Wie bändige ich meine Microservices?

Istio - Eine offene Plattform zum Verbinden, Verwalten und Sichern von Microservices

JBFOne 2018, Andreas Neeb

Innovate. Thinking.

redhat

FIDUCIA GAD
ZUKUNFTSERFAHREN

whoami

Andreas Neeb,

Chief Architect Financial Services

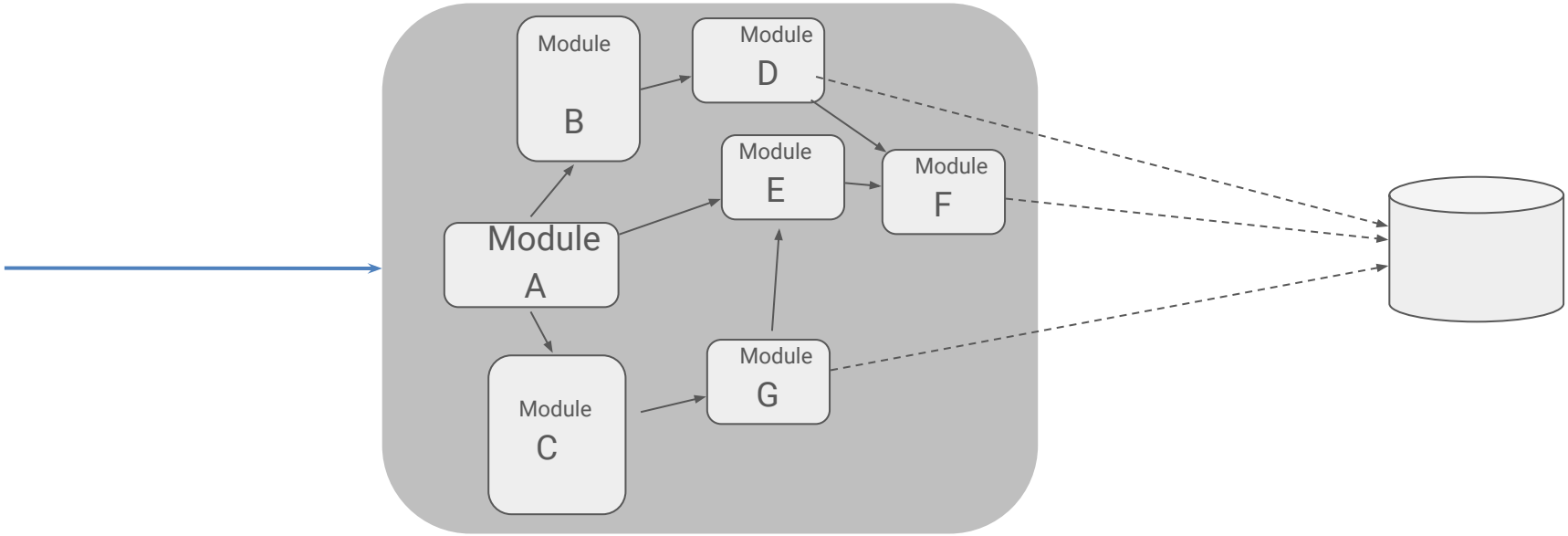
Red Hat GmbH, aneeb@redhat.com

- Strategist
- Container Guy
- Vater & Ehemann
- Nerd
- Fußball Fan

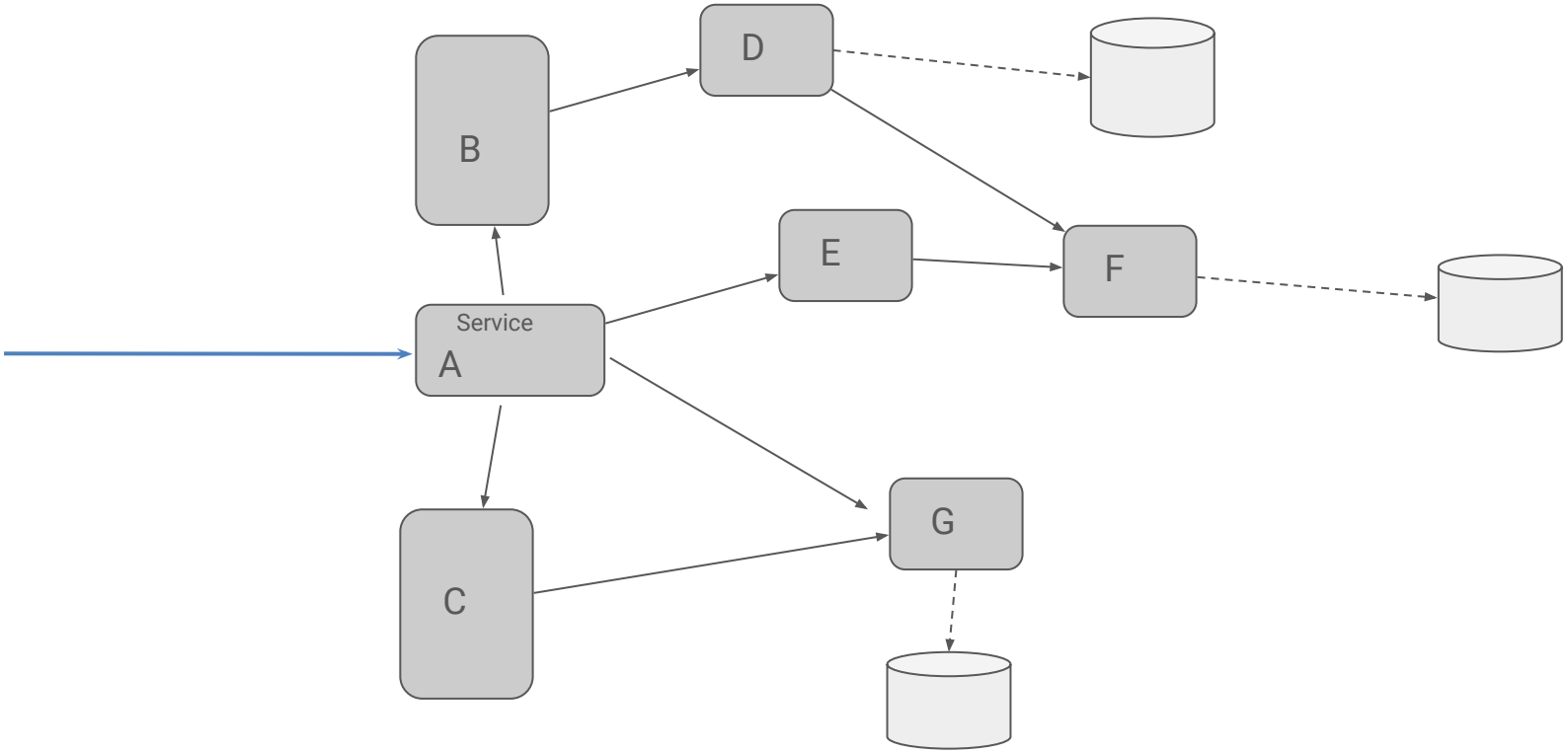
Nicht immer in dieser Reihenfolge


A
B

Es war einmal, vor langer Zeit … der Monolith



Microservices!



Microservices!

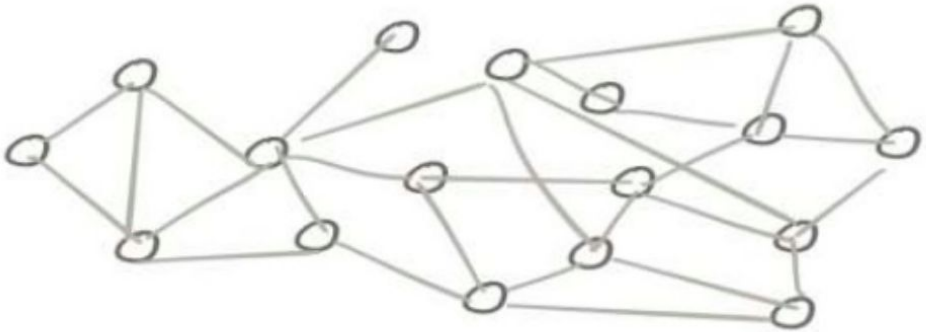
- ★ Agilität
- ★ Abstraktion
- ★ Skalierbarkeit

Problem gelöst!

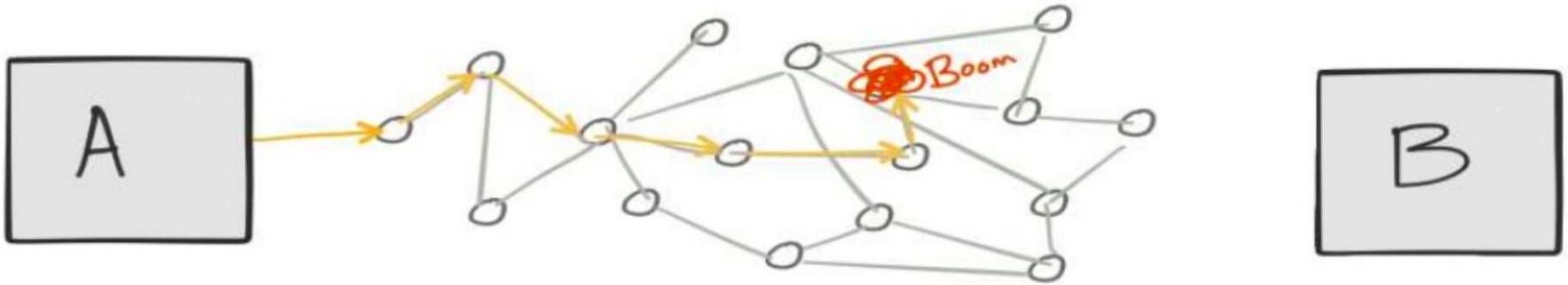

Die verstärkte Nutzung Service orientierter
Architekturen verschiebt die Komplexität in
den Raum zwischen den Services.

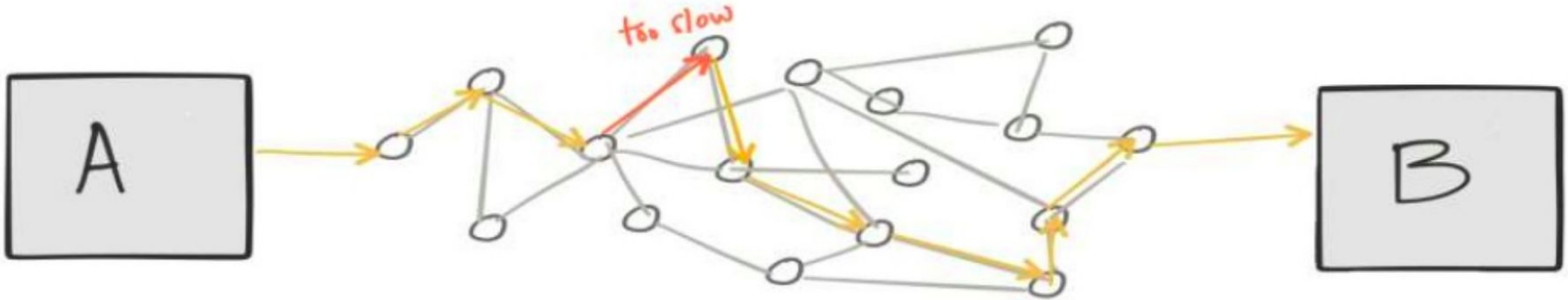

A
B

A



B


A
Boom
B


too slow
A
B

Jede Applikation muss Sorge tragen für ...

Service discovery

Retries

Timeouts

Load balancing

Rate limiting

Thread bulk heading

Circuit breaking

Routing (zone-aware)

Deadlines

Back pressure

Outlier detection

Health checking

Traffic shaping

Request shadowing

kubernetes

Kubernetes ist eine Deployment Platform

Kubernetes adressiert Service Kommunikation nicht ausreichend

Es existieren ganze Frameworks um Entwicklern diese Arbeit zu erleichtern

Netflix Hystrix (circuit breaking / bulk heading)

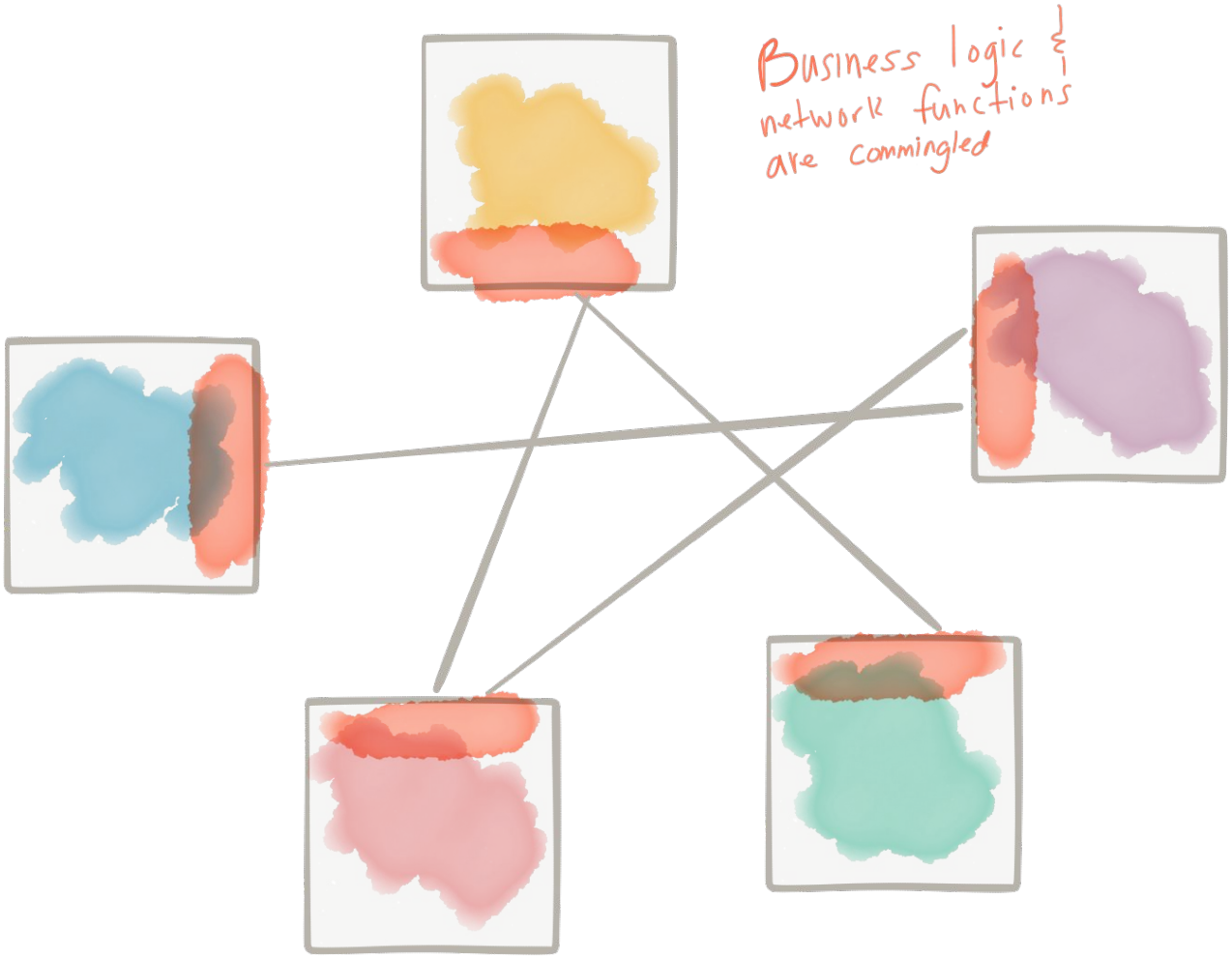
Netflix Zuul (edge router)

Netflix Ribbon (client-side service discovery / load balance)

Netflix Eureka (service discovery registry)

Brave / Zipkin (tracing)

Netflix spectator / atlas (metrics)


Business logic &
network functions
are commingled

Aber ich nutze …

Spring

Vert.x

NodeJS

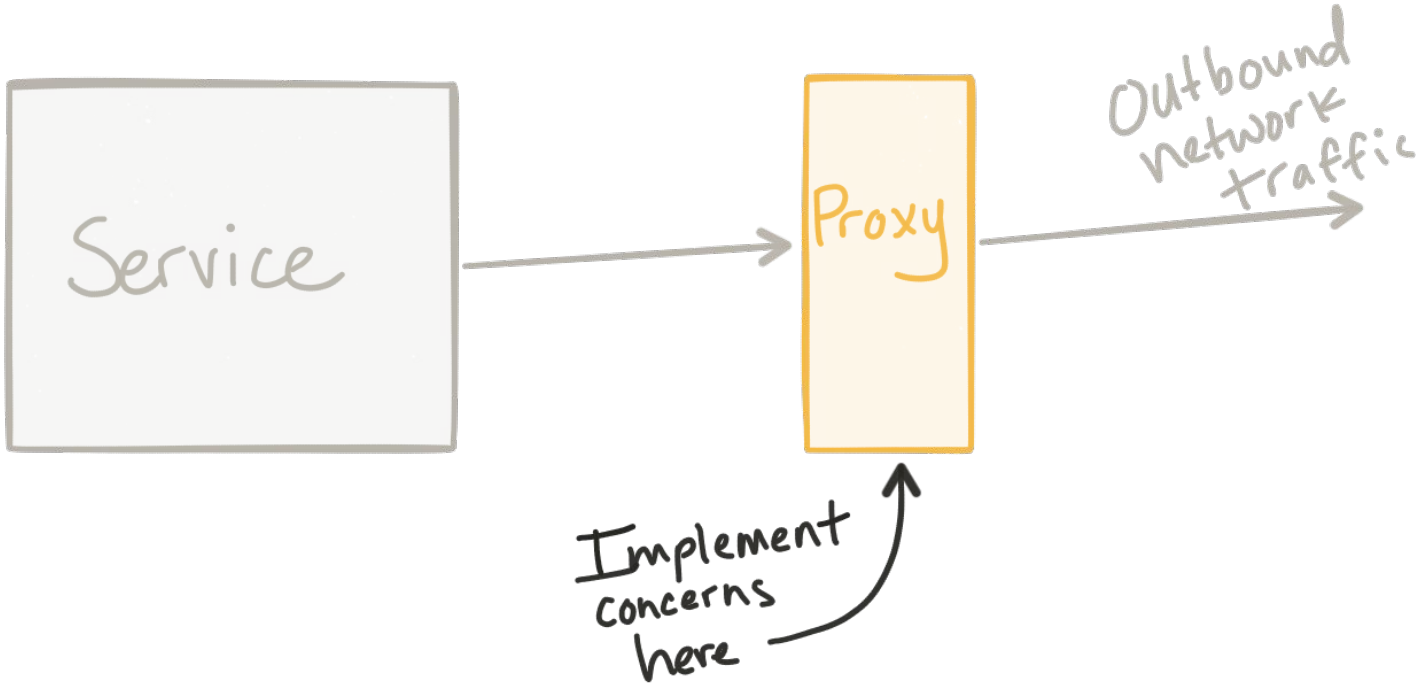
Go

Python

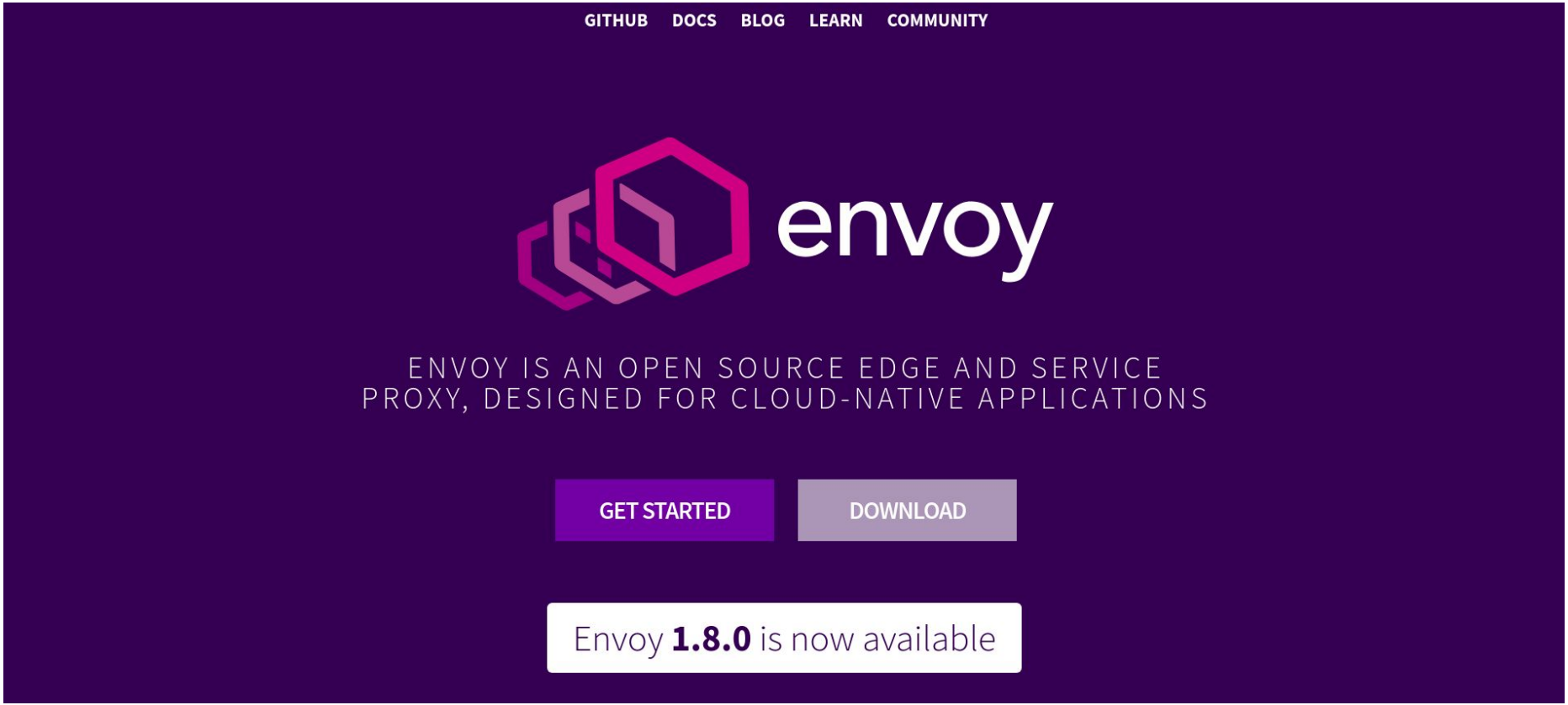
Ruby

Perl

Wie wäre es damit?



Hallo Envoy!


GITHUB DOCS BLOG LEARN COMMUNITY
envoy
ENVOY IS AN OPEN SOURCE EDGE AND SERVICE PROXY, DESIGNED FOR CLOUD-NATIVE APPLICATIONS
GET STARTED
DOWNLOAD
Envoy 1.8.0 is now available

Was ist Envoy?

service proxy

C++, highly parallel, non-blocking

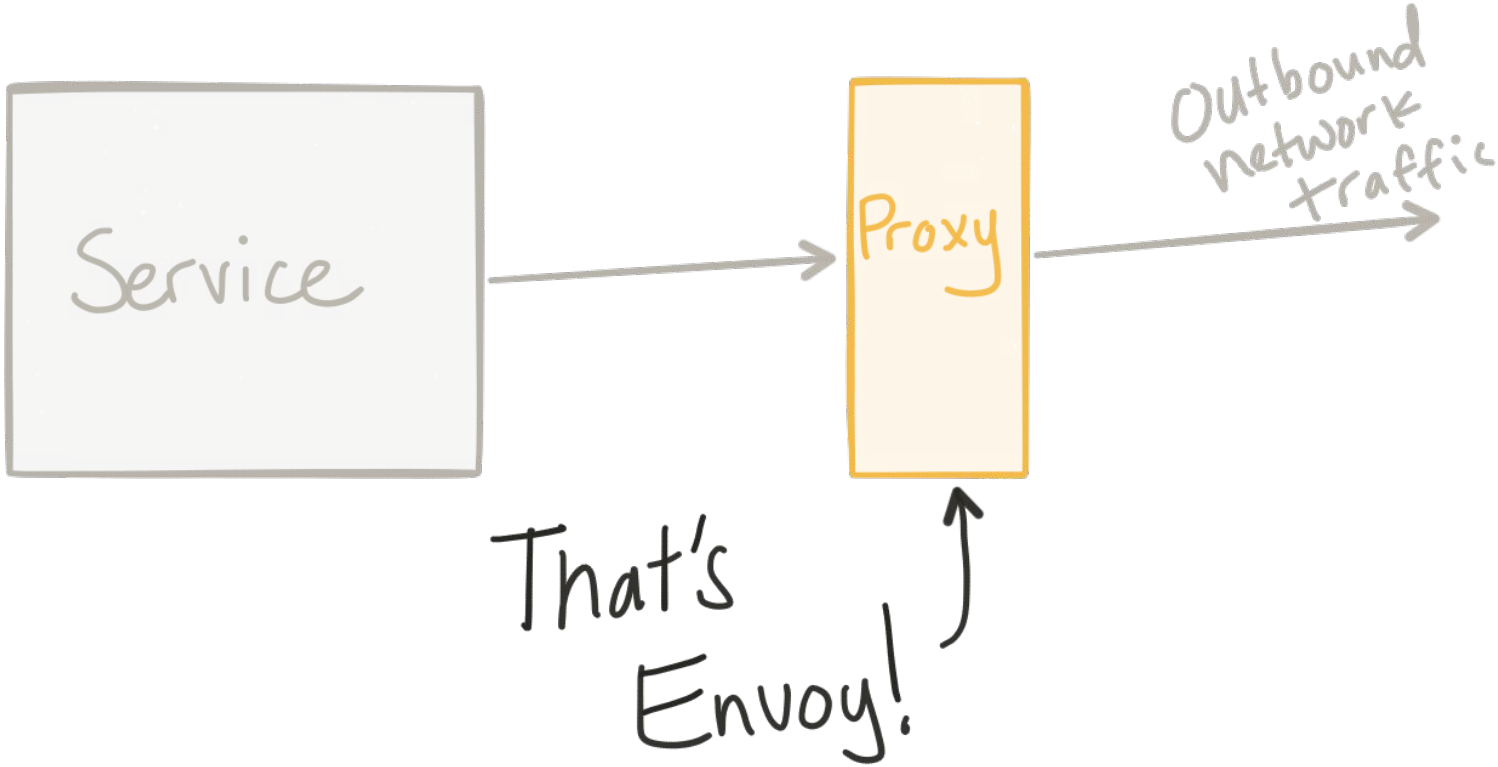
L3/4 network filter, out of the box L7 filters

HTTP 2, including gRPC

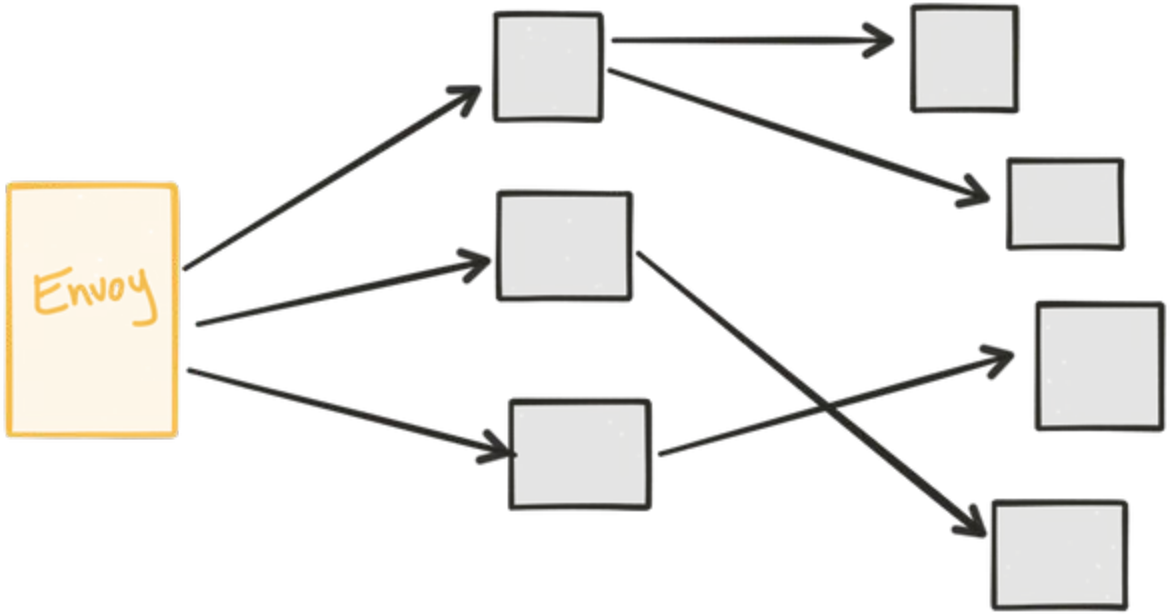
service discovery/health checking

advanced load balancing

stats, metrics, tracing

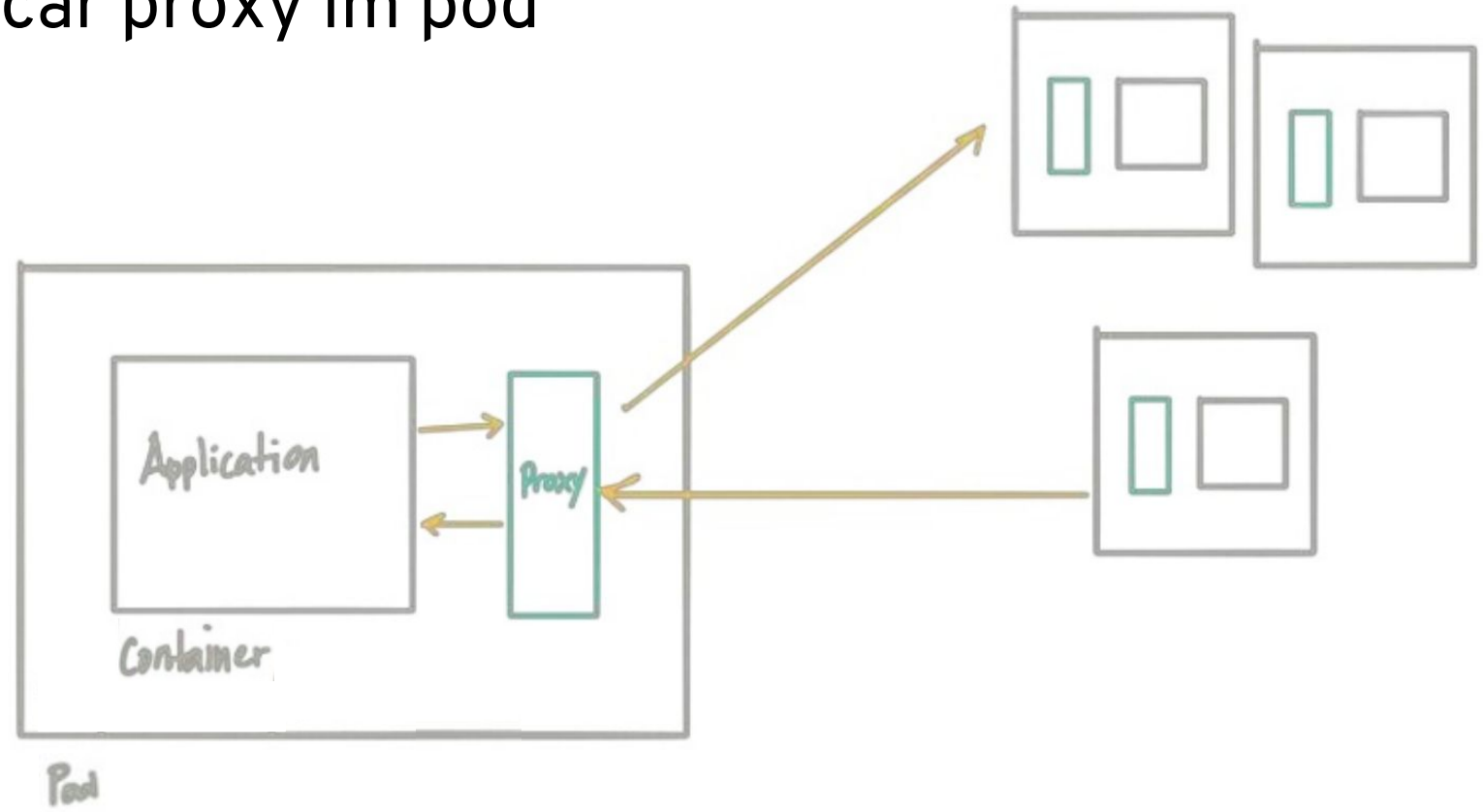

Service
Proxy
Outbound network traffic
That's Envoy!

Envoy als edge proxy

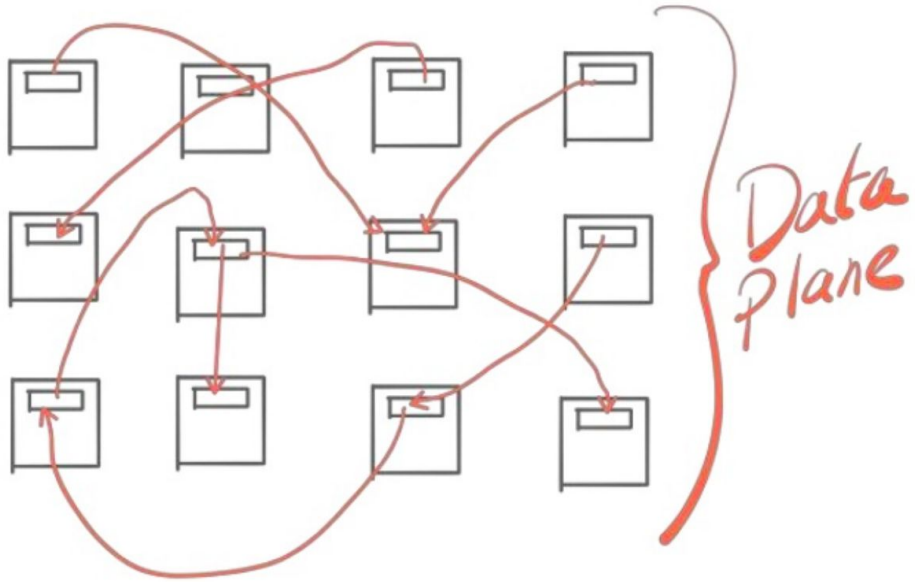


Envoy als sidecar proxy

Sidecar proxy im pod



Jegliche Kommunikation zwischen Applikationen fließt durch diese Proxy. Dies ist die so genannte “Data Plane”



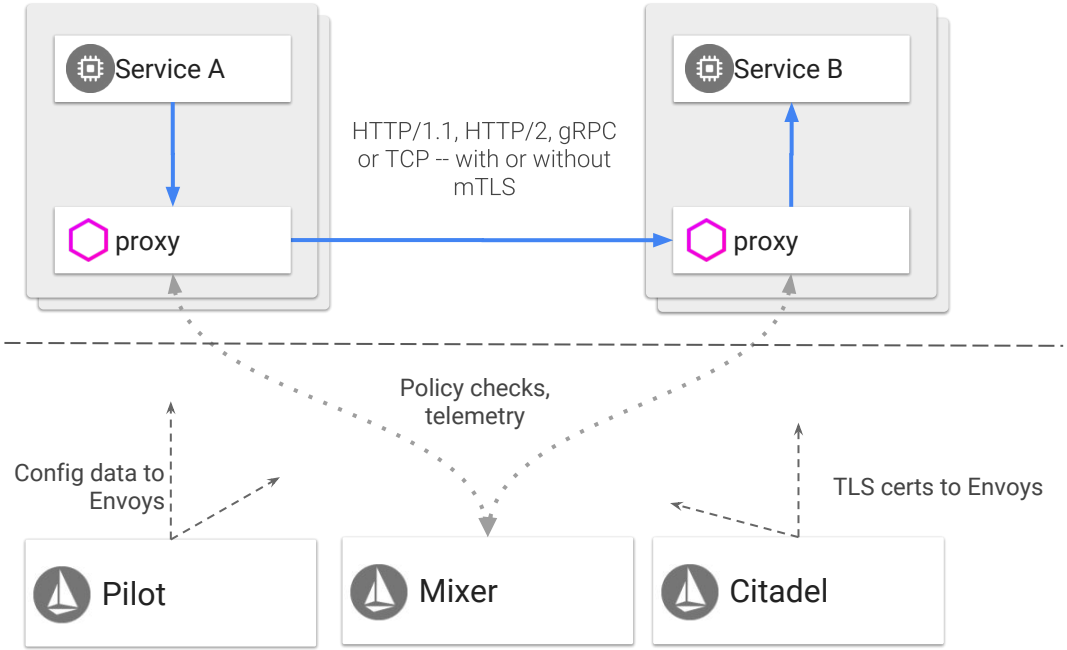
Wie konfiguriere & verwalte ich eine große Anzahl Envoy Proxy?


istio.io
service mesh

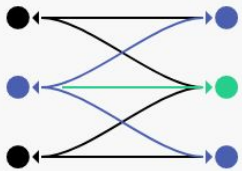
Istio Architektur

Data Plane

Control Plane



Istio Service Mesh -- Connect

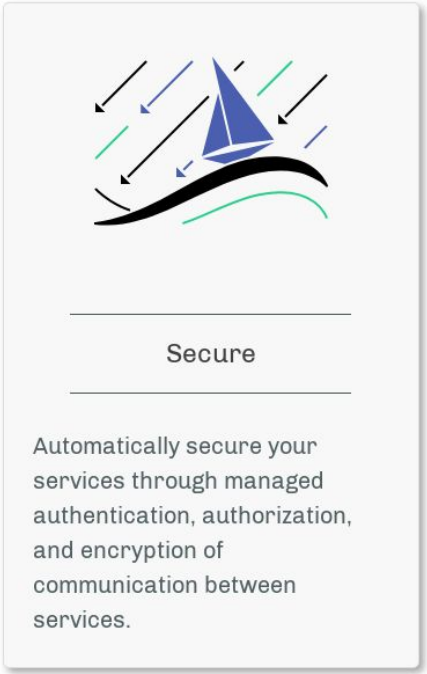


Connect

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

- Request routing
- Timeouts
- Retries
- Circuit breaking
- Fault injection
- Traffic shifting
- Mirroring

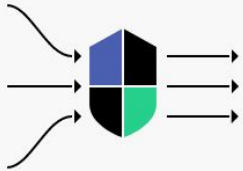
Istio Service Mesh -- Secure



- Identity
- PKI
- Mutual TLS authentication
- Authorization

Istio Service Mesh -- Control

- Rate limits
- Denials
- White & blacklisting

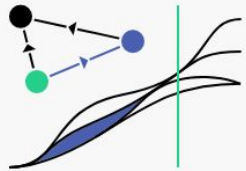


Control

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

Istio Service Mesh -- Observe

- Service graph
- Metrics
- Logging
- Distributed tracing



Observe

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

live demo

Setup

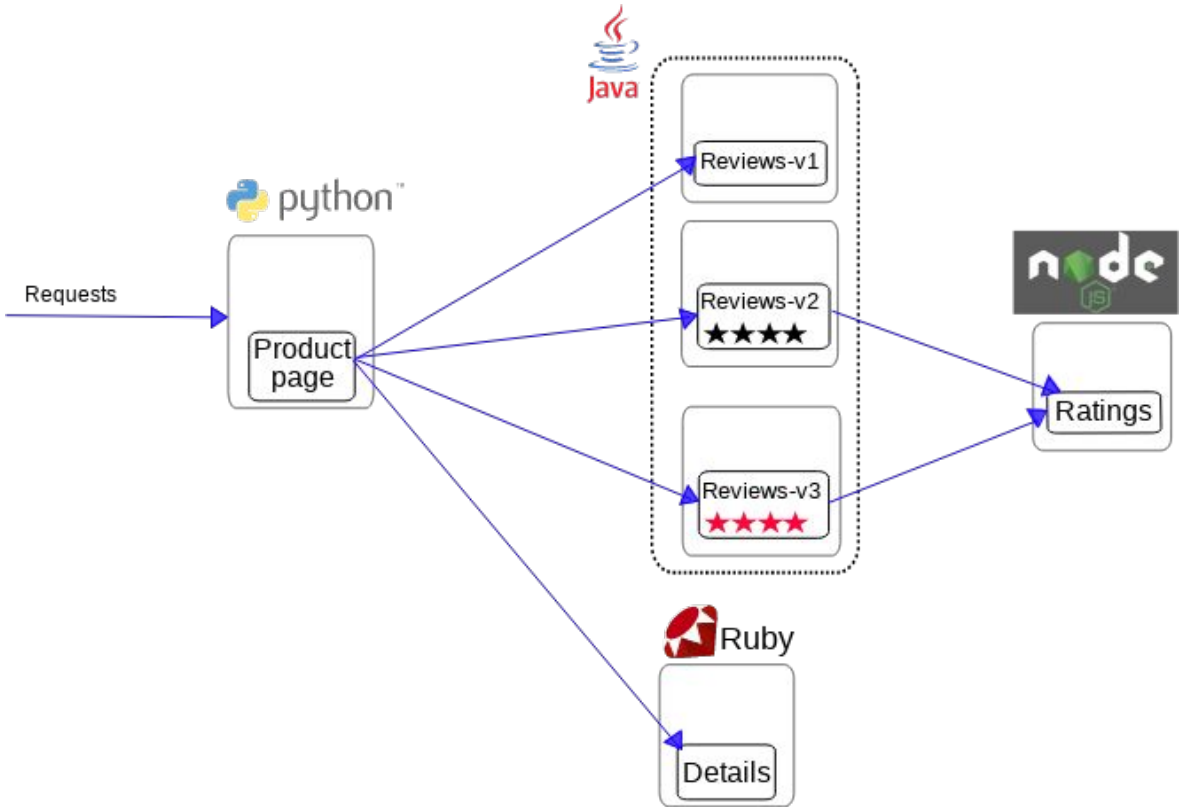
- OpenShift 3.11 (Kubernetes 1.11)
- OpenShift Service Mesh 0.3 (Istio + Kiali + Prometheus + Grafana + Jaeger)
- Bookinfo

Selber testen?

https://learn.openshift.com

https://github.com/redhat-developer-demos/istio-tutorial

bookinfo



https://bit.ly/2zk7czt

Use Cases

Request routing

(all) -> reviews v1
(user: jason) -> reviews v2

Fault injection

(user: jason) -> reviews v2 - (delay) -> ratings

Traffic Shifting

(all) - (canary) -> reviews v3

USE CASE 1 - Request routing

Use Case 1

Request routing

(all) -> reviews v1
(user: jason) -> reviews v2

Send all traffic to v1

oc apply -f virtual-service-all-v1.yaml

oc get virtualservice reviews -o yaml

```
metadata:
  name: reviews

spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
```

oc get destinationrule reviews -o yaml

```
metadata:
  name: reviews

spec:
  host: reviews
  subsets:
  - labels:
      version: v1
    name: v1
  - labels:
      version: v2
    name: v2
  - labels:
      version: v3
    name: v3
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
```

```
oc get pods --show-labels | grep reviews

reviews-v1-8568fdcf99-28hw9     2/2     Running     0       49m     version=v1
reviews-v2-8596f84c5-6bhmz      2/2     Running     0       49m     version=v2
reviews-v3-56568b7595-6hhpg     2/2     Running     0       49m     version=v3
```

Send only user ‘Jason’ to v2

oc apply -f virtual-service-reviews-test-v2.yaml

oc get virtualservice reviews -o yaml

```
metadata:
  name: reviews

spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v1
```

USE CASE 2 - Fault injection

Use Case 2

Fault injection

(user: jason) -> reviews v2 - (delay) -> ratings

Inject delay for user ‘Jason’

oc apply -f virtual-service-ratings-test-delay.yaml

oc get virtualservice ratings -o yaml

```
metadata:
  name: ratings

spec:
  hosts:
  - ratings
  http:
  - fault:
      delay:
        fixedDelay: 7s
        percent: 100
    match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: ratings
        subset: v1

...
```

Understand what happend

The timeout between the productpage and the reviews service is 6 seconds - coded as 3s + 1 retry for 6s total. The timeout between the reviews and ratings service is hard-coded at 10 seconds. Because of the delay we introduced, the /productpage times out prematurely and throws the error.

USE CASE 3 - Traffic shifting

Use Case 3

Traffic Shifting

(all) - (canary) -> reviews v3

Cleanup: all to v1

oc apply -f virtual-service-all-v1.yaml

Transfer 50% from reviews:v1 to reviews:v3

oc apply -f virtual-service-reviews-50-v3.yaml

oc get virtualservice reviews -o yaml

```
metadata:
  name: reviews

spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
      weight: 50
    - destination:
        host: reviews
        subset: v3
      weight: 50
```

Route 100% to reviews:v3

oc apply -f virtual-service-reviews-v3.yaml

redhat.

DANKE

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos