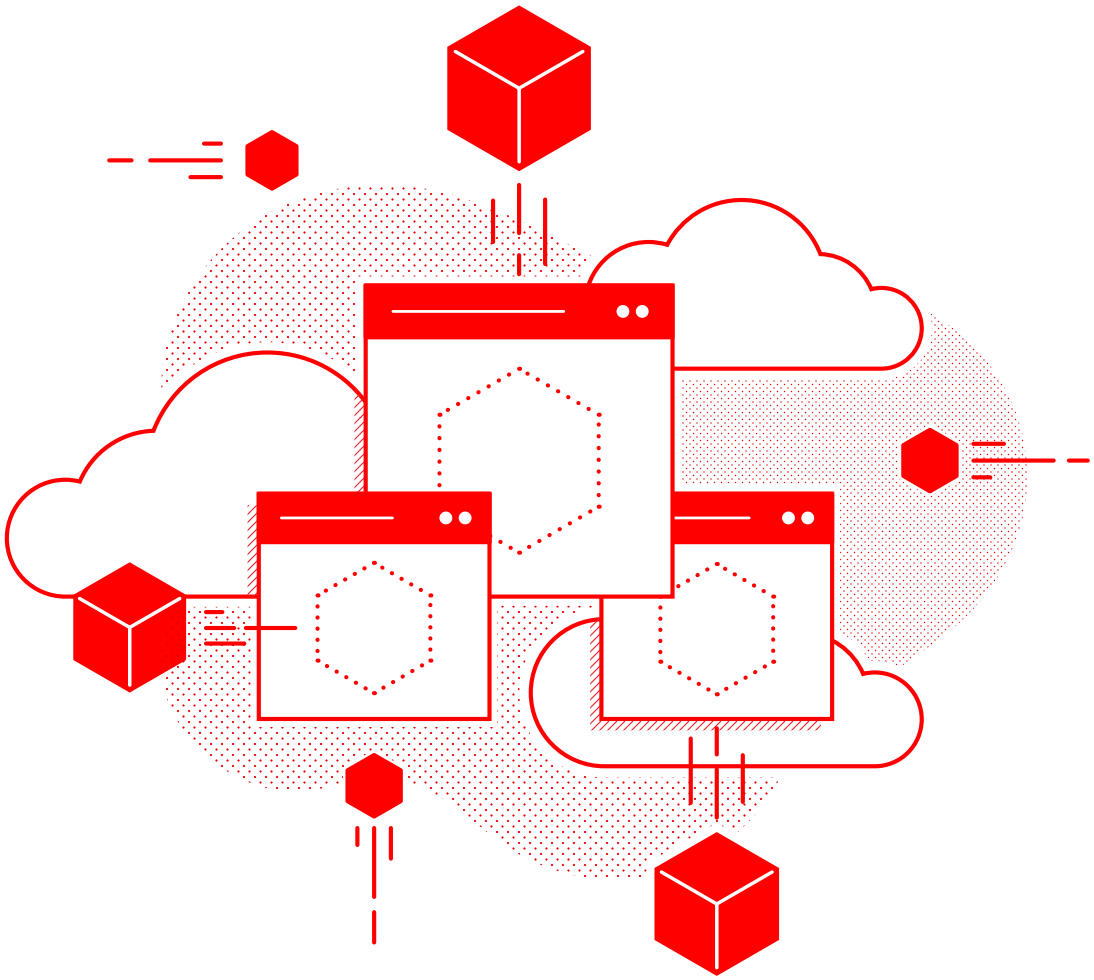
Red Hat



Red Hat OpenShift Service Mesh 시작하기

프로덕션 배포 및 Day 2 오퍼레이션에 대한 가이드

작성자: 선임 애플리케이션 아키텍트 Stelios Kousouris, 수석 제품 마케팅 매니저 Ortwin Schneider

목차

소개

기술과 애플리케이션은 오늘날 디지털 비즈니스의 핵심이며, 많은 조직이 신속하고 반복적인 애플리케이션 개발 및 배포를 지원하기 위해 현대적인 서비스 관리 아키텍처를 구축하고 있습니다. **서비스 메쉬**는 이러한 아키텍처에서 핵심적인 역할을 수행하며, 일련의 개별 서비스를 서로 연결하여 완전한 애플리케이션으로 구성합니다. 또한 애플리케이션 개발 간소화, 트러블슈팅 간소화, 관측성 강화, 복구 능력 증진, 성능 튜닝 촉진을 수행합니다.

하지만 서비스 메쉬 구축은 첫 번째 단계일 뿐입니다. 서비스 메쉬 구축에 따른 가치를 끌어내려면 조직의 요구 사항에 따라 서비스 메쉬를 구성, 튜닝, 유지 관리해야 합니다. 이 e-book에서는 거버넌스, 설계 사례, 프로덕션용 Red Hat® OpenShift® Service Mesh 구성, Day 2 오퍼레이션 수행에 관한 지침을 제공합니다.

Red Hat OpenShift Service Mesh 기본 사항

Red Hat OpenShift Service Mesh는 Red Hat OpenShift에 포함되어 있으며 마이크로서비스 기반 애플리케이션을 일관된 방식으로 연결, 관리, 관측할 수 있도록 지원합니다. 또한 마이크로서비스 간 트래픽을 통합, 관리, 추적, 모니터링, 분석하기 위한 다음과 같은 일련의 오픈소스 프로젝트를 통합합니다.

- Istio: 서비스 간 트래픽을 통합하고 관리하기 위한 오픈소스 프로젝트
- Jaeger: 서비스 간에 이동하는 요청을 추적하는 개방형 분산 추적 시스템
- Kiali: 구성 조회, 트래픽 모니터링, 추적 분석을 위한 오픈소스 프로젝트
- 여러 개의 네트워킹 인터페이스
- Red Hat 3scale API Management와의 통합을 위한 Red Hat 3scale Istio 플러그인

주요 특징 및 기능

Red Hat OpenShift Service Mesh는 서비스 네트워크 전반에서 다수의 주요 기능을 일관성 있게 제공합니다.

- **트래픽 관리.** 서비스 간 트래픽과 애플리케이션 프로그래밍 인터페이스(API) 호출의 흐름을 제어하고, 호출의 신뢰성을 높이며, 이상 조건에서 네트워크를 강화합니다.
- **서비스 Identity 및 보안.** 검증 가능한 Identity로 메쉬에서 서비스를 제공하고 다양한 신뢰성 수준의 네트워크를 통해 흐르는 서비스 트래픽을 보호할 수 있는 기능을 제공합니다.
- **정책 실행.** 조직의 정책을 서비스 간 상호 작용에 적용하고, 액세스 정책이 시행되고 리소스가 소비자들 사이에서 공평하게 분배되도록 보장합니다.
- **텔레메트리.** 서비스 간 종속성과 서비스 간 트래픽의 성격 및 흐름을 이해함으로써 문제를 신속히 파악할 수 있도록 지원합니다.

Red Hat OpenShift Service Mesh를 활용하면 다음을 실현할 수 있습니다.

- 개발자들이 서비스를 연결하는 대신 비즈니스 가치를 추가하는 일에 집중할 수 있습니다.
- 분산형 요청 추적과 가시적인 인프라 계층을 통해 문제를 더 쉽게 파악하고 진단할 수 있습니다.
- 서비스 메쉬는 장애가 발생한 서비스로부터 요청을 재라우팅할 수 있기 때문에 다운타임 발생 시 애플리케이션 복구 능력이 향상됩니다.
- 실행(runtime) 환경의 통신을 성능 메트릭과 관측성을 사용해 최적화할 수 있습니다.

Red Hat OpenShift Service Mesh 체험하기

로컬 환경에서 Red Hat OpenShift Service Mesh를 시작하세요. Red Hat OpenShift Local을 노트북에 설치한 다음, 이 e-book의 실습을 따라해 보세요. Red Hat OpenShift Local을 이용해 서비스 메쉬와 기타 Red Hat OpenShift 기능을 직접 실험해 볼 수도 있습니다.

Red Hat OpenShift Local을 다운로드하세요. →

이 e-book을 위한 온라인 리소스

이 e-book 외에도 Red Hat은 관련 리소스를 모아놓은 **온라인 리포지토리**를 구축했습니다. 이 리포지토리에는 이 e-book과 함께 참고할 수 있는 추가적인 세부 정보, 설정 스크립트, 코드 예시가 포함되어 있습니다. 텍스트 전반에 링크가 포함되어 있으며, 각 섹션 옆에 있는 링크를 클릭하면 온라인 리포지토리의 해당 섹션으로 바로 이동할 수 있습니다.

추가 자료

Red Hat OpenShift Service Mesh에 대해 알아보고, 이를 통해 팀이 마이크로서비스 기반 애플리케이션을 연결, 관리, 제어할 수 있는 방법을 알아보세요.

- **Red Hat OpenShift Service Mesh 제품 설명서**
- **Red Hat OpenShift Service Mesh 제품 정보**
- **서비스 메쉬 또는 API 관리 e-book**

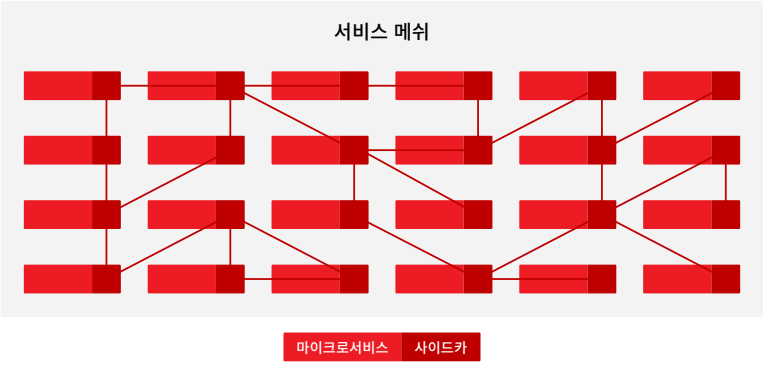


그림 1. 개념적인 서비스 메쉬 아키텍처

1부: 프로덕션 단계로 진행

조직은 많은 활용 사례를 위해 서비스 메쉬를 배포하지만 대부분 공통된 기반이 있습니다. 실제 Red Hat OpenShift Service Mesh 배포 및 운영을 더 잘 설명하기 위해 가상의 조직인 Travel by Keyboard가 서비스 메쉬를 설정, 운영, 유지 관리하는 과정을 따라가 보겠습니다.

Travel by Keyboard 소개

Travel by Keyboard는 전 세계 목적지에 대한 올인원 여행 패키지 전문 에이전시입니다. 이 에이전시는 항공사, 호텔, 렌터카 서비스, 보험사와 협력하여 고객을 위한 완벽한 여행 경험을 제공합니다.

Travel by Keyboard의 IT 부서는 이러한 비즈니스를 지원하기 위해 여행 포털 및 예약 플랫폼 시스템 아키텍처를 운영합니다. 현재 여행 포털 및 예약 플랫폼에서는 서비스 메쉬를 사용하고 있지 않지만 유연성과 복구 능력을 향상하는 동시에 개발 및 운영을 간소화하기 위해 서비스 메쉬를 배포하고자 합니다.

Travel by Keyboard의 타겟 서비스 메쉬 아키텍처는 그림 2와 같습니다. 여행, 항공편, 호텔, 차량, 보험을 위한 서비스가 포함된 마이크로서비스 백엔드 아키텍처가 특징입니다(`travel-agency` 네임스페이스에 포함되어 있음). `travel-portal` 네임스페이스에는 고객과 파트너를 위한 테넌트가 포함됩니다. 또한 Travel by Keyboard는 필요에 따라 서비스를 확장할 수 있도록 API를 통해 외부 여행 포털과 통합할 수 있는 옵션을 원합니다.

Travel by Keyboard 조직 내의 여러 팀과 역할은 서비스 메쉬 아키텍처와 다음과 같이 상호 작용합니다.

- 여행 포털 제품 팀은 여행 포털 인프라 및 애플리케이션을 소유합니다.
- 여행 서비스 제품 팀은 여행 서비스 인프라 및 애플리케이션을 소유합니다.
- 한 명의 제품 소유자는 완전한 여행사 플랫폼을 소유합니다.
- 플랫폼 관리자는 Red Hat OpenShift 플랫폼 및 오퍼레이터를 유지 관리합니다.
- 서비스 메쉬 오퍼레이터는 서비스 메쉬 컨트롤 플레인을 관리합니다.
- 여러 명의 서비스 메쉬 개발자는 서비스 메쉬에 대한 트래픽 구성을 생성합니다.

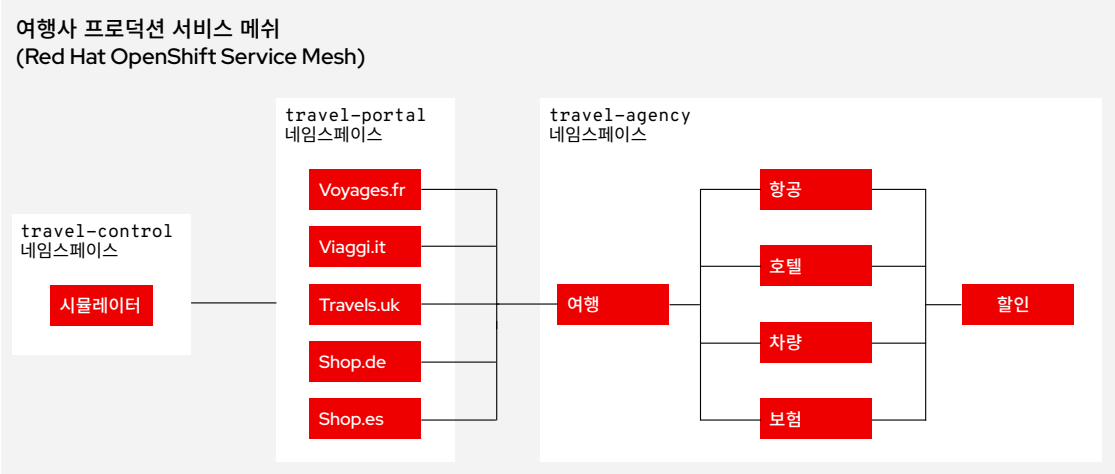


그림 2. Travel by Keyboard의 타겟 서비스 메쉬 아키텍처

1장

서비스 메쉬 요구 사항 및 전제 조건

서비스 메쉬를 프로덕션 단계로 배포하기 위한 첫 단계는 비즈니스 및 아키텍처 요구 사항을 파악하는 것입니다. 이러한 요구 사항을 다양한 방식으로 수집할 수 있지만 모든 이해관계자의 기대치를 충족하고 이해하는 것이 중요합니다.

서비스 메쉬 요구 사항 파악

이 예시에서 주요 이해관계자는 개발, 제품, 보안, 플랫폼 팀입니다. 프로젝트 시작 회의에서 이해관계자들은 다음과 같은 요구 사항을 파악했습니다.

개발 팀 요구 사항:

- 개발 중에 모든 요청을 추적하고 프로덕션 단계에서 트래픽의 20%를 샘플링할 수 있는 기능
- 애플리케이션 복구 능력, 안정성, 신뢰성을 높이기 위한 기능 및 구성

제품 팀 요구 사항:

- 보고할 가치가 있는 성능 및 사용 메트릭을 수집할 수 있는 기능
- 최대 1주 동안의 메트릭을 저장할 수 있는 기능

보안 팀 요구 사항:

- 모든 인트라메쉬 및 인터메쉬 통신을 위한 mTLS(mutual Transport Layer Security)

플랫폼 팀 요구 사항:

- 중앙에서 관리하는 보안
- 서비스 메쉬 구성 및 관리를 위한 선언적 접근 방식(예: GitOps)

서비스 메쉬의 역할 및 책임 정의

그다음 단계는 서비스 메쉬 구축 및 관리에 수반되는 사용자 유형, 역할, 책임을 정의하는 것입니다. 조직, 운영, 거버넌스와 관련한 몇 가지 요인은 이러한 항목이 설정되는 방식에 영향을 미칠 수 있습니다. 해당 요인은 다음과 같습니다.

- **사용되는 클러스터 및 메쉬의 유형.** 클러스터는 다중 영역 애플리케이션 클러스터나 중요 클러스터일 수 있는 반면, 메쉬는 멀티테넌트 또는 단일 테넌트일 수 있습니다.
- **클라우드 서비스 구성에 사용되는 자동화의 유형.** 파이프라인, GitOps, 자동화 플랫폼, 스크립트, 관리 툴 등을 선택할 수 있습니다.
- **플랫폼 또는 서비스 메쉬 운영 모델.** 이는 서비스 메쉬에 대한 액세스 방식을 정의합니다. 생산자-소비자 플랫폼을 통해 관리자와 오퍼레이터는 개발자가 사용할 수 있도록 모든 구성을 배포할 수 있고, 셀프 서비스 플랫폼을 통해 개발자는 사전 승인된 구성을 직접 프로비저닝할 수 있습니다.
- **애플리케이션 및 클라우드 구성 제공을 위한 DevSecOps 접근 방식 도입 여부.** DevSecOps 접근 방식은 개발, 보안, 운영을 협업적인 공유 책임 패러다임으로 결합합니다. 목표는 조직 전반에서 역할, 분야, 팀 간의 장벽을 허물어 협업을 증진하고 공동의 목표를 향해 노력하는 것입니다. DevSecOps 접근 방식은 인력, 프로세스, 기술, 거버넌스를 포괄합니다.

운영 설정

이 예시에서는 다음과 같은 운영 설정을 사용합니다.

- 클러스터 유형: 중요 클러스터
- 메쉬 유형: 클러스터 전반의 단일 테넌트
- 자동화: 시간이 지나면서 GitOps로 이동하는 스크립트
- 운영 모델: 셀프 서비스(제한적)
- DevSecOps: 도입됨

이제 엔터프라이즈 사용자 유형을 정의하고 서비스 메쉬 역할 및 책임을 해당 사용자 유형에 할당할 수 있습니다. 이 예시에서 각 역할의 **설정 스크립트**는 표에 링크되어 있으며 온라인 리소스 리포지토리에서도 제공됩니다.

표 1. 엔터프라이즈 사용자 유형, 서비스 메쉬 역할, 책임

엔터프라이즈 사용자 유형	서비스 메쉬 역할	책임
플랫폼 관리자	클러스터 관리자(기본 관리자 역할)	클러스터 관리자는 Red Hat OpenShift 클러스터를 소유 및 운영하고 조직의 정책을 설정합니다. 이들은 모두 클러스터 권한을 보유합니다. ▸ 클러스터 오퍼레이터 추가, 제거, 업데이트 ▸ 컨테이너 이미지를 이미지 레지스트리에 설치 ▸ OpenShift 버전 업데이트 ▸ 라우터, 네트워킹, 노드, 리소스 등 클러스터 인프라 및 구성 설정 ▸ 보안 설정 ▸ 트러블슈팅을 위해 로그를 조회할 수 있는 서비스 메쉬 리소스 제공
메쉬 오퍼레이터	**메쉬 오퍼레이터**	메쉬 오퍼레이터는 서비스 메쉬 컨트롤 플레인 네임스페이스를 운영합니다. 이들은 운영 설정에 따라 한 개 이상의 서비스 메쉬를 관리합니다. ▸ 소유한 네임스페이스 내의 서비스 메쉬 컨트롤 플레인, 구성원 역할, 구성원, Istio 리소스를 추가, 제거, 업데이트 ▸ 서비스 메쉬 관측성 스택 구성 ▸ 메쉬 보안 및 인증서 설정 ▸ north-south 및 east-west 트래픽을 위한 인그레스 및 이그레스 게이트웨이 구성
도메인 소유자(기술 리드)	**메쉬 개발자**	도메인 소유자는 서비스 메쉬 및 온보드 개발자와 상호 작용하는 애플리케이션의 내부 및 외부 종속성을 이해합니다. ▸ 도메인 기반 애플리케이션을 위한 환경 정의 ▸ 서비스 메쉬 데이터 플레인에 대한 Istio 구성 정의 ▸ 메쉬 오퍼레이터와 협력하여 인그레스 및 이그레스 트래픽과 리소스 구성

엔터프라이즈 사용자 유형	서비스 메쉬 역할	책임
개발자	**메쉬 애플리케이션 뷰어**(개발 환경)	개발자는 구성된 플랫폼, 메쉬, 관측성 스택을 사용해 애플리케이션의 기능과 성능을 검토하고 트러블슈팅합니다. ▸ 애플리케이션의 상태, 성능, 기능적 역량 모니터링 ▸ 개발 환경에서 Kiali 시각화, Jaeger 텔레메트리, Prometheus 메트릭, 포드 로그에 액세스
애플리케이션 오퍼레이터	**메쉬 개발자**(비개발 환경)	애플리케이션 오퍼레이터는 클러스터와 도메인 호스팅 메쉬 내에 배포된 애플리케이션을 모니터링하고 유지 관리합니다. ▸ 로그 및 envoy 프록시 구성, 텔레메트리, 추적에 액세스하여 도메인 내의 문제 검증 ▸ 정보를 추출하고 메쉬 오퍼레이터 및 개발자와 협력하여 문제 식별 ▸ 메쉬 오퍼레이터 및 개발자에게 Istio 구성 제안
제품 소유자	**메쉬 애플리케이션 뷰어**(비개발 환경)	제품 소유자는 관측성 스택을 사용해 메쉬 내부와 외부에서 제품을 구성하는 애플리케이션을 모니터링합니다. ▸ 도메인 내의 제품 상태, 사용량, 비용, 기타 메트릭 모니터링 ▸ 관측성 스택 정보 수집 후 최대 1주 동안 액세스

엔터프라이즈 사용자를 서비스 메쉬 역할에 매핑

사용자 유형, 역할, 책임을 정의하면 이를 팀원에게 매핑할 수 있습니다. 모범 사례에 따라 여기서는 사용자를 개발 및 프로덕션 환경 내부의 사용자 유형, 역할, 네임스페이스에 매핑합니다. 다음 표는 Travel by Keyboard를 위한 개발 및 프로덕션 환경 역할을 정리한 것입니다. 이 표에는 이 e-book 전체의 예시에서 사용될 사용자 이름과 비밀번호도 포함되어 있습니다. 예시에서는 편의상 각 사용자의 사용자 이름은 각자의 이름(모두 소문자)이고 비밀번호는 사용자 이름과 동일합니다.

예시에 사용된 로그인 정보

이 e-book에서는 특정 사용자 유형이 사용해야 하는 예시 코드, 스크립트, 인터페이스를 제공합니다. 예시 시나리오에 있는 각 사용자의 로그인 정보(사용자 이름 및 비밀번호)는 표 2 및 3에 나와 있습니다.

표 2. 개발 환경을 대한 사용자 매핑

이름	사용자 이름/비밀번호	엔터프라이즈 사용자 유형	서비스 메쉬 역할	네임스페이스
Phillip	`phillip/phillip`	플랫폼 관리자	클러스터 관리자	`dev-istio-system`
Emma	`emma/emma`	메쉬 오퍼레이터	메쉬 오퍼레이터	`dev-istio-system`
Cristina	`cristina/cristina`	여행 포털 도메인 소유자(기술 리드)	메쉬 개발자	`dev-travel-portal`, `dev-travel-control`
Farid	`farid/farid`	여행 서비스 도메인 소유자(기술 리드)	메쉬 개발자	`dev-travel-agency`
John	`john/john`	여행 포털 개발자	메쉬 애플리케이션 뷰어	`dev-travel-portal`, `dev-travel-control`

이름	사용자 이름/비밀번호	엔터프라이즈 사용자 유형	서비스 메쉬 역할	네임스페이스
Mia	`mia/mia`	여행 서비스 개발자	메쉬 애플리케이션 뷰어	`dev-travel-agency`
Mus	`mus/mus`	제품 소유자	메쉬 애플리케이션 뷰어	`dev-travel-portal`, `dev-travel-control`, `dev-travel-agency`

표 3. 프로덕션 환경에 대한 사용자 매핑

이름	사용자 이름/비밀번호	엔터프라이즈 사용자 유형	서비스 메쉬 역할	네임스페이스
Phillip	`phillip/phillip`	플랫폼 관리자	클러스터 관리자	`prod-istio-system`
Emma	`emma/emma`	메쉬 오퍼레이터	메쉬 오퍼레이터	`prod-istio-system`
Cristina	`cristina/cristina`	여행 포털 도메인 소유자(기술 리드)	메쉬 개발자	`prod-travel-portal`, `prod-travel-control`
Farid	`farid/farid`	여행 서비스 도메인 소유자(기술 리드)	메쉬 개발자	`prod-travel-agency`
Craig	`craig/craig`	애플리케이션 오퍼레이터(플랫폼 팀)	메쉬 개발자	`prod-travel-portal`, `prod-travel-control`, `prod-travel-agency`
Mus	`mus/mus`	제품 소유자	메쉬 애플리케이션 뷰어	`prod-travel-portal`, `prod-travel-control`, `prod-travel-agency`

서비스 메쉬 배포 결정

끝으로, 서비스 메쉬에서 사용될 배포 구성 요소를 식별해야 합니다. 표 4에서는 Travel by Keyboard의 개발 환경에서 사용될 구성 요소와 각 구성 요소의 인스턴스 수를 보여줍니다. 다음 장에서 배포 프로세스를 살펴보면서 최종 프로덕션 설정을 정의하겠습니다.

표 4. 개발 환경 서비스 메쉬 아키텍처 구성 요소

구성 요소	인스턴스 수
`grafana`	1
`istiod`	1
`istio-egressgateway`	1
`istio-ingressgateway`	1

구성 요소	인스턴스 수
`jaeger`	1
`kiali`	1
`prometheus`	1
`wasm-cacher-client-side-tenant`	1

2장

개발 환경 설정

이 장에서는 여행 포털과 여행사 팀을 위한 개발 환경을 설정합니다. 특정 역할이 각 태스크를 수행해야 합니다. 여기에선 플랫폼 관리자, 메쉬 오퍼레이터, 도메인 소유자가 환경을 설정합니다. 제품 소유자, 도메인 소유자, 개발자, 메쉬 오퍼레이터는 관측성 스택을 사용해 설정을 검증합니다.

이 장과 다음 장에 걸쳐 설명되는 각 단계에 예시 코드 및 스크립트가 제공됩니다. **추가 정보와 스크립트** 자체는 온라인 리소스 리포지토리에 포함되어 있습니다. →

개발 환경 설정

이 섹션에서는 개발 환경 설정에 관한 정보를 제공합니다. 이 예시에서는 Phillip, Emma, Farid, Cristina가 이 태스크를 수행합니다.

클러스터 로그인 URL 정의

플랫폼 관리자(클러스터 관리자 역할)인 Phillip은 전체 클러스터 및 플랫폼 설정을 담당합니다.

1. 사용자가 로그인할 수 있도록 클러스터 URL을 다음과 같이 정의합니다.

```
export CLUSTER_API=<YOUR-CLUSTER-API-URL>
```

서비스 메쉬 오퍼레이터, 네임스페이스, 역할, 사용자 준비

플랫폼 관리자(클러스터 관리자 역할)인 Phillip은 개발 환경 내에서 오퍼레이터, 네임스페이스, 역할을 설정하는 업무도 담당합니다.

1. 다음과 같이 개발 설정 디렉터리로 이동한 후 Phillip으로 로그인합니다.

```
cd ossm-heading-to-production-and-day-2/scenario-2-dev-setup
./login-as.sh phillip
```

2. 다음과 같이 **Red Hat OpenShift 마켓플레이스**를 통해 Red Hat OpenShift Service Mesh 오퍼레이터를 클러스터에 추가합니다.

```
../common-scripts/add-operators-subscriptions-sm.sh
```

3. 다음과 같이 필요한 개발 여행사 네임스페이스를 생성합니다.

```
../common-scripts/create-travel-agency-namespaces.sh dev
```

4. 1장에서 정의한 **서비스 메쉬 역할**을 생성합니다.

5. **개발 환경 서비스 메쉬 사용자**를 생성하고 1장에서 정의한 역할을 할당합니다.

서비스 메쉬 제어 네임스페이스 생성

메쉬 오퍼레이터인 Emma는 개발 환경 내에서 컨트롤 플레인 네임스페이스와 `ServiceMeshControlPlane`(또는 `smcp`) 리소스 생성을 담당합니다.

1. 개발 환경 내에서 다음과 같이 `dev-basic` 서비스 메쉬 컨트롤 플레인을 생성합니다.

```
./login-as.sh emma
./scripts/create-dev-smcp.sh dev-istio-system dev-basic
```

서비스 메쉬 멤버십 구성과 애플리케이션 배포

도메인 소유자(메쉬 개발자 역할)인 Farid와 Cristina는 네임스페이스를 서비스 메쉬의 구성원으로 등록하고 개발 환경 내에서 애플리케이션을 배포하는 업무를 담당합니다.

여행 서비스 도메인(`dev-travel-agency` 네임스페이스)

Farid는 여행 서비스 도메인을 소유하고 이 도메인에 애플리케이션을 메쉬 개발자로 배포합니다.

1. 서비스 메쉬 멤버십을 구성하기 전에 다음과 같이 프로젝트 레이블을 확인합니다.

```
./login-as.sh farid
../common-scripts/check-project-labels.sh dev-travel-agency
```

2. 다음과 같이 `ServiceMeshMember` 리소스를 추가함으로써 `dev-travel-agency` 네임스페이스에 대한 멤버십을 `dev-basic` 서비스 메쉬 컨트롤 플레인에 추가합니다.

```
../common-scripts/create-membership.sh dev-istio-system dev-basic dev-travel-agency
../common-scripts/check-project-labels.sh dev-travel-agency
```

3. 다음과 같이 애플리케이션을 `dev-travel-agency` 네임스페이스에 배포합니다.

```
./scripts/deploy-travel-services-domain.sh dev dev-istio-system
```

여행 포털 도메인(`dev-travel-portal` 및 `dev-travel-control` 네임스페이스)

Cristina는 여행 포털 도메인을 소유하고 이 도메인에 애플리케이션을 메쉬 개발자로 배포합니다.

1. 서비스 메쉬 멤버십을 구성하기 전에 다음과 같이 프로젝트 레이블을 확인합니다.

```
./login-as.sh cristina
../common-scripts/check-project-labels.sh dev-travel-control
../common-scripts/check-project-labels.sh dev-travel-portal
```

2. 다음과 같이 각각에 대해 `ServiceMeshMember` 리소스를 추가함으로써 `dev-travel-control` 및 `dev-travel-portal` 네임스페이스에 대한 멤버십을 `dev-basic` 서비스 메쉬 컨트롤 플레인에 추가합니다.

```
../common-scripts/create-membership.sh dev-istio-system dev-basic dev-travel-control
../common-scripts/check-project-labels.sh dev-travel-control
../common-scripts/create-membership.sh dev-istio-system dev-basic dev-travel-portal
../common-scripts/check-project-labels.sh dev-travel-portal
```

3. 다음과 같이 애플리케이션을 `dev-travel-control` 및 `dev-travel-portal` 네임스페이스에 배포합니다.

```
./scripts/deploy-travel-portal-domain.sh dev dev-istio-system
```

여행사 포털에 연결하기 위한 게이트웨이 생성

메쉬 오퍼레이터인 Emma는 여행사 포털에 대한 연결을 노출하기 위해 Istio **게이트웨이** 리소스 생성을 담당합니다.

1. Istio **게이트웨이** 리소스를 다음과 같이 생성합니다.

```
./login-as.sh emma
./scripts/create-ingress-gateway.sh dev-istio-system
```

개발 관측성 스택 사용 및 설정 확인

이 섹션에서는 관측성 스택을 사용해 개발 환경의 설정을 확인하는 방법에 관한 정보를 제공합니다. 또한 각 역할이 관측성 스택을 사용하는 방법에 관한 정보도 제공합니다. **관측성 스택을 각 역할로 사용하기 위한 실습**과 기대되는 성과의 스크린샷은 온라인 리소스 리포지토리에 자세히 나와 있습니다. →

여행 제어 대시보드에 액세스

클러스터 관리자, 메쉬 오퍼레이터, 메쉬 개발자 역할(이 예시에서는 Phillip, Emma, Cristina, Farid)은 다음과 같이 여행 제어 대시보드의 URL 캡처를 담당합니다.

```
./login-as.sh <choose user>
echo "http://$(oc get route istio-ingressgateway -o jsonpath='{.spec.host}' -n dev-istio-system)"
```

그런 다음, 이 URL을 웹 브라우저에 입력하여 여행 제어 대시보드에 액세스할 수 있습니다.

관측성 스택에 액세스

클러스터 관리자, 메쉬 오퍼레이터, 메쉬 개발자 역할(이 예시에서는 Phillip, Emma, Cristina, Farid)은 다음과 같이 Kiali, Jaeger, Prometheus, Grafana 구성 요소 등 관측성 스택의 URL을 캡처할 수 있습니다.

```
./login-as.sh <choose user>
echo "http://$(oc get route kiali -o jsonpath='{.spec.host}' -n dev-istio-system)"
echo "https://$(oc get route jaeger -o jsonpath='{.spec.host}' -n dev-istio-system)"
echo "https://$(oc get route prometheus -o jsonpath='{.spec.host}' -n dev-istio-system)"
echo "https://$(oc get route grafana -o jsonpath='{.spec.host}' -n dev-istio-system)"
```

창 상단의 이름 왼쪽에 있는 물음표를 클릭하여 Kiali 대시보드에서 Grafana 및 Jaeger URL에 액세스한 다음, 팝업 메뉴에서 *정보*를 선택할 수도 있습니다. **인터페이스**를 확인하세요. →

관측성 스택을 제품 소유자로 사용

제품 소유자(메쉬 애플리케이션 뷰어의 역할)인 Mus는 Kiali 대시보드를 통해 세 가지 데이터 플레인 네임스페이스(`dev-travel-portal`, `dev-travel-control`, `dev-travel-agency`)와 컨트롤 플레인 네임스페이스에 액세스할 수 있습니다. Mus의 **Kiali 대시보드**를 확인하세요. →

제품 소유자 메쉬 애플리케이션 뷰어인 경우 다음 작업을 수행할 수 있습니다.

- **전체 솔루션에 대한 추적 확인.** 대시보드 왼쪽의 Kiali 메뉴에서 *분산 추적*을 선택한 후 자격 증명으로 로그인하여 추적 콘솔에 액세스합니다.
- **전체 솔루션에 대한 메트릭 확인.** *워크로드* 섹션에서 애플리케이션 워크로드를 선택하여 인바운드 및 아웃바운드 메트릭을 확인합니다. 이전 섹션의 URL을 사용해 Prometheus 대시보드에 직접 로그인할 수도 있습니다. 다음과 같은 커맨드를 그래프 보기에 적용할 수 있습니다.
  - `istio_requests_total{destination_workload="discounts-v1", \app="discounts"}`
  - `istio_request_duration_milliseconds_count{app="discounts"}`
  - `istio_response_bytes_bucket`
- **전체 솔루션에 대한 Grafana 대시보드 보기.** 자격 증명을 사용해 Grafana URL에 로그인합니다. *대시보드* > *관리* > *Istio* > *Istio Mesh Dashboard*를 선택하여 여행사 솔루션의 상태 및 성능을 확인할 수 있습니다. **Istio 메쉬 대시보드** 및 **성능 보기**를 확인하세요. →

제품 소유자인 경우 Istio 구성을 볼 수는 있지만 수정할 수는 없습니다. Istio 로그는 볼 수 없습니다. 왼쪽 메뉴에서 *Istio 구성* 및 *워크로드* 옵션을 사용해 구성 또는 로그 보기를 시도하여 역할 기반 액세스 제어가 올바르게 구성되어 있는지 확인할 수 있습니다.

관측성 스택을 도메인 소유자로 사용

도메인 소유자(메쉬 개발자 역할)인 Cristina와 Farid는 각자의 해당 도메인 네임스페이스에 액세스할 수 있습니다. 이 예시에서 Cristina는 여행사 포털 도메인과 컨트롤 플레인 네임스페이스에서 두 개의 데이터 플레인 네임스페이스(`dev-travel-control` 및 `dev-travel-portal`)에 액세스할 수 있습니다. Farid는 여행 서비스 도메인과 컨트롤 플레인 네임스페이스에서 하나의 데이터 플레인 네임스페이스(`dev-travel-agency`)에 액세스할 수 있습니다. **Cristina** 및 **Farid**의 Kiali 대시보드를 확인하세요. →

도메인 소유자 메쉬 개발자인 경우 다음 작업을 수행할 수 있습니다.

- **전체 솔루션에 대한 추적 확인.** 대시보드 왼쪽의 Kiali 메뉴에서 *분산 추적*을 선택한 후 자격 증명으로 로그인하여 추적 콘솔에 액세스합니다.
- **전체 솔루션에 대한 메트릭 확인.** 이전 섹션의 URL을 사용해 Prometheus 대시보드에 로그인합니다. 다음과 같은 커맨드를 그래프 보기에 적용할 수 있습니다.
  - `istio_requests_total{destination_workload="discounts-v1", \app="discounts"}`
  - `istio_request_duration_milliseconds_count{app="discounts"}`
  - `istio_response_bytes_bucket`
- **도메인에서 워크로드에 대한 로그 확인.** 왼쪽 메뉴의 *워크로드* 섹션에서 워크로드를 선택합니다. *로그* 탭에 프록시 및 포드 로그가 표시됩니다. **로그 보기**를 확인하세요. →
- **도메인에서 Istio 구성 확인 및 수정.** 왼쪽 메뉴에서 *Istio 구성*을 선택하여 역할에 사용 가능한 구성을 확인합니다. **Istio 구성 보기**를 확인하세요. →
- **Grafana 대시보드 보기.** *대시보드* > *관리* > *Istio* > *Istio 서비스 대시보드* 또는 *대시보드* > *관리* > *Istio* > *Istio 워크로드 대시보드*를 선택하여 도메인에서 서비스 또는 워크로드의 상태를 각각 확인합니다. **Istio 서비스**와 **Istio 워크로드 대시보드**를 확인하세요. →

관측성 스택을 개발자로 사용

개발자(메쉬 애플리케이션 뷰어 역할)인 Mia와 John은 허용된 네임스페이스 내에서 추적, 메트릭, 대시보드에 액세스할 수 있습니다. 이 예시에서 Mia는 여행 서비스 도메인의 개발자이며 `dev-travel-agency` 네임스페이스에 관한 정보에 액세스할 수 있습니다. 여행 포털 도메인의 개발자인 John은 `dev-travel-control` 및 `dev-travel-portal` 네임스페이스에 관한 정보에 액세스할 수 있습니다. Mia와 John의 Kiali 대시보드를 확인하세요. →

개발자 메쉬 애플리케이션 뷰어인 경우 다음 작업을 수행할 수 있습니다.

- **도메인에서 워크로드에 대한 추적 확인.** 대시보드 왼쪽의 Kiali 메뉴에서 *분산 추적*을 선택한 후 자격 증명으로 로그인하여 추적 콘솔에 액세스합니다.
- **도메인에서 워크로드에 대한 메트릭 확인.** *워크로드* 섹션에서 애플리케이션 워크로드를 선택하여 인바운드 및 아웃바운드 메트릭을 확인합니다.
- **전체 솔루션에 대한 Grafana 대시보드 보기.** 자격 증명을 사용해 Grafana URL에 로그인합니다. *대시보드* > *관리* > *Istio* > *Istio Mesh Dashboard*를 선택하여 여행사 솔루션의 상태 및 성능을 확인할 수 있습니다.

개발자인 경우 Istio 구성을 볼 수는 있지만 수정할 수는 없습니다. Istio 로그는 볼 수 없습니다. 왼쪽 메뉴에서 *Istio 구성* 및 *워크로드* 옵션을 사용해 구성 세부 정보 또는 로그 보기를 시도하여 역할 기반 액세스 제어가 올바르게 구성되어 있는지 확인할 수 있습니다.

관측성 스택을 메쉬 오퍼레이터로 사용

메쉬 오퍼레이터인 Emma는 세 가지 데이터 플레인 네임스페이스(`dev-travel-control`, `dev-travel-portal`, `dev-travel-agency`)와 컨트롤 플레인 네임스페이스에 대한 전체 액세스 권한이 있습니다. Emma의 Kiali 대시보드를 확인하세요. →

메쉬 오퍼레이터인 경우 다음 작업을 수행할 수 있습니다.

- **Kiali에서 모든 네임스페이스 보기.** Kiali 메뉴에서 *그래프* > *앱 그래프* > *표시*를 선택하여 요청 분산, 네임스페이스 상자, 트래픽 애니메이션, 보안 설정을 표시합니다.
- **모든 워크로드에 대한 로그 보기.** 왼쪽 메뉴의 *워크로드* 옵션에서 워크로드를 선택합니다. *로그* 탭에 프록시 및 포드 로그가 표시됩니다.
- **Istio 구성 확인 및 수정.** 왼쪽 메뉴에서 *Istio 구성*을 선택해 모든 구성에 액세스하고 수정합니다.
- **대시보드 보기.** 이 장의 앞 부분에서 정의한 URL을 통해 Prometheus, Jaeger, Grafana 대시보드에 액세스합니다. *대시보드* > *관리* > *Istio* > *Istio 컨트롤 플레인 대시보드*를 선택하여 서비스 메쉬 컨트롤 플레인의 상태를 시각화합니다. **Istio 컨트롤 플레인 대시보드**를 확인하세요. →

3장

프로덕션 환경 설정

이 장에서는 1장에서 설명한 요구 사항에 따라 여행 포털 및 여행사 팀의 프로덕션 환경을 설정합니다. 개발 환경과 마찬가지로 특정 역할이 각 태스크를 수행해야 합니다. 여기에선 플랫폼 관리자와 메쉬 오퍼레이터가 프로덕션 환경을 설정하고 구성합니다. 다른 엔터프라이즈 사용자 유형과 서비스 메쉬 역할은 전과 같이 관측성 스택의 지정된 부분에 액세스합니다.

프로덕션 환경 설정

이 섹션에서는 프로덕션 환경을 설정하는 방법에 대해 설명합니다. 이 예시에서는 Phillip과 Emma가 이 태스크를 수행합니다.

클러스터 로그인 URL 정의

개발 환경과 마찬가지로 플랫폼 관리자(클러스터 관리자 역할)인 Phillip은 전체 프로덕션 클러스터 및 플랫폼 설정을 담당합니다.

1. 사용자가 로그인할 수 있도록 클러스터 URL을 다음과 같이 정의합니다.

```
export CLUSTER_API=<YOUR-CLUSTER-API-URL>
```

서비스 메쉬 오퍼레이터, 네임스페이스, 역할, 사용자 준비

플랫폼 관리자(클러스터 관리자 역할)인 Phillip은 개발 환경 내에서 오퍼레이터, 네임스페이스, 역할 설정을 담당합니다. 개발과 프로덕션을 위한 별도의 Red Hat OpenShift 클러스터가 없는 경우 Red Hat OpenShift Service Mesh 오퍼레이터를 추가(2단계)할 필요가 없습니다. 자세한 내용은 1장의 **엔터프라이즈 사용자를 서비스 메쉬 역할에 매핑** 섹션을 참조하세요.

1. 다음과 같이 프로덕션 설정 디렉터리로 이동한 후 Phillip으로 로그인합니다.

```
cd ossm-heading-to-production-and-day-2/scenario-2-dev-setup
./login-as.sh phillip
```

중요: **별도의 개발 및 프로덕션 클러스터가 없는 경우 2단계와 3단계를 건너뛰세요.**

2. 다음과 같이 **Red Hat OpenShift 마켓플레이스**를 통해 Red Hat OpenShift Service Mesh 오퍼레이터를 클러스터에 추가합니다.

```
../common-scripts/add-operators-subscriptions-sm.sh
```

3. 1장에서 정의한 **프로덕션 서비스 메쉬 역할**을 생성합니다.

4. 다음과 같이 필요한 프로덕션 여행사 네임스페이스를 생성합니다.

```
../common-scripts/create-travel-agency-namespaces.sh prod
```

5. **프로덕션 환경 서비스 메쉬 사용자**를 생성하고 1장에서 정의한 역할을 할당합니다.

프로덕션을 위한 Jaeger, Elasticsearch, 서비스 메쉬 컨트롤 플레인 구성

메쉬 오퍼레이터인 Emma는 프로덕션 환경 내에서 서비스 메쉬 컨트롤 플레인 및 추적 구성을 담당합니다. 이러한 태스크를 수행하기 위한 두 가지 옵션이 있습니다.

- 옵션 1: **컨트롤 플레인을 통한 최소 Elasticsearch 매개 변수**
- 옵션 2: **Jaeger 구성을 통해 전체 사용자 정의된 Elasticsearch 매개 변수**

이 예시에서는 옵션 2를 사용해 서비스 메쉬 컨트롤 플레인 및 추적 배포를 전체 사용자 정의합니다. 외부 Jaeger 리소스 구성에 관한 추가 정보는 다음 Jaeger 도큐멘테이션을 참조하세요.

- **사용자 정의 리소스 정의에 대한 이해**
- **Elasticsearch 스토리지가 포함된 jaeger-collector에 대한 커맨드라인 인터페이스(CLI) 플래그**

중요: **예제 스크립트를 사용하려면 경로에 OpenSSL이 설치되어 있고 액세스할 수 있어야 합니다.**

1. Jaeger 구성:

```
./login-as.sh emma
./scripts/create-prod-smcp-1-tracing.sh prod-istio-system production
```

이 스크립트는 `prod-istio-system` 네임스페이스에서 다음과 같은 설정으로 Jaeger 리소스를 생성합니다.

- **프로덕션 중심 설정**
- **퍼시스턴트 스토리지용 Elasticsearch**
- **7일 후 점진적으로 삭제되는 인덱스**
- **한 개의 Elasticsearch 노드**
- **1Gi의 Elasticsearch 인덱스 크기**
- **노드 리소스 요청 및 제한**
- **Elasticsearch 노드 이중화 없음**

2. 다음과 같이 Jaeger 리소스가 생성되었는지 확인합니다.

```
oc get jaeger/jaeger-small-production -n prod-istio-system
```

3. 다음과 같이 모든 구성 요소(Jaeger Collector, Jaeger 쿼리, Elasticsearch 배포)가 생성되었는지 확인합니다.

```
oc get deployment -n prod-istio-system
NAME                                                  READY   UP-TO-DATE   AVAILABLE   AGE
elasticsearch-cdm-prodistiosystemjaegersmallproduction-1   1/1     1            1           7m27s
jaeger-small-production-collector                     1/1     1            1           7m25s
jaeger-small-production-query                         1/1     1            1           7m25s
```

또한 `create-prod-smcp-1-tracing.sh` 스크립트는 1장에서 정의한 **프로덕션 요구 사항**에 따라 서비스 메쉬를 구성합니다. **전체 YAML 파일**을 참조하세요. →

- **모든 추적의 20%**를 샘플링을 위해 수집
- 등록되지 않은 호스트에 대한 **외부 발신 통신은 허용되지 않음**
- 서비스 메쉬 내부의 Jaeger 리소스에 저장된 추적
- Elasticsearch를 통해 추적 지속성 제공
- **외부 Jaeger 리소스** 통합 및 사용

데이터 플레인을 구성하고 애플리케이션을 프로덕션으로 배포

도메인 소유자(메쉬 개발자 역할)인 Farid와 Cristina는 데이터 플레인을 구성하고 애플리케이션을 프로덕션으로 배포하는 업무를 담당합니다. 이 예시에서는 서비스 메쉬에 다음 항목이 포함되어 있습니다.

- 주 애플리케이션 컨테이너 안팎으로의 모든 통신을 가로채고 서비스 메쉬 구성을 적용하기 위한 `istio-proxy` 사이드카 컨테이너
- Red Hat OpenShift 클러스터에서 멀티테넌시를 허용하기 위한 `jaeger-agent` 사이드카 컨테이너 자세한 내용은 **Jaeger 배포 모범 사례** 도큐멘테이션을 참조하세요.

다음과 같이 `one-step-add-prod-deployments.sh` 스크립트를 통해 Farid와 Cristina는 여행사 및 여행 서비스 도메인에 애플리케이션을 배포할 수 있습니다.

```
./scripts/one-step-add-prod-deployments.sh <OCP CLUSTER DOMAIN eg. apps.example.com>
```

애플리케이션을 프로덕션에 단계적으로 배포하는 방식을 선호하는 경우 온라인 리소스 리포지토리에서 **프로덕션 환경 설정** 섹션을 참조하세요.

Istio 게이트웨이 액세스 및 보안 인증서 구성

메쉬 오퍼레이터인 Emma는 여행사 대시보드에 대한 액세스를 허용하도록 Istio **게이트웨이** 리소스 및 보안 인증서를 구성하는 업무를 담당합니다. 이 예시에서는 **경로** 리소스를 사용하여 TLS(Transport Layer Security)를 통해 여행사 대시보드를 노출합니다. 인증서는 인그레스 게이트웨이의 `control-gateway` 리소스를 통해 호스팅되며, **경로** 리소스는 패스스루 모드로 설정됩니다. 이로써 노출된 각 서비스에 대해 별도의 인증서를 정의할 수 있지만 메쉬 오퍼레이터 역할이 각 인증서를 별도로 교체해야 하므로 추가 유지 관리도 필요합니다.

경로 및 **게이트웨이** 리소스와 인증서를 다음과 같이 생성합니다.

```
./login-as.sh emma
./scripts/create-https-ingress-gateway.sh prod-istio-system /
  <OCP CLUSTER DOMAIN eg. apps.example.com>
```

그 결과로 나타나는 **여행 제어 대시보드**를 확인하세요. →

프로덕션을 위해 Prometheus 구성

메쉬 오퍼레이터인 Emma는 프로덕션 환경을 위해 Prometheus 모니터링 구성을 담당합니다. Red Hat OpenShift는 관측성 스택을 제공하지만 Red Hat OpenShift Service Mesh의 현재 버전은 이 스택과 통합되거나 페더레이션되지 않습니다. 이를 해결하기 위한 몇 가지 옵션은 다음과 같습니다.

- **옵션 1:** 퍼시스턴트 볼륨을 추가하여 기존 Prometheus 배포 강화
- **옵션 2:** **Prometheus 오퍼레이터를 사용해 외부 Prometheus 배포 생성**
- **옵션 3:** Red Hat OpenShift 모니터링 스택과의 통합을 통해서만 데이터 플레인 메트릭 수집
- **옵션 4**: 외부 모니터링 툴을 구성하고 사용해 메트릭 수집

이 예시에서는 다음과 같이 옵션 1을 사용해 메트릭 유지를 7일로 연장하고 장기적인 메트릭 스토리지를 위해 퍼시스턴트 볼륨을 추가합니다.

```
./login-as.sh emma
./scripts/update-prod-smcp-2-option1-prometheus.sh prod-istio-system
```

기타 세 가지 옵션에 대한 **추가 정보**는 온라인 리소스 리포지토리에서 확인할 수 있습니다. →

최종 서비스 메쉬 프로덕션 설정

이 섹션에서는 추적, 메트릭, 확장된 컨트롤 플레인 구성 요소, 런타임 리소스 할당 등 최종적인 프로덕션 설정을 제공합니다. 1장에서 정의한 목표, 원칙, 요구 사항은 최종 설정을 안내하고 서비스 메쉬 IT 리소스 및 자산의 배포와 사용에 대한 일반적인 규칙 및 지침을 확립할 수 있는 시작점의 역할을 합니다. 이 예시에서 여행사 아키텍트는 다음과 같은 서비스 메쉬 목표 및 아키텍처 원칙을 다시 검토하여 확정했습니다.

서비스 메쉬 목표

- 서비스 간 통신의 보안 유지
- 서비스 간 통신의 사용 및 상태 모니터링
- 팀이 개별적으로 작업하여 솔루션의 여러 부분을 제공하도록 허용

서비스 메쉬 아키텍처 원칙

- 트래픽 암호화, 인증, 권한 부여를 위한 외부 구성 메커니즘 사용
- 추가 서비스를 투명하게 통합하여 기능성 확장
- 외부 트래픽 관리 및 오케스트레이션 메커니즘 사용
- 고가용성을 염두에 두고 모든 구성 요소를 구성
- 감사보다는 시스템 운영을 확인하기 위해 관측성 기능 사용

최종 프로덕션 서비스 메쉬 설정

이러한 목적 및 원칙에 따른 최종 프로덕션 서비스 메쉬 설정은 다음과 같습니다.

- **추적:** 모든 추적의 5%가 디버그 목적으로 샘플링되어 Elasticsearch 클러스터에 7일 동안 저장됩니다.
- **메트릭:** 수집된 메트릭은 7일 동안 저장되며, 이력을 비교할 수 있도록 향후 아카이빙됩니다.
- **Grafana:** Grafana는 분석을 위해 퍼시스턴트 스토리지를 사용합니다.
- **인그레스 및 이그레스:** 고가용성을 위해 각기 두 개의 인그레스 및 이그레스 포드 인스턴스가 배포됩니다.
- **`Istiod`:** 고가용성을 위해 두 개의 Istiod 인스턴스가 배포됩니다.

메쉬 오퍼레이터인 Emma는 확정된 요구 사항을 충족하기 위해 다음과 같이 프로덕션 배포를 업데이트할 책임이 있습니다.

```
./login-as.sh emma
./scripts/update-prod-smcp-3-final.sh prod-istio-system production
```

프로덕션 관측성 스택 사용 및 설정 확인

개발 환경과 마찬가지로 관측성 스택을 사용해 프로덕션 환경 설정을 확인해야 합니다. 설정 확인과 각 역할이 스택을 사용하는 방식에 관한 자세한 내용은 2장의 **개발 관측성 스택 사용 및 설정 확인** 섹션을 참조하세요. 4장에서는 관측성 스택을 사용해 서비스 메쉬 성능을 튜닝하고 이에 따라 서비스 메쉬 구성 요소의 규모를 조정하는 방법에 관한 세부 정보를 제공합니다.

4장

외부 여행 포털 온보딩

이 장에서는 외부 여행 포털을 Travel by Keyboard의 환경에 연결하여 새로운 비즈니스 기회를 제공합니다.

새로운 기회에 대한 요구 사항 정의

Travel by Keyboard의 비즈니스 개발 팀은 외부 여행 조직인 Global Travel Organization(GTO)과의 비즈니스 거래를 시작했습니다. GTO는 Travel by Keyboard의 여행 서비스 도메인에서 오퍼를 제공할 것입니다. 다음과 같은 기술 요구 사항이 적용됩니다.

- GTO 여행 포털은 API를 통해 Travel by Keyboard의 여행 서비스 도메인에 연결됩니다.
- 외부 클라이언트와의 모든 통신은 mTLS를 통해 수행됩니다.
- 유효한 JSON Web Token(JWT)을 사용하는 인증으로 향후 추가 권한 부여 정책을 지원합니다.

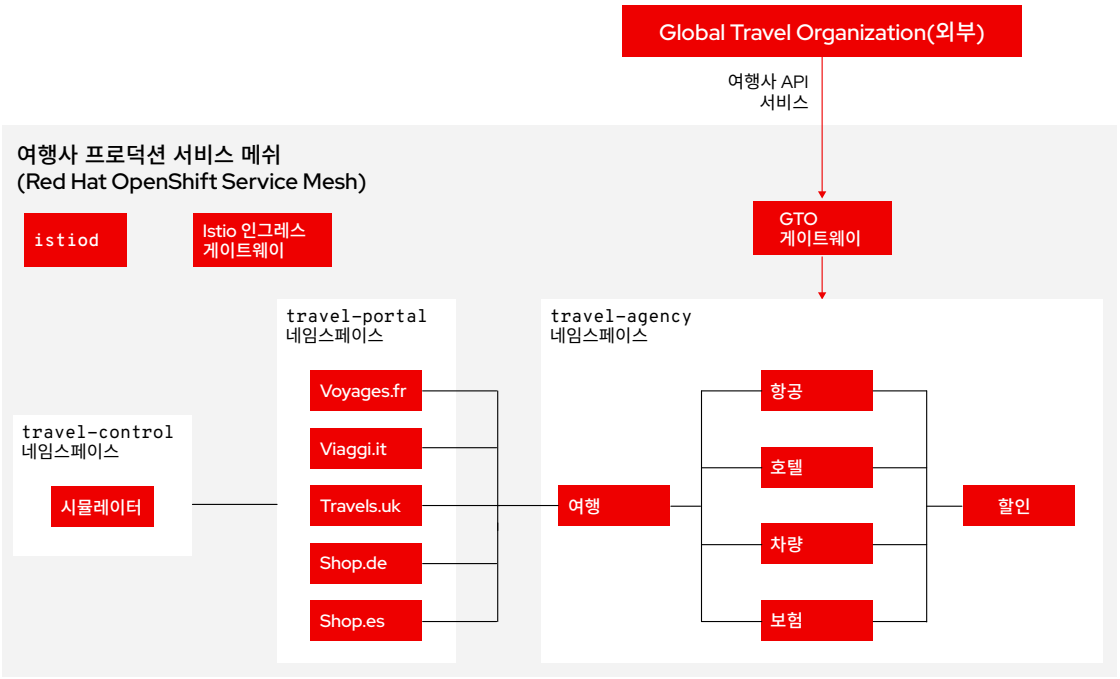


그림 3. API를 통해 GTO 여행 포털에 접속할 수 있는 기능이 포함된 Travel by Keyboard의 타겟 서비스 메쉬 아키텍처

API 서비스를 위한 추가 인그레스 생성

메쉬 오퍼레이터인 Emma는 mTLS를 통해 외부에서 여행사 서비스에 액세스할 수 있도록 추가 Istio 인그레스 게이트웨이 및 인증서 생성을 담당합니다.

1. 사용자가 로그인할 수 있도록 클러스터 URL을 다음과 같이 정의합니다.

```
export CLUSTER_API=<YOUR-CLUSTER-API-URL>
```

2. 다음과 같이 Istio **경로** 리소스, CA 루트 키 및 인증서, 클라이언트/서버 인증서, **게이트웨이** 리소스를 생성합니다.

```
cd ossm-heading-to-production-and-day-2/scenario-4-onboard-new-portal-with-authentication
./login-as.sh emma

# Add to SMCP production resource in prod-istio-system
    additionalIngress:
      gto-external-ingressgateway:
        enabled: true
        runtime:
          deployment:
            autoScaling:
              enabled: false
        service:
          metadata:
            labels:
              app: gto-external-ingressgateway
          selector:
            app: gto-external-ingressgateway

./scripts/create-external-mtls-https-ingress-gateway.sh prod-istio-system \
  <OCP CLUSTER DOMAIN e.g. apps.example.com>
```

이로써 새로운 `gto-external-ingressgateway` 포드, 액세스에 대한 TLS 패스스루 **경로** 리소스, `travel-api-gateway` Istio 구성이 생성됩니다. `travel-api-gateway` Istio 구성은 mTLS 요구 사항과 필요한 인증서 및 키가 포함된 암호를 정의합니다.

스크립트 출력 확인: **경로**, **포드**, **Istio 구성**, **YAML** →

새로운 Istio 구성 배포

여행 서비스 도메인 소유자(메쉬 개발자 역할)인 Farid는 새로운 게이트웨이의 요청이 서비스 메쉬의 **차량**, **보험**, **항공편**, **호텔**, **여행** 서비스에 도달하도록 새로운 `travel-api VirtualService` Istio 구성을 `prod-travel-agency` 네임스페이스로 배포하는 업무를 담당합니다.

```
./login-as.sh farid
./scripts/create-client-certs-keys.sh curl-client
./scripts/deploy-external-travel-api-mtls-vs.sh prod prod-istio-system
curl -v -X GET --cacert ca-root.crt --key curl-client.key --cert curl-client.crt \
  https://gto-external-prod-istio-system.apps.<CLUSTERNAME>.<DOMAINNAME>/flights/Tallinn
```

이로써 새로 생성된 `gto-external-ingressgateway`를 통해 GTO 클라이언트와 여행사 API 간에 mTLS 핸드셰이크가 생성됩니다.

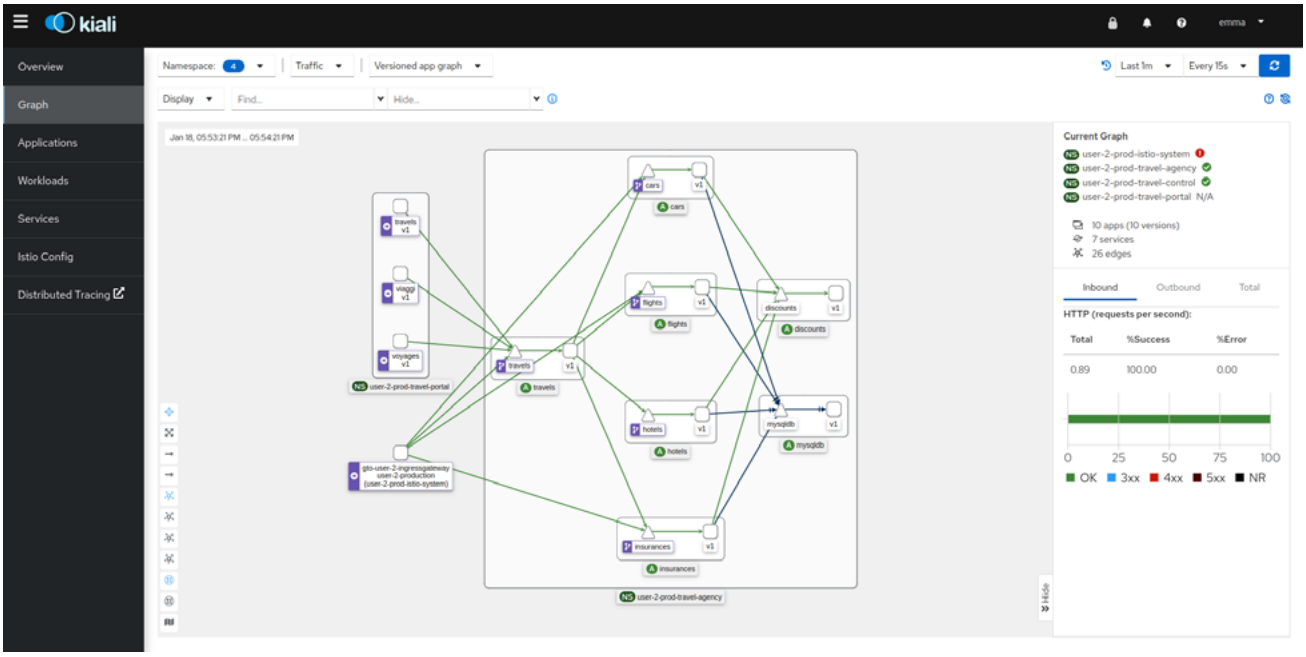


그림 4. 새로운 Istio 구성의 Kiali 시각화

JWT 인증 활성화

GTO 요청에 대한 의도된 최종 인증 워크플로우는 다음과 같이 mTLS 핸드셰이크와 JWT를 모두 사용합니다.

1. 사용자는 Red Hat Single Sign-On으로 인증하고 JWT를 수신합니다.
2. 사용자는 `https://<route>/<service>`에 대한 요청을 시작하고 이 요청과 함께 JWT를 전달합니다. 서비스는 **차량**, **보험**, **항공편**, **호텔** 또는 **여행**일 수 있습니다.
3. `gto-external-ingressgateway` 포드의 `istio-proxy` 컨테이너는 `RequestAuthentication` 오브젝트가 정의한 JWT 토큰과 `AuthorizationPolicy` 오브젝트에 정의된 권한의 유효성을 확인합니다.
4. JWT가 유효한 경우 사용자는 경로에 액세스할 수 있습니다. 유효하지 않다면 오류 메시지가 사용자에게 반환됩니다.

이 접근 방식은 두 개의 사용자 정의 리소스(CR)만 배포하면 되므로 단순하며 JWT 필드에 따라 세부적인 권한 부여를 제공합니다. 하지만 OpenID Connect 워크플로우를 사용하지 않으므로 사용자는 JWT 자체에서 획득하고 전달해야 합니다. 또한 서비스 메쉬 내부의 보호되는 각 애플리케이션에 대해 `RequestAuthentication` 및 `AuthorizationPolicy` 오브젝트를 정의해야 합니다.

Red Hat Single Sign-On 설정

플랫폼 관리자(클러스터 관리자 역할)인 Phillip은 Red Hat Single Sign-On 서버를 설정하고 TLS 인증서를 `istiod`에 마운트하는 업무를 담당합니다.

1. Single Sign-On 서버에 대한 전제 조건을 다음과 같이 구성합니다.

```
./login-as.sh phillip
./prerequisites-setup.sh <CLUSTERNAME> <BASEDOMAIN> (eg.for apps.ocp4.example.com \
  prerequisites-setup.sh ocp4 example.com)
```

2. Red Hat Single Sign-On TLS 인증서를 `istiod`에 다음과 같이 마운트합니다.

```
./scripts/mount-rhsso-cert-to-istiod.sh prod-istio-system production <CLUSTERNAME> <BASEDOMAIN>
```

게이트웨이에 대한 인증 및 권한 부여 설정

메쉬 오퍼레이터인 Emma는 `gto-external-ingressgateway`에 액세스할 수 있도록 `RequestAuthentication` 및 `AuthorizationPolicy` 오브젝트 생성을 담당합니다. `RequestAuthentication` 오브젝트의 경우 인증 및 권한 부여 요청은 Red Hat Single Sign-On에서 발급한 JWT만 사용해야 하는 반면, `AuthorizationPolicy` 오브젝트의 경우 Red Hat Single Sign-On에서 발급할 JWT가 유효한 것이어야 합니다.

```
./login-as.sh emma
oc -n prod-istio-system apply -f approach_1/yaml/istio/jwt/01_requestauthentication.yaml
oc -n prod-istio-system apply -f \
  approach_1/yaml/istio/jwt/02_authpolicy_allow_from_servicemesh-lab_realm.yaml
```

이를 위해서는 `gto-external` 경로 및 `gto-external-ingressgateway`를 통해 수신되는 모든 요청에 TLS 인증서와 유효한 JWT가 포함되어 계속 진행하도록 허용되어야 합니다.

JWT를 사용해 여행사 API 액세스 테스트

JWT를 사용해 인증 및 권한 부여를 설정했으면 테스트를 실시하여 모든 것이 기대한 대로 작동하는지 확인해야 합니다.

1. 다음과 같이 유효한 JWT가 없으면 여행사 서비스에 액세스할 수 없는지 확인합니다.

```
export GATEWAY_URL=$(oc -n prod-istio-system get route gto-external -o jsonpath='{.spec.host}')
curl -v -X GET --cacert ca-root.crt --key curl-client.key --cert curl-client.crt \
  https://$GATEWAY_URL/cars/Tallinn |jq
curl -v -X GET --cacert ca-root.crt --key curl-client.key --cert curl-client.crt \
  https://$GATEWAY_URL/travels/Tallinn |jq
curl -v -X GET --cacert ca-root.crt --key curl-client.key --cert curl-client.crt \
  https://$GATEWAY_URL/flights/Tallinn |jq
curl -v -X GET --cacert ca-root.crt --key curl-client.key --cert curl-client.crt \
  https://$GATEWAY_URL/insurances/Tallinn |jq
curl -v -X GET --cacert ca-root.crt --key curl-client.key --cert curl-client.crt \
  https://$GATEWAY_URL/hotels/Tallinn |jq

HTTP/1.1 403 Forbidden
```

2. 사용자 *gtouser*에 대한 JWT를 검색합니다. `bcd06d5bdd1dbaaf81853d10a66aeb989a38dd51`은 이 예시의 Red Hat Single Sign-On **클라이언트 암호 생성** 중에 정의된 `CLIENT_SECRET`입니다.

```
TOKEN=$(curl -Lk --data "username=gtouser&password=gtouser&grant_type=password&client_id=istio\
&client_secret=bcd06d5bdd1dbaaf81853d10a66aeb989a38dd51" \
https://keycloak-rhsso.apps.ocp4.rhlab.de/auth/realms/servicemesh-lab/protocol/openid-connect/token \
| jq .access_token)

echo $TOKEN
```

3. 다음과 같이 JWT를 사용해 *gtouser*로 여행 서비스 API에 액세스합니다.

```
./scripts/call-via-mtls-and-jwt-travel-agency-api.sh prod-istio-system gto-external $TOKEN
```

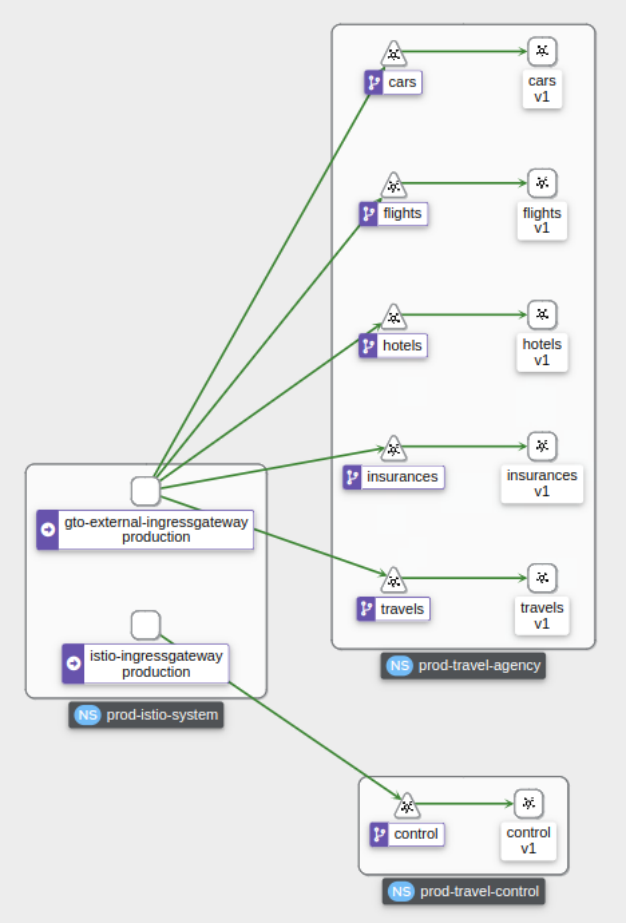


그림 5. GTO 사용자는 여행 서비스 API를 통해 Travel by Keyboard의 서비스에 액세스할 수 있습니다.

5장

새로운 보안 규정 준수

이 장에서는 새로운 보안 규정에 대처하기 위해 서비스 메쉬 및 환경 보안을 개선하고 강화합니다.

새로운 규정의 요구 사항 정의

새로운 보안 규정이 배포되었고, Travel by Keyboard의 보안 및 제품 팀은 규정 준수를 위해 다음과 같이 몇 가지를 변경해야 함을 알게 되었습니다.

- 메쉬 내 통신을 위해 기업 (중간) 인증 기관(CA)을 사용해 mTLS 인증서 생성
- 메쉬 내 특정 서비스에 액세스하기 위해 추가적인 권한 부여 필요

모든 서비스 전반에서 mTLS 구현

새로운 보안 규정을 준수하기 위해 서비스 메쉬를 강화하여 기업 CA 및 중간 CA에 따라 메쉬 내 mTLS 인증서를 생성하고 교체합니다. 다음 섹션에서는 인증서, 키, 인증서 체인을 생성하여 프로덕션 서비스 메쉬 컨트롤 플레인 리소스에 추가하고 변경 사항을 검증합니다.

기업 인증 기관 및 인증서 생성

플랫폼 관리자(클러스터 관리자 역할)인 Phillip은 프로덕션 환경에서 필요한 기업 CA 및 인증서 생성을 담당합니다. 보안 팀은 일반적으로 실제 프로덕션 시나리오에서 CA 및 인증서를 제공합니다.

다음과 같이 5장 디렉터리로 변경한 다음, CA 루트, 중간 키 및 인증서, 체인 인증서를 생성합니다.

```
cd ossm-heading-to-production-and-day-2/scenario-5-new-regulations-mtls-everywhere
```

리소스 리포지토리에서 **루트, 키, 인증서를 생성**하는 방법에 관한 자세한 지침을 따르세요. →

프로덕션 서비스 메쉬 컨트롤 플레인 테넌트 업데이트

메쉬 오퍼레이터인 Emma는 새로운 기업 CA, 중간 CA, 체인 인증서를 사용할 수 있도록 프로덕션 서비스 메쉬 컨트롤 플레인 테넌트를 수정하는 업무를 담당합니다.

컨트롤 플레인과 데이터 플레인이 현재 사용 중인 인증서의 발급자를 확인하는 것부터 시작합니다. 각각 `Issuer: O = cluster.local`과 같은 형식이어야 합니다.

1. 다음과 같이 기본 서비스 메쉬 인증서의 발급자를 확인합니다.

```
./login-as.sh emma
oc get -o yaml secret istio-ca-secret -n prod-istio-system | grep ca-cert | awk '{print $2}' | \
  base64 -d | openssl x509 -noout -text
```

2. 다음과 같이 `istiod`와의 통신에 사용되는 인증서의 발급자를 확인합니다.

```
oc exec "$(oc get pod -l app=istio-ingressgateway -n prod-istio-system -o \
  jsonpath={.items..metadata.name})" -c istio-proxy -n prod-istio-system -- openssl s_client \
  -showcerts -connect $(oc get svc istiod-production -o jsonpath={.spec.clusterIP}):15012
```

3. 다음과 같이 포드 간 통신에 사용되는 인증서의 발급자를 확인합니다.

```
oc exec "$(oc get pod -l app=hotels -n prod-travel-agency -o jsonpath={.items..metadata.name})" \
  -c istio-proxy -n prod-travel-agency -- openssl s_client -showcerts -connect \
  $(oc get svc discounts -o jsonpath={.spec.clusterIP}):8000
```

이어서 `ca-cert.pem`, `ca-key.pem`, `root-cert.pem`, `cert-chain.pem` 입력 파일을 포함하는 `cacerts`라는 암호를 생성합니다. 추가 정보는 **Red Hat OpenShift 도큐멘테이션**에서 확인할 수 있습니다. ➔

4. 다음과 같이 `cacerts` 암호를 생성하고 입력 파일을 추가합니다.

```
oc create secret generic cacerts -n prod-istio-system \
  --from-file=ca-cert.pem=certs-resources/intermediate/certs/intermediate.cert.pem \
  --from-file=ca-key.pem=certs-resources/intermediate/private/intermediate.key.pem \
  --from-file=root-cert.pem=certs-resources/certs/ca.cert.pem \
  --from-file=cert-chain.pem=certs-resources/intermediate/certs/ca-chain.cert.pem
```

이 예시에서 보유한 항목은 다음과 같습니다.

- `intermediate.cert.pem`(`ca-cert.pem`): 중간 CA의 인증서
- `intermediate.key.pem`(`ca-key.pem`): 중간 CA 인증서의 키
- `ca.cert.pem`(`root-cert.pem`): 루트 CA 인증서
- `ca-chain.cert.pem`(`cert-chain.pem`): 두 인증서를 모두 포함하는 체인

서비스 메쉬 컨트롤 플레인이 기본적으로 생성하는 `istio-system-ca` 암호로 인해 `istiod`가 엔터프라이즈 인증서를 잘못 선택할 수 있으므로 간섭을 방지하기 위해 이 암호를 제거하겠습니다.

5. 다음과 같이 `istio-system-ca` 암호를 제거합니다.

```
oc get secret istio-ca-secret -n prod-istio-system -o yaml > istio-ca-secret-default.yaml
oc delete secret istio-ca-secret -n prod-istio-system
```

6. 다음과 같이 `cacerts` 암호를 `prod-istio-system` 네임스페이스의 **프로덕션** 서비스 메쉬 컨트롤 플레인 리소스에 추가합니다.

```
# Add in production SMCP in prod-istio-system
  security:
    certificateAuthority:
      type: Istiod
      istiod:
        type: PrivateKey
        privateKey:
          rootCADir: /etc/cacerts
```

새로운 인증서를 추가했다면 컨트롤 플레인 및 데이터 플레인 리소스를 다시 시작하여 변경 사항을 구현해야 합니다.

7. 다음과 같이 컨트롤 플레인 `istiod`와 게이트웨이 포드를 다시 시작합니다.

```
oc -n prod-istio-system delete pods \
  -l 'app in (istiod,istio-ingressgateway, istio-egressgateway,gto-external-ingressgateway)'
oc -n prod-istio-system get -w pods
```

8. 다음과 같이 데이터 플레인 포드를 다시 시작합니다.

```
oc -n prod-travel-control delete pods --all
oc -n prod-travel-agency delete pods --all
oc -n prod-travel-portal delete pods --all
```

9. 다음과 같이 `cacerts` 발급자가 `Issuer: C = GB, ST = England, L = London, O = Travel Agency Ltd, OU = Travel Agency Ltd Certificate Authority, CN = Travel Agency Ltd Root CA`인지 확인합니다.

```
oc get -o yaml secret cacerts -n prod-istio-system | grep ca-cert | awk '{print $2}' \
  | base64 -d | openssl x509 -noout -text
```

10. 데이터 플레인 통신의 보안이 새로운 기업 인증서로 유지되는지 확인합니다.

```
./verify-dataplane-certs.sh
```

11. 컨트롤 플레인 통신의 보안이 새로운 기업 인증서로 유지되는지 확인합니다.

```
./verify-controlplane-certs.sh
```

특정 서비스에 대해 STRICT mTLS 비활성화

서비스 메쉬에서 실행되는 일부 워크로드는 자체 mTLS 인증서를 제공하거나 mTLS를 지원하지 않을 수 있습니다. 서비스 메쉬를 구성하여 이러한 상황을 처리할 수 있습니다.

차량 서비스에 대한 mTLS 비활성화를 위한 자세한 실습이 온라인 리포지토리에 포함되어 있습니다. →

새로운 권한 부여 정책 구현

새로운 보안 규정에 맞춰 조정하기 위해 새로운 권한 부여 정책도 구현합니다. 이 예시에서는 메쉬 오퍼레이터(Emma)와 도메인 소유자(Cristina 및 Farid)가 모범 사례 및 비즈니스 요구 사항에 따라 특정 서비스에 대한 액세스를 제한합니다.

권한 부여에 관한 **상세 정보**는 Istio 도큐멘테이션에서 제공됩니다. →

기본 권한 부여 정책 확인

먼저 기본 서비스 메쉬 권한 부여 정책이 모든 통신을 허용하는지 확인합니다.

1. 다음과 같이 `control.prod-travel-control` 통신을 테스트합니다.

```
https://travel-prod-istio-system.apps.<CLUSTERNAME>.<BASEDOMAIN>/
```

2. 다음과 같이 스크립트를 실행하여 나머지 통신을 테스트합니다.

```
./scripts/check-authz-all.sh ALLOW prod-istio-system <CLUSTERNAME> <BASEDOMAIN> \
  <CERTS_LOCATION> (CERTS_LOCATION ../scenario-4-onboard-new-portal-with-authentication)
```

출력에는 모든 통신이 기본적으로 허용되는 것으로 표시되어야 합니다.

```
Authorization prod-istio-system --> prod-travel-agency
-----------------------------------------------------------------------
[ALLOW] gto-external-ingressgateway --> travels.prod-travel-agency
[ALLOW] gto-external-ingressgateway --> cars.prod-travel-agency
[ALLOW] gto-external-ingressgateway --> flights.prod-travel-agency
[ALLOW] gto-external-ingressgateway --> insurances.prod-travel-agency
[ALLOW] gto-external-ingressgateway --> hotels.prod-travel-agency

Authorization prod-travel-control --> prod-travel-agency
-----------------------------------------------------------------------
[ALLOW] control.prod-travel-control --> travels.prod-travel-agency
[ALLOW] control.prod-travel-control --> cars.prod-travel-agency
[ALLOW] control.prod-travel-control --> flights.prod-travel-agency
[ALLOW] control.prod-travel-control --> insurances.prod-travel-agency
[ALLOW] control.prod-travel-control --> hotels.prod-travel-agency

Authorization prod-travel-portal --> prod-travel-agency
-----------------------------------------------------------------------
[ALLOW] viaggi.prod-travel-portal --> travels.prod-travel-agency
[ALLOW] viaggi.prod-travel-portal --> cars.prod-travel-agency
[ALLOW] viaggi.prod-travel-portal --> flights.prod-travel-agency
[ALLOW] viaggi.prod-travel-portal --> insurances.prod-travel-agency
[ALLOW] viaggi.prod-travel-portal --> hotels.prod-travel-agency

Authorization prod-travel-agency --> prod-travel-agency
-----------------------------------------------------------------------
[ALLOW] travels.prod-travel-portal --> discounts.prod-travel-agency
[ALLOW] travels.prod-travel-portal --> cars.prod-travel-agency
[ALLOW] travels.prod-travel-portal --> flights.prod-travel-agency
[ALLOW] travels.prod-travel-portal --> insurances.prod-travel-agency
[ALLOW] travels.prod-travel-portal --> hotels.prod-travel-agency
```

모두 거부(deny-all) 정책 적용

서비스 메쉬 보안을 위한 모범 사례에서는 **기본-거부 패턴(default-deny pattern)**을 사용합니다. 4장에서는 `gto-external-ingressgateway`를 통한 요청을 허용하도록 `authpolicy-gto-external` 리소스를 구성했습니다. 이제는 기본적으로 모든 요청을 거부하고 권한 부여 정책에 지정된 요청만 허용하겠습니다. 도메인 소유자(메쉬 개발자 역할)인 Farid와 Cristina는 이러한 변경 사항을 구현할 수 있습니다.

1. 다음과 같이 `prod-travel-control`과 `prod-travel-agency` 네임스페이스에 기본-거부 패턴을 적용합니다.

```
cd ossm-heading-to-production-and-day-2/scenario-5-new-regulations-mtls-everywhere
./login-as.sh cristina
oc apply -f authz-resources/01-default-deny-travel-portal-access.yaml
./login-as.sh farid
oc apply -f authz-resources/01-default-deny-travel-agency-access.yaml
```

2. 웹 브라우저를 통한 액세스가 거부되는지 확인합니다. 다음 링크를 클릭하면 `RBAC: access denied`라는 메시지가 표시되어야 합니다.

```
https://travel-prod-istio-system.apps.<CLUSTERNAME>.<BASEDOMAIN>/
```

3. 다음과 같이 스크립트를 실행하여 나머지 통신을 확인합니다.

```
./scripts/check-authz-all.sh DENY prod-istio-system <CLUSTERNAME> <BASEDOMAIN> \
  <CERTS_LOCATION> (CERTS_LOCATION ../scenario-4-onboard-new-portal-with-authentication)
```

출력에는 모든 통신이 기본적으로 거부되는 것으로 표시되어야 합니다.

```
Authorization prod-istio-system --> prod-travel-agency
------------------------------------------------------------------------
[DENY] gto-external-ingressgateway --> travels.prod-travel-agency
[DENY] gto-external-ingressgateway --> cars.prod-travel-agency
[DENY] gto-external-ingressgateway --> flights.prod-travel-agency
[DENY] gto-external-ingressgateway --> insurances.prod-travel-agency
[DENY] gto-external-ingressgateway --> hotels.prod-travel-agency

Authorization prod-travel-control --> prod-travel-agency
------------------------------------------------------------------------
[DENY] control.prod-travel-control --> travels.prod-travel-agency
[DENY] control.prod-travel-control --> cars.prod-travel-agency
[DENY] control.prod-travel-control --> flights.prod-travel-agency
[DENY] control.prod-travel-control --> insurances.prod-travel-agency
[DENY] control.prod-travel-control --> hotels.prod-travel-agency

Authorization prod-travel-portal --> prod-travel-agency
------------------------------------------------------------------------
[DENY] viaggi.prod-travel-portal --> travels.prod-travel-agency
[DENY] viaggi.prod-travel-portal --> cars.prod-travel-agency
[DENY] viaggi.prod-travel-portal --> flights.prod-travel-agency
[DENY] viaggi.prod-travel-portal --> insurances.prod-travel-agency
[DENY] viaggi.prod-travel-portal --> hotels.prod-travel-agency

Authorization prod-travel-agency --> prod-travel-agency
------------------------------------------------------------------------
[DENY] travels.prod-travel-portal --> discounts.prod-travel-agency
[DENY] travels.prod-travel-portal --> cars.prod-travel-agency
[DENY] travels.prod-travel-portal --> flights.prod-travel-agency
[DENY] travels.prod-travel-portal --> insurances.prod-travel-agency
[DENY] travels.prod-travel-portal --> hotels.prod-travel-agency
```

Kiali를 통해 **통신 정책을 시각적으로 확인**할 수도 있습니다. →

비즈니스에 적합한 권한 부여 정책 적용

이제는 비즈니스 요구 사항에 따라 특정 서비스 및 요청에 대한 액세스를 허용하겠습니다.

`Prod-travel-control` 네임스페이스

메쉬 오퍼레이터인 Emma는 다음과 같이 기본 `istio-ingressgateway`를 통해 서비스 메쉬에 액세스할 수 있도록 허용하는 업무를 담당합니다.

```
./login-as.sh emma
oc apply -f authz-resources/02-travel-portal-allow-external-traffic.yaml
```

여행 포털 도메인

여행 포털 도메인 소유자(메쉬 개발자 역할)인 Cristina는 `istio-ingressgateway`에서 여행 포털에 액세스할 수 있도록 허용하는 업무를 담당합니다. 다음과 같이 `istio-ingressgateway`가 모든 경로에서 호출을 허용하도록 설정하고, 주체인 `cluster.local/ns/prod-istio-system/sa/istio-ingressgateway-service-account`는 `prod-travel-control` 네임스페이스를 호출할 수 있도록 허용됩니다.

```
./login-as.sh cristina
oc apply -f authz-resources/02-travel-portal-allow-istio-ingressgateway-traffic.yaml
```

여행 포털 인터페이스에 대한 액세스는 몇 초 내로 복구됩니다. 앞서 수행한 것처럼 `https://travel-prod-istio-system.apps.<CLUSTERNAME>.<BASEDOMAIN>/`를 통한 연결을 테스트할 수 있습니다.

여행 서비스 도메인

여행 서비스 도메인 소유자(메쉬 개발자 역할)인 Farid는 GTO가 검색 요청을 수행할 수 있도록 `gto-external-ingressgateway`에서 `travels.prod-travel-agency`, `hotels.prod-travel-agency`, `cars.prod-travel-agency`, `insurances.prod-travel-agency`, `flights.prod-travel-agency` 서비스에 액세스할 수 있도록 허용하는 업무를 담당합니다. 앞 섹션에서는 모든 외부 주체(`requestPrincipals[*]`)에 대한 모든 경로에서 `gto-external-ingressgateway`에 호출할 수 있도록 허용했습니다. 이 섹션에서는 주체인 `cluster.local/ns/prod-istio-system/sa/gto-external-ingressgateway-service-account`만이 `prod-travel-agency` 네임스페이스를 호출할 수 있도록 허용합니다.

1. 다음과 같이 GTO에 대한 액세스를 허용합니다.

```
./login-as.sh farid
oc apply -f authz-resources/03-gto-external-travels-to-travel-agency-allow.yaml
```

2. GTO에서 여행을 검색하는 것이 허용되는지 확인합니다. 다음과 같이 "유효하지 않은 연결"이라는 사유가 포함된 HTTP `500` 오류를 수신해야 합니다.

```
TOKEN=$(curl -Lk --data "username=gtouser&password=gtouser&grant_type=password&\
client_id=istio&client_secret=bcd06d5bdd1dbaaf81853d10a66aeb989a38dd51" \
https://keycloak-rhsso.apps.ocp4.rhlab.de/auth/realms/servicemesh-lab/protocol/openid-connect/
token \
| jq .access_token)

../scenario-4-onboard-new-portal-with-authentication/scripts/call-via-mtls-and-jwt-travel-agency-api.
sh \
  prod-istio-system gto-external $TOKEN
```

3. GTO로부터의 액세스가 인가되었으므로 403 오류는 더 이상 없지만 연결에 실패하고 있습니다. Kiali에서 통신을 검사하세요. `gto-external-ingressgateway`에서 `*.prod-travel-agency`로의 요청이 허용되었지만 **항공편** 서비스에서 `mysqldb` 서비스로의 연결은 허용되지 않아야 합니다. 이에 대해서는 `flights-v1` 워크로드의 `istio-proxy` 로그에서 확인할 수 있습니다.

4. 주체인 `cluster.local/ns/prod-travel-agency/sa/default`가 `prod-travel-agency`네임스페이스를 호출하도록 허용하기 위해 `prod-travel-agency` 네임스페이스 내 통신을 허용합니다.

```
./login-as.sh farid
oc apply -f authz-resources/04-intra-travel-agency-allow.yaml
```

5. 다음과 같이 현재 GTO에서의 여행 검색이 허용된 상태인지 확인합니다.

```
TOKEN=$(curl -Lk --data "username=gtouser&password=gtouser&grant_type=password&\
client_id=istio&client_secret=bcd06d5bdd1dbaaf81853d10a66aeb989a38dd51" \
https://keycloak-rhsso.apps.<CLUSTERNAME>.<BASEDOMAIN>/auth/realms/servicemesh-lab/protocol/openid-connect/
token \
| jq .access_token)

../scenario-4-onboard-new-portal-with-authentication/scripts/call-via-mtls-and-jwt-travel-agency-api.
sh \
prod-istio-system gto-external $TOKEN
```

6. 다음과 같이 네임스페이스 내의 통신을 테스트합니다.

```
./scripts/check-authz-all.sh 'ALLOW intra' prod-istio-system <CLUSTERNAME> <BASEDOMAIN> \
  <CERTS_LOCATION> (CERTS_LOCATION ../scenario-4-onboard-new-portal-with-authentication)
```

출력에는 업데이트된 통신 권한 부여가 표시되어야 합니다.

```
Authorization prod-istio-system --> prod-travel-agency
------------------------------------------------------------------
[ALLOW] gto-external-ingressgateway --> travels.prod-travel-agency
[ALLOW] gto-external-ingressgateway --> cars.prod-travel-agency
[ALLOW] gto-external-ingressgateway --> flights.prod-travel-agency
[ALLOW] gto-external-ingressgateway --> insurances.prod-travel-agency
[ALLOW] gto-external-ingressgateway --> hotels.prod-travel-agency

Authorization prod-travel-control --> prod-travel-agency
------------------------------------------------------------------
[DENY] control.prod-travel-control --> travels.prod-travel-agency
[DENY] control.prod-travel-control --> cars.prod-travel-agency
[DENY] control.prod-travel-control --> flights.prod-travel-agency
[DENY] control.prod-travel-control --> insurances.prod-travel-agency
[DENY] control.prod-travel-control --> hotels.prod-travel-agency

Authorization prod-travel-portal --> prod-travel-agency
------------------------------------------------------------------
[DENY] viaggi.prod-travel-portal --> travels.prod-travel-agency
[DENY] viaggi.prod-travel-portal --> cars.prod-travel-agency
[DENY] viaggi.prod-travel-portal --> flights.prod-travel-agency
[DENY] viaggi.prod-travel-portal --> insurances.prod-travel-agency
[DENY] viaggi.prod-travel-portal --> hotels.prod-travel-agency
```

```
Authorization prod-travel-agency --> prod-travel-agency
-----------------------------------------------------------------------
[ALLOW] travels.prod-travel-portal --> discounts.prod-travel-agency
[ALLOW] travels.prod-travel-portal --> cars.prod-travel-agency
[ALLOW] travels.prod-travel-portal --> flights.prod-travel-agency
[ALLOW] travels.prod-travel-portal --> insurances.prod-travel-agency
[ALLOW] travels.prod-travel-portal --> hotels.prod-travel-agency
```

7. 그러면 도메인 소유자(메쉬 개발자 역할)인 Farid는 다음과 같이 주체인 `cluster.local/ns/prod-travel-portal/sa/default`가 `prod-travel-agency` 네임스페이스를 호출하도록 허용할 수 있습니다.

```
./login-as.sh farid
oc apply -f authz-resources/05-travel-portal-to-travel-agency-allow.yaml
```

권한 부여 정책 확인

앞 섹션에서 적용된 권한 부여 정책은 이 예시 환경 내에서 비즈니스에 적합한 연결을 복원했습니다.

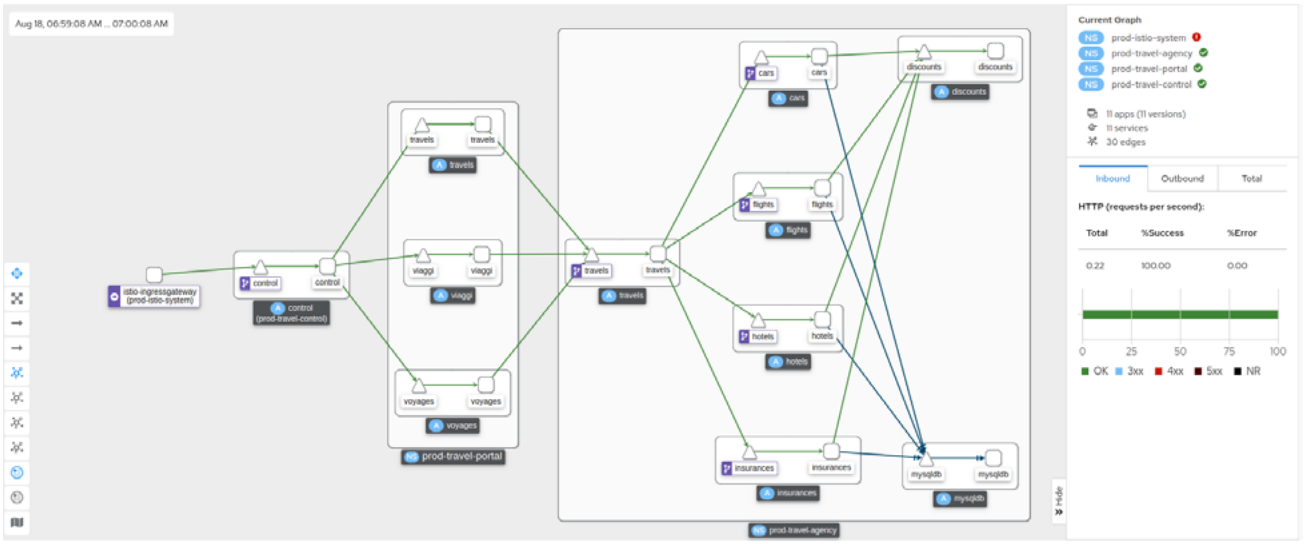


그림 6. 비즈니스에 적합한 연결을 포함한 권한 부여 정책의 Kiali 시각화

이 구성은 이 장에서 시작한 모두 허용(allow-all) 정책과 다릅니다. 메쉬 오퍼레이터인 Emma는 다음과 같이 `travel-portal-control` 네임스페이스가 `travel-portal-agency` 네임스페이스에 여전히 액세스할 수 없는지 여부를 확인할 수 있습니다.

```
./login-as.sh emma
./scripts/check-authz-all.sh 'ALLOW intra' prod-istio-system <CLUSTERNAME> <BASEDOMAIN> \
  <CERTS_LOCATION> (CERTS_LOCATION ../scenario-4-onboard-new-portal-with-authentication)
```

파트너 포털에 대한 상세 권한 부여 정책 적용

이 예시에서 Travel by Keyboard의 비즈니스 개발 팀은 파트너인 GTO와의 비즈니스 계약을 마무리지었습니다. 결과적으로 GTO는 여행사 API를 통해 **항공편** 및 **보험** 서비스에 *한해* 오퍼를 제공할 수 있게 됩니다. 이 섹션에서는 이러한 비즈니스 계약에 맞춰 권한 부여 정책을 구성합니다. 메쉬 오퍼레이터인 Emma는 세부적인 권한 부여 정책 구성을 담당합니다.

1. 다음과 같이 **notPaths** 연산을 사용해 `/flights` 및 `/insurances`를 *제외한* 모든 경로에 대해 `gto-external-ingressgateway`에서 시작되는 통신을 거부합니다.

```
./login-as.sh emma
oc apply -f authz-resources/06-gto-external-travels-only-flights-insurances-paths-allow.yaml
```

2. 현재 올바른 연결이 허용되지 않는 상태인지 확인합니다.

```
./scripts/verify-fine-grained-authz.sh prod-istio-system <CLUSTERNAME> <BASEDOMAIN> \
  <CERTS_LOCATION> (CERTS_LOCATION ../scenario-4-onboard-new-portal-with-authentication)

[DENY] GTO --> /travels
[DENY] GTO --> /cars
[ALLOW] GTO --> /flights
[ALLOW] GTO --> /insurances
[DENY] GTO --> /hotels
```

그림 7에서는 이 예시를 위한 최종 인증 및 권한 부여 구성을 보여줍니다.

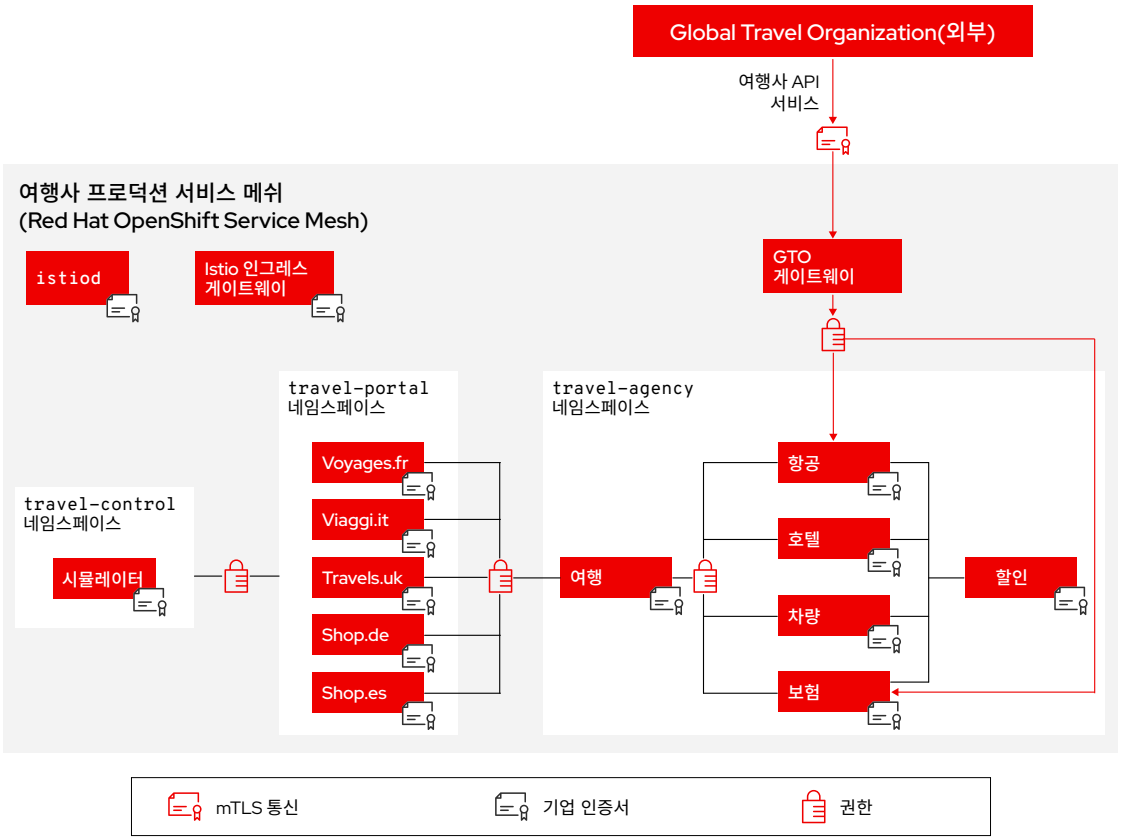


그림 7. 여행사 서비스 메쉬를 위한 최종 인증 및 권한 부여 구성

6장

파트너십 확장과 브로커 서비스 구현

이 장에서는 보험 브로커 파트너를 온보딩하고 특정 보험 요청을 이 파트너에게 전달하여 인기 있는 목적지로 여행하는 고객에게 프리미엄 보험 패키지를 제공합니다.

새로운 기회에 대한 요구 사항 정의

Travel by Keyboard의 비즈니스 개발 팀은 외부 보험 브로커인 Super Insurance와 파트너십을 맺고 고객에게 추가적인 프리미엄 보험 패키지를 제공합니다. 따라서 특정 보험 요청은 다음과 같이 이 파트너에게 전달됩니다.

- 목적지가 **런던**, **로마**, **파리**, **베를린**, **뮌헨**, **더블린**인 여행과 관련된 보험에 대한 모든 요청은 Super Insurance로 전달됩니다.
- 외부 파트너 서비스와의 모든 통신은 mTLS를 통해 수행됩니다.

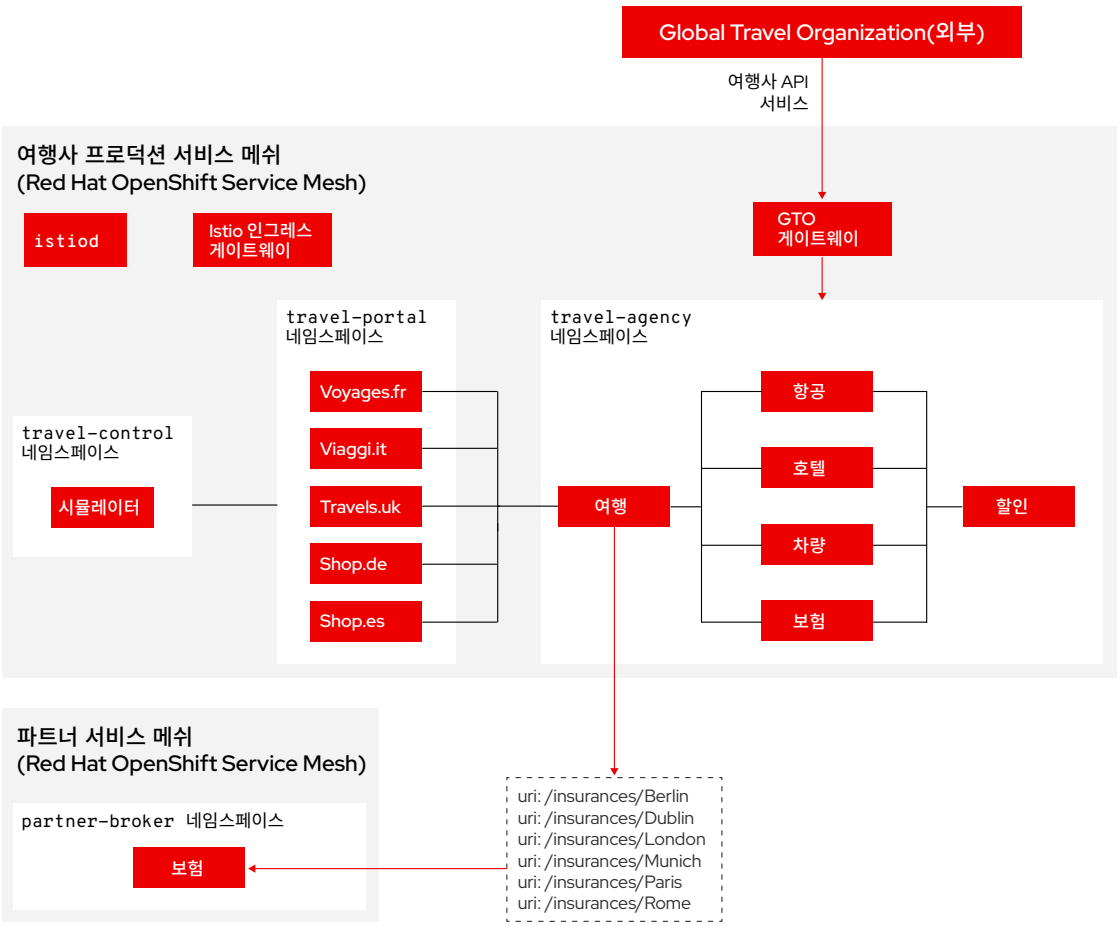


그림 8. 외부 보험 브로커 파트너 시스템에 대한 연결이 포함된 여행사 서비스 메쉬 아키텍처

파트너 보험 서비스 배포

먼저 Super Insurance의 시스템 내에서 서비스 메쉬 구성 요소를 배포 및 구성해야 합니다. 외부 조직은 Travel by Keyboard를 위해 설정한 것과 유사한 서비스 메쉬 역할 및 책임을 지닌 일련의 엔터프라이즈 사용자 유형도 보유해야 하지만, 여기에선 이러한 단계를 클러스터 관리자 역할(Phillip)로 수행하겠습니다.

클러스터 관리자 역할은 다음을 포함한 필수 서비스 메쉬 구성 요소 배포를 담당합니다.

- `partner-istio-system` 네임스페이스에서 서비스 메쉬 컨트롤 플레인 **파트너** 리소스 배포
- `premium-broker` 네임스페이스를 생성하고 이 네임스페이스에 대한 서비스 메쉬 멤버십을 파트너의 서비스 메쉬 컨트롤 플레인에 추가
- `premium-broker` 네임스페이스에서 **보험** 서비스 배포
- `premium-broker` 네임스페이스에서 **보험** 서비스로의 트래픽 라우팅 활성화
- 트래픽이 파트너 **보험** 서비스로 라우팅되는지 확인

이 예시에서는 이 모든 태스크를 다음과 같이 스크립트를 사용해 수행합니다.

```
cd ossm-heading-to-production-and-day-2/scenario-6-partner-agency-multi-mesh
./login-as.sh phillip

./create-premium-insurance-broker.sh <PARTNER INSURANCE NAMESPACE> \
  <PARTNER ISTIO CP NAMESPACE> <CLUSTER.DOMAIN eg. apps.ocp4.rhlab.de> <PARTNER SMCP NAME>
./create-premium-insurance-broker.sh premium-broker partner-istio-system \
  <CLUSTER.DOMAIN eg. apps.ocp4.example.com> partner
```

요청을 보험 브로커 파트너에게 라우팅

다음으로, Travel by Keyboard의 시스템이 앞서 정의된 프리미엄 목적지에 대한 보험 요청을 분리하여 Super Insurance로 전달하도록 구성하겠습니다. 이를 수행할 수 있는 세 가지 옵션은 다음과 같습니다. 이들 옵션 중 애플리케이션을 변경해야 하는 것은 없습니다.

- **옵션 1: Non-mTLS 외부 보험 서비스 호출**
  이 옵션은 `VirtualService` 및 `DestinationRule` 리소스를 적용하여 트래픽을 원격 서비스 위치로 라우팅하고 원격 서비스 목적지 위치에 대한 `ServiceEntry` 리소스를 생성합니다.
- **옵션 2: 페더레이션 없이 mTLS를 사용하는 멀티테넌시**
  이 옵션은 프로덕션 서비스 메쉬의 이그레스 게이트웨이를 사용해 호출을 원격 **프리미엄-브로커/보험** 서비스로 리디렉션하여 게이트웨이에서 원격 인증서를 공유 및 적용합니다. 이 설정의 예시는 **이 블로그 포스트**에서 확인할 수 있습니다.
- **옵션 3: 프로덕션 메쉬와 파트너 메쉬 간 페더레이션**
  이 옵션은 프로덕션 및 파트너 서비스 메쉬 인스턴스를 페더레이션하고 파트너 **보험** 서비스를 프로덕션 서비스 메쉬로 가져옵니다. **서비스 메쉬 페더레이션 계획 수립을 위한 중요 고려 사항**은 온라인 리소스 리포지토리에 포함되어 있습니다. →

이 예시에서는 옵션 3을 사용해 두 개의 서비스 메쉬를 페더레이션합니다. 이를 위해서는 다음 작업을 수행해야 합니다.

- 페더레이션된 연결을 위해 **프로덕션** 서비스 메쉬 컨트롤 플레인 리소스를 업데이트하여 두 개의 추가 게이트웨이(`partner-mesh-egress` 및 `partner-mesh-ingress`) 선언
- 페더레이션된 연결을 위해 **파트너** 서비스 메쉬 컨트롤 플레인 리소스를 업데이트하여 두 개의 추가 게이트웨이(`production-mesh-egress` 및 `production-mesh-ingress`) 선언
- 각 메쉬에서 구성 맵인 `istio-ca-root-cert`를 추출해 반대편 메쉬의 컨트롤 플레인 네임스페이스에 공유하여 TLS 핸드셰이크 지원
- `prod-istio-system`에서 `partner ServiceMeshPeer` 리소스를 생성하여 **프로덕션** 메쉬에서 **파트너** 메쉬로의 피어링 시작
- `partner-istio-system`에서 `production ServiceMeshPeer` 리소스를 생성하여 **파트너** 메쉬에서 **프로덕션** 메쉬로의 피어링 시작
- `premium-broker` 네임스페이스에서 (`ExportedServiceSet`를 통해) **보험** 서비스를 내보내고 (`ExportedServiceSet`를 통해) `prod-travel-agency` 네임스페이스로 가져오기

이 예시에서는 클러스터 관리자인 Phillip이 다음과 같이 스크립트를 사용해 이 모든 태스크를 수행합니다.

```
./scripts/option-3-execute-federation-setup.sh <1_SMCP_NAMESPACE> <1_SMCP_NAME> \
  <2_SMCP_NAMESPACE> <2_SMCP_NAME> <PREMIUM NAMESPACE>
./scripts/option-3-execute-federation-setup.sh prod-istio-system production partner-istio-system \
  partner premium-broker
```

이러한 구성은 `prod-travel-agency/travels` 서비스에 도착하는 보험 견적 요청이 페더레이션된 `insurances.premium-broker.svc.partner-imports.local` 서비스로 전달되는 결과로 나타납니다.

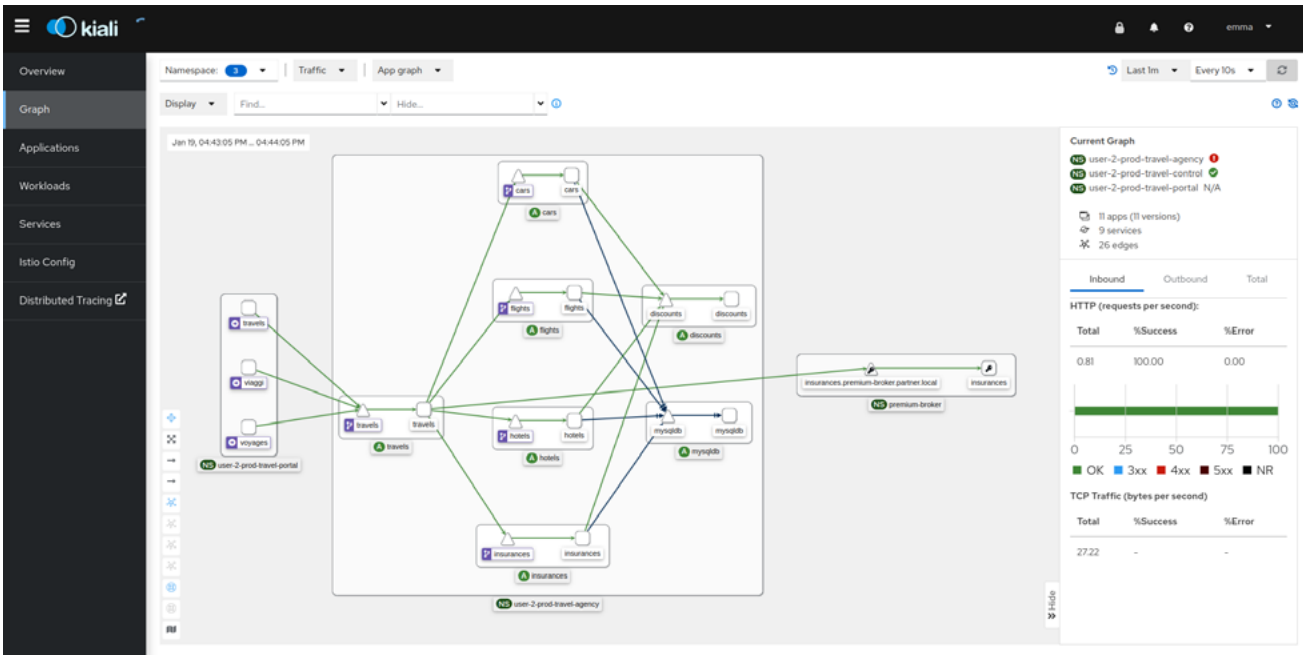


그림 9. 가져온 서비스인 insurances.premium-broker.svc.partner-imports.local에 요청을 페더레이션하는 프로덕션 메쉬

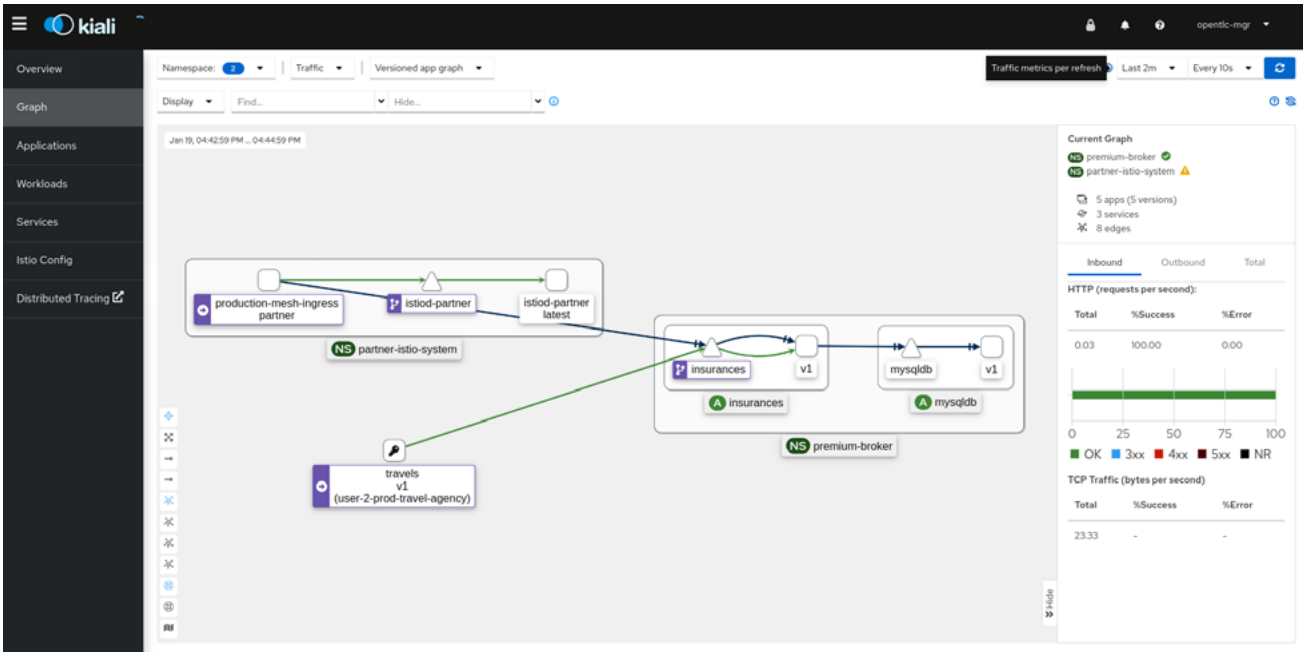


그림 10. 프로덕션 메쉬의 travels-v1.prod-travel-agency로 내보낸 서비스인 insurances.premium-broker를 통해 요청을 제공하는 파트너 메쉬

`insurances.premium-broker` 포드의 로그에는 이 포드가 지정된 프리미엄 목적지만 지원하는 반면, 다른 모든 목적지는 `insurances.prod-travel-agency` 포드가 지원하는 것으로 표시됩니다.

이 장의 리소스

- Red Hat OpenShift 도큐멘테이션: **페더레이션된 메쉬에서 서비스 내보내기**
- 블로그 포스트: **이그레스 엣지 트래픽에 대한 mTLS가 포함된 보안 옵션**
- 블로그 포스트: **자동화된 페더레이션 설정**

2부: Day 2 오퍼레이션

서비스 메쉬를 프로덕션으로 배포했다면 서비스 메쉬를 최적화, 튜닝, 유지 관리해야 합니다. Day 2 오퍼레이션은 모든 기술의 지속적인 효율성, 안정성, 유용성에 매우 중요하며, 서비스 메쉬도 마찬가지입니다. 이 e-book의 1부에서는 보안, 적응성, 관측성을 위한 플랫폼을 구성하여 Day 2 오퍼레이션에 대한 기반을 확립했습니다. 2부에서는 서비스 메쉬 트러블슈팅, 튜닝, 업그레이드를 위한 방법 및 기법에 대해 알아보겠습니다.

7장

메쉬 트러블슈팅

이 장에서는 서비스 메쉬 트러블슈팅을 위한 몇 가지 기법에 대해 살펴보겠습니다.

주요 질문

각각의 Istio 네트워크 구성 디버깅 여정은 서로 다르지만 다음과 같은 10가지 주요 질문으로 시작하면 도움이 됩니다.

1. Istio 네트워크 구성이 구문 측면에서 유효한가?
2. 네트워크 구성에 오류가 있거나 경고 상태가 설정되어 있는가?
3. 리소스 이름이 올바른가? 리소스가 올바른 네임스페이스에 있는가?
4. 리소스 선택기가 올바른가?
5. Envoy가 구성을 수락(ACK)했는가?
6. 구성이 기대한 대로 Envoy에 표시되었는가?
7. `istiod`(Pilot) 로그 오류가 발생했는가?
8. Red Hat OpenShift Service Mesh 오퍼레이터와 컨트롤 플레인의 상태가 양호한가?
9. 애플리케이션이 기대한 메쉬 인스턴스의 일부인가?
10. 관련 인증서가 유효한가?

다음 섹션에서는 이러한 각 질문에 답하고 그 과정에서 서비스 메쉬 문제를 해결하기 위해 필요한 정보를 수집하는 방법에 관한 지침을 제공합니다.

메쉬의 구성 요소 이해

거의 모든 디버깅 여정의 첫 걸음은 현재 자신이 무엇을 다루고 있는지 아는 것입니다. 보유한 **메쉬 구성 요소**와 이 구성 요소의 목적 및 구성, 그리고 이 구성이 적용되는 위치에 대해 이해하는 것이 중요합니다. 메쉬 내 트래픽의 예상 흐름과 메쉬 오퍼레이터 및 애플리케이션 운영 팀 사용자 유형이 디버깅 작업에 사용할 수 있는 툴에 대해서도 이해해야 합니다. 이 예시에서는 다음과 같은 메쉬 구성 요소를 중점적으로 살펴보겠습니다.

- 메쉬로 이동하는 트래픽을 허용하는 **인그레스 구성 요소**. 이 예시에서는 `istio-ingressgateway`와 `gto-ingressgateway`가 north 인바운드 클라이언트 트래픽을 운반하고 `partner-mesh-ingress`는 페더레이션된 west 인바운드 트래픽을 운반합니다.
- 메쉬 외부로 이동하는 트래픽을 허용하는 **이그레스 구성 요소**. 이 예시에서 `istio-egressgateway`는 south 아웃바운드 트래픽을 운반하고 `partner-mesh-egress`는 페더레이션된 east 아웃바운드 트래픽을 운반합니다.
- 주요 애플리케이션 컨테이너의 안팎으로 이동하는 TCP, HTTP, HTTP/2, GRPC[1] 트래픽을 가로채는 **`istio-proxy` 사이드카 컨테이너**(메쉬의 일부인 각 포드에 하나씩)
- 트래픽, 보안, 관측성 등을 위해 메쉬 구성을 적용하는 **`istiod` 구성 요소**

서비스 메쉬의 주요 작업 중 두 가지는 메쉬에 대한 액세스를 통제하고 제한하는 Red Hat OpenShift 네트워크 정책을 적용하는 것과 서비스 메쉬 `istio-proxy` 사이드카를 통해서만 각 포드에 액세스할 수 있도록 IP 테이블 룰을 다시 작성하는 것입니다. `NetworkPolicy` 리소스는 이를 달성하기 위해 메쉬 격리 및 멀티테넌시를 적용할 수 있습니다.

네트워크 정책에 대한 자세한 내용은 다음과 같은 **Red Hat OpenShift 도큐멘테이션**에서 확인할 수 있습니다.

- **멀티테넌시와 클러스터 전체의 메쉬 설치 간 비교**
- **메쉬 네트워크 정책에 대한 이해**
- **올바른 네트워크 정책 설정**

컨트롤 플레인 네트워크 정책

이 예시에서는 `prod-istio-system` 네임스페이스에서 다음과 같은 `NetworkPolicy` 리소스가 적용되었습니다.

표 5. `prod-istio-system` 네임스페이스를 위한 `NetworkPolicy` 리소스 애플리케이션

이름	포드 선택기
`gto-external-ingressgateway`	`app=gto-external-ingressgateway,istio=ingressgateway`
`istio-expose-route-production`	`maistra.io/expose-route=true`
`istio-grafana-ingress`	`app=grafana`
`istio-ingressgateway`	`app=istio-ingressgateway,istio=ingressgateway`
`istio-istiod-production`	`app=istiod,istio.io/rev=production`

1 *Transmission Control Protocol(TCP), Hypertext Transfer Protocol(HTTP), HTTP/2, Google Remote Procedure Call(GRPC)*

이름	포드 선택기
istio-jaeger-ingress	app.kubernetes.io/component in (all-in-one,query), app.kubernetes.io/instance=jaeger-small-production, app.kubernetes.io/managed-by=jaeger-operator, app.kubernetes.io/part-of=jaeger
istio-kiali-ingress	app=kiali
istio-mesh-production	<none>
istio-prometheus-ingress	app=prometheus
partner-mesh-ingress	app=partner-mesh-ingress, federation.maistra.io/ingress-for=partner-mesh-ingress, istio=ingressgateway

데이터 플레인 네트워크 정책

데이터 플레인 네임스페이스에서 다음과 같은 NetworkPolicy 리소스가 적용되었습니다.

표 6. 데이터 플레인 네임스페이스를 위한 NetworkPolicy 리소스 애플리케이션

이름	포드 선택기
istio-expose-route-production	maistra.io/expose-route=true
istio-mesh-production	<none>

istio-proxy 포트 및 용도

끝으로, istio-proxy가 노출하는 주요 포트와 그 용도입니다.

- 15001: 트래픽을 주요 워크로드 컨테이너를 향해 사이드카 밖으로 운반하거나 워크로드 컨테이너의 트래픽 대응/요청으로 운반
- 15006: 트래픽을 사이드카 안으로 운반
- 15000: 액세스 진단
- 15020: Istio 에이전트, Envoy, 애플리케이션에서 병합된 Prometheus 텔레메트리에 액세스

Istio가 사용하는 포트에 관한 자세한 내용은 Istio 웹사이트에서 확인할 수 있습니다. →

메쉬 구성 확인

서비스 메쉬에 대한 구성을 제공하는 리소스가 많이 있지만 istiod 구성 요소만이 이 구성을 데이터 플레인과 istio-proxy 사이드카에 적용합니다. 메쉬 오퍼레이터인 Emma는 istioctl CLI 툴을 사용해 현재 적용된 구성의 상태를 확인하고 트러블슈팅할 수 있습니다.

구성이 적용되었는지 확인

먼저 서비스 메쉬 구성이 실제로 적용되었는지 확인합니다.

1. analyze 명령을 사용해 전체 서비스 메쉬 구성에 오류가 있는지 확인합니다. 규모가 매우 큰 데이터 플레인의 경우 시간이 오래 걸릴 수 있습니다.

   ```
   istioctl analyze
   ```

2. 서비스 메쉬 Envoy 프록시에 대한 검색 서비스의 XDS 프로토콜이 컨트롤 플레인과 데이터 플레인 간에 동기화되어 있는지 확인합니다.

   ```
   istioctl proxy-status istio-ingressgateway-6b948db88c-2sqth -i prod-istio-system \
     -n prod-istio-system
   ```

   또는

   ```
   istioctl proxy-status -i prod-istio-system -n prod-istio-system
   ```

구성의 적용 여부와 적용된 위치 확인

구성이 누락되거나 잘못 배치되거나 잘못된 것으로 의심되는 경우 관측성 스택과 CLI 툴을 사용해 조사할 수 있습니다.

1. Kiali를 사용해 각 네임스페이스에서 **Istio 구성**을 시각화하고 모든 검증 오류를 확인합니다.
2. 기대되는 구성을 강화 또는 재정의하기 위해 추가된 모든 구성의 `ServiceMeshControlPlane` 리소스 및 워크로드 배포를 확인합니다. Istio 프록시 주석에 관한 자세한 내용은 **Istio 웹사이트**와 **Red Hat OpenShift 도큐멘테이션**에서 확인할 수 있습니다. →
3. 다음과 같이 워크로드가 서비스 메쉬의 일부이며 올바른 메쉬의 구성원인지 확인합니다.
   - 워크로드에 대한 **사이드카 주입을 검증**합니다.
   - 워크로드가 포함된 네임스페이스의 `maistra.io/member-of` 레이블이 올바른 컨트롤 플레인 네임스페이스를 가리키는지 확인합니다.
4. Prometheus를 사용해 트래픽 유형 관련 문제가 있는지 확인합니다. `istio_agent_pilot_duplicate_envoy_clusters` 및 `istio_agent_pilot_destrule_subsets` 메트릭이 0보다 큰지 확인하고, 중복 여부를 확인합니다. 이렇게 하면 "no healthy upstream(양호한 업스트림이 없음)"으로 보고되는 문제가 해결되는 경우가 많습니다.
5. Kiali 또는 `istioctl`를 사용해 포드 판독, 서비스 노출 포트와 적용된 Istio 구성을 생성합니다.
   - Kiali에서, *워크로드* 섹션에서 보고 싶은 워크로드를 선택합니다.
   - 다음과 같이 `istioctl` CLI에서 describe 커맨드를 사용합니다.

     ```
     istioctl experimental describe pod cars-v1-594b79cfbf-wlcg9.prod-travel-agency -i \
     prod-istio-system -n prod-travel-agency
     ```

상세 구성 분석

상세 구성 분석 및 디버깅 실습은 온라인 리포지토리에서 확인할 수 있습니다. Istio 도큐멘테이션의 **Envoy 및 Istiod 디버깅** 섹션에서도 자세한 내용을 확인할 수 있습니다.

런타임 상태 및 보안 확인

서비스 메쉬에서 런타임 관련 문제가 있는 경우 컨트롤 플레인과 데이터 플레인의 상태를 확인하는 것이 중요합니다. 메쉬 오퍼레이터인 Emma와 메쉬 개발자인 Farid 및 Cristina는 Kiali, Jaeger, Prometheus를 사용해 자신에게 해당되는 도메인 내부에 있는 구성 요소의 상태를 조사할 수 있습니다.

컨트롤 플레인의 상태 확인

메쉬 오퍼레이터인 Emma는 컨트롤 플레인의 상태를 확인할 수 있습니다.

1. **서비스 메쉬 오퍼레이터** 설치를 확인하고 **컨트롤 플레인 설치**를 확인합니다.
2. 다음과 같이 모든 인스턴스의 `istiod` 로그를 확인합니다.

```
oc logs -f istiod-production-<POD-HASH>
```

데이터 플레인의 상태 확인

메쉬 개발자인 Farid와 Cristina는 자신에게 해당되는 도메인 내부에서 데이터 플레인의 상태를 확인할 수 있습니다.

1. Kiali를 사용해 애플리케이션, Istio 구성, 서비스, 워크로드의 상태 및 현황에 대한 **개요**와 **상세 보기**를 확인합니다.
2. Jaeger를 사용해 **잠재적인 상태 관련 문제를 식별**합니다. →
3. Prometheus를 사용해 Envoy 및 애플리케이션 **메트릭**을 쿼리합니다. **Prometheus에서 메트릭을 사용하는 방법**에 관한 추가 정보는 온라인 리포지토리에서 확인할 수 있습니다. →
4. 추가 인사이트를 얻으려면 `istio-proxy` 로깅 수준을 높이세요. Kiali에서, 또는 `oc` CLI를 통해 확장된 콘텐츠를 볼 수 있습니다.
   - `istio-proxy` 컨테이너와 인그레스/이그레스 게이트웨이를 포함한 전체 서비스 메쉬에 대한 **Envoy 액세스 로그를 활성화**합니다.
   - 다음과 같이 현재 istio-proxy 로깅 수준을 확인합니다.

     ```
     ./istioctl proxy-config log <POD NAME>
     ```

   - 필요에 따라 새로운 수준을 적용합니다.

     ```
     ./istioctl proxy-config log <POD NAME> --level http2:debug,grpc:debug
     ```

   결과로 나타난 로그를 구문 분석할 때 가능한 **응답 플래그**를 반드시 고려하세요. 수신 트래픽 흐름에 대한 전체 뷰를 얻으려면 **인그레스 액세스 로깅**을 구성해야 할 수도 있습니다.

보안 상태 확인

서비스 메쉬 보안 요구 사항이 더 복잡해짐에 따라 인증 및 트래픽 암호화에 사용되는 인증서가 올바르고 유효한지 확인하는 것이 중요합니다.

1. 5장에서는 이러한 인증서를 확인하는 두 가지 스크립트(`verify-controlplane-certs.sh` 와 `verify-dataplane-certs.sh`)를 정의했습니다. 메쉬 오퍼레이터인 Emma는 이러한 스크립트를 사용해 기대한 인증서가 마련되었는지 확인할 수 있습니다.

2. **ksniff 커뮤니티 툴**을 사용해 트래픽 수준에서 TLS가 어떤 방식으로 적용되는지 확인할 수도 있습니다.

```
WORKLOAD="istio-egressgateway-8598cbf7cb-nl68z"
NAMESPACE="istio-system-egressgw-mtls-client"
oc sniff $WORKLOAD -p -n $NAMESPACE -o output.pcap
```

3. 제공된 인증서와 관련해 발생할 수 있는 문제에 대한 `istiod` 로그를 확인합니다. 예를 들어, 아래의 로그 판독은 중간 CA 키가 비밀번호로 보호되어 있는 경우 발생하는 일반적인 오류를 보여줍니다.

```
2022-09-16T11:50:06.830472Z     error   failed to create discovery service: failed to create
CA: failed to create an istiod CA: failed to create CA KeyCertBundle (failed to parse private
key PEM: failed to parse the RSA private key)

Error: failed to create discovery service: failed to create CA: failed to create an istiod CA:
failed to create CA KeyCertBundle (failed to parse private key PEM: failed to parse the RSA
private key)
```

이 장의 리소스

- Istio 도큐멘테이션: **Envoy 기반 추적은 어떤 방식으로 작동하나요?**
- Istio 도큐멘테이션: **Envoy 및 Istiod 디버깅**
- Red Hat Developer: **서비스 메쉬 트러블슈팅 툴**
- Red Hat Customer Portal: **OpenShift 4와 함께 커뮤니티 ksniff를 사용해 포드 내에서 패킷 캡처**
- Red Hat Customer Portal: **통합된 트러블슈팅 문서 OpenShift Service Mesh 2.x**

8장

서비스 메쉬 튜닝

이 장에서는 서비스 메쉬 튜닝 기법에 대해 살펴보겠습니다.

원하는 튜닝 작업 결과 정의

Travel by Keyboard의 애플리케이션 및 플랫폼 팀은 서비스 메쉬 튜닝에 대한 비기능적 요구 사항과 원하는 튜닝 작업 결과를 제공했습니다.

애플리케이션 팀의 요구 사항

제품 소유자, 기술 리드 및 메쉬 개발자 역할을 포함하는 애플리케이션 팀은 기대되는 고객 부하를 처리하기 위해 애플리케이션 측면에서 메쉬를 튜닝해야 합니다. 이 경우 1일당 250,000건의 요청이 예상되며, 초당 요청(rps)은 최고 250건에 달합니다.

플랫폼 팀의 요구 사항

클러스터 오퍼레이터, 메쉬 오퍼레이터, 플랫폼(애플리케이션 운영자) 역할을 포함하는 플랫폼 팀은 관측성, 인그레스/이그레스 애플리케이션 런타임, 구성을 최적화하기 위해 컨트롤 플레인 측면에서 메쉬를 튜닝해야 합니다. 이 예시에서 플랫폼 팀은 모범 사례, 사이징 가이드, 서비스 메쉬 평가 및 측정을 위한 벤치마크를 정의하여 프로젝트 팀에게 가능한 한 최상의 경험을 제공하고자 합니다. 이를 위해서는 다음 질문에 답해야 합니다.

- Red Hat OpenShift Service Mesh 인스턴스의 최대 용량을 정의하려면 어떻게 해야 하는가?
- 향후 메쉬에 합류할 수 있는 애플리케이션의 최대 수를 정의하려면 어떻게 해야 하는가?
- 용량을 통제하고 평가하려면 어떤 기준과 메트릭을 사용해야 하는가?
- Red Hat OpenShift Service Mesh 인스턴스의 현재 컨트롤 플레인 및 데이터 플레인 한도는 얼마인가?

서비스 메쉬 튜닝 전제 조건 이해

서비스 메쉬의 목표, 아키텍처 원리, 프로덕션 설정에 따라 필요한 튜닝의 유형이 결정됩니다. 이 문서의 예시에서는 이러한 측면을 3장의 **최종 서비스 메쉬 프로덕션 설정**에서 정의했습니다.

클라우드 기반 환경에는 방화벽, 로드 밸런서, 컨테이너 플랫폼 등 튜닝해야 할 구성 요소가 많이 있습니다. 하지만 이 예시에서는 다음 두 가지 주요 영역을 중점적으로 다루겠습니다.

- **데이터 플레인:** 모든 `istio-proxy`(Envoy) 사이드카와 인그레스 및 이그레스 게이트웨이 구성 요소로 구성되어 있으며, 워크로드의 수신 트래픽을 처리하는 일을 담당합니다.
- **컨트롤 플레인:** `istiod` 서비스와 관측성 스택을 포함합니다. 컨트롤 플레인은 프록시가 최신 구성 및 인증서로 업데이트되도록 유지하는 업무를 담당합니다.

다음 섹션에서는 서비스 메쉬의 성능 테스트, 사이징 요구 사항 측정, 전체 튜닝 옵션에 대한 지침을 제공합니다.

서비스 메쉬 데이터 플레인의 규모 조정

애플리케이션 팀은 메모리, CPU, 스레드가 앞서 정의한 대기 시간 및 처리량을 충족하도록 인그레스 및 이그레스 워크로드와 `istio-proxy` 사이드카와 같은 데이터 플레인 구성 요소를 튜닝해야 합니다. 서비스 메쉬 데이터 플레인을 올바르게 사이징하려면 예상되는 실제 부하에 따라 일련의 시나리오를 테스트해야 합니다. 원하는 성능 결과에 도달할 때까지 다양한 구성을 부하 테스트, 평가, 조정해야 합니다.

세부적인 데이터 플레인 튜닝 실습은 온라인 리소스 리포지토리에서 제공됩니다. 이 예시에서는 테스트를 위해 메쉬를 구성하고, 기본 데이터 플레인 가치를 테스트하고, `istio-proxy` 스레드, `CPU` 리소스, `MEM` 리소스, `mysql` 데이터베이스를 튜닝하고, 튜닝된 구성을 테스트합니다. →

데이터 플레인 모니터링

사이징에 관해 정보에 근거한 의사 결정을 하려면 데이터 플레인의 특정 측면을 모니터링해야 합니다. Istio는 HTTP, HTTP/2 GRPC, TCP 트래픽을 모니터링할 수 있는 **메트릭**을 정의합니다. 주요 메트릭은 다음과 같습니다.

- `istio_requests_total`: 요청의 총 개수를 측정하는 카운터
- `istio_request_duration_milliseconds`: 요청의 대기 시간을 측정하는 분배

Grafana 및 **Kiali**를 사용해 이러한 메트릭을 모니터링하고 이 메트릭에 대해 Prometheus 경고를 설정하여 데이터 플레인 튜닝을 지원할 수도 있습니다. 이 **메트릭**에 대한 추가 세부 정보는 온라인 리소스 리포지토리에서 제공됩니다.

끝으로, **세부적인 데이터 플레인 튜닝 실습**에서 제공하는 **containers-mem-cpu.sh 스크립트**를 사용해 `istio-proxy`와 주요 워크로드 컨테이너의 CPU 및 메모리에 관한 정보를 검색할 수 있습니다. Istio 메트릭과 마찬가지로 Prometheus를 통해 액세스할 수도 있습니다.

데이터 플레인 튜닝 관련 권장 사항

각 튜닝 실습은 서로 다르지만 시작할 때 다음 사항을 고려하는 것이 좋습니다.

가용성 관련 고려 사항

- 대부분의 중요 포드는 일정 조정과 관련해 우선권이 있으므로 포드 우선권 및 선취권을 확인합니다.
- 활성 상태, 준비 상태 및 스타트업 프로브를 구성합니다.
- 각 컨테이너에 대해 알려진 한도를 사용해 컨테이너에 대해 현실성 있는 컴퓨팅 리소스를 설정하고, 이에 따라 **수평적 포드 자동 스케일러(HPA)**를 구성해야 합니다.
- 배포 전략 선택을 확인합니다.
- 관리형 애플리케이션 및 데이터베이스 연결 풀을 구성하고 튜닝합니다.

프록시(Envoy) 관련 고려 사항

- 애플리케이션 동시성이 너무 얇아 처리량을 향상할 수 없을 때는 애플리케이션 동시성을 높입니다.
- HTTP2로 가는 트래픽을 업그레이드하여 동일한 연결을 경유하는 몇 개의 요청을 멀티플렉싱하고 새로운 연결 생성과 관련된 오버헤드를 방지합니다.
- Istio `DestinationRules` 구성을 통한 클라이언트 풀 연결을 튜닝하여 네트워크의 성능을 향상합니다.
- 데이터 플레인과 컨트롤 플레인에서 불필요한 오버헤드가 발생하지 않도록 각 프록시에 필요한 구성만 전송되도록 합니다.

고처리량 워크로드는 서비스 메쉬에 추가 수요를 발생시키므로 추가 튜닝을 통해 혜택을 누릴 수 있습니다. 온라인 리소스 리포지토리에는 **고처리량 수요를 위한 튜닝**에 관한 추가 정보가 포함되어 있습니다. →

서비스 메쉬 컨트롤 플레인의 규모 조정

컨트롤 플레인 튜닝의 목표는 데이터 플레인을 효과적으로 지원하고, 확장된 데이터 플레인 용량 중에서 일정한 양을 처리하고, 관측성 스택에 필요한 리소스를 제공할 수 있도록 보장하는 것입니다. 온라인 리소스 리포지토리에는 **컨트롤 플레인 튜닝**에 대한 추가 정보뿐 아니라 **세부적인 튜닝 실습**도 포함되어 있습니다. →

컨트롤 플레인 모니터링

데이터 플레인과 마찬가지로 메트릭은 컨트롤 플레인 튜닝의 핵심 요소입니다. Istio는 컨트롤 플레인 튜닝에 도움이 되는 **메트릭**도 제공합니다. 주요 메트릭은 다음과 같습니다.

- `pilot_xds`: xDS를 사용하는 istiod에 연결된 엔드포인트의 수, 또는 컨트롤 플레인이 항상 최신 상태로 유지해야 하는 클라이언트의 수
- `pilot_xds_pushes`: 전송된 xDS 메시지의 수(빌드 및 전송 오류 포함)
- `pilot_proxy_convergence_time`: 새로운 구성을 Envoy 프록시에 푸시하는 데 걸리는 시간(단위: 밀리초)

istiod 튜닝 관련 권장 사항

컨트롤 플레인의 핵심 기능 중 하나는 데이터 플레인을 지원하는 것이므로 데이터 플레인의 규모가 예를 들어 수백 개의 포드로 크게 증가하는 경우 `istiod` 튜닝을 조정해야 합니다. 이러한 시나리오에서는 다음과 같이 수행하는 것이 좋습니다.

1. 새로운 요구 사항에 다른 모델이 더 적합할 수 있으므로 서비스 메쉬의 **배포 모델**을 검토합니다.
2. 사이드카 리소스를 적용하여 동일한 메쉬 내의 무관한 네임스페이스에 대한 리소스 구성의 분리된 가시성을 허용합니다.
3. `istiod` 구성 요소에 대한 HPA 설정을 조정하여 xDS 클라이언트 수의 사전 정의된 증가를 허용합니다.

관측성 스택에 대한 용량 계획 수립

관측성 스택 설정은 컨트롤 플레인의 사이징 및 튜닝에도 영향을 미칩니다. 관측성 스택에 대한 용량 계획 수립에는 Kiali, Jaeger, ElasticSearch, Prometheus, Grafana 등 런타임 구성 요소 사이징과 메트릭, 그래프, 추적 지속성을 위한 관련 장기 스토리지가 수반됩니다. 이러한 용량 요구 사항은 데이터 플레인의 규모, 수신 요청의 수, 메트릭 및 추적에 대한 캡처 및 유지 설정에 따라

좌우됩니다. 온라인 리소스 리포지토리에는 **세부적인 관측성 스택 튜닝 실습** 이 포함되어 있습니다. 이 실습에서는 예시를 위해 최종 프로덕션 설정을 확인하고, 확정된 비기능적 요구 사항에 부합하도록 설정을 조정합니다. →

서비스 메쉬 계층 전반에서 튜닝 수행

지금까지 용량 요구 사항과 기본적인 튜닝을 위한 지침을 제공했습니다. 하지만 서비스 메쉬를 미세 튜닝하려면 컨트롤 플레인 계층과 데이터 플레인 계층에 모두 적용되는 측면도 고려해야 합니다. 이러한 측면(예: 클러스터 안팎으로의 통신에 대한 TLS 설정, 서비스 간 통신 요구 사항, 부트스트래핑 구성)에 대한 세부적인 이해는 미세 튜닝을 위한 일련의 안정적인 벤치마크를 생성하는 데 필요합니다.

이 장의 리소스

`istio-proxy` 메트릭

- Istio 도큐멘테이션: **프록시 수준 메트릭**
- Istio 도큐멘테이션: **Istio 표준 메트릭**
- Istio 도큐멘테이션: **Envoy 통계**
- Envoy 도큐멘테이션: **통계 개요**

`istiod` 메트릭

- Istio 도큐멘테이션: **내보낸 메트릭**

관측성 스택

- Kiali 도큐멘테이션: **Prometheus 튜닝**
- Istio 도큐멘테이션: **프로덕션 스케일 모니터링을 위해 Prometheus 사용**

메쉬 성능 테스트 리소스

- IstioCon 프레젠테이션: **멀티클러스터 Istio를 이용해 1M RPS로 스케일링**
- 툴: **Locust 부하 테스트 툴**

사례 사이징 및 튜닝

- Istio 도큐멘테이션: **성능 및 확장성**
- Istio 블로그: **모범 사례: 서비스 메쉬 성능 벤치마킹**
- Red Hat OpenShift 도큐멘테이션: **네트워킹 최적화**

9장

서비스 메쉬를 새로운 버전으로 업그레이드

이 장에서는 서비스 메쉬를 새로운 버전으로 업그레이드하는 방법에 대해 살펴보겠습니다.

현재 사용 중인 버전 식별

서비스 메쉬 업그레이드 작업 계획을 수립할 때는 현재 어떤 서비스 메쉬 구성 요소 버전을 사용 중인지 아는 것이 중요합니다. 버전에 관한 자세한 내용은 Red Hat OpenShift 도큐멘테이션의 **서비스 메쉬 버전에 대한 이해** 및 **버전 관리가 서비스 메쉬 업그레이드에 미치는 영향** 섹션에서 확인할 수 있습니다.

Red Hat OpenShift Service Mesh에는 버전 관리의 영향을 받는 다음 네 가지 구성 요소가 포함되어 있습니다.

- Red Hat OpenShift Service Mesh 오퍼레이터
- Red Hat OpenShift Service Mesh 컨트롤 플레인
- 데이터 플레인 사이드카 이미지
- 관측성 스택 구성 요소 오퍼레이터

Red Hat OpenShift Service Mesh 오퍼레이터 버전

Red Hat OpenShift Service Mesh 오퍼레이터 버전에 따라 다음 사항이 결정됩니다.

- 메쉬를 운영하는 데 사용되는 포드 이미지
- 기본적으로 포함되는 **특정 구성 요소의 버전**(예: Istio, Envoy Proxy, Jaeger, Kiali)
- 지원되는 Red Hat OpenShift Service Mesh 컨트롤 플레인 버전
- 지원되는 **사용자 정의 리소스 정의(CRD)**

이 예시에서 다음과 같이 현재의 Red Hat OpenShift Service Mesh 오퍼레이터를 식별합니다.

```
oc -n prod-istio-system get csv|grep servicemesh
```

Red Hat OpenShift Service Mesh 컨트롤 플레인 버전

Red Hat OpenShift Service Mesh 컨트롤 플레인 버전에 따라 컨트롤 플레인 기능의 가용성 및 구성이 결정됩니다. 또한 사용될 Red Hat OpenShift Service Mesh의 버전이 결정되며, 그 결과 Red Hat OpenShift Service Mesh 오퍼레이터가 무엇을 배포할지 결정됩니다. 컨트롤 플레인 버전을 업데이트하면 인그레스 및 이그레스 게이트웨이와 `istiod`를 포함한 모든 컨트롤 플레인 구성 요소의 버전과 데이터 플레인 포드에 주입된 사이드카 이미지의 버전에 영향을 미칩니다.

최신 Red Hat OpenShift Service Mesh 오퍼레이터는 현재 여러 개의 컨트롤 플레인 버전을 지원하므로 지원되는 버전을 자동으로 업데이트하지 않습니다. 따라서 Red Hat OpenShift Service Mesh 오퍼레이터를 업그레이드하는 것과 더불어 컨트롤 플레인 버전을 업데이트하지 않아도 됩니다. 컨트롤 플레인 버전에 관한 자세한 내용은 Red Hat OpenShift 도큐멘테이션의 **컨트롤 플레인 업그레이드** 섹션에서 확인할 수 있습니다.

관측성 스택 구성 요소 오퍼레이터 버전

Red Hat OpenShift Service Mesh는 Jaeger, ElasticSearch, Prometheus, Kiali 구성 요소에 의존하여 메쉬에 대한 관측성을 제공합니다. Red Hat OpenShift Service Mesh 오퍼레이터 및 컨트롤 플레인 버전에 따라 필요한 관측성 구성 요소의 버전이 결정됩니다. 또한 각 관측성 구성 요소 오퍼레이터가 관리하는 리소스는 설치된 해당 오퍼레이터 버전과 호환되어야 합니다. 결과적으로 이러한 리소스는 오퍼레이터가 업그레이드될 때 함께 업데이트되어야 합니다.

서비스 메쉬 업그레이드

버전 종속성에 따라 서비스 메쉬 업그레이드에는 필수 및 선택적 구성 요소 업그레이드가 포함됩니다.

표 7. 서비스 메쉬 업그레이드를 위한 필수 및 선택적 구성 요소 업그레이드

구성 요소	업그레이드 요구 사항
Red Hat OpenShift Service Mesh 오퍼레이터	필수 업그레이드
관측성 구성 요소 오퍼레이터	필수 업그레이드
Red Hat OpenShift Service Mesh 컨트롤 플레인	선택적 업그레이드
사이드카 이미지(재시작을 거침)	선택적 업그레이드

Red Hat OpenShift Service Mesh 오퍼레이터 업그레이드

새로운 오퍼레이터 버전이 포함된 새로운 카탈로그 소스가 설치될 때마다 Red Hat OpenShift는 새로운 설치 계획을 자동으로 생성합니다. 최초 Red Hat OpenShift 설치 중에 수동 업데이트 승인을 선택한 경우 업데이트를 시작하려면 설치 계획을 수동으로 수락해야 합니다. **서로 분리된 여러 환경에서의 업그레이드**에 관한 정보와 추가 리소스는 온라인 리소스 리포지토리에서 확인할 수 있습니다.

설치 계획이 자동으로 또는 수동으로 수락되면 업그레이드 절차가 시작됩니다. 먼저 오퍼레이터가 **업그레이드됩니다**. 이어서 모든 컨트롤 플레인 구성 요소가 다시 시작되어 새로운 Red Hat OpenShift Service Mesh 오퍼레이터 버전을 현재 컨트롤 플레인 버전으로 수신합니다. 새로운 Red Hat OpenShift Service Mesh 오퍼레이터 버전은 컨트롤 플레인이 업그레이드된 후에만 활성화됩니다.

컨트롤 플레인 업그레이드

다음 단계는 컨트롤 플레인을 업그레이드하는 것입니다. 이를 위해서는 다음과 같이 컨트롤 플레인 리소스의 버전 필드(`.spec.version`)를 업데이트해야 합니다.

```
$ oc patch smcp <control_plane_name> --type json \
  --patch '[{"op": "replace","path":"/spec/version","value":"v2.2"}]'
```

그러면 모든 컨트롤 플레인 구성 요소(인그레스 및 이그레스 게이트웨이와 `istiod`)가 자동으로 다시 시작되고 최신 버전 이미지로 업그레이드됩니다. 새로운 컨트롤 플레인 버전이 배포되었는지 확인합니다.

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

데이터 플레인 업그레이드

그다음에는 애플리케이션 사이드카 컨테이너와 Envoy 프록시 및 구성을 업그레이드합니다. 이를 위해 다음과 같이 애플리케이션 포드를 다시 시작합니다.

```
$ oc rollout restart deployment $DEPLOYMENT-NAME
```

이 장의 리소스

- Red Hat OpenShift 도큐멘테이션: **서비스 메쉬 업그레이드**
- Red Hat OpenShift 도큐멘테이션: **사이드카 프록시 업데이트**

선임 애플리케이션 아키텍트 Stelios Kousouris 소개

소프트웨어 제공 분야에서 20년간 경력을 쌓은 Stelios Kousouris는 비즈니스 솔루션을 프로덕션 환경에 도입하고 소프트웨어 런타임을 최적화하는 데 항상 관심을 갖고 있었습니다. 현재 Red Hat OpenShift Service Mesh 네트워킹과 서버리스 배포 패러다임을 최대한 활용하는 방법을 이해하는 데 주력하고 있으며, 클라우드 환경을 위한 애플리케이션 현대화를 목표로 팀을 이끌고 있습니다.

수석 제품 마케팅 매니저 Ortwin Schneider 소개

Ortwin Schneider는 다양한 산업을 위한 개별 소프트웨어 솔루션 개발 분야에서 20년이 넘는 전문 경력을 보유하고 있습니다. 과거에는 애자일 개발 팀을 이끌면서 고객이 비즈니스 가치를 제공하고 민첩성을 지속하도록 지원했으며, 하이브리드 클라우드 및 클라우드 네이티브 모델을 사용하는 소프트웨어 아키텍처에 관심이 많습니다. Ortwin은 현재 Red Hat의 Red Hat OpenShift 팀에서 수석 기술 마케팅 매니저로 일하면서 클라우드 네이티브 솔루션 패턴을 만드는 데 주력하고 있습니다.

206057_0123_KVM

Red Hat