Red Hat Summit

Connect

Red Hat OpenShift Service Mesh

Czy to recepta na liczne wymagania stawiane aplikacjom?

Krzysztof Korzela
RHCA, Senior Solution Architect

Red Hat | intel.

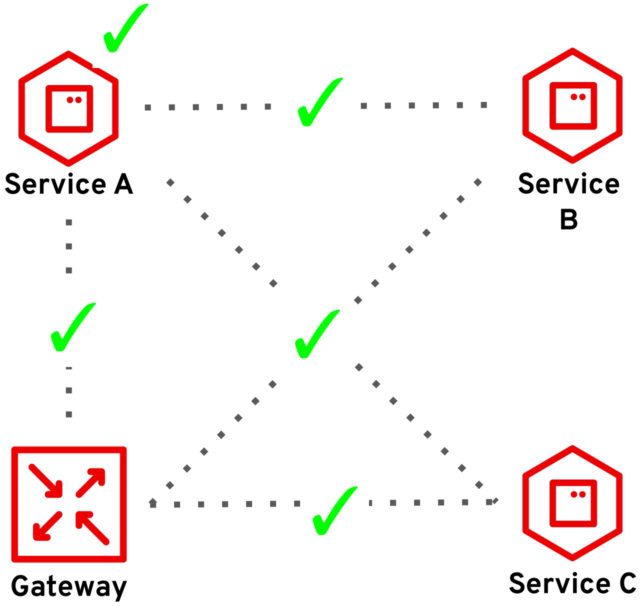
Why Service Mesh?

“Distributed Systems are hard”

Developing Microservices

A Common Pattern

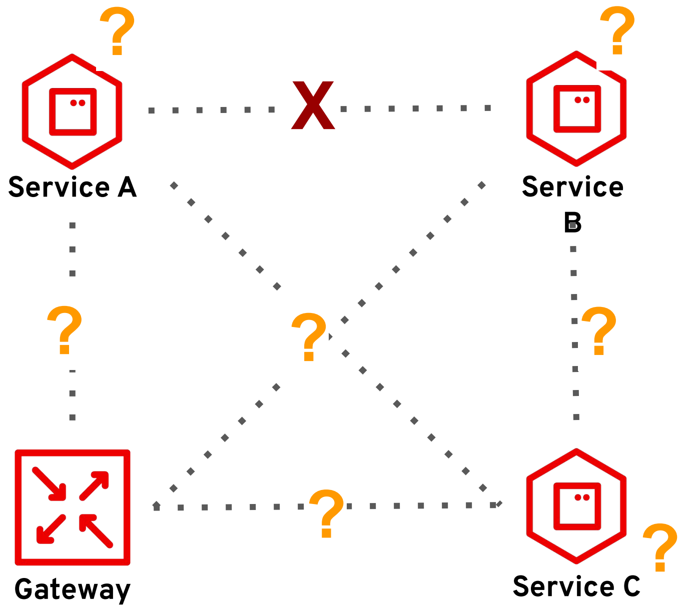
- A common pattern when developing microservices.
- In Development:
  - New services are written.
  - They are tested locally - looks good!
  - The are tested in a staging cluster - looks good!
- LGTM, Ship it!



Microservice in Production

A Common Pattern

- In production, things become less predictable:
  - Sporadic delays and failures are seen.
  - Performance is not as expected.
  - Security holes may be discovered.
  - Services are scaled, but performance doesn't improve.
  - Fixes are made, but upgrades cause further issues.

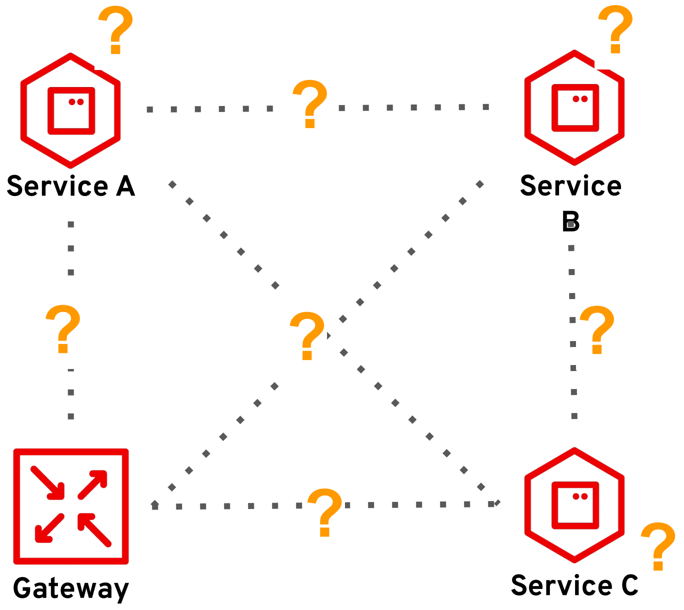


- **Microservices are distributed systems and troubleshooting distributed systems is hard!**

The Fallacies of Distributed Computing

Microservices are Distributed Systems

- These challenges are a result of the fallacies of distributed computing:
  - The network is reliable.
  - Latency is zero.
  - Bandwidth is infinite.
  - The network is secure.
  - Topology doesn't change.
  - There is one administrator.
  - Transport cost is zero.
  - The network is homogeneous.

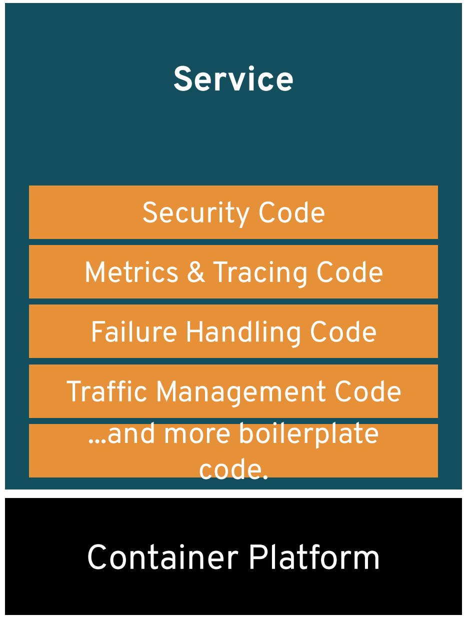


Why Service Mesh?

Solving Microservices Challenges with Code

- These challenges are often mitigated with:
  - **Code** to handle failures between services.
  - Logs, metrics and traces in **source code**.
  - **3rd party libraries** for managing deployments, security and more.
- This results in:
  - Different solutions in different services.
  - Boilerplate code.
  - New dependencies to keep up date.

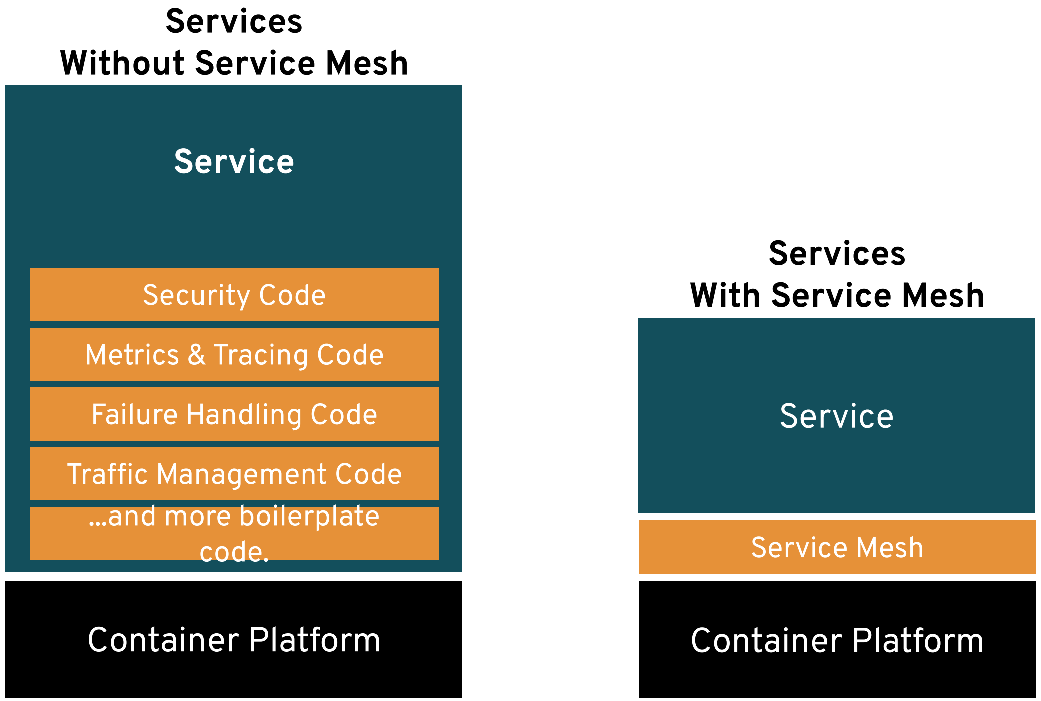
Every Service



Why Service Mesh?

An Abstraction for Microservice Challenges

- Service Mesh solve distributed systems challenges at **a common infrastructure layer.**
- This reduces boilerplate code and copy/paste errors across services.
- Enforces common policies across all services.
- Removes the obligation to implement cross cutting concerns from developers.



Services Without Service Mesh

Service

Security Code

Metrics & Tracing Code

Failure Handling Code

Traffic Management Code

...and more boilerplate code.

Container Platform

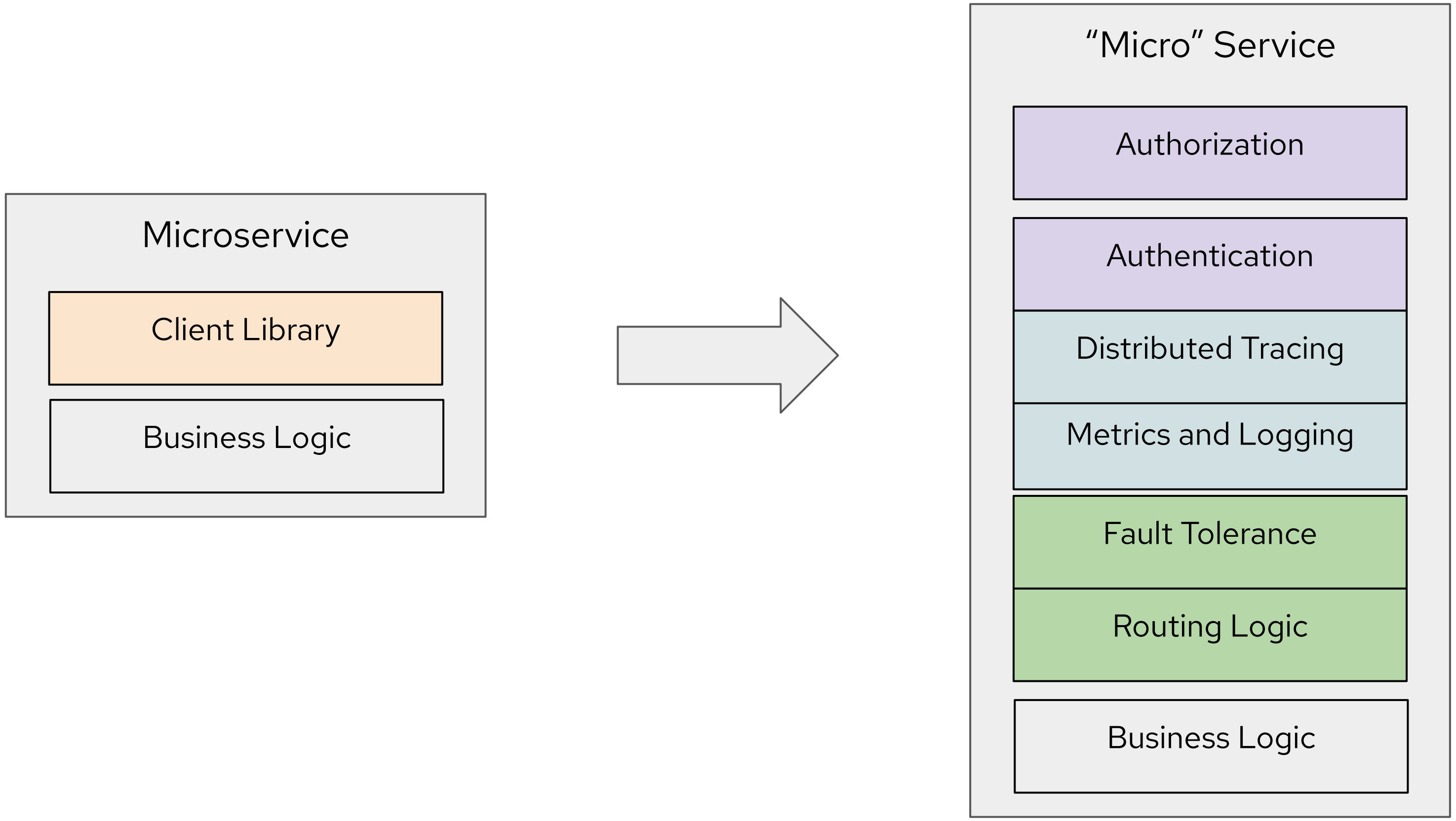
Services With Service Mesh

Service

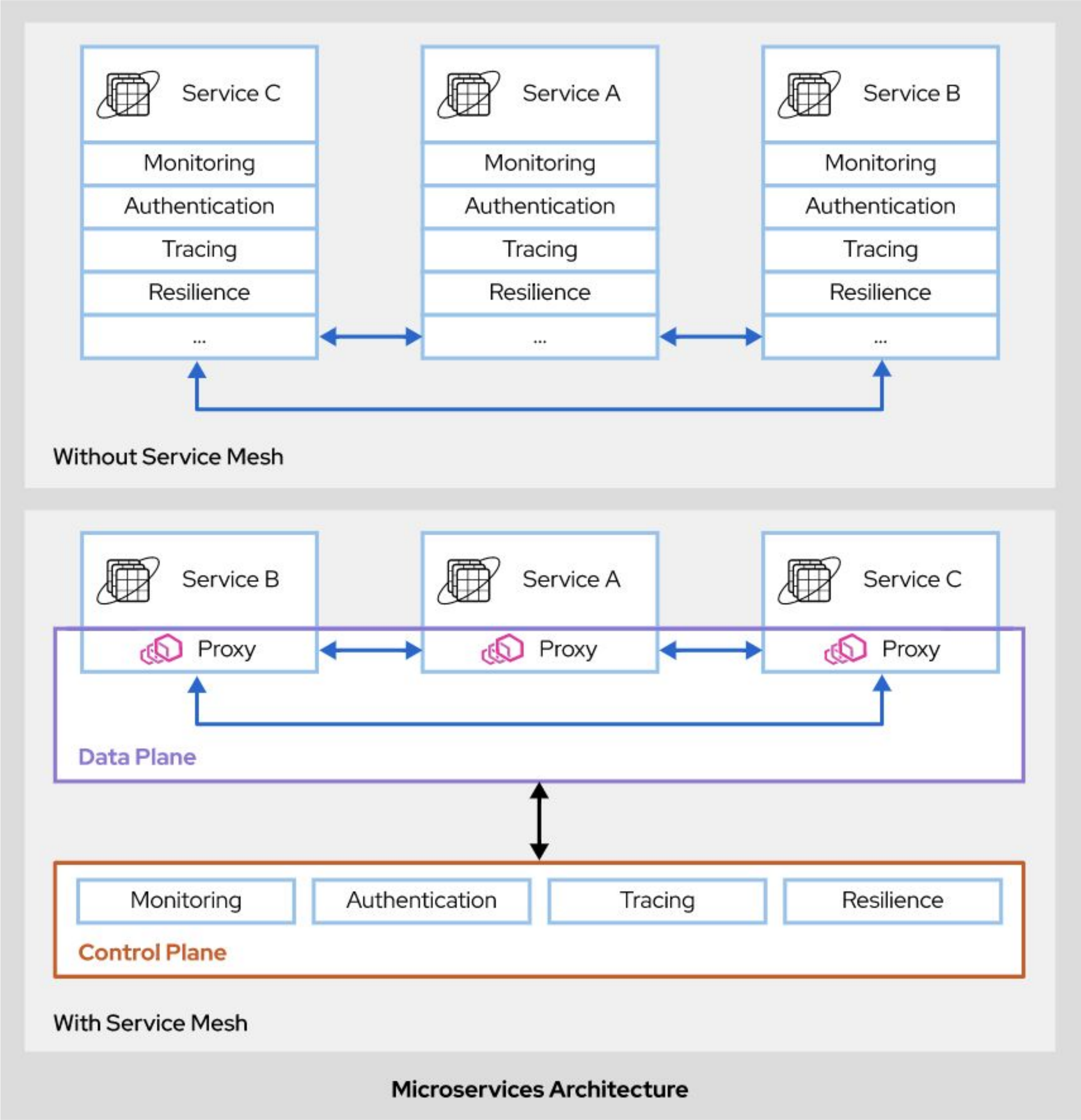
Service Mesh

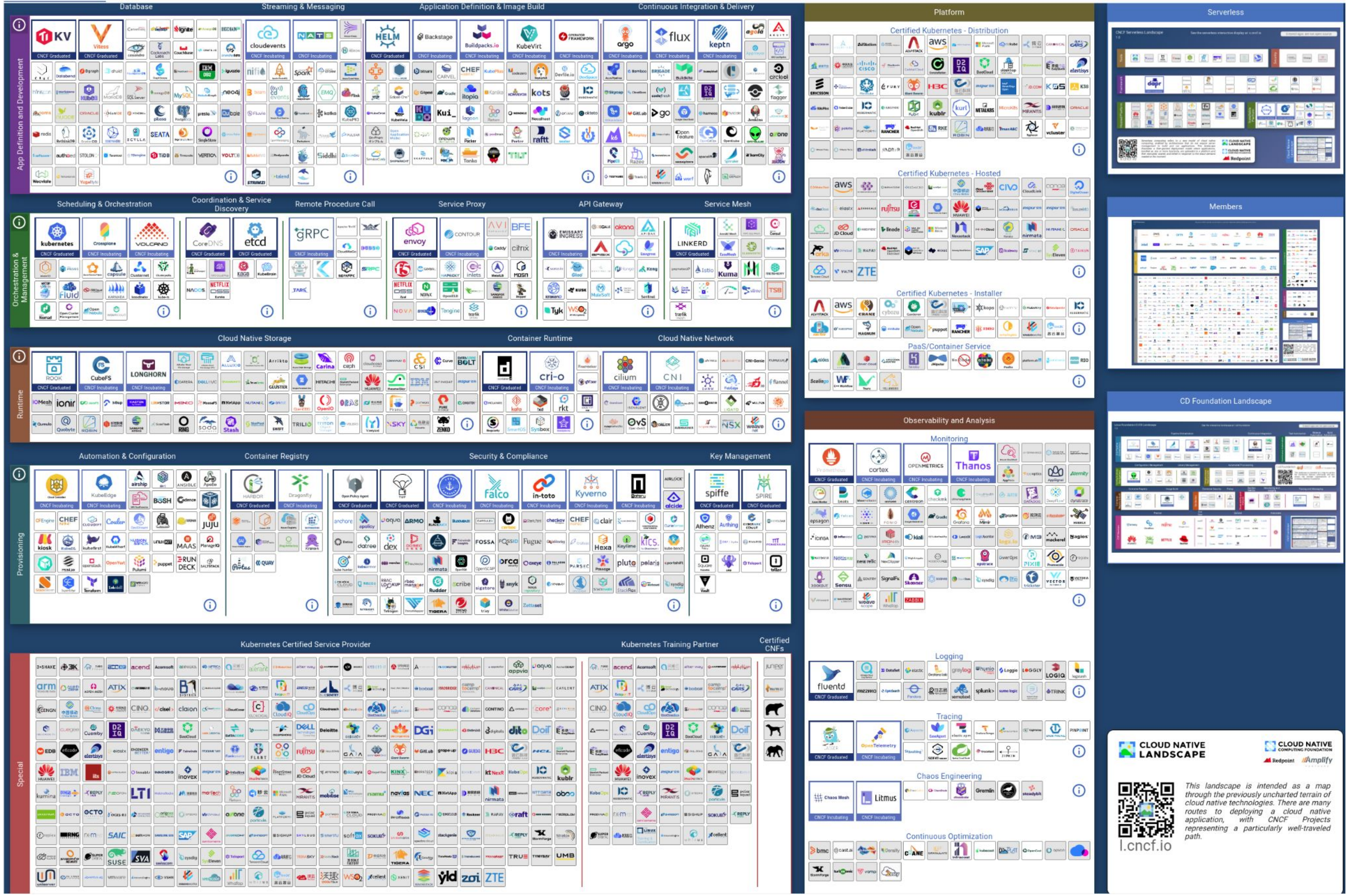
Container Platform

Is This a “Micro” Service?



Does it *really* make sense to push operational challenges to developers to deal with?


Service C
Monitoring
Authentication
Tracing
Resilience
...
Service A
Monitoring
Authentication
Tracing
Resilience
...
Service B
Monitoring
Authentication
Tracing
Resilience
...
Without Service Mesh
Service B
Proxy
Service A
Proxy
Service C
Proxy
Data Plane
Monitoring
Authentication
Tracing
Resilience
Control Plane
With Service Mesh
Microservices Architecture


App Definition and Development
Database
Streaming & Messaging
Application Definition & Image Build
Continuous Integration & Delivery
Orchestration & Management
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
API Gateway
Service Mesh
Runtime
Cloud Native Storage
Container Runtime
Cloud Native Network
Provisioning
Automation & Configuration
Container Registry
Security & Compliance
Key Management
Special
Kubernetes Certified Service Provider
Kubernetes Training Partner
Certified CNFs
Platform
Certified Kubernetes - Distribution
Certified Kubernetes - Hosted
Certified Kubernetes - Installer
PaaS/Container Service
Observability and Analysis
Monitoring
Logging
Tracing
Chaos Engineering
Continuous Optimization
Serverless
Members
CD Foundation Landscape
CLOUD NATIVE LANDSCAPE
CLOUD NATIVE COMPUTING FOUNDATION
Redpoint
Amplify

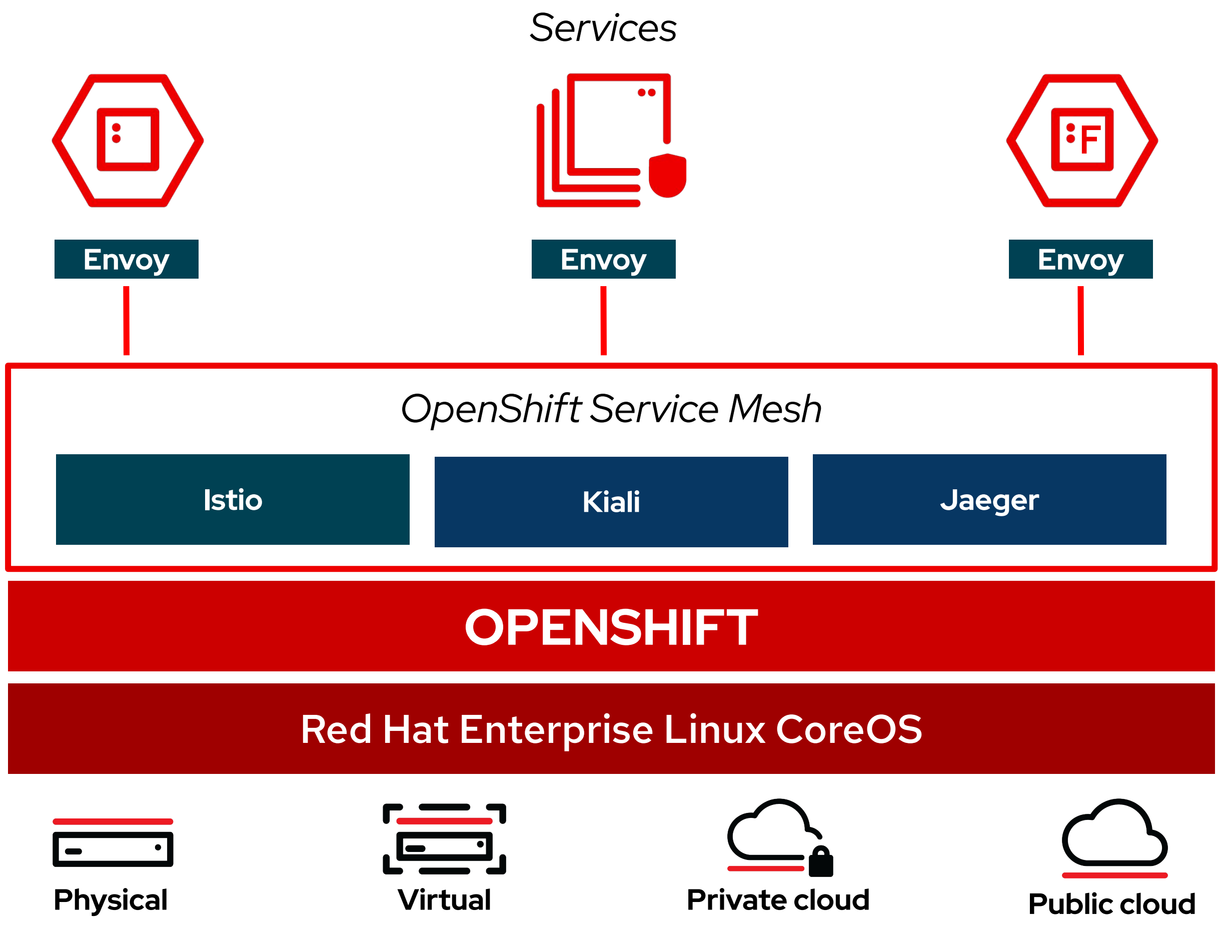
This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.

l.cncf.io

OpenShift Service Mesh

Connect, Secure, Control and Observe Services on OpenShift

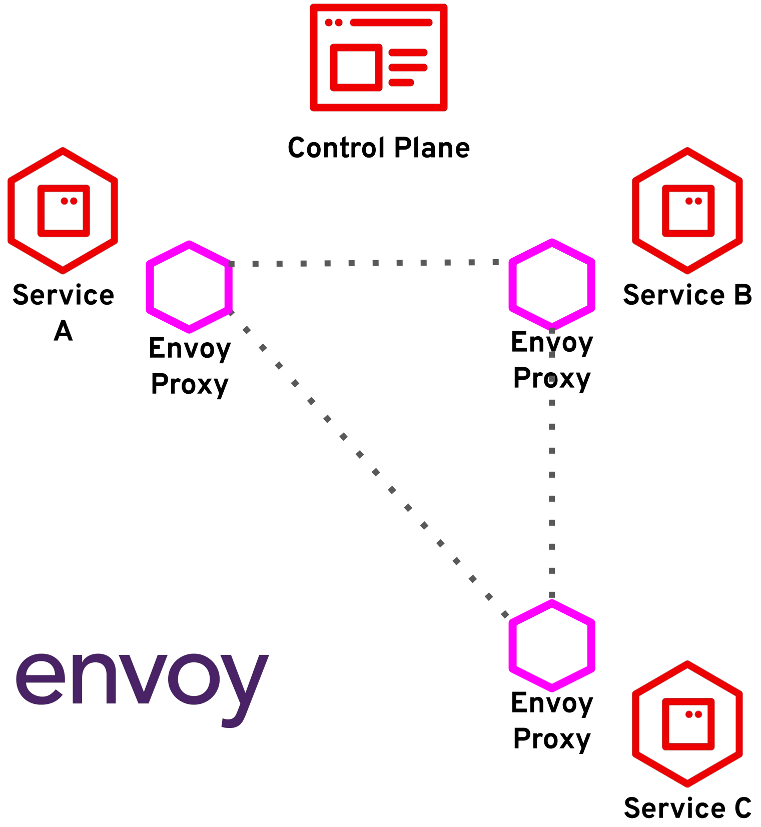
- A software infrastructure layer between Kubernetes and your services for managing communications.
- **Handles common "microservice" challenges, so that developers don't have to:**
  - Security
  - Monitoring & Observability
  - Application Resilience
  - Upgrades, Rollouts & A/B Testing
  - And more...



What is a Service Mesh?
“Envoy Proxies and a Control Plane”

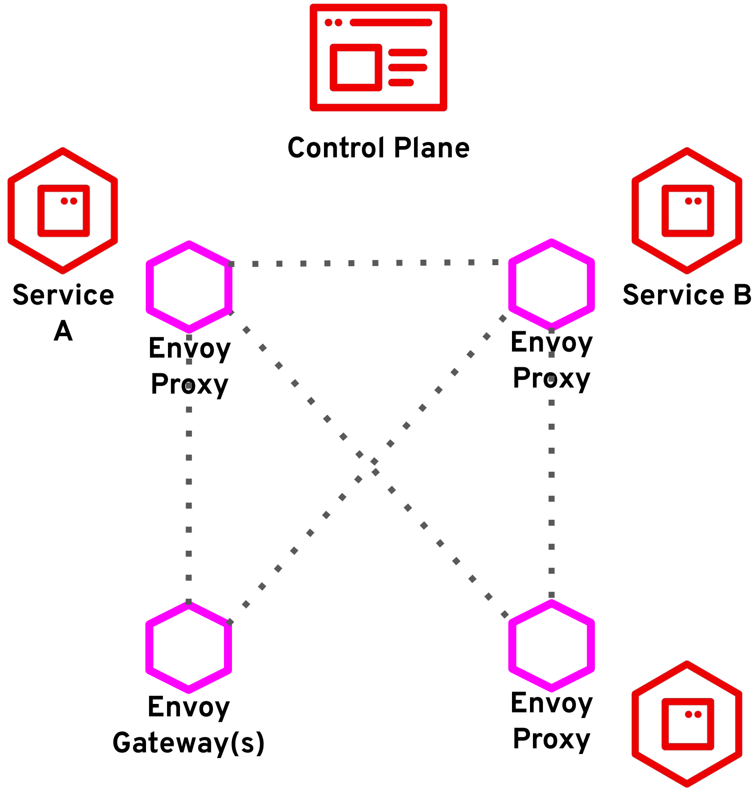
Connecting Services within the Mesh

- 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**.

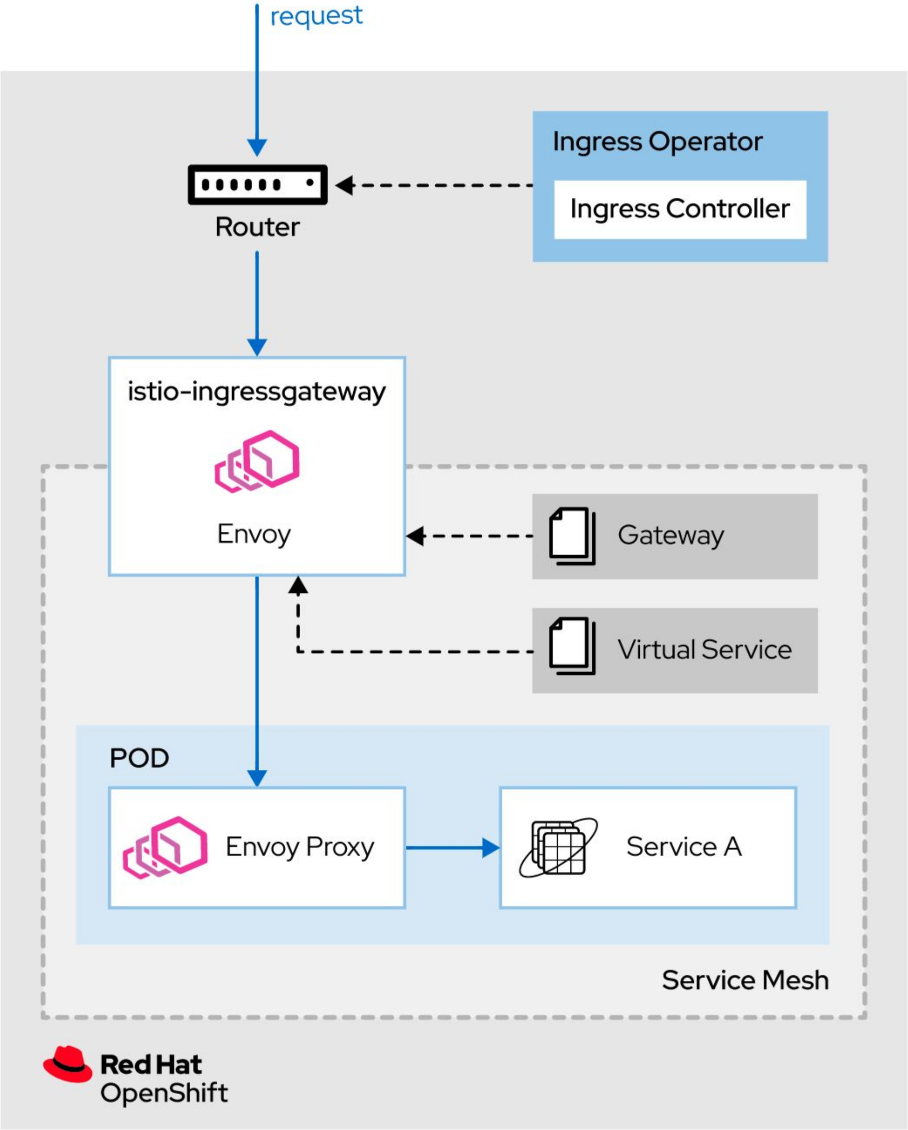


Connecting Services Outside the Mesh

- External communication occurs via **Gateway proxies**, that are also part of the mesh.
- **Ingress Gateways** manage traffic entering the mesh.
    - An alternative to Kubernetes Ingress, with additional mesh features.
- **Egress Gateways** manage traffic exiting the mesh.
    - Can require all external services to be registered.
- On OpenShift, Service Mesh **Ingress Gateways** can be used in conjunction with an **OpenShift route** or on their own.



What is a Service Mesh ?



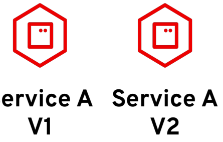
Service Mesh Use Cases

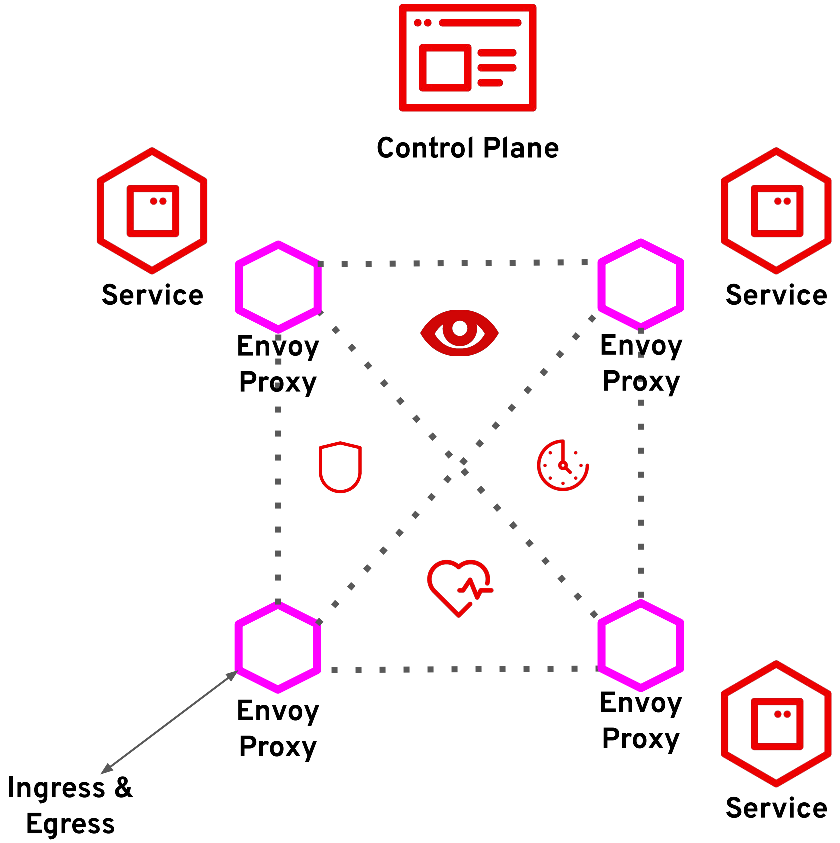
“Connect, secure, observe and control traffic”

Red Hat | intel.

Service Mesh Use Cases

- Securing Services
- Management
- Monitoring & Observing Services
- Building Resilient Services
- Releasing Services





Red Hat Summit

Connect

Thank you

in linkedin.com/company/red-hat

f facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat | intel.