **API** 호출 흐름을 제어하고, 호출을 더 안정적으로 만들며, 불리한 조건에서도 네트워크를 보다 견고하게 만듭니다.
- 서비스 **ID** 및 보안 - 메시에서 확인 가능한 **ID**로 서비스를 제공하고 다양한 수준의 신뢰도를 갖춘 네트워크를 통해 전달될 때 서비스 트래픽을 보호할 수 있는 기능을 제공합니다.
- 정책 강화- 서비스 간 상호 작용에 조직 정책을 적용하여 액세스 정책이 시행되고 리소스가 소비자 간에 공정하게 배포되도록 합니다. 애플리케이션 코드를 변경하는 것이 아니라 메시를 구성하여 정책 변경을 수행합니다.
- **Telemetry** - 서비스 간의 종속성과 트래픽 속성 및 흐름을 이해하여 문제를 신속하게 식별할 수 있는 기능을 제공합니다.

2.1.5.1. Red Hat OpenShift Service Mesh 1.1.18.2 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)를 제공합니다.

2.1.5.1.1. Red Hat OpenShift Service Mesh 버전 1.1.18.2에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.4.10
Jaeger	1.30.2
Kiali	1.12.21.1
3scale Istio 어댑터	1.0.0

2.1.5.2. Red Hat OpenShift Service Mesh 1.1.18.1 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)를 제공합니다.

2.1.5.2.1. Red Hat OpenShift Service Mesh 버전 1.1.18.1에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.4.10
Jaeger	1.30.2
Kiali	1.12.20.1
3scale Istio 어댑터	1.0.0

2.1.5.3. Red Hat OpenShift Service Mesh 1.1.18 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures)를 제공합니다.

2.1.5.3.1. Red Hat OpenShift Service Mesh 버전 1.1.18에 포함된 구성 요소 버전

구성 요소	버전
Istio	1.4.10
Jaeger	1.24.0
Kiali	1.12.18
3scale Istio 어댑터	1.0.0

2.1.5.4. Red Hat OpenShift Service Mesh 1.1.17.1 새 기능

이번 **Red Hat OpenShift Service Mesh** 릴리스는 **CVE(Common Vulnerabilities and Exposures)**를 제공합니다.

2.1.5.4.1. Red Hat OpenShift Service Mesh가 URI 조각을 처리하는 방법 변경

Red Hat OpenShift Service Mesh에는 원격으로 악용할 수 있는 취약점인 **CVE-2021-39156** 이 포함되어 있습니다. 여기서 **URI** 경로의 **#** 문자로 시작하는 **URL** 섹션이 **Istio URI** 경로 기반 권한 부여 정책을 우회할 수 있습니다. 예를 들어 **Istio** 권한 부여 정책은 **URI** 경로 **/user/profile** 에 전송된 요청을 거부합니다. 취약한 버전에서 **URI** 경로 **/user/profile#section1** 이 있는 요청이 거부 정책 및 경로를 백엔드로 바이패스합니다(정규화된 **URI** 경로 **/user/profile%23section1)**으로 인해 보안 문제가 발생할 수 있습니다.

DENY 작업 및 **operation.paths.paths**와 함께 권한 부여 정책을 사용하거나 **ALLOW** 작업 및 **operation.notPaths** 와 함께 권한 부여 정책을 사용하는 경우 이 취약점의 영향을 받습니다.

완화 기능을 사용하면 권한 부여 및 라우팅 전에 요청 **URI**의 조각 부분이 제거됩니다. 이렇게 하면 **URI**에 조각이 있는 요청이 내용 부분이 없는 **URI**를 기반으로 하는 권한 부여 정책을 우회하지 않습니다.

2.1.5.4.2. 권한 부여 정책에 필요한 업데이트

Istio는 호스트 이름 자체와 일치하는 모든 포트 모두에 대한 호스트 이름을 생성합니다. 예를 들어 **"httpbin.foo"** 호스트의 가상 서비스 또는 게이트웨이는 **"httpbin.foo** 및 **httpbin.foo:*"**과 일치하는 구성을 생성합니다. 그러나 정확한 권한 부여 정책은 **hosts** 또는 **notHosts** 필드에 지정된 정확한 문자열과만 일치합니다.

규칙에서 호스트 또는 **notHosts** 를 결정하는 데 정확한 문자열 비교를 사용하여 **AuthorizationPolicy** 리소스가 있는 경우 클러스터에 영향을 받습니다.

정확한 일치 대신 접두사를 사용하도록 권한 부여 정책 규칙을 업데이트해야 합니다. 예를 들어 첫 번

째 AuthorizationPolicy 예제에서 hosts: ["httpbin.com"] 을 hosts: ["httpbin.com:*"] 로 바꿉니다.

접두사 일치를 사용하는 첫 번째 AuthorizationPolicy 예

```
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:*"]
```

접두사 일치를 사용하는 AuthorizationPolicy 예

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

2.1.5.5. Red Hat OpenShift Service Mesh 1.1.17 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

2.1.5.6. Red Hat OpenShift Service Mesh 1.1.16 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

2.1.5.7. Red Hat OpenShift Service Mesh 1.1.15 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

2.1.5.8. Red Hat OpenShift Service Mesh 1.1.14 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

중요

CVE-2021-29492 및 CVE-2021-31920 문제를 해결하려면 수동 단계가 완료되어야 합니다.

2.1.5.8.1. CVE-2021-29492 및 CVE-2021-31920에서 필요한 수동 업데이트

Istio에는 경로 기반 권한 부여 규칙이 사용될 때 여러 슬래시 또는 이스케이프된 슬래시 문자 (%2F` or %5C`)가 있는 HTTP 요청 경로가 잠재적으로 Istio 권한 부여 정책을 우회할 수 있는 원격으로 악용 가능한 취약점이 포함되어 있습니다.

예를 들어 Istio 클러스터 관리자가 경로 /admin에 있는 요청을 거부하도록 권한 부여 DENY 정책을 정의한다고 가정합니다. //admin URL 경로에 전송된 요청이 권한 부여 정책에서 거부되지 않습니다.

RFC 3986에 따르면 여러 개의 슬래시가 있는 //admin 경로는 기술적으로 /admin과 다른 경로로 처리되어야 합니다. 그러나 일부 백엔드 서비스는 여러 슬래시를 단일 슬래시로 병합하여 URL 경로를 정규화하도록 선택합니다. 이로 인해 권한 부여 정책(//admin이 /admin과 일치하지 않음)을 우회할 수 있으며 사용자는 백엔드의 /admin 경로에 있는 리소스에 액세스할 수 있습니다. 결과적으로 이는 보안 문제로 나타날 수 있습니다.

ALLOW action + notPaths 필드 또는 DENY action + paths field 경로 필드 패턴을 사용하는 권한 부여 정책이 있는 경우 클러스터는 이 취약점의 영향을 받습니다. 이러한 패턴은 예기치 않은 정책 우회에 취약합니다.

다음과 같은 경우 클러스터는 이 취약점의 영향을 받지 않습니다.

- 권한 부여 정책이 없습니다.
- 권한 부여 정책은 **paths** 또는 **notPaths** 필드를 정의하지 않습니다.
- 권한 부여 정책은 **ALLOW action + paths** 필드 또는 **DENY action + notPaths** 필드 패턴을 사용합니다. 이러한 패턴은 정책 우회 대신 예기치 않은 거부를 유발할 수 있습니다. 이러한 경우 업그레이드는 선택 사항입니다.

참고

경로 정규화를 위한 **Red Hat OpenShift Service Mesh** 구성 위치는 **Istio** 구성과 다릅니다.

2.1.5.8.2. 경로 정규화 구성 업데이트

Istio 권한 부여 정책은 **HTTP** 요청의 **URL** 경로를 기반으로 할 수 있습니다. **URI** 정규화라고도 하는 경로 정규화는 들어오는 요청의 경로를 수정 및 표준화하여 정규화된 경로를 표준 방식으로 처리할 수 있도록 합니다. 구문적으로 경로 정규화 후에는 다른 경로가 동일할 수 있습니다.

Istio는 권한 부여 정책에 대해 평가하고 요청을 라우팅하기 전에 요청 경로에서 다음 정규화 체계를 지원합니다.

표 **2.1.** 정규화 체계

옵션	설명	예제	참고
NONE	정규화는 수행되지 않습니다. Envoy가 수신한 모든 항목은 정확히 그대로 모든 백엔드 서비스에 전달됩니다.	**../%2FA../b**는 권한 부여 정책에 의해 평가되고 서비스로 전송됩니다.	이 설정은 CVE-2021-31920에 취약합니다.

옵션	설명	예제	참고
BASE	현재 이는 Istio의 **기본** 설치에 사용되는 옵션입니다. 이로 인해 Envoy 프록시에 **normalize_path** 옵션을 적용하며, RFC 3986에 따라 백슬래시를 슬래시로 변환하는 추가 정규화를 따릅니다.	**/a/../b**는 **/b**로 정규화됩니다. **\da**는 **/da**로 정규화됩니다.	이 설정은 CVE-2021-31920에 취약합니다.
MERGE_SLASHES	*BASE* 정규화 후 슬래시가 병합됩니다.	**/a//b**는 **/a/b**로 정규화됩니다.	CVE-2021-31920을 완화하려면 이 설정으로 업데이트합니다.
DECODE_AND_MERGE_SLASHES	기본적으로 모든 트래픽을 허용할 때 가장 엄격한 설정입니다. 이 설정은 권한 부여 정책 경로를 철저하게 테스트해야 한다는 경고와 함께 권장됩니다. 백분율로 인코딩된 슬래시 및 백슬래시 문자(**%2F**, **%2f**, **%5C** 및 **%5c**)는 **MERGE_SLASHES** 정규화 전에 / 또는 \로 디코딩됩니다.	**/a%2fb**는 **/a/b**로 정규화됩니다.	CVE-2021-31920을 완화하려면 이 설정으로 업데이트합니다. 이 설정은 더 안전하지만 애플리케이션이 중단될 수도 있습니다. 프로덕션에 배포하기 전에 애플리케이션을 테스트합니다.

정규화 알고리즘은 다음 순서로 수행됩니다.

1. **백분율로 디코딩된 %2F, %2f, %5C 및 %5c.**
2. **Envoy의 normalize_path 옵션에 의해 구현된 RFC 3986 및 기타 정규화입니다.**
3. **슬래시를 병합합니다.**

주의

이러한 정규화 옵션은 **HTTP** 표준 및 일반적인 업계 관행의 권장 사항을 나타내지만 애플리케이션은 원하는 방식으로 **URL**을 해석할 수 있습니다. 거부 정책을 사용할 때 애플리케이션이 작동하는 방식을 이해해야 합니다.

2.1.5.8.3. 경로 정규화 구성 예

Envoy는 백엔드 서비스의 기대치와 일치하도록 요청 경로를 표준화하여 시스템 보안에 매우 중요합니다. 다음 예제는 시스템을 구성하기 위한 참조로 사용할 수 있습니다. 정규화된 **URL** 경로 또는 **NONE**이 선택된 경우 원래 **URL** 경로는 다음과 같습니다.

1. 권한 부여 정책을 확인하는 데 사용됩니다.
2. 백엔드 애플리케이션으로 전달됩니다.

표 2.2. 구성 예

애플리케이션 조건	선택...
프록시를 사용하여 정규화를 수행합니다.	**BASE**,**MERGE_SLASHES** 또는 **DECODE_AND_MERGE_SLASHES**
RFC 3986을 기반으로 요청 경로를 정규화하고 슬래시를 병합하지 않습니다.	**BASE**
RFC 3986을 기반으로 요청 경로를 정규화하고 슬래시를 병합하지만 백분율로 인코딩된 슬래시를 디코딩하지는 않습니다.	**MERGE_SLASHES**
RFC 3986을 기반으로 요청 경로를 표준화하고, 백분율로 인코딩된 슬래시를 디코딩하고, 슬래시를 병합합니다.	**DECODE_AND_MERGE_SLASHES**
프로세스는 RFC 3986과 호환되지 않는 방식으로 요청 경로를 처리합니다.	**NONE**

2.1.5.8.4. 경로 정규화를 위해 SMCP 구성

Red Hat OpenShift Service Mesh에 대한 경로 정규화를 구성하려면 **ServiceMeshControlPlane**에

서 다음을 지정합니다. 시스템 설정을 결정하는 데 도움이 되도록 구성 예제를 사용합니다.

SMCP v1 pathNormalization

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

2.1.5.9. Red Hat OpenShift Service Mesh 1.1.13 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

2.1.5.10. Red Hat OpenShift Service Mesh 1.1.12 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

2.1.5.11. Red Hat OpenShift Service Mesh 1.1.11 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

2.1.5.12. Red Hat OpenShift Service Mesh 1.1.10 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

2.1.5.13. Red Hat OpenShift Service Mesh 1.1.9 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

2.1.5.14. Red Hat OpenShift Service Mesh 1.1.8 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

2.1.5.15. Red Hat OpenShift Service Mesh 1.1.7 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

2.1.5.16. Red Hat OpenShift Service Mesh 1.1.6 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

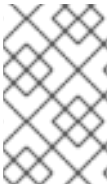
2.1.5.17. Red Hat OpenShift Service Mesh 1.1.5 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

또한 이 릴리스에는 암호화 제품군 구성에 대한 지원이 추가되었습니다.

2.1.5.18. Red Hat OpenShift Service Mesh 1.1.4 새 기능

이번 Red Hat OpenShift Service Mesh 릴리스는 CVE(Common Vulnerabilities and Exposures) 및 버그 수정을 제공합니다.

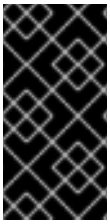


참고

CVE-2020-8663 문제를 해결하려면 수동 단계가 완료되어야 합니다.

2.1.5.18.1. CVE-2020-8663에서 필요한 수동 업데이트

CVE-2020-8663: envoy: Resource exhaustion when accepting too many connections에 대한 수정이 다운스트림 연결에 구성 가능한 제한을 추가했습니다. 이 제한에 대한 구성 옵션은 이 취약점을 완화하도록 설정되어야 합니다.



중요

이러한 수동 단계는 **Red Hat OpenShift Service Mesh**의 **1.1** 버전을 사용하든 **1.0** 버전을 사용하든 이 **CVE**를 완화하는 데 필요합니다.

이 새로운 설정 옵션은 **overload.global_downstream_max_connections**라고 하며 프록시 **runtime** 설정으로 구성할 수 있습니다. 수신 게이트웨이에서 제한을 구성하려면 다음 단계를 수행합니다.

프로세스

1. 다음 텍스트로 **bootstrap-override.json**이라는 파일을 생성하여 프록시에서 부트스트랩 템플릿을 재정의하고 디스크의 런타임 구성을 로드하도록 적용합니다.

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

2. **bootstrap-override.json** 파일에서 시크릿을 생성하여 **<SMCPnamespace>**를 **Service Mesh Control Plane(SMCP)**을 생성한 네임스페이스로 바꿉니다.

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

3. **SMCP** 구성을 업데이트하여 재정의를 활성화합니다.

업데이트된 **SMCP** 구성 예 **#1**

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    gateways:
      istio-ingressgateway:
        env:
          ISTIO_BOOTSTRAP_OVERRIDE: /var/lib/istio/envoy/custom-
bootstrap/bootstrap-override.json
        secretVolumes:
        - mountPath: /var/lib/istio/envoy/custom-bootstrap
          name: custom-bootstrap
          secretName: gateway-bootstrap
```

-

4. 새 구성 옵션을 설정하려면 **overload.global_downstream_max_connections** 설정에 필요한 값을 보유한 시크릿을 생성합니다. 다음 예제에서는 값 **10000**을 사용합니다.

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

5. **SMCP**를 다시 업데이트하여 **Envoy**가 런타임 구성을 찾고 있는 위치에 시크릿을 다시 마운트합니다.

업데이트된 SMCP 구성 예 #2

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

2.1.5.18.2. Elasticsearch 5에서 Elasticsearch 6으로 업그레이드

Elasticsearch 5에서 **Elasticsearch 6**으로 업데이트할 때 인증서 문제로 인해 **Jaeger** 인스턴스를 삭제한 다음 **Jaeger** 인스턴스를 다시 생성해야 합니다. **Jaeger** 인스턴스 트리거를 다시 생성하여 새 인증서 세트를 생성합니다. 영구 스토리지를 사용하는 경우, 새 **Jaeger** 인스턴스의 **Jaeger** 이름과 네임스페이스

가 삭제된 **Jaeger** 인스턴스와 동일하다면 새 **Jaeger** 인스턴스에 대해 동일한 볼륨을 마운트할 수 있습니다.

Jaeger가 **Red Hat Service Mesh**의 일부로 설치된 경우 프로세스

1. **Jaeger** 사용자 정의 리소스 파일의 이름을 결정합니다.

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

다음과 같은 내용이 표시됩니다.

```
NAME     AGE
jaeger   3d21h
```

2. 생성된 사용자 정의 리소스 파일을 임시 디렉터리에 복사합니다.

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

3. **Jaeger** 인스턴스를 삭제합니다.

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

4. 사용자 정의 리소스 파일의 사본에서 **Jaeger** 인스턴스를 재생성합니다.

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

5. 생성된 사용자 정의 리소스 파일의 사본을 삭제합니다.

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

Jaeger가 **Red Hat Service Mesh**의 일부로 설치되지 않은 경우 프로세스

시작하기 전에 **Jaeger** 사용자 정의 리소스 파일의 사본을 만듭니다.

1. 사용자 정의 리소스 파일을 삭제하여 **Jaeger** 인스턴스를 삭제합니다.

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

-

예를 들면 다음과 같습니다.

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

2. 사용자 정의 리소스 파일의 백업 사본에서 **Jaeger** 인스턴스를 재생성합니다.

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

3. **Pod**가 다시 시작되었는지 확인합니다.

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

2.1.5.19. Red Hat OpenShift Service Mesh 1.1.3 새 기능

이번 **Red Hat OpenShift Service Mesh** 릴리스는 **CVE(Common Vulnerabilities and Exposures)** 및 버그 수정을 제공합니다.

2.1.5.20. Red Hat OpenShift Service Mesh 1.1.2 새 기능

이번 **Red Hat OpenShift Service Mesh** 릴리스는 보안 취약점을 해결합니다.

2.1.5.21. Red Hat OpenShift Service Mesh 1.1.1 새 기능

이번 **Red Hat OpenShift Service Mesh** 릴리스는 연결이 중단된 설치를 지원합니다.

2.1.5.22. Red Hat OpenShift Service Mesh 1.1.0 새 기능

이번 **Red Hat OpenShift Service Mesh** 릴리스에는 **Istio 1.4.6** 및 **Jaeger 1.17.1**에 대한 지원이 추가되었습니다.

2.1.5.22.1. 1.0에서 1.1로 수동 업데이트

Red Hat OpenShift Service Mesh 1.0에서 **1.1**로 업데이트하는 경우 컨트롤 플레인 구성 요소를 새 버전으로 업데이트하기 위해 **ServiceMeshControlPlane** 리소스를 업데이트해야 합니다.

1

i. 웹 콘솔에서 **Red Hat OpenShift Service Mesh Operator**를 클릭합니다.

2. 프로젝트 메뉴를 클릭하고 **ServiceMeshControlPlane**이 목록에서 배포되는 프로젝트를 선택합니다(예: **istio-system**).

3. 컨트롤 플레인의 이름을 클릭합니다(예: **basic-install**).

4. **YAML** 을 클릭하고 **ServiceMeshControlPlane**리소스의 **spec:**에 버전 필드를 추가합니다. 예를 들어 **Red Hat OpenShift Service Mesh 1.1.0**으로 업데이트하려면 **version: v1.1**을 추가합니다.

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

version 필드는 설치할 **Service Mesh** 버전을 지정하고 기본값을 최신 사용 가능한 최신 버전으로 지정합니다.

참고

Red Hat OpenShift Service Mesh v1.0 지원은 **2020**년 **10**월에 종료되었습니다. **v1.1** 또는 **v2.0**으로 업그레이드해야 합니다.

2.1.6. 더 이상 사용되지 않는 기능

이전 릴리스에서 사용 가능하던 일부 기능이 더 이상 사용되지 않거나 삭제되었습니다.

더 이상 사용되지 않는 기능은 여전히 **OpenShift Container Platform**에 포함되어 있으며 계속 지원됩니다. 그러나 이 기능은 향후 릴리스에서 제거될 예정이므로 새로운 배포에는 사용하지 않는 것이 좋습니다.

2.1.6.1. Red Hat OpenShift Service Mesh 1.1.5의 중단된 기능

다음 사용자 정의 리소스는 릴리스 **1.1.5**에서 더 이상 사용되지 않으며 릴리스 **1.1.12**에서 제외되었습니다.

-

Policy - Policy 리소스는 더 이상 사용되지 않으며 향후 릴리스에서는 **PeerAuthentication** 리소스로 대체됩니다.

- **MeshPolicy - MeshPolicy** 리소스는 더 이상 사용되지 않으며 향후 릴리스에서는 **PeerAuthentication** 리소스로 대체됩니다.

- **v1alpha1 RBAC API -The v1alpha1 RBAC** 정책은 **v1beta1 AuthorizationPolicy**에서 더 이상 사용되지 않습니다. **RBAC(Role Based Access Control)**는 **ServiceRole** 및 **ServiceRoleBinding** 오브젝트를 정의합니다.

  - **ServiceRole**

  - **ServiceRoleBinding**

- **RbacConfig - RbacConfig**는 **Istio RBAC** 동작을 제어하기 위해 사용자 정의 리소스 정의를 구현합니다.

  - **ClusterRbacConfig(Red Hat OpenShift Service Mesh** 버전 **1.0** 이전**)**

  - **ServiceMeshRbacConfig(Red Hat OpenShift Service Mesh** 버전 **1.0** 이상**)**

- **Kiali**에서는 **login** 및 **LDAP** 전략이 더 이상 사용되지 않습니다. 향후 버전에서는 **OpenID** 공급자를 사용한 인증을 도입할 예정입니다.

다음 구성 요소는 이 릴리스에서 더 이상 사용되지 않으며 향후 릴리스에서 **Istiod** 구성 요소로 대체됩니다.

- **Mixer -** 액세스 제어 및 사용 정책

- **Pilot -** 서비스 검색 및 프록시 구성

- **Citadel -** 인증서 생성

- Galley - 구성 검증 및 배포

2.1.7. 확인된 문제

이러한 제한 사항은 Red Hat OpenShift Service Mesh에 있습니다.

- Red Hat OpenShift Service Mesh는 IPv6를 지원하지 않습니다. 업스트림 Istio 프로젝트에서 지원하지 않거나 OpenShift Container Platform에서 완전히 지원하지 않기 때문입니다.

- 그래프 레이아웃 - 애플리케이션 아키텍처 및 표시할 데이터(그래프 노드 및 상호 작용 수)에 따라 Kiali 그래프의 레이아웃이 다르게 렌더링됩니다. 모든 상황에 적합하게 렌더링되는 단일 레이아웃을 만드는 것이 불가능하지는 않지만 어렵기 때문에 Kiali는 다양한 레이아웃 옵션을 제공합니다. 다른 레이아웃을 선택하려면 그래프 설정 메뉴에서 다른 레이아웃 스키마를 선택할 수 있습니다.

- Kiali 콘솔의 Jaeger 및 Grafana와 같은 관련 서비스에 처음 액세스하는 경우 인증서를 수락하고 OpenShift Container Platform 로그인 자격 증명을 사용하여 다시 인증해야 합니다. 이것은 프레임워크가 콘솔에 포함된 페이지를 표시하는 방법에 문제가 있기 때문입니다.

2.1.7.1. 서비스 메시의 알려진 문제

이는 Red Hat OpenShift Service Mesh에서 알려진 문제입니다.

- Jaeger/Kiali Operator 업그레이드가 operator 보류로 차단됩니다. Service Mesh 1.0.x가 설치된 Jaeger 또는 Kiali Operator를 업그레이드할 때 Operator 상태가 보류 중으로 표시됩니다.

  해결방법: 자세한 내용은 연결된 기술 자료 문서를 참조하십시오.

- Istio-14743 이 Red Hat OpenShift Service Mesh 릴리스의 기반이 되는 Istio 버전의 제한으로 인해 현재 Service Mesh와 호환되지 않는 여러 애플리케이션이 있습니다. 자세한 내용은 링크 커뮤니티 관련 문제를 참조하십시오.

- MAISTRA-858 Istio 1.1.x와 관련된 더 이상 사용하지 않는 옵션 및 구성을 설명하는 다음과 같은 Envoy 로그 메시지가 예상됩니다.

  -

[2019-06-03 07:03:28.943][19][warning][misc] [external/envoy/source/common/protobuf/utility.cc:129] Using deprecated option 'envoy.api.v2.listener.Filter.config'. 이 구성은 곧 Envoy에서 삭제될 예정입니다.

- [2019-08-12 22:12:59.001][13][warning][misc] [external/envoy/source/common/protobuf/utility.cc:174] Using deprecated option 'envoy.api.v2.Listener.use_original_dst' from file lds.proto. 이 구성은 곧 Envoy에서 삭제될 예정입니다.

- MAISTRA-806 제거된 Istio Operator pod로 인해 메시 및 CNI가 배포되지 않습니다.

  해결방법: 제어 창을 배포하는 동안 istio-operator Pod가 제거되면 제거된 istio-operator Pod를 삭제합니다.

- MAISTRA-681 컨트롤 플레인에 네임스페이스가 많은 경우 성능 문제가 발생할 수 있습니다.

- MAISTRA-465 Maistra Operator가 Operator 지표에 대한 서비스를 생성하지 못합니다.

- MAISTRA-453 새 프로젝트를 생성하고 즉시 pod를 배포하면 사이드카 삽입이 발생하지 않습니다. pod가 생성되기 전에 Operator에서 maistra.io/member-of를 추가하지 못하므로 사이드카 삽입을 수행하려면 pod를 삭제하고 다시 생성해야 합니다.

- MAISTRA-158 동일한 호스트 이름을 참조하는 여러 게이트웨이를 적용하면 모든 게이트웨이가 작동을 중지합니다.

2.1.7.2. Kiali의 확인된 문제

참고

Kiali의 새로운 문제는 OpenShift Service Mesh 프로젝트에서 생성되어야 하며 Component가 Kiali로 설정되어야 합니다.

다음은 Kiali에서 알려진 문제입니다.

- KIALI-2206 처음으로 Kiali 콘솔에 액세스했을 때 Kiali에 대해 캐시된 브라우저 데이터가 없

는 경우 Kiali 서비스 상세 정보 페이지의 Metrics 탭에 있는 'Grafana에서 보기' 링크가 잘못된 위치로 리디렉션됩니다. 이 문제가 발생하는 유일한 상황은 Kiali에 처음 액세스하는 경우입니다.

- KIALI-507 Kiali는 Internet Explorer 11을 지원하지 않습니다. 기본 프레임워크가 Internet Explorer를 지원하지 않기 때문입니다. Kiali 콘솔에 액세스하려면 Chrome, Edge, Firefox 또는 Safari 브라우저의 두 가지 최신 버전 중 하나를 사용하십시오.

2.1.8. 수정된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

2.1.8.1. 서비스 메시의 수정된 문제

- MAISTRA-2371 listerInformer에서 tombstones를 처리합니다. 업데이트된 캐시 코드베이스는 네임스페이스 캐시에서 집계된 캐시로 이벤트를 변환할 때 tombstones를 처리하지 않아 go 루틴에서 패닉이 발생했습니다.
- OSSM-542 Galley는 순환 후 새 인증서를 사용하지 않습니다.
- OSSM-99 레이블 없이 직접 Pod에서 생성된 워크로드는 Kiali를 충돌할 수 있습니다.
- OSSM-93 IstioConfigList는 두 개 이상의 이름으로 필터링할 수 없습니다.
- OSSM-92 VS/DR YAML 편집 페이지에서 저장되지 않은 변경 사항을 취소해도 변경 사항이 취소되지 않습니다.
- OSSM-90 추적은 서비스 세부 정보 페이지에서 사용할 수 없습니다.
- MAISTRA-1649 헤드리스 서비스가 다른 네임스페이스에서 충돌합니다. 다른 네임스페이스에 헤드리스 서비스를 배포할 때 끝점 구성이 병합되고 잘못된 Envoy 구성이 사이드카로 푸시됩니다.
- MAISTRA-1541 컨트롤러가 소유자 참조에 설정되지 않은 경우 kubernetesenv에서 패닉이 발생합니다. pod에 컨트롤러를 지정하지 않는 ownerReference가 있는 경우 kubernetesenv cache.go 코드 내에서 패닉이 발생할 수 있습니다.

- MAISTRA-1352 컨트롤 플레인 설치의 Cert-manager CRD(Custom Resource Definitions)가 이 릴리스와 향후 릴리스에서 제외됩니다. Red Hat OpenShift Service Mesh를 이미 설치한 경우 cert-manager가 사용되지 않는 경우 CRD를 수동으로 제거해야 합니다.

- MAISTRA-1001 HTTP/2 연결을 종료하면 istio-proxy에서 세그먼트 오류가 발생할 수 있습니다.

- MAISTRA-932 Jaeger Operator와 OpenShift Elasticsearch Operator 간의 종속성 관계를 추가하기 위해 requires 메타데이터를 추가했습니다. Jaeger Operator가 설치되면 사용할 수 없는 경우 OpenShift Elasticsearch Operator가 자동으로 배포됩니다.

- MAISTRA-862 Galley는 여러 네임스페이스를 삭제하고 다시 만든 뒤, 감시를 중단하고 다른 구성 요소에 대한 구성 제공을 중단했습니다.

- MAISTRA-833 Pilot은 여러 네임스페이스를 삭제하고 다시 만든 뒤, 구성 전달을 중단했습니다.

- MAISTRA-684 istio-operator의 기본 Jaeger 버전은 1.12.0이며, 이는 Red Hat OpenShift Service Mesh 0.12.TechPreview에 제공된 Jaeger 버전 1.13.1과 일치하지 않습니다.

- MAISTRA-622 Maistra 0.12.0/TP12에서는 허용 모드가 작동하지 않습니다. 사용자에게는 일반 텍스트 모드 또는 상호 TLS 모드를 사용하는 옵션이 있지만 허용되지는 않습니다.

- MAISTRA-572 Jaeger는 Kiali와 함께 사용할 수 없습니다. 이 릴리스에서 Jaeger는 OAuth 프록시를 사용하도록 구성되어 있지만, 브라우저를 통해서만 작동하도록 구성되어 서비스 액세스를 허용하지 않습니다. Kiali는 Jaeger 끝점과 올바르게 통신할 수 없으며 Jaeger가 비활성화된 것으로 간주합니다. 또한 TRACING-591을 참조하십시오.

- MAISTRA-357 AWS의 OpenShift 4 Beta에서는 기본적으로 포트 80 이외의 포트에서 수신 게이트웨이를 통해 TCP 또는 HTTPS 서비스에 액세스할 수 없습니다. AWS 로드 밸런서에는 서비스 끝점의 포트 80이 활성화되어 있는지 확인하는 상태 점검 기능이 있습니다. 포트 80에서 서비스를 실행하지 않으면 로드 밸런서 상태 점검에 실패합니다.

- MAISTRA-348 AWS의 OpenShift 4 Beta는 80 또는 443 이외의 포트에서 수신 게이트웨이 트래픽을 지원하지 않습니다. 80 또는 443 이외의 포트 번호로 TCP 트래픽을 처리하도록 수신 게이트웨이를 구성하려면, 이 문제를 해결하기 위해 OpenShift 라우터 대신 AWS 로드 밸런서에서 제공하는 서비스 호스트 이름을 사용해야 합니다.

- MAISTRA-193 citadel에 대해 상태 확인이 활성화되면 예기치 않은 콘솔 정보 메시지가 표시됩니다.

- 버그 1821432 OpenShift Container Platform 제어 리소스 세부 정보 페이지의 토글 제어가 CR을 올바르게 업데이트하지 않습니다. OpenShift Container Platform 웹 콘솔의 SMCP(Service Mesh Control Plane) 개요 페이지의 UI 토글 제어가 리소스에서 잘못된 필드를 업데이트하는 경우가 있습니다. ServiceMeshControlPlane 리소스를 업데이트하려면 토글 제어를 클릭하는 대신 YAML 콘텐츠를 직접 편집하거나 명령줄에서 리소스를 업데이트합니다.

2.1.8.2. Kiali의 수정된 문제

- KIALI-3239 Kiali Operator Pod가 "Evicted" 상태로 실패한 경우 Kiali Operator가 배포되지 않습니다. 해결방법은 Evicted pod를 삭제하고 Kiali Operator를 재배포하는 것입니다.

- KIALI-3118 ServiceMeshMemberRoll을 변경(예: 프로젝트 추가 또는 삭제)한 후 Kiali pod가 다시 시작되며, Kiali pod가 다시 시작되는 동안 그래프 페이지에 오류가 표시됩니다.

- KIALI-3096 서비스 메시에서 런타임 지표가 실패합니다. 서비스 메시와 Prometheus 사이에 OAuth 필터가 있으며 액세스 권한이 부여되기 전에 전달자 토큰을 Prometheus에 전달해야 합니다. Kiali는 Prometheus 서버와 통신할 때 이 토큰을 사용하도록 업데이트되었지만 현재 애플리케이션 지표에 403 오류가 발생하고 있습니다.

- KIALI-3070 이 버그는 기본 대시보드가 아닌 사용자 정의 대시보드에만 영향을 미칩니다. 지표 설정에서 레이블을 선택하고 페이지를 새로 고침하면 선택 사항이 메뉴에는 유지하지만 차트에 표시되지 않습니다.

- KIALI-2686 컨트롤 플레인에 네임스페이스가 많은 경우 성능 문제가 발생할 수 있습니다.

2.2. 서비스 메시 이해

주의

더 이상 지원되지 않는 **Red Hat OpenShift Service Mesh** 릴리스에 대한 문서를 보고 있습니다.

서비스 메시 버전 **1.0** 및 **1.1** 컨트롤 플레인은 더 이상 지원되지 않습니다. 서비스 메시 컨트롤 플레인 업그레이드에 대한 자세한 내용은 서비스 메시 업그레이드를 참조하십시오.

특정 **Red Hat OpenShift Service Mesh** 릴리스의 지원 상태에 대한 자세한 내용은 제품 라이프사이클 페이지를 참조하십시오.

Red Hat OpenShift Service Mesh는 서비스 메시에서 네트워크로 연결된 마이크로 서비스에 대해 동작 정보 및 운영 제어용 플랫폼을 제공합니다. **Red Hat OpenShift Service Mesh**를 사용하면 **OpenShift Container Platform** 환경에서 마이크로 서비스를 연결, 보호 및 모니터링할 수 있습니다.

2.2.1. 서비스 메시 이해

*서비스 메시*는 분산 마이크로 서비스 아키텍처에서 애플리케이션을 구성하는 마이크로 서비스 네트워크와 이러한 마이크로 서비스 간의 상호 작용입니다. 서비스 메시의 크기와 복잡성이 증가함에 따라 이를 이해하고 관리하는 것이 어려워질 수 있습니다.

오픈 소스 **Istio** 프로젝트를 기반으로 하는 **Red Hat OpenShift Service Mesh**는 서비스 코드를 변경할 필요 없이 기존 분산 애플리케이션에 투명 계층을 추가합니다. 마이크로 서비스 간의 모든 네트워크 통신을 차단하는 메시의 관련 서비스에 특수 사이드카 프록시를 배포하여 **Red Hat OpenShift Service Mesh** 지원을 서비스에 추가합니다. 서비스 메시 컨트롤 플레인 기능을 사용하여 서비스 메시를 구성하고 관리합니다.

Red Hat OpenShift Service Mesh를 사용하면, 다음과 같은 기능을 제공하는 배포된 서비스 네트워크를 쉽게 생성할 수 있습니다.

- 검색

- 로드 밸런싱
- 서비스 간 인증
- 장애 복구
- 지표
- 모니터링

Red Hat OpenShift Service Mesh는 다음과 같은 보다 복잡한 운영 기능을 제공합니다:

- A/B 테스트
- Canary 릴리스
- 액세스 제어
- 엔드 투 엔드 인증

2.2.2. Red Hat OpenShift Service Mesh 아키텍처

Red Hat OpenShift Service Mesh는 논리적으로 데이터 플레인과 컨트롤 플레인으로 분할됩니다.

데이터 플레인은 사이드카로 배포된 지능적 프록시 세트입니다. 이러한 프록시는 서비스 메시에서 마이크로 서비스 간의 모든 인바운드 및 아웃바운드 네트워크 통신을 가로채기하고 제어합니다. 또한 사이드카 프록시는 범용 정책 및 Telemetry 허브인 Mixer와 통신합니다.

- Envoy proxy는 서비스 메시의 모든 서비스에 대한 인바운드 및 아웃바운드 트래픽을 가로채기합니다. Envoy는 동일한 pod에서 관련 서비스에 사이드카로 배포됩니다.

컨트롤 플레인은 트래픽을 라우팅하기 위해 프록시를 관리 및 구성하며, 정책을 적용하고 **Telemetry**를 수집하도록 **Mixers**를 구성합니다.

- **Mixer**는 액세스 제어 및 사용 정책(예: 권한 부여, 속도 제한, 할당량, 인증 및 요청 추적)을 시행하고 **Envoy** 프록시 및 기타 서비스에서 **Telemetry** 데이터를 수집합니다.
- **Pilot**은 런타임 시 프록시를 구성합니다. **Pilot**은 **Envoy** 사이드카에 대한 서비스 검색, 지능형 라우팅을 위한 트래픽 관리 기능(예: **A/B** 테스트 또는 카나리아 배포) 및 복구 기능(시간 초과, 재시도 및 회로 차단기)을 제공합니다.
- **Citadel**은 인증서를 발행 및 교체합니다. **Citadel**은 내장 **ID** 및 자격 증명 관리를 통해 강력한 서비스 간 인증과 최종 사용자 인증을 제공합니다. **Citadel**을 사용하여 서비스 메시에서 암호화되지 않은 트래픽을 업그레이드할 수 있습니다. **Operator**는 **Citadel**을 사용하여 네트워크 제어가 아닌 서비스 **ID**를 기반으로 정책을 시행할 수 있습니다.
- **Galley**는 서비스 메시 구성을 수집한 다음, 구성의 유효성을 검증, 처리 및 배포합니다. **Galley**는 다른 서비스 메시 구성 요소가 **OpenShift Container Platform**에서 사용자 구성 세부 정보를 얻지 못하도록 보호합니다.

또한 **Red Hat OpenShift Service Mesh**는 **istio-operator**를 사용하여 컨트롤 플레인 설치를 관리합니다. ***Operator***는 **OpenShift Container Platform** 클러스터에서 공통 활동을 구현하고 자동화할 수 있는 소프트웨어입니다. 컨트롤러 역할을 하여 클러스터에서 원하는 오브젝트 상태를 설정하거나 변경할 수 있습니다.

2.2.3. Kiali 이해

Kiali는 서비스 메시의 마이크로 서비스와 해당 연결 방법을 표시하여 서비스 메시를 시각화할 수 있습니다.

2.2.3.1. Kiali 개요

Kiali는 **OpenShift Container Platform**에서 실행 중인 서비스 메시에 대한 관찰 기능을 제공합니다. **Kiali**는 **Istio** 서비스 메시를 정의하고 검증하며 관찰하는 데 도움이 됩니다. 이를 통해 토폴로지를 유추하여 서비스 메시의 구조를 이해하고 서비스 메시의 상태에 대한 정보를 제공할 수 있습니다.

Kiali는 회로 차단기, 요청 속도, 대기 시간, 트래픽 흐름 그래프와 같은 기능에 대한 가시성을 제공하는 네임스페이스의 대화형 그래프 보기를 실시간으로 제공합니다. **Kiali**는 애플리케이션에서 서비스 및 워크로드에 이르기까지 다양한 수준의 구성 요소에 대한 통찰력을 제공하며, 선택한 그래프 노드 또는 에지에서 상황별 정보에 대한 상호 작용과 차트를 표시할 수 있습니다. **Kiali**는 게이트웨이, 대상 규칙, 가상

서비스, 메시 정책 등과 같은 Istio 구성의 유효성을 확인하는 기능도 제공합니다. Kiali는 자세한 지표를 제공하며 고급 쿼리에 기본 Grafana 통합이 가능합니다. Jaeger를 Kiali 콘솔에 통합하면 분산 추적이 제공됩니다.

Kiali는 기본적으로 Red Hat OpenShift Service Mesh의 일부로 설치됩니다.

2.2.3.2. Kiali 아키텍처

Kiali는 오픈 소스 Kiali 프로젝트를 기반으로 합니다. Kiali는 Kiali 애플리케이션과 Kiali 콘솔이라는 두 가지 구성 요소로 구성됩니다.

- Kiali 애플리케이션(백엔드) - 이 구성 요소는 컨테이너 애플리케이션 플랫폼에서 실행되고 서비스 메시 구성 요소와 통신하며, 데이터를 검색 및 처리하고, 이 데이터를 콘솔에 노출합니다. Kiali 애플리케이션에는 스토리지가 필요하지 않습니다. 클러스터에 애플리케이션을 배포할 때 구성은 ConfigMaps 및 시크릿에 설정됩니다.

- Kiali 콘솔(프론트엔드) - Kiali 콘솔은 웹 애플리케이션입니다. Kiali 애플리케이션은 Kiali 콘솔을 제공하며 이를 사용자에게 표시하기 위해 데이터의 백엔드를 쿼리합니다.

또한 Kiali는 컨테이너 애플리케이션 플랫폼과 Istio에서 제공하는 외부 서비스 및 구성 요소에 따라 달라집니다.

- Red Hat Service Mesh(Istio) - Istio는 Kiali 요구 사항입니다. Istio는 서비스 메시를 제공하고 제어하는 구성 요소입니다. Kiali와 Istio를 별도로 설치할 수 있지만 Kiali는 Istio에 따라 달라지며 Istio가 존재하지 않는 경우 작동하지 않습니다. Kiali는 Prometheus 및 클러스터 API를 통해 노출되는 Istio 데이터와 구성을 검색해야 합니다.

- Prometheus - 전용 Prometheus 인스턴스는 Red Hat OpenShift Service Mesh 설치의 일부로 포함되어 있습니다. Istio Telemetry가 활성화되면 지표 데이터가 Prometheus에 저장됩니다. Kiali는 이 Prometheus 데이터를 사용하여 메시 토폴로지 확인, 지표 표시, 상태 계산, 가능한 문제 표시 등의 작업을 수행합니다. Kiali는 Prometheus와 직접 통신하고 Istio Telemetry에서 사용하는 데이터 스키마를 가정합니다. Prometheus는 Istio 종속성 및 Kiali에 대한 하드 종속성이며, 대부분의 Kiali 기능은 Prometheus없이 작동하지 않습니다.

- 클러스터 API - Kiali는 서비스 메시 구성을 가져와 해결하기 위해 OpenShift Container Platform(클러스터 API)의 API를 사용합니다. Kiali는 클러스터 API를 쿼리하여 네임스페이스, 서비스, 배포, pod 및 기타 엔터티에 대한 정의를 검색합니다. 또한 Kiali는 다른 클러스터 엔티티 간의 관계를 해결하기 위해 쿼리를 만듭니다. 클러스터 API는 가상 서비스, 대상 규칙, 경로 규칙, 게이트웨이, 할당량 등과 같은 Istio 구성을 검색하도록 쿼리합니다.

- **Jaeger - Jaeger**는 선택 사항이지만 **Red Hat OpenShift Service Mesh**의 일부로 설치됩니다. 기본 **Red Hat OpenShift Service Mesh** 설치의 일부로 분산 추적 플랫폼(**Jaeger**)을 설치할 때 **Kiali** 콘솔에는 분산 추적 데이터를 표시하는 탭이 포함됩니다. **Istio**의 분산 추적 기능을 비활성화하면 추적 데이터를 사용할 수 없습니다. 또한 사용자가 추적 데이터를 보기 위해 **Service Mesh Control Plane**이 설치된 네임스페이스에 대한 액세스 권한이 있어야 합니다.

- **Grafana - Grafana**는 선택 사항이지만 **Red Hat OpenShift Service Mesh**의 일부로 설치됩니다. 사용 가능한 경우, **Kiali**의 지표 페이지에 사용자를 **Grafana**의 동일한 지표로 안내하는 링크가 표시됩니다. 사용자가 **Grafana** 대시보드에 대한 링크를 보고 **Grafana** 데이터를 보려면 **Service Mesh Control Plane**이 설치된 네임스페이스에 대한 액세스 권한이 있어야 합니다.

2.2.3.3. Kiali 기능

Kiali 콘솔은 **Red Hat Service Mesh**와 통합되어 다음 기능을 제공합니다.

- 상태 **-** 애플리케이션, 서비스 또는 워크로드에 대한 문제를 빠르게 식별합니다.

- 토폴로지 **-** 애플리케이션, 서비스 또는 워크로드가 **Kiali** 그래프를 통해 통신하는 방식을 시각화합니다.

- 지표 **-** 사전 정의된 지표 대시 보드를 통해 **Go, Node.js. Quarkus, Spring Boot, Thorntail, Vert.x**에 대한 서비스 메시 및 애플리케이션 성능을 차트로 작성할 수 있습니다. 또한 사용자 정의 대시보드를 생성할 수도 있습니다.

- 추적 **- Jaeger**와의 통합을 통해 애플리케이션을 구성하는 다양한 마이크로 서비스를 통해 요청 경로를 따를 수 있습니다.

- 검증 **-** 가장 일반적인 **Istio** 오브젝트에 대한 고급 검증(대상 규칙, 서비스 항목, 가상 서비스 등)을 수행합니다.

- 구성 **-** 마법사를 사용하거나 **Kiali** 콘솔의 **YAML** 편집기에서 직접 **Istio** 라우팅 구성을 생성, 업데이트 및 삭제할 수 있는 옵션입니다.

2.2.4. Jaeger 이해

사용자가 애플리케이션에서 작업을 수행할 때마다 응답을 생성하기 위해 참여하도록 다양한 서비스를 필요로 할 수 있는 아키텍처에 의해 요청이 실행됩니다. 이 요청의 경로는 분산 트랜잭션입니다. **Jaeger**를

사용하면 애플리케이션을 구성하는 다양한 마이크로 서비스를 통해 요청의 경로를 따르는 분산 추적을 수행할 수 있습니다.

분산 추적은 분산 트랜잭션에 있는 전체 이벤트 체인을 이해하기 위해 일반적으로 다양한 프로세스 또는 호스트에서 실행되는 다양한 작업 단위에 대한 정보를 결합하는 데 사용되는 기술입니다. 분산 추적을 통해 개발자는 대규모 서비스 지향 아키텍처에서 호출 흐름을 시각화할 수 있습니다. 직렬화, 병렬 처리 및 대기 시간 소스를 이해하는 데 유용할 수 있습니다.

Jaeger는 마이크로 서비스의 전체 스택에서 개별 요청 실행을 기록하고 이를 추적으로 제공합니다. 추적은 시스템을 통한 데이터/실행 경로입니다. 엔드 투 엔드 추적은 하나 이상의 기간으로 구성됩니다.

기간은 작업 이름, 작업의 시작 시간 및 기간이 있는 **Jaeger**의 논리적 작업 단위를 나타냅니다. 기간은 중첩되어 인과 관계를 모델링하도록 주문될 수 있습니다.

2.2.4.1. 분산 추적 개요

서비스 소유자는 분산 추적을 사용하여 서비스 아키텍처에 대한 통찰력을 수집하기 위해 서비스를 계측할 수 있습니다. **Red Hat OpenShift** 분산 추적 플랫폼을 사용하여 최신 클라우드 네이티브, 마이크로 서비스 기반 애플리케이션의 구성 요소 간의 상호 작용을 모니터링, 네트워크 프로파일링 및 문제 해결에 사용할 수 있습니다.

분산 추적 플랫폼을 사용하면 다음 기능을 수행할 수 있습니다.

- 분산 트랜잭션 모니터링
- 성능 및 대기 시간 최적화
- 근본 원인 분석 수행

분산 추적 플랫폼은 다음 세 가지 구성 요소로 구성됩니다.

- **Red Hat OpenShift distributed tracing platform(Jaeger)** 은 오픈 소스 **Jaeger** 프로젝트를 기반으로 합니다.
-

Red Hat OpenShift distributed tracing platform(Tempo) 은 오픈 소스 Grafana Tempo 프로젝트를 기반으로 합니다.

- Red Hat OpenShift distributed tracing 데이터 수집 은 오픈 소스 OpenTelemetry 프로젝트를 기반으로 합니다.

2.2.4.2. 분산 추적 아키텍처

분산 추적 플랫폼(Jaeger)은 오픈 소스 Jaeger 프로젝트를 기반으로 합니다. 분산 추적 플랫폼(Jaeger)은 함께 작동하여 추적 데이터를 수집, 저장 및 표시하는 여러 구성 요소로 구성됩니다.

- Jaeger Client(Tracer, Reporter, 조정된 애플리케이션, 클라이언트 라이브러리)- Jaeger 클라이언트는 OpenTracing API의 언어 특정 구현입니다. 수동으로 또는 이미 OpenTracing과 통합된 Camel(Fuse), Spring Boot(RHOAR), MicroProfile(RHOAR/T©tail), Wildfly(EAP) 등의 다양한 기존 오픈 소스 프레임워크를 사용하여 분산 추적에 대해 애플리케이션을 조정하는 데 사용할 수 있습니다.

- Jaeger 에이전트(Server Queue, Processor Workers) - Jaeger 에이전트는 UDP(User Datagram Protocol)를 통해 전송되는 기간을 수신 대기하는 네트워크 데몬으로, 수집기에 배치 및 전송합니다. 에이전트는 조정된 애플리케이션과 동일한 호스트에 배치되어야 합니다. 일반적으로 Kubernetes와 같은 컨테이너 환경에서 사이드카를 보유하여 수행됩니다.

- Jaeger 수집기(Queue, Workers) - 에이전트와 유사하게 수집기는 기간을 수신하여 처리를 위한 내부 큐에 배치할 수 있습니다. 그러면 수집기는 기간이 스토리지로 이동할 때까지 대기하지 않고 클라이언트/에이전트로 즉시 돌아갈 수 있습니다.

- 스토리지(데이터 저장소) - 수집기에는 영구 스토리지 백엔드가 필요합니다. Jaeger에는 기간 스토리지를 위한 플러그인 메커니즘이 있습니다. 이 릴리스에서 지원되는 유일한 스토리지는 Elasticsearch입니다.

- 쿼리(쿼리 서비스) - 쿼리는 스토리지에서 추적을 검색하는 서비스입니다.

- Ingester(Ingester 서비스) - Jaeger는 수집기와 실제 백업 스토리(Elasticsearch) 간의 버퍼로 Apache Kafka를 사용할 수 있습니다. Ingester는 Kafka에서 데이터를 읽고 다른 스토리지 백엔드(Elasticsearch)에 쓰는 서비스입니다.

- Jaeger 콘솔 - Jaeger는 분산 추적 데이터를 시각화할 수 있는 사용자 인터페이스를 제공합니다. 검색 페이지에서 추적을 찾고 개별 추적을 구성하는 기간의 세부 사항을 확인할 수 있습니

다.

2.2.4.3. Red Hat OpenShift distributed tracing 플랫폼 기능

Red Hat OpenShift distributed tracing 플랫폼은 다음과 같은 기능을 제공합니다.

- Kiali와의 통합 - 올바르게 구성된 경우 Kiali 콘솔에서 분산 추적 플랫폼 데이터를 볼 수 있습니다.
- 높은 확장성 - 분산 추적 플랫폼 백엔드는 단일 장애 지점이 없으며 비즈니스 요구 사항에 따라 확장할 수 있도록 설계되었습니다.
- 분산 컨텍스트 전파 - 다른 구성 요소의 데이터를 함께 연결하여 완전한 엔드 투 엔드 추적을 만들 수 있습니다.
- Zipkin과의 역호환성 - Red Hat OpenShift distributed tracing 플랫폼에는 Zipkin의 드롭인 대체 기능으로 사용할 수 있는 API가 있지만 Red Hat은 이 릴리스에서 Zipkin 호환성을 지원하지 않습니다.

2.2.5. 다음 단계

- OpenShift Container Platform 환경에 Red Hat OpenShift Service Mesh를 설치할 준비를 합니다.

2.3. 서비스 메시 및 ISTIO 차이점

주의

더 이상 지원되지 않는 **Red Hat OpenShift Service Mesh** 릴리스에 대한 문서를 보고 있습니다.

서비스 메시 버전 **1.0** 및 **1.1** 컨트롤 플레인은 더 이상 지원되지 않습니다. 서비스 메시 컨트롤 플레인 업그레이드에 대한 자세한 내용은 서비스 메시 업그레이드를 참조하십시오.

특정 **Red Hat OpenShift Service Mesh** 릴리스의 지원 상태에 대한 자세한 내용은 제품 라이프사이클 페이지를 참조하십시오.

Red Hat OpenShift Service Mesh 설치는 여러 가지 면에서 업스트림 **Istio** 커뮤니티 설치와 다릅니다. **Red Hat OpenShift Service Mesh**에 대한 수정은 **OpenShift Container Platform**에 배포할 때 문제를 해결하거나, 추가 기능을 제공하거나, 차이점을 처리하기 위해 필요한 경우가 있습니다.

Red Hat OpenShift Service Mesh의 현재 릴리스는 다음과 같은 점에서 현재 업스트림 **Istio** 커뮤니티 릴리스와 다릅니다.

2.3.1. 다중 테넌트 설치

업스트림 **Istio**는 하나의 테넌트 접근법을 사용하지만 **Red Hat OpenShift Service Mesh**는 클러스터 내에서 여러 개의 독립적인 컨트롤 플레인을 지원합니다. **Red Hat OpenShift Service Mesh**는 다중 테넌트 연산자를 사용하여 컨트롤 플레인 라이프사이클을 관리합니다.

Red Hat OpenShift Service Mesh는 기본적으로 다중 테넌트 컨트롤 플레인을 설치합니다. 서비스 메시에 액세스할 수 있는 프로젝트를 지정하고 다른 컨트롤 플레인 인스턴스에서 서비스 메시를 분리합니다.

2.3.1.1. 멀티 테넌시 대 클러스터 전체 설치

다중 테넌트 설치와 클러스터 전체 설치의 주요 차이점은 **istod**에서 사용하는 권한 범위입니다. 구성 요소는 더 이상 클러스터 범위의 역할 기반 액세스 제어(**RBAC**) 리소스 **ClusterRoleBinding**을 사용하지 않습니다.

ServiceMeshMemberRoll members 목록에 있는 모든 프로젝트는 컨트롤 플레인 배포와 관련된 각 서비스 계정에 대해 **RoleBinding**을 가지며, 각 컨트롤 플레인 배포는 해당하는 멤버 프로젝트만 감시합니다. 각 멤버 프로젝트에는 **maistra.io/member-of** 레이블이 추가됩니다. 여기서 **member-of** 값은 컨트롤 플레인 설치가 포함된 프로젝트입니다.

Red Hat OpenShift Service Mesh는 각 멤버 프로젝트를 구성하여 자체, 컨트롤 플레인 및 기타 멤버 프로젝트 간의 네트워크 액세스를 보장합니다. 정확한 구성은 **OpenShift Container Platform** 소프트웨어 정의 네트워킹(**SDN**)이 구성된 방법에 따라 다릅니다. 자세한 내용은 **OpenShift SDN** 정보를 참조하십시오.

OpenShift Container Platform 클러스터가 **SDN** 플러그인을 사용하도록 구성된 경우:

- **NetworkPolicy: Red Hat OpenShift Service Mesh**는 각 멤버 프로젝트에서 **NetworkPolicy** 리소스를 생성하여 다른 멤버 및 컨트롤 플레인에서 모든 **pod**로 수신할 수 있습니다. **Service Mesh**에서 멤버를 제거하면 이 **NetworkPolicy** 리소스는 프로젝트에서 삭제됩니다.

참고

또한 멤버 프로젝트 전용 수신으로 제한합니다. 멤버 외 프로젝트에서 수신이 필요한 경우 해당 트래픽을 허용하기 위해 **NetworkPolicy**를 생성해야 합니다.

- 다중 테넌트: **Red Hat OpenShift Service Mesh**는 각 멤버 프로젝트의 **NetNamespace**를 컨트롤 플레인 프로젝트의 **NetNamespace**에 결합합니다(**oc adm pod-network join-projects --to control-plane-project member-project**). 서비스 메시에서 멤버를 제거하면 해당 **NetNamespace**가 컨트롤 플레인과 분리됩니다(**oc adm pod-network isolate-projects member-project** 실행과 동일).

- 서브넷: 추가 구성이 수행되지 않습니다.

2.3.1.2. 클러스터 범위 리소스

업스트림 **Istio**에는 의존하는 두 개의 클러스터 범위 리소스가 있습니다. **MeshPolicy** 및 **ClusterRbacConfig** 이는 다중 테넌트 클러스터와 호환되지 않으며 아래에 설명된 대로 교체되었습니다.

- ***ServiceMeshPolicy***는 컨트롤 플레인 전체의 인증 정책 구성을 위해 **MeshPolicy**를 대체합니다. 이는 컨트롤 플레인과 동일한 프로젝트에서 생성되어야 합니다.

- *ServicemeshRbacConfig*는 컨트롤 플레인 전체 역할 기반 액세스 제어 구성을 위해 **ClusterRbacConfig** 를 대체합니다. 이는 컨트롤 플레인과 동일한 프로젝트에서 생성되어야 합니다.

2.3.2. Istio와 Red Hat OpenShift Service Mesh 간의 차이점

Red Hat OpenShift Service Mesh 설치는 여러 가지 면에서 **Istio** 설치와 다릅니다. **Red Hat OpenShift Service Mesh**에 대한 수정은 **OpenShift Container Platform**에 배포할 때 문제를 해결하거나, 추가 기능을 제공하거나, 차이점을 처리하기 위해 필요한 경우가 있습니다.

2.3.2.1. 명령줄 도구

Red Hat OpenShift Service Mesh의 명령줄 도구는 **oc**입니다. **Red Hat OpenShift Service Mesh**는 **istioctl**을 지원하지 않습니다.

2.3.2.2. 자동 삽입

업스트림 **Istio** 커뮤니티 설치는 레이블을 지정한 프로젝트 내의 **pod**에 사이드카를 자동으로 삽입합니다.

Red Hat OpenShift Service Mesh는 사이드카를 **Pod**에 자동으로 삽입하지 않지만 프로젝트에 레이블을 지정하지 않고 주석을 사용하여 삽입해야 합니다. 이 방법은 더 적은 권한이 필요하며, **builder pod**와 같은 다른 **OpenShift** 기능과 충돌하지 않습니다. 자동 삽입을 활성화하려면 자동 사이드카 삽입 섹션에 설명된 대로 **sidecar.istio.io/inject** 주석을 지정합니다.

2.3.2.3. Istio 역할 기반 액세스 제어 기능

역할 기반 액세스 제어(**RBAC**)는 서비스에 대한 액세스를 제어하는 데 사용할 수 있는 메커니즘을 제공합니다. 사용자 이름별로, 또는 속성 집합을 지정하여 제목을 식별하고 그에 따라 액세스 제어를 적용할 수 있습니다.

업스트림 **Istio** 커뮤니티 설치에는 정확한 헤더 일치를 수행하거나, 헤더에서 와일드카드를 일치시키거나, 특정 접두사 또는 접미사가 포함된 헤더를 확인하는 옵션이 포함되어 있습니다.

Red Hat OpenShift Service Mesh는 정규식을 사용하여 요청 헤더를 일치시키는 기능을 확장합니다. 정규식이 있는 **request.regex.headers**의 속성 키를 지정합니다.

요청 헤더와 일치하는 업스트림 **Istio** 커뮤니티 예

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

정규식을 사용하여 요청 헤더를 일치시키는 **Red Hat OpenShift Service Mesh**

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

2.3.2.4. OpenSSL

Red Hat OpenShift Service Mesh는 **BoringSSL**을 **OpenSSL**로 대체합니다. **OpenSSL**은 **SSL(Secure Sockets Layer)** 및 **TLS(Transport Layer Security)** 프로토콜의 오픈 소스 구현이 포함된 소프트웨어 라이브러리입니다. **Red Hat OpenShift Service Mesh** 프록시 바이너리는 기본 **Red Hat Enterprise Linux** 운영 체제에서 **OpenSSL** 라이브러리(**libssl** 및 **libcrypto**)를 동적으로 연결합니다.

2.3.2.5. 구성 요소 수정

- ***maistra-version*** 레이블이 모든 리소스에 추가되었습니다.

- 모든 Ingress 리소스가 OpenShift Route 리소스로 변환되었습니다.

- Grafana, Tracing(Jaeger) 및 Kiali는 기본적으로 활성화되어 OpenShift 경로를 통해 노출됩니다.

- Godebug가 모든 템플릿에서 제거됨

- istio-multi ServiceAccount과 ClusterRoleBinding, istio-reader ClusterRole이 제거되었습니다.

2.3.2.6. Envoy, Secret Discovery Service, 및 인증서

- Red Hat OpenShift Service Mesh는 QUIC 기반 서비스를 지원하지 않습니다.

- Istio의 SDS(Security Discovery Service) 기능을 사용한 TLS 인증서 배포는 현재 Red Hat OpenShift Service Mesh에서 지원되지 않습니다. Istio 구현은 hostPath 마운트를 사용하는 nodeagent 컨테이너에 따라 다릅니다.

2.3.2.7. Istio CNI(Container Network Interface) 플러그인

Red Hat OpenShift Service Mesh에는 CNI 플러그인이 포함되어 있으며, 이를 통해 애플리케이션 Pod 네트워킹을 구성할 수 있는 대체 방법을 제공합니다. CNI 플러그인은 승격된 권한으로 SCC(보안 컨텍스트 제약 조건)에 대한 서비스 계정 및 프로젝트 액세스 권한을 부여할 필요가 없도록 init-container 네트워크 구성을 대체합니다.

2.3.2.8. Istio 게이트웨이 경로

Istio 게이트웨이의 OpenShift 경로는 Red Hat OpenShift Service Mesh에서 자동으로 관리됩니다. Istio 게이트웨이가 서비스 메시 내부에서 생성, 업데이트 또는 삭제될 때마다 OpenShift 경로가 생성, 업데이트 또는 삭제됩니다.

IOR(Istio OpenShift Routing)이라는 Red Hat OpenShift Service Mesh Control Plane 구성 요소는 게이트웨이 경로를 동기화합니다. 자세한 내용은 자동 경로 생성을 참조하십시오.

2.3.2.8.1. catch-all 도메인

catch-all 도메인("*")은 지원되지 않습니다. 게이트웨이 정의에서 이 도메인이 발견되면 **Red Hat OpenShift Service Mesh**는 경로를 *생성하지만* 기본 호스트 이름을 만들기 위해 **OpenShift**에 의존합니다. 즉, 새로 생성된 경로는 **catch-all ("*")** 경로가 *아니며*, 대신 **r<route-name>[-<project>].<suffix>** 형식의 호스트 이름이 있습니다. 기본 호스트 이름이 작동하는 방법과 클러스터 관리자가 사용자 지정할 수 있는 방법에 대한 자세한 내용은 **OpenShift** 문서를 참조하십시오.

2.3.2.8.2. 하위 도메인

하위 도메인(예: "*.domain.com")이 지원됩니다. 그러나 이 기능은 **OpenShift Container Platform**에서 기본적으로 활성화되어 있지 않습니다. 즉, **Red Hat OpenShift Service Mesh**는 하위 도메인이 있는 경로를 *생성하지만* **OpenShift Container Platform**이 이것을 활성화하도록 구성된 경우에만 적용됩니다.

2.3.2.8.3. TLS(Transport layer security)

TLS(Transport Layer Security)가 지원됩니다. 즉, 게이트웨이에 **tls** 섹션이 포함된 경우 **OpenShift** 경로는 **TLS**를 지원하도록 구성됩니다.

추가 리소스

- 자동 경로 생성

2.3.3. Kiali 및 서비스 메시

OpenShift Container Platform의 서비스 메시를 통해 **Kiali**를 설치하는 것은 여러 가지 면에서 커뮤니티 **Kiali** 설치와 다릅니다. 이러한 수정은 **OpenShift Container Platform**에 배포할 때 문제를 해결하거나, 추가 기능을 제공하거나, 차이점을 처리하기 위해 필요한 경우가 있습니다.

- **Kiali**는 기본적으로 활성화되어 있습니다.
- **Ingress**는 기본적으로 활성화되어 있습니다.
- **Kiali ConfigMap**이 업데이트되었습니다.
- **Kiali**의 **ClusterRole** 설정이 업데이트되었습니다.
- 서비스 메시 또는 **Kiali Operator**가 변경 사항을 덮어쓸 수 있으므로 **ConfigMap**을 편집하지 마십시오. **Kiali Operator**가 관리하는 파일에는 **kiali.io/** 레이블 또는 주석이 있습니다. **Operator**

파일을 업데이트하려면 **cluster-admin** 권한이 있는 사용자로 제한해야 합니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **Operator** 파일을 업데이트하려면 **dedicated-admin** 권한이 있는 사용자로 제한해야 합니다.

2.3.4. 분산 추적 및 서비스 메시

OpenShift Container Platform의 **Service Mesh**를 사용하여 분산 추적 플랫폼(**Jaeger**)을 설치하는 것은 여러 가지 면에서 커뮤니티 **Jaeger** 설치와 다릅니다. 이러한 수정은 **OpenShift Container Platform**에 배포할 때 문제를 해결하거나, 추가 기능을 제공하거나, 차이점을 처리하기 위해 필요한 경우가 있습니다.

- 분산 추적은 기본적으로 서비스 메시에 대해 활성화되어 있습니다.
- **Ingress**는 기본적으로 서비스 메시에 대해 활성화되어 있습니다.
- **Zipkin** 포트 이름의 이름이 **jaeger-collector-zipkin(http)**으로 변경되었습니다.
- **Jaeger**는 **production** 또는 **streaming** 배포 옵션을 선택할 때 기본적으로 스토리지에 **Elasticsearch**를 사용합니다.
- **Istio** 커뮤니티 버전은 일반적인 **"tracing"** 경로를 제공합니다. **Red Hat OpenShift Service Mesh**는 **Red Hat OpenShift distributed tracing Platform (Jaeger) Operator**가 설치하고 이미 **OAuth**에 의해 보호되는 **"jaeger"** 경로를 사용합니다.
- **Red Hat OpenShift Service Mesh**는 **Envoy** 프록시에 사이드카를 사용하며 **Jaeger** 또한 **Jaeger** 에이전트에 사이드카를 사용합니다. 이 두 가지 사이드카는 별도로 구성되어 있으며 서로 혼동해서는 안 됩니다. 프록시 사이드카는 **Pod**의 수신 및 송신 트래픽과 관련된 기간을 생성합니다. 에이전트 사이드카는 응용 프로그램에서 발송되는 기간을 수신하여 **Jaeger** 수집기로 보냅니다.

2.4. SERVICE MESH 설치 준비

주의

더 이상 지원되지 않는 **Red Hat OpenShift Service Mesh** 릴리스에 대한 문서를 보고 있습니다.

서비스 메시 버전 **1.0** 및 **1.1** 컨트롤 플레인은 더 이상 지원되지 않습니다. 서비스 메시 컨트롤 플레인 업그레이드에 대한 자세한 내용은 서비스 메시 업그레이드를 참조하십시오.

특정 **Red Hat OpenShift Service Mesh** 릴리스의 지원 상태에 대한 자세한 내용은 제품 라이프사이클 페이지를 참조하십시오.

Red Hat OpenShift Service Mesh를 설치하려면 설치 활동을 검토하고 사전 요구 사항을 충족해야 합니다.

2.4.1. 사전 요구 사항

- **Red Hat** 계정에 유효한 **OpenShift Container Platform** 서브스크립션이 있어야 합니다. 서브스크립션이 없는 경우 영업 담당자에게 자세한 내용을 문의하십시오.
- OpenShift Container Platform 4.11 개요 를 검토합니다.
- **OpenShift Container Platform 4.11**을 설치합니다.
  - AWS에 OpenShift Container Platform 4.11 설치
  - 사용자 프로비저닝 AWS에 OpenShift Container Platform 4.11 설치
  - 베어 메탈에 OpenShift Container Platform 4.11 설치

- vSphere에 OpenShift Container Platform 4.11 설치

참고

제한된 네트워크에 Red Hat OpenShift Service Mesh를 설치하는 경우 선택한 OpenShift Container Platform 인프라에 대한 지침을 따르십시오.

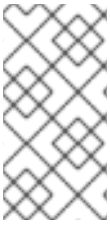
- OpenShift Container Platform 버전과 일치하는 OpenShift Container Platform 명령줄 유틸리티(oc 클라이언트 도구) 버전을 설치하고 해당 경로에 추가합니다.

  - OpenShift Container Platform 4.11을 사용하는 경우 OpenShift CLI 정보를 참조하십시오.

2.4.2. Red Hat OpenShift Service Mesh 지원 구성

다음은 Red Hat OpenShift Service Mesh에 지원되는 구성입니다.

- OpenShift Container Platform 버전 4.6 이상



참고

OpenShift Online 및 Red Hat OpenShift Dedicated는 Red Hat OpenShift Service Mesh에서 지원되지 않습니다.

- 배포가 통합되지 않은 단일 OpenShift Container Platform 클러스터에 포함되어야 합니다.

- 이번 Red Hat OpenShift Service Mesh 릴리스는 OpenShift Container Platform x86_64에서만 사용 가능합니다.

- 이 릴리스에서는 모든 Service Mesh 구성 요소가 작동하는 OpenShift Container Platform 클러스터에 포함된 구성만 지원합니다. 클러스터 외부에 있거나 멀티 클러스터 시나리오에 있는 마이크로 서비스 관리는 지원하지 않습니다.

- 이 릴리스는 가상 머신과 같은 외부 서비스를 통합하지 않는 구성만 지원합니다.

Red Hat OpenShift Service Mesh 라이프사이클 및 지원되는 구성에 대한 자세한 내용은 지원 정책을 참조하십시오.

2.4.2.1. Red Hat OpenShift Service Mesh에서 Kiali에 지원되는 구성

- **Kiali Observation Console**은 **Chrome, Edge, Firefox** 또는 **Safari** 브라우저의 두 가지 최신 버전에서만 지원됩니다.

2.4.2.2. 지원되는 Mixer 어댑터

- 이 릴리스에서는 다음 **Mixer** 어댑터만 지원합니다.
  - **3scale Istio** 어댑터

2.4.3. Operator 개요

Red Hat OpenShift Service Mesh에는 다음과 같은 네 가지 **Operator**가 필요합니다.

- **OpenShift Elasticsearch -** (선택 사항)은 분산 추적 플랫폼(Jaeger)을 사용하여 추적 및 로깅을 위한 데이터베이스 스토리지를 제공합니다. 오픈 소스 **Elasticsearch** 프로젝트를 기반으로 합니다.

- **Red Hat OpenShift distributed tracing platform(Jaeger) -** 분산 추적을 제공하여 복잡한 분산 시스템의 트랜잭션을 모니터링하고 문제를 해결합니다. 오픈 소스 **Jaeger** 프로젝트를 기반으로 합니다.

- **Red Hat**에서 제공하는 **Kiali Operator -** 서비스 메시에 대한 관찰 기능을 제공합니다. 단일 콘솔에서 구성을 보고, 트래픽을 모니터링하고, 추적을 분석할 수 있습니다. 오픈 소스 **Kiali** 프로젝트를 기반으로 합니다.

- **Red Hat OpenShift Service Mesh -** 애플리케이션을 구성하는 마이크로 서비스를 연결, 보안, 제어 및 관찰할 수 있습니다. **Service Mesh Operator**는 **Service Mesh** 구성 요소의 배포, 업데이트 및 삭제를 관리하는 **ServiceMeshControlPlane** 리소스를 정의하고 모니터링합니다. 오픈소스 **Istio** 프로젝트를 기반으로 합니다.

주의

프로덕션 환경에서 Elasticsearch 의 기본 Jaeger 매개변수를 구성하는 방법에 대한 자세한 내용은 로그 저장소 구성을 참조하십시오.

2.4.4. 다음 단계

- OpenShift Container Platform 환경에 Red Hat OpenShift Service Mesh를 설치합니다.

2.5. SERVICE MESH 설치

주의

더 이상 지원되지 않는 Red Hat OpenShift Service Mesh 릴리스에 대한 문서를 보고 있습니다.

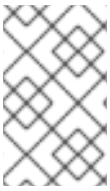
서비스 메시 버전 1.0 및 1.1 컨트롤 플레인은 더 이상 지원되지 않습니다. 서비스 메시 컨트롤 플레인 업그레이드에 대한 자세한 내용은 서비스 메시 업그레이드를 참조하십시오.

특정 Red Hat OpenShift Service Mesh 릴리스의 지원 상태에 대한 자세한 내용은 제품 라이프사이클 페이지를 참조하십시오.

Service Mesh를 설치하려면 OpenShift Elasticsearch, Jaeger, Kiali, Service Mesh Operator를 설치하고, ServiceMeshControlPlane 리소스를 생성 및 관리하여 컨트롤 플레인을 배포하며, Service MeshMemberRoll 리소스를 생성하여 Service Mesh와 연결된 네임스페이스를 지정해야 합니다.

참고

Mixer의 정책 적용은 기본적으로 비활성화되어 있습니다. 정책 작업을 실행하려면 활성화해야 합니다. Mixer 정책 시행 활성화에 대한 지침은 Mixer 정책 시행 업데이트를 참조하십시오.



참고

멀티 테넌트 컨트롤 플레인 설치는 기본 구성입니다.

참고

Service Mesh 문서는 istio-system을 예제 프로젝트로 사용하지만, 모든 프로젝트에 서비스 메시를 배포할 수 있습니다.

2.5.1. 사전 요구 사항

- Red Hat OpenShift Service Mesh 설치 준비 프로세스를 따르십시오.
- cluster-admin 역할이 있는 계정.

서비스 메시 설치 프로세스는 OperatorHub를 사용하여 openshift-operators 프로젝트 내에 ServiceMeshControlPlane 사용자 정의 리소스 정의를 설치합니다. Red Hat OpenShift Service Mesh는 컨트롤 플레인의 배포, 업데이트 및 삭제와 관련된 ServiceMeshControlPlane을 정의하고 모니터링합니다.

Red Hat OpenShift Service Mesh 1.1.18.2부터 OpenShift Elasticsearch Operator, Jaeger Operator 및 Kiali Operator를 설치해야 Red Hat OpenShift Service Mesh Operator가 컨트롤 플레인을 설치할 수 있습니다.

2.5.2. OpenShift Elasticsearch Operator 설치

기본 Red Hat OpenShift distributed tracing Platform(Jaeger) 배포에서는 Red Hat OpenShift distributed tracing 플랫폼을 평가하거나 시연을 제공하거나 테스트 환경에서 Red Hat OpenShift distributed tracing Platform (Jaeger)을 사용하도록 설계되었기 때문에 메모리 내 스토리지를 사용합니다. 프로덕션에서 Red Hat OpenShift distributed tracing Platform(Jaeger)을 사용하려면 영구 스토리지 옵션을 설치하고 구성해야 합니다(이 경우 Elasticsearch).

사전 요구 사항

- OpenShift Container Platform 웹 콘솔에 액세스할 수 있습니다.
- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.

주의

Operator의 커뮤니티 버전은 설치하지 마십시오. 커뮤니티 Operator는 지원되지 않습니다.

참고

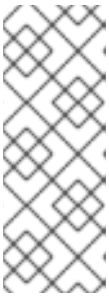
이미 OpenShift Elasticsearch Operator를 OpenShift 로깅의 일부로 설치한 경우 OpenShift Elasticsearch Operator를 다시 설치할 필요가 없습니다. Red Hat OpenShift distributed tracing Platform(Jaeger) Operator는 설치된 OpenShift Elasticsearch Operator를 사용하여 Elasticsearch 인스턴스를 생성합니다.

프로세스

1. cluster-admin 역할의 사용자로 OpenShift Container Platform 웹 콘솔에 로그인합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.
2. Operators → OperatorHub로 이동합니다.
3. Elasticsearch를 필터 상자에 입력하여 OpenShift Elasticsearch Operator를 찾습니다.
4. Red Hat에서 제공하는 OpenShift Elasticsearch Operator를 클릭하여 Operator에 대한 정보를 표시합니다.
5. 설치를 클릭합니다.

6. **Operator** 설치 페이지에서 **stable** 업데이트 채널을 선택합니다. 이렇게 하면 새 버전이 릴리스되면 **Operator**가 자동으로 업데이트됩니다.

7. 클러스터의 기본 모든 네임스페이스(기본값)를 수락합니다. 이렇게 하면 기본 **openshift-operators-redhat** 프로젝트에 **Operator**가 설치되고 클러스터의 모든 프로젝트에서 **Operator**를 사용할 수 있습니다.



참고

Elasticsearch 설치에는 **OpenShift Elasticsearch Operator**의 **openshift-operators-redhat** 네임스페이스가 필요합니다. 다른 **Red Hat OpenShift distributed tracing Platform Operator**는 **openshift-operators** 네임스페이스에 설치됩니다.

8. 기본 자동 승인 전략을 수락합니다. 기본적으로 이 **Operator**의 새 버전이 사용 가능하면 **OLM(Operator Lifecycle Manager)**은 개입 없이 **Operator**의 실행 중인 인스턴스를 자동으로 업그레이드합니다. 수동 업데이트를 선택하면 최신 버전의 **Operator**가 사용 가능할 때 **OLM**에서 업데이트 요청을 생성합니다. 클러스터 관리자는 **Operator**를 새 버전으로 업데이트하려면 **OLM** 업데이트 요청을 수동으로 승인해야 합니다.

참고

수동 승인 전략을 사용하려면 적절한 인증 정보가 있는 사용자가 **Operator** 설치 및 서브스크립션 프로세스를 승인해야 합니다.

9. 설치를 클릭합니다.

10. 설치된 **Operator** 페이지에서 **openshift-operators-redhat** 프로젝트를 선택합니다. 계속하기 전에 **OpenShift Elasticsearch Operator**에 **"InstallSucceeded"** 상태가 표시될 때까지 기다립니다.

2.5.3. Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 설치

Red Hat OpenShift distributed tracing Platform (Jaeger)을 설치하려면 **OperatorHub** 를 사용하여 **Red Hat OpenShift distributed tracing Platform (Jaeger) Operator**를 설치합니다.

기본적으로 Operator는 openshift-operators 프로젝트에 설치됩니다.

사전 요구 사항

- OpenShift Container Platform 웹 콘솔에 액세스할 수 있습니다.
- cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.
- 영구 스토리지가 필요한 경우 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator를 설치하기 전에 OpenShift Elasticsearch Operator도 설치해야 합니다.

주의

Operator의 커뮤니티 버전은 설치하지 마십시오. 커뮤니티 Operator는 지원되지 않습니다.

절차

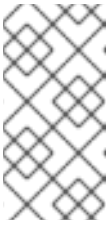
1. cluster-admin 역할의 사용자로 OpenShift Container Platform 웹 콘솔에 로그인합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.
2. Operators → OperatorHub로 이동합니다.
3. 분산 추적 플랫폼을 필터에 입력하여 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator를 찾습니다.
4. Red Hat에서 제공하는 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 를 클릭하여 Operator에 대한 정보를 표시합니다.

5. 설치를 클릭합니다.

6. Operator 설치 페이지에서 stable 업데이트 채널을 선택합니다. 이렇게 하면 새 버전이 릴리스되면 Operator가 자동으로 업데이트됩니다.

7. 클러스터의 기본 모든 네임스페이스(기본값)를 수락합니다. 이렇게 하면 기본 openshift-operators 프로젝트에서 Operator가 설치되고 클러스터의 모든 프로젝트에서 Operator를 사용할 수 있습니다.

- 기본 자동 승인 전략을 수락합니다. 기본적으로 이 Operator의 새 버전이 사용 가능하면 OLM(Operator Lifecycle Manager)은 개입 없이 Operator의 실행 중인 인스턴스를 자동으로 업그레이드합니다. 수동 업데이트를 선택하면 최신 버전의 Operator가 사용 가능할 때 OLM에서 업데이트 요청을 생성합니다. 클러스터 관리자는 Operator를 새 버전으로 업데이트하려면 OLM 업데이트 요청을 수동으로 승인해야 합니다.



참고

수동 승인 전략을 사용하려면 적절한 인증 정보가 있는 사용자가 Operator 설치 및 서브스크립션 프로세스를 승인해야 합니다.

8. 설치를 클릭합니다.

9. Operators → 설치된 Operator로 이동합니다.

10. 설치된 Operator 페이지에서 openshift-operators 프로젝트를 선택합니다. 계속하기 전에 Red Hat OpenShift distributed tracing Platform (Jaeger) Operator에 "Succeeded" 상태가 표시될 때까지 기다립니다.

2.5.4. Kiali Operator 설치

Service Mesh Control Plane을 설치하려면 Red Hat OpenShift Service Mesh Operator에 Kiali Operator를 설치해야 합니다.

주의

Operator의 커뮤니티 버전은 설치하지 마십시오. 커뮤니티 **Operator**는 지원되지 않습니다.

사전 요구 사항

- **OpenShift Container Platform** 웹 콘솔에 액세스합니다.

프로세스

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. **Operators → OperatorHub**로 이동합니다.
3. **Kiali**를 필터 상자에 입력하여 **Kiali Operator**를 찾습니다.
4. **Red Hat**에서 제공하는 **Kiali Operator**를 클릭하여 **Operator**에 대한 정보를 표시합니다.
5. 설치를 클릭합니다.
6. **Operator** 설치 페이지에서 **stable** 업데이트 채널을 선택합니다.
7. 클러스터의 모든 네임스페이스(기본값)를 선택합니다. 이렇게 하면 기본 **openshift-operators** 프로젝트에서 **Operator**가 설치되고 클러스터의 모든 프로젝트에서 **Operator**를 사용할 수 있습니다.
8. 자동 승인 전략을 선택합니다.

참고

수동 승인 전략을 사용하려면 적절한 인증 정보를 가진 사용자가 **Operator** 설치 및 서브스크립션 프로세스를 승인해야 합니다.

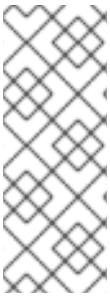
9. 설치를 클릭합니다.

10. 설치된 **Operator** 페이지에 **Kiali Operator**의 설치 진행 상황을 표시합니다.

2.5.5. Operator 설치

Red Hat OpenShift Service Mesh를 설치하려면 다음 **Operator**를 이 순서대로 설치합니다. 각 **Operator**에 대한 절차를 반복합니다.

- **OpenShift Elasticsearch**
- **Red Hat OpenShift distributed tracing platform (Jaeger)**
- **Red Hat**에서 제공하는 **Kiali Operator**
- **Red Hat OpenShift Service Mesh**



참고

이미 **OpenShift Elasticsearch Operator**를 **OpenShift** 로깅의 일부로 설치한 경우 **OpenShift Elasticsearch Operator**를 다시 설치할 필요가 없습니다. **Red Hat OpenShift distributed tracing Platform (Jaeger) Operator**는 설치된 **OpenShift Elasticsearch Operator**를 사용하여 **Elasticsearch** 인스턴스를 생성합니다.

프로세스

1. **cluster-admin** 역할의 사용자로 **OpenShift Container Platform** 웹 콘솔에 로그인합니다. **Red Hat OpenShift Dedicated**를 사용하는 경우 **dedicated-admin** 역할의 계정이 있어야 합니다.

2. OpenShift Container Platform 웹 콘솔에서 Operator → OperatorHub를 클릭합니다.

3. Operator 이름을 필터 상자에 입력하고 Operator의 Red Hat 버전을 선택합니다. Operator의 커뮤니티 버전은 지원되지 않습니다.

4. 설치를 클릭합니다.

5. 각 Operator의 Operator 설치 페이지에서 기본 설정을 수락합니다.

6. 설치를 클릭합니다. 목록에서 다음 Operator에 대한 단계를 반복하기 전에 Operator가 설치될 때까지 기다립니다.

   - OpenShift Elasticsearch Operator는 openshift-operators-redhat 네임스페이스에 설치되며 클러스터의 모든 네임스페이스에서 사용할 수 있습니다.
   - Red Hat OpenShift distributed tracing platform(Jaeger)은 openshift-distributed-tracing 네임스페이스에 설치되고 클러스터의 모든 네임스페이스에서 사용할 수 있습니다.
   - Red Hat 및 Red Hat OpenShift Service Mesh Operator에서 제공하는 Kiali Operator는 openshift-operators 네임스페이스에 설치되고 클러스터의 모든 네임스페이스에서 사용할 수 있습니다.

7. 4개의 Operator를 모두 설치한 후 Operators → 설치된 Operators를 클릭하여 Operator가 설치되었는지 확인합니다.

2.5.6. Red Hat OpenShift Service Mesh Control Plane 배포

ServiceMeshControlPlane 리소스는 설치 중에 사용할 구성을 정의합니다. Red Hat에서 제공하는 기본 구성을 배포하거나 비즈니스 요구에 맞게 ServiceMeshControlPlane 파일을 사용자 지정할 수 있습니다.

OpenShift Container Platform 웹 콘솔을 사용하거나 oc 클라이언트 도구를 사용하는 명령줄에서 Service Mesh Control Plane을 배포할 수 있습니다.

2.5.6.1. 웹 콘솔에서 컨트롤 플레인 배포

웹 콘솔을 사용하여 **Red Hat OpenShift Service Mesh Control Plane**을 배포하려면 다음 절차를 따르십시오. 이 예제에서 **istio-system**은 컨트롤 플레인 프로젝트의 이름입니다.

사전 요구 사항

- **Red Hat OpenShift Service Mesh Operator**가 설치되어 있어야 합니다.
- **Red Hat OpenShift Service Mesh** 설치를 사용자 지정하는 방법에 대한 지침을 검토하십시오.
- **cluster-admin** 역할이 있는 계정.

절차

1. **cluster-admin** 역할의 사용자로 **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. **istio-system**이라는 프로젝트를 생성합니다.
   a. 홈 → 프로젝트로 이동합니다.
   b. 프로젝트 만들기를 클릭합니다.
   c. 이름 필드에 **istio-system**을 입력합니다.
   d. 생성을 클릭합니다.
3. **Operators** → 설치된 **Operator**로 이동합니다.
4. 필요한 경우 프로젝트 메뉴에서 **istio-system**을 선택합니다. **Operator**가 새 프로젝트에 복사될 때까지 몇 분 정도 기다려야 할 수 있습니다.

5. **Red Hat OpenShift Service Mesh Operator**를 클릭합니다. 제공된 **API**에서 **Operator**는 두 가지 리소스 유형을 생성할 수 있는 링크를 제공합니다.

   - **ServiceMeshControlPlane** 리소스
   - **ServiceMeshMemberRoll** 리소스

6. **Istio Service Mesh Control Plane**에서 **ServiceMeshControlPlane** 만들기를 클릭합니다.

7. **Service Mesh Control Plane** 페이지에서 필요에 따라 기본 **ServiceMeshControlPlane** 템플릿에 대한 **YAML**을 수정합니다.

참고

컨트롤 플레인 사용자 지정에 대한 자세한 내용은 **Red Hat OpenShift Service Mesh** 설치 사용자 지정을 참조하십시오. 프로덕션의 경우 기본 **Jaeger** 템플릿을 *변경해야 합니다*.

8. 생성을 클릭하여 컨트롤 플레인을 생성합니다. **Operator**는 구성 매개변수를 기반으로 **pods**, 서비스 및 **Service Mesh Control Plane** 구성 요소를 생성합니다.

9. **Istio Service Mesh Control Plane** 탭을 클릭합니다.

10. 새 컨트롤 플레인의 이름을 클릭합니다.

11. 리소스 탭을 클릭하여 **Operator**가 생성 및 구성된 **Red Hat OpenShift Service Mesh Control Plane** 리소스를 확인합니다.

2.5.6.2. CLI에서 컨트롤 플레인 배포

Red Hat OpenShift Service Mesh Control Plane에 명령줄을 배포하려면 다음 절차를 따르십시오.

사전 요구 사항

- Red Hat OpenShift Service Mesh Operator가 설치되어 있어야 합니다.
- Red Hat OpenShift Service Mesh 설치를 사용자 지정하는 방법에 대한 지침을 검토하십시오.
- cluster-admin 역할이 있는 계정.
- OpenShift CLI(oc)에 액세스합니다.

프로세스

1. cluster-admin 역할의 사용자로 OpenShift Container Platform CLI에 로그인합니다.

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

2. istio-system이라는 프로젝트를 생성합니다.

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

3. "Red Hat OpenShift Service Mesh 설치 사용자 지정"에서 확인된 예제를 사용하여 istio-installation.yaml이라는 ServiceMeshControlPlane 파일을 생성합니다. 필요에 따라, 사용 사례와 일치하도록 사용자 지정할 수 있습니다. 프로덕션 배포의 경우 기본 Jaeger 템플릿을 *변경해야 합니다*.

4. 다음 명령을 실행하여 컨트롤 플레인을 배포합니다.

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

5. 다음 명령을 실행하여 컨트롤 플레인 설치 상태를 확인합니다.

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

STATUS 열이 ComponentsReady 이면 설치가 성공적으로 완료되었습니다.

```
NAME          READY   STATUS          PROFILES     VERSION   AGE
basic-install   11/11   ComponentsReady   ["default"]   v1.1.18   4m25s
```

6. 다음 명령을 실행하여 설치 프로세스 중에 **Pod**의 진행 상황을 확인합니다.

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

출력은 다음과 유사합니다.

출력 예

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

멀티 테넌트 설치의 경우, **Red Hat OpenShift Service Mesh**는 클러스터 내에서 여러 개의 독립적인 컨트롤 플레인을 지원합니다. **ServiceMeshControlPlane** 템플릿을 사용하여 재사용 가능한 구성을 생성할 수 있습니다. 자세한 내용은 컨트롤 플레인 템플릿 생성을 참조하십시오.

2.5.7. Red Hat OpenShift Service Mesh 멤버 롤 생성

ServiceMeshMemberRoll 은 **Service Mesh Control Plane**에 속한 프로젝트를 나열합니다. **ServiceMeshMemberRoll**에 나열된 프로젝트만 컨트롤 플레인의 영향을 받습니다. 특정 컨트롤 플레인 배포의 멤버 롤에 추가할 때까지 프로젝트는 서비스 메시에 속하지 않습니다.

ServiceMeshControlPlane과 동일한 프로젝트에서 **default** 라는 **ServiceMeshMemberRoll** 리소스를 생성해야 합니다. **(예: istio-system)**

2.5.7.1. 웹 콘솔에서 멤버 롤 생성

웹 콘솔에서 서비스 메시 멤버 롤에 하나 이상의 프로젝트를 추가할 수 있습니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 프로젝트의 이름입니다.

사전 요구 사항

- **Red Hat OpenShift Service Mesh Operator** 설치 및 검증.
- 서비스 메시에 추가할 기존 프로젝트 목록.

프로세스

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. 메시에 대한 서비스가 아직 없거나 처음부터 시작하려는 경우 애플리케이션에 대한 프로젝트를 생성합니다. **Service Mesh Control Plane**을 설치한 프로젝트와 달라야 합니다.
   a. 홈 → 프로젝트로 이동합니다.
   b. 이름 필드에 이름을 입력합니다.
   c. 생성을 클릭합니다.
3. **Operators** → 설치된 **Operator**로 이동합니다.
4. 프로젝트 메뉴를 클릭하고 목록에서 **ServiceMeshControlPlane** 리소스가 배포되는 프로젝트를 선택합니다(예: **istio-system**).
5. **Red Hat OpenShift Service Mesh Operator**를 클릭합니다.
6. **Istio Service Mesh** 멤버 목록 탭을 클릭합니다.

7. **ServiceMeshMemberRoll** 만들기를 클릭합니다.

8. **Members**를 클릭한 다음 **Value** 필드에 프로젝트 이름을 입력합니다. 여러 프로젝트를 추가할 수 있지만 프로젝트는 하나의 **ServiceMeshMemberRoll** 리소스에만 속할 수 있습니다.

9. 생성을 클릭합니다.

2.5.7.2. CLI에서 멤버 롤 생성

명령줄의 **ServiceMeshMemberRoll**에 프로젝트를 추가할 수 있습니다.

사전 요구 사항

- **Red Hat OpenShift Service Mesh Operator** 설치 및 검증.
- 서비스 메시에 추가할 프로젝트 목록.
- **OpenShift CLI(oc)**에 액세스합니다.

프로세스

1. **OpenShift Container Platform CLI**에 로그인합니다.

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

2. 메시에 대한 서비스가 아직 없거나 처음부터 시작하려는 경우 애플리케이션에 대한 프로젝트를 생성합니다. **Service Mesh Control Plane**을 설치한 프로젝트와 달라야 합니다.

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

3. 프로젝트를 멤버로 추가하려면 다음 예제 **YAML**을 수정합니다. 여러 프로젝트를 추가할 수 있지만 프로젝트는 하나의 **ServiceMeshMemberRoll** 리소스에만 속할 수 있습니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 프로젝트의 이름입니다.

servicemeshmemberroll-default.yaml 예

```
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. 다음 명령을 실행하여 **istio-system** 네임스페이스에 **ServiceMeshMemberRoll** 리소스를 업로드하고 만듭니다.

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

5. 다음 명령을 실행하여 **ServiceMeshMemberRoll**이 성공적으로 생성되었는지 확인합니다.

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

STATUS 열이 **Configured**인 경우 설치가 성공적으로 완료된 것입니다.

2.5.8. 서비스 메시에서 프로젝트 추가 또는 제거

웹 콘솔을 사용하여 기존 **Service Mesh ServiceMeshMemberRoll** 리소스에서 프로젝트를 추가하거나 제거할 수 있습니다.

- 여러 프로젝트를 추가할 수 있지만 프로젝트는 하나의 **ServiceMeshMemberRoll** 리소스에만 속할 수 있습니다.
- 해당 **ServiceMeshControlPlane** 리소스가 삭제되면 **ServiceMeshMemberRoll** 리소스가 삭제됩니다.

2.5.8.1. 웹 콘솔을 사용하여 멤버 롤에서 프로젝트 추가 또는 제거

사전 요구 사항

- Red Hat OpenShift Service Mesh Operator 설치 및 검증.
- 기존 ServiceMeshMemberRoll 리소스.
- ServiceMeshMemberRoll 리소스를 사용한 프로젝트의 이름.
- 메시에서 추가하거나 삭제하려는 프로젝트의 이름.

프로세스

1. OpenShift Container Platform 웹 콘솔에 로그인합니다.
2. Operators → 설치된 Operator로 이동합니다.
3. 프로젝트 메뉴를 클릭하고 목록에서 ServiceMeshControlPlane 리소스가 배포되는 프로젝트를 선택합니다(예: istio-system).
4. Red Hat OpenShift Service Mesh Operator를 클릭합니다.
5. Istio Service Mesh 멤버 목록 탭을 클릭합니다.
6. default 링크를 클릭합니다.
7. YAML 탭을 클릭합니다.
8. YAML을 수정하여 프로젝트를 멤버로 추가하거나 제거합니다. 여러 프로젝트를 추가할 수 있지만 프로젝트는 하나의 ServiceMeshMemberRoll 리소스에만 속할 수 있습니다.
9. 저장을 클릭합니다.

10. 새로 고침을 클릭합니다.

2.5.8.2. CLI를 사용하여 멤버 롤에서 프로젝트 추가 또는 제거

명령줄을 사용하여 기존 서비스 메시 멤버 목록을 수정할 수 있습니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh Operator 설치 및 검증.
- 기존 ServiceMeshMemberRoll 리소스.
- ServiceMeshMemberRoll 리소스를 사용한 프로젝트의 이름.
- 메시에서 추가하거나 삭제하려는 프로젝트의 이름.
- OpenShift CLI(oc)에 액세스합니다.

프로세스

1. OpenShift Container Platform CLI에 로그인합니다.
2. ServiceMeshMemberRoll 리소스를 편집합니다.

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

3. YAML을 수정하여 프로젝트를 멤버로 추가하거나 제거합니다. 여러 프로젝트를 추가할 수 있지만 프로젝트는 하나의 ServiceMeshMemberRoll 리소스에만 속할 수 있습니다.

servicemeshmemberroll-default.yaml 예

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

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

2.5.9. 수동 업데이트

수동으로 업데이트하는 경우 OLM(Operator Lifecycle Manager)은 클러스터에서 Operator의 설치, 업그레이드 및 역할 기반 액세스 제어(RBAC)를 제어합니다. OLM은 OpenShift Container Platform에서 기본적으로 실행됩니다. OLM은 Operator Registry API를 사용하는 CatalogSources를 통해 활용 가능한 Operator와 설치된 Operator의 업그레이드를 쿼리합니다.

- OpenShift Container Platform의 업그레이드 처리 방법에 대한 자세한 내용은 Operator Lifecycle Manager 설명서를 참조하십시오.

2.5.9.1. 사이드카 프록시 업데이트

사이드카 프록시의 구성을 업데이트하려면 애플리케이션 관리자가 애플리케이션 Pod를 다시 시작해야 합니다.

배포 시 자동 사이드카 삽입을 사용하는 경우 주석을 추가하거나 수정하여 배포에서 pod 템플릿을 업데이트할 수 있습니다. 다음 명령을 실행하여 pod를 다시 배포합니다.

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

배포에서 자동 사이드카 삽입을 사용하지 않는 경우 배포 또는 Pod에 지정된 사이드카 컨테이너 이미지를 수정하여 사이드카를 수동으로 업데이트한 다음 Pod를 다시 시작해야 합니다.

2.5.10. 다음 단계

- Red Hat OpenShift Service Mesh에 애플리케이션을 배포할 준비를 합니다.

2.6. 서비스 메시에서 보안 사용자 정의

주의

더 이상 지원되지 않는 **Red Hat OpenShift Service Mesh** 릴리스에 대한 문서를 보고 있습니다.

서비스 메시 버전 **1.0** 및 **1.1** 컨트롤 플레인은 더 이상 지원되지 않습니다. 서비스 메시 컨트롤 플레인 업그레이드에 대한 자세한 내용은 서비스 메시 업그레이드를 참조하십시오.

특정 **Red Hat OpenShift Service Mesh** 릴리스의 지원 상태에 대한 자세한 내용은 제품 라이프사이클 페이지를 참조하십시오.

서비스 메시 애플리케이션이 복잡한 마이크로 서비스를 사용하여 구성된 경우 **Red Hat OpenShift Service Mesh**를 사용하여 해당 서비스 간 통신 보안을 사용자 지정할 수 있습니다. 서비스 메시의 트래픽 관리 기능과 함께 **OpenShift Container Platform**의 인프라는 애플리케이션의 복잡성을 관리하고 마이크로서비스에 서비스 및 **ID** 보안을 제공할 수 있습니다.

2.6.1. mTLS(mutual Transport Layer Security) 활성화

mTLS(mutual Transport Layer Security)은 두 당사자가 서로 인증하는 프로토콜입니다. 일부 프로토콜**(IKE, SSH)**에서는 기본 인증 모드이며, 다른 프로토콜**(TLS)**에서는 선택적입니다.

mTLS는 애플리케이션이나 서비스 코드에 대한 변경 없이 사용할 수 있습니다. **TLS**는 서비스 메시 인프라와 두 사이드카 프록시 사이에서 전적으로 처리됩니다.

기본적으로 **Red Hat OpenShift Service Mesh**가 허용 모드로 설정됩니다. 여기서 서비스 메시의 사이드카는 일반 텍스트 트래픽과 **mTLS**를 사용하여 암호화된 연결을 모두 허용합니다. 메시의 서비스가 메시 외부 서비스와 통신하는 경우 엄격한 **mTLS**가 해당 서비스 간의 통신을 중단할 수 있습니다. 워크로드를 서비스 메시로 마이그레이션하는 동안 허용 모드를 사용합니다.

2.6.1.1. 메시에서 엄격한 mTLS 활성화

워크로드가 메시 외부의 서비스와 통신하지 않고 암호화된 연결만 수락하여 통신이 중단되지 않는 경

우 메시 전체에서 **mTLS**를 빠르게 활성화할 수 있습니다. **ServiceMeshControlPlane** 리소스에서 **spec.istio.global.mtls.enabled**를 **true**로 설정합니다. **Operator**는 필요한 리소스를 생성합니다.

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

2.6.1.1.1. 특정 서비스의 수신 연결에 대해 사이드카 구성

정책을 생성하여 개별 서비스 또는 네임스페이스에 대해 **mTLS**를 구성할 수도 있습니다.

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

2.6.1.2. 발신 연결에 대한 사이드카 구성

메시에서 다른 서비스로 요청을 보낼 때 **mTLS**를 사용하도록 서비스 메시를 구성하는 대상 규칙을 생성합니다.

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

2.6.1.3. 최소 및 최대 프로토콜 버전 설정

사용자 환경에 서비스 메시의 암호화된 트래픽에 대한 특정 요구 사항이 있는 경우 **ServiceMeshControlPlane** 리소스에 **spec.security.controlPlane.tls.minProtocolVersion** 또는 **spec.security.controlPlane.tls.maxProtocolVersion**을 설정하여 허용되는 암호화 기능을 제어할 수 있

습니다. 컨트롤 플레인 리소스에 구성된 해당 값은 **TLS**를 통해 안전하게 통신할 때 메시 구성 요소에서 사용하는 최소 및 최대 **TLS** 버전을 정의합니다.

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  istio:
    global:
      tls:
        minProtocolVersion: TLSv1_2
        maxProtocolVersion: TLSv1_3
```

기본값은 **TLS_AUTO**이며 **TLS** 버전을 지정하지 않습니다.

표 **2.3.** 유효한 값

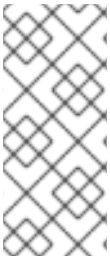
값	설명
TLS_AUTO	default
TLSv1_0	TLS 버전 1.0
TLSv1_1	TLS 버전 1.1
TLSv1_2	TLS 버전 1.2
TLSv1_3	TLS 버전 1.3

2.6.2. 암호화 제품군 및 ECDH 곡선 구성

암호화 제품군 및 **ECDH(Elliptic-curve Diffie–Hellman)** 곡선은 서비스 메시를 보호하는 데 도움이 될 수 있습니다. **ServiceMeshControlPlane** 리소스에서 **spec.istio.global.tls.cipherSuites**를 사용하는 암호화 제품군과 **spec.istio.global.tls.ecdhCurves**를 사용하는 **ECDH** 곡선을 쉼표로 구분된 목록으로 정의할 수 있습니다. 이러한 속성 중 하나가 비어 있으면 기본값이 사용됩니다.

서비스 메시에서 **TLS 1.2** 또는 이전 버전을 사용하는 경우 **cipherSuites** 설정이 적용됩니다. **TLS 1.3**을 사용할 때는 효과가 없습니다.

우선순위에 따라 암호화 제품군을 쉼표로 구분된 목록으로 설정합니다. 예를 들어 **ecdhCurves: CurveP256, CurveP384**는 **CurveP256**을 **CurveP384**보다 높은 우선순위로 설정합니다.



참고

암호화 제품군을 구성할 때 **TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256** 또는 **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256**을 포함해야 합니다. **HTTP/2** 지원에는 이러한 암호화 제품군 중 하나 이상이 필요합니다.

지원되는 암호화 제품군은 다음과 같습니다.

- **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

지원되는 ECDH 곡선은 다음과 같습니다.

- CurveP256
- CurveP384
- CurveP521
- X25519

2.6.3. 외부 인증 기관 키 및 인증서 추가

기본적으로 Red Hat OpenShift Service Mesh는 자체 서명된 루트 인증서와 키를 생성하고 이를 사용하여 워크로드 인증서에 서명합니다. 사용자 정의 인증서 및 키를 사용하여 사용자 정의 루트 인증서로

워크로드 인증서에 서명할 수도 있습니다. 이 작업은 인증서와 키를 서비스 메시에 연결하는 예제를 보여줍니다.

사전 요구 사항

- 인증서를 구성하려면 상호 TLS가 활성화된 Red Hat OpenShift Service Mesh를 설치해야 합니다.
- 이 예제에서는 Maistra 리포지토리의 인증서를 사용합니다. 프로덕션의 경우 인증 기관의 자체 인증서를 사용합니다.
- 이러한 지침으로 결과를 확인하려면 Bookinfo 샘플 애플리케이션을 배포해야 합니다.

2.6.3.1. 기존 인증서 및 키 추가

기존 서명(CA) 인증서 및 키를 사용하려면 CA 인증서, 키, 루트 인증서가 포함된 신뢰 파일 체인을 생성해야 합니다. 해당 인증서 각각에 대해 다음과 같은 정확한 파일 이름을 사용해야 합니다. CA 인증서를 ca-cert.pem, 키는 ca-key.pem이라고 합니다. ca-cert.pem을 서명하는 루트 인증서는 root-cert.pem이라고 합니다. 워크로드에서 중개 인증서를 사용하는 경우 cert-chain.pem 파일에 인증서를 지정해야 합니다.

다음 단계에 따라 서비스 메시에 인증서를 추가합니다. Maistra 리포지토리에서 로컬로 예제 인증서를 저장하고 &lt ;path& gt;를 인증서 경로로 바꿉니다.

1. 입력 파일 ca-cert.pem, ca-key.pem, root-cert.pem, cert-chain.pem을 포함하는 시크릿 cacert를 생성합니다.

```
$ oc create secret generic cacerts -n istio-system --from-file=<path>/ca-cert.pem \
    --from-file=<path>/ca-key.pem --from-file=<path>/root-cert.pem \
    --from-file=<path>/cert-chain.pem
```

2. ServiceMeshControlPlane 리소스에서 global.mtls.enabled 를 true 로 설정하고 security.selfSigned 를 false 로 설정합니다. 서비스 메시는 secret-mount 파일에서 인증서와 키를 읽습니다.

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

```
    mtls:
      enabled: true
  security:
    selfSigned: false
```

3. 워크로드에 새 인증서를 즉시 추가하려면 서비스 메시에서 생성한 **istio.***라는 시크릿을 삭제합니다. 이 예제에서는 **istio.default** 입니다. 서비스 메시는 워크로드에 대한 새 인증서를 발행합니다.

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

2.6.3.2. 인증서 확인

Bookinfo 샘플 애플리케이션을 사용하여 인증서가 올바르게 마운트되었는지 확인합니다. 먼저 마운트된 인증서를 검색합니다. 그런 다음 **pod**에 마운트된 인증서를 확인합니다.

1. **pod** 이름을 변수 **RATINGSPOD**에 저장합니다.

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

2. 다음 명령을 실행하여 프록시에 마운트된 인증서를 검색합니다.

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

/tmp/pod-root-cert.pem 파일에는 **Pod**로 전달된 루트 인증서가 포함되어 있습니다.

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

/tmp/pod-cert-chain.pem 파일에는 **Pod**로 전달된 워크로드 인증서와 **CA** 인증서가 포함되어 있습니다.

3. 루트 인증서가 **Operator**가 지정한 것과 동일한지 확인합니다. **<path>**를 인증서 경로로 교체합니다.

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

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

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

출력 대상이 비어 있을 것으로 예상됩니다.

4. CA 인증서가 Operator가 지정한 것과 동일한지 확인합니다. <path>를 인증서 경로로 교체합니다.

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

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

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

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

출력 대상이 비어 있을 것으로 예상됩니다.

5. 루트 인증서에서 워크로드 인증서로의 인증서 체인을 확인합니다. <path>를 인증서 경로로 교체합니다.

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

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

출력 예

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

2.6.3.3. 인증서 제거

추가한 인증서를 제거하려면 다음 단계를 따르십시오.

1. 시크릿 **cacerts**를 제거합니다.

```
$ oc delete secret cacerts -n istio-system
```

2. **ServiceMeshControlPlane** 리소스에서 자체 서명된 루트 인증서로 서비스 메시를 재배포합니다.

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

2.7. 트래픽 관리

주의

더 이상 지원되지 않는 **Red Hat OpenShift Service Mesh** 릴리스에 대한 문서를 보고 있습니다.

서비스 메시 버전 **1.0** 및 **1.1** 컨트롤 플레인은 더 이상 지원되지 않습니다. 서비스 메시 컨트롤 플레인 업그레이드에 대한 자세한 내용은 서비스 메시 업그레이드를 참조하십시오.

특정 **Red Hat OpenShift Service Mesh** 릴리스의 지원 상태에 대한 자세한 내용은 제품 라이프사이클 페이지를 참조하십시오.

Red Hat OpenShift Service Mesh의 서비스 간 트래픽 및 **API** 호출 흐름을 제어할 수 있습니다. 예를 들어, 서비스 메시의 일부 서비스는 메시 내에서 통신해야 하며 다른 서비스는 숨겨야 할 수 있습니다. 트래픽을 관리하여 특정 백엔드 서비스를 숨기고, 서비스를 노출하며, 테스트 또는 버전 관리 배포를 생성하거나 서비스 세트에 보안 계층을 추가합니다.

2.7.1. 게이트웨이 사용

게이트웨이를 사용하여 메시에 대한 인바운드 및 아웃바운드 트래픽을 관리하여 메시에 들어가거나 나가려는 트래픽을 지정할 수 있습니다. 게이트웨이 구성은 서비스 워크로드와 함께 실행되는 사이드카 **Envoy** 프록시가 아닌, 메시의 에지에서 실행되는 독립 실행형 **Envoy** 프록시에 적용됩니다.

Kubernetes Ingress API와 같이 시스템으로 들어오는 트래픽을 제어하는 다른 메커니즘과 달리 **Red Hat OpenShift Service Mesh** 게이트웨이는 트래픽 라우팅의 모든 기능과 유연성을 사용합니다.

Red Hat OpenShift Service Mesh 게이트웨이 리소스는 **Red Hat OpenShift Service Mesh TLS** 설정을 노출하고 구성하기 위해 포트와 같은 계층 **4-6** 로드 밸런싱 속성을 사용할 수 있습니다. 애플리케이션 계층 트래픽 라우팅(**L7**)을 동일한 **API** 리소스에 추가하는 대신, 일반 **Red Hat OpenShift Service Mesh** 가상 서비스를 게이트웨이에 바인딩하고 서비스 메시의 다른 데이터 플레인 트래픽처럼 게이트웨이 트래픽을 관리할 수 있습니다.

게이트웨이는 주로 수신 트래픽을 관리하는 데 사용되지만 송신 게이트웨이를 구성할 수도 있습니다. 송신 게이트웨이를 사용하면 메시를 나가는 트래픽에 대해 전용 종료 노드를 구성할 수 있습니다. 이를 통해 외부 네트워크에 대한 액세스 권한이 있는 서비스를 제한하여 서비스 메시에 보안 제어를 추가할 수 있습니다. 게이트웨이를 사용하여 전적으로 내부 프록시를 구성할 수도 있습니다.

게이트웨이 예제

게이트웨이 리소스는 들어오거나 나가는 **HTTP/TCP** 연결을 수신하는 메시의 에지에서 작동하는 로드 밸런서를 설명합니다. 사양은 노출되는 포트 세트, 사용할 프로토콜 유형, 로드 밸런서에 대한 **SNI** 구성 등을 설명합니다.

다음 예제는 외부 **HTTPS** 수신 트래픽에 대해 샘플 게이트웨이 구성을 보여줍니다.

```
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
```

이 게이트웨이 구성으로 ext-host.example.com의 HTTPS 트래픽을 포트 443의 메시로 허용할 수 있지만 트래픽에 라우팅을 지정하지 않습니다.

라우팅을 지정하고 게이트웨이가 의도한 대로 작동하려면 게이트웨이도 가상 서비스에 바인딩해야 합니다. 다음 예와 같이 가상 서비스의 게이트웨이 필드를 사용하여 이 작업을 수행합니다.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: virtual-svc
spec:
  hosts:
  - ext-host.example.com
  gateways:
    - ext-host-gwy
```

그러면 외부 트래픽에 대한 라우팅 규칙으로 가상 서비스를 구성할 수 있습니다.

2.7.2. 수신 게이트웨이 구성

Ingress 게이트웨이는 들어오는 HTTP/TCP 연결을 수신하는 메시의 에지에서 작동하는 로드 밸런서입니다. 노출된 포트와 프로토콜을 구성하지만 트래픽 라우팅 구성은 포함하지 않습니다. Ingress 트래픽의 트래픽 라우팅은 내부 서비스 요청과 동일한 방식으로 라우팅 규칙으로 구성됩니다.

다음 단계에서는 게이트웨이를 만들고 Bookinfo 샘플 애플리케이션에서 서비스를 /productpage 및 /login. 경로의 외부 트래픽에 노출하도록 VirtualService를 구성하는 방법을 보여줍니다.

절차

1. 트래픽을 수락하기 위해 게이트웨이를 만듭니다.

   a. YAML 파일을 생성한 후 다음 YAML을 이 파일에 복사합니다.

   게이트웨이 예제 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 파일을 적용합니다.

```
$ oc apply -f gateway.yaml
```

2. VirtualService 오브젝트를 생성하여 호스트 헤더를 다시 작성합니다.

a. YAML 파일을 생성한 후 다음 YAML을 이 파일에 복사합니다.

가상 서비스 예

```
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 파일을 적용합니다.

```
$ oc apply -f vs.yaml
```

3. 게이트웨이 및 VirtualService가 올바르게 설정되었는지 확인합니다.

a. 게이트웨이 URL을 설정합니다.

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

b. 포트 번호를 설정합니다. 이 예제에서 istio-system 은 Service Mesh Control Plane 프로젝트의 이름입니다.

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

c. 명시적으로 노출된 페이지를 테스트합니다.

```
curl -s -I "$GATEWAY_URL/productpage"
```

예상 결과는 200입니다.

2.7.3. Ingress 트래픽 관리

Red Hat OpenShift Service Mesh에서 Ingress Gateway는 모니터링, 보안 및 라우팅 규칙과 같은 기능을 클러스터에 들어오는 트래픽에 적용할 수 있도록 합니다. 서비스 메시 게이트웨이를 사용하여 서비스 메시 외부에서 서비스를 노출합니다.

2.7.3.1. Ingress IP 및 포트 확인

Ingress 구성은 환경에서 외부 로드 밸런서를 지원하는지 여부에 따라 달라집니다. 외부 로드 밸런서는 클러스터의 Ingress IP 및 포트에 설정됩니다. 클러스터의 IP 및 포트가 외부 로드 밸런서에 구성되어 있는지 확인하려면 다음 명령을 실행합니다. 이 예제에서 istio-system 은 Service Mesh Control Plane 프로젝트의 이름입니다.

```
$ oc get svc istio-ingressgateway -n istio-system
```

해당 명령은 네임스페이스에 있는 각 항목의 NAME, TYPE, CLUSTER-IP, EXTERNAL-IP, PORT(S), AGE를 반환합니다.

EXTERNAL-IP 값이 설정되면 해당 환경에 Ingress 게이트웨이에 사용할 수 있는 외부 로드 밸런서가 있습니다.

EXTERNAL-IP 값이 <none> 또는 영구적으로 <pending>인 경우, 해당 환경은 Ingress 게이트웨이에 외부 로드 밸런서를 제공하지 않습니다. 서비스의 노드 포트를 사용하여 게이트웨이에 액세스할 수 있습니다.

2.7.3.1.1. 로드 밸런서를 사용하여 Ingress 포트 확인

환경에 외부 로드 밸런서가 있는 경우 다음 지침을 따릅니다.

절차

1. 다음 명령을 실행하여 Ingress IP 및 포트를 설정합니다. 이 명령은 터미널에서 변수를 설정합니다.

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

2. 다음 명령을 실행하여 Ingress 포트를 설정합니다.

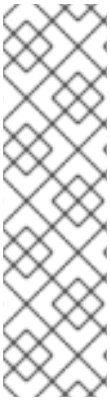
```
$ export INGRESS_PORT=$(oc -n istio-system get service istio-ingressgateway -o
jsonpath='{.spec.ports[?(@.name=="http2")].port}')
```

3. 다음 명령을 실행하여 보안 Ingress 포트를 설정합니다.

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

4. 다음 명령을 실행하여 **TCP Ingress** 포트를 설정합니다.

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



참고

일부 환경에서는 **IP** 주소 대신 호스트 이름을 사용하여 로드 밸런서가 노출될 수 있습니다. 이 경우 **Ingress** 게이트웨이의 **EXTERNAL-IP** 값은 **IP** 주소가 아닙니다. 대신 호스트 이름이며 이전 명령은 **INGRESS_HOST** 환경 변수를 설정하지 못합니다.

이 경우 다음 명령을 사용하여 **INGRESS_HOST** 값을 수정합니다.

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

2.7.3.1.2. 로드 밸런서 없이 Ingress 포트 확인

환경에 외부 로드 밸런서가 없는 경우 **Ingress** 포트를 확인하고 대신 노드 포트를 사용합니다.

절차

1. **Ingress** 포트를 설정합니다.

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

2. 다음 명령을 실행하여 보안 **Ingress** 포트를 설정합니다.

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

3. 다음 명령을 실행하여 **TCP Ingress** 포트를 설정합니다.

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

2.7.4. 자동 경로 생성

Istio 게이트웨이의 OpenShift 경로는 Red Hat OpenShift Service Mesh에서 자동으로 관리됩니다. Istio 게이트웨이가 서비스 메시 내부에서 생성, 업데이트 또는 삭제될 때마다 OpenShift 경로가 생성, 업데이트 또는 삭제됩니다.

2.7.4.1. 자동 경로 생성 활성화

IOR(Istio OpenShift Routing)이라는 Red Hat OpenShift Service Mesh Control Plane 구성 요소는 게이트웨이 경로를 동기화합니다. 컨트롤 플레인 배포의 일부로 IOR을 활성화합니다.

게이트웨이에 TLS 섹션이 포함된 경우 OpenShift 경로는 TLS를 지원하도록 구성됩니다.

1. ServiceMeshControlPlane리소스에서 ior_enabled 매개변수를 추가하고 true로 설정합니다. 예를 들어, 다음 리소스 스니펫을 참조하십시오.

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

2.7.4.2. 하위 도메인

Red Hat OpenShift Service Mesh 는 하위 도메인으로 경로를 생성하지만 이를 활성화하려면 OpenShift Container Platform을 구성해야 합니다. 하위 도메인(예: *.domain.com)은 지원되지만 기본적으로 아닙니다. 와일드카드 호스트 게이트웨이를 구성하기 전에 OpenShift Container Platform 와일드카드 정책을 구성합니다. 자세한 내용은 “링크" 섹션을 참조하십시오.

다음 게이트웨이가 생성되는 경우:

```
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
```

이제 다음 **OpenShift** 경로가 자동으로 생성됩니다. 다음 명령을 사용하여 경로가 생성되었는지 확인할 수 있습니다.

```
$ oc -n <control_plane_namespace> get routes
```

예상 출력

```
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
```

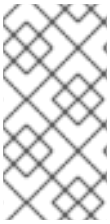
게이트웨이가 삭제되면 **Red Hat OpenShift Service Mesh**가 경로를 삭제합니다. 그러나 수동으로 생성된 경로는 **Red Hat OpenShift Service Mesh**에 의해 수정되지 않습니다.

2.7.5. 서비스 항목 이해

서비스 항목은 **Red Hat OpenShift Service Mesh**가 내부적으로 관리하는 서비스 레지스트리에 항목을 추가합니다. 서비스 항목을 추가한 후 **Envoy** 프록시는 메시의 서비스인 것처럼 서비스에 트래픽을 보냅니다. 서비스 항목을 사용하면 다음을 수행할 수 있습니다.

- 서비스 메시 외부에서 실행되는 서비스의 트래픽을 관리합니다.
- 웹에서 소비된 **API** 또는 레거시 인프라의 서비스에 대한 트래픽과 같은 외부 대상의 트래픽을 리디렉션 및 전달합니다.
- 외부 대상에 대한 재시도, 시간 초과 및 오류 삽입 정책을 정의합니다.

- VM(가상 머신)에서 메시에 VM을 추가하여 메시 서비스를 실행합니다.



참고

Kubernetes에서 다중 클러스터 Red Hat OpenShift Service Mesh 메시를 구성하기 위해 다른 클러스터의 서비스를 메시에 추가합니다.

서비스 항목 예

다음 예제는 Red Hat OpenShift Service Mesh 서비스 레지스트리에 ext-resource 외부 종속성을 추가하는 mesh-external 서비스 항목입니다.

```
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 필드를 사용하여 외부 리소스를 지정합니다. 완전히 한정하거나 와일드카드 접두사 도메인 이름을 사용할 수 있습니다.

메시의 다른 서비스에 대한 트래픽을 구성하는 것과 동일한 방식으로 서비스 항목에 대한 트래픽을 제어하도록 가상 서비스 및 대상 규칙을 구성할 수 있습니다. 예를 들어 다음 대상 규칙은 서비스 항목을 사용하여 구성된 ext-svc.example.com 외부 서비스에 대한 연결을 보호하기 위해 상호 TLS를 사용하도록 트래픽 경로를 구성합니다.

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: ext-res-dr
spec:
  host: ext-svc.example.com
  trafficPolicy:
    tls:
      mode: MUTUAL
```

```
clientCertificate: /etc/certs/myclientcert.pem
privateKey: /etc/certs/client_private_key.pem
caCertificates: /etc/certs/rootcacerts.pem
```

2.7.6. VirtualService 사용

가상 서비스가 있는 **Red Hat OpenShift Service Mesh**를 통해 여러 버전의 마이크로 서비스로 요청을 동적으로 라우팅할 수 있습니다. 가상 서비스를 사용하면 다음을 수행할 수 있습니다.

- 단일 가상 서비스를 통해 여러 애플리케이션 서비스를 처리합니다. 예를 들어 메시에서 **Kubernetes**를 사용하는 경우 특정 네임스페이스의 모든 서비스를 처리하도록 가상 서비스를 구성할 수 있습니다. 가상 서비스를 사용하면 모놀리식 애플리케이션을 원활한 소비자 환경을 통해 별도의 마이크로 서비스로 구성된 서비스로 전환할 수 있습니다.
- 게이트웨이와 결합하여 트래픽 규칙을 구성하고 수신 및 송신 트래픽을 제어합니다.

2.7.6.1. VirtualService 구성

요청은 가상 서비스를 통해 서비스 메시 내의 서비스로 라우팅됩니다. 각 가상 서비스는 순서대로 평가되는 라우팅 규칙 세트로 구성됩니다. **Red Hat OpenShift Service Mesh**는 가상 서비스에 대해 주어진 각 요청을 메시 내의 실제 특정 대상에 연결합니다.

가상 서비스가 없으면 **Red Hat OpenShift Service Mesh**는 모든 서비스 인스턴스 간에 최소 요청 부하 분산을 사용하여 트래픽을 배포합니다. 가상 서비스에서는 하나 이상의 호스트 이름에 대한 트래픽 동작을 지정할 수 있습니다. 가상 서비스의 라우팅 규칙은 가상 서비스에 대한 트래픽을 적절한 대상으로 전송하는 방법을 **Red Hat OpenShift Service Mesh**에 알립니다. 경로 대상은 동일한 서비스 또는 완전히 다른 서비스 버전일 수 있습니다.

절차

1. 다음 예제를 사용하여 **YAML** 파일을 생성하여 애플리케이션에 연결하는 사용자에 따라 **Bookinfo** 샘플 애플리케이션 서비스의 다양한 버전으로 요청을 라우팅합니다.

예: VirtualService.yaml

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

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

2. 다음 명령을 실행하여 **VirtualService.yaml**을 적용합니다. 여기서 **VirtualService.yaml**은 파일 경로입니다.

```
$ oc apply -f <VirtualService.yaml>
```

2.7.6.2. VirtualService 구성 참조

매개변수	설명
`spec:` `  hosts:`	**hosts** 필드에는 라우팅 규칙이 적용되는 가상 서비스의 대상 주소가 나열됩니다. 이는 서비스에 요청을 보내는 데 사용되는 주소입니다. 가상 서비스 호스트 이름은 IP 주소, DNS 이름 또는 정규화된 도메인 이름으로 확인되는 짧은 이름일 수 있습니다.

`spec:`
`  http:`
`  - match:`
	http 섹션에는 호스트 필드에 지정된 대상으로 전송된 HTTP/1.1, HTTP2, gRPC 트래픽을 라우팅하기 위한 일치 조건 및 작업을 설명하는 가상 서비스의 라우팅 규칙이 포함됩니다. 라우팅 규칙은 트래픽을 이동할 대상과 지정된 일치 조건으로 구성됩니다. 예제의 첫 번째 라우팅 규칙에는 일치 필드로 시작하는 조건이 있습니다. 이 예제에서 이 라우팅은 사용자 **jason**의 모든 요청에 적용됩니다. **headers**, **end-user**, **exact** 필드를 추가하여 적절한 요청을 선택합니다.

매개변수	설명

spec:
  http:
  - match:
  - destination:
	경로 섹션의 **destination** 필드는 이 조건과 일치하는 트래픽에 대한 실제 대상을 지정합니다. 가상 서비스의 호스트와 달리 대상 호스트는 Red Hat OpenShift Service Mesh 서비스 레지스트리에 있는 실제 대상이어야 합니다. 프록시가 있는 메시 서비스 또는 서비스 항목을 사용하여 추가된 비 메시 서비스일 수 있습니다. 이 예제에서 호스트 이름은 Kubernetes 서비스 이름입니다.

2.7.7. 대상 규칙 이해

대상 규칙은 가상 서비스 라우팅 규칙이 평가된 후에 적용되므로 트래픽의 실제 대상에 적용됩니다. 가상 서비스는 트래픽을 대상으로 라우팅합니다. 대상 규칙은 해당 대상의 트래픽에 발생하는 요소를 설정합니다.

기본적으로 Red Hat OpenShift Service Mesh는 최소 요청 부하 분산 정책을 사용합니다. 여기서 활성 연결 수가 가장 적은 풀의 서비스 인스턴스에서 요청을 수신합니다. 또한 Red Hat OpenShift Service Mesh는 특정 서비스 또는 서비스 하위 집합에 대한 요청의 대상 규칙에 지정할 수 있는 다음과 같은 모델을 지원합니다.

- **Random:** 요청은 풀의 인스턴스에 무작위로 전달됩니다.
- **Weighted:** 요청은 구체적인 비율에 따라 풀의 인스턴스로 전달됩니다.
- **Least requests:** 요청은 요청 수가 가장 적은 인스턴스로 전달됩니다.

대상 규칙 예

다음 예제 대상 규칙은 서로 다른 로드 밸런싱 정책을 사용하여 my-svc 대상 서비스에 대해 세 가지 다른 하위 집합을 구성합니다.

```
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
```

이 안내서는 **Bookinfo** 샘플 애플리케이션을 참조하여 예제 애플리케이션에 라우팅 예제를 제공합니다. **Bookinfo** 애플리케이션을 설치하여 이러한 라우팅 예제가 작동하는 방법을 확인합니다.

2.7.8. Bookinfo 라우팅 튜토리얼

Service Mesh Bookinfo 샘플 애플리케이션은 각각 여러 가지 버전이 있는 네 개의 마이크로 서비스로 구성됩니다. **Bookinfo** 샘플 애플리케이션을 설치한 후에는 **reviews** 마이크로 서비스의 세 가지 버전이 동시에 실행됩니다.

브라우저에서 **Bookinfo** 앱 **/product** 페이지에 액세스하여 여러 번 새로 고침하면 북 리뷰 출력에 별점이 포함된 경우도 있고 그렇지 않은 경우도 있습니다. 라우팅할 명시적인 기본 서비스 버전이 없으면 서비스 메시는 사용 가능한 모든 버전으로 차례대로 요청을 라우팅합니다.

이 튜토리얼은 모든 트래픽을 마이크로 서비스의 **v1**(버전 **1**)으로 라우팅하는 규칙을 적용하는 데 도움이 됩니다. 나중에 **HTTP** 요청 헤더의 값을 기반으로 트래픽을 라우팅하는 규칙을 적용할 수 있습니다.

사전 요구 사항

- 다음 예제에서 작동하도록 **Bookinfo** 샘플 애플리케이션을 배포하십시오.

2.7.8.1. 가상 서비스 적용

다음 절차에서 가상 서비스는 마이크로 서비스의 기본 버전을 설정하는 가상 서비스를 적용하여 모든 트래픽을 각 마이크로 서비스의 **v1**로 라우팅합니다.

절차

1. 가상 서비스를 적용합니다.

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

2. 가상 서비스를 적용했는지 확인하려면 다음 명령을 사용하여 정의된 경로를 표시합니다.

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

이 명령은 **kind: VirtualService**의 리소스를 **YAML** 형식으로 반환합니다.

reviews 서비스 버전 **1**을 포함하여 서비스 메시를 **Bookinfo** 마이크로 서비스 **v1** 버전으로 라우팅하도록 구성했습니다.

2.7.8.2. 새 경로 구성 테스트

Bookinfo 앱의 **/productpage**를 다시 새로 고침하여 새 구성을 테스트합니다.

절차

1. **GATEWAY_URL** 매개변수 값을 설정합니다. 이 변수를 사용하여 나중에 **Bookinfo** 제품 페이지의 **URL**을 찾을 수 있습니다. 이 예제에서 컨트롤 플레인 프로젝트는 **istio-system**입니다.

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

2. 다음 명령을 실행하여 제품 페이지의 **URL**을 검색합니다.

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

3. 브라우저에서 **Bookinfo** 사이트를 엽니다.

페이지의 리뷰 부분은 새로 고침 횟수와 관계없이 별점 없이 표시됩니다. 이는 리뷰 서비스의 모든 트래픽을 **reviews:v1** 버전으로 라우팅하도록 서비스 메시를 구성했기 때문이며, 이 서비스 버전은 별점 서비스에 액세스할 수 없습니다.

이제 서비스 메시가 트래픽을 하나의 서비스 버전으로 라우팅합니다.

2.7.8.3. 사용자 ID 기반 경로

특정 사용자의 모든 트래픽이 특정 서비스 버전으로 라우팅되도록 경로 구성을 변경합니다. 이 경우 **jason**이라는 사용자의 모든 트래픽은 서비스 **reviews:v2**로 라우팅됩니다.

서비스 메시에는 사용자 **ID**에 대한 특별한 기본 이해가 없습니다. 이 예제는 **productpage** 서비스가 모든 아웃바운드 **HTTP** 요청에 대한 사용자 정의 **end-user** 헤더를 검토 서비스에 추가한다는 사실에 의해 활성화됩니다.

절차

1. 다음 명령을 실행하여 **Bookinfo** 샘플 애플리케이션에서 사용자 기반 라우팅을 활성화하도록 설정합니다.

```
$ oc apply -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.4/samples/info/networking/virtual-service-reviews-test-v2.yaml
```

2. 다음 명령을 실행하여 규칙이 생성되었는지 확인합니다. 이 명령은 **kind: VirtualService**의 모든 리소스를 **YAML** 형식으로 반환합니다.

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

3. **Bookinfo** 앱의 **/productpage**에서 암호없이 **jason**으로 로그인합니다.

4. 브라우저를 새로 고침합니다. 별점은 각 리뷰 옆에 표시됩니다.

5. 다른 사용자로 로그인합니다(원하는 이름 선택). 브라우저를 새로 고침합니다. 이제 별이 사라졌습니다. **Jason**을 제외한 모든 사용자에 대해 트래픽이 **reviews:v1**으로 라우팅됩니다.

사용자 **ID**를 기반으로 트래픽을 라우팅하도록 **Bookinfo** 샘플 애플리케이션을 성공적으로 구성했습니다.

2.7.9. 추가 리소스

OpenShift Container Platform 와일드카드 정책 구성에 대한 자세한 내용은 와일드카드 경로 사용을 참조하십시오.

2.8. SERVICE MESH에 애플리케이션 배포

주의

더 이상 지원되지 않는 Red Hat OpenShift Service Mesh 릴리스에 대한 문서를 보고 있습니다.

서비스 메시 버전 1.0 및 1.1 컨트롤 플레인은 더 이상 지원되지 않습니다. 서비스 메시 컨트롤 플레인 업그레이드에 대한 자세한 내용은 서비스 메시 업그레이드를 참조하십시오.

특정 Red Hat OpenShift Service Mesh 릴리스의 지원 상태에 대한 자세한 내용은 제품 라이프사이클 페이지를 참조하십시오.

Service Mesh에 애플리케이션을 배포할 때 Istio의 업스트림 커뮤니티 버전과 Red Hat OpenShift Service Mesh 설치 내의 애플리케이션 동작 간의 몇 가지 차이점이 있습니다.

2.8.1. 사전 요구 사항

- Red Hat OpenShift Service Mesh 및 업스트림 Istio 커뮤니티 설치 비교
- Review Installing Red Hat OpenShift Service Mesh

2.8.2. 컨트롤 플레인 템플릿 생성

ServiceMeshControlPlane 템플릿을 사용하여 재사용 가능한 구성을 생성할 수 있습니다. 개별 사용자는 생성한 템플릿을 자체 구성으로 확장할 수 있습니다. 템플릿은 다른 템플릿의 구성 정보를 상속할 수도 있습니다. 예를 들어, 회계 팀에 대한 계정 컨트롤 플레인과 마케팅 팀에 대한 마케팅 컨트롤 플레인을 생성할 수 있습니다. 개발 템플릿과 프로덕션 템플릿을 생성하는 경우 마케팅 팀과 회계 팀의 구성원은 팀별 사용자 지정을 통해 개발 및 프로덕션 템플릿을 확장할 수 있습니다.

ServiceMeshControlPlane과 동일한 구문을 따르는 컨트롤 플레인 템플릿을 구성하면, 사용자는 계층적으로 설정을 상속합니다. Operator는 Red Hat OpenShift Service Mesh의 기본 설정이 포함된 default 템플릿과 함께 제공됩니다. 사용자 지정 템플릿을 추가하려면 openshift-operators 프로젝트에서 smcp-templates라는 ConfigMap을 생성하고 /usr/local/share/istio-operator/templates에서 Operator 컨테이너에 ConfigMap을 마운트해야 합니다.

2.8.2.1. ConfigMap 생성

다음 절차에 따라 ConfigMap을 생성합니다.

사전 요구 사항

- Service Mesh Operator 설치 및 검증.
- cluster-admin 역할이 있는 계정.
- Operator 배포 위치.
- OpenShift CLI(oc)에 액세스합니다.

절차

1. OpenShift Container Platform CLI에 클러스터 관리자로 로그인합니다.
2. CLI에서 이 명령을 실행하여 openshift-operators 프로젝트에서 smcp-templates라는 ConfigMap을 생성하고 <templates-directory>를 로컬 디스크의 ServiceMeshControlPlane 파일의 위치로 교체합니다.

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

3. Operator ClusterServiceVersion 이름을 찾습니다.

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

출력 예

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

4. Operator 클러스터 서비스 버전을 편집하여 Operator에서 smcp-templates ConfigMap을 사용하도록 지시합니다.

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

5. Operator 배포에 볼륨 마운트 및 볼륨을 추가합니다.

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

6. 변경 사항을 저장하고 편집기를 종료합니다.

7. 이제 ServiceMeshControlPlane에서 template 매개변수를 사용하여 템플릿을 지정할 수 있습니다.

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

2.8.3. 자동 사이드카 삽입 활성화

애플리케이션을 배포할 때 배포 오브젝트에서 **spec.template.metadata.annotations** 에서 **true**로 주석 **sidecar.istio.io/inject** 를 **true** 로 구성하여 삽입을 선택해야 합니다. 이 설정을 통해 사이드카 삽입이 **OpenShift Container Platform** 에코시스템 내 여러 프레임 워크에서 사용되는 **builder pod**와 같은 다른 **OpenShift Container Platform** 기능을 방해하지 않도록 할 수 있습니다.

사전 요구 사항

- 서비스 메시의 일부인 네임스페이스와 자동 사이드카 삽입이 필요한 배포를 식별합니다.

절차

1. 배포를 찾으려면 **oc get** 명령을 사용합니다.

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

예를 들어 **info** 네임스페이스에서 **'ratings-v1'** 마이크로 서비스에 대한 배포 파일을 보려면 다음 명령을 사용하여 **YAML** 형식의 리소스를 확인합니다.

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

2. 편집기에서 애플리케이션의 배포 구성 **YAML** 파일을 엽니다.

3. 다음 예와 같이 **spec.template.metadata.annotations.sidecar.istio/inject** 를 **Deployment YAML**에 추가하고 **sidecar.istio.io/inject** 를 **true** 로 설정합니다.

info deployment-ratings-v1.yaml의 스니펫 예

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ratings-v1
  namespace: info
  labels:
    app: ratings
    version: v1
spec:
  template:
```

```
  metadata:
    annotations:
      sidecar.istio.io/inject: 'true'
```

4. 배포 구성 파일을 저장합니다.

5. 앱이 포함된 프로젝트에 파일을 다시 추가합니다.

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

이 예제에서 **info** 는 **ratings-v1 app** 및 **deployment-ratings-v1.yaml** 이 포함된 프로젝트의 이름입니다.

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

6. 리소스가 업로드되었는지 확인하려면 다음 명령을 실행합니다.

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

예를 들면 다음과 같습니다.

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

2.8.4. 주석을 통해 프록시 환경 변수 설정

Envoy 사이드카 프록시에 대한 구성은 **ServiceMeshControlPlane** 에서 관리합니다.

injection-template.yaml 파일의 배포에 **Pod** 주석을 추가하여 애플리케이션의 사이드카 프록시의 환경 변수를 설정할 수 있습니다. 환경 변수가 사이드카에 삽입됩니다.

예: **injection-template.yaml**

```
apiVersion: apps/v1
```

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

주의

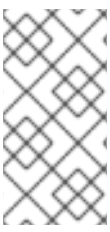
고유한 사용자 정의 리소스를 생성할 때 **maistra.io/** 레이블 및 주석을 포함하지 않아야 합니다. 이러한 라벨 및 주석은 **Operator**에서 리소스를 생성하고 관리함을 나타냅니다. 자체 리소스를 생성할 때 **Operator** 생성 리소스에서 콘텐츠를 복사하는 경우 **maistra.io/** 로 시작하는 레이블 또는 주석을 포함하지 마십시오. 이러한 라벨 또는 주석을 포함하는 리소스는 다음 조정 중에 **Operator**에 의해 덮어쓰거나 삭제됩니다.

2.8.5. Mixer 정책 시행 업데이트

이전 버전의 **Red Hat OpenShift Service Mesh**에서는 기본적으로 **Mixer**의 정책 시행이 활성화되었습니다. 이제 **Mixer** 정책 시행은 기본적으로 비활성화되었습니다. 정책 작업을 실행하기 전에 활성화해야 합니다.

사전 요구 사항

- **OpenShift CLI(oc)**에 액세스합니다.



참고

예에서는 **<istio-system>**을 컨트롤 플레인 네임스페이스로 사용합니다. 이 값을 **SMCP(Service Mesh Control Plane)**를 배포한 네임스페이스로 교체합니다.

프로세스

1. OpenShift Container Platform CLI에 로그인합니다.

2. 이 명령을 실행하여 현재 Mixer 정책 시행 상태를 확인합니다.

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

3. disablePolicyChecks: true인 경우 Service Mesh ConfigMap을 편집합니다.

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

4. ConfigMap 내에서 disablePolicyChecks: true를 찾고 값을 false로 변경합니다.

5. 구성을 저장하고 편집기를 종료합니다.

6. Mixer 정책 시행 상태를 다시 점검하여 false로 설정되어 있는지 확인합니다.

2.8.5.1. 올바른 네트워크 정책 설정

서비스 메시는 서비스 메시 컨트롤 플레인과 멤버 네임스페이스에서 네트워크 정책을 생성하여 트래픽을 허용합니다. 배포하기 전에 다음 조건을 고려하여 OpenShift Container Platform 경로를 통해 이전에 노출된 서비스 메시의 서비스를 확인하십시오.

- Istio가 제대로 작동하려면 서비스 메시로 들어오는 트래픽이 항상 Ingress-gateway를 통과해야 합니다.
- 서비스 메시에 없는 별도의 네임스페이스에서 서비스 메시 외부에 서비스를 배포합니다.
- 서비스 메시 등록 네임스페이스에 배포해야 하는 메시 외 서비스는 해당 배포 maistra.io/expose-route: "true"에 레이블을 지정하여 OpenShift Container Platform 경로가 여전히 작동하도록 해야 합니다.

2.8.6. Bookinfo 예제 애플리케이션

Bookinfo 예제 애플리케이션에서는 OpenShift Container Platform에서 Red Hat OpenShift Service Mesh 2.4.4 설치를 테스트할 수 있습니다.

Bookinfo 애플리케이션은 온라인 서점의 단일 카탈로그 항목과 유사하게 한 권의 책에 대한 정보를 표시합니다. 애플리케이션은 도서 설명, 도서 세부 정보(ISBN, 페이지 수, 기타 정보), 도서 리뷰가 설명된 페이지를 표시합니다.

Bookinfo 애플리케이션은 이러한 마이크로 서비스로 구성됩니다.

- productpage 마이크로 서비스는 details 및 reviews 마이크로 서비스를 호출하여 페이지를 채웁니다.
- details 마이크로 서비스에는 도서 정보가 포함되어 있습니다.
- reviews 마이크로 서비스에는 도서 리뷰가 포함되어 있습니다. 이를 ratings 마이크로 서비스라고도 합니다.
- ratings 마이크로 서비스에는 도서 리뷰와 함께 제공되는 도서 순위 정보가 포함되어 있습니다.

리뷰 마이크로 서비스의 세 가지 버전이 있습니다.

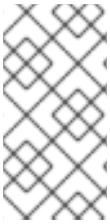
- 버전 v1에서는 ratings 서비스를 호출하지 않습니다.
- 버전 v2는 ratings 서비스를 호출하고 각 평가를 1~5개의 검정별로 표시합니다.
- 버전 v3은 ratings 서비스를 호출하고 각 평가를 1~5개의 빨강별로 표시합니다.

2.8.6.1. Bookinfo 애플리케이션 설치

이 튜토리얼에서는 프로젝트를 생성하고, Bookinfo 애플리케이션을 해당 프로젝트에 배포하고, 서비스 메시에서 실행 중인 애플리케이션을 확인하여 샘플 애플리케이션을 생성하는 방법을 안내합니다.

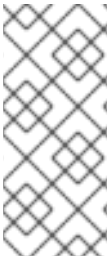
사전 요구 사항

- OpenShift Container Platform 4.1 이상이 설치되었습니다.
- Red Hat OpenShift Service Mesh 2.4.4가 설치되었습니다.
- OpenShift CLI(oc)에 액세스합니다.
- cluster-admin 역할이 있는 계정.



참고

Bookinfo 샘플 애플리케이션은 IBM Z 및 IBM Power Systems에 설치할 수 없습니다.



참고

이 섹션의 명령은 Service Mesh Control Plane 프로젝트가 istio-system 이라고 가정합니다. 컨트롤 플레인을 다른 네임스페이스에 설치한 경우 실행하기 전에 각 명령을 편집합니다.

프로세스

1. OpenShift Container Platform 웹 콘솔에 cluster-admin 권한이 있는 사용자로 로그인합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.
2. 홈 → 프로젝트를 클릭합니다.
3. 프로젝트 만들기를 클릭합니다.
4. Project Name 으로 info 를 입력하고 Display Name 을 입력하고 Description 을 입력한 다음 Create 를 클릭합니다.

- 또는 CLI에서 이 명령을 실행하여 info 프로젝트를 생성할 수도 있습니다.

```
$ oc new-project info
```

5. Operators → 설치된 Operators를 클릭합니다.

6. 프로젝트 메뉴를 클릭하고 Service Mesh Control Plane 네임스페이스를 사용합니다. 이 예제에서는 istio-system을 사용합니다.

7. Red Hat OpenShift Service Mesh Operator를 클릭합니다.

8. Istio Service Mesh 멤버 목록 탭을 클릭합니다.

   a. 이미 Istio Service Mesh 멤버 롤을 생성한 경우, 이름을 클릭한 다음 YAML 탭을 클릭하여 YAML 편집기를 엽니다.

   b. ServiceMeshMemberRoll을 생성하지 않은 경우 ServiceMeshMemberRoll 생성을 클릭합니다.

9. Members를 클릭한 다음 Value 필드에 프로젝트 이름을 입력합니다.

10. 생성을 클릭하여 업데이트된 서비스 메시 멤버 롤을 저장합니다.

    a. 또는 다음 예제를 YAML 파일에 저장합니다.

Bookinfo ServiceMeshMemberRoll example servicemeshmemberroll-default.yaml

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

b.

다음 명령을 실행하여 해당 파일을 업로드하고 **istio-system** 네임스페이스에 **ServiceMeshMemberRoll** 리소스를 만듭니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 프로젝트의 이름입니다.

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

11.

다음 명령을 실행하여 **ServiceMeshMemberRoll**이 성공적으로 생성되었는지 확인합니다.

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

STATUS 열이 **Configured**인 경우 설치가 성공적으로 완료된 것입니다.

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

12.

CLI에서 **bookinfo.yaml** 파일을 적용하여 ***'info'*** 프로젝트에 **Bookinfo** 애플리케이션을 배포합니다.

```
$ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.4/samples/bookinfo/platform/kube/bookinfo.yaml
```

출력은 다음과 유사합니다.

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

13. info-gateway.yaml 파일을 적용하여 수신 게이트웨이를 생성합니다.

```
$ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.4/samples/bookinfo/networking/bookinfo-gateway.yaml
```

출력은 다음과 유사합니다.

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

14. GATEWAY_URL 매개변수 값을 설정합니다.

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

2.8.6.2. 기본 대상 규칙 추가

Bookinfo 애플리케이션을 사용하기 전에 먼저 기본 대상 규칙을 추가해야 합니다. 상호 TLS(Transport layer security) 인증을 활성화했는지 여부에 따라 사전 구성된 YAML 파일이 두 개 있습니다.

프로세스

1. 대상 규칙을 추가하려면 다음 명령 중 하나를 실행합니다.

- 상호 TLS를 활성화하지 않은 경우:

```
$ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.4/samples/bookinfo/networking/destination-rule-all.yaml
```

- 상호 TLS를 활성화한 경우:

```
$ oc apply -n info -f https://raw.githubusercontent.com/Maistra/istio/maistra-2.4/samples/bookinfo/networking/destination-rule-all-mtls.yaml
```

출력은 다음과 유사합니다.

```
destinationrule.networking.istio.io/productpage created
```

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

2.8.6.3. Bookinfo 설치 확인

샘플 Bookinfo 애플리케이션이 성공적으로 배포되었는지 확인하려면 다음 단계를 수행합니다.

사전 요구 사항

- Red Hat OpenShift Service Mesh가 설치되어 있어야 합니다.
- Bookinfo 샘플 애플리케이션을 설치하는 단계를 완료합니다.

CLI의 프로시저

1. OpenShift Container Platform CLI에 로그인합니다.
2. 다음 명령으로 모든 pod가 준비되었는지 확인합니다.

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

모든 pod의 상태는 Running이어야 합니다. 출력은 다음과 유사합니다.

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

3. 다음 명령을 실행하여 제품 페이지의 URL을 검색합니다.

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

4. 웹 브라우저에 출력을 복사하여 붙여넣어 Bookinfo 제품 페이지가 배포되었는지 확인합니다.

Kiali 웹 콘솔의 절차

1. **Kiali 웹 콘솔의 주소를 가져옵니다.**

   a. **OpenShift Container Platform 웹 콘솔에 cluster-admin 권한이 있는 사용자로 로그인합니다. Red Hat OpenShift Dedicated를 사용하는 경우 dedicated-admin 역할의 계정이 있어야 합니다.**

   b. **네트워킹 → 경로로 이동합니다.**

   c. **경로 페이지의 네임스페이스 메뉴에서 Service Mesh Control Plane 프로젝트(예: istio-system)를 선택합니다.**

      위치 열은 각 경로에 대한 연결된 주소를 표시합니다.

   d. **Kiali의 위치 열에서 링크를 클릭합니다.**

   e. **OpenShift로 로그인 을 클릭합니다. Kiali 개요 화면에 각 프로젝트 네임스페이스에 대한 타일이 표시됩니다.**

2. **Kiali에서 그래프 를 클릭합니다.**

3. **네임스페이스 목록에서 info를 선택하고 그래프 유형 목록에서 앱 그래프 를 선택합니다.**

4. **디스플레이 메뉴에서 유휴 노드 표시를 클릭합니다.**

이렇게 하면 정의되었지만 요청 수신 또는 전송되지 않은 노드가 표시됩니다. 애플리케이션이 올바르게 정의되었지만 요청 트래픽이 보고되지 않았는지 확인할 수 있습니다.

- **Duration** 메뉴를 사용하여 오래된 트래픽이 캡처되도록 기간을 늘립니다.
- **Refresh Rate** 메뉴를 사용하여 트래픽을 더 자주 새로 고치거나 전혀 새로 고침하지 않습니다.

5. 서비스,워크로드 또는 **Istio Config** 를 클릭하여 정보 구성 요소의 목록 보기를 확인하고 정상인지 확인합니다.

2.8.6.4. Bookinfo 애플리케이션 제거

다음 단계에 따라 **Bookinfo** 애플리케이션을 제거하십시오.

사전 요구 사항

- **OpenShift Container Platform 4.1** 이상이 설치되었습니다.
- **Red Hat OpenShift Service Mesh 2.4.4**가 설치되었습니다.
- **OpenShift CLI(oc)**에 액세스합니다.

2.8.6.4.1. Bookinfo 프로젝트 삭제

절차

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. 홈 → 프로젝트를 클릭합니다.
3. 정보 메뉴

를 클릭한 다음 프로젝트 삭제 를 클릭합니다.

4. 확인 대화 상자에 **info** 를 입력한 다음 삭제 를 클릭합니다.

- 또는 **CLI**를 사용하여 이 명령을 실행하여 **info** 프로젝트를 생성할 수도 있습니다.

```
$ oc delete project info
```

2.8.6.4.2. 서비스 메시 멤버 롤에서 Bookinfo 프로젝트를 제거

절차

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.

2. **Operators** → 설치된 **Operators**를 클릭합니다.

3. 프로젝트 메뉴를 클릭하고 목록에서 **istio-system** 을 선택합니다.

4. **Red Hat OpenShift Service Mesh Operator**에 대해 제공된 **APIS**에서 **Istio Service Mesh** 멤버 롤 링크를 클릭합니다.

5. **ServiceMeshMemberRoll** 메뉴

를 클릭하고 서비스 메시 멤버 롤 편집 을 선택합니다.

6. 기본 서비스 메시 멤버 롤 **YAML**을 편집하고 멤버 목록에서 정보를 제거합니다.

- 또는 **CLI**를 사용하여 이 명령을 실행하여 **ServiceMeshMemberRoll** 에서 **info** 프로젝트를 제거할 수 있습니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 프로젝트의 이름입니다.

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

7. 저장을 클릭하여 서비스 메시 멤버 롤을 업데이트합니다.

2.8.7. 예제 추적 생성 및 추적 데이터 분석

Jaeger는 오픈 소스 분산 추적 시스템입니다. **Jaeger**를 사용하면 애플리케이션을 구성하는 다양한 마이크로 서비스를 통해 요청의 경로를 따라 추적할 수 있습니다. **Jaeger**는 기본적으로 서비스 메시의 일부로 설치됩니다.

이 튜토리얼에서는 서비스 메시와 **Bookinfo** 샘플 애플리케이션을 사용하여 **Jaeger**로 분산 추적을 수행하는 방법을 보여줍니다.

사전 요구 사항

- **OpenShift Container Platform 4.1** 이상이 설치되었습니다.
- **Red Hat OpenShift Service Mesh 2.4.4**가 설치되었습니다.
- 설치 중에 **Jaeger**가 활성화되었습니다.
- **Bookinfo** 예제 애플리케이션이 설치되었습니다.

절차

1. **Bookinfo** 샘플 애플리케이션을 설치한 후 트래픽을 메시로 보냅니다. 다음 명령을 여러 번 입력합니다.

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

이 명령은 애플리케이션의 **productpage** 마이크로 서비스에 액세스하는 사용자를 시뮬레이션합니다.

2. **OpenShift Container Platform** 콘솔에서 네트워킹 → 경로로 이동하여 위치에 나열된 **URL**인 **Jaeger** 경로를 검색합니다.
   - 다른 방법으로 **CLI**를 사용하여 경로에 대한 세부 정보를 쿼리합니다. 이 예제에서 **istio-system** 은 **Service Mesh Control Plane** 네임스페이스입니다.

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

a. 다음 명령을 입력하여 **Jaeger** 콘솔의 **URL**을 표시합니다. 결과를 브라우저에 붙여 넣고 해당 **URL**로 이동합니다.

```
echo $JAEGER_URL
```

3. **OpenShift Container Platform** 콘솔에 액세스하는 데 사용하는 것과 동일한 사용자 이름 및 암호를 사용하여 로그인합니다.

4. 서비스 메뉴에서 **Jaeger** 대시보드 왼쪽 창에 있는 **productpage.info** 를 선택하고 창 하단에 있는 추적 찾기를 클릭합니다. 추적 목록이 표시됩니다.

5. 목록의 추적 중 하나를 클릭하여 해당 추적에 대한 상세 보기를 엽니다. 목록의 첫 번째 (가장 최근) 추적을 클릭하면 **/productpage**의 최신 새로 고침에 해당하는 세부 사항이 표시됩니다.

2.9. 데이터 시각화 및 관찰 기능

주의

더 이상 지원되지 않는 **Red Hat OpenShift Service Mesh** 릴리스에 대한 문서를 보고 있습니다.

서비스 메시 버전 **1.0** 및 **1.1** 컨트롤 플레인은 더 이상 지원되지 않습니다. 서비스 메시 컨트롤 플레인 업그레이드에 대한 자세한 내용은 서비스 메시 업그레이드를 참조하십시오.

특정 **Red Hat OpenShift Service Mesh** 릴리스의 지원 상태에 대한 자세한 내용은 제품 라이프사이클 페이지를 참조하십시오.

Kiali 콘솔에서 애플리케이션의 토폴로지, 상태 및 지표를 볼 수 있습니다. 서비스에 문제가 있는 경우 **Kiali** 콘솔은 서비스를 통해 데이터 흐름을 시각화하는 방법을 제공합니다. 추상 애플리케이션, 서비스 및

워크로드를 포함하여 다양한 수준에서 메시 구성 요소에 대한 인사이트를 볼 수 있습니다. 또한 네임스페이스에 대한 대화형 그래프 보기도 실시간으로 제공합니다.

시작하기 전

애플리케이션이 설치된 경우 애플리케이션을 통한 데이터 흐름을 확인할 수 있습니다. 자체 애플리케이션이 설치되어 있지 않은 경우 Bookinfo 샘플 애플리케이션을 설치하여 Red Hat OpenShift Service Mesh에서 관찰 기능이 작동하는 방식을 확인할 수 있습니다.

2.9.1. 서비스 메시 데이터 보기

Kiali Operator는 Red Hat OpenShift Service Mesh에서 수집된 Telemetry 데이터를 사용하여 네임스페이스의 애플리케이션, 서비스 및 워크로드에 대한 그래프와 실시간 네트워크 다이어그램을 제공합니다.

Kiali 콘솔에 액세스하려면 Red Hat OpenShift Service Mesh가 설치되어 있고 서비스 메시에 대해 구성된 프로젝트가 있어야 합니다.

절차

1. 관점 전환 기능을 사용하여 관리자 관점으로 전환합니다.
2. 홈 → 프로젝트를 클릭합니다.
3. 프로젝트 이름을 클릭합니다. 예를 들어 info 를 클릭합니다.
4. 시작 도구 섹션에서 Kiali 를 클릭합니다.
5. OpenShift Container Platform 콘솔에 액세스하는 데 사용하는 것과 동일한 사용자 이름 및 암호를 사용하여 Kiali 콘솔에 로그인합니다.

Kiali 콘솔에 처음 로그인하면 볼 수 있는 권한이 있는 서비스 메시에 모든 네임스페이스가 표시되는 개요 페이지가 표시됩니다.

콘솔 설치를 확인하는 경우 표시할 데이터가 없을 수 있습니다.

2.9.2. Kiali 콘솔에서 서비스 메시 데이터 보기

Kiali Graph는 메시 트래픽의 강력한 시각화를 제공합니다. 토폴로지는 실시간 요청 트래픽을 **Istio** 구성 정보와 결합하여 서비스 메시의 동작에 대한 즉각적인 통찰력을 제공하므로 문제를 신속하게 파악할 수 있습니다. 여러 그래프 유형을 사용하면 높은 수준의 서비스 토폴로지, 하위 수준 워크로드 토폴로지 또는 애플리케이션 수준 토폴로지로 트래픽을 시각화할 수 있습니다.

몇 가지의 그래프를 선택할 수 있습니다.

- 앱 그래프는 동일한 레이블이 있는 애플리케이션에 대한 집계 워크로드를 보여줍니다.
- 서비스 그래프는 메시의 각 서비스에 대한 노드를 표시되지만 그래프에서 모든 애플리케이션과 워크로드는 제외됩니다. 높은 수준의 보기를 제공하며 정의된 서비스에 대한 모든 트래픽을 집계합니다.
- 버전이 지정된 앱 그래프는 애플리케이션의 각 버전에 대한 노드를 보여줍니다. 모든 애플리케이션 버전이 함께 그룹화됩니다.
- 워크로드 그래프는 서비스 메시의 각 워크로드에 대한 노드를 표시합니다. 이 그래프는 애플리케이션 및 버전 레이블을 사용할 필요가 없습니다. 애플리케이션에서 버전 레이블을 사용하지 않는 경우 이 그래프를 사용하십시오.

그래프 노드는 다양한 정보로 장식되어 가상 서비스 및 서비스 항목과 같은 다양한 경로 라우팅 옵션과 오류 삽입 및 회로 차단기와 같은 특수 구성을 가리킵니다. **mTLS** 문제, 대기 시간 문제, 오류 트래픽 등을 확인할 수 있습니다. 그래프는 구성 가능하며, 트래픽 애니메이션을 표시할 수 있으며 강력한 찾기 및 숨기기 기능을 제공합니다.

범례 버튼을 클릭하여 그래프에 표시되는 모양, 색상, 화살표 및 배지에 대한 정보를 봅니다.

지표 요약을 보려면 그래프에서 노드 또는 에지를 선택하여 요약 세부 정보 패널에 지표 세부 정보를 표시합니다.

2.9.2.1. Kiali에서 그래프 레이아웃 변경

Kiali 그래프의 레이아웃은 애플리케이션 아키텍처 및 표시할 데이터에 따라 다르게 렌더링될 수 있습니다. 예를 들어 그래프 노드 수와 상호 작용은 **Kiali** 그래프가 렌더링되는 방법을 결정할 수 있습니다. 모

든 상황에 적합하게 렌더링되는 단일 레이아웃을 생성할 수 없기 때문에 Kiali는 다양한 레이아웃을 선택할 수 있습니다.

사전 요구 사항

- 자체 애플리케이션이 설치되어 있지 않은 경우 Bookinfo 샘플 애플리케이션을 설치합니다. 그런 다음 다음 명령을 여러 번 입력하여 Bookinfo 애플리케이션에 대한 트래픽을 생성합니다.

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

이 명령은 애플리케이션의 productpage 마이크로 서비스에 액세스하는 사용자를 시뮬레이션합니다.

절차

1. Kiali 콘솔을 시작합니다.
2. OpenShift로 로그인 을 클릭합니다.
3. Kiali 콘솔에서 그래프 를 클릭하여 네임스페이스 그래프를 확인합니다.
4. 네임스페이스 메뉴에서 애플리케이션 네임스페이스를 선택합니다(예: info).
5. 다른 그래프 레이아웃을 선택하려면 다음 중 하나 또는 둘 다 수행합니다.
   - 그래프 상단에 있는 메뉴에서 다른 그래프 데이터 그룹화를 선택합니다.
     - 앱 그래프
     - 서비스 그래프
     - 버전이 지정된 앱 그래프(기본값)

- 워크로드 그래프

- 그래프 하단의 범인에서 다른 그래프 레이아웃을 선택합니다.
  - 레이아웃 기본 **dagre**
  - 레이아웃 **1** 공동 기능
  - 레이아웃 **2 cola**

2.10. 사용자 정의 리소스

주의

더 이상 지원되지 않는 **Red Hat OpenShift Service Mesh** 릴리스에 대한 문서를 보고 있습니다.

서비스 메시 버전 **1.0** 및 **1.1** 컨트롤 플레인은 더 이상 지원되지 않습니다. 서비스 메시 컨트롤 플레인 업그레이드에 대한 자세한 내용은 서비스 메시 업그레이드를 참조하십시오.

특정 **Red Hat OpenShift Service Mesh** 릴리스의 지원 상태에 대한 자세한 내용은 제품 라이프사이클 페이지를 참조하십시오.

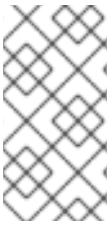
기본 서비스 메시 사용자 정의 리소스를 수정하거나 새 사용자 정의 리소스를 생성하여 **Red Hat OpenShift Service Mesh**를 사용자 지정할 수 있습니다.

2.10.1. 사전 요구 사항

- **cluster-admin** 역할이 있는 계정.

- Red Hat OpenShift Service Mesh 설치 준비 프로세스 완료.
- Operator 설치.

2.10.2. Red Hat OpenShift Service Mesh 사용자 정의 리소스



참고

istio-system 프로젝트는 서비스 메시 문서 전체에서 예제로 사용되지만, 필요에 따라 다른 프로젝트를 사용할 수 있습니다.

*사용자 지정 리소스*를 사용하여 Red Hat OpenShift Service Mesh 프로젝트 또는 클러스터에서 API를 확장할 수 있습니다. 서비스 메시를 배포할 때 프로젝트 매개변수를 변경하기 위해 수정할 수 있는 기본 ServiceMeshControlPlane을 생성합니다.

Service Mesh Operator는 ServiceMeshControlPlane 리소스 유형을 추가하여 API를 확장하며, 이를 통해 프로젝트 내에서 ServiceMeshControlPlane 오브젝트를 생성할 수 있습니다. ServiceMeshControlPlane 오브젝트를 생성하여 Operator에 ServiceMeshControlPlane 오브젝트에 설정한 매개변수로 구성된 Service Mesh Control Plane을 프로젝트에 설치하도록 지시합니다.

이 예제 ServiceMeshControlPlane 정의에는 지원되는 모든 매개변수가 포함되어 있으며 RHEL(Red Hat Enterprise Linux)을 기반으로 하는 Red Hat OpenShift Service Mesh 1.1.18.2 이미지를 배포합니다.

중요

3scale Istio Adapter는 사용자 정의 리소스 파일에 배포 및 구성됩니다. 또한 작동 중인 3scale 계정(SaaS 또는 On-Premises)이 필요합니다.

예: istio-installation.yaml

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

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

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

  mixer:
    policy:
      autoscaleEnabled: false

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

  pilot:
    autoscaleEnabled: false
    traceSampling: 100

  kiali:
    enabled: true

  grafana:
    enabled: true

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

2.10.3. ServiceMeshControlPlane 매개변수

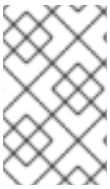
다음 예제에서는 **ServiceMeshControlPlane** 매개변수의 사용을 보여주고, 표에서는 지원되는 매개변수에 대한 추가 정보를 제공합니다.

중요

CPU, 메모리 및 Pod 수를 포함하여 이러한 매개변수를 사용하여 Red Hat OpenShift Service Mesh에 대해 구성하는 리소스는 OpenShift Container Platform 클러스터 구성을 기반으로 합니다. 현재 클러스터 구성의 사용 가능한 리소스에 따라 이러한 매개변수를 구성합니다.

2.10.3.1. Istio 글로벌 예

다음 예제는 ServiceMeshControlPlane의 Istio 전역 매개변수와 적절한 값과 함께 사용 가능한 매개변수에 대한 설명을 보여줍니다.



참고

3scale Istio Adapter가 작동하려면 disablePolicyChecks가 false여야 합니다.

전역 매개변수 예

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

표 2.4. 전역 매개변수

매개변수	설명	값	기본값
disablePolicyChecks	이 매개변수는 정책 검사를 활성화/비활성화합니다.	**true/false**	**true**
policyCheckFailOpen	이 매개변수는 Mixer 정책 서비스에 도달할 수 없는 경우 트래픽이 Envoy 사이드카를 통과할 수 있는지 여부를 나타냅니다.	**true/false**	**false**
tag	Operator가 Istio 이미지를 가져오는 데 사용하는 태그입니다.	유효한 컨테이너 이미지 태그.	**1.1.0**
hub	Operator가 Istio 이미지를 가져오는 데 사용하는 허브입니다.	유효한 이미지 리포지토리	**maistra/** 또는 **registry.redhat.io/openshift-service-mesh/**
mtls	이 매개변수는 기본적으로 서비스 간에 mTLS(mutual Transport Layer Security)를 활성화/비활성화할지 여부를 제어합니다.	**true/false**	**false**
imagePullSecrets	Istio 이미지를 제공하는 레지스트리에 대한 액세스가 안전한 경우, 여기에 imagePullSecret을 나열하십시오.	redhat-registry-pullsecret 또는 quay-pullsecret	없음

이러한 매개 변수는 전역 매개변수의 프록시 하위 집합에 따라 다릅니다.

표 2.5. 프록시 매개변수

유형	매개변수	설명	값	기본값
requests	**cpu**	Envoy 프록시에 대해 요청된 CPU 리소스의 양입니다.	사용자 환경 구성에 따라 코어 또는 밀리코어(예: 200m, 0.5, 1)로 지정된 CPU 리소스입니다.	**10m**

유형	매개변수	설명	값	기본값
	memory	Envoy 프록시에 대해 요청된 메모리 양입니다.	사용자 환경 구성에 따라 사용 가능한 바이트 단위 메모리(예: 200Ki, 50Mi, 5Gi)입니다.	**128Mi**
limits	**cpu**	Envoy 프록시에 대해 요청된 최대 CPU 리소스 양입니다.	사용자 환경 구성에 따라 코어 또는 밀리코어(예: 200m, 0.5, 1)로 지정된 CPU 리소스입니다.	**2000m**
	memory	Envoy 프록시가 사용할 수 있는 최대 메모리 양입니다.	사용자 환경 구성에 따라 사용 가능한 바이트 단위 메모리(예: 200Ki, 50Mi, 5Gi)입니다.	**1024Mi**

2.10.3.2. Istio 게이트웨이 구성

다음 예제는 **ServiceMeshControlPlane**의 **Istio** 게이트웨이 매개변수와 적절한 값과 함께 사용 가능한 매개변수에 대한 설명을 보여줍니다.

게이트웨이 매개변수 예

```
gateways:
  egress:
    enabled: true
    runtime:
      deployment:
        autoScaling:
          enabled: true
          maxReplicas: 5
          minReplicas: 1
  enabled: true
  ingress:
    enabled: true
    runtime:
      deployment:
        autoScaling:
          enabled: true
          maxReplicas: 5
          minReplicas: 1
```

-

표 2.6. Istio 게이트웨이 매개변수

매개변수	설명	값	기본값
gateways.egress.runtime.deployment.autoScaling.enabled	이 매개변수는 자동 스케일링을 활성화/비활성화 합니다.	**true/false**	**true**
gateways.egress.runtime.deployment.autoScaling.minReplicas	**autoscaleEnabled** 설정을 기반으로 송신 게이트웨이에 배포할 최소 pod 수입니다.	사용자 환경 구성에 따라 할당 가능한 유효한 Pod 수입니다.	**1**
gateways.egress.runtime.deployment.autoScaling.maxReplicas	**autoscaleEnabled** 설정을 기반으로 송신 게이트웨이에 배포할 최대 pod 수입니다.	사용자 환경 구성에 따라 할당 가능한 유효한 Pod 수입니다.	**5**
gateways.ingress.runtime.deployment.autoScaling.enabled	이 매개변수는 자동 스케일링을 활성화/비활성화 합니다.	**true/false**	**true**
gateways.ingress.runtime.deployment.autoScaling.minReplicas	**autoscaleEnabled** 설정을 기반으로 수신 게이트웨이에 배포할 최소 pod 수입니다.	사용자 환경 구성에 따라 할당 가능한 유효한 Pod 수입니다.	**1**
gateways.ingress.runtime.deployment.autoScaling.maxReplicas	**autoscaleEnabled** 설정을 기반으로 수신 게이트웨이에 배포할 최대 pod 수입니다.	사용자 환경 구성에 따라 할당 가능한 유효한 Pod 수입니다.	**5**

클러스터 관리자는 하위 도메인을 활성화하는 방법에 대한 지침은 와일드카드 경로 사용을 참조할 수 있습니다.

2.10.3.3. Istio Mixer 구성

다음 예제는 ServiceMeshControlPlane의 Mixer 매개변수와 적절한 값과 함께 사용 가능한 매개변수에 대한 설명을 보여줍니다.

Mixer 매개변수 예

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

표 2.7. Istio Mixer 정책 매개변수

매개변수	설명	값	기본값
enabled	이 매개변수는 Mixer를 활성화/비활성화합니다.	**true/false**	**true**
autoscaleEnabled	이 매개변수는 자동 스케일링을 활성화/비활성화합니다. 작은 환경에서는 이 값을 비활성화합니다.	**true/false**	**true**
autoscaleMin	**autoscaleEnabled** 설정을 기반으로 배포할 최소 pod 수입니다.	사용자 환경 구성에 따라 할당 가능한 유효한 Pod 수입니다.	**1**
autoscaleMax	**autoscaleEnabled** 설정을 기반으로 배포할 최대 pod 수입니다.	사용자 환경 구성에 따라 할당 가능한 유효한 Pod 수입니다.	**5**

표 2.8. Istio Mixer Telemetry 매개변수

유형	매개변수	설명	값	기본값
requests	**cpu**	Mixer Telemetry에 요청된 CPU 리소스의 백분율입니다.	사용자 환경 구성을 기반으로 하는 밀리코어 단위의 CPU 리소스입니다.	**10m**
	memory	Mixer Telemetry에 요청된 메모리 양입니다.	사용자 환경 구성에 따라 사용 가능한 바이트 단위 메모리(예: 200Ki, 50Mi, 5Gi)입니다.	**128Mi**

유형	매개변수	설명	값	기본값
limits	**cpu**	Mixer telemetry가 사용할 수 있는 CPU 리소스의 최대 백분율입니다.	사용자 환경 구성을 기반으로 하는 밀리코어 단위의 CPU 리소스입니다.	**4800m**
	memory	Mixer telemetry가 사용할 수 있는 메모리 최대 크기입니다.	사용자 환경 구성에 따라 사용 가능한 바이트 단위 메모리(예: 200Ki, 50Mi, 5Gi)입니다.	**4G**

2.10.3.4. Istio Pilot 구성

리소스 할당에 대한 일정 또는 제한을 설정하도록 **Pilot**을 구성할 수 있습니다. 다음 예제는 **ServiceMeshControlPlane**의 **Pilot** 매개변수와 적절한 값과 함께 사용 가능한 매개변수에 대한 설명을 보여줍니다.

pilot 매개변수 예

```
spec:
  runtime:
    components:
      pilot:
        deployment:
          autoScaling:
            enabled: true
            minReplicas: 1
            maxReplicas: 5
            targetCPUUtilizationPercentage: 85
        pod:
          tolerations:
          - key: node.kubernetes.io/unreachable
            operator: Exists
            effect: NoExecute
            tolerationSeconds: 60
          affinity:
            podAntiAffinity:
              requiredDuringScheduling:
              - key: istio
                topologyKey: kubernetes.io/hostname
                operator: In
                values:
                - pilot
        container:
          resources:
```

```
      limits:
        cpu: 100m
        memory: 128M
```

표 2.9. Istio Pilot 매개변수

매개변수	설명	값	기본값
cpu	Pilot에 요청된 CPU 리소스의 백분율입니다.	사용자 환경 구성을 기반으로 하는 밀리코어 단위의 CPU 리소스입니다.	**10m**
memory	Pilot에 대해 요청된 메모리 양입니다.	사용자 환경 구성에 따라 사용 가능한 바이트 단위 메모리(예: 200Ki, 50Mi, 5Gi)입니다.	**128Mi**
autoscaleEnabled	이 매개변수는 자동 스케일링을 활성화/비활성화합니다. 작은 환경에서는 이 값을 비활성화합니다.	**true/false**	**true**
traceSampling	이 값은 임의의 샘플링이 발생하는 빈도를 제어합니다. **참고:** 개발 또는 테스트를 할 때는 늘리십시오.	유효한 백분율입니다.	**1.0**

2.10.4. Kiali 구성

Service Mesh Operator에서 ServiceMeshControlPlane을 생성할 때 Kiali 리소스도 처리합니다. 그런 다음 Kiali Operator는 Kiali 인스턴스를 생성할 때 이 오브젝트를 사용합니다.

ServiceMeshControlPlane에 지정된 기본 Kiali 매개변수는 다음과 같습니다.

Kiali 매개변수 예

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
   kiali:
     enabled: true
```

```
    dashboard:
      viewOnlyMode: false
    ingress:
      enabled: true
```

표 2.10. Kiali 매개변수

매개변수	설명	값	기본값
enabled	이 매개변수는 Kiali를 활성화/비활성화합니다. Kiali는 기본적으로 활성화되어 있습니다.	**true/false**	**true**
dashboard viewOnlyMode	이 매개변수는 Kiali 콘솔에 대해 보기 전용 모드를 활성화/비활성화합니다. 보기 전용 모드가 활성화되면 콘솔을 사용하여 서비스 메시를 변경할 수 없습니다.	**true/false**	**false**
ingress enabled	이 매개변수는 Kiali에 대해 수신을 활성화/비활성화합니다.	**true/false**	**true**

2.10.4.1. Grafana에 대한 Kiali 설정

Kiali 및 Grafana를 Red Hat OpenShift Service Mesh의 일부로 설치할 때 Operator는 기본적으로 다음을 구성합니다.

- **Grafana가 Kiali의 외부 서비스로 활성화됨**
- **Kiali 콘솔에 대한 Grafana 인증**
- **Kiali 콘솔의 Grafana URL**

Kiali는 Grafana URL을 자동으로 감지할 수 있습니다. 그러나 Kiali에서 쉽게 자동 감지할 수 없는 사용자 지정 Grafana 설치가 있는 경우 ServiceMeshControlPlane 리소스에서 URL 값을 업데이트해야 합니다.

추가 Grafana 매개변수

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

2.10.4.2. Jaeger에 대한 Kiali 설정

Kiali 및 Jaeger를 Red Hat OpenShift Service Mesh의 일부로 설치할 때 Operator는 기본적으로 다음을 구성합니다.

- Jaeger가 Kiali의 외부 서비스로 활성화됨
- Kiali 콘솔에 대한 Jaeger 인증
- Kiali 콘솔의 Jaeger URL

Kiali는 Jaeger URL을 자동으로 감지할 수 있습니다. 그러나 Kiali에서 쉽게 자동 감지할 수 없는 사용자 지정 Jaeger 설치가 있는 경우 ServiceMeshControlPlane 리소스에서 URL 값을 업데이트해야 합니다.

추가 Jaeger 매개변수

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

2.10.5. Jaeger 구성

Service Mesh Operator가 ServiceMeshControlPlane 리소스를 생성할 때 분산 추적을 위한 리소스도 생성할 수 있습니다. 서비스 메시는 분산 추적을 위해 Jaeger를 사용합니다.

다음 두 가지 방법 중 하나로 Jaeger 설정을 지정할 수 있습니다.

- ServiceMeshControlPlane 리소스에서 Jaeger를 구성합니다. 이 방법에는 몇 가지 제한 사항이 있습니다.
- 사용자 지정 Jaeger 리소스에서 Jaeger를 구성한 다음 ServiceMeshControlPlane 리소스에서 Jaeger 인스턴스를 참조합니다. name 값과 일치하는 Jaeger 리소스가 있으면 컨트롤 플레인에서 기존 설치를 사용합니다. 이 방법을 사용하면 Jaeger 설정을 완전히 사용자 지정할 수 있습니다.

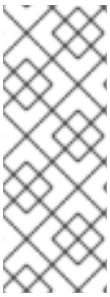
ServiceMeshControlPlane에 지정된 기본 Jaeger 매개변수는 다음과 같습니다.

기본 all-in-one Jaeger 매개변수

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
spec:
  version: v1.1
  istio:
    tracing:
      enabled: true
      jaeger:
        template: all-in-one
```

표 2.11. Jaeger 매개변수

매개변수	설명	값	기본값
tracing: enabled:	이 매개변수는 Service Mesh Operator에 의한 설치 및 추적을 활성화/비활성화합니다. Jaeger 설치는 기본적으로 활성화되어 있습니다. 기존 Jaeger 배포를 사용하려면 이 값을 **false**로 설정합니다.	**true/false**	**true**
jaeger: template:	이 매개변수는 사용할 Jaeger 배포 전략을 지정합니다.	• **all-in-one** - 개발, 테스트, 시연, 개념 증명용. • **production-elasticsearch** - 프로덕션용.	**all-in-one**



참고

ServiceMeshControlPlane 리소스의 기본 템플릿은 메모리 내 스토리지를 사용하는 all-in-one 배포 전략입니다. 프로덕션의 경우 지원되는 유일한 스토리지 옵션은 Elasticsearch이므로, 프로덕션 환경에서 서비스 메시를 배포할 때 production-elasticsearch 템플릿을 요청하도록 ServiceMeshControlPlane을 구성해야 합니다.

2.10.5.1. Elasticsearch 구성

기본 Jaeger 배포 전략에서는 최소한의 리소스를 사용하여 설치를 완료할 수 있도록 all-in-one 템플릿을 사용합니다. 하지만 all-in-one 템플릿은 메모리 내 스토리지를 사용하므로 개발, 데모 또는 테스트 목적으로만 권장되며 프로덕션 환경에 사용해서는 안 됩니다.

프로덕션 환경에서 서비스 메시와 Jaeger를 배포하는 경우, 템플릿을 Jaeger의 스토리지 요건에 Elasticsearch를 사용하는 production-elasticsearch 템플릿으로 변경해야 합니다.

Elasticsearch는 메모리를 많이 사용하는 애플리케이션입니다. 기본 OpenShift Container Platform 설치에 지정된 초기 노드 세트는 Elasticsearch 클러스터를 지원하기에 충분히 크지 않을 수 있습니다. 사용 사례와 OpenShift Container Platform 설치에 요청한 리소스가 일치하도록 기본 Elasticsearch 구성을 수정해야 합니다. 유효한 CPU 및 메모리값으로 리소스 블록을 수정하여 각 구성 요소의 CPU 및 메모리 제한을 모두 조정할 수 있습니다. 권장 메모리 양, 또는 그 이상으로 실행하려는 경우 클러스터에 추가 노드를 추가해야 합니다. OpenShift Container Platform 설치에 요청된 리소스를 초과하지 않는지 확인합니다.

Elasticsearch를 사용하는 기본 "production" Jaeger 매개변수

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

표 2.12. Elasticsearch 매개변수

매개변수	설명	값	기본값	예제
tracing: enabled:	이 매개변수는 서비스 메시에서 추적을 활성화/비활성화합니다. Jaeger는 기본적으로 설치되어 있습니다.	**true/false**	**true**	
ingress: enabled:	이 매개변수는 Jaeger에 대해 수신을 활성화/비활성화합니다.	**true/false**	**true**	
jaeger: template:	이 매개변수는 사용할 Jaeger 배포 전략을 지정합니다.	**all-in-one/production-elasticsearch**	**all-in-one**	
elasticsearch: nodeCount:	생성할 Elasticsearch 노드 수입니다.	정수 값.	1	개념 증명 = 1, 최소 배포 = 3

매개변수	설명	값	기본값	예제
requests: cpu:	사용자 환경 구성에 따른 요청에 대한 중앙 처리 단위 수입니다.	코어 또는 밀리코어(예: 200m, 0.5, 1)에 지정되어 있습니다.	1Gi	개념 증명 = 500m, 최소 배포 = 1
requests: memory:	환경 구성에 따른 요청에 사용 가능한 메모리입니다.	바이트로 지정됩니다(예: 200Ki, 50Mi, 5Gi).	500m	개념 증명 = 1Gi, 최소 배포 = 16Gi*
limits: cpu:	사용자 환경 구성에 따른 중앙 처리 장치 수에 대한 제한입니다.	코어 또는 밀리코어(예: 200m, 0.5, 1)에 지정되어 있습니다.		개념 증명 = 500m, 최소 배포 = 1
limits: memory:	사용자 환경 구성에 따라 사용 가능한 메모리 제한입니다.	바이트로 지정됩니다(예: 200Ki, 50Mi, 5Gi).		개념 증명 = 1Gi, 최소 배포 = 16Gi*
	각 Elasticsearch 노드는 더 낮은 메모리 설정으로 작동할 수 있지만 프로덕션 배포에는 권장되지 **않습니다**. 프로덕션 용도의 경우 기본적으로 각 Pod에 할당된 16Gi 미만이 있어야 하지만 Pod당 최대 64Gi까지 할당할 수도 있습니다.			

프로세스

1. **cluster-admin 역할의 사용자로 OpenShift Container Platform 웹 콘솔에 로그인합니다.**
2. **Operators → 설치된 Operator로 이동합니다.**
3. **Red Hat OpenShift Service Mesh Operator를 클릭합니다.**
4. **Istio Service Mesh Control Plane 탭을 클릭합니다.**
5. **컨트롤 플레인 파일의 이름을 클릭합니다(예: basic-install).**
6. **YAML 탭을 클릭합니다.**
7. **Jaeger 매개변수를 편집하여, 기본 all-in-one 템플릿을 사용 사례에 맞게 수정한**

production-elasticsearch 템플릿의 매개변수로 바꿉니다. 들여쓰기가 올바른지 확인합니다.

8. 저장을 클릭합니다.

9. 새로 고침을 클릭합니다. **OpenShift Container Platform**은 **Jaeger**를 재배포하고 지정된 매개변수를 기반으로 **Elasticsearch** 리소스를 생성합니다.

2.10.5.2. 기존 Jaeger 인스턴스에 연결

SMCP가 기존 **Jaeger** 인스턴스에 연결하려면 다음이 **true**여야 합니다.

- **Jaeger** 인스턴스는 컨트롤 플레인과 동일한 네임스페이스에 배포됩니다(예: **istio-system** 네임스페이스).
- 서비스 간에 보안 통신을 활성화하려면, **Jaeger** 인스턴스에 대한 통신을 보호하는 **oauth-proxy**를 활성화하고 **Kiali**와 통신할 수 있도록 시크릿이 **Jaeger** 인스턴스에 마운트되었는지 확인해야 합니다.
- 사용자 지정 또는 이미 존재하는 **Jaeger** 인스턴스를 사용하려면 **spec.istio.tracing.enabled**를 "**false**"로 설정하여 **Jaeger** 인스턴스 배포를 비활성화합니다.
- **spec.istio.global.tracer.zipkin.address**를 **jaeger-collector** 서비스의 호스트 이름 및 포트로 설정하여 정확한 **jaeger-collector** 끝점을 **Mixer**에 제공합니다. 서비스의 호스트 이름은 일반적으로 **<jaeger-instance-name>-collector.<namespace>.svc.cluster.local**입니다.
- **spec.istio.kiali.jaegerInClusterURL**을 **jaeger-query** 서비스의 호스트 이름으로 설정하여 추적 수집에 올바른 **Jaeger-query** 끝점을 **Kiali**에 제공합니다. 기본적으로 포트는 **443**을 사용하므로 일반적으로 필요하지 않습니다. 서비스의 호스트 이름은 일반적으로 **<jaeger-instance-name>-query.<namespace>.svc.cluster.local**입니다.
- **Kiali** 콘솔을 통해 **Jaeger**에 액세스할 수 있도록 **Kiali**에 **Jaeger** 인스턴스의 대시보드 **URL**을 제공하십시오. **Jaeger Operator**가 생성한 **OpenShift** 경로에서 **URL**을 검색할 수 있습니다. **Jaeger** 리소스를 **external-jaeger**라고 하고 **istio-system** 프로젝트에 있는 경우, 다음 명령을 사용하여 경로를 검색할 수 있습니다.

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

출력 예

```
NAME                HOST/PORT                                  PATH   SERVICES            [...]
external-jaeger     external-jaeger-istio-system.apps.test            external-jaeger-query
[...]
```

HOST/PORT 아래의 값은 Jaeger 대시보드의 외부 액세스 URL입니다.

Jaeger 리소스 예

```
apiVersion: jaegertracing.io/v1
kind: "Jaeger"
metadata:
  name: "external-jaeger"
  # Deploy to the Control Plane Namespace
  namespace: istio-system
spec:
  # Set Up Authentication
  ingress:
    enabled: true
    security: oauth-proxy
    openshift:
      # This limits user access to the Jaeger instance to users who have access
      # to the control plane namespace. Make sure to set the correct namespace here
      sar: '{"namespace": "istio-system", "resource": "pods", "verb": "get"}'
      htpasswdFile: /etc/proxy/htpasswd/auth

  volumeMounts:
  - name: secret-htpasswd
    mountPath: /etc/proxy/htpasswd
  volumes:
  - name: secret-htpasswd
    secret:
      secretName: htpasswd
```

다음 ServiceMeshControlPlane 예는 Jaeger Operator 및 Jaeger 리소스 예제를 사용하여 Jaeger를 배포했다고 가정합니다.

외부 Jaeger가 있는 ServiceMeshControlPlane 예

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
metadata:
  name: external-jaeger
  namespace: istio-system
spec:
  version: v1.1
  istio:
    tracing:
      # Disable Jaeger deployment by service mesh operator
      enabled: false
    global:
      tracer:
        zipkin:
          # Set Endpoint for Trace Collection
          address: external-jaeger-collector.istio-system.svc.cluster.local:9411
    kiali:
      # Set Jaeger dashboard URL
      dashboard:
        jaegerURL: https://external-jaeger-istio-system.apps.test
      # Set Endpoint for Trace Querying
      jaegerInClusterURL: external-jaeger-query.istio-system.svc.cluster.local
```

2.10.5.3. Elasticsearch 구성

기본 Jaeger 배포 전략에서는 최소한의 리소스를 사용하여 설치를 완료할 수 있도록 all-in-one 템플릿을 사용합니다. 하지만 all-in-one 템플릿은 메모리 내 스토리지를 사용하므로 개발, 데모 또는 테스트 목적으로만 권장되며 프로덕션 환경에 사용해서는 안 됩니다.

프로덕션 환경에서 서비스 메시와 Jaeger를 배포하는 경우, 템플릿을 Jaeger의 스토리지 요건에 Elasticsearch를 사용하는 production-elasticsearch 템플릿으로 변경해야 합니다.

Elasticsearch는 메모리를 많이 사용하는 애플리케이션입니다. 기본 OpenShift Container Platform 설치에 지정된 초기 노드 세트는 Elasticsearch 클러스터를 지원하기에 충분히 크지 않을 수 있습니다. 사용 사례와 OpenShift Container Platform 설치에 요청한 리소스가 일치하도록 기본 Elasticsearch 구성을 수정해야 합니다. 유효한 CPU 및 메모리값으로 리소스 블록을 수정하여 각 구성 요소의 CPU 및 메모리 제한을 모두 조정할 수 있습니다. 권장 메모리 양, 또는 그 이상으로 실행하려는 경우 클러스터에 추가 노드를 추가해야 합니다. OpenShift Container Platform 설치에 요청된 리소스를 초과하지 않는지 확인합니다.

Elasticsearch를 사용하는 기본 "production" Jaeger 매개변수

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

표 2.13. Elasticsearch 매개변수

매개변수	설명	값	기본값	예제
tracing: enabled:	이 매개변수는 서비스 메시에서 추적을 활성화/비활성화합니다. Jaeger는 기본적으로 설치되어 있습니다.	**true/false**	**true**	
ingress: enabled:	이 매개변수는 Jaeger에 대해 수신을 활성화/비활성화합니다.	**true/false**	**true**	
jaeger: template:	이 매개변수는 사용할 Jaeger 배포 전략을 지정합니다.	**all-in-one/production-elasticsearch**	**all-in-one**	
elasticsearch: nodeCount:	생성할 Elasticsearch 노드 수입니다.	정수 값.	1	개념 증명 = 1, 최소 배포 = 3

매개변수	설명	값	기본값	예제
requests: cpu:	사용자 환경 구성에 따른 요청에 대한 중앙 처리 단위 수입니다.	코어 또는 밀리코어(예: 200m, 0.5, 1)에 지정되어 있습니다.	1Gi	개념 증명 = 500m, 최소 배포 = 1
requests: memory:	환경 구성에 따른 요청에 사용 가능한 메모리입니다.	바이트로 지정됩니다(예: 200Ki, 50Mi, 5Gi).	500m	개념 증명 = 1Gi, 최소 배포 = 16Gi*
limits: cpu:	사용자 환경 구성에 따른 중앙 처리 장치 수에 대한 제한입니다.	코어 또는 밀리코어(예: 200m, 0.5, 1)에 지정되어 있습니다.		개념 증명 = 500m, 최소 배포 = 1
limits: memory:	사용자 환경 구성에 따라 사용 가능한 메모리 제한입니다.	바이트로 지정됩니다(예: 200Ki, 50Mi, 5Gi).		개념 증명 = 1Gi, 최소 배포 = 16Gi*
	각 Elasticsearch 노드는 더 낮은 메모리 설정으로 작동할 수 있지만 프로덕션 배포에는 권장되지 **않습니다**. 프로덕션 용도의 경우 기본적으로 각 Pod에 할당된 16Gi 미만이 있어야 하지만 Pod당 최대 64Gi까지 할당할 수도 있습니다.			

프로세스

1. **cluster-admin 역할의 사용자로 OpenShift Container Platform 웹 콘솔에 로그인합니다.**
2. **Operators → 설치된 Operator로 이동합니다.**
3. **Red Hat OpenShift Service Mesh Operator를 클릭합니다.**
4. **Istio Service Mesh Control Plane 탭을 클릭합니다.**
5. **컨트롤 플레인 파일의 이름을 클릭합니다(예: basic-install).**
6. **YAML 탭을 클릭합니다.**
7.

Jaeger 매개변수를 편집하여, 기본 all-in-one 템플릿을 사용 사례에 맞게 수정한 production-elasticsearch 템플릿의 매개변수로 바꿉니다. 들여쓰기가 올바른지 확인합니다.

8. 저장을 클릭합니다.

9. 새로 고침을 클릭합니다. OpenShift Container Platform은 Jaeger를 재배포하고 지정된 매개변수를 기반으로 Elasticsearch 리소스를 생성합니다.

2.10.5.4. Elasticsearch 인덱스 정리 작업 구성

Service Mesh Operator가 ServiceMeshControlPlane을 생성할 때 Jaeger에 대한 사용자 정의 리소스(CR)도 생성합니다. Red Hat OpenShift distributed tracing platform(Jaeger) Operator는 Jaeger 인스턴스를 생성할 때 이 CR을 사용합니다.

Elasticsearch 스토리지를 사용하는 경우 기본적으로 오래된 추적을 정리하는 작업이 생성됩니다. 이 작업에 대한 옵션을 설정하려면 Jaeger 사용자 정의 리소스(CR)를 편집하여 사용 사례에 맞게 사용자 지정할 수 있습니다. 관련 옵션은 아래에 나열되어 있습니다.

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

표 2.14. Elasticsearch 인덱스 정리 매개변수

매개변수	값	설명
활성화됨:	true/ false	인덱스 정리 작업을 활성화하거나 비활성화합니다.
numberOfDays:	정수 값	인덱스를 삭제하기 전에 대기하는 날의 수입니다.
schedule:	"55 23 * * *"	실행할 작업의 Cron 표현식

OpenShift Container Platform을 사용한 Elasticsearch 구성에 대한 자세한 내용은 로그 저장소 구성을 참조하십시오.

2.10.6. 3scale 구성

다음 표는 ServiceMeshControlPlane 리소스의 3scale Istio 어댑터에 대한 매개변수를 설명합니다

3scale 매개변수 예

```
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
```

표 2.15. 3scale 매개변수

매개변수	설명	값	기본값
enabled	3scale 어댑터 사용 여부	true/false	false
PARAM_THREESCALE_LISTEN_ADDR	gRPC 서버의 수신 주소를 설정	유효한 포트 번호	3333
PARAM_THREESCALE_LOG_LEVEL	최소 로그 출력 수준을 설정합니다.	debug, info, warn, error 또는 none	info
PARAM_THREESCALE_LOG_JSON	로그 형식이 JSON인지 여부를 제어	true/false	true

매개변수	설명	값	기본값
PARAM_THREESCALE_LOG_GRPC	로그에 gRPC 정보가 포함되었는지 여부를 제어	**true/false**	**true**
PARAM_THREESCALE_REPORT_METRICS	3scale 시스템 및 백엔드 지표가 수집되어 Prometheus에 보고되는지 제어	**true/false**	**true**
PARAM_THREESCALE_METRICS_PORT	3scale **/metrics** 끝점을 스크랩할 수 있는 포트를 설정	유효한 포트 번호	**8080**
PARAM_THREESCALE_CACHE_TTL_SECONDS	캐시에서 만료된 항목을 제거하기 전에 대기하는 시간(초)	시간(초)	**300**
PARAM_THREESCALE_CACHE_REFRESH_SECONDS	캐시 요소를 새로 고침하려고 할 때 만료되기 전 시간	시간(초)	**180**
PARAM_THREESCALE_CACHE_ENTRIES_MAX	언제든지 캐시에 저장할 수 있는 항목의 최대 수. 캐싱을 비활성화하려면 **0**으로 설정합니다.	유효한 번호	**1000**
PARAM_THREESCALE_CACHE_REFRESH_RETRIES	캐시 업데이트 루프 중에 연결할 수 없는 호스트가 재시도되는 횟수	유효한 번호	**1**
PARAM_THREESCALE_ALLOW_INSECURE_CONN	**3scale** API를 호출할 때 인증서 확인을 건너뛸 수 있습니다. 이 설정 사용은 권장되지 않습니다.	**true/false**	**false**
PARAM_THREESCALE_CLIENT_TIMEOUT_SECONDS	3scale System 및 백엔드에 대한 요청을 종료하기 전 대기하는 시간(초)을 설정합니다.	시간(초)	**10**
PARAM_THREESCALE_GRPC_CONN_MAX_SECONDS	연결이 닫히기 전에 연결할 수 있는 최대 시간(초) (+/-10% jitter)을 설정합니다.	시간(초)	60

매개변수	설명	값	기본값
PARAM_USE_CACHE_BACKEND	true인 경우, 권한 부여 요청에 대해 메모리 내 apisonator 캐시를 생성합니다.	**true/false**	**false**
PARAM_BACKEND_CACHE_FLUSH_INTERVAL_SECONDS	백엔드 캐시가 활성화된 경우 3scale에 대해 캐시를 플러싱하는 간격(초)을 설정합니다.	시간(초)	15
PARAM_BACKEND_CACHE_POLICY_FAIL_CLOSED	백엔드 캐시가 권한 부여 데이터를 검색할 수 없을 때마다 요청을 거부(닫기)할지, 허용할지(열기) 여부	**true/false**	**true**

2.11. 3SCALE ISTIO 어댑터 사용

주의

더 이상 지원되지 않는 **Red Hat OpenShift Service Mesh** 릴리스에 대한 문서를 보고 있습니다.

서비스 메시 버전 **1.0** 및 **1.1** 컨트롤 플레인은 더 이상 지원되지 않습니다. 서비스 메시 컨트롤 플레인 업그레이드에 대한 자세한 내용은 서비스 메시 업그레이드를 참조하십시오.

특정 **Red Hat OpenShift Service Mesh** 릴리스의 지원 상태에 대한 자세한 내용은 제품 라이프사이클 페이지를 참조하십시오.

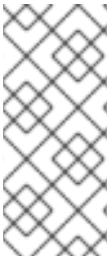
3scale Istio Adapter는 **Red Hat OpenShift Service Mesh** 내에서 실행되는 서비스에 레이블을 지정하고 해당 서비스를 **3scale API** 관리 솔루션과 통합할 수 있는 선택적 어댑터입니다. **Red Hat OpenShift Service Mesh**에는 필요하지 않습니다.

2.11.1. Red Hat OpenShift Service Mesh와 3scale 어댑터 통합

이러한 예제를 사용하여 3scale Istio 어댑터로 서비스에 대한 요청을 구성할 수 있습니다.

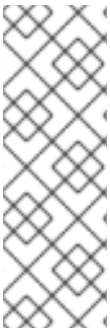
사전 요구 사항

- Red Hat OpenShift Service Mesh 버전 1.x
- 작업 중인 3scale 계정(SaaS 또는 3scale 2.5 on-Premises)
- 백엔드 캐시를 활성화하려면 3scale 2.9 이상 필요
- Red Hat OpenShift Service Mesh 사전 요구 사항



참고

3scale Istio Adapter를 구성하려면 사용자 정의 리소스 파일에 어댑터 매개변수를 추가하는 방법에 대한 Red Hat OpenShift Service Mesh 사용자 정의 리소스를 참조하십시오.



참고

특히 kind: handler 리소스에 주의하십시오. 3scale 계정 인증 정보로 업데이트해야 합니다. 선택적으로 service_id를 처리기에 추가할 수 있지만 3scale 계정의 하나의 서비스에만 처리기를 렌더링하므로 이전 버전과의 호환성을 위해서만 유지됩니다. service_id를 처리기에 추가하는 경우 다른 서비스에 3scale을 활성화하려면 다른 service_ids로 더 많은 처리기를 생성해야 합니다.

아래 단계에 따라 3scale 계정당 단일 처리기를 사용합니다.

절차

1. 3scale 계정에 대한 처리기를 생성하고 계정 인증 정보를 지정합니다. 서비스 식별자를 생략합니다.

```
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"
```

필요한 경우, 3scale 구성에서 제공하는 URL을 재정의하기 위해 *params* 섹션에 backend_url 필드를 제공할 수 있습니다. 어댑터가 3scale 온프레미스 인스턴스와 동일한 클러스터에서 실행되고 내부 클러스터 DNS를 사용하려는 경우 유용할 수 있습니다.

2. 다음과 같이 3scale 계정에 속하는 서비스의 배포 리소스를 편집하거나 패치합니다.

   a. 유효한 service_id에 해당하는 값을 사용하여 "service-mesh.3scale.net/service-id" 레이블을 추가합니다.

   b. 1단계에서 *처리기 리소스의 이름*이 값이 되도록 "service-mesh.3scale.net/credentials" 레이블을 추가합니다.

3. 더 많은 서비스를 추가하려는 경우 2단계를 수행하여 3scale 계정 인증 정보 및 서비스 식별자에 연결합니다.

4. 3scale 구성으로 규칙 구성을 수정하여 3scale 처리기에 규칙을 전송합니다.

규칙 구성 예

```
apiVersion: "config.istio.io/v1alpha2"
kind: rule
metadata:
 name: threescale
spec:
 match: destination.labels["service-mesh.3scale.net"] == "true"
 actions:
  - handler: threescale.handler
    instances:
     - threescale-authorization.instance
```

2.11.1.1. 3scale 사용자 정의 리소스 생성

어댑터에는 handler, instance, rule 사용자 정의 리소스를 생성할 수 있는 도구가 포함되어 있습니다.

표 2.16. 사용법

옵션	설명	필수 항목	기본값
-h, --help	사용 가능한 옵션에 대한 도움말 출력 생성	아니요	
--name	이 URL의 고유 이름, 토큰 쌍	예	
-n, --namespace	템플릿을 생성할 네임스페이스	아니요	istio-system
-t, --token	3scale 액세스 토큰	예	
-u, --url	3scale 관리자 포털 URL	예	
--backend-url	3scale 백엔드 URL. 설정하면 시스템 설정에서 읽은 값을 재정의합니다.	아니요	
-s, --service	3scale API/서비스 ID	아니요	
--auth	지정을 위한 3scale 인증 패턴(1=API Key, 2=App Id/App Key, 3=OIDC)	아니요	하이브리드
-o, --output	생성된 매니페스트를 저장할 파일	아니요	표준 출력
--version	CLI 버전을 출력하고 즉시 종료합니다.	아니요	

2.11.1.1.1. URL 예제에서 템플릿 생성

참고

- 배포된 어댑터에서 매니페스트 생성에 있는 **3scale** 어댑터 컨테이너 이미지에서 **oc exec**를 통해 다음 명령을 실행합니다.
- **3scale-config-gen** 명령을 사용하여 **YAML** 구문 및 들여쓰기 오류를 방지할 수 있습니다.
- 주석을 사용하는 경우 **--service**를 생략할 수 있습니다.
- 이 명령은 **oc exec**를 통해 컨테이너 이미지 내에서 호출해야 합니다.

절차

- **3scale-config-gen** 명령을 사용하여 토큰, **URL** 쌍을 단일 처리기로 여러 서비스에서 공유할 수 있도록 템플릿 파일을 자동 생성합니다.

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

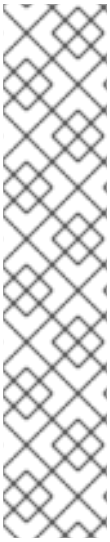
- 다음 예제에서는 처리기에 포함된 서비스 **ID**로 템플릿을 생성합니다.

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

추가 리소스

- 토큰.

2.11.1.2. 배포된 어댑터에서 매니페스트 생성



참고

- NAME은 3scale로 관리 중인 서비스와 식별하는 데 사용하는 식별자입니다.
- CREDENTIALS_NAME 참조는 규칙 구성의 match 섹션에 해당하는 식별자입니다. CLI 툴을 사용하는 경우 NAME 식별자로 자동 설정됩니다.
- 해당 값은 특정할 필요가 없습니다. 레이블 값은 규칙의 내용과 일치해야 합니다. 자세한 정보는 어댑터를 통한 서비스 트래픽 라우팅을 참조하십시오.

1. 이 명령을 실행하여 istio-system 네임스페이스의 배포된 어댑터에서 매니페스트를 생성합니다.

```
$ 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}
```

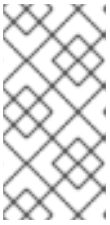
2. 터미널에 샘플 출력이 생성됩니다. 필요한 경우 이러한 샘플을 편집하고 oc create 명령을 사용하여 오브젝트를 생성합니다.

3. 요청이 어댑터에 도달하면 어댑터는 서비스가 3scale의 API에 매핑되는 방식을 알아야 합니다. 다음 두 가지 방법으로 이러한 정보를 제공할 수 있습니다.

   a. 워크로드에 레이블 지정(권장)

   b. 처리기를 service_id로 하드 코딩

4. 필요한 주석으로 워크로드를 업데이트합니다.



참고

처리기에 아직 포함되지 않은 경우, 이 예제에 제공된 서비스 **ID**만 업데이트해야 합니다. 처리기의 설정이 우선합니다.

```
$ export CREDENTIALS_NAME="replace-me"
export SERVICE_ID="replace-me"
export DEPLOYMENT="replace-me"
patch="$(oc get deployment "${DEPLOYMENT}"
patch="$(oc get deployment "${DEPLOYMENT}" --template='{"spec":{"template":{"metadata":
{"labels":{ {{ range $k,$v := .spec.template.metadata.labels }}"{{ $k }}":"{{ $v }}",{{ end
}}"service-mesh.3scale.net/service-id":"""${SERVICE_ID}""","service-
mesh.3scale.net/credentials":"""${CREDENTIALS_NAME}"""}}}}}' )"
oc patch deployment "${DEPLOYMENT}" --patch ""${patch}""
```

2.11.1.3. 어댑터를 통한 서비스 트래픽 라우팅

3scale 어댑터를 통해 서비스 트래픽을 유도하려면 다음 단계를 따르십시오.

사전 요구 사항

- **3scale** 관리자의 인증 정보 및 서비스 **ID**

절차

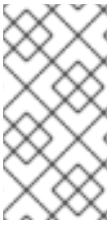
1. **kind: rule** 리소스의 구성에서 이전에 생성한 **destination.labels["service-mesh.3scale.net/credentials"] == "threescale"** 규칙과 일치합니다.

2. 서비스를 통합하기 위해 대상 워크로드 배포에서 위의 레이블을 **PodTemplateSpec**에 추가합니다. **threescale** 값은 생성된 처리기의 이름을 나타냅니다. 이 처리기에서는 **3scale**를 호출하는 데 필요한 액세스 토큰을 저장합니다.

3. **destination.labels["service-mesh.3scale.net/service-id"] == "replace-me"** 레이블을 워크로드에 추가하여 요청 시 인스턴스를 통해 서비스 **ID**를 어댑터에 전달합니다.

2.11.2. 3scale로 통합 설정 구성

3scale 통합 설정을 구성하려면 다음 절차를 따르십시오.



참고

3scale SaaS 고객의 경우, Red Hat OpenShift Service Mesh는 조Early Access 프로그램의 일부로 활성화됩니다.

절차

1. [your_API_name] → 통합으로 이동합니다.
2. 설정을 클릭합니다.
3. *배포*에서 Istio 옵션을 선택합니다.
   - *인증*에서 API Key (user_key) 옵션은 기본적으로 선택됩니다.
4. 제품 업데이트를 클릭하여 선택 사항을 저장합니다.
5. 설정을 클릭합니다.
6. 구성 업데이트를 클릭합니다.

2.11.3. 캐싱 동작

3scale System API의 응답은 기본적으로 어댑터 내에서 캐시됩니다. 항목이 cacheTTLSeconds 값보다 오래되면 캐시에서 제거됩니다. 또한 기본적으로 캐시된 항목의 자동 새로 고침은 cacheRefreshSeconds 값에 따라 만료되기 몇 초 전에 시도됩니다. 이 값을 cacheTTLSeconds 값보다 높게 설정하여 자동 새로 고침을 비활성화할 수 있습니다.

cacheEntriesMax를 양수가 아닌 값으로 설정하여 캐싱을 완전히 비활성화할 수 있습니다.

새로 고침 프로세스를 사용하면 호스트가 연결할 수 없는 캐시된 값은 만료가 지나면 제거되기 전에 다시 시도됩니다.

2.11.4. 요청 인증

이 릴리스에서는 다음 인증 방법을 지원합니다.

- 표준 **API** 키: 식별자와 시크릿 토큰으로 작동하는 임의의 단일 문자열 또는 해시입니다.
- 애플리케이션 식별자 및 키 쌍: 변경 불가능한 식별자 및 변경 가능한 시크릿 키 문자열입니다.
- **OpenID** 인증 방법: **JSON** 웹 토큰에서 구문 분석된 클라이언트 **ID** 문자열입니다.

2.11.4.1. 인증 패턴 적용

인증 동작을 구성하려면 다음 인증 방법 예제에 설명된 인스턴스 사용자 정의 리소스를 수정합니다. 다음에서 인증 자격 증명을 허용할 수 있습니다.

- 요청 헤더
- 요청 매개변수
- 요청 헤더 및 쿼리 매개변수 둘 다

참고

헤더에서 값을 지정하는 경우 소문자여야 합니다. 예를 들어 **User-Key**로 헤더를 보내려면 구성에서 **request.headers["user-key"]**로 참조되어야 합니다.

2.11.4.1.1. API 키 인증 방법

서비스 메시는 **subject** 사용자 정의 리소스 매개변수의 **user** 옵션에 지정된 대로 쿼리 매개변수 및 요청 헤더에서 **API** 키를 찾습니다. 사용자 정의 리소스 파일에 지정된 순서로 값을 확인합니다. 원하지 않는 옵션을 생략하여 **API** 키 검색을 쿼리 매개변수 또는 요청 헤더로 제한할 수 있습니다.

이 예에서 서비스 메시는 **user_key** 쿼리 매개변수에서 **API** 키를 찾습니다. **API** 키가 쿼리 매개변수에 없으면 서비스 메시가 **user-key** 헤더를 확인합니다.

API 키 인증 방법 예

```
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"
```

어댑터가 다른 쿼리 매개변수 또는 요청 헤더를 검사하도록 하려면 이름을 적절하게 변경합니다. 예를 들어 "key"라는 쿼리 매개변수에서 API 키를 확인하려면 request.query_params["user_key"]을 request.query_params["key"]로 변경합니다.

2.11.4.1.2. 애플리케이션 ID 및 애플리케이션 키 쌍 인증 방법

서비스 메시는 subject 사용자 정의 리소스 매개변수의 properties 옵션에 지정된 대로 쿼리 매개변수 및 요청 헤더에서 애플리케이션 ID와 애플리케이션 키를 찾습니다. 애플리케이션 키는 선택 사항입니다. 사용자 정의 리소스 파일에 지정된 순서로 값을 확인합니다. 원하지 않는 옵션을 제외하여 자격 증명 검색을 쿼리 매개변수 또는 요청 헤더로 제한할 수 있습니다.

이 예에서 서비스 메시는 쿼리 매개변수의 애플리케이션 ID 및 애플리케이션 키를 먼저 찾고 필요한 경우 요청 헤더로 이동합니다.

애플리케이션 ID 및 애플리케이션 키 쌍 인증 방법 예

```
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"
```

어댑터가 다른 쿼리 매개변수 또는 요청 헤더를 검사하도록 하려면 이름을 적절하게 변경합니다. 예를 들어, identification라는 쿼리 매개변수의 애플리케이션 ID를 확인하려면 request.query_params["app_id"]을 request.query_params["identification"]로 변경합니다.

2.11.4.1.3. OpenID 인증 방법

*OIDC(OpenID Connect) 인증 방법*을 사용하려면 subject 필드에서 properties 값을 사용하여 client_id 또는 필요한 경우 app_key로 설정할 수 있습니다.

이전에 설명된 방법을 사용하여 이 오브젝트를 조작할 수 있습니다. 아래 설정 예에서 클라이언트 식별자(애플리케이션 ID)는 *azp* 레이블 아래에 있는 JSON 웹 토큰(JWT)에서 구문 분석됩니다. 필요에 따라 수정할 수 있습니다.

OpenID 인증 방법 예

```
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"] | ""
```

이 통합이 올바르게 작동하려면 클라이언트가 ID 공급자(IdP)에서 생성되도록 OIDC를 3scale에서 수행해야 합니다. 해당 서비스와 동일한 네임스페이스에서 보호하려는 서비스에 대해 요청 권한 부여를 생성해야 합니다. JWT는 요청의 Authorization 헤더로 전달됩니다.

아래에 정의된 샘플 RequestAuthentication에서 issuer, jwksUri, selector를 적절하게 대체합니다.

OpenID 정책 예

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: info
spec:
  selector:
    matchLabels:
      app: productpage
  jwtRules:
  - issuer: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-keycloak
    jwksUri: >-
      http://keycloak-keycloak.34.242.107.254.nip.io/auth/realms/3scale-
keycloak/protocol/openid-connect/certs
```

2.11.4.1.4. 하이브리드 인증 방법

특정 인증 방법을 적용하지 않도록 선택하고, 두 방법에 대해 유효한 자격 증명을 수락할 수 있습니다. API 키와 애플리케이션 ID/애플리케이션 키 쌍이 모두 제공되면 서비스 메시는 API 키를 사용합니다.

이 예제에서 서비스 메시는 쿼리 매개변수에서 API 키를 확인한 다음 요청 헤더를 확인합니다. API 키가 없는 경우 쿼리 매개변수에서 애플리케이션 ID와 키를 확인한 다음 요청 헤더를 확인합니다.

하이브리드 인증 방법 예

```
apiVersion: "config.istio.io/v1alpha2"
kind: instance
metadata:
  name: threescale-authorization
spec:
```

```
template: authorization
params:
  subject:
    user: request.query_params["user_key"] | request.headers["user-key"] |
    properties:
      app_id: request.query_params["app_id"] | request.headers["app-id"] | ""
      app_key: request.query_params["app_key"] | request.headers["app-key"] | ""
      client_id: request.auth.claims["azp"] | ""
  action:
    path: request.url_path
    method: request.method | "get"
    service: destination.labels["service-mesh.3scale.net/service-id"] | ""
```

2.11.5. 3scale Adapter 지표

기본적으로 어댑터는 /metrics 끝점의 포트 8080에서 공개되는 다양한 Prometheus 지표를 보고합니다. 이러한 지표는 어댑터와 3scale 간의 상호 작용 수행 방식에 대한 인사이트를 제공합니다. 이 서비스는 Prometheus에서 자동으로 검색 및 스크랩하도록 레이블이 지정됩니다.

2.11.6. 3scale Istio 어댑터 검증

3scale Istio 어댑터가 예상대로 작동하는지 확인해야 할 수 있습니다. 어댑터가 작동하지 않는 경우 다음 단계를 사용하여 문제를 해결합니다.

절차

1. Service Mesh Control Plane 네임스페이스에서 *3scale-adapter* Pod가 실행 중인지 확인합니다.

```
$ oc get pods -n <istio-system>
```

2. *3scale-adapter* Pod에서 버전과 같이 자체 부팅에 대한 정보를 출력했는지 확인합니다.

```
$ oc logs <istio-system>
```

3. 3scale 어댑터 통합으로 보호되는 서비스에 대한 요청을 수행할 때 항상 올바른 인증 정보가 없는 요청을 시도하여 실패하는지 확인합니다. 3scale 어댑터 로그를 확인하여 추가 정보를 수집합니다.

추가 리소스

- Pod 및 컨테이너 로그 검사

2.11.7. 3scale Istio 어댑터 문제 해결 체크리스트

3scale Istio 어댑터를 설치하는 관리자는 통합이 제대로 작동하지 않을 수 있는 여러 시나리오가 있습니다. 다음 목록을 사용하여 설치 문제를 해결합니다.

- YAML 들여쓰기가 잘못되었습니다.
- YAML 섹션이 누락되었습니다.
- YAML 변경 사항을 클러스터에 적용하는 것을 잊어버렸습니다.
- service-mesh.3scale.net/credentials 키를 사용하여 서비스 워크로드의 레이블을 지정하는 것을 잊어버렸습니다.
- 서비스 워크로드에 service_id가 포함되지 않은 처리기를 사용할 때 service-mesh.3scale.net/service-id로 레이블을 지정하여 계정별로 재사용할 수 있도록 하는 것을 잊었버렸습니다.
- *Rule* 사용자 지정 리소스는 잘못된 처리기 또는 인스턴스 사용자 지정 리소스를 가리키거나 참조에 해당 네임스페이스 접미사가 없습니다.
- *Rule* 사용자 정의 리소스 match 섹션은 구성 중인 서비스와 일치하지 않거나 현재 실행 중이거나 존재하지 않는 대상 워크로드를 가리킵니다.
- 처리기의 3scale 관리 포털의 잘못된 액세스 토큰 또는 URL입니다.
- *인스턴스* 사용자 정의 리소스의 params/subject/properties 섹션은 쿼리 매개 변수, 헤더 및 권한 부여 클레임과 같은 잘못된 위치를 지정하거나 매개 변수 이름이 테스트에 사용되는 요청과 일치하지 않기 때문에 app_id, app_key, 또는 client_id에 대한 올바른 매개 변수를 나열하지 못합니다.
-

구성 생성기가 실제로 어댑터 컨테이너 이미지에 있고 **oc exec** 호출 해야 한다는 사실을 인식하지 못한 채 구성 생성기를 사용하지 못했습니다 .

2.12. 서비스 메시 제거

주의

더 이상 지원되지 않는 **Red Hat OpenShift Service Mesh** 릴리스에 대한 문서를 보고 있습니다.

서비스 메시 버전 **1.0** 및 **1.1** 컨트롤 플레인은 더 이상 지원되지 않습니다. 서비스 메시 컨트롤 플레인 업그레이드에 대한 자세한 내용은 서비스 메시 업그레이드를 참조하십시오.

특정 **Red Hat OpenShift Service Mesh** 릴리스의 지원 상태에 대한 자세한 내용은 제품 라이프사이클 페이지를 참조하십시오.

기존 **OpenShift Container Platform** 인스턴스에서 **Red Hat OpenShift Service Mesh**를 제거하려면 **operator**를 제거하기 전에 컨트롤 플레인을 제거하십시오.

2.12.1. Red Hat OpenShift Service Mesh Control Plane 제거

기존 **OpenShift Container Platform** 인스턴스에서 **Service Mesh**를 설치 제거하려면 먼저 **Service Mesh Control Plane**과 **Operator**를 삭제합니다. 그런 다음 명령을 실행하여 남은 리소스를 제거합니다.

2.12.1.1. 웹 콘솔을 사용하여 Service Mesh Control Plane 제거

웹 콘솔을 사용하여 **Red Hat OpenShift Service Mesh Control Plane**을 제거할 수 있습니다.

절차

1. **OpenShift Container Platform** 웹 콘솔에 로그인합니다.

2.

프로젝트 메뉴를 클릭하고 **Service Mesh Control Plane**을 설치한 프로젝트(예: **istio-system**)를 선택합니다.

3. **Operators** → 설치된 **Operator**로 이동합니다.

4. 제공된 **API**에서 **Service Mesh Control Plane**을 클릭합니다.

5. **ServiceMeshControlPlane** 메뉴

를 클릭합니다.

6. **Service Mesh Control Plane** 삭제를 클릭합니다.

7. 확인 대화 상자에서 삭제를 클릭하여 **ServiceMeshControlPlane**을 삭제합니다.

2.12.1.2. CLI를 사용하여 Service Mesh Control Plane 제거

CLI를 사용하여 **Red Hat OpenShift Service Mesh Control Plane**을 제거할 수 있습니다. 이 예제에서 **istio-system**은 컨트롤 플레인 프로젝트의 이름입니다.

절차

1. **OpenShift Container Platform CLI**에 로그인합니다.

2. 다음 명령을 실행하여 **ServiceMeshMemberRoll** 리소스를 삭제합니다.

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

3. 이 명령을 실행하여 설치된 **ServiceMeshControlPlane**의 이름을 검색합니다.

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

4. **<name_of_custom_resource>**을 이전 명령의 출력으로 바꾸고, 이 명령을 실행하여 사용

자 정의 리소스를 삭제합니다.

```
$ oc delete smcp -n istio-system <name_of_custom_resource>
```

2.12.2. 설치된 Operator 제거

Red Hat OpenShift Service Mesh를 성공적으로 제거하려면 Operator를 제거해야 합니다. Red Hat OpenShift Service Mesh Operator를 제거한 후 Kiali Operator, Red Hat OpenShift distributed tracing Platform (Jaeger) Operator 및 OpenShift Elasticsearch Operator를 제거해야 합니다.

2.12.2.1. Operator 제거

Red Hat OpenShift Service Mesh를 구성하는 Operator를 제거하려면 다음 절차를 따르십시오. 다음 각 Operator에 대해 단계를 반복합니다.

- Red Hat OpenShift Service Mesh
- Kiali
- Red Hat OpenShift distributed tracing platform (Jaeger)
- OpenShift Elasticsearch

절차

1. OpenShift Container Platform 웹 콘솔에 로그인합니다.
2. Operator → 설치된 Operator 페이지에서 스크롤하거나 이름별 필터링에 키워드를 입력하여 각 Operator를 찾습니다. 그런 다음 Operator 이름을 클릭합니다.
3. Operator 상세 정보 페이지의 작업 메뉴에서 Operator 제거를 선택합니다. 프롬프트에 따라 각 Operator를 제거합니다.

2.12.2.2. Operator 리소스 정리

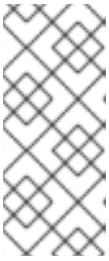
다음 절차에 따라 **OpenShift Container Platform** 웹 콘솔을 사용하여 **Red Hat OpenShift Service Mesh Operator**를 제거한 후 남은 리소스를 수동으로 제거하십시오.

사전 요구 사항

- 클러스터 관리 권한이 있는 계정.
- **OpenShift CLI(oc)**에 액세스합니다.

절차

1. **OpenShift Container Platform CLI**에 클러스터 관리자로 로그인합니다.
2. **Operator**를 제거한 후 다음 명령을 실행하여 리소스를 정리합니다. 서비스 메시 없이 **Jaeger**를 독립형 서비스로 계속 사용하려면 **Jaeger** 리소스를 삭제하지 마십시오.



참고

Operator는 기본적으로 **openshift-operators** 네임스페이스에 설치됩니다. 다른 네임스페이스에 **Operators**를 설치한 경우**openshift-operators**를 **Red Hat OpenShift Service Mesh Operator**가 설치된 프로젝트의 이름으로 교체합니다.

```
$ oc delete validatingwebhookconfiguration/openshift-operators.servicemesh-resources.maistra.io
```

```
$ oc delete mutatingwebhookconfiguration/openshift-operators.servicemesh-resources.maistra.io
```

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

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

```
$ oc delete clusterrole istio-view istio-edit
```

```
$ oc delete clusterrole jaegers.jaegertracing.io-v1-admin jaegers.jaegertracing.io-v1-crdview jaegers.jaegertracing.io-v1-edit jaegers.jaegertracing.io-v1-view
```

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

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

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

```
$ oc delete crds jaegers.jaegertracing.io
```

```
$ oc delete svc admission-controller -n <operator-project>
```

```
$ oc delete project <istio-system-project>
```