



# Service Mesh: End user authentication with Keycloak (RH Single Sign-On)

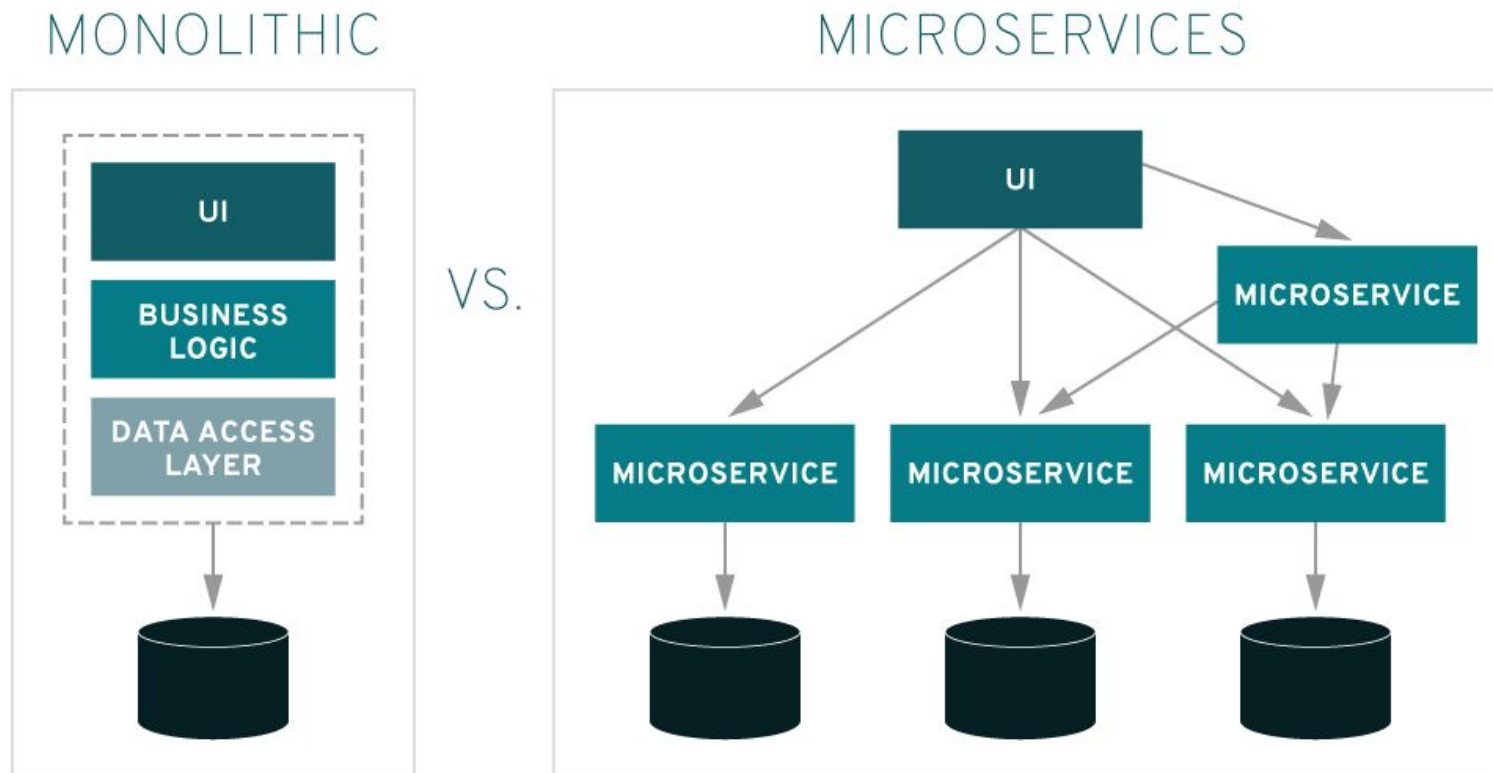
# What we'll discuss today

- Recap: What is a Service Mesh?
- Istio and OpenShift Service Mesh
- What's new in Service Mesh release 2.0?
- Service Mesh components
  - Traffic Management
  - Observability (Jaeger, Kiali)
  - Security
- Keycloak
- End user authentication with keycloak

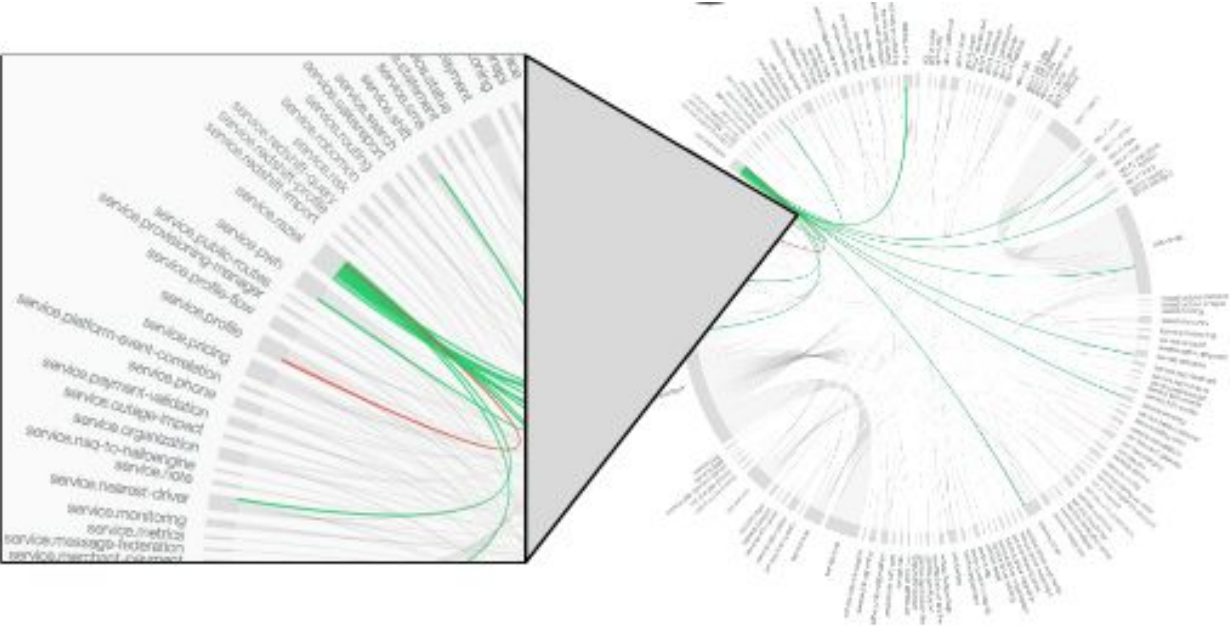
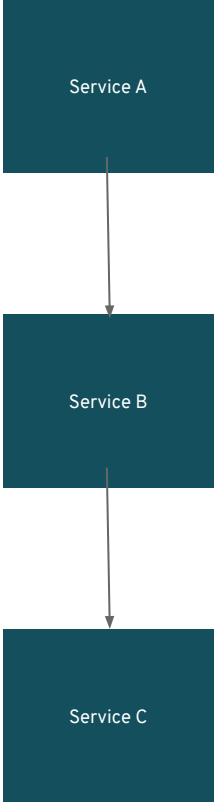
# What is a Service Mesh?

# Microservices Approach

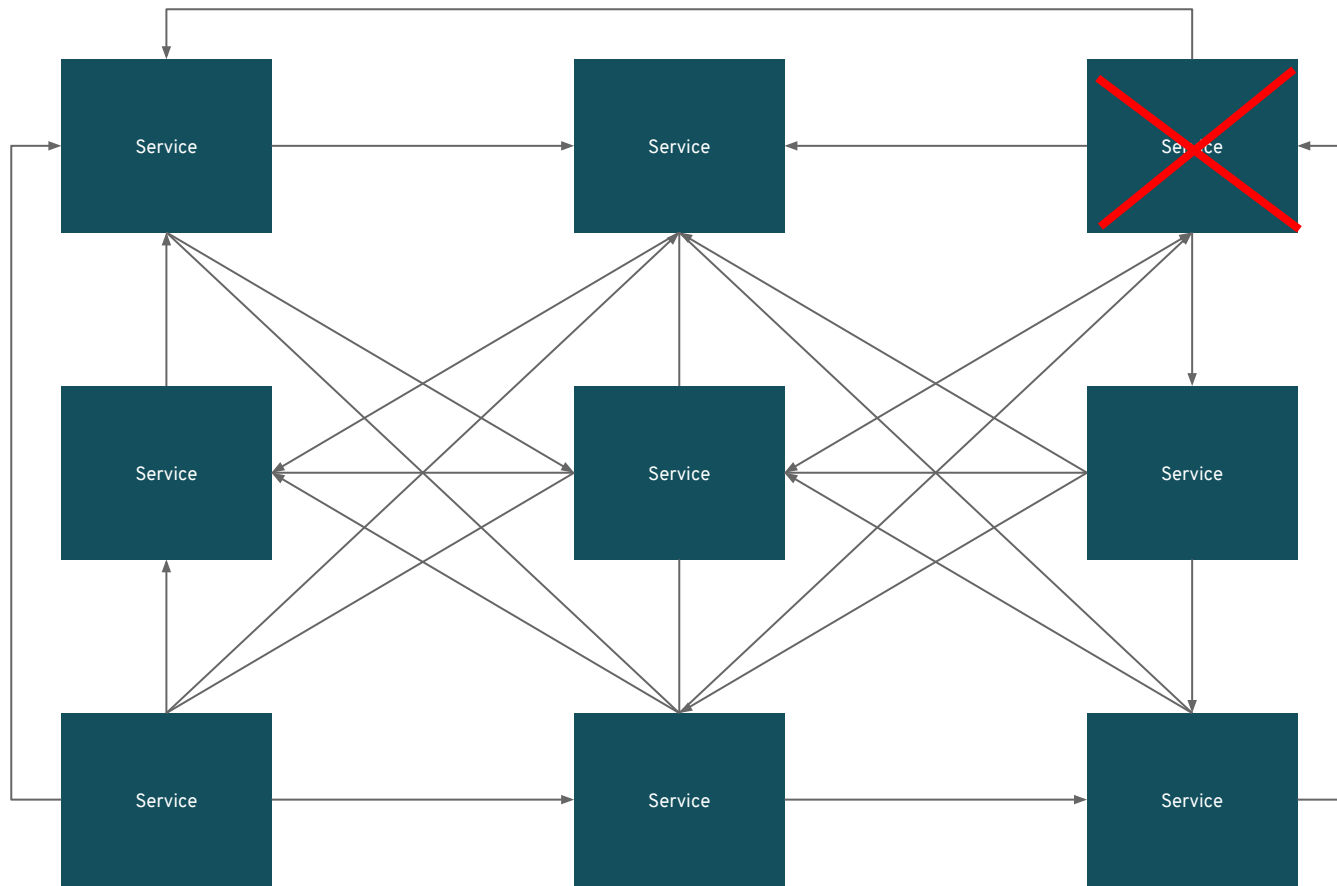
Microservices are an architectural approach to building application that consists of distributed and loosely coupled services, in a way that one team's changes won't break the entire app.



# Expectation vs Reality



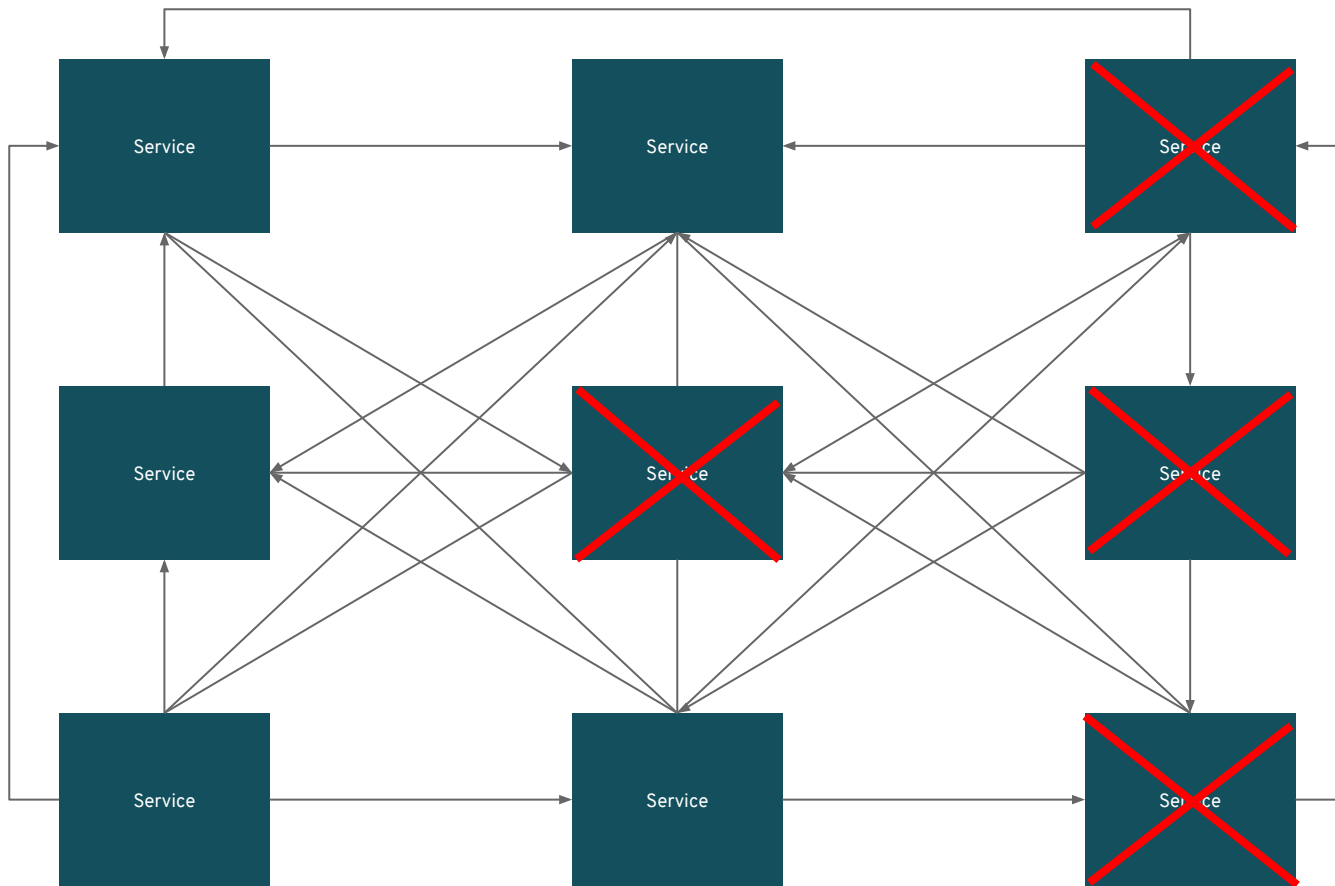
# Services Failure



## Microservices Fallacies

- Network is reliable
- Latency is zero
- Bandwidth is infinite
- Network is secure
- Topology does not change
- There is one administrator
- Transport cost is zero
- Network is homogeneous

# Cascading Failure

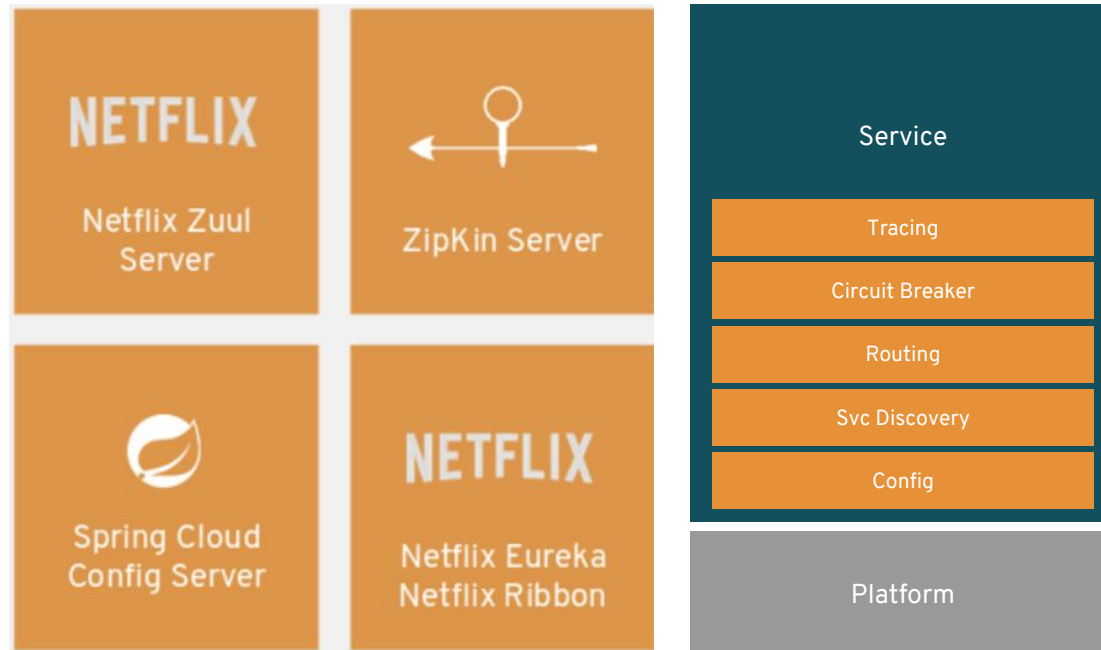


## Microservices are hard:

- Unpredictable failures
- End-to-end application correctness
- System degradation
- Topology changes
- Elastic/ephemeral/transient resources
- Distributed logs
- The fallacies of distributed computing

# Supporting Services for Distributed Applications

To address the challenges a set of supporting services must be added to your code



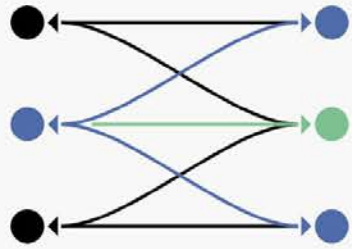
- Configuration
- Service Discovery
- Dynamic Routing
- Resilience
- Observability



# Istio

# Istio Project

Istio makes it easy to create a network of deployed services with load balancing, service-to-service authentication, monitoring, and more, without any changes in the service code.



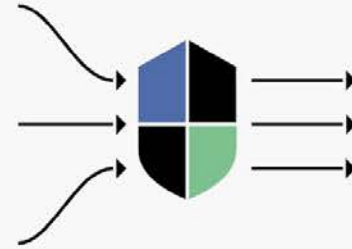
Connect

Intelligently control the flow of traffic and API calls between services, conduct a range of tests, and upgrade gradually with red/black deployments.



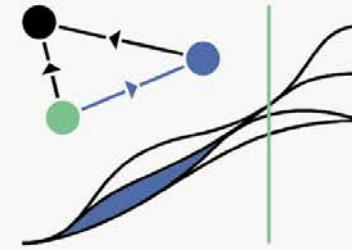
Secure

Automatically secure your services through managed authentication, authorization, and encryption of communication between services.



Control

Apply policies and ensure that they're enforced, and that resources are fairly distributed among consumers.



Observe

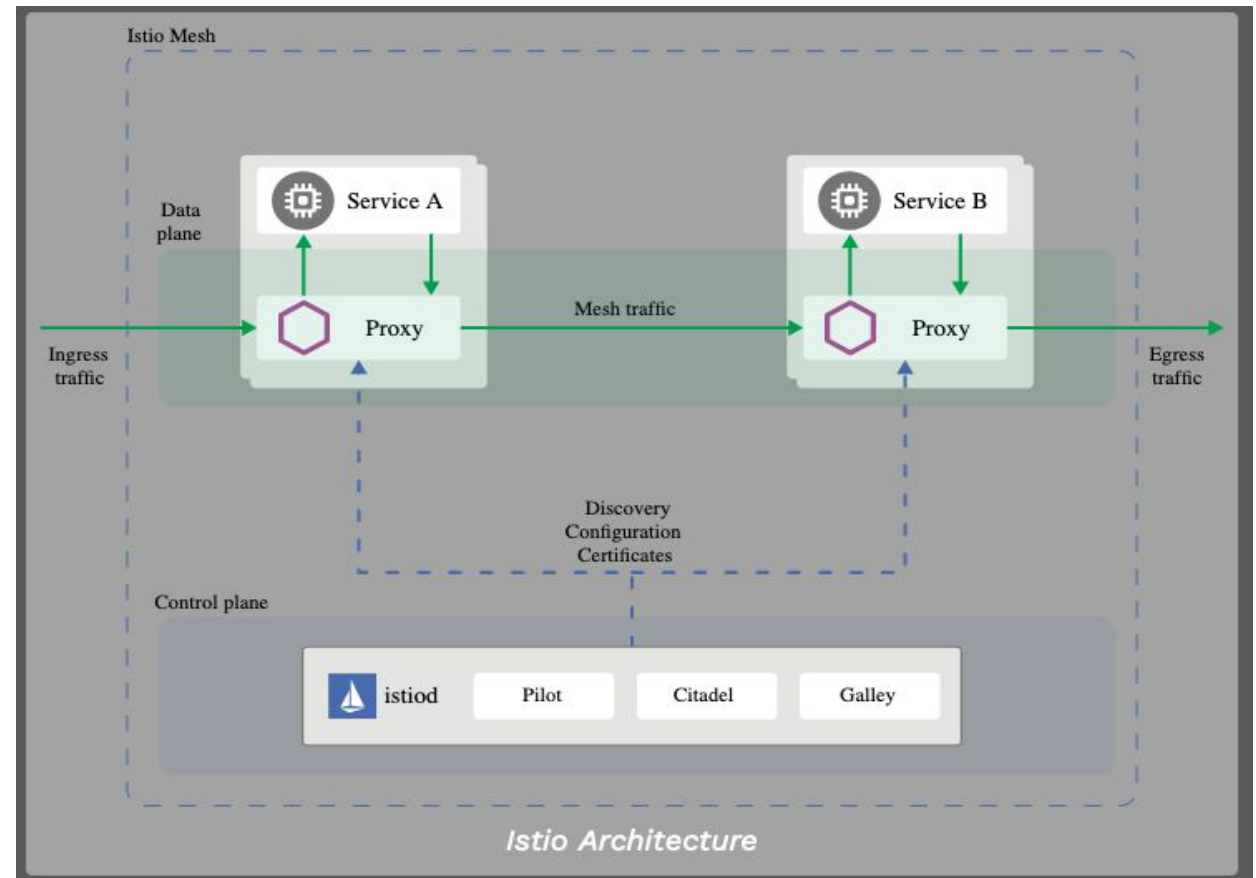
See what's happening with rich automatic tracing, monitoring, and logging of all your services.

# Service Mesh

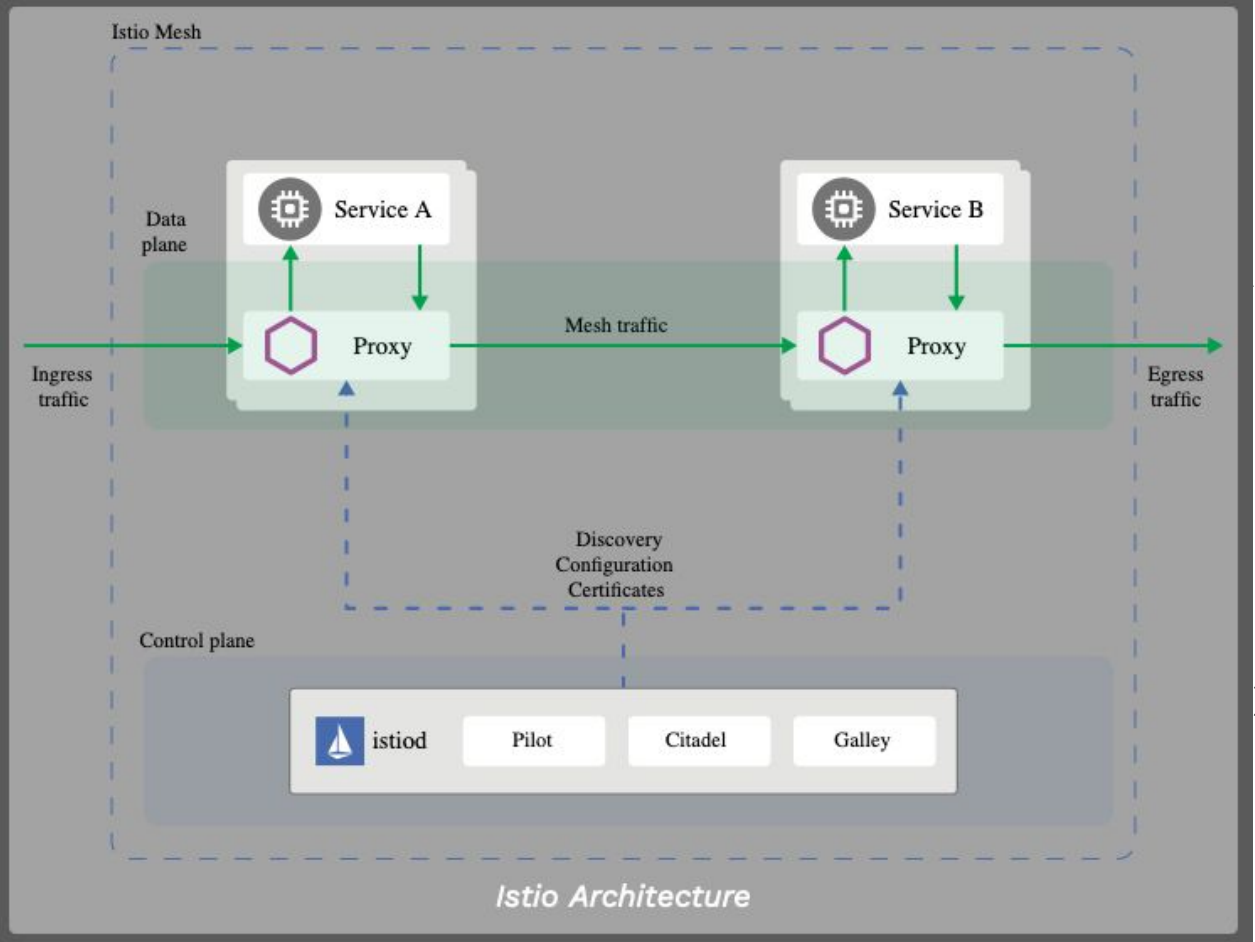
A Service Mesh is a logical space comprising one or more namespaces in which network rules are expressed declaratively to a control plane and enforced by a sidecar proxy (Envoy).

## Features:

- Load balancing
- Routing rules
- Service monitoring and logging
- Secure cross-service communications



# Architecture



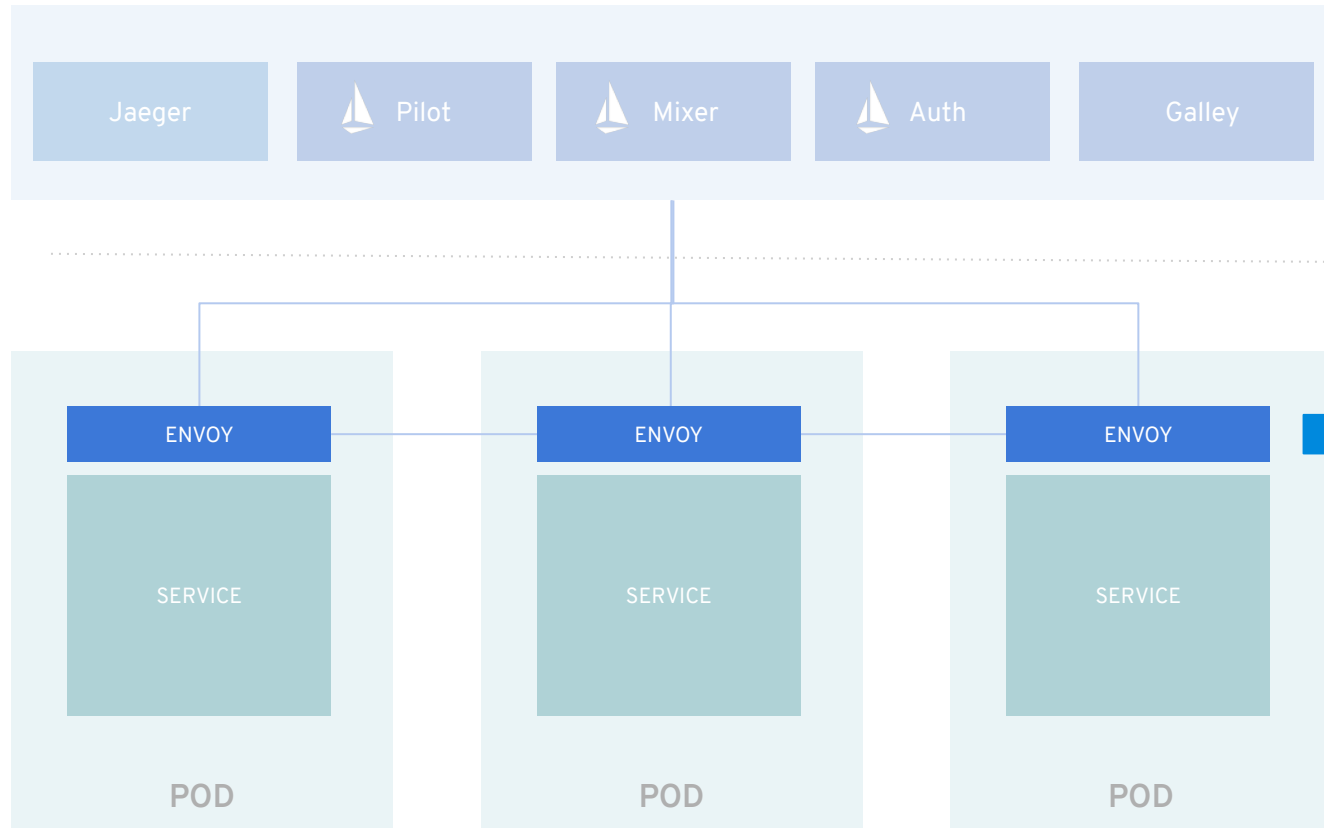
Applies security, route rules, policies and reports traffic telemetry at the pod level

Data Plane

The control plane manages and configures the proxies to route traffic and configures

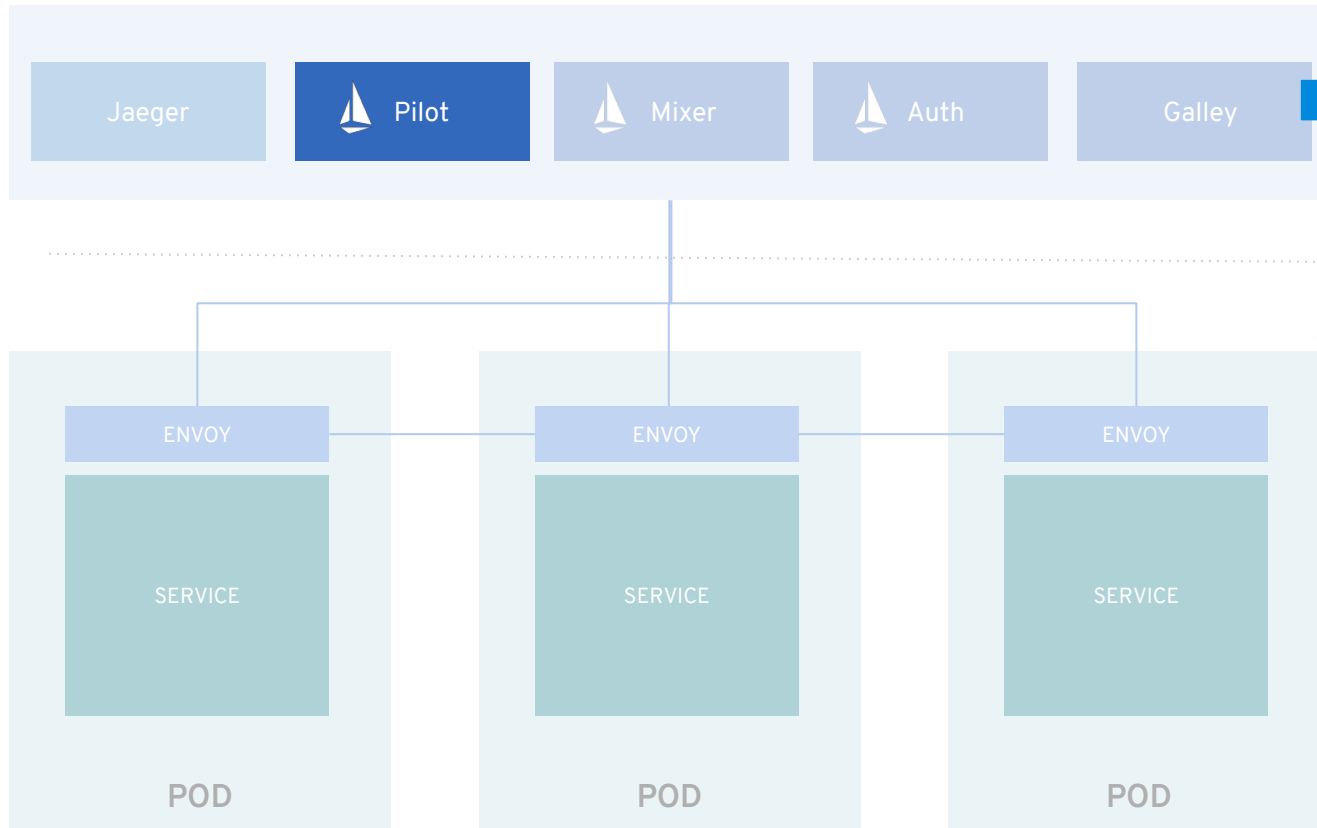
Control Plane

# Architecture - Envoy



**Envoy** is a high-performance proxy that intercepts all inbound and outbound traffic for all services in the service mesh.

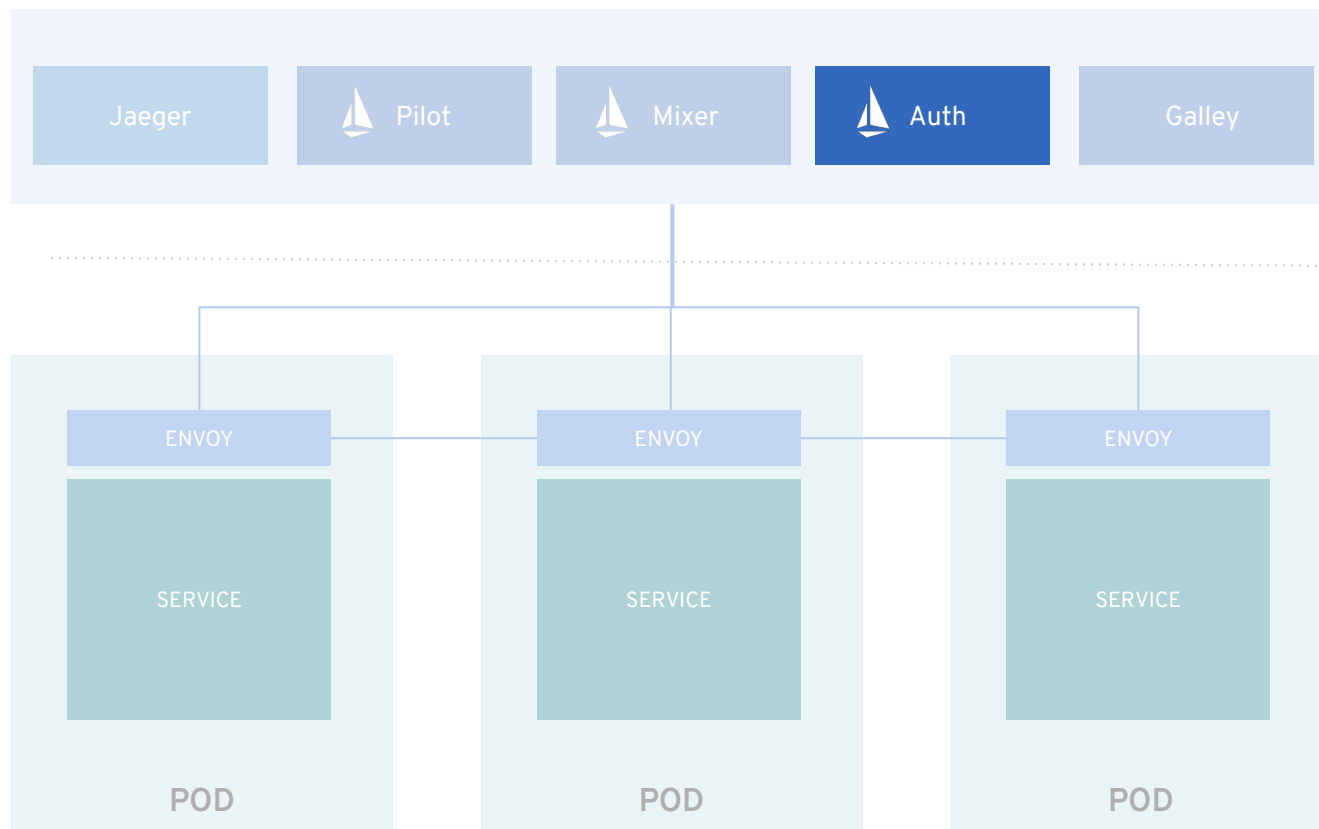
# Architecture - Pilot



**Pilot** configures the proxies at runtime and provides:

- Service discovery for the Envoy sidecars.
- Traffic management capabilities for intelligent routing.
- Resiliency.

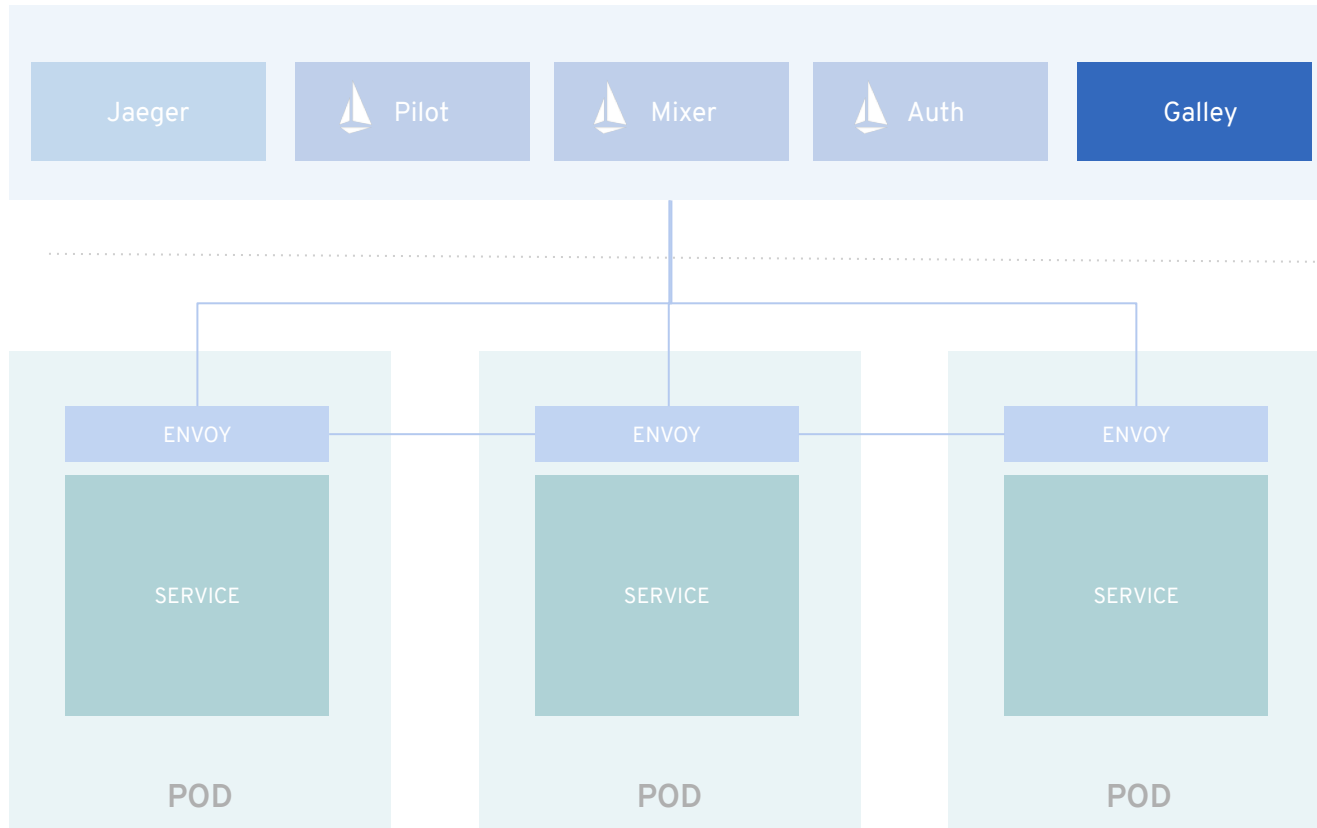
# Architecture - Auth



**Citadel** issues and rotates certificates. Citadel provides:

- Strong service-to-service and end-user authentication with built-in identity and credential management.
- Access control to services using authorization policies

# Architecture - Galley



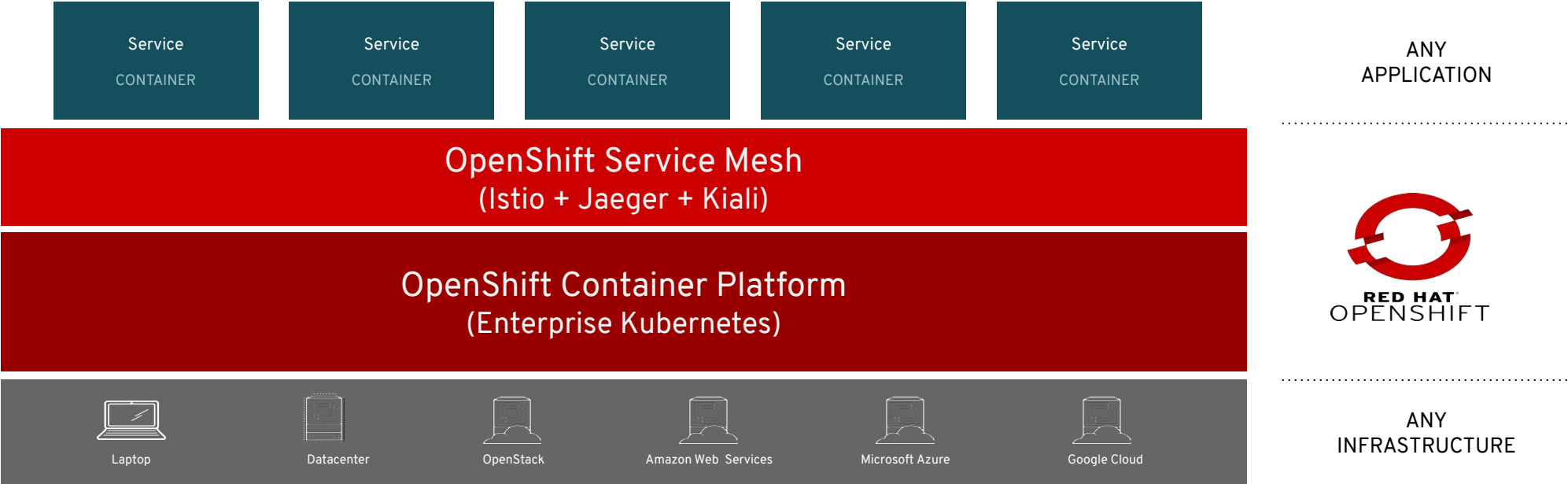
**Galley** ingests the service mesh configuration, then validates, processes, and distributes the configuration.



# OpenShift Service Mesh (OSSM)

# Red Hat OpenShift Service Mesh

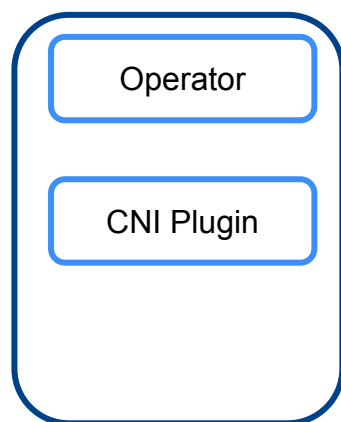
Based on the open source Istio project, Red Hat OpenShift Service Mesh provides a platform for behavioral insight and operational control over your networked microservices in a service mesh.



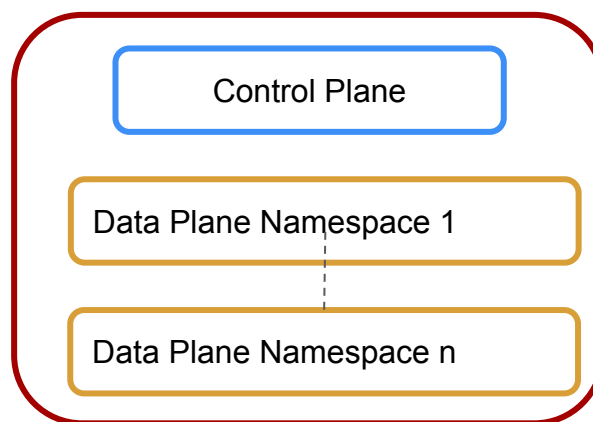
# Comparing OpenShift Service Mesh with Istio

- One of the main differences between Istio and OSSM is that OpenShift Service Mesh supports soft multi-tenancy
  - Several instances of system can run side-by-side in isolated manner
  - Several Istio control planes can run on single Kubernetes cluster, forming multiple meshes
- Automatic Injection
- Automatic OpenShift route creation
- CNI Plugin

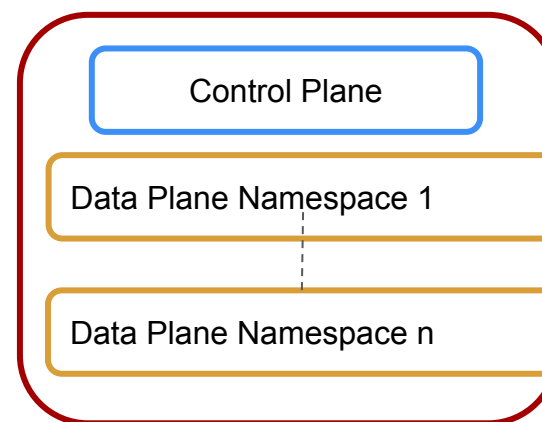
Operator Namespace



Service Mesh 1



Service Mesh n



# What's new in OpenShift ServiceMesh 2.0

# OpenShift Service Mesh 2.0

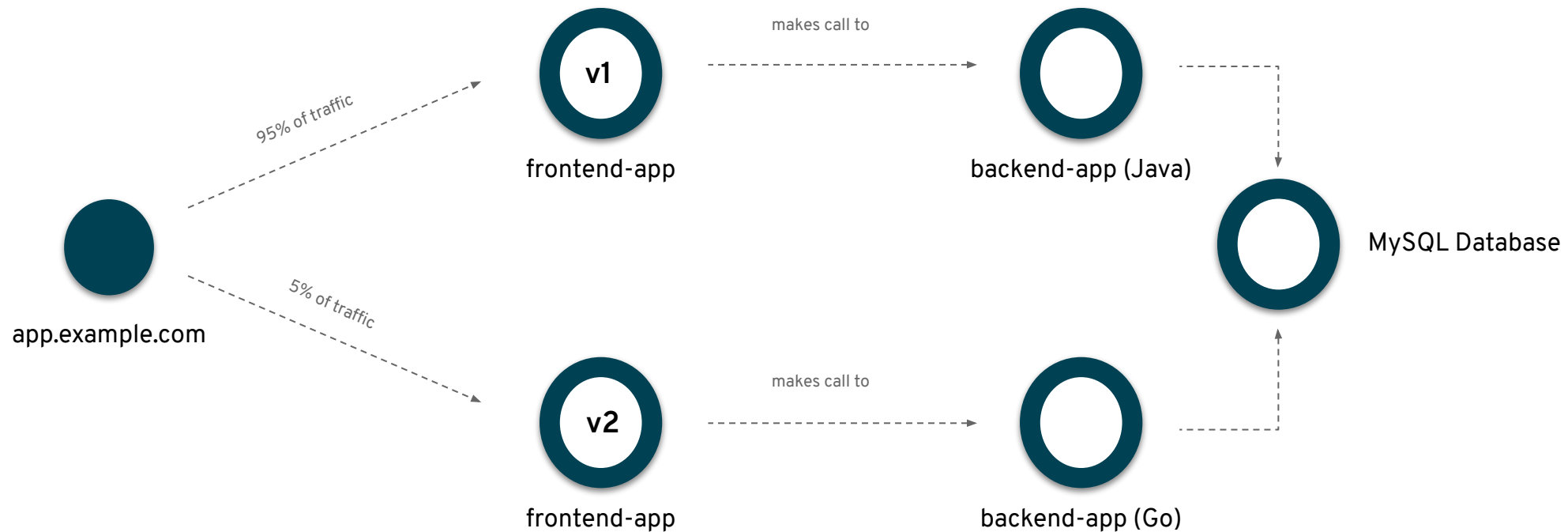
- Mixer component has been **deprecated**
- Pilot, Galley, Citadel, have been combined into a single binary known as ***Istiod***. The "d" stands for daemon
- Support for Envoy's Secret Discovery Service (SDS). SDS is a more secure and efficient mechanism for delivering secrets to Envoy side car proxies
  - Removes the need to use Kubernetes Secrets, which have well known security risks
  - Improves performance during certificate rotation, as proxies no longer require a restart to recognize new certificates
- Updates the ServiceMeshControlPlane resource to v2
- Introduces WebAssembly extensions as a **Technology Preview** feature



# Traffic Management

# Traffic Management

Traffic management decouples traffic flow and infrastructure scaling. This flexibility allows you to use Pilot to specify which rules to apply for traffic management between pods. Pilot and Envoy manage which pods receive traffic.



# Traffic Management

- Splitting traffic between versions
- Injecting faults
- Conditional rules, Destination Rules
- Advanced routing
  - Auto retries
  - Retry budgets
  - Request deadlines
  - Circuit breaking
- Advanced orchestration
  - Canary, blue/green
  - Per request routing
- Rate limiting



# Virtual Services

You can route requests dynamically to multiple versions of a microservice through Red Hat OpenShift Service Mesh with a virtual service.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
      end-user:
        exact: jason
      route:
      - destination:
          host: reviews
          subset: v2
        timeout: 10s
        retries:
          attempts: 3
          perTryTimeout: 2s
    - route:
      - destination:
          host: reviews
          subset: v3
        timeout: 10s
        retries:
          attempts: 3
          perTryTimeout: 2s
```

Condition and destination to which the request is routed if the condition is fulfilled

Timeout. Default: 15s

Automatic retry when error code 503 returned

# DestinationRule

Virtual services route traffic to a destination. Destination rules configure what happens to traffic at that destination.

```

apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: my-destination-rule
spec:
  host: my-svc
  trafficPolicy:
    loadBalancer:
      simple: RANDOM
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
    trafficPolicy:
      loadBalancer:
        simple: ROUND_ROBIN
  - name: v3
    labels:
      version: v3

```

- **ROUND\_ROBIN:** Default configuration, each instance gets a request in turn
- **RANDOM:** Requests are forwarded at random to instances in the pool.
- **WEIGHTED:** Requests are forwarded to instances in the pool according to a specific percentage.
- **LEAST\_REQUESTS:** Requests are forwarded to instances with the least number of requests.

Subset of my-svc

at

# Gateways

Gateways are primarily used to manage ingress traffic, but you can also configure egress gateways.

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: ext-host-gwy
spec:
  selector:
    app: my-gateway-controller
  servers:
  - port:
      number: 443
      name: https
      protocol: HTTPS
    hosts:
    - ext-host.example.com
    tls:
      mode: SIMPLE
      credentialName: ext-host-cert
```

HTTPS traffic from **ext-host.example.com** into the mesh

# ServiceEntry

A service entry adds an entry to the service registry that Red Hat OpenShift Service Mesh maintains internally. After you add the service entry, the Envoy proxies can send traffic to the service as if it was a service in your mesh.

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: svc-entry
spec:
  hosts:
  - ext-svc.example.com
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
```



Add the external host **ext-svc.example.com** to your OpenShift Service Mesh

You can configure virtual services and destination rules to control traffic to a service entry in the same way you configure traffic for any other service in the mesh.

# Observability

# Observability

A System is observable if current state can be understood from outside. Observability is an important characteristic of cloud-native distributed systems that helps you understand, operate, maintain and evolve the system

## Distributed Tracing Jaeger

Trace the path of a request as it travels across a complex system, discover the latency of the components along that path, and know which component in the path is creating a bottleneck.

## Monitoring Prometheus

All service-to-service communication goes through Envoy proxies, and the service mesh control plane is able to gather logs and metrics from these proxies.

## Visualization Kiali

Helps you define, validate, and observe the connections and microservices of the service mesh. It visualizes the service mesh topology and provides visibility into features such as request routing, circuit breakers, request rates, latency and more.



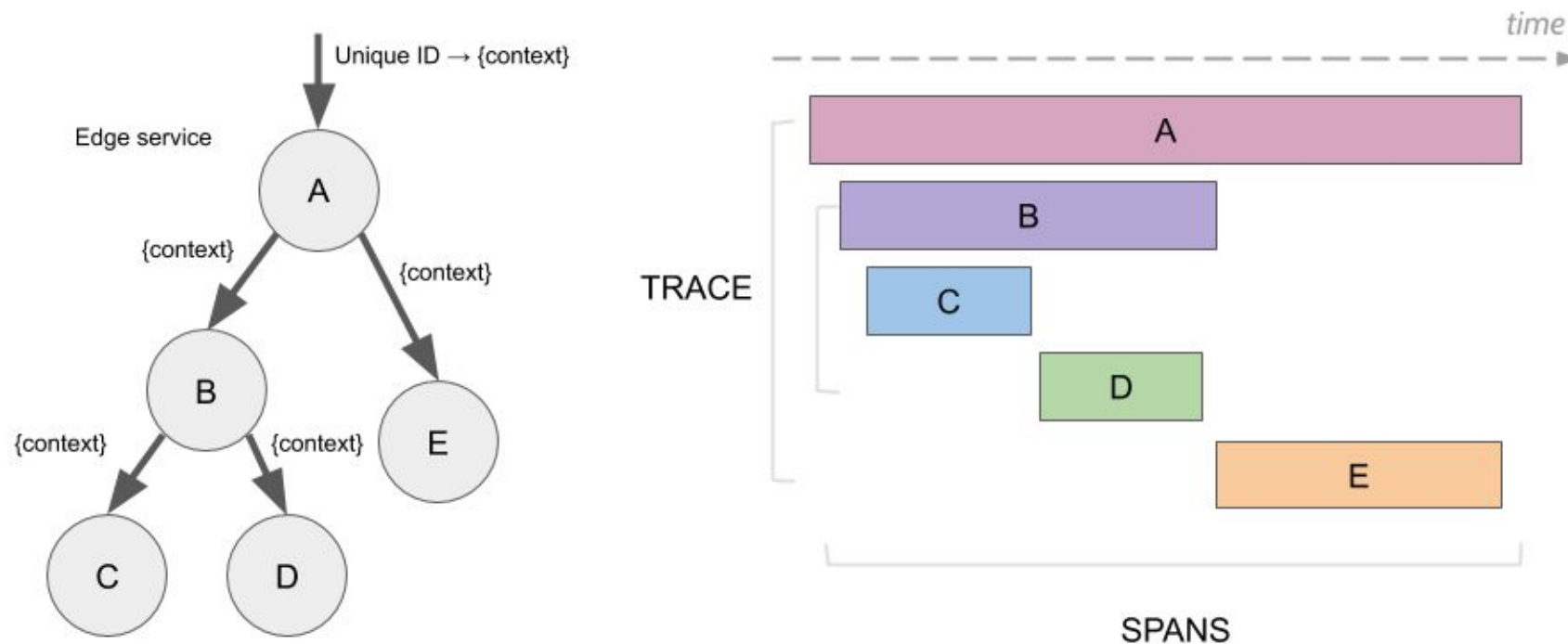
# Distributed Tracing

## Spans

Represents an individual unit of work done in a distributed system which consists of a named, timed operation representing a piece of the workflow.

## Trace

A visualization of the life of a request as it moves through a distributed system consisting of multiple related spans assembled together



# Distributed Tracing

Jaeger UI    Lookup by Trace ID...    Search    Compare    About Jaeger ▾

Search    JSON File

Service (4)  
 ▾

Operation (2)  
 ▾


Tags ⓘ

Lookback  
 ▾


Min Duration

Max Duration

Limit Results  
 ▾



Duration



Select a service

12 Traces    Sort: Most Recent ▾

Compare traces by selecting result items

<input type="checkbox"/>	istio-ingressgateway: user50-incident-service.user50-er-demo.svc.cluster.local:8080/incidents* 2d8b6be	8.82ms
2 Spans <input type="button" value="istio-ingressgateway (1)"/> <input type="button" value="user50-incident-service.user50-er-demo (1)"/>		
Today 2:52:05 pm 5 minutes ago		
<input type="checkbox"/>	istio-ingressgateway: user50-incident-service.user50-er-demo.svc.cluster.local:8080/incidents* e7f8778	9.3ms
2 Spans <input type="button" value="istio-ingressgateway (1)"/> <input type="button" value="user50-incident-service.user50-er-demo (1)"/>		
Today 2:52:04 pm 5 minutes ago		
<input type="checkbox"/>	istio-ingressgateway: user50-incident-service.user50-er-demo.svc.cluster.local:8080/incidents* d1b7e29	9.14ms
2 Spans <input type="button" value="istio-ingressgateway (1)"/> <input type="button" value="user50-incident-service.user50-er-demo (1)"/>		
Today 2:52:03 pm 5 minutes ago		
<input type="checkbox"/>	istio-ingressgateway: user50-incident-service.user50-er-demo.svc.cluster.local:8080/incidents* 3905c0c	12.42ms
Today 2:52:02 pm 5 minutes ago		

Trace your services



# Distributed Tracing

Jaeger UI  Search Compare About Jaeger ▾

← ▾ istio-ingressgateway: user50-incident-service.user50-er-demo.svc.cluster.local:8080/incidents\* 2d8b6be  Trace Timeline ▾

Trace Start **November 18, 2019 2:52 PM** Duration **8.82ms** Services **2** Depth **2** Total Spans **2**

Service & Operation ▾ > ▾ >> 0ms 2.2ms 4.41ms 6.61ms 8.82ms

▾ istio-ingressgateway user50-incident-service.user50-er-demo...

**user50-incident-service.user50-er-demo.svc.cluster.local:8080/incidents\*** Service: **istio-ingressgateway** Duration: **8.82ms** Start Time: **0ms**

▾ **Tags**

component	"proxy"
node_id	"router-10.131.1.134-istio-ingressgateway-58d75cdb88-8nsbg.admin50-istio-system-admin50-istio-system.svc.cluster.local"
guid:x-request-id	"a93aae5c-4cfa-9f6c-80f7-d96870bd1f55"
http.url	"https://incident-service.user50.apps.cluster-e3eb.e3eb.example.opentlc.com/incidents"
http.method	"GET"
downstream_cluster	"-"
user_agent	"curl/7.65.3"
http.protocol	"HTTP/2"
request_size	0
upstream_cluster	"outbound 8080  user50-incident-service.user50-er-demo.svc.cluster.local"
http.status_code	200
response_size	1158
response_flags	"-"
span.kind	"client"
internal.span.format	"zipkin"

> **Process:** ip = 10.131.1.134

SpanID: ec2d9a33dfcf6bc2 vvvvvv000

Click on a trace to see trace details

# Metrics and Monitoring

Prometheus Alerts Graph Status Help

## Targets

All Unhealthy

citadel (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.128.2.6:15014/metrics	UP	instance="10.128.2.6:15014" job="citadel"	9.243s ago	4.034ms	

envoy-stats (16/16 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.128.2.23:15090/stats/prometheus	UP	app="istio-egressgateway" chart="gateways" heritage="Tiller" instance="10.128.2.23:15090" istio="egressgateway" job="envoy-stats" maistra_control_plane="admin50-istio-system" namespace="admin50-istio-system" pod_name="istio-egressgateway-675fb4f47-zrvwg" pod_template_hash="675fb4f47" release="istio"	7.19s ago	3.505ms	
http://10.128.2.29:15090/stats/prometheus	UP	app="user50-incident-service" deployment="user50-incident-service-9" deploymentconfig="user50-incident-service" group="erd-services" instance="10.128.2.29:15090" job="envoy-stats" namespace="user50-er-demo" pod_name="user50-incident-service-9-ja28c" version="v1"	10.145s ago	3.741ms	
http://10.128.2.33:15090/stats/prometheus	UP	app="user50-disaster-simulator" deployment="user50-disaster-simulator-3" deploymentconfig="user50-disaster-simulator" group="erd-services" instance="10.128.2.33:15090" job="envoy-stats" namespace="user50-er-demo" pod_name="user50-disaster-simulator-3-prz6r"	12.541s ago	3.832ms	
http://10.128.2.34:15090/stats/prometheus	UP	app="postgresql" deployment="postgresql-2" deploymentconfig="postgresql" instance="10.128.2.34:15090" job="envoy-stats" name="postgresql" namespace="user50-er-demo" pod_name="postgresql-2-2hq9g"	8.376s ago	3.838ms	
http://10.128.2.4:15090/stats/prometheus	UP	app="telemetry" chart="mixer" heritage="Tiller" instance="10.128.2.4:15090" istio="mixer" istio_mixer_type="telemetry" job="envoy-stats" maistra_control_plane="admin50-istio-sys"	6.373s ago	6.513ms	

- **Envoy-stats:** The different envoy proxies.
- **Istio-mesh:** Service mesh metrics
- **citadel, pilot, galley, istio-telemetry, istio-policy:** The metrics exposed by the control plane components about themselves.
- **kubernetes-service-endpoints:** Service endpoints that do not necessarily belong to the mesh.

# Kiali

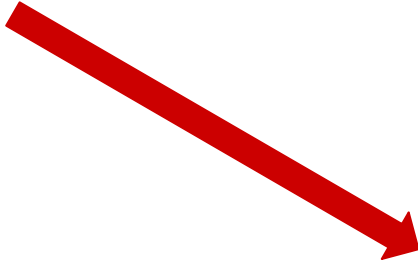
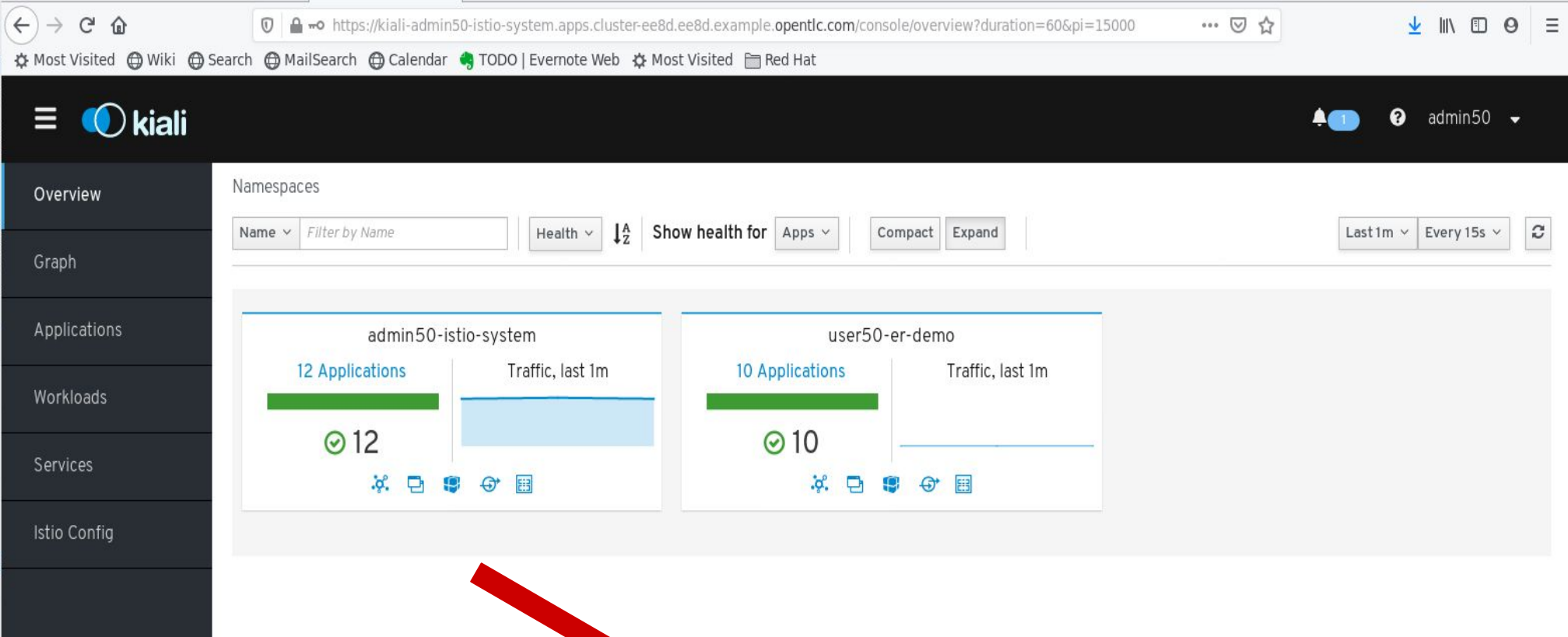
Kiali provides observability for your service mesh. By using Kiali you can view configurations, monitor traffic, and view and analyze traces in a single console.

- OpenShift Service Mesh console
- Visualizes service mesh topology in real time
- Provides visibility into features like request routing, circuit breakers, request rates, latency, etc.
- Inline edition of YAML representation of Istio resources, with powerful semantic validation
- Actions to create, update, delete Istio configuration resources, driven by wizards
- Custom metrics dashboards
- Integrated with distributed tracing



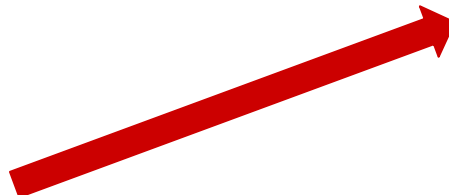
Kiali Logo Color Exploration: Round 1 - Version 3

# Kiali



Namespaces that form the Service Mesh

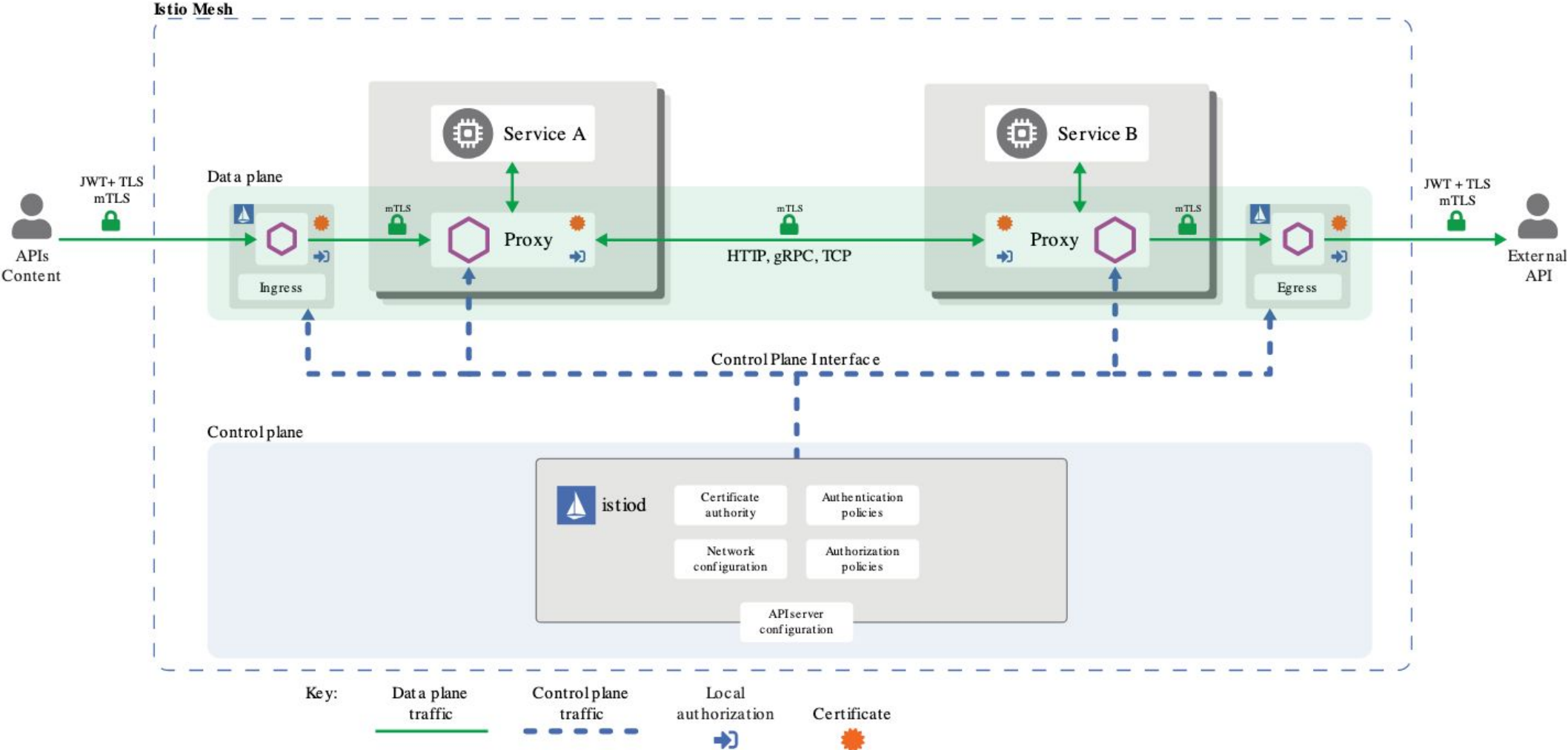
View topology graph of the service mesh based on real-time traffic. The graph changes when traffic hits the services



The screenshot shows the Kiali web interface. The left sidebar has a menu with 'Overview', 'Graph', 'Applications', 'Workloads', 'Services', and 'Istio Config'. The 'Graph' item is highlighted with a red box. The main area displays a topology graph for the namespace 'user50-er-demo'. The graph shows several services: 'user50-process-service' (highlighted in a box), 'user50-responder-service', 'user50-responder-simulator', 'user50-emergency-console', 'user50-incident-priority-service', 'user50-incident-service', and 'user50-mission-service'. There are also 'postgresql' services. Edges connect these services, indicating traffic flow. A red box highlights the 'user50-process-service' node and its connection to a 'postgresql' node. The right sidebar shows a 'Current Graph' summary with 8 apps, 2 services, and 5 edges. Below that is a traffic chart for HTTP and TCP, showing 'Sent' and 'Received' traffic over time.

# Security

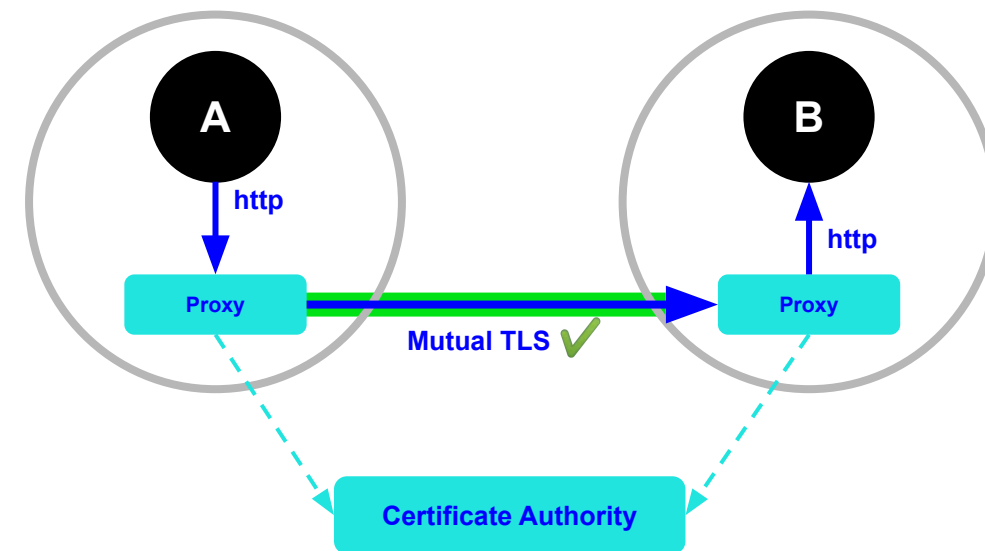
# Istio Security



# Transport authentication

- Connections can be setup to use mTLS.
- No configuration needed on the apps.
- Certificates are provisioned by Istio (Citadel)

```
apiVersion: maistra.io/v1
kind: ServiceMeshControlPlane
metadata:
  name: custom-install
  namespace: istio-system
spec:
  istio:
    global:
      tag: 1.1.0
      hub: registry.redhat.io/openshift-service-mesh/
      mtls:
        enabled: false
      disablePolicyChecks: true
```



mTLS is set to false by default. This means that mTLS is not enforced, and services are able to communicate over plain HTTP.



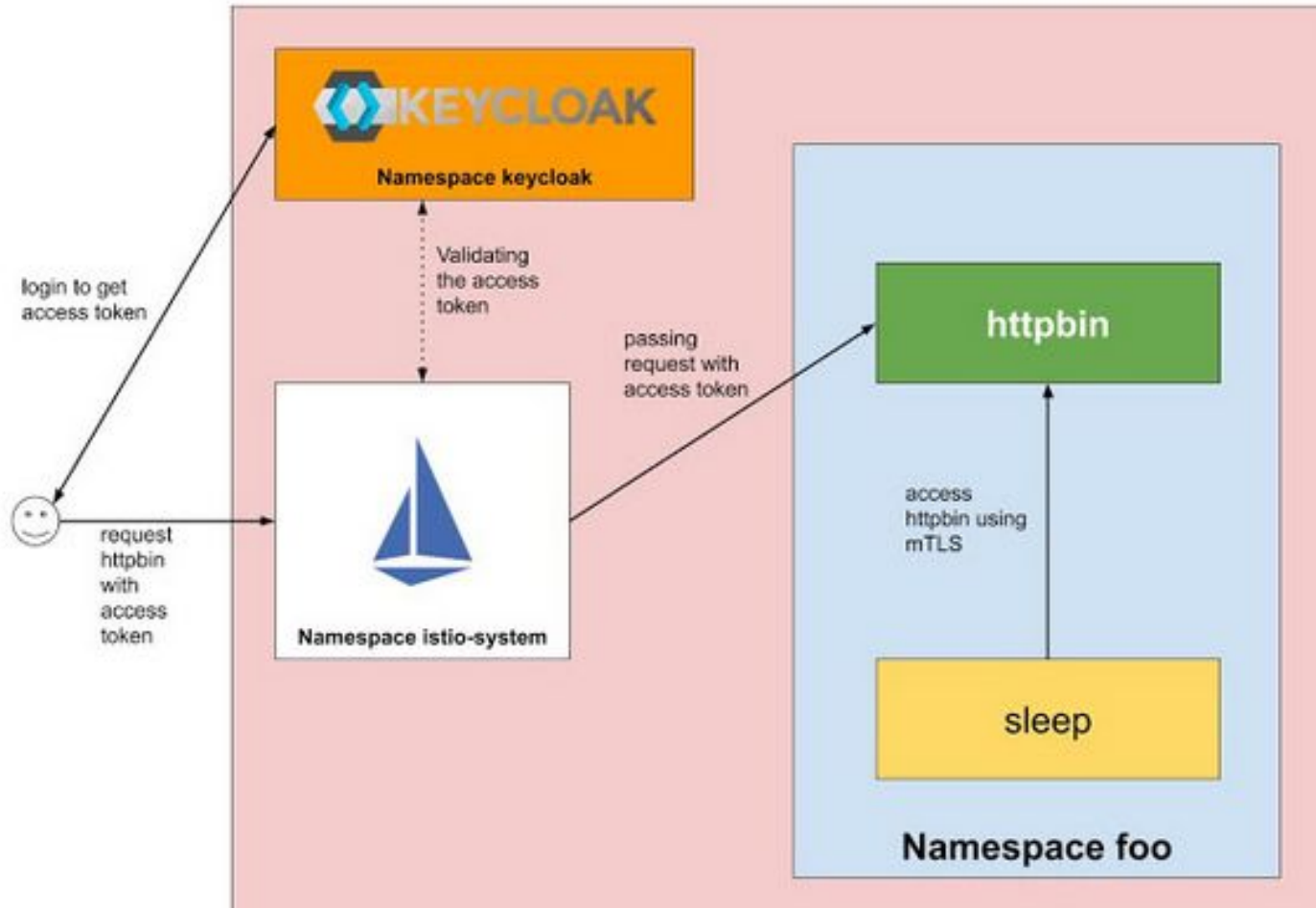
# Demo

- Install OpenShift Service Mesh
- Deploy “bookinfo” application
- Observability
  - Tracing
  - Kiali
- Traffic Control
  - Simple routing
- Security
  - Mutual TLS

# Red Hat SSO (Keycloak)

- Keycloak is an open source **identity** and **Access management** solution
- Red Hat Single Sign-On (RH-SSO) is based on the Keycloak project
- Keycloak is a single sign-on solution for web apps and RESTful web services
- Keycloak supports standard protocols like OAuth 2.0, OpenID Connect, SAML 2.0
  - Acts as a centralized authentication server
  - Provides user federation to sync users from LDAP and Active Directory servers
  - Integrates with 3rd party identity providers including social networks
  - Provides Rest APIs and an administration GUI for central management of users, roles, role mappings, clients and configuration.
- Installation and configuration of the Keycloak SSO server on OpenShift can now be automated using the SSO operator in OpenShift

# OpenID Connect Flow



# Authentication

A request authentication policy with jwt issuer

```
$ kubectl apply -f - <<EOF
apiVersion: "security.istio.io/v1beta1"
kind: "RequestAuthentication"
metadata:
  name: "jwt-example"
  namespace: foo
spec:
  selector:
    matchLabels:
      app: httpbin
  jwtRules:
  - issuer: "testing@secure.istio.io"
    jwksUri: "https://raw.githubusercontent.com/istio/istio/release-1.8/security/tools/jwt/samples/jwks.json"
EOF
```

# Authorization Policy

A authorization policy to allow request

```
$ kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: require-jwt
  namespace: foo
spec:
  selector:
    matchLabels:
      app: httpbin
  action: ALLOW
  rules:
  - from:
    - source:
      requestPrincipals: ["testing@secure.istio.io/testing@secure.istio.io"]
    when:
    - key: request.auth.claims[groups]
      values: ["group1"]
EOF
```

# Demo


- Security
  - Setup keycloak server
  - Apply configuration to enable authentication with Keycloak user

# Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

 [linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)

 [facebook.com/redhatinc](https://www.facebook.com/redhatinc)

 [youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)

 [twitter.com/RedHat](https://twitter.com/RedHat)

