Red Hat | intel

Open Tour

Connecting people and solutions
to accelerate your business

About me

Nikolaus Lemberski

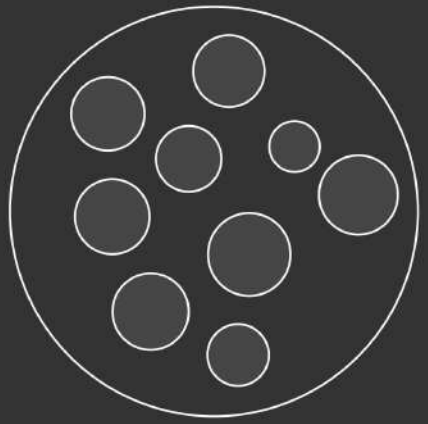
Senior Specialist Solution Architect

- more than 20 years Java EE and Spring development
- Cloud Native Software Development
- Microservice Architectures
- Kubernetes

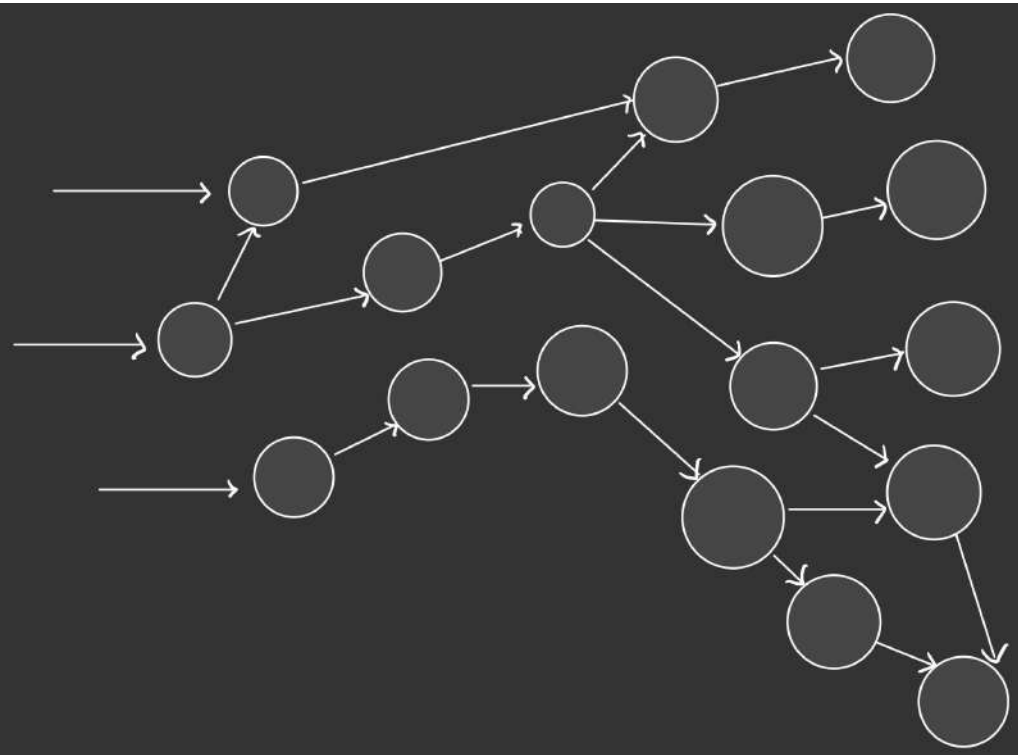
www.linkedin.com/in/nikolauslemberski

From Monolith...

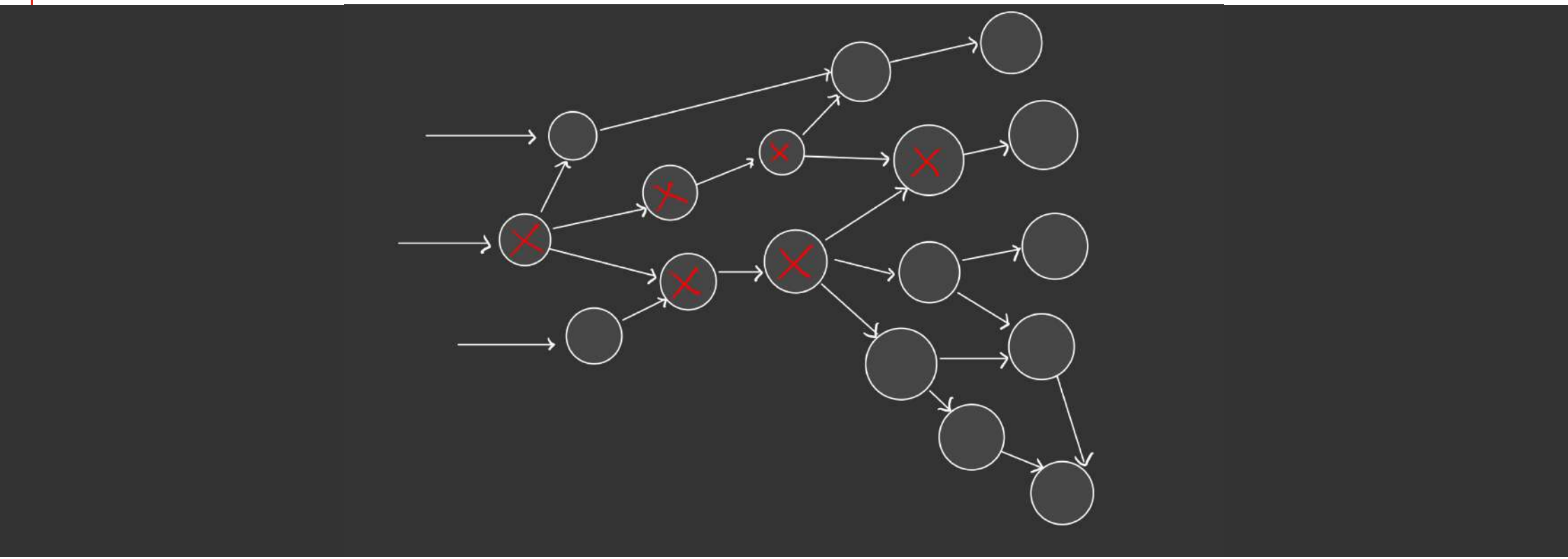
...to Microservices



Network of Microservices



Cascading Failures, Service Discovery, Network Latency, Observability... 😱



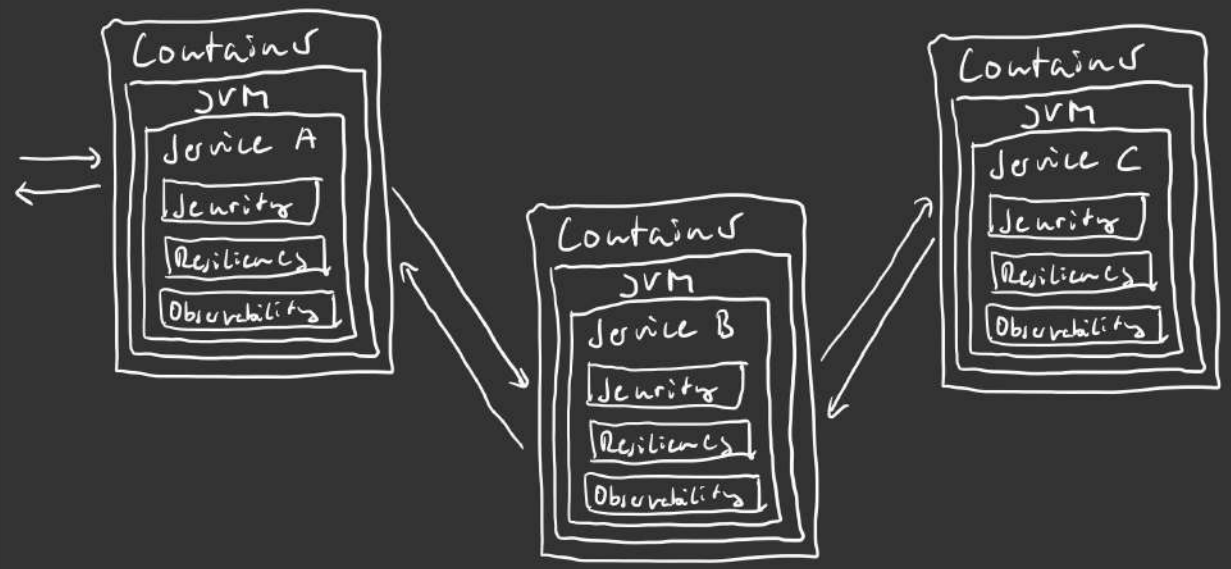
Microservice pattern to the rescue

Spring Microservices in Action

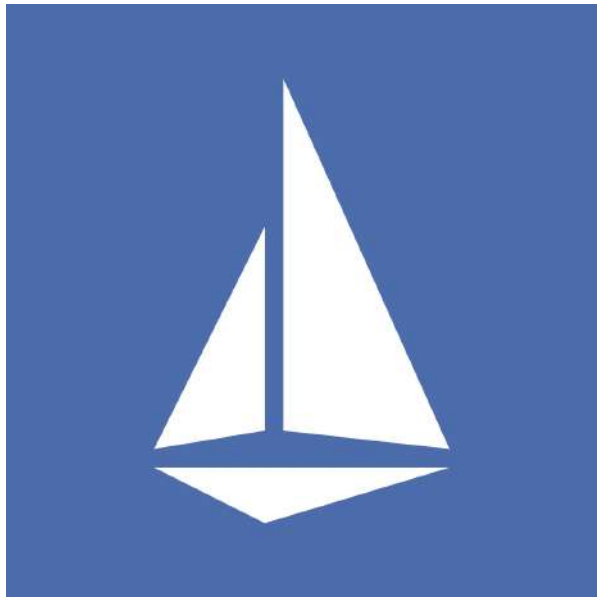
John Carnell

June 2017

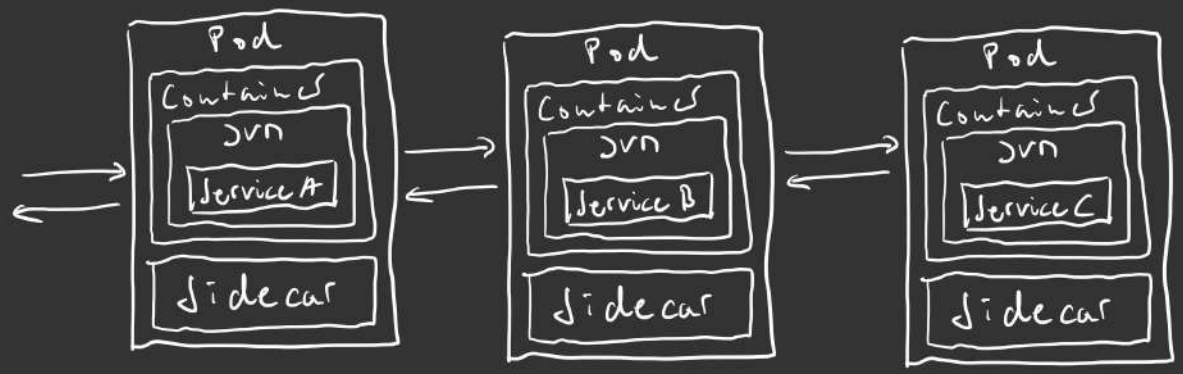
Embedding capacities in our Microservices



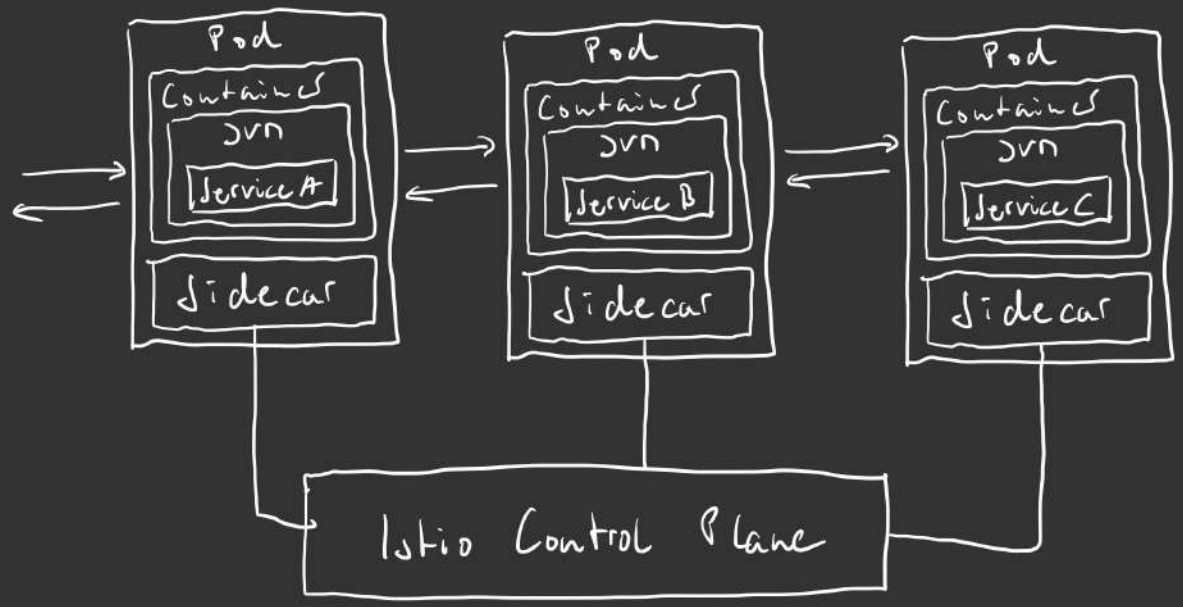
Red Hat OpenShift Service Mesh, based on Istio



Envoy Sidecar



Data Plane and Control Plane



Installation

Red Hat OpenShift

kubeadmin

Administrator

Home

Operators

OperatorHub

Installed Operators

Workloads

Networking

Storage

Builds

Observe

Compute

User Management

Administration

Project: All Projects

OperatorHub

Discover Operators from the Kubernetes community and Red Hat partners, curated by Red H
ons and shared services to your developers. After installation, the Operator capabilities will ap

All Items

AI/Machine Learning

Application Runtime

Big Data

Cloud Provider

Database

Developer Tools

Development Tools

Drivers and plugins

Integration & Delivery

Logging & Tracing

Modernization & Migration

Monitoring

Networking

OpenShift Optional

Security

Storage

Streaming & Messaging

Other

Source

Red Hat (2)

Certified (2)

All Items

service me

Certified

Citrix ADC CPX Istio Sidecar Injector Operator

provided by citrix

In Istio service mesh, a sidecar proxy runs alongside application pods and it intercepts and...

Community

Maistra Service Mesh

provided by Red Hat, Inc.

The Maistra Operator enables you to install, configure, and manage an instance of Maistra service...

Red Hat OpenShift Service Mesh

2.2.2-0 provided by Red Hat, Inc.

Install

Latest version

2.2.2-0

Capability level

- Basic Install
- Seamless Upgrades
- Full Lifecycle
- Deep Insights
- Auto Pilot

Source

Red Hat

Provider

Red Hat, Inc.

Infrastructure features

Disconnected

Valid Subscriptions

OpenShift Container Platform
OpenShift Platform Plus

Repository

https://github.com/maistra/istio-operator

Red Hat OpenShift Service Mesh is a platform that provides behavioral insight and operational control over a service mesh, providing a uniform way to connect, secure, and monitor microservice applications.

Overview

Red Hat OpenShift Service Mesh, based on the open source istio project, adds a transparent layer on existing distributed applications without requiring any changes to the service code. You add Red Hat OpenShift Service Mesh support to services by deploying a special sidecar proxy throughout your environment that intercepts all network communication between microservices. You configure and manage the service mesh using the control plane features.

Red Hat OpenShift Service Mesh provides an easy way to create a network of deployed services that provides discovery, load balancing, service-to-service authentication, failure recovery, metrics, and monitoring. A service mesh also provides more complex operational functionality, including A/B testing, canary releases, rate limiting, access control, and end-to-end authentication.

Core Capabilities

Red Hat OpenShift Service Mesh supports uniform application of a number of key capabilities across a network of services:

- **Traffic Management** - Control the flow of traffic and API calls between services, make calls more reliable, and make the network more robust in the face of adverse conditions.
- **Service Identity and Security** - Provide services in the mesh with a verifiable identity and provide the ability to protect service traffic as it flows over networks of varying degrees of trustworthiness.
- **Policy Enforcement** - Apply organizational policy to the interaction between services, ensure access policies are enforced and resources are fairly distributed among consumers. Policy changes are made by configuring the mesh, not by changing application code.
- **Telemetry** - Gain understanding of the dependencies between services and the nature and flow of traffic between them, providing the ability to quickly identify issues.

The sample apps

code ~ sources

main.py x | handler.ts | App.java

```python
import os
import httpx

port = int(os.environ.get("PORT", default=8080)
service_b_host = os.environ.get("SERVICE_B_HOST"


async def app(scope, receive, send):
    assert scope["type"] == "http"

    match scope["path"]:
        case "/":
            text, status = await index(scope["he
        case "/health":
            text, status = await health()
        case _:
            text, status = await page_not_found(
```

bash ~ sources

The sample apps

code ~ sources

main.py | handler.ts x | App.java

```
1  async function handler(request: Request): Promis
2      switch (new URL(request.url).pathname) {
3          case "/":
4              return await index(request.headers);
5
6          case "/health":
7              return health();
8
9          default:
10             return notFound();
11     }
12 }
13
14 async function index(req_headers: Headers): Pror
15     const headers = extractTracingHeaders(req_he
16     const service_c_host = Deno.env.get("SERVICI
17
```

bash ~ sources

The sample apps

code ~ sources

main.py handler.ts App.java x

```
1  package com.lemberski.servicec;
2
3  import static io.javalin.http.HttpCode.INTERNAL_
4  import static java.lang.String.format;
5
6  import io.javalin.Javalin;
7
8  public final class App {
9
10     private boolean crash = false;
11     private int count = 0;
12
13     public static void main(String[] args) {
14         Environment env = new Environment();
15         new App().startServer(env);
16     }
17
```

bash ~ sources

Hands on

https://red.ht/opentour-wien-lab

github.com/sa-mw-dach/opentour-2022-servicemesh

Sample apps deployment

code ~ sources/kubernetes

a-deploy.yml x | b-deploy.yml | c-v1-deploy.yml | ...

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: service-a
  name: service-a
spec:
  replicas: 1
  selector:
    matchLabels:
      app: service-a
  template:
    metadata:
      labels:
        app: service-a
        version: v1
      annotations:
```

bash ~ sources

Control Plane and Member Roll

code ~ sources/kubernetes

controlplane.yml x memberroll.yml

```yaml
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  tracing:
    type: Jaeger
    sampling: 1000
  policy:
    type: Istiod
  addons:
    grafana:
      enabled: true
    jaeger:
      install:
        storage:
```

bash ~ sources

Control Plane and Member Roll

code ~ sources/kubernetes

controlplane.yml | memberroll.yml x

```yaml
kind: ServiceMeshMemberRoll
apiVersion: maistra.io/v1
metadata:
  name: default
  namespace: istio-system
spec:
  members:
    - servicemesh-apps
```

bash ~ sources

Exposing our apps

code ~ sources/kubernetes

gateway.yml x

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: services-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
```

bash ~ sources

Use Case: Canary Releases

code ~ sources/kubernetes

destination-rules.yml x | c-v1-deploy.yml | c-v2-deploy…

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: service-c
spec:
  host: service-c
  subsets:
  - name: version-v1
    labels:
      version: v1
  - name: version-v2
    labels:
      version: v2
```

bash ~ sources

Use Case: Canary Releases

code ~ sources/kubernetes/canary

1-vs-v1.yml x | 2-vs-v1_and_v2_90_10.yml | 3-vs-v1_and_v2_50_50.yml | 4-vs-v2.yml

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: service-c
spec:
  hosts:
  - service-c
  http:
  - route:
    - destination:
        host: service-c
        subset: version-v1
      weight: 100
```

Use Case: Canary Releases

code ~ sources/kubernetes/canary

1-vs-v1.yml | 2-vs-v1_and_v2_90_10.yml x | 3-vs-v1_and_v2_50_50.yml | 4-vs-v2.yml

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: service-c
spec:
  hosts:
  - service-c
  http:
  - route:
    - destination:
        host: service-c
        subset: version-v1
      weight: 90
    - destination:
        host: service-c
        subset: version-v2
      weight: 10
```

Use Case: Canary Releases

code ~ sources/kubernetes/canary

1-vs-v1.yml | 2-vs-v1_and_v2_90_10.yml | 3-vs-v1_and_v2_50_50.yml x | 4-vs-v2.yml

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: service-c
spec:
  hosts:
  - service-c
  http:
  - route:
    - destination:
        host: service-c
        subset: version-v1
      weight: 50
    - destination:
        host: service-c
        subset: version-v2
      weight: 50
```

Use Case: Canary Releases

code ~ sources/kubernetes/canary

1-vs-v1.yml | 2-vs-v1_and_v2_90_10.yml | 3-vs-v1_and_v2_50_50.yml | 4-vs-v2.yml x

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: service-c
spec:
  hosts:
  - service-c
  http:
  - route:
    - destination:
        host: service-c
        subset: version-v2
      weight: 100
```

Use Case: Circuit Breaker & Retry

code ~ sources/kubernetes/circuit-breaker

1-vs.yml x | 2-destination-rules.yml | 3-vs-retry.yml

```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: service-c
spec:
  hosts:
  - service-c
  http:
  - route:
    - destination:
        host: service-c
```

Use Case: Circuit Breaker & Retry

code ~ sources/kubernetes/circuit-breaker

1-vs.yml | 2-destination-rules.yml x | 3-vs-retry.yml

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: service-c
spec:
  host: service-c
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 100
      http:
        maxConnections: 100
    outlierDetection:
      consecutive5xxErrors: 1
      interval: 2s
      baseEjectionTime: 10s
      maxEjectionPercent: 100
```

Use Case: Circuit Breaker & Retry

code ~ sources/kubernetes/circuit-breaker

1-vs.yml | 2-destination-rules.yml | 3-vs-retry.yml x

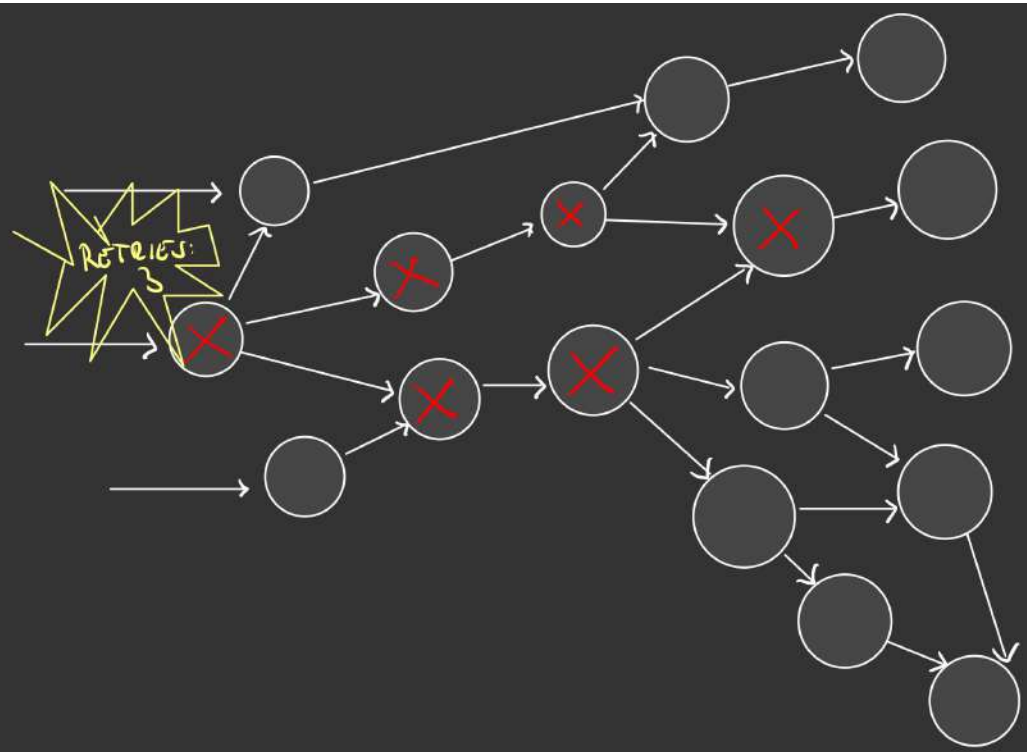
```yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: service-c
spec:
  hosts:
  - service-c
  http:
  - route:
    - destination:
        host: service-c
    retries:
        attempts: 3
        perTryTimeout: 1s
        retryOn: 5xx
```

How to start?

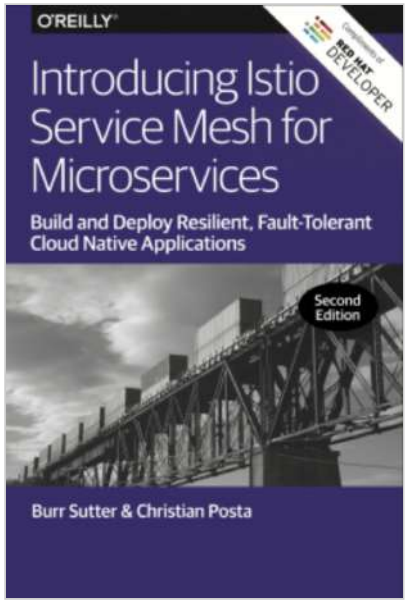
Build a team

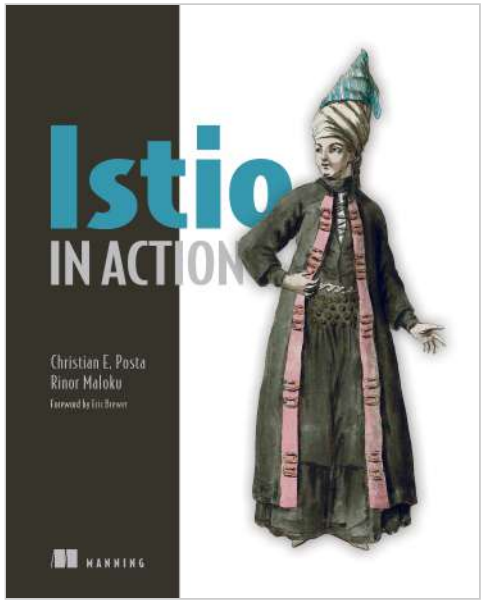
Tiny steps

Warning: Zombies ahead!



Books


O'REILLY
Introducing Istio Service Mesh for Microservices
Build and Deploy Resilient, Fault-Tolerant Cloud Native Applications
Second Edition
Burr Sutter & Christian Posta


Istio
IN ACTION
Christian E. Posta
Rinor Maloku
MANNING

Red Hat

Red Hat Developer

Join Red Hat Developer.

Build here. Go anywhere.

youtube.com/RedHatDevelopers

linkedin.com/showcase/red-hat-developer

facebook.com/RedHatDeveloperProgram

twitter.com/rhdevelopers