OpenShift Service Mesh

Anisse Tiouajni
Specialist Solution Architect
atiouajn@redhat.com

Octobre 2025

Red Hat

AGENDA

- Introduction / Rappel concept
- Présentation d’OpenShift Service Mesh
- Cas d'utilisation et exemples concrets
- Démonstration

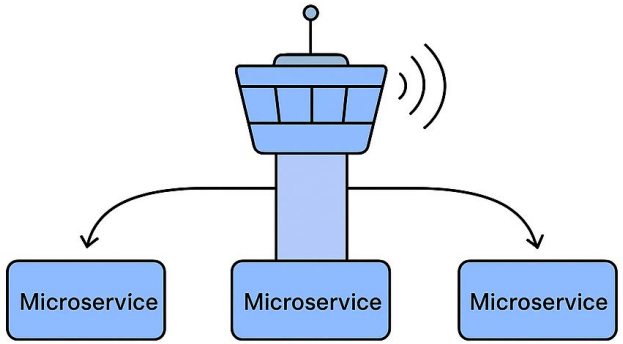
Red Hat

Définition du Service Mesh

Un **Service Mesh** est une **couche réseau intelligente** qui rend les communications entre microservices **plus sûres, observables et contrôlables**, sans changer le code applicatif.

C'est comme une tour de contrôle du trafic entre microservices :

elle observe, sécurise et dirige les échanges réseau, pendant que les applications se concentrent sur leur métier.



Version plus imagée

Red Hat

Pourquoi avons-nous besoin d'un Service Mesh ?

Résilience pour les Users

- **Disponibilité**:
  L'application doit être disponible.
- **Performance**:
  L'application doit fonctionner dans le cadre de certains paramètres définis (ex : temps de réponse).
- **Sécurité:**
  L'accès aux données doit être sécurisé et empêché de toute attaque malveillante sans surveillance.
- **Graceful error handling**:
  Si le service externe n'est pas disponible, comment le gérer correctement sans bloquer l'application.

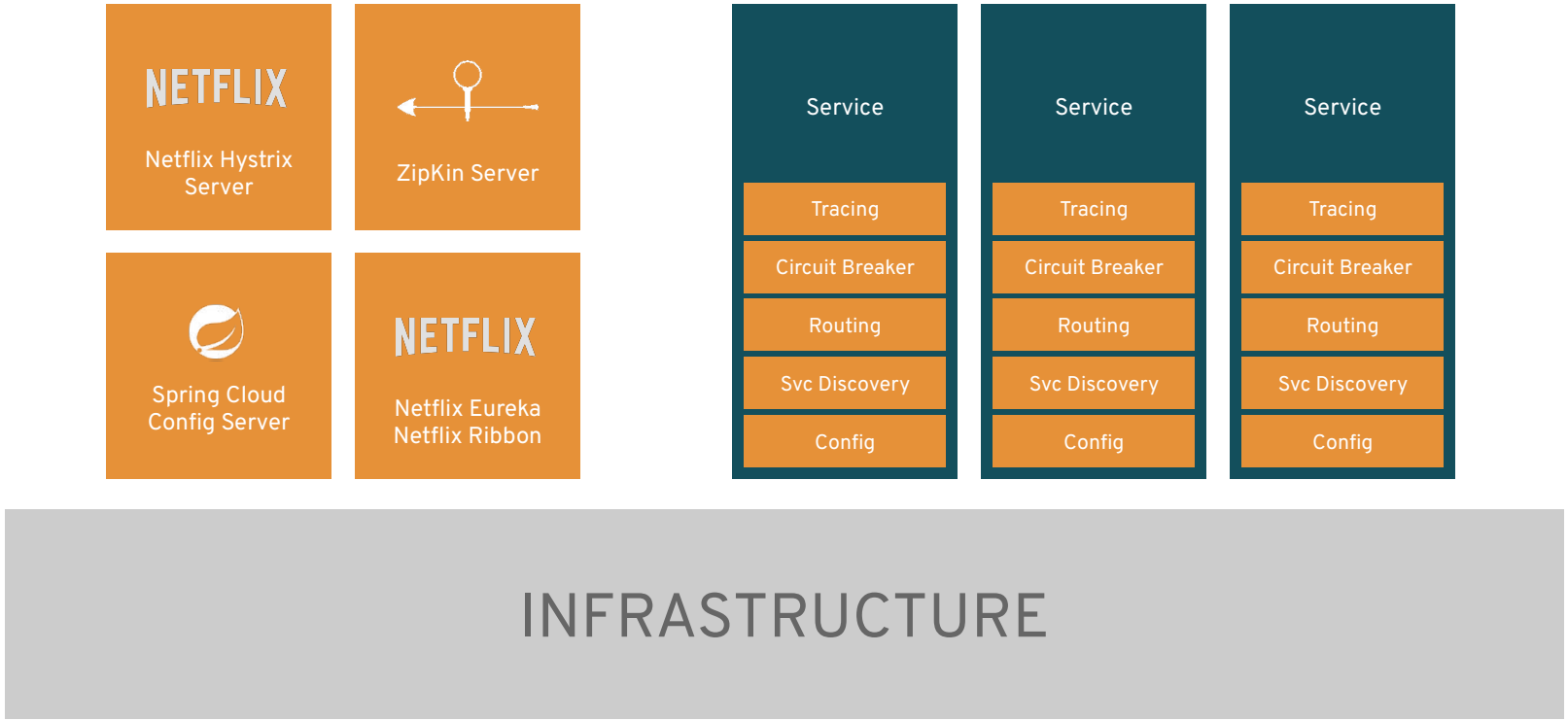
Service Mesh pour le DevSecOps

- **Découverte automatique.**
- **Surveillance**:
  traçage, observabilité.
- **Tests** :
  unitaire, intégration, régression, chaos.
- **Déploiement** :
  automatisé, A/B, canari, miroir.
- **Sécurité** :
  qui peut accéder à quoi.

Red Hat

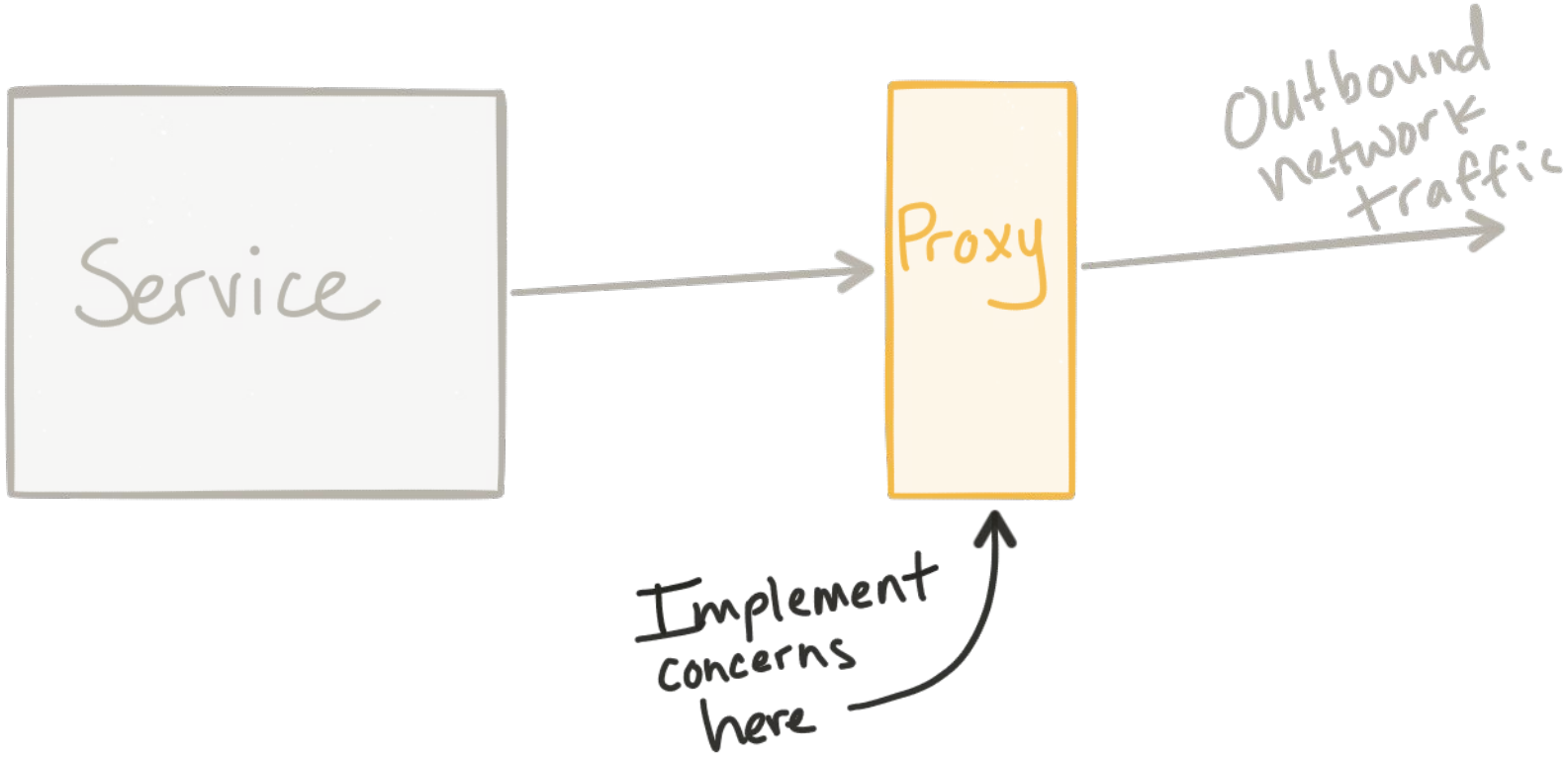
La première approche orientée librairies

Netflix OSS and Spring Cloud



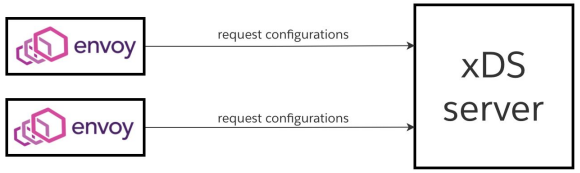
Red Hat

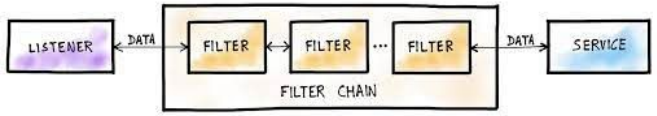
Pourquoi ne pas mettre toutes ces préoccupations au même endroit, indépendamment du language ?



Le projet Envoy

- Envoy est un proxy L7
- Open Source, sous la licence Apache 2.0
- Conçu pour les architectures modernes orientées services.
- Le filtrage est fait en L3 et L4
- Développée en C++
- Configuration dynamique via xDS APIs





Red Hat

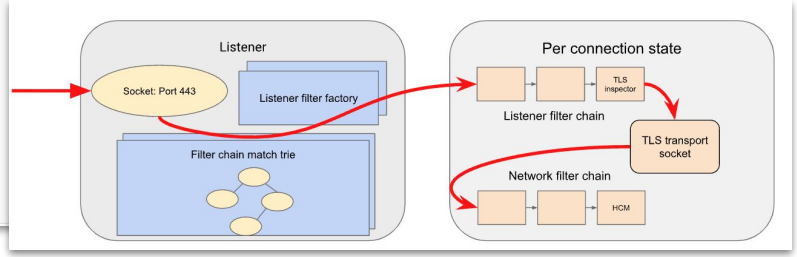
Le projet Envoy

Les fonctionnalités clés :

- Stratégies de routage avancées
- Équilibrage de charge
- Transformation des requêtes et des réponses
  - Injection de défaillance (Rejet des requêtes, retardement des réponses, modification du codes HTTP)
- Contrôle d'accès et sécurité
  - circuit breaker
  - limites de débit
  - délai de connexion
- Observabilité

```yaml
admin:
  address:
    socket_address: { address: 127.0.0.1, port_value: 9901 }

static_resources:
  listeners:
  - name: listener_0
    address:
      socket_address: { address: 127.0.0.1, port_value: 10000 }
    filter_chains:
    - filters:
      - name: envoy.filters.network.http_connection_manager
        typed_config:
          "@type": type.googleapis.com/envoy.extensions.filters.network.http_connection_manager.v3.HttpConnectionManager
          stat_prefix: ingress_http
          codec_type: AUTO
          route_config:
            name: local_route
            virtual_hosts:
            - name: local_service
              domains: ["*"]
              routes:
              - match: { prefix: "/" }
                route: { cluster: some_service }
          http_filters:
          - name: envoy.filters.http.router
            typed_config:
              "@type": type.googleapis.com/envoy.extensions.filters.http.router.v3.Router
  clusters:
  - name: some_service
    connect_timeout: 0.25s
    type: STATIC
    lb_policy: ROUND_ROBIN
    load_assignment:
      cluster_name: some_service
      endpoints:
      - lb_endpoints:
        - endpoint:
            address:
              socket_address:
                address: 127.0.0.1
                port_value: 1234
```

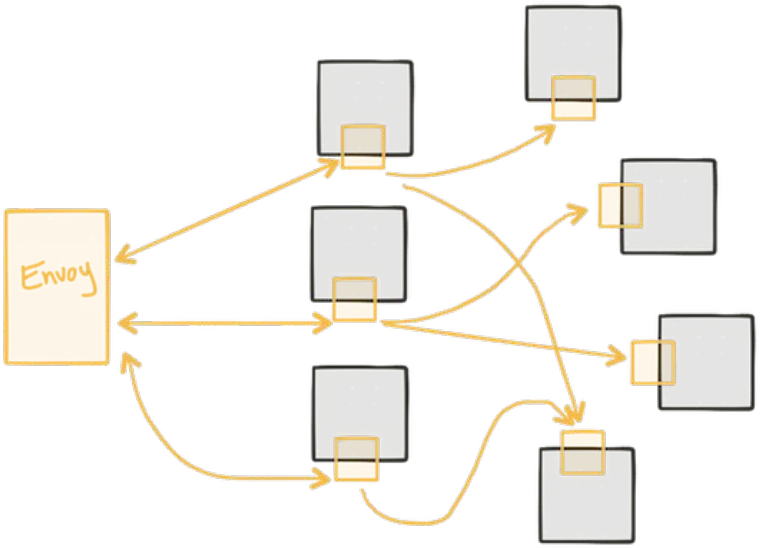


Pourquoi ne pas généraliser cette approche en utilisant les conteneurs sidecar ?

envoy

ENVOY IS AN OPEN SOURCE EDGE AND SERVICE PROXY, DESIGNED FOR CLOUD-NATIVE APPLICATIONS

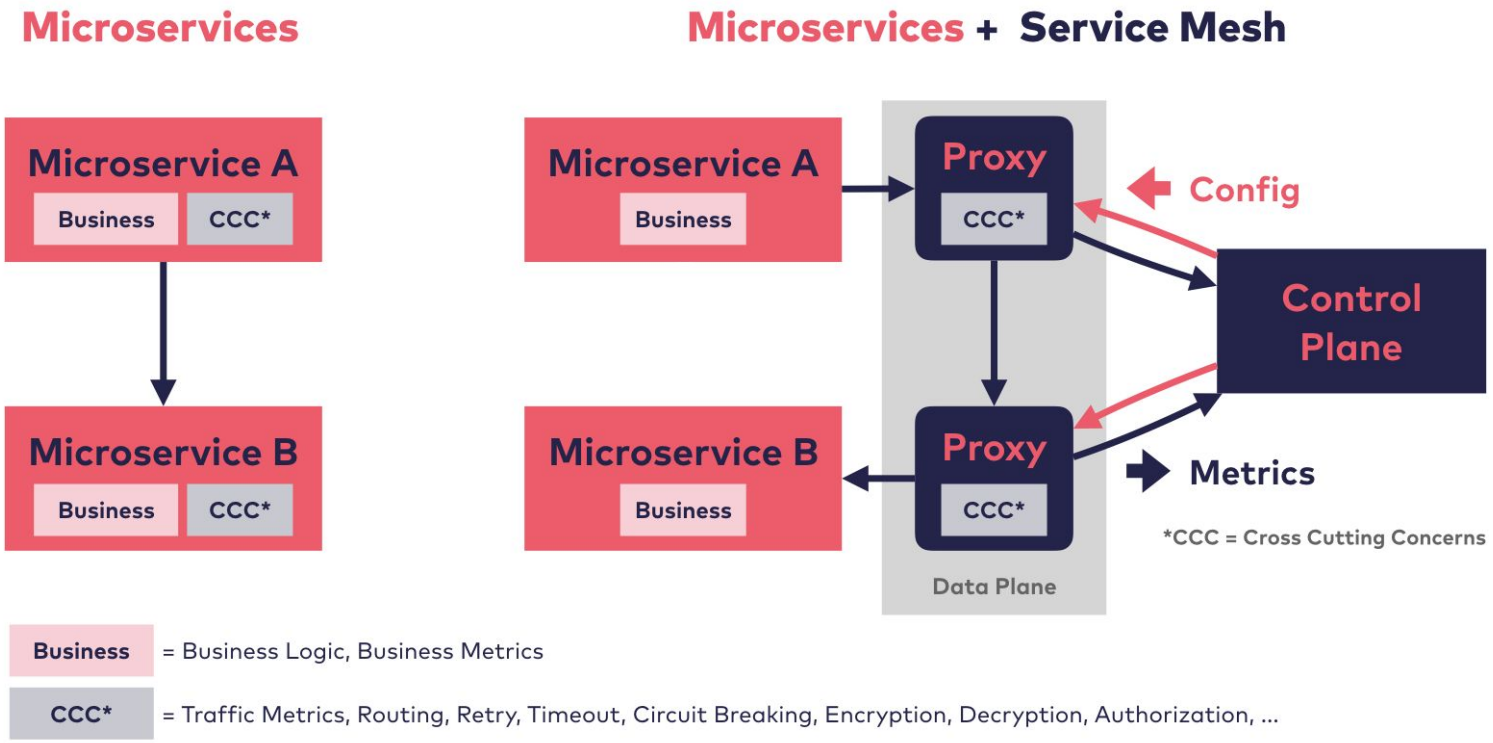
Envoy is a lightweight proxy written in C++ made for cloud-native concerns



Sidecar proxies injected within Pods and controlling inbound / outbound traffic

On obtient le fameux **“maillage de service”**

Un nouveau pattern d'architecture

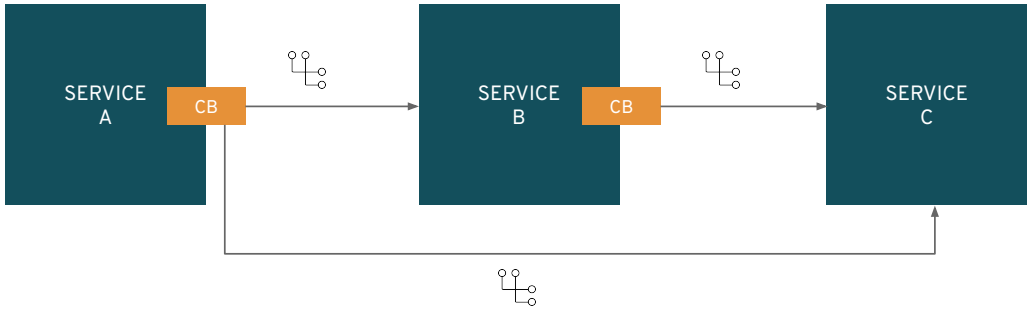


Red Hat

TOLERANCE AUX PANNES

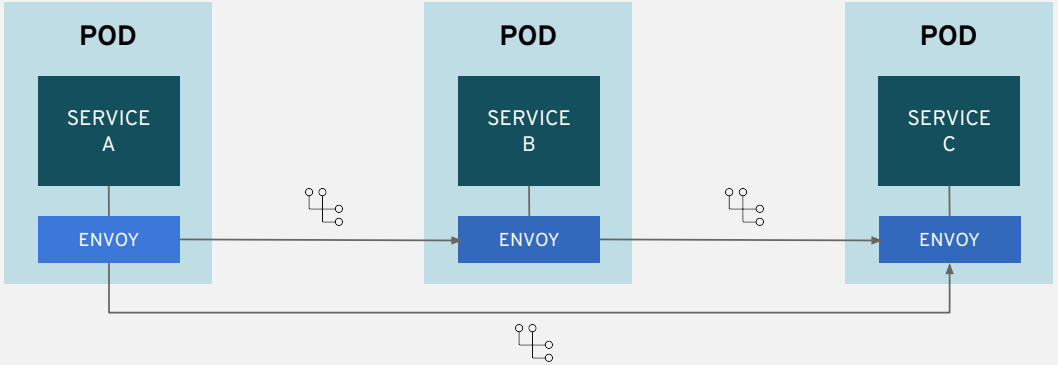
Circuit Breaker / Coupe circuit

sans service mesh



Intégré directement au code

avec service mesh



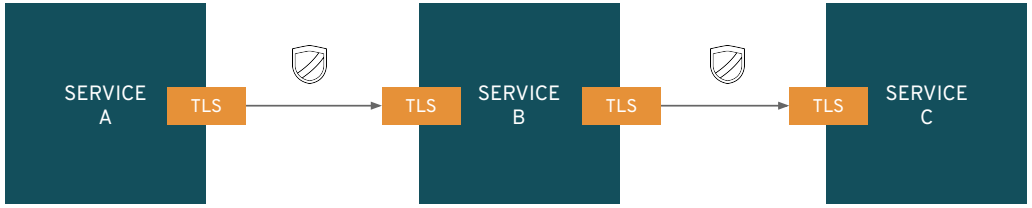
Transparent aux services

Autre ex.: Canary testing, routage dynamique, double run...

SECURITE

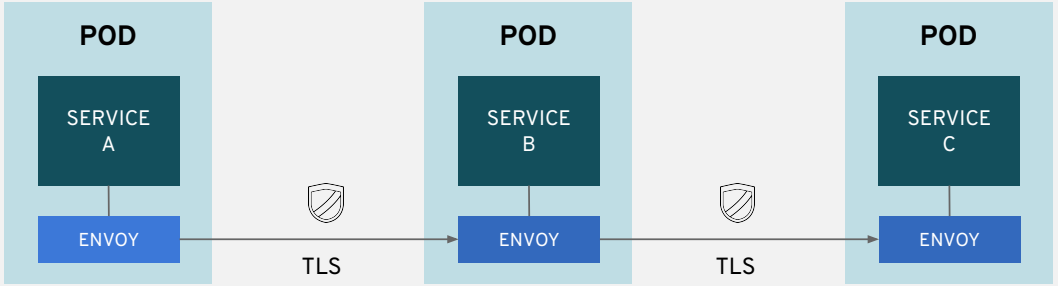
Chiffrement TLS

sans service mesh



Intégré directement au code

avec service mesh



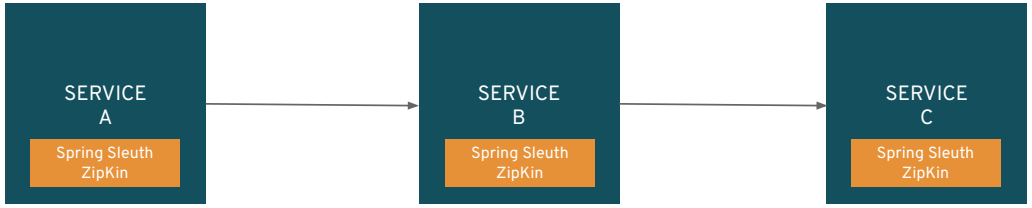
Transparent aux services

Red Hat

DISTRIBUTED TRACING

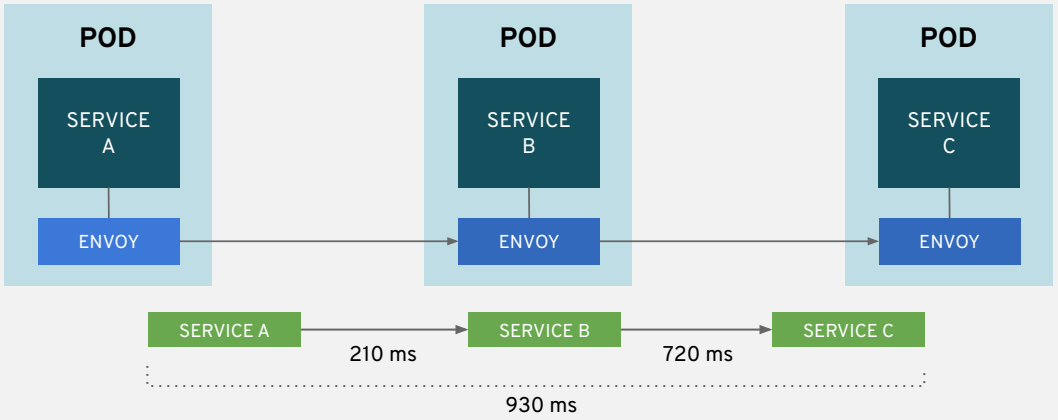
Temps de réponse

sans service mesh



Intégré directement au code

avec service mesh



Transparent aux services

Red Hat

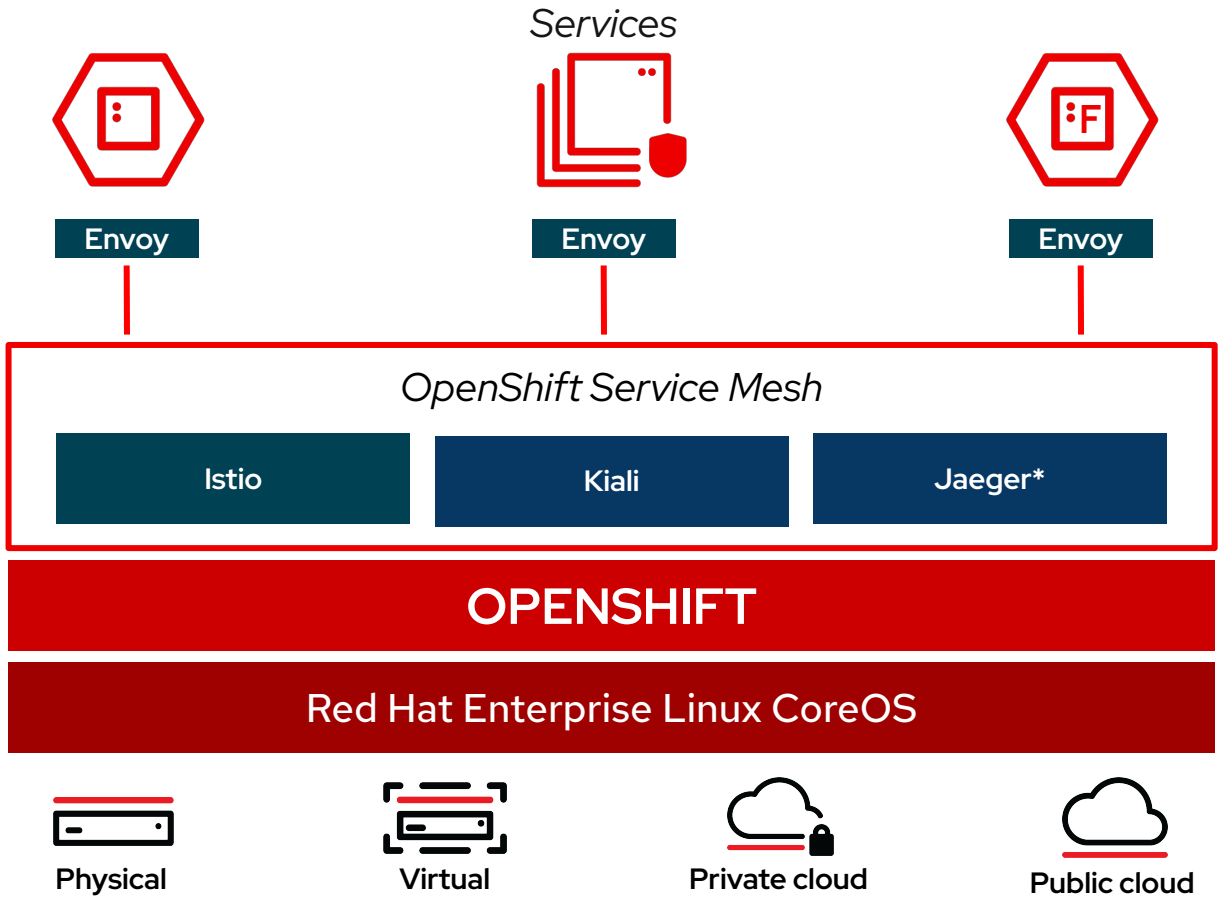
OpenShift Service Mesh

Red Hat

OpenShift Service Mesh

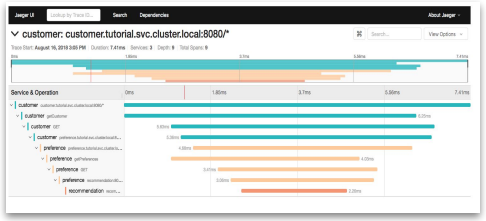
Connect, Secure, Control and Observe Services on OpenShift

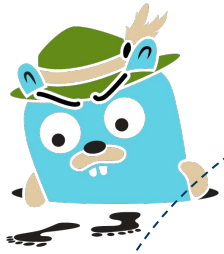
- A software infrastructure layer between Kubernetes and your services for managing communications.
- **Handles common "microservice" challenges, so that developers don't have to:**
  - Security
  - Monitoring & Observability
  - Application Resilience
  - Upgrades, Rollouts & A/B Testing
  - And more…



GESTION DU TRAFIC	Connecter les microservices au moment de l'exécution, de manière dynamique, utilisant les patterns de déploiement "Blue/Green" et "Canary Testing".
OBSERVABILITÉ	Voir ce qui se passe, avec un tracing, une surveillance et une journalisation automatiques riches de tous vos services.
SÉCURITÉ AUTHENTIFICATION ET CHIFFREMENT	Sécurisez automatiquement les services grâce à l'authentification, l'autorisation et le chiffrement gérés des communications entre les services.
GOUVERNANCE	Appliquer les politiques et s'assurer qu'elles sont appliquées et que les ressources sont équitablement réparties entre les consommateurs.

Une solution complète





Jaeger

Secure

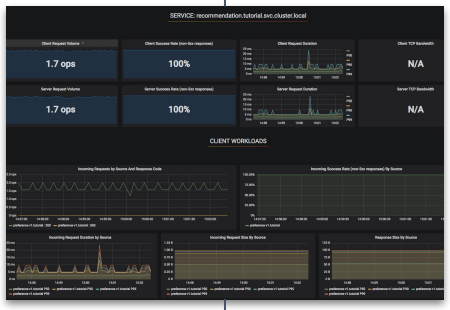
Istio

Observe

Connect

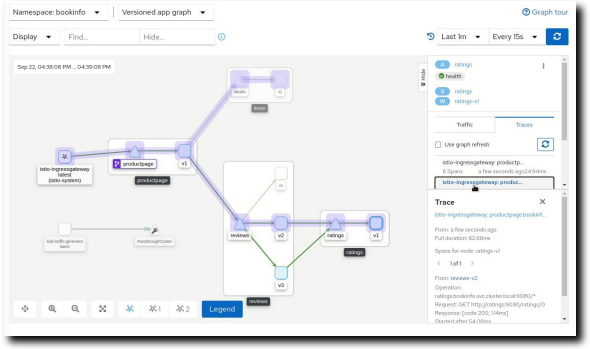
Control

Prometheus



Observe

Grafana

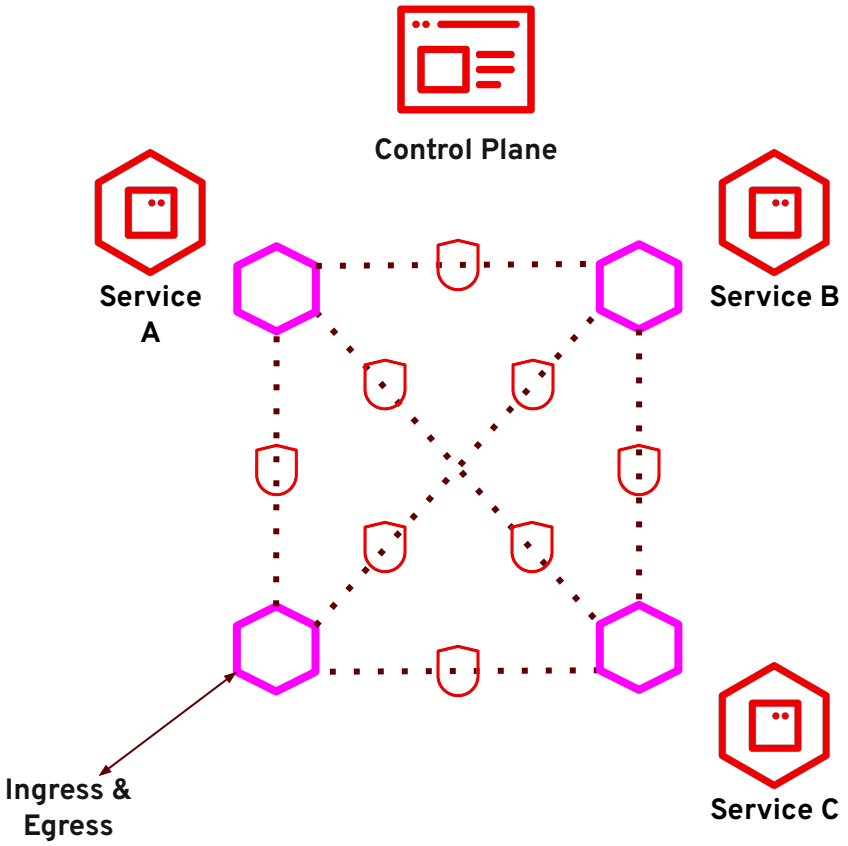


Kiali

Red Hat

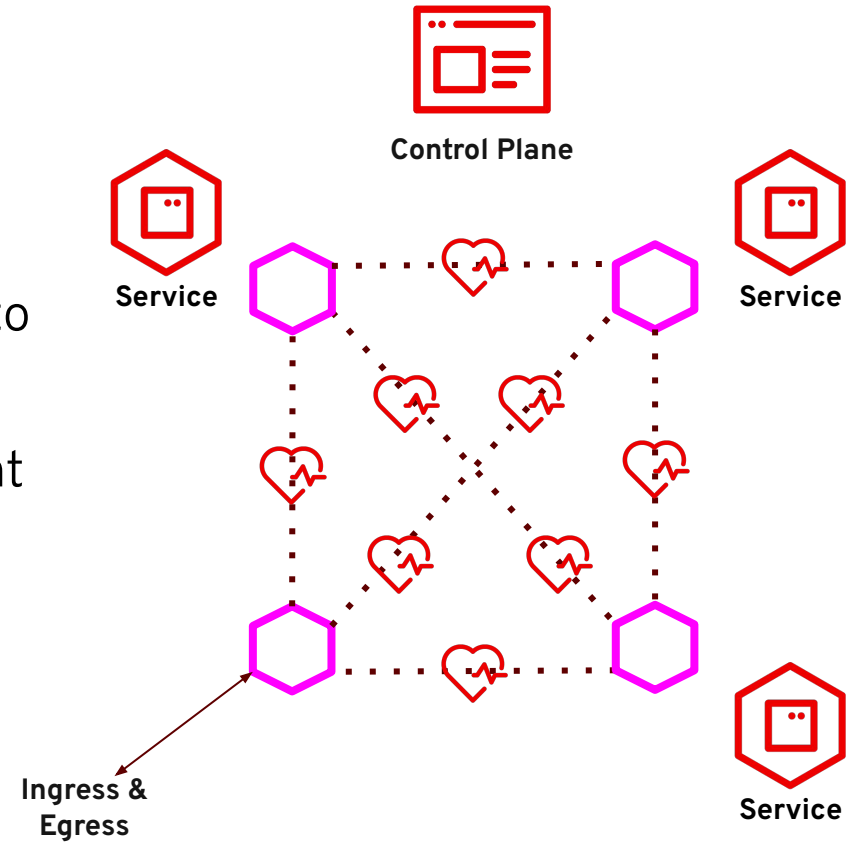
Sécuriser tous les services

- Appliquer et gérer les politiques de sécurité sur l'ensemble des services **sans modification du code source**.
  - Appliquez l'utilisation du **chiffrement mTLS** à tous les services.
  - **Authentifiez** les demandes à l'aide d'un JSON Web Token (JWT).
  - Définissez des **politiques d'autorisation** de service à service et d'utilisateur à service.
  - Sécurisez le control plan avec des politiques **RBAC**.



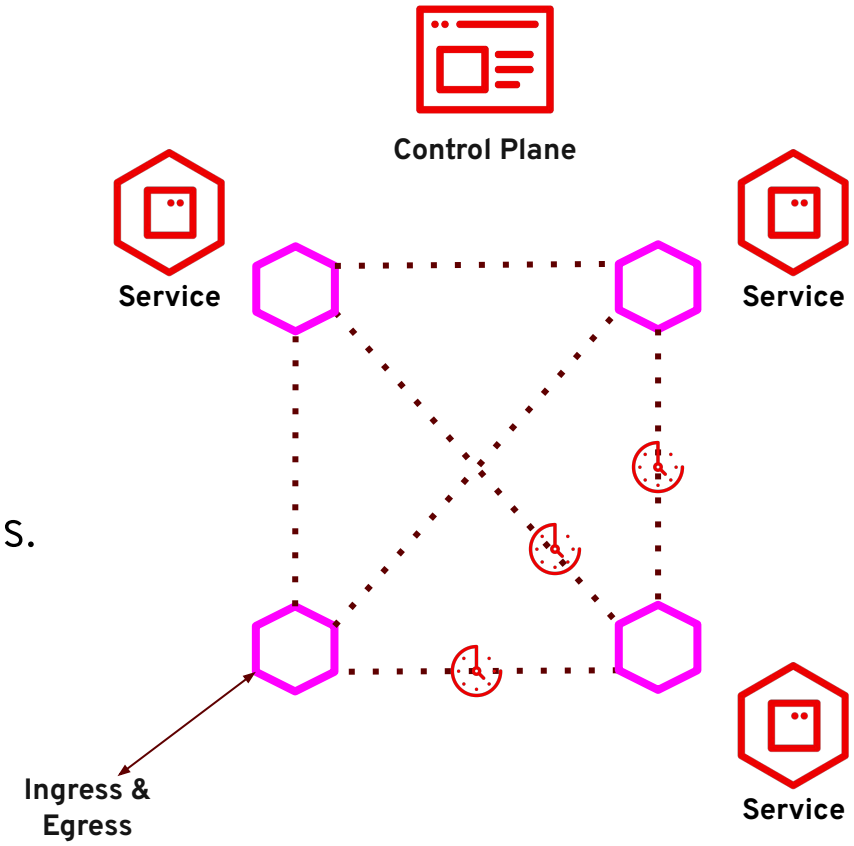
Construire des services plus résilients

- Recall, the fallacies of distributed computing.
  - “The network is always reliable”
- As communication is via proxies, we can add resilience to our system **without source code changes**.
- **Circuit breakers** and **timeouts** work together to prevent cascading failures between services.
- **Configurable retries** provide the ability to overcome short term failures of networks and services.
- **Fault injection** provides the ability to validate how services will perform when failures inevitably occur.
  - Enables chaos engineering.



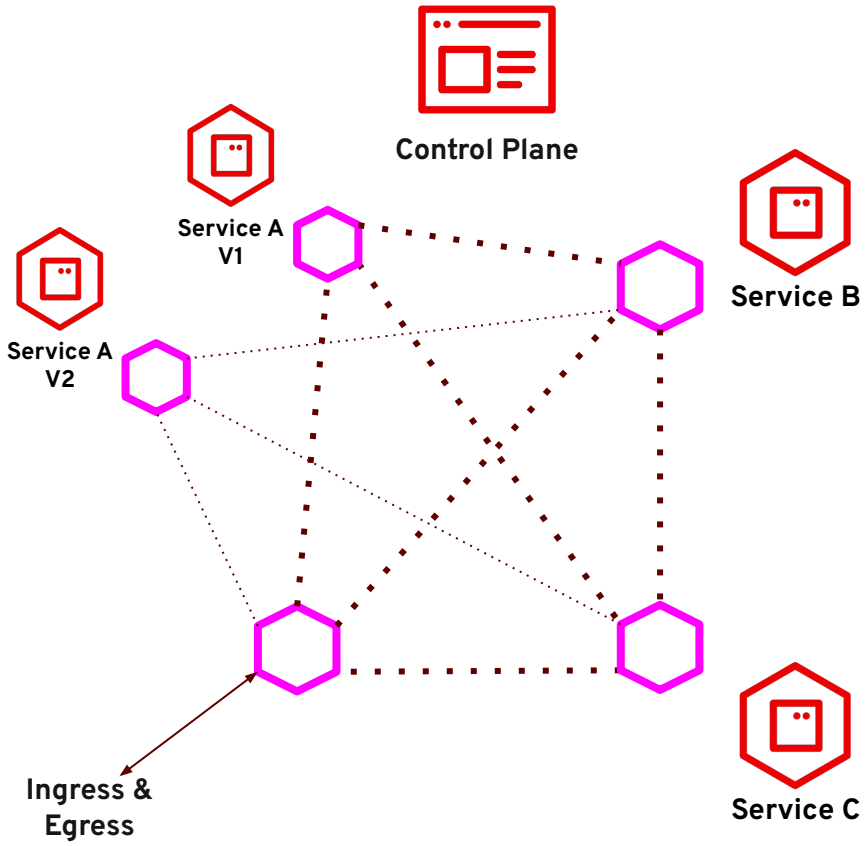
Tester la résilience aux pannes

- **Fault injection** provides the ability to validate how services will perform when failures inevitably occur.
  - Examples:
    - 50% of requests to a service will fail with error code 503.
    - 20% of services will be delayed by 5 seconds.
  - Enables chaos engineering.



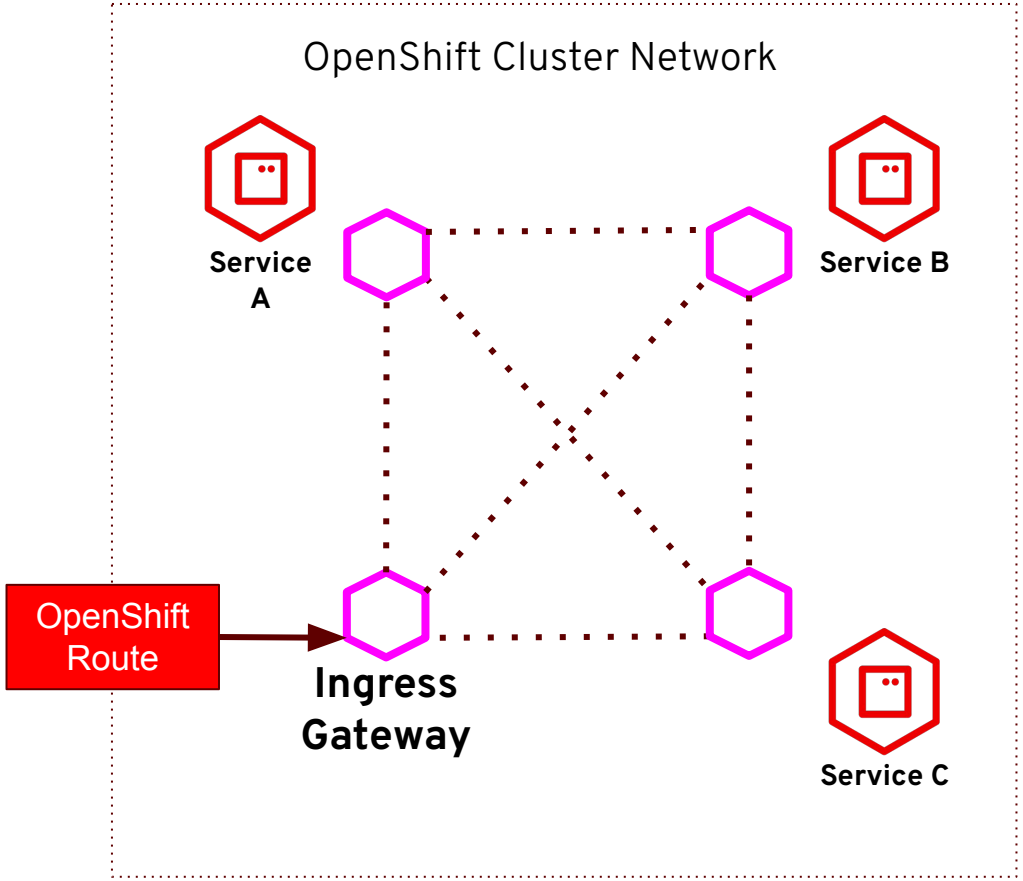
Utiliser de nouvelles stratégies de déploiement

- Control all communications allows for fine-grained traffic control between services **without source code changes or restarting services**.
- Create multiple “subsets” of a service (e.g. different versions) to enable:
  - **Canary Deployments** - apply a small amount of traffic to a new subset using weights.
  - **A/B Testing** - Apply a fraction of traffic to a different service using weights.
  - **Mirrored Launches** - duplicate live traffic loads across services to see how a new service handles the real world.



Intégrer le Service Mesh avec les Routes OpenShift

- In Service Mesh, an **Ingress Gateway** is used for accessing services within the Mesh.
- The **Ingress Gateway** is a standalone Envoy proxy that acts as an entry point into the mesh.
- In OpenShift, a **route*** acts as an entrypoint into the cluster, backed by **HAProxy**.
- OpenShift Service Mesh automatically creates and configures **routes** when **Ingress Gateways** are created.



* OpenShift also supports Kubernetes Ingress (which were inspired by routes) and Red Hat is an active contributor in the next generation of Ingress - Service APIs.

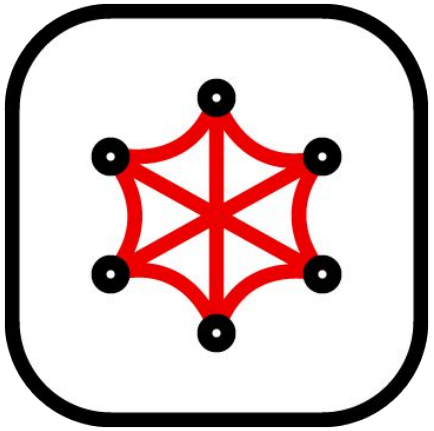
Red Hat
OpenShift

What's New in Red Hat OpenShift Service Mesh 3.0

Red Hat

A new beginning for OpenShift Service Mesh

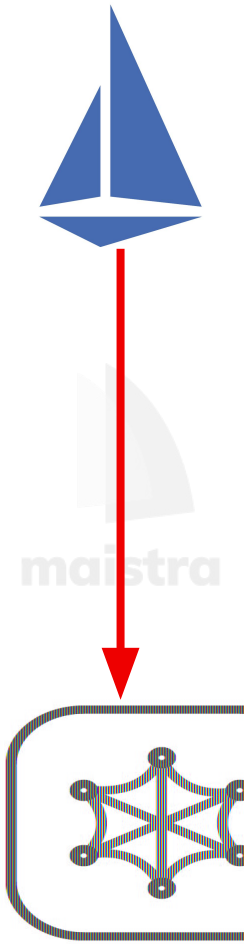
- Major releases are avoided to maintain stability
- OpenShift Service Mesh 3 represents **a foundation** that we will continue to build on in collaboration with customers
- Goals:
  - Support customers on the latest stable Istio features
  - Increase our collaboration with the Istio community
  - Unlock greater value with OpenShift Container Platform and OpenShift Platform Plus



Red Hat

OpenShift Service Mesh 3 in a Nutshell

- **A new operator**
  - New configuration
  - Simpler modular architecture: integrations over addons
  - Less “auto-config” magic: More declarative configuration
- **Istio**
  - No longer using Maistra.io
  - Support for ***most**** stable upstream features



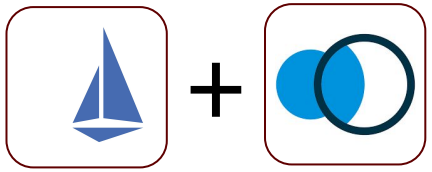
*We still validate & document community features on OpenShift to declare support, but any fixes/enhancements are done in the upstream Istio.io project.

Observability Integrations over components

- **The Service Mesh 3 operator ONLY manages Istio**
- Observability components are managed by standalone operators:
  - **Kiali** as a standalone operator
  - **Tracing** via OpenShift Distributed Tracing
  - **Metrics & Dashboards** from OpenShift user-workload monitoring
- While this means "getting started" takes longer, this provides greater control and flexibility over how these integrations are configured and managed.
- Sample configurations and "cook books" will be available for quickly configuring service mesh + observability for development purposes.

OpenShift Service Mesh 2

OpenShift Service Mesh 3



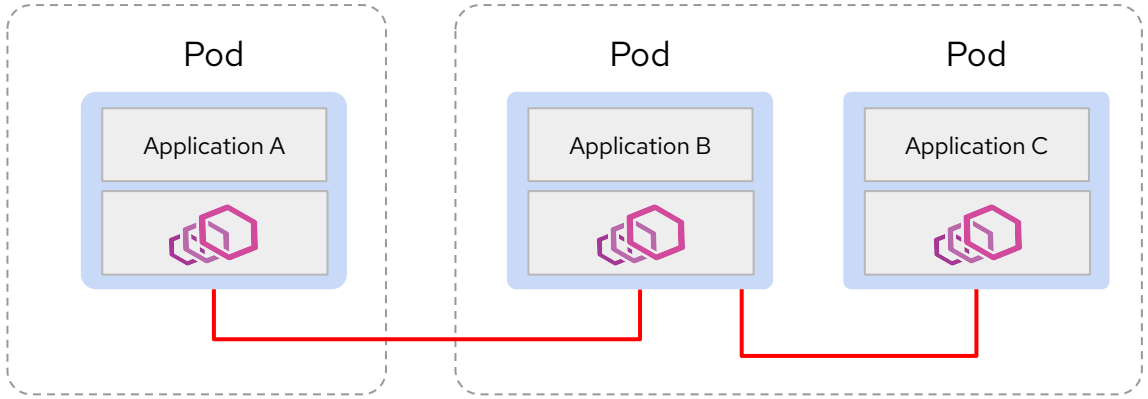
\+ Integrations...

Red Hat

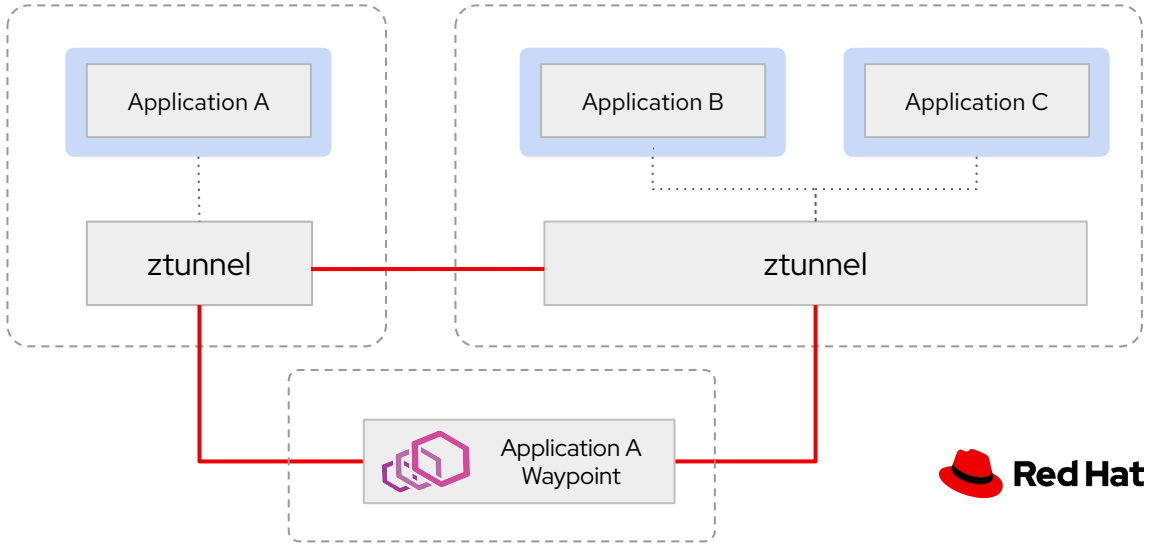
OpenShift Service Mesh 3 Features

- Kubernetes Gateway API is supported for use with Istio
  - Community CRD installation is required; will be part of OCP 4.19.
- IPv4/IPv6 dual-stack is technology preview
- **Istio's "sidecar-less" ambient mode is now GA**
  - High performance **ZTunnel** proxy supports lightweight L4 encryption
  - **Waypoint** proxy provides L7 mesh features.

Side car mode (traditional)



Ambient mode (sidecar-less)



Red Hat

Istio Ambient Mesh

ZTunnel proxy

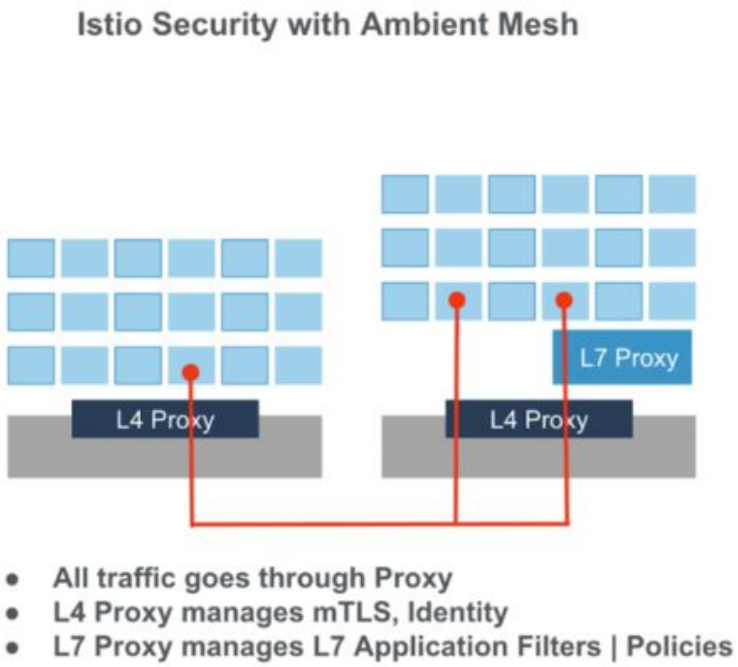
A per-node proxy that manages secure, transparent Transmission Control Protocol (TCP) connections for all workloads on the node. It operates at Layer 4 (L4), offloading mutual Transport Layer Security (mTLS) and L4 policy enforcement from application pods.

Waypoint proxy

An optional proxy that runs per service account or namespace to provide advanced Layer 7 (L7) features such as traffic management, policy enforcement, and observability. You can apply L7 features selectively to avoid the overhead of sidecars for every service.

Istio CNI plugin

Redirects traffic to the Ztunnel proxy on each node, enabling transparent interception without requiring modifications to application pods.



https://docs.redhat.com/en/documentation/red_hat_openshift_service_mesh/3.1/html/installing/ossm-istio-ambient-mode

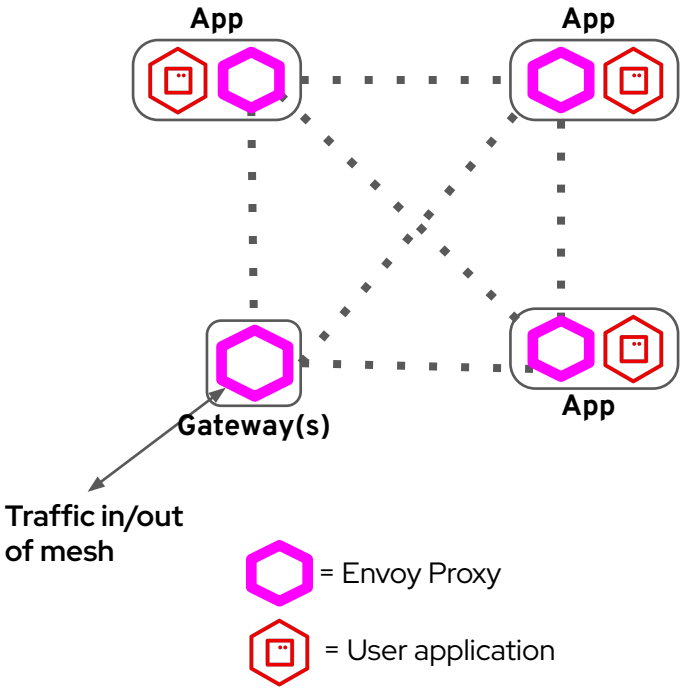
Kubernetes Gateway API Support

- Kubernetes Gateway API is the future of both Kubernetes networking and Istio traffic management (will be default in future)
- Red Hat OpenShift Service Mesh 3 includes support for using Kubernetes Gateway API to configure Istio
- Requires the installation of Kuberenetes Gateway API CRDs, which are not supported on OpenShift as of OCP 4.18.
  - GA support is targeted for OCP 4.19

Red Hat

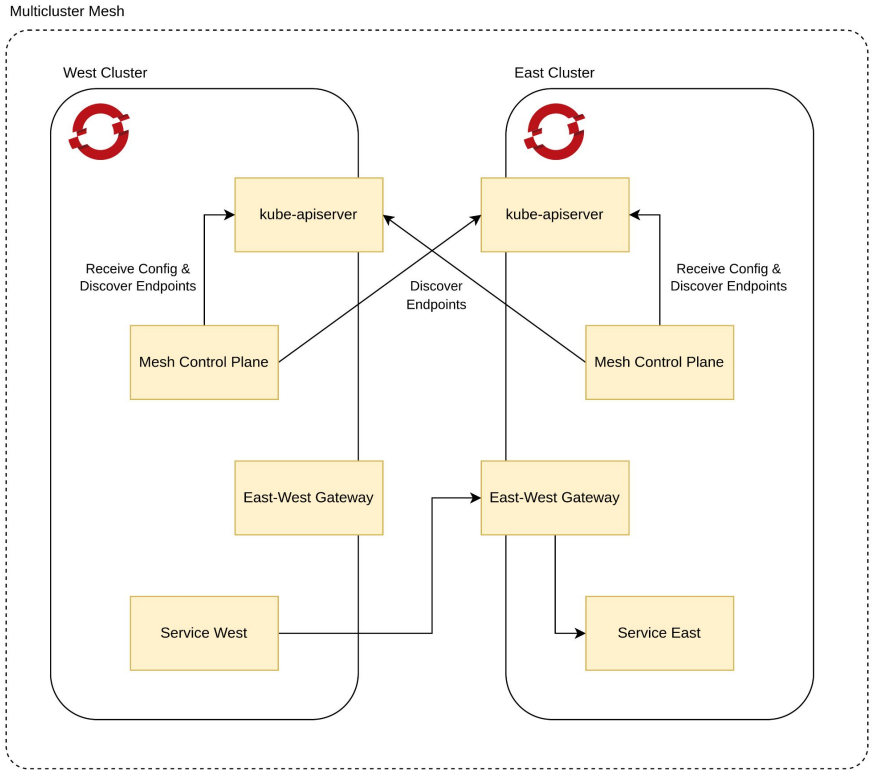
Gateways in OpenShift Service Mesh 3

- In OpenShift Service Mesh 3, **gateways are no longer managed by the operator**.
- This provides greater simplicity, flexibility and encourages the best practice of gateways managed with applications.
- Gateways can be created with:
  - Gateway Injection using a Kubernetes **Deployment** and exposed via:
    - An OpenShift **Route** resource
    - A Kubernetes **Service** of type **LoadBalancer**
  - Kubernetes **Gateway API resources**



Red Hat

Support for Istio's multi-cluster toplogies



- Istio's multi-cluster topologies allow a service mesh to be stretched across multiple OpenShift clusters.
- Istio's multi-cluster topologies include:
  - Multi-primary control planes
  - Primary-remote control planes
  - External control plane(s)

Red Hat

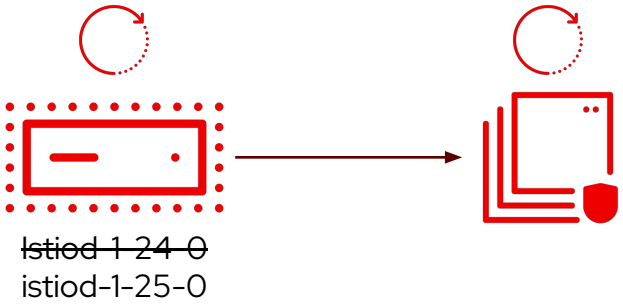
In Place vs Revision Based Control Plane Upgrades

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.

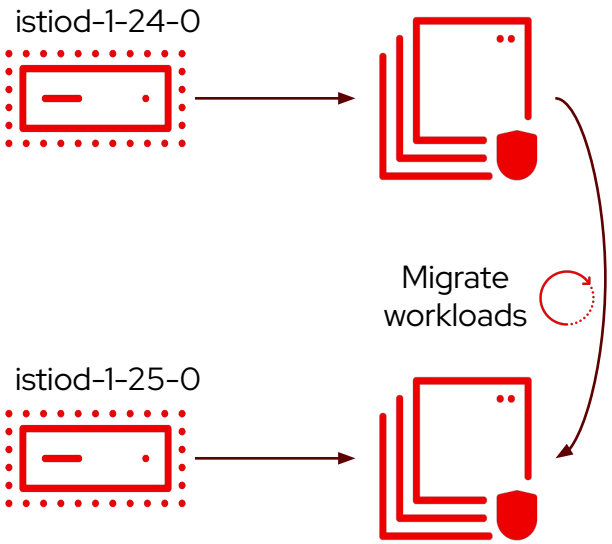


Revision Based Upgrades

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

Migrating to Red Hat OpenShift Service Mesh 3

Red Hat

Preparing to migrate to Service Mesh 3

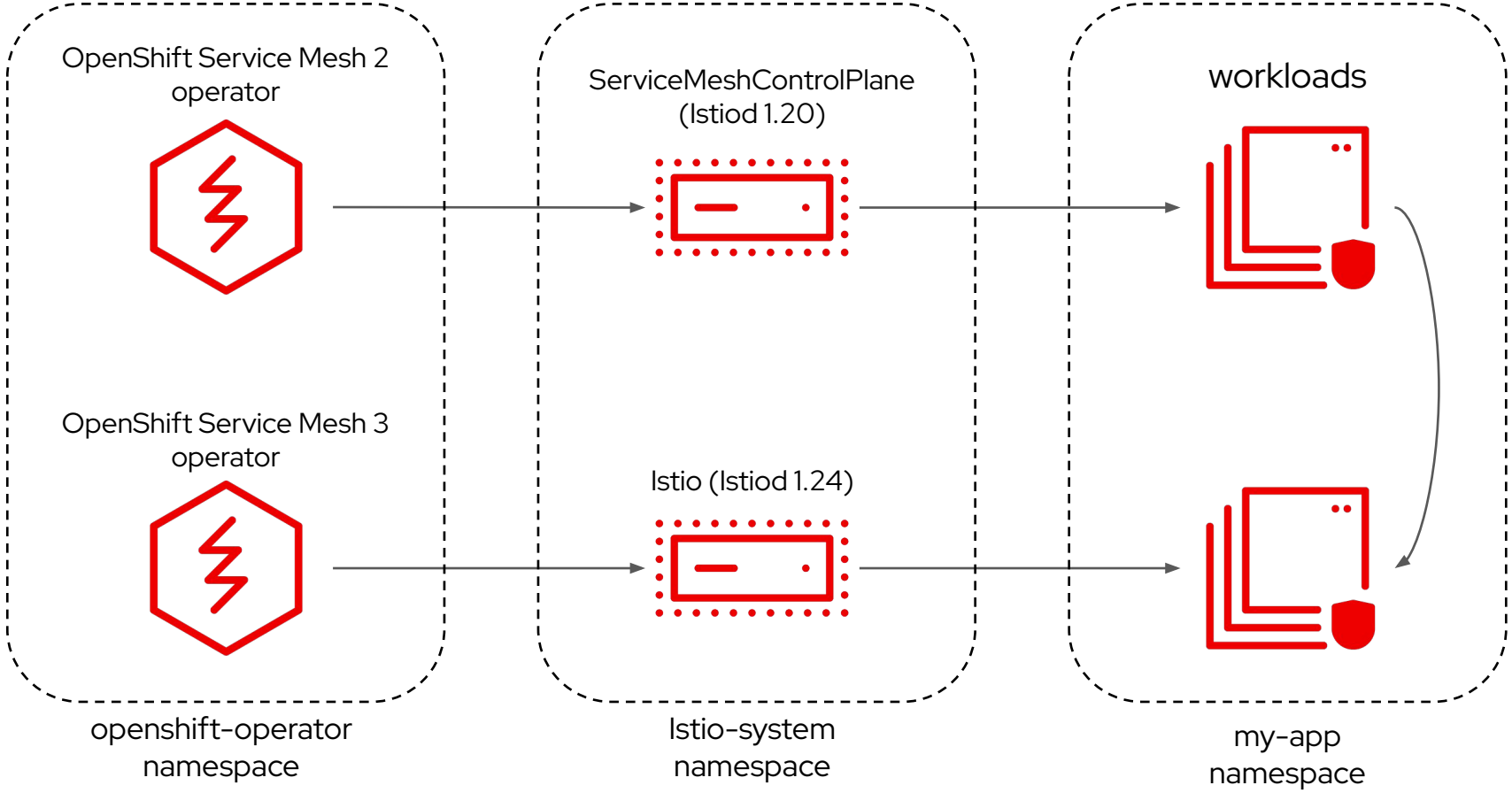
- There are several things that customers can do **TODAY** to prepare for the upgrade to Service Mesh 3:
  - Upgrade to the latest available version of OpenShift Service Mesh 2.6
  - Move to **Gateway Injection** for creating and managing all Gateways
  - **Disable IoR** and explicitly manage Gateways with Routes resources
  - Use **OpenShift user-defined project monitoring** for metrics
  - Migrate to **OpenTelemetry** and **Tempo** for distributed tracing.
  - Configure an **external Kiali** resource to manage Kiali
  - Disable OpenShift Service Mesh 2.6's network policy management
- Detailed procedures for all of the above are part of the migration guide.

Migrating workloads to OpenShift Service Mesh 3

- The documented migration procedures aim to move workloads to Red Hat OpenShift Service Mesh 3 while maintaining application connectivity.
- Service Mesh 2 and 3 control planes are deployed in parallel in the same namespace (e.g. istio-system)
- Workloads labels are then configured to migrate to the Service Mesh 3 control plane during the next rollout.

Migrating workloads to OpenShift Service Mesh 3

Deploy OpenShift Service Mesh 3 operator and control plane **in existing namespaces**



Migrating workloads:

1. Add labels:
   a. Ignore 2.6 control plane
   b. Connect to 3.0 control plane
2. Restart workloads to move to new control plane

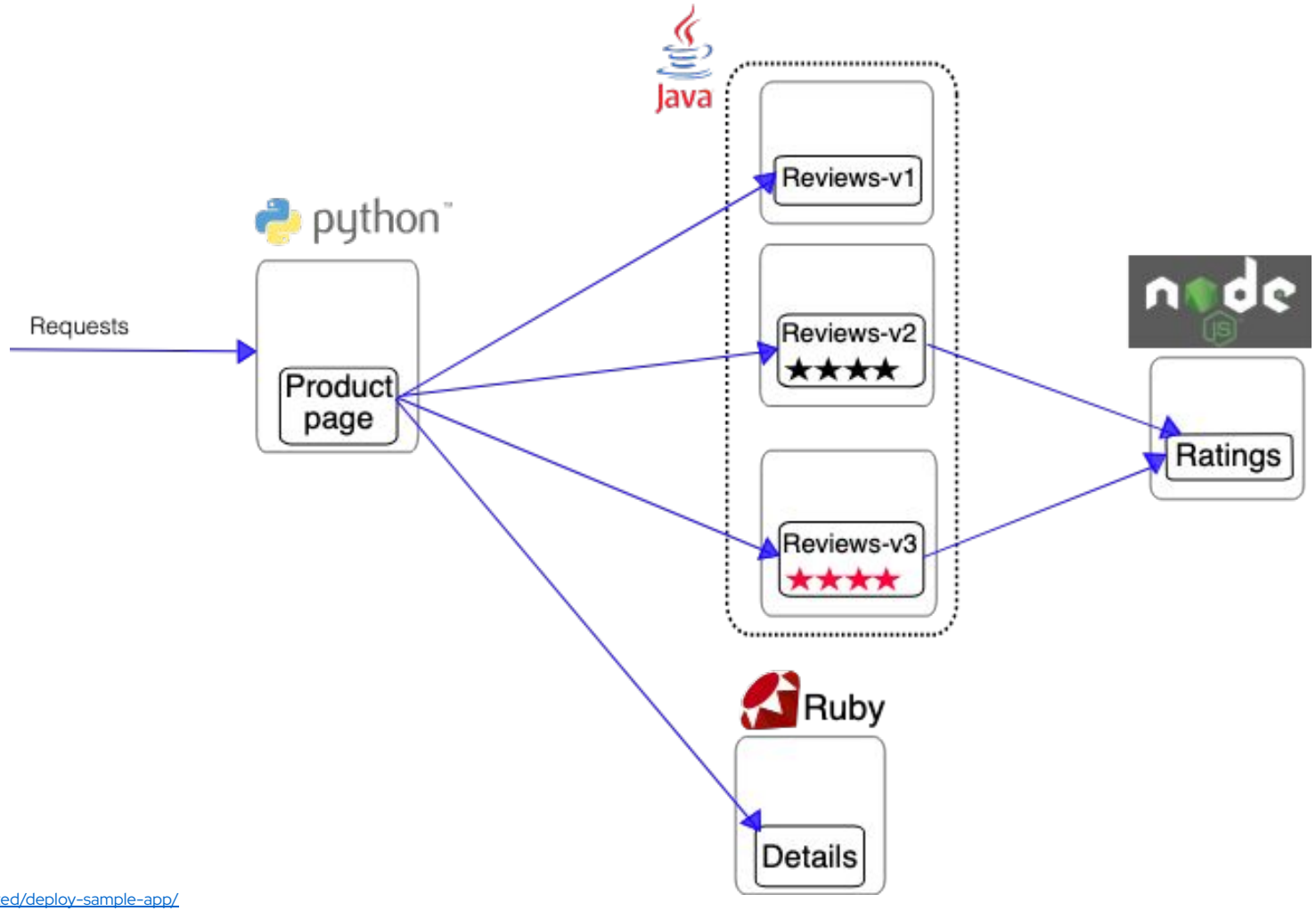
Red Hat

Red Hat
OpenShift

Démonstration

Red Hat

Cas d'usage : **bookinfo**



https://istio.io/latest/docs/ambient/getting-started/deploy-sample-app/

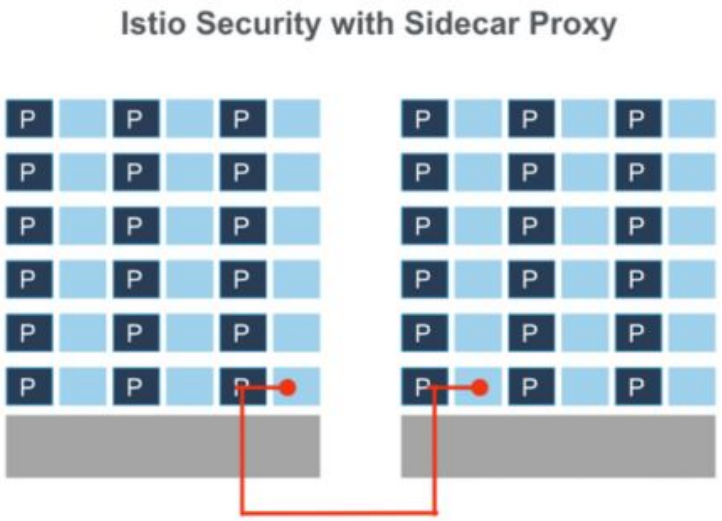
https://docs.redhat.com/en/documentation/red_hat_openshift_service_mesh/3.1/html/observability/kiali-operator-provided-by-red-hat#ossm-config-openshift-monitoring-kiali_ossm-kiali

Red Hat

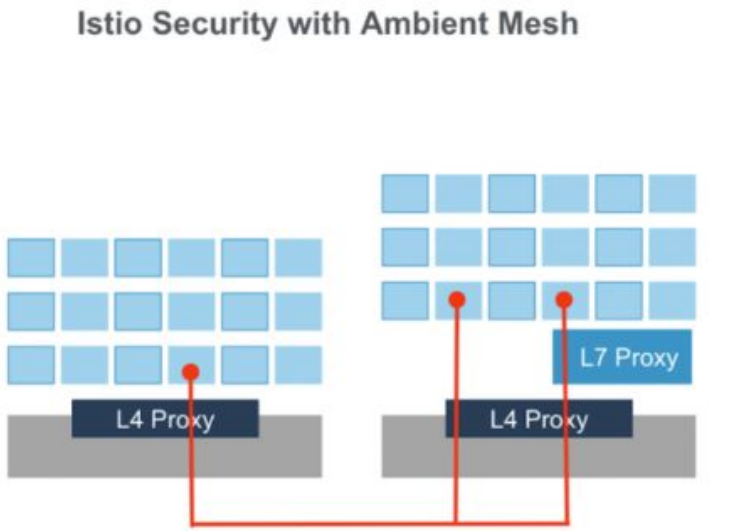
Merci

Red Hat

Ambient Mesh



- All traffic goes through Proxy
- Proxy manages mTLS, Identity
- Proxy manages L7 Application Filters | Policies



- All traffic goes through Proxy
- L4 Proxy manages mTLS, Identity
- L7 Proxy manages L7 Application Filters | Policies

https://www.solo.io/blog/understanding-istio-ambient-ztunnel-and-secure-overlay/
https://www.oreilly.com/library/view/istio-ambient-explained/9781098142698/ch01.html

```yaml
apiVersion: sailoperator.io/v1
kind: Istio
metadata:
  name: default
spec:
  namespace: istio-system
  profile: ambient
  values:
    pilot:
      trustedZtunnelNamespace: ztunnel
```

```yaml
apiVersion: sailoperator.io/v1alpha1
kind: ZTunnel
metadata:
  name: default
spec:
  namespace: ztunnel
  profile: ambient
```

```yaml
apiVersion: sailoperator.io/v1
kind: IstioCNI
metadata:
  name: default
spec:
  namespace: istio-cni
  profile: ambient
```

- Il faut définir obligatoirement la version Istio dans les CRD Istio, IstioCNI et ZTunnel
- Par défaut, le control plan détecte tous les workloads dans tous les namespaces du cluster-> Discovery Selectors

Installing Istio ambient mode
bookinfo App

Red Hat

Istio Version *

v1.26.4

Defines the version of Istio to install.
Must be one of: v1.26-latest, v1.26.4, v1.26.3, v1.26.2, v1.24-latest, v1.24.6, v1.24.5, v1.24.4, v1.24.3.

Name	Kind	Status
ICNI default	IstioCNI	State: Healthy
IR default	IstioRevision	State: Healthy
I default	Istio	State: Healthy
ZT default	ZTunnel	State: Healthy

HR ztunnel

Details | Resources | Release no

DaemonSets

DS ztunnel

ServiceAccounts

SA ztunnel

ClusterRoles

CR ztunnel

ClusterRoleBindings

CRB ztunnel

D istiod

Details | Resources | O

Pods

P istiod-69b5fc4898-6rwsq

Services

S istiod
Service port: grpc-xds → Pod port: 15010
Service port: https-dns → Pod port: 15012
Service port: https-webhook → Pod port: 15017
Service port: http-monitoring → Pod port: 15014

Routes

No Routes found for this resource.

HR istio-cni

Details | Resources | Release notes

DaemonSets

DS istio-cni-node

ConfigMaps

CM istio-cni-config

ServiceAccounts

SA istio-cni

Network Attachment Definitions

NAD istio-cni

ClusterRoles

CR istio-cni
CR istio-cni-repair-role

ClusterRoleBindings

CRB istio-cni
CRB istio-cni-repair-rolebinding

```
oc apply -f https://github.com/kubernetes-sigs/gateway-api/releases/download/v1.2.0/standard-install.yaml
```

CRD Gateway	gateway.networking.k8s.io	v1
CRD GatewayClass	gateway.networking.k8s.io	v1
CRD GRPCRoute	gateway.networking.k8s.io	v1
CRD HTTPRoute	gateway.networking.k8s.io	v1

```
https://raw.githubusercontent.com/openshift-service-mesh/istio/refs/heads/master/samples/bookinfo/gateway-api/bookinfo-gateway.yaml
```

Gateways gateway.networking.k8s.io/v1

Create Gateway

Name	Namespace
bookinfo-gateway	bookinfo

HTTPRoutes gateway.networking.k8s.io/v1

Create HTTPRoute

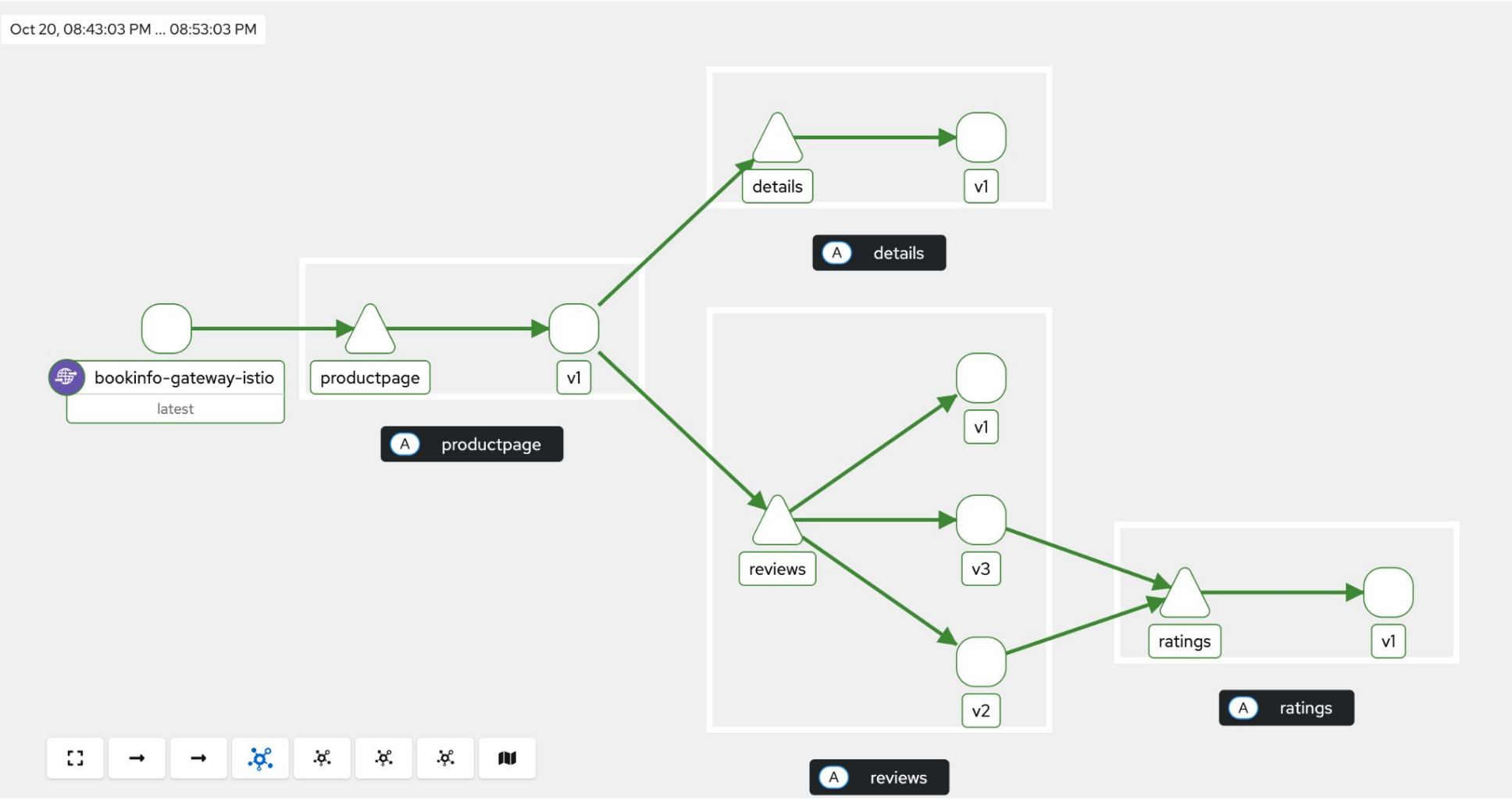
Name	Namespace
bookinfo	bookinfo

Name	Labels	Pod selector	Location
bookinfo-gateway-istio	gateway.istio.io/managed=istio.io-gateway-controller gateway.networking.k8s.io/gateway-name=bookinfo-gateway istio.io/dataplane-mode=none	gateway.networking.k8s.io/gateway-name=bookinfo-gateway	Pending

Une annotation existe pour modifier le service en ClusterIP

RT bookinfo-gateway-istio	Accepted	https://bookinfo-gateway-istio-bookinfo.apps.cluster-dkstz.dynamic.redhatworkshops.io	bookinfo-gateway-istio

oc expose -n bookinfo svc/bookinfo-gateway-istio --port http


Oct 20, 08:43:03 PM ... 08:53:03 PM
bookinfo-gateway-istio
latest
productpage
v1
A productpage
details
v1
A details
reviews
v1
v3
v2
A reviews
ratings
v1
A ratings

Red Hat