CONFIDENTIAL designator

Red Hat OpenShift

Service Mesh

Overview

Ryan Noblett

Senior Specialist Solution Architect

Last Update: Dec, 2025 (For 3.2)

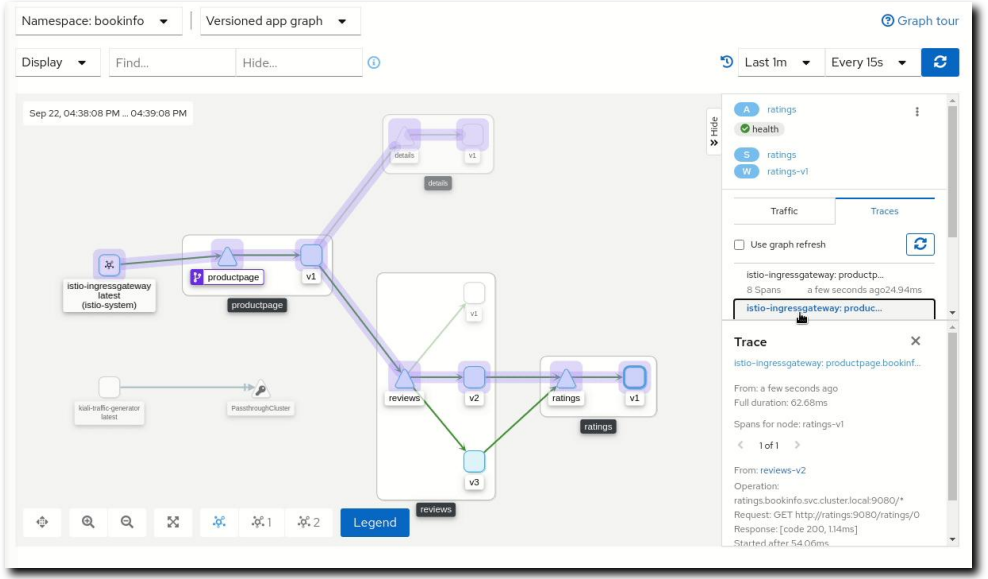
Red Hat

Why service mesh?

Red Hat

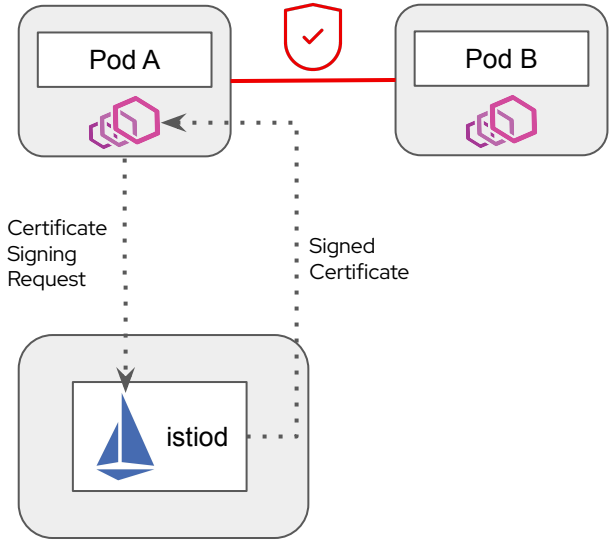
Instant Traffic Visibility

- Out of the box:
  - Request/Response Metrics
  - Distributed Tracing
  - Request logging
- Kiali dashboard provides topology and traffic visualization
- Feeds OpenShift Observability for visibility, monitoring and analysis with the rest of our apps



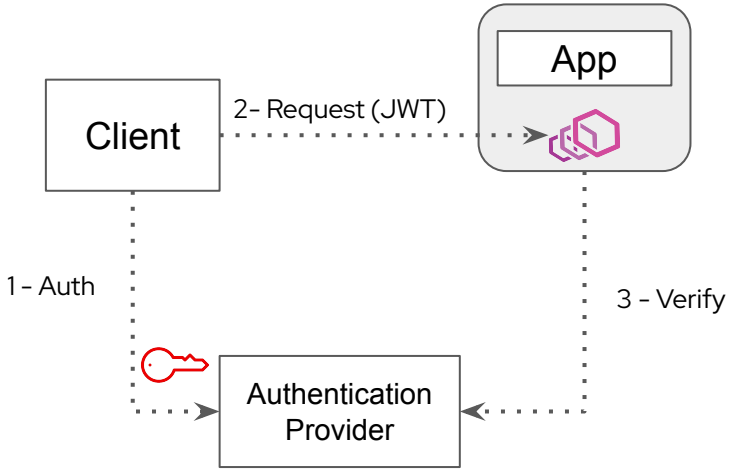
Automated & Enforced End-to-End Encryption

- Service mesh provides a standard approach for **end-to-end encryption at scale**
- mTLS **encryption** can be applied and enforce across all services - **at the workload level**.
- Each workload is given a **unique identity**
- X.509 certificates are automatically created, distributed and rotated **for all workloads**
- Integration with corporate issuers, such as Hashicorp Vault can be made with **cert-manager**



Add Request Authentication with JWT Tokens

- JWTs (JSON Web Tokens) enable user authentication without the need for server stored sessions.
- Istio Authentication policies enable request-level JWT verification to be added independent of applications
- Integrates with a custom authentication sources such as OpenID Connect providers
  - E.g. Keycloak, Auth0, etc.



Red Hat

Fine-Grain Zero-Trust Security

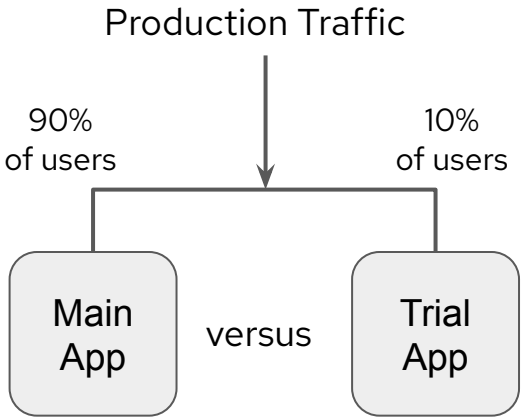
- Service mesh enables **concise security policies for true zero-trust**
- Traditional network security uses firewalls and broad rules (IP ranges, port ranges, namespaces, etc).
- With service mesh, **each application has a unique identity** created and managed by Istio.
- This enables policies to be written in terms of specific application identities and L7 behaviors.



Testing in Production

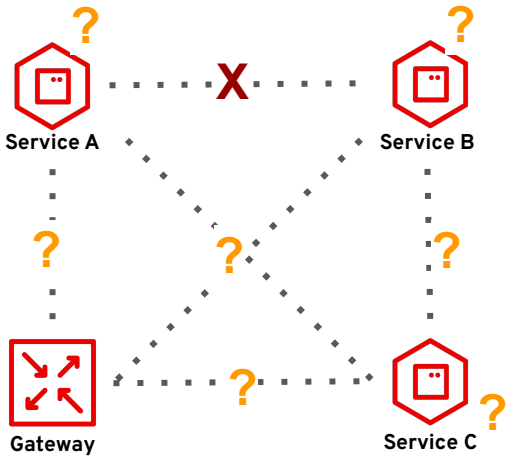
Blue/Green, Canary Rollouts & A/B Testing

- Even with testing, production can bring surprises:
  - Real data produces unanticipated scenarios.
  - New features may have been built on false assumptions.
- Service Mesh is **part of the CI/CD tool chain**.
- New releases are automatically deployed and configured to receive a small portion of production traffic.
- Based on how the new version performs, it is either promoted to main or reverted.
- This allows the customer to **constantly test the effectiveness of new releases using real production data**



Microservice Reliability & Resilience

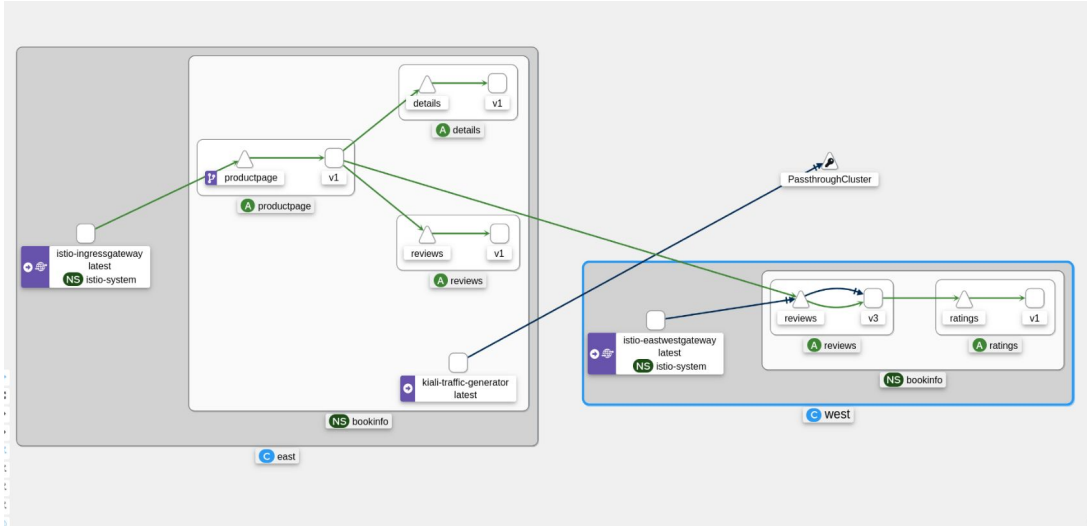
- Failures are inevitable in distributed systems:
  - Network interruptions, resource exhaustion, intermittent bugs, physical outages and more we can't predict!
- Istio automates the use of **retries**, **timeouts**, **circuit breakers** to mitigate failures:
  - Prevents cascading failures
  - Manage momentary outages so that users aren't impacted
  - Help systems to self-heal with Kubernetes pod restarts



Service Mesh across clusters

Add high-availability to your service mesh

- Mesh features may be stretched across multiple clusters.
- Enables load balancing for **high-availability** with fine-tuning based on:
  - Region (e.g. “us-east”)
  - Zone (e.g. an availability zone)
  - Sub-Zone (e.g. “rack 1”)
- Kiali provides a multi-cluster view with a common Prometheus metrics store



Red Hat

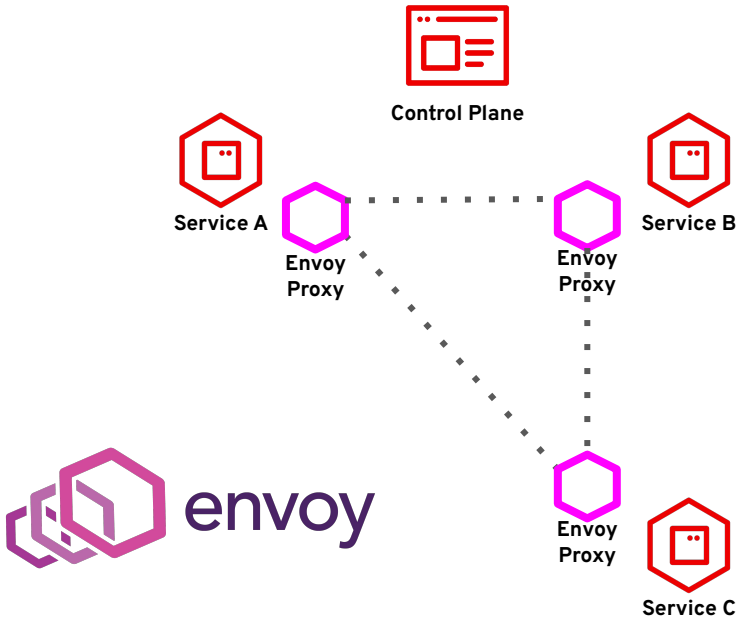
How a service mesh works:

Sidecars vs ambient mode dataplanes

Red Hat

Traditional sidecar mode

- All service pods are given an **Envoy proxy** as a sidecar container**.** Together, these form the **Data Plane**.
- All communications occur through these **proxies.**
- This creates a **mesh** of communication that has full visibility and control of all traffic.
- The proxies - and thus the mesh, are configured and managed by a central **Control Plane**.



Benefits and Tradeoffs of Envoy sidecar proxies

Sidecars have challenges, which ambient aims to address

Benefits

- A simple deployment model - one pod per proxy
- Highly customizable through Istio and Envoy APIs
- Resource usage efficient ***when carefully managed***
- Tolerable latency addition for most user facing applications

Trade offs

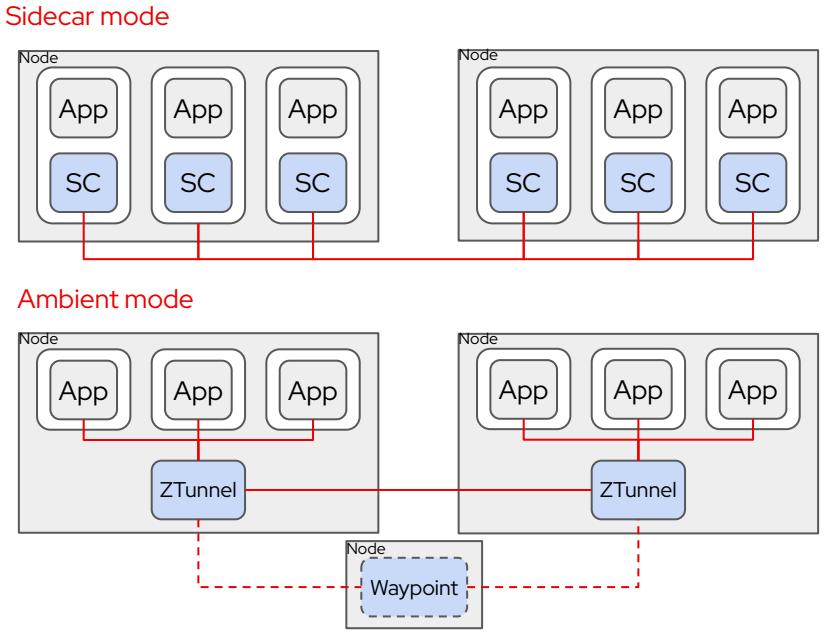
- With Envoy's flexibility, comes complexity!
- One proxy per pod means a **lot of proxies**!
- Without careful management, **resource usage can grow out of control!**
- Requires modifying pods to inject proxies
- **Perf impacts** for low latency, high throughput applications may be unacceptable
- May be "**too heavy**" if only some mesh features needed.

Red Hat

Istio's Ambient mode

Service mesh without sidecars

- **Significantly lower resource usage** compared to sidecar mode
- **Easier to adopt**, as application workloads do not need to be modified to add sidecars
- Two-levels of adoption:
  - ZTunnel (L4) for **lightweight pod-to-pod mTLS encryption** and L4 features
  - ZTunnel (L4) + Waypoint (L7) for a full suite of mesh features
- **Generally available with OpenShift Service Mesh 3.2**



Istio's Ambient mode: Benefits and Trade offs

Ambient mode has great benefits, but also tradeoffs

Benefits

- Significantly reduces resource usage
  - 90%+ memory; 50%+ CPU vs sidecars
- Significant scalability improvements
  - Potential for 100k+ workloads
- Very lightweight pod-to-pod mTLS encryption
  - Enables zero-trust
- Easier for applications to adopt - no changes to pods, no application restarts
- Potential performance improvements (WIP)

Trade offs

- Relatively new dataplane and proxy(ztunnel)
- Not all feature available yet
  - Multi-cluster(alpha)
  - Multi-control plane not supported
- L7 mesh feature require deploying a secondary "waypoint" proxy (extra network hop)
- Upgrading ztunnel node proxy may result in dropped traffic if node is not drained

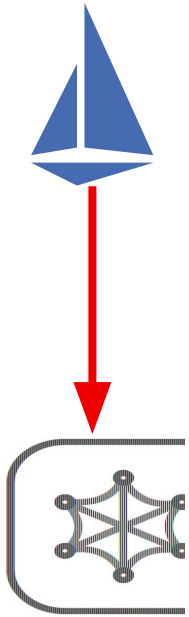
Red Hat

OpenShift Service Mesh 3

Red Hat

OpenShift Service Mesh 3 in a Nutshell

- **The OpenShift Service Mesh operator**
  - Supports Istio lifecycle management with both “in-place” and “revision-based” (canary) upgrades
  - Uses community Istio configuration with validation
- Red Hat Productized **Istio**
  - Security patches for ***a minimum of 18 months***
- **Kiali** and **OpenShift Service Mesh Console Plugin**
  - Console for visualizing, validating and troubleshoot Istio
- Supported **OpenShift platform integrations**
  - Observability, security, GitOps, networking, virtualization and more



*We validate & document community features on OpenShift to declare support, while fixes/enhancements are done in the upstream Istio.io project. See feature support table for details.

Red Hat

The Istio.io Project

- Founded by Google and IBM in 2017
- One of the most widely used and feature-rich service mesh projects
    - Based on the **highly customizable** Envoy proxy
- A graduated Cloud Native Computing Foundation (CNCF) project
- **Red Hat the 3rd biggest contributor in 2025**
- In addition to Red Hat, maintained by vendors including IBM, Google, Microsoft, Intel, Ericsson, Salesforce, Solo.io and Tetrate.io



Service Mesh 3 = Red Hat Productized Istio

- **Based on the community Istio.io project**
- Productized as part of the **openshift-service-mesh** GitHub organization:
  - A downstream distribution of Istio to ensure long term support
  - OpenShift specific test and debug infrastructure
  - Patches/Fixes/CVEs for releases out of community support
- **Supported with security patches for a minimum of 18 months per Istio version.**
  - Compare to 6-7 months for community Istio or 1 year for most Istio vendors

Red Hat

OpenSSL for Encryption

FIPS ready

- OpenShift Service Mesh uses Red Hat Enterprise Linux (RHEL)'s OpenSSL library for encryption in place of upstream Istio/Envoy's BoringSSL library
- This ensures OpenShift Service Mesh is designed for FIPS and supported on FIPS enabled OpenShift clusters
- Red Hat submits RHEL OpenSSL to NIST for FIPS 140-2/140-3 validation on x86_64, ppc64le, and s390x architectures.



The OpenShift Service Mesh 3 Operator

- OpenShift Service Mesh installs and manages Istio components using an operator.
- This includes:
  - Declarative custom resource definitions (CRDs) based on Istio's helm values with validations:
    - **`Istio`** to manage Istio control planes
    - **`IstioCNI`** to manage the istio-cni deamonset used for pod network configuration
  - Support for "in-place" and "revision-based" (canary) Istio control plane upgrade procedures:
    - **`IstioRevision`** and **`IstioRevisionTag`** resources manage control planes and groups of workloads..
  - Support for **multiple independent control planes in a single cluster**.
- Based on the community maintained "Sail Operator" for Istio that is part of the istio-ecosystem organization

Red Hat

`Istio` control plane configuration

- Manages the configuration and lifecycle of Istio control planes.
- A cluster-wide resource, though multiple instances with unique names may run in parallel.
- Notable fields:
  - **`version`**: Uses Istio releases, may use the form "v1.24-latest" for automatic patch updates
  - **`updateStrategy`**: "InPlace" and "RevisionBased" upgrades are supported.
  - **`values`**: Takes Istio Helm values configuration (as used with community Istio) + validations

```
apiVersion: sailoperator.io/v1
kind: Istio
metadata:
  name: default
spec:
  version: v1.24.3
  namespace: istio-system
  updateStrategy:
    type: InPlace
  values:
    pilot:
      resources:
        requests:
          cpu: 100m
          memory: 1024Mi
```


Red Hat

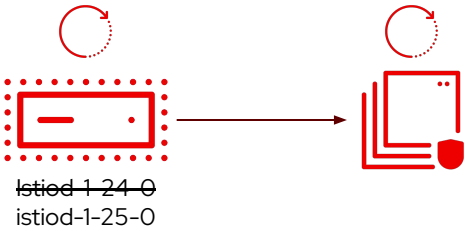
In Place vs Revision Based Control Plane Upgrades

The OpenShift Service Mesh operator supports two update strategies

In Place Upgrades

1. The Istio control plane updates after a restart.

2. Workloads and gateways are restarted to update proxies.

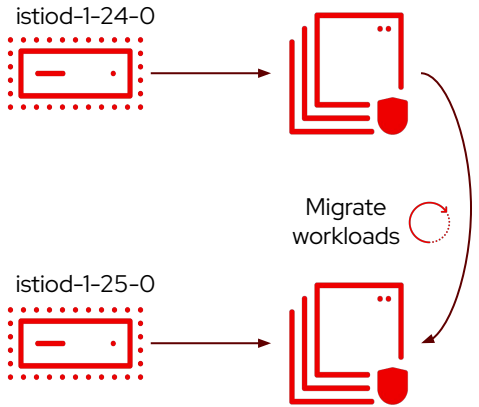


Revision Based Upgrades

1. An updated Istio control plane is created in parallel with the original.

2. Workloads are incrementally migrated to the new control plane.

3. The original control plane is removed.

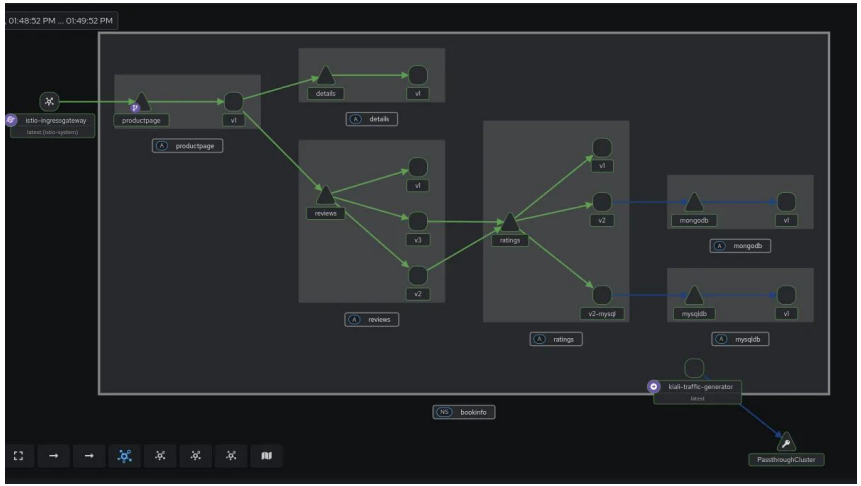


Red Hat

The Kiali Console

Configure, visualize, validate and troubleshoot your mesh with Kiali

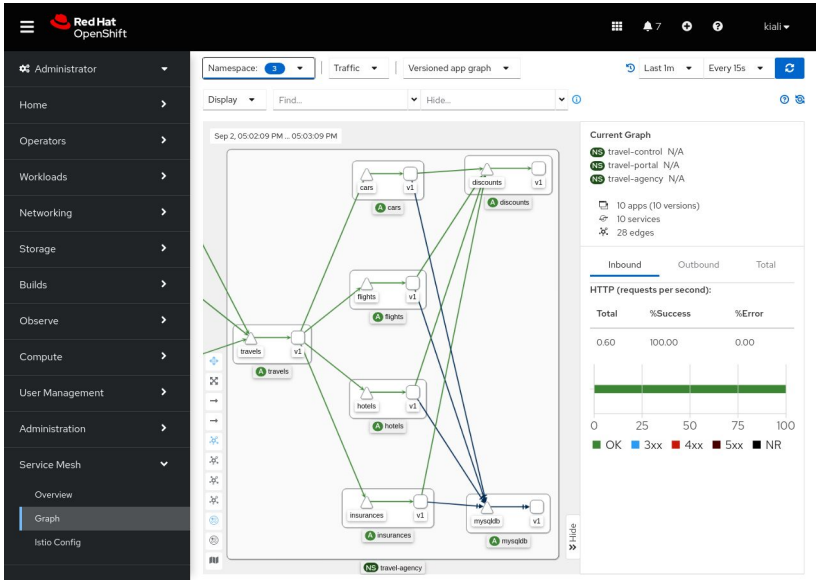
- OpenShift Service Mesh includes support for the Kiali console:
  - Visualize services and traffic patterns
  - Monitor the health of Istio components and addons
  - Troubleshoot Istio components and services with logs, metrics and traces
  - Catch errors in Istio configuration with validations
- Deployed with the Kiali operator provided by Red Hat.
- Integrates with an Istio control plane, metrics, logging and distributed tracing
- Multi-cluster support with a common telemetry store



Red Hat

OpenShift Service Mesh Console Plugin

Service Mesh in the OpenShift Console



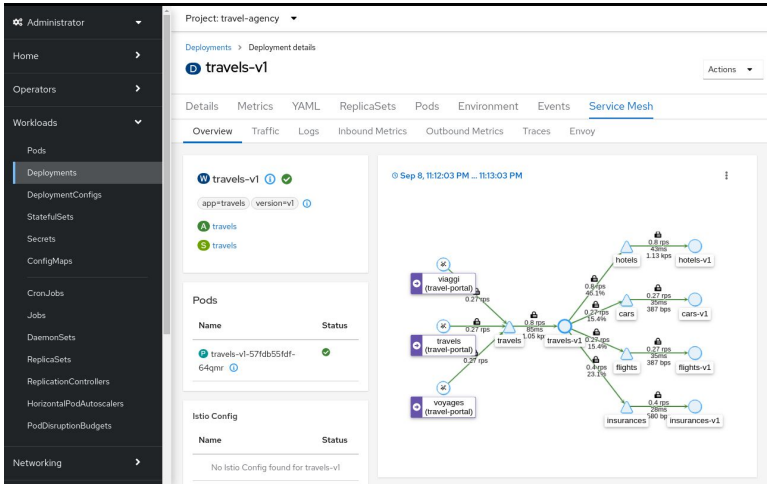
- Adds a service mesh console to the OpenShift Console, including the Kiali topology view
- Adds a “Service Mesh” tab to the navigation bar and workload pages
- Adds Istio and Envoy details to Workload, Deployment, Networking and Service pages
- Configured via the Kiali operator provided by Red Hat

OpenShift Platform Integrations

A service mesh is part of an application platform, and not used in isolation

OpenShift Service Mesh is part of a unified platform experience across the OpenShift ecosystem, including integrations with:

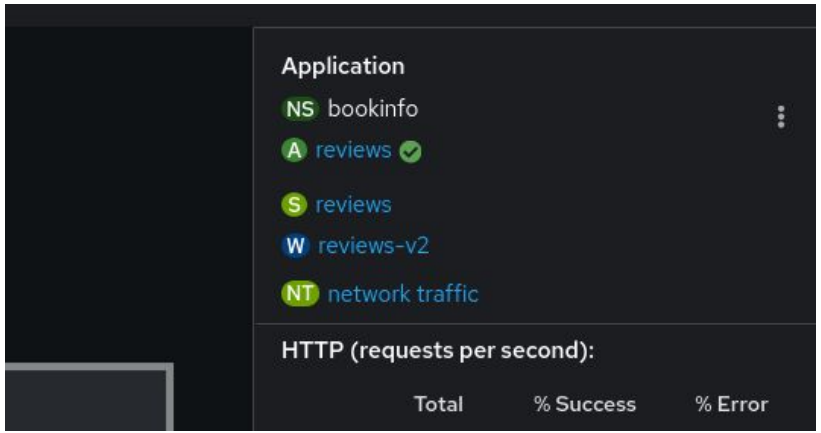
- OpenShift Ingress & Kubernetes Gateway API
- OpenShift Observability
  - Logs, metrics, distributed tracing and more
- Certificate & Identity Management
- GitOps(Argo Rollouts) and Developer Workflows
- Console (OpenShift Service Mesh Console plugin)
- OpenShift Virtualization with service mesh



Red Hat

Service Mesh Console and Network Observability

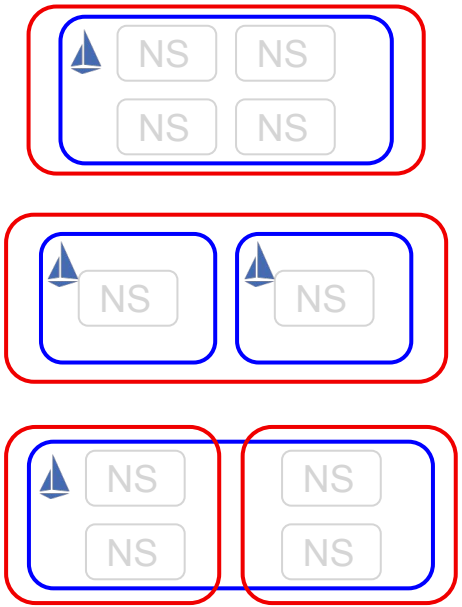
OpenShift Service Mesh Console - a Kiali plugin for the OpenShift Console



- The OpenShift NetObserv operator provides lower level (L3/4) traffic metrics and visuals using eBPF
- A “network traffic” (NT) badge and link provide users with direct access to the network observability tab inside of Deployments when it is installed with OpenShift Service Mesh

Service Mesh at Scale

Scale and Multi-tenancy with Service Mesh



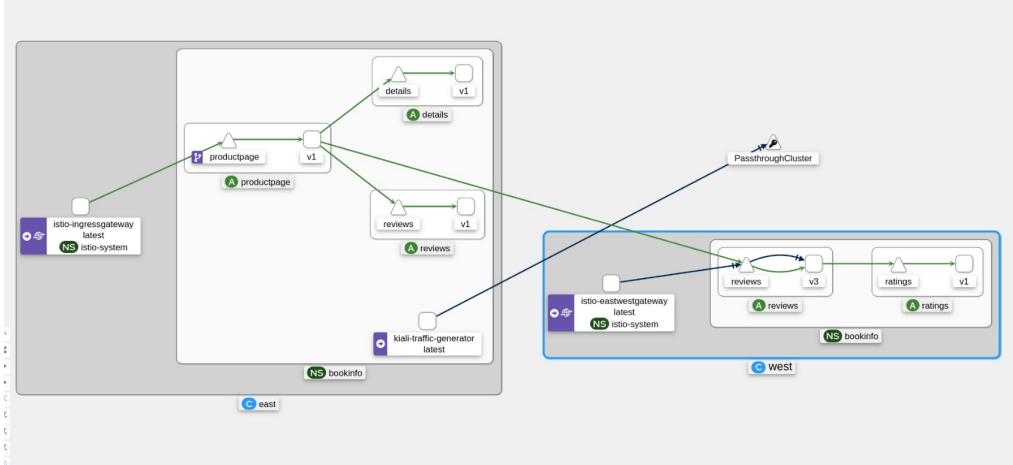
- A service mesh can scale to support thousands of services across many teams, but it must be done with planning.
  - By default, Istio is not configured for scale.
- To scale, a mesh must be divided based on domain, organization and/or administrative considerations
  - Similar to multi-tenancy
- Approaches to scaling and multi-tenancy:
  - Single service mesh scoping & isolation
  - Multiple meshes within a cluster
  - Multi-cluster topologies
- These can be combined as needs evolve
- All supported with Red Hat OpenShift Service Mesh

Red Hat

Service Mesh for High-Availability

Istio's multi-cluster service mesh topologies

- OpenShift Service Mesh supports Istio's multi-cluster topologies.
- Topologies depend on control plane deployment:
  - Multi-primary control planes
  - Primary-remote control planes
  - External control planes
- All services shared by default, according to "namespace sameness"
- Primary use case: high-availability



Red Hat