13.11.2019
Mannheim

[Container Conf] » Continuous Lifecycle »

Service Mesh -

Was die neue Infrastruktur für Microservices taugt

INNOQ

Hanna Prinz

Warum brauchen Microservices eine neue Infrastruktur?

Warum Microservices?

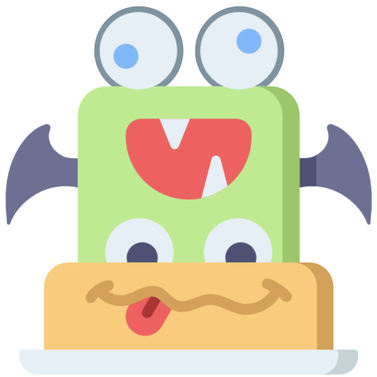
Microservices

+

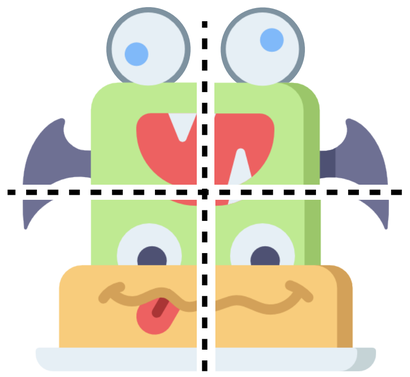
Kürzere Time-to-Market

Skalierbarkeit

Technologiefreiheit

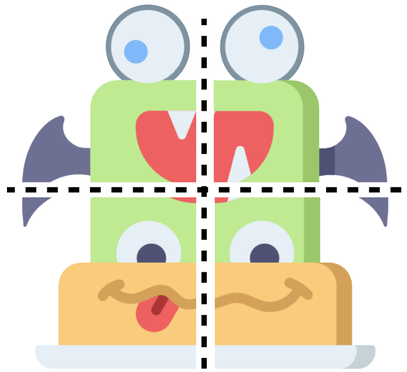


Monolith

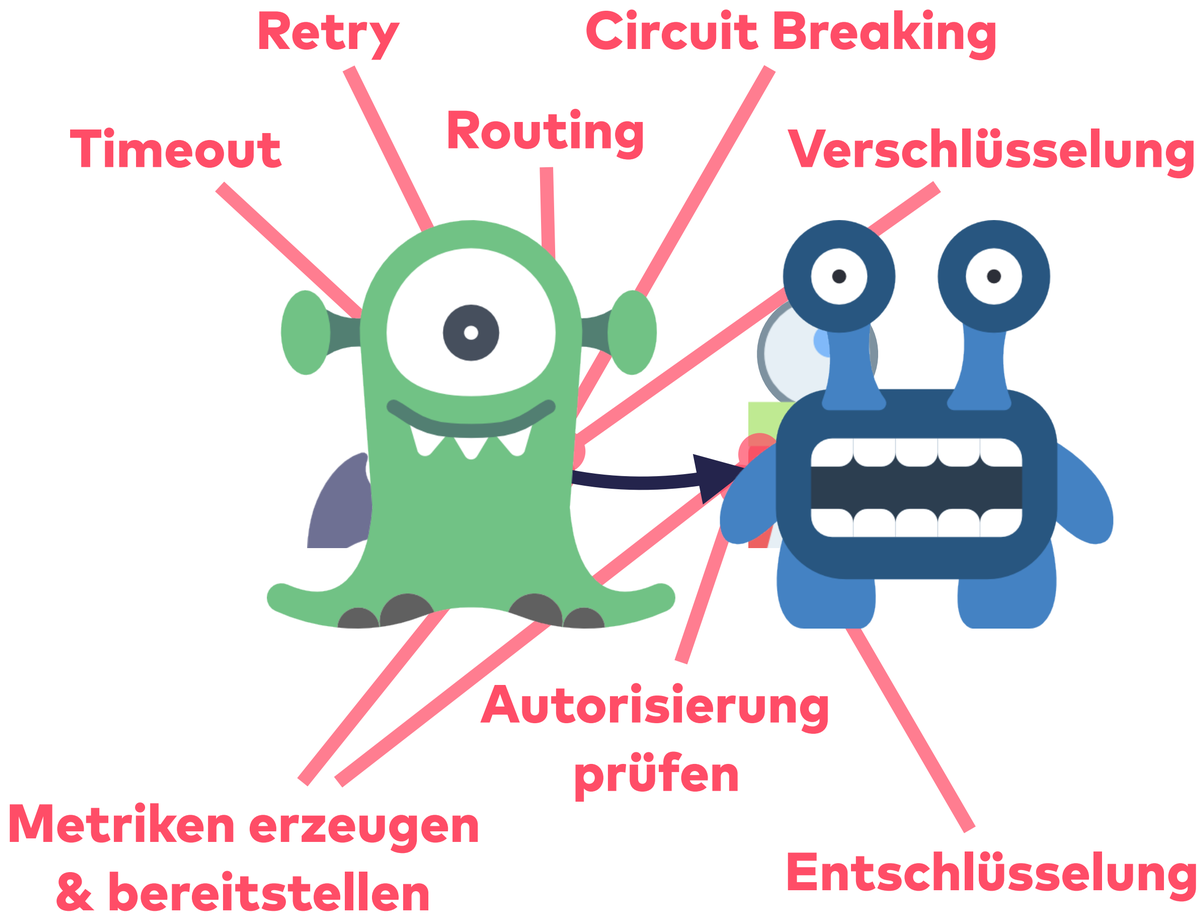


Microservices

Warum Service Meshes?

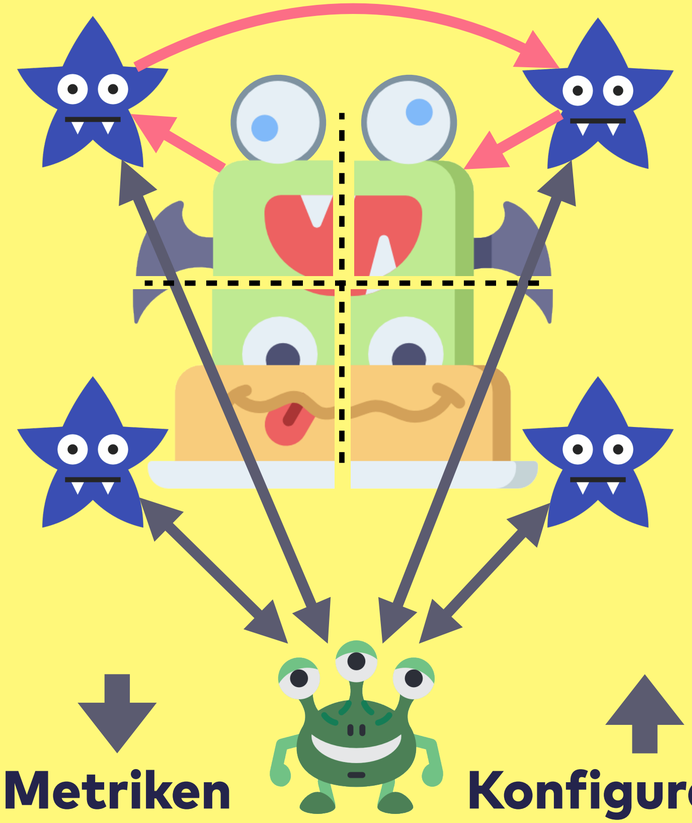


Microservices


Retry
Circuit Breaking
Timeout
Routing
Verschlüsselung
Autorisierung
prüfen
Metriken erzeugen
& bereitstellen
Entschlüsselung

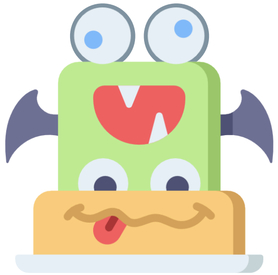
Service Mesh

Retry
Timeout
Circuit Breaker
Routing
Verschlüsseln
Entschlüsseln
Autorisierung
Metriken
...

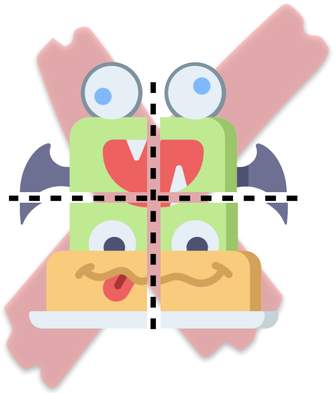


Service Mesh Architektur

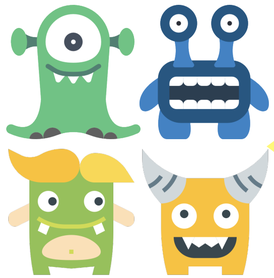
Monolith



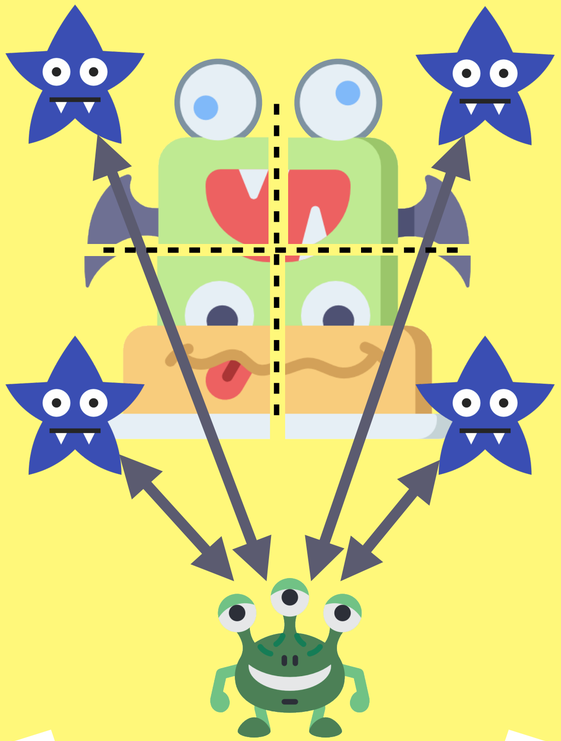
Microservices Theorie



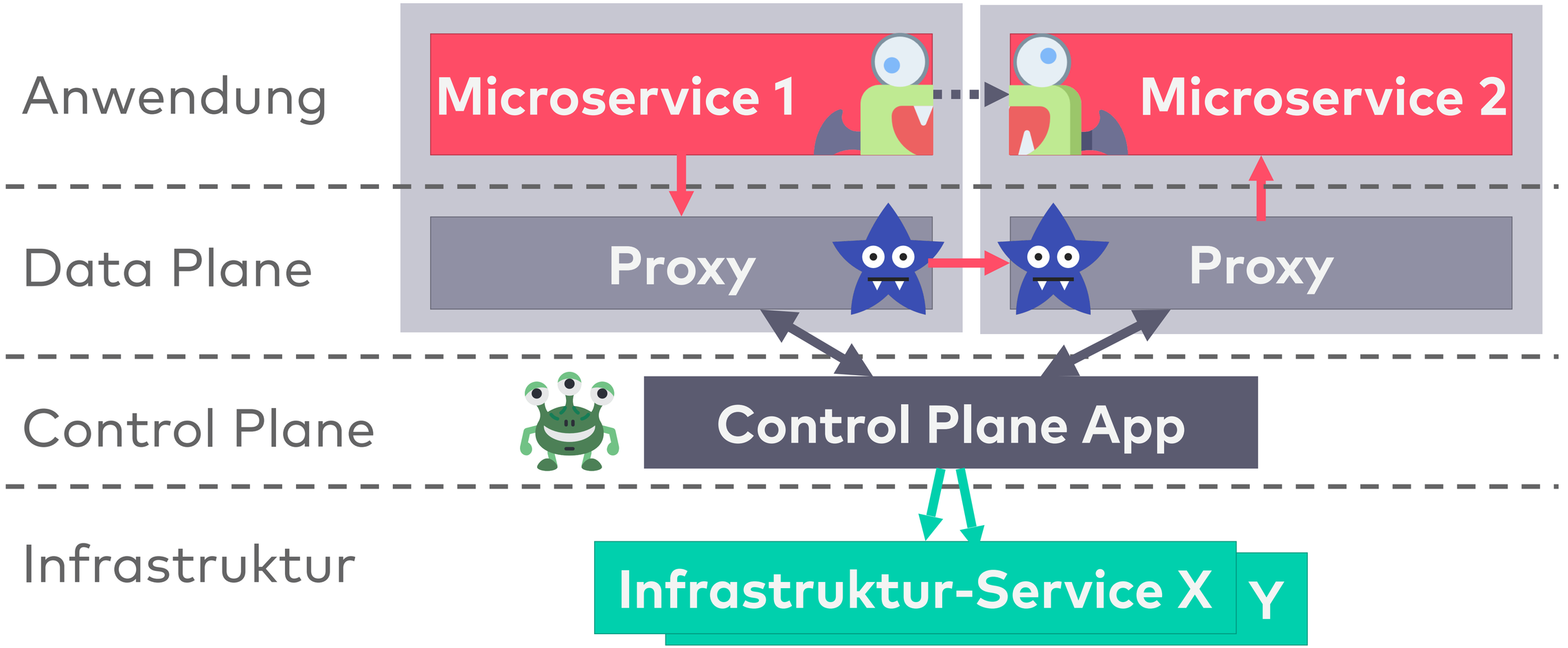
Microservices Praxis



Microservices mit Service Mesh



Service Mesh Architektur



Was kann ein Service Mesh?

Service Mesh Features

Observability

Routing

Resilience

Security

Beispiele mit dem Istio Service Mesh

Monitoring

Ein Service Mesh kann automatisch Daten für alle 4 "Golden Signals" liefern:

Latenz

Requests/Sekunde

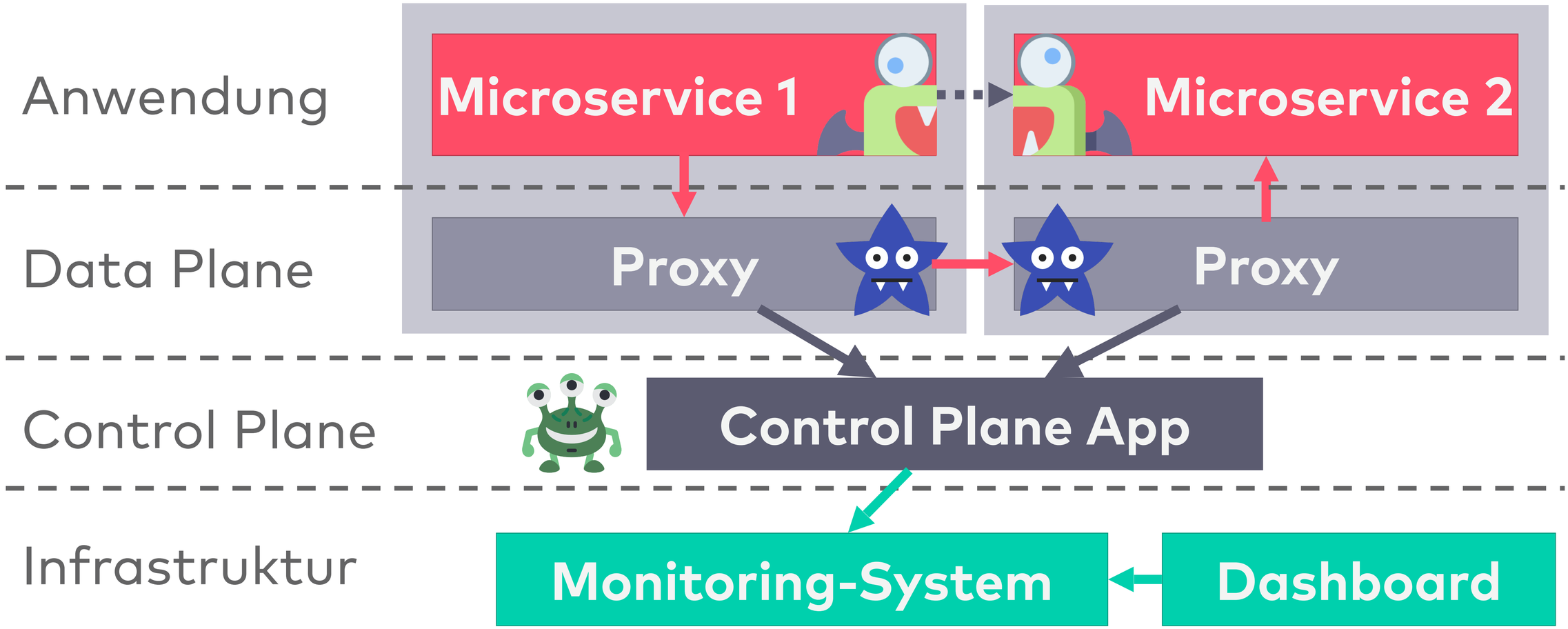
Status Codes (Fehler)

Auslastung

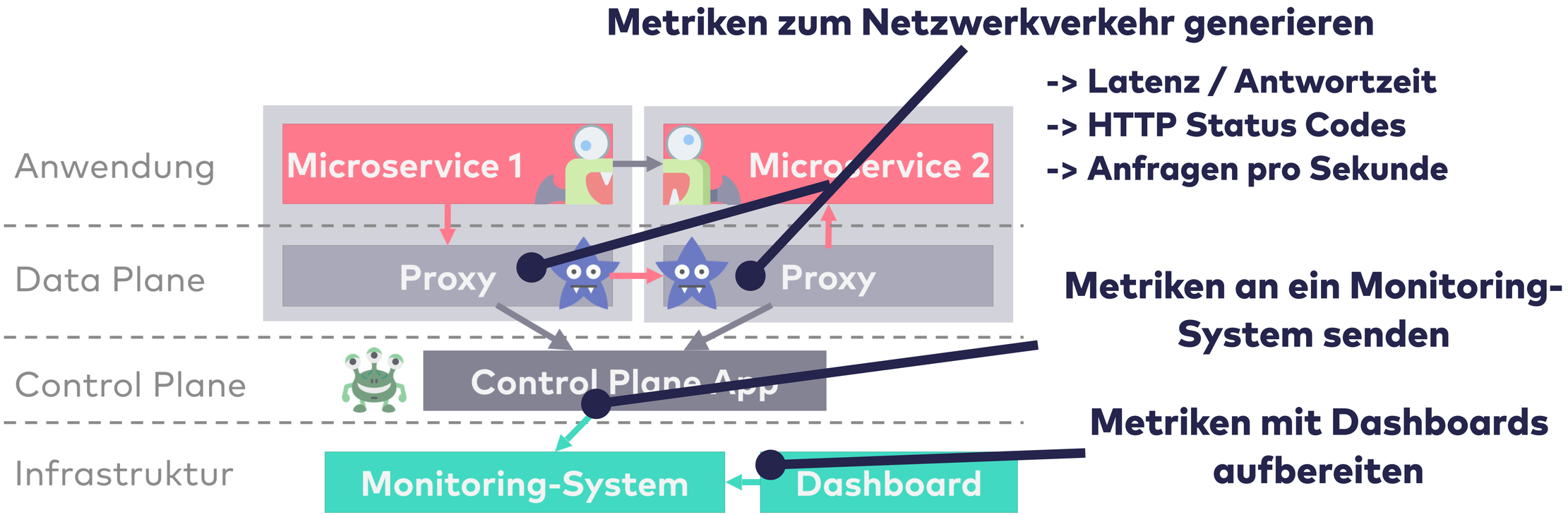
... es hat aber keinen Einblick in die Microservices

https://landing.google.com/sre/sre-book/chapters/monitoring-distributed-systems/#xref_monitoring_golden-signals

Monitoring mit Service Mesh



Monitoring mit Service Mesh



Monitoring mit Istio

Monitoring mit Istio

basiert auf...

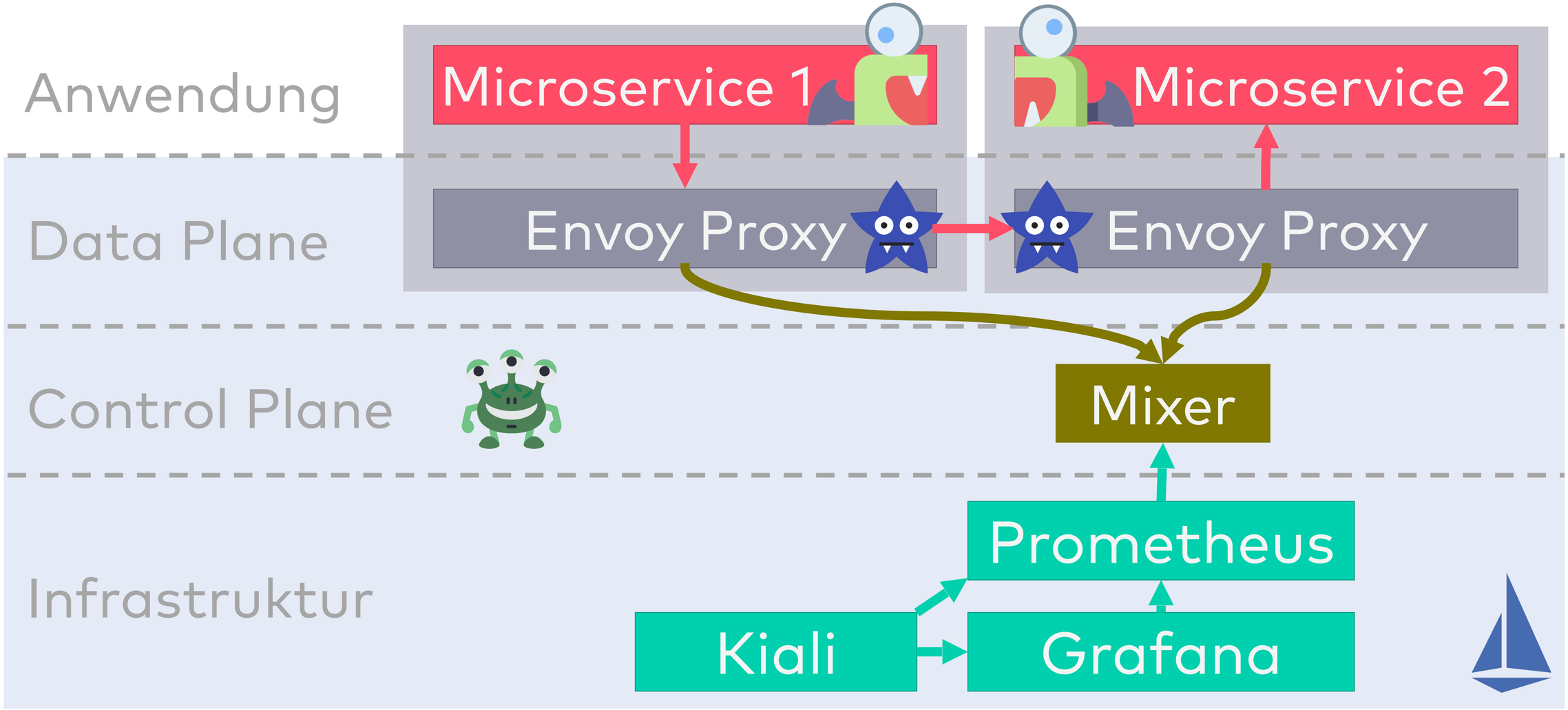
Prometheus 

Grafana 

Kiali 

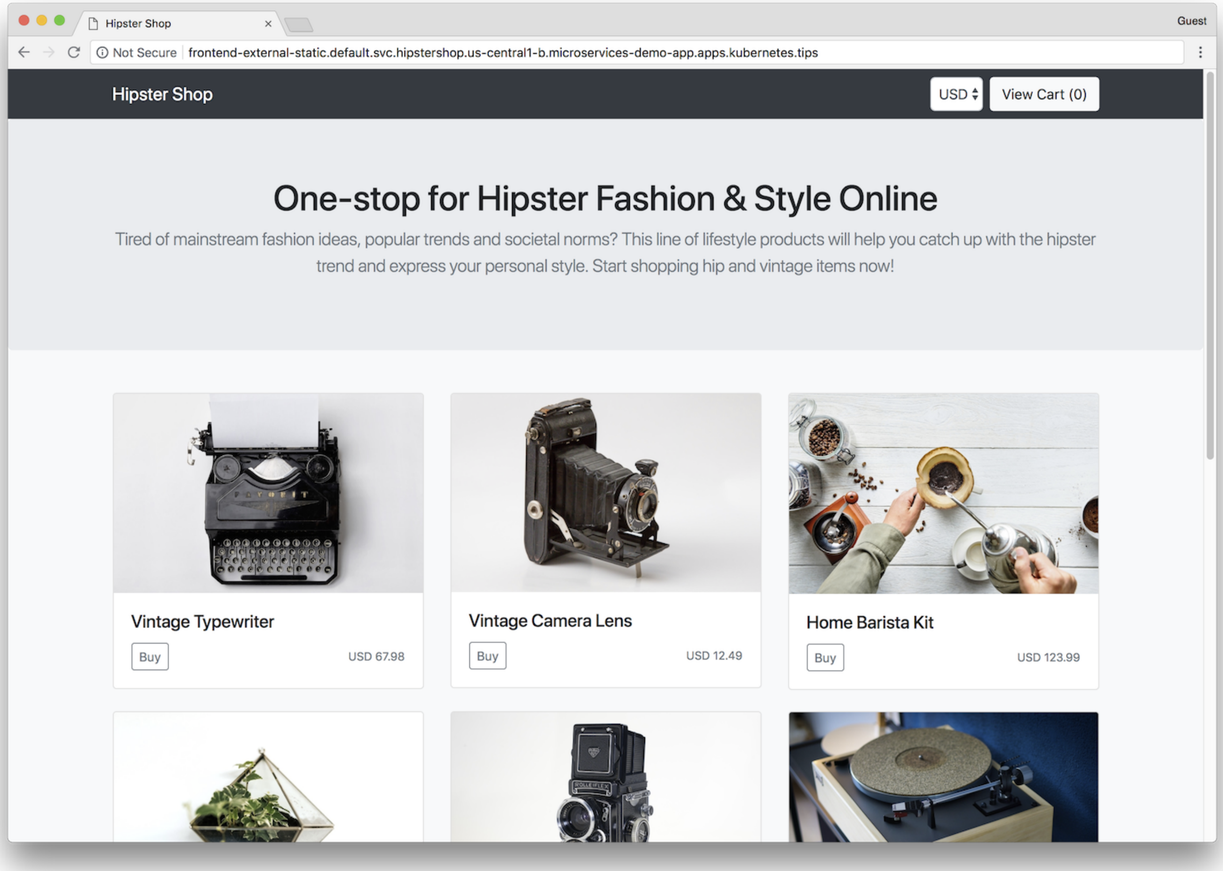
Jaeger 

Monitoring mit Istio

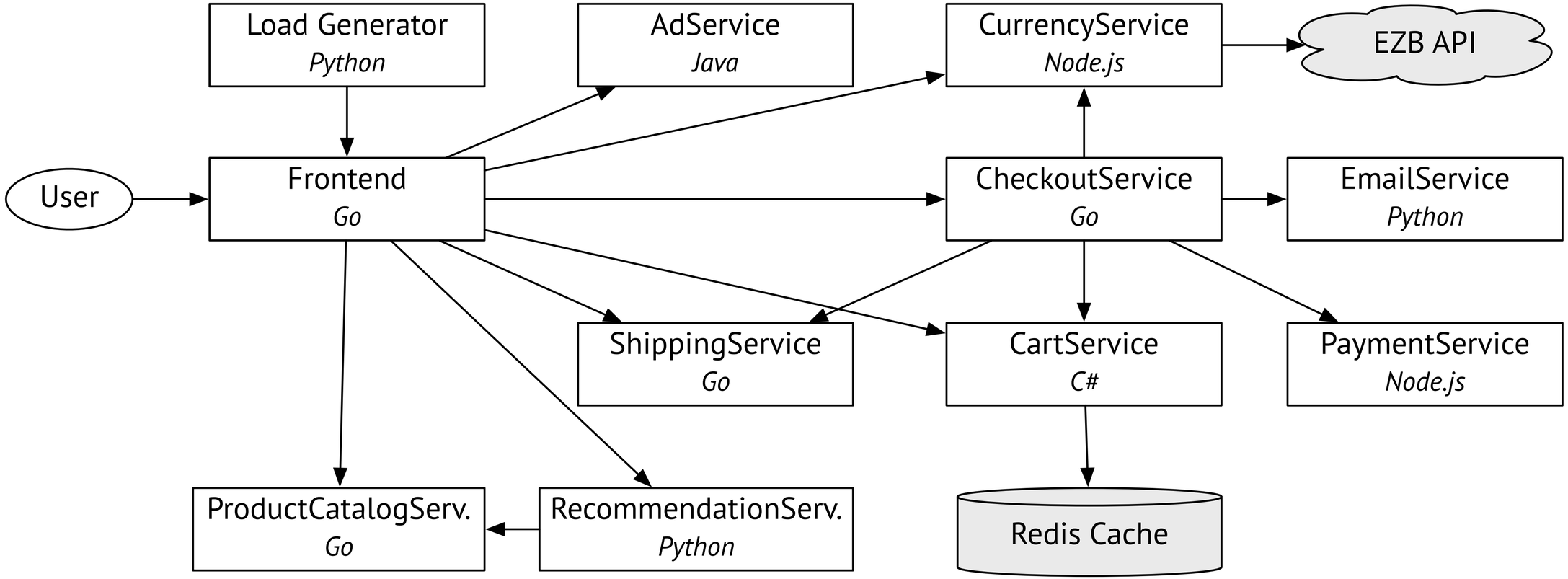


Beispielanwendung: Hipstershop

- Fachlich:
  - Online-Shop für Hipsterbedarf
- Technologisch:
  - **10 Microservices** mit verschiedenen Technologien
  - Aufruf einer **externen API** für Währungsumrechnungen
  - **Redis** für Warenkorb
  - Kommunikation über **gRPC**



Hipstershop Services



Demo

~/istio-demo (zsh)

```
(⎈ |istio:default)➜  istio-demo
```

~ (zsh)

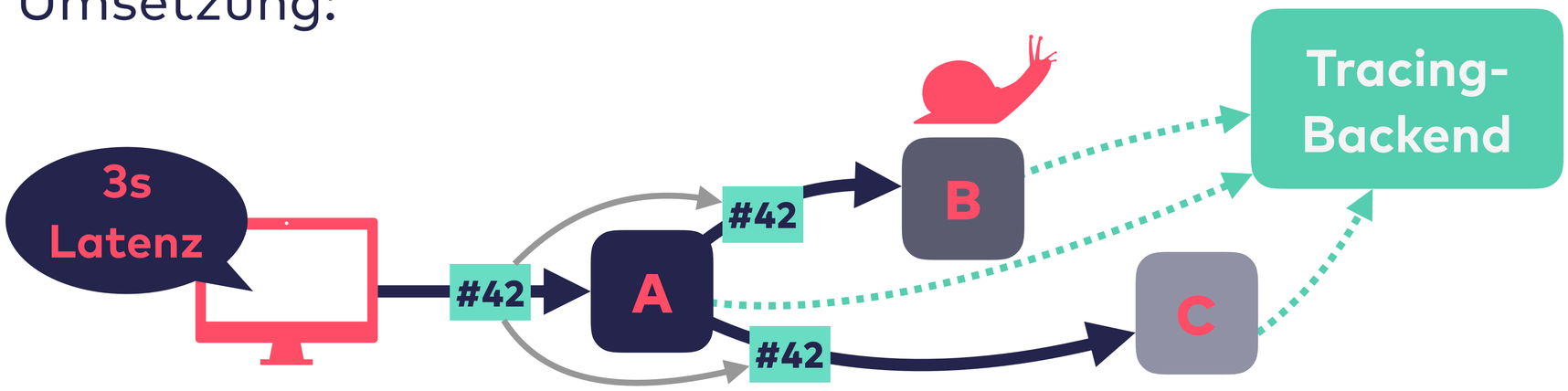
```
Last login: Wed Nov 13 00:27:15 on ttys028
(⎈ |istio:default)➜  ~
```

demo.sh

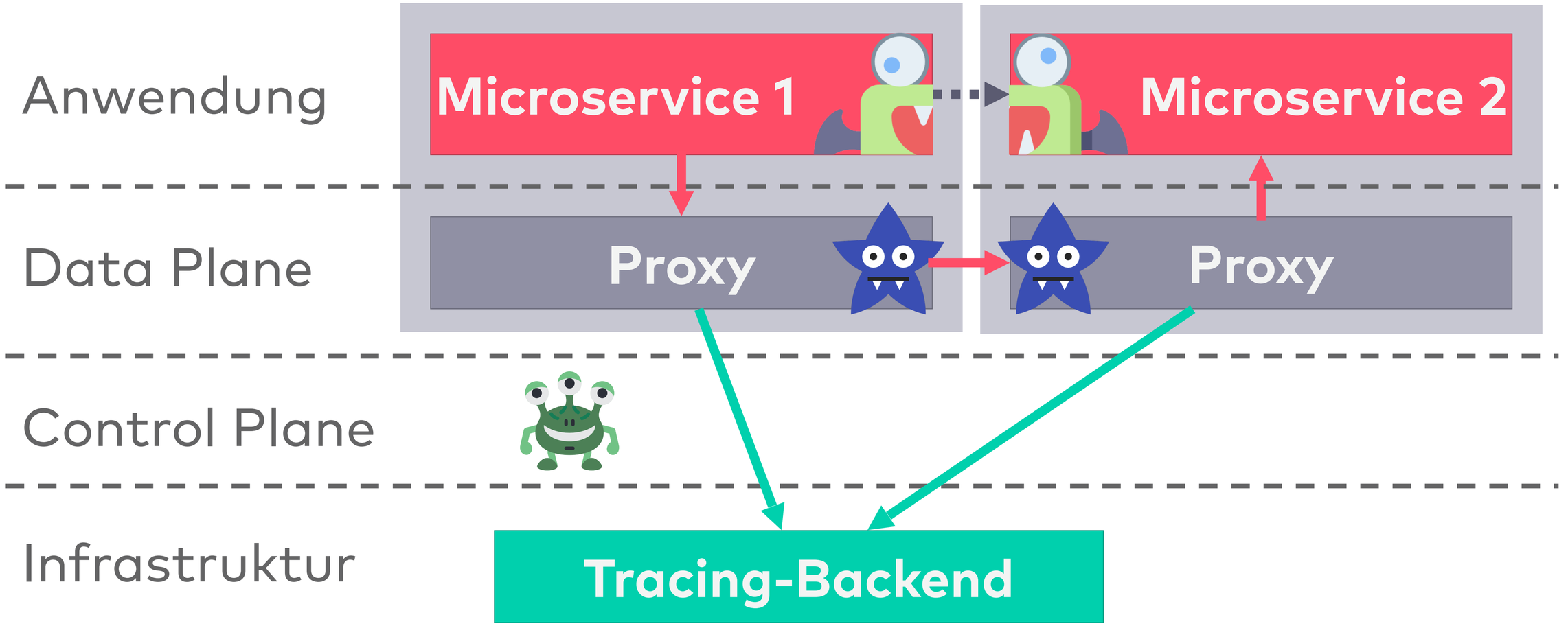
```
# Monitoring | Ingress IP
INGRESS_HOST="$(kubectl -n istio-system get service
istio-ingressgateway -o jsonpath='
{.status.loadBalancer.ingress[0].ip}')"
echo http://"$INGRESS_HOST"

# IP-Adresse zeigen

# -------------M-O-N-I-T-O-R-I-N-G----------------

# Load?
kubectl apply -f istio-config/01_loadgenerator.yaml

# Monitoring | Kiali
istioctl dashboard kiali

# Monitoring | Jaeger
istioctl dashboard jaeger

# -------------R-O-U-T-I-N-G----------------

# Frontend Version 2
open 02_frontend_v2.yaml
kubectl apply -f istio-config/02_frontend_v2.yaml
kubectl get pods

# Frontend Version nach URL
# Virtual Service
open istio-config/03_frontend-virtual-service-uri.yaml
kubectl apply -f istio-config/
03_frontend-virtual-service-uri.yaml
# Destionation Rule
open istio-config/04_frontend-destination-rule.yaml
kubectl apply -f istio-config/
04_frontend-destination-rule.yaml

# Prozentuales Routing
open istio-config/05_routing-percemt.yaml
kubectl apply -f istio-config/05_routing-percemt.yaml

# Test
for i in $(seq 20); do curl -s http://$INGRESS_HOST |
grep 'Hipster Shop' | grep -v '<title>'; done;
```

Tracing

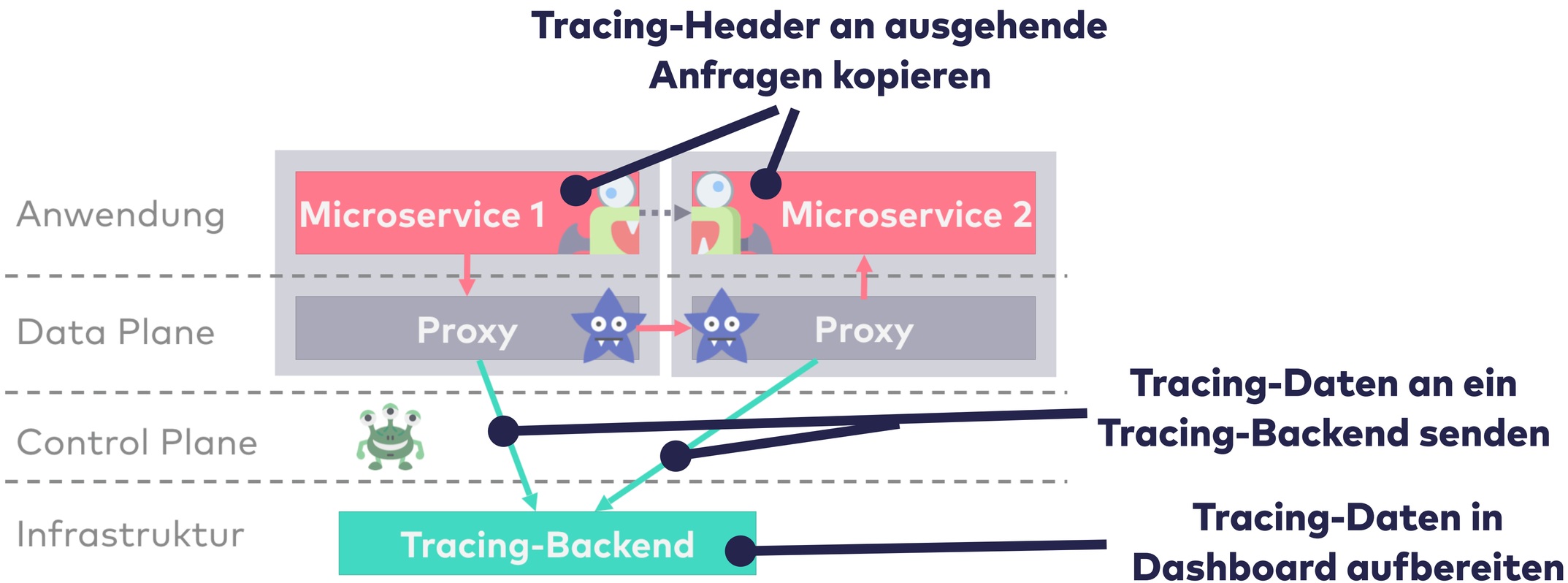
- Beantwortet…
  - Welcher Aufruf erzeugt welchen **Folgeaufruf**?
  - Welchen Anteil hat jeder Teilaufruf an der **Latenz**?
  - Wo ist ein **Fehler** entstanden?
- Umsetzung:



Tracing mit Service Mesh | Config



Tracing mit Service Mesh



Demo

istioctl (istioctl)

```
(⎈ |istio:default)➜  istio-demo INGRESS_HOST="$(kubectl -n istio-system get service istio-ingress
gateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}')"
echo http://"$INGRESS_HOST"
http://34.90.62.137
(⎈ |istio:default)➜  istio-demo kubectl apply -f istio-config/01_loadgenerator.yaml
pod/loadgenerator created
(⎈ |istio:default)➜  istio-demo istioctl dashboard kiali
http://localhost:60262/kiali
```

~ (zsh)

```
Last login: Wed Nov 13 00:27:15 on ttys028
(⎈ |istio:default)➜  ~
```

demo.sh

```
# Monitoring | Ingress IP
INGRESS_HOST="$(kubectl -n istio-system get service
istio-ingressgateway -o jsonpath='
{.status.loadBalancer.ingress[0].ip}')"
echo http://"$INGRESS_HOST"

# IP-Adresse zeigen

# -------------M-O-N-I-T-O-R-I-N-G----------------

# Load?
kubectl apply -f istio-config/01_loadgenerator.yaml

# Monitoring | Kiali
istioctl dashboard kiali

# Monitoring | Jaeger
istioctl dashboard jaeger

# -------------R-O-U-T-I-N-G----------------

# Frontend Version 2
open 02_frontend_v2.yaml
kubectl apply -f istio-config/02_frontend_v2.yaml
kubectl get pods

# Frontend Version nach URL
# Virtual Service
open istio-config/03_frontend-virtual-service-uri.yaml
kubectl apply -f istio-config/
03_frontend-virtual-service-uri.yaml
# Destionation Rule
open istio-config/04_frontend-destination-rule.yaml
kubectl apply -f istio-config/
04_frontend-destination-rule.yaml

# Prozentuales Routing
open istio-config/05_routing-percemt.yaml
kubectl apply -f istio-config/05_routing-percemt.yaml

# Test
for i in $(seq 20); do curl -s http://$INGRESS_HOST |
grep 'Hipster Shop' | grep -v '<title>'; done;
```

Service Mesh Features

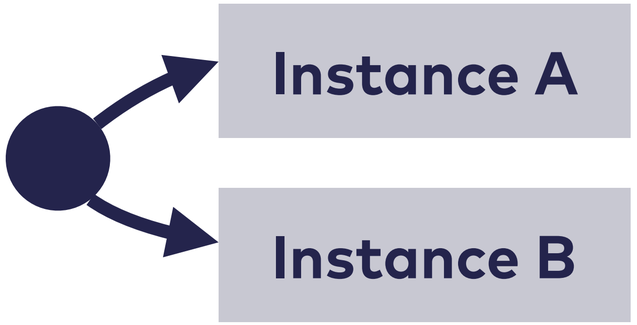
Observability

Routing

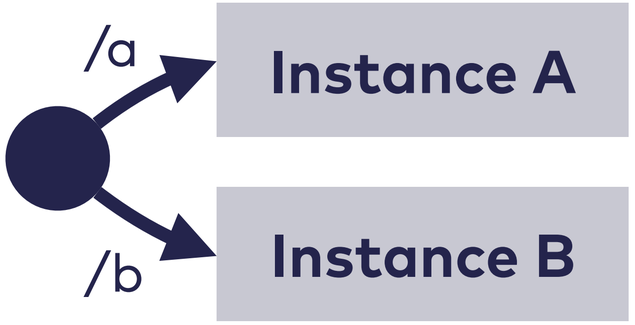
Resilience

Security

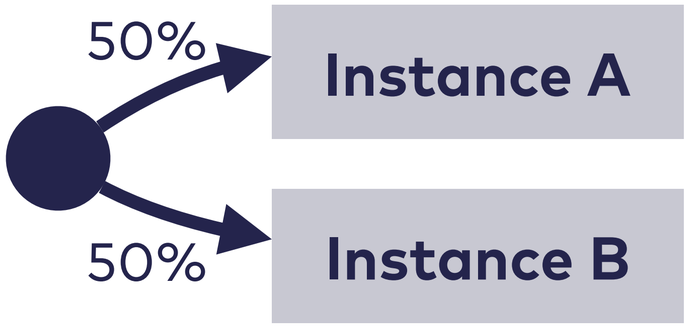
Routing



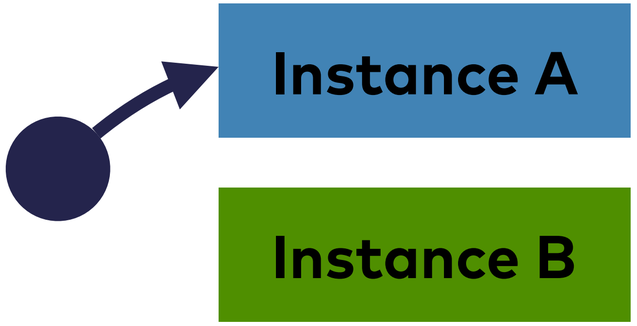
Load Balancing



Pfad-basiertes Routing



A/B-Testing

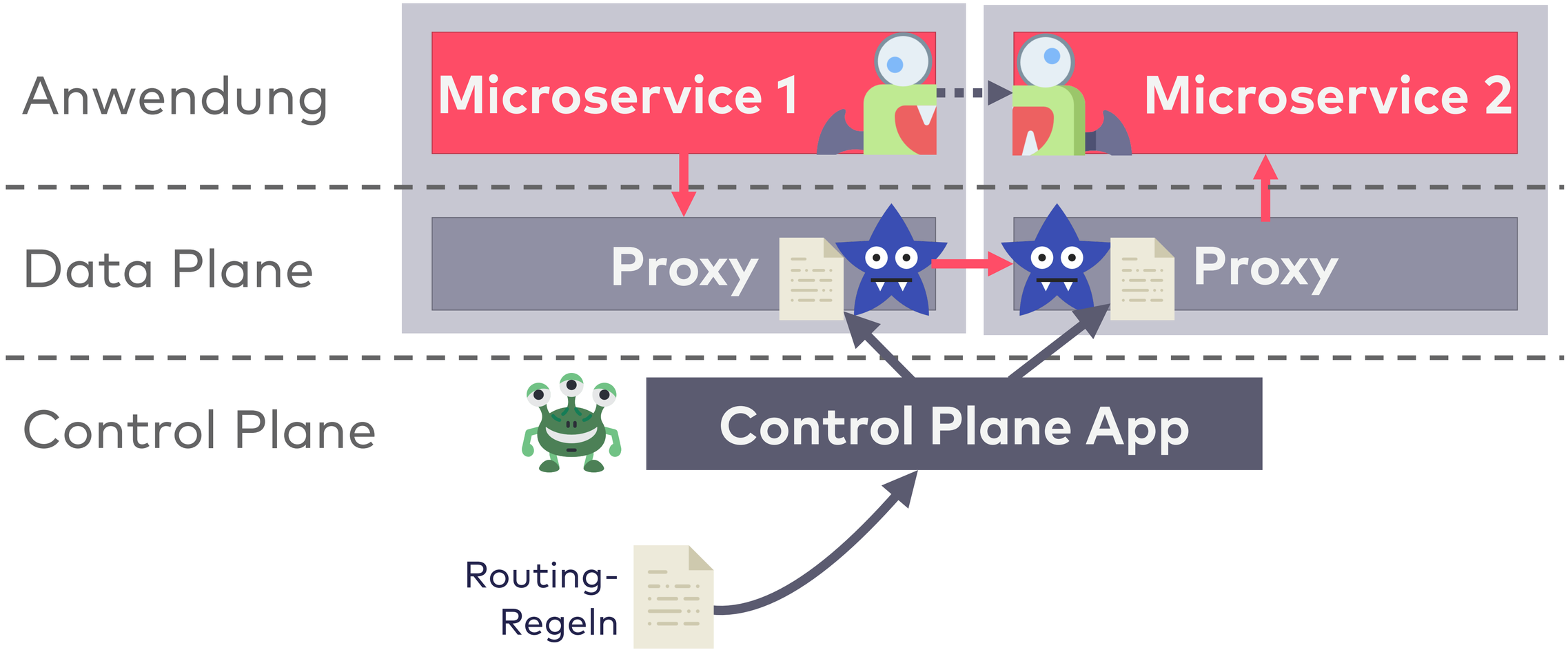


Blue/Green Deployment

Canary Releasing

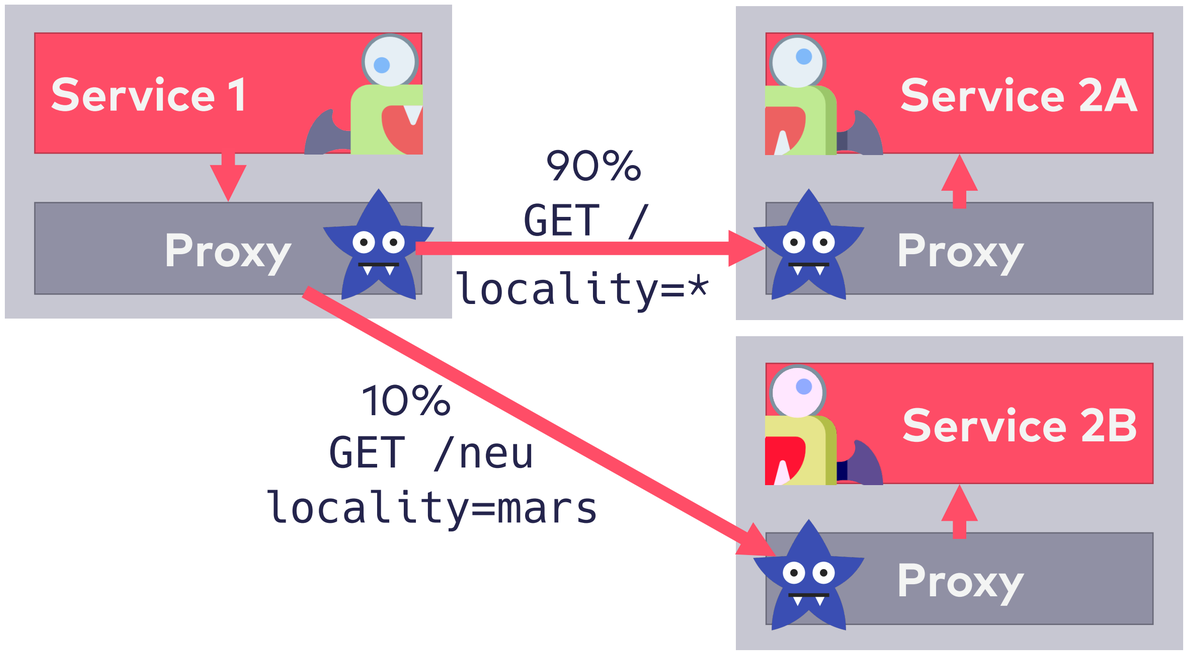
Typischerweise im **Edge Router / API-Gateway** implementiert z.B. NGINX, Kong, Ambassador, Traefik

Routing mit Service Mesh | Config

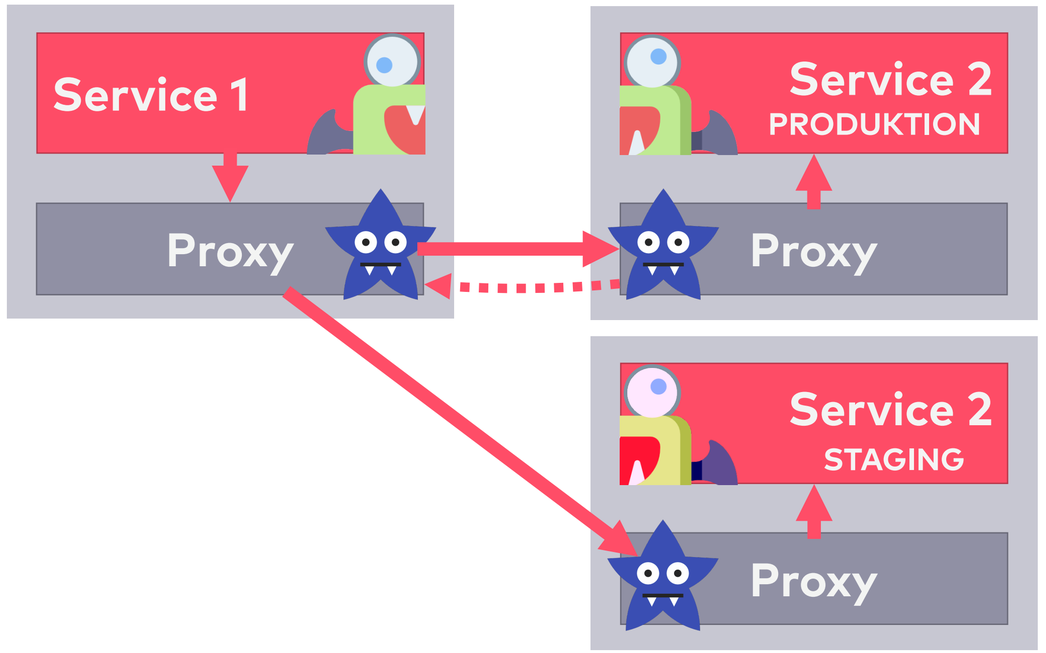


Routing mit Service Mesh

Komplexe Routing-Regeln
für A/B Testing und Canary Releasing

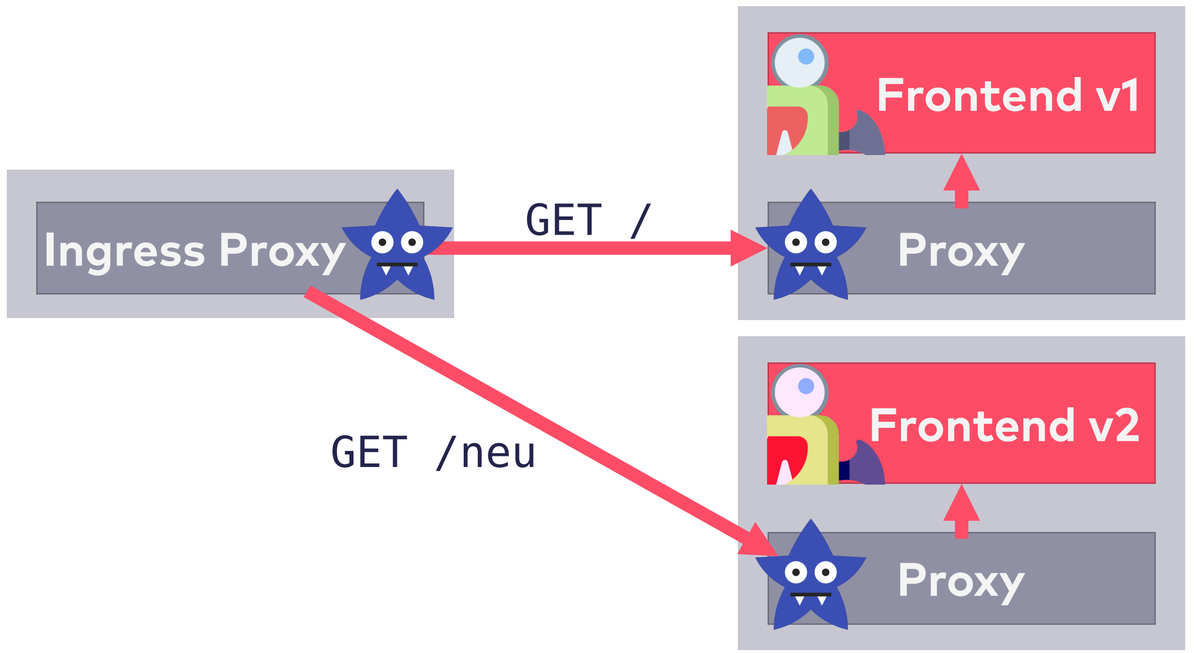


Traffic Mirroring



Routing mit Istio

Beispiel-Routing mit Istio



Routing Config

```yaml
apiVersion: networking.istio.io/…
kind: DestinationRule
metadata:
  name: frontend

spec:
  host: frontend
  subsets:
  - name: v2
    labels:
      version: "2"
```

Subset “**v2**”:
Alle Pods mit Label `version: "2"`

```yaml
  - name: v1
    labels:
      version: "1"
```

Subset “**v1**”:
Alle Pods mit Label `version: "1"`

```yaml
apiVersion: networking.istio.io/…
kind: VirtualService
metadata:
  name: frontend-ingress
spec:
  hosts:
  - "*"
  http:
  - match:
    - uri:
        prefix: "/neu"
    route:
    - destination:
        host: frontend
        subset: v2
  - route:
    - destination:
        host: frontend
        subset: v1
```

Regel 1:
Wenn die **URI** `"/neu"` **enthält**, dann leite an `frontend` **v2** weiter

Regel 2:
Ansonsten leite an `frontend` **v1** weiter

Demo

~/istio-demo (zsh)

```
(⎈ |istio:default)➜  istio-demo INGRESS_HOST="$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}')"
echo http://"$INGRESS_HOST"
http://34.90.62.137
(⎈ |istio:default)➜  istio-demo kubectl apply -f istio-config/01_loadgenerator.yaml
pod/loadgenerator created
(⎈ |istio:default)➜  istio-demo istioctl dashboard kiali
http://localhost:60262/kiali
^C%
(⎈ |istio:default)➜  istio-demo istioctl dashboard jaeger
http://localhost:60281
^C%
(⎈ |istio:default)➜  istio-demo open 02_frontend_v2.yaml
The file /Users/hanna/istio-demo/02_frontend_v2.yaml does not exist.
(⎈ |istio:default)➜  istio-demo
```

~ (zsh)

```
Last login: Wed Nov 13 00:27:15 on ttys028
(⎈ |istio:default)➜  ~
```

demo.sh

```
# Monitoring | Ingress IP
INGRESS_HOST="$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}')"
echo http://"$INGRESS_HOST"

# IP-Adresse zeigen

# -------------M-O-N-I-T-O-R-I-N-G----------------

# Load?
kubectl apply -f istio-config/01_loadgenerator.yaml

# Monitoring | Kiali
istioctl dashboard kiali

# Monitoring | Jaeger
istioctl dashboard jaeger

# -------------R-O-U-T-I-N-G----------------

# Frontend Version 2
open 02_frontend_v2.yaml
kubectl apply -f istio-config/02_frontend_v2.yaml
kubectl get pods

# Frontend Version nach URL
# Virtual Service
open istio-config/03_frontend-virtual-service-uri.yaml
kubectl apply -f istio-config/03_frontend-virtual-service-uri.yaml
# Destionation Rule
open istio-config/04_frontend-destination-rule.yaml
kubectl apply -f istio-config/04_frontend-destination-rule.yaml

# Prozentuales Routing
open istio-config/05_routing-percemt.yaml
kubectl apply -f istio-config/05_routing-percemt.yaml

# Test
for i in $(seq 20); do curl -s http://$INGRESS_HOST | grep 'Hipster Shop' | grep -v '<title>'; done;
```

Service Mesh Features

Observability

Routing

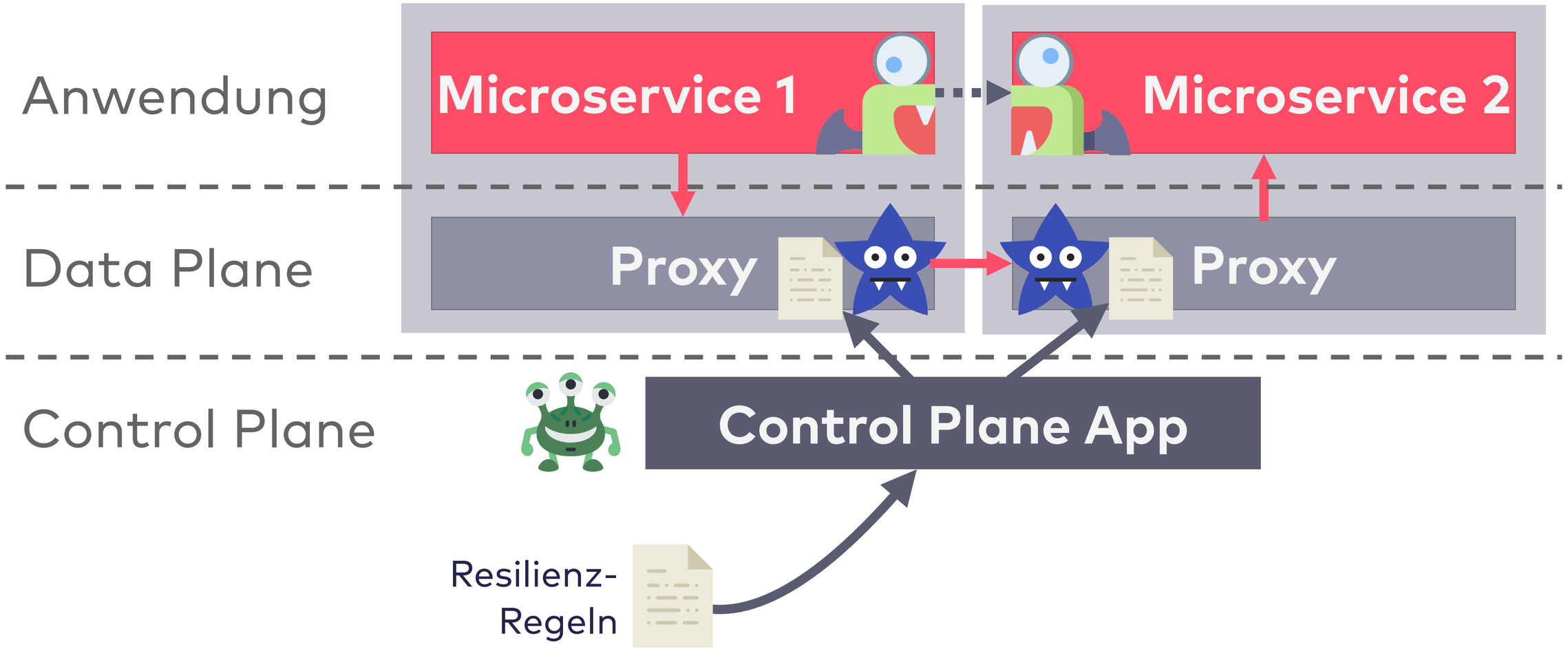
Resilience

Security

Resilience

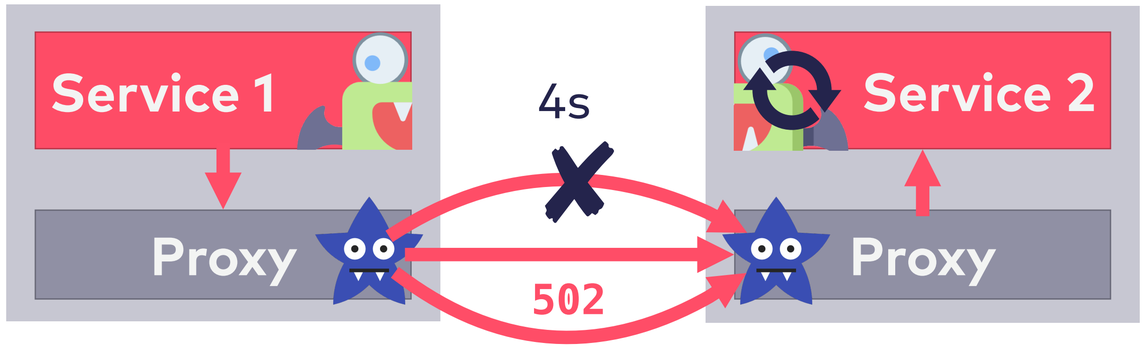
- Was wenn ein Service nicht wie erwartet zur Verfügung steht?
  - Fehler (5xx)
  - Delay
- Ziel: **Gesamtsystem funktioniert weiter**
  - ggf. mit Einschränkungen
- **Resilience-Maßnahmen**: Retry, Timeout, Circuit Breaking

Resilience mit Service Mesh | Config

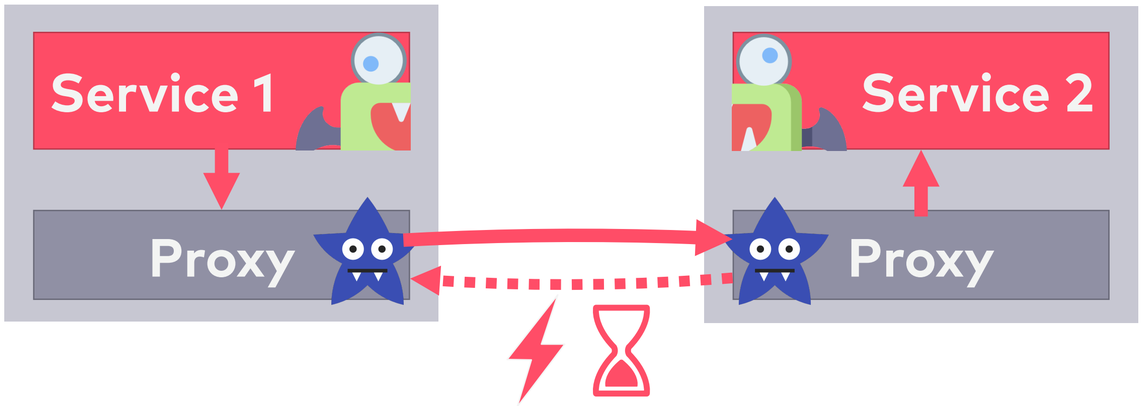


Resilience mit Service Mesh

Timeout
Retry



Fault Injection
Delay Injection



Resilience mit Istio

Retry & Timeout

Timeout & Retry

hosts: Zieladresse des Client-Requests

max. Anzahl der Retrys

Timeout pro Retry

Bedingungen für ein Retry

Timeout für alle Retrys zusammen

destination: Wo der Request tatsächlich hingeht

```yaml
apiVersion: networking.istio.io/...
kind: VirtualService
metadata:
name: recommendationservice
spec:
  hosts:
  - recommendationservice
  http:
  - retries:
      attempts: 3
      perTryTimeout: 1s
      retryOn: connect-failure,5xx
    timeout: 2s
    route:
    - destination:
        host: recommendationservice
```

Service Mesh Features

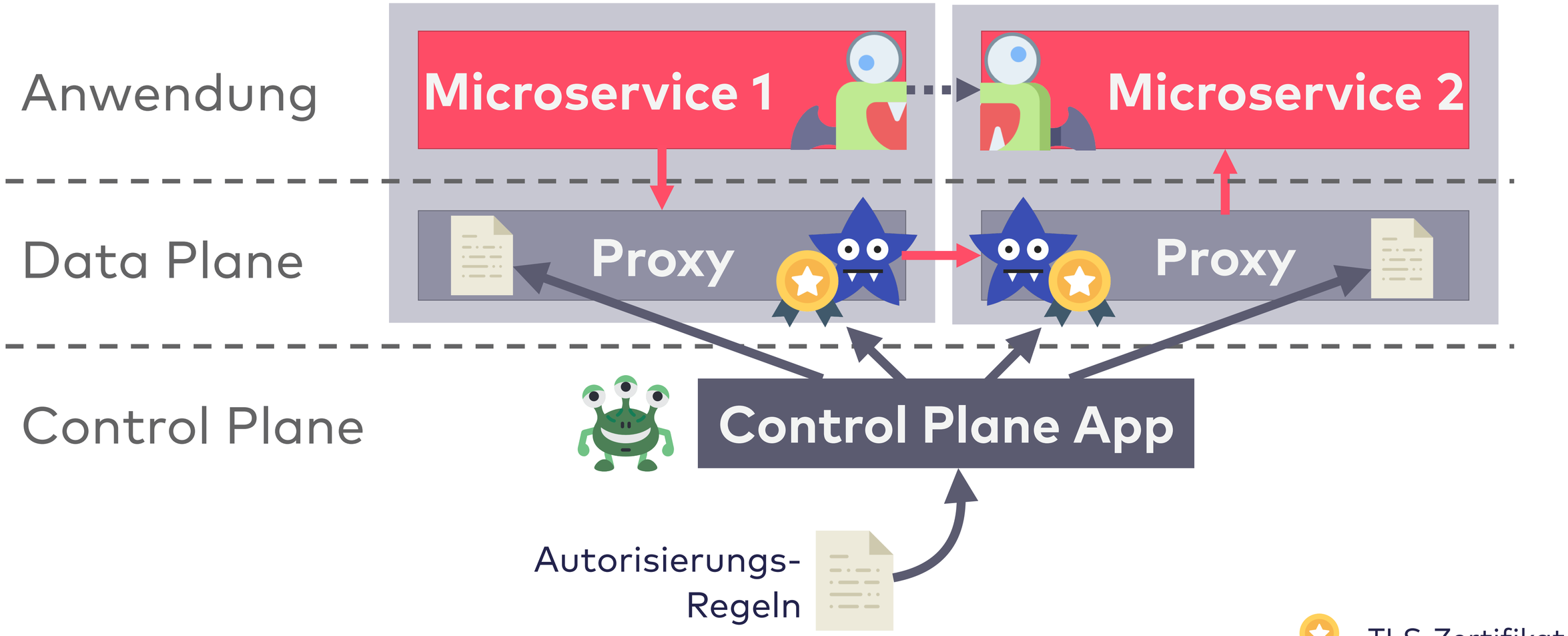
Observability

Routing

Resilience

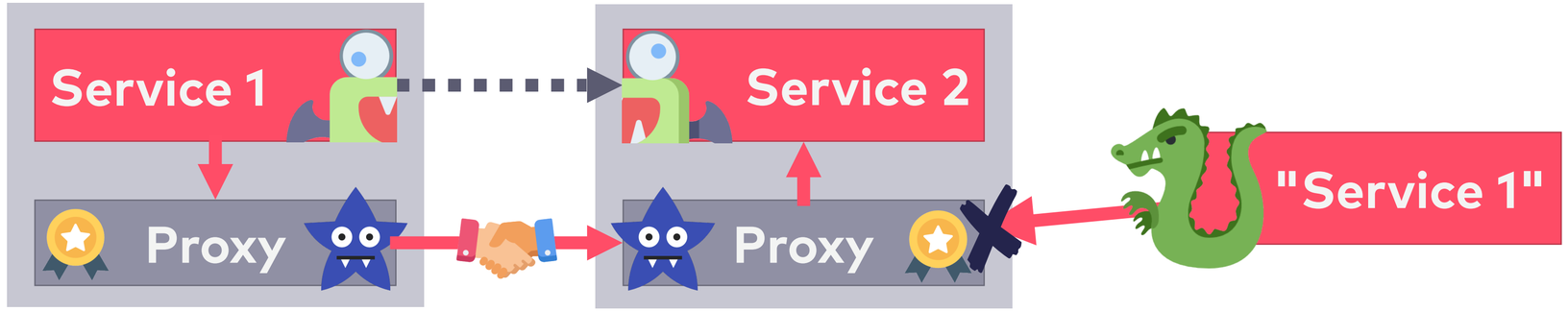
Security

Security mit Service Mesh | Config

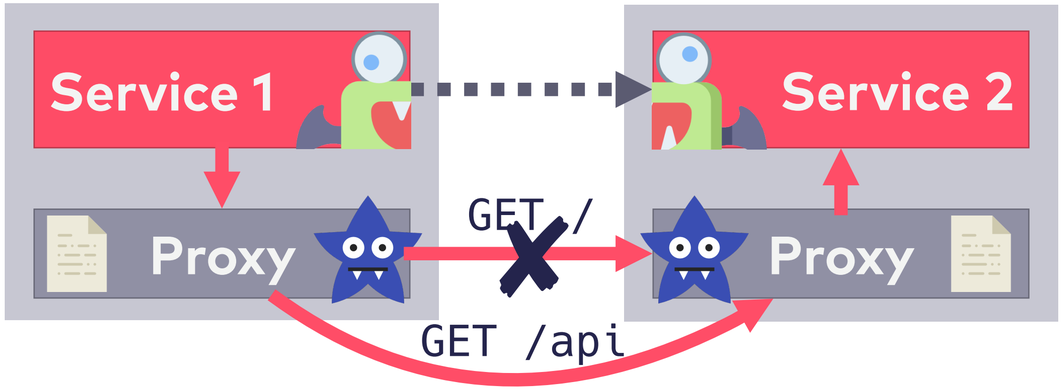


Security mit Service Mesh

Authentifizierung mit mTLS



Autorisierung



TLS-Zertifikat

Autorisierungs-Regel

Istio Security

mTLS

Demo

~/istio-demo (zsh)

```
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop II
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop
                    Hipster Shop II
                    Hipster Shop
                    Hipster Shop
(⎈ |istio:default)➜  istio-demo istioctl dashboard kiali
http://localhost:60681/kiali
^C%
(⎈ |istio:default)➜  istio-demo open istio-config/06_routing-header.yaml
(⎈ |istio:default)➜  istio-demo kubectl apply -f istio-config/06_routing-header
virtualservice.networking.istio.io/frontend-ingress configured
(⎈ |istio:default)➜  istio-demo
```

~ (zsh)

```
Last login: Wed Nov 13 00:27:15 on ttys028
(⎈ |istio:default)➜  ~
```

demo.sh | 06_routing-header.yaml

```
33  # Prozentuales Routing
34  open istio-config/05_routing-percemt.yaml
35  kubectl apply -f istio-config/05_routing-percemt.yaml
36
37  # Test
38  for i in $(seq 20); do curl -s http://$INGRESS_HOST |
     grep 'Hipster Shop' | grep -v '<title>'; done;
39
40
41  # Header-Routing
42  open istio-config/06_routing-header.yaml
43  kubectl apply -f istio-config/06_routing-header.yaml
44
45  # -------------S-E-C-U-R-I-T-Y---------------
46
47  # Permissive Policy
48  open istio-config/07_mtls_permissive.yaml
49  kubectl apply -f istio-config/07_mtls_permissive.yaml
50
51  # mTLS Rules
52  open istio-config/08_mtls-rules.yaml
53  kubectl apply -f istio-config/08_mtls-rules.yaml
54
55  # Kiali
56
57  istioctl dashboard kiali
58
```

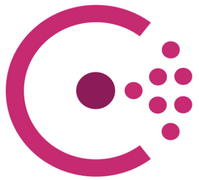
Service Mesh Features

Netzwerk-Metriken und -Logs Daten an Tracing-Backend senden	 Observability	Business-Metriken oder -Logs Weitergabe von Tracing-Headern
Komplexe Routing-Regeln Canary Releasing & A/B-Testing	 Routing	Routing anhand von Request Body Automatisierung Canary Releasing
Automatische Timeouts & Retrys Automatisches Circuit Breaking	 Resilience	Cache oder Standardantworten in Circuit Breaker nutzen
Authentifizierung mit mTLS Autorisierung	 Security	

One Mesh  to rule them all?

Service Mesh Implementierungen

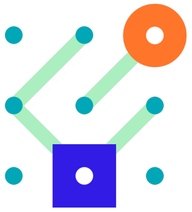
LINKERD



AWS App Mesh

mæsh

Istio



Service
Mesh
Interface

Kuma

Vergleich auf servicemesh.es

Was ist wichtig für die Auswahl eines Service Mesh?

Features

Kompatibilität

Performance

Usability

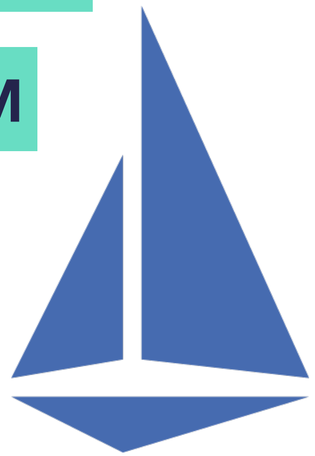
Istio VS Linkerd 2

*2017

von **Google & IBM**

optimiert auf **Featureanzahl** und **Konfigurierbarkeit**

optimiert für Kubernetes, aber nicht exklusiv



*2017

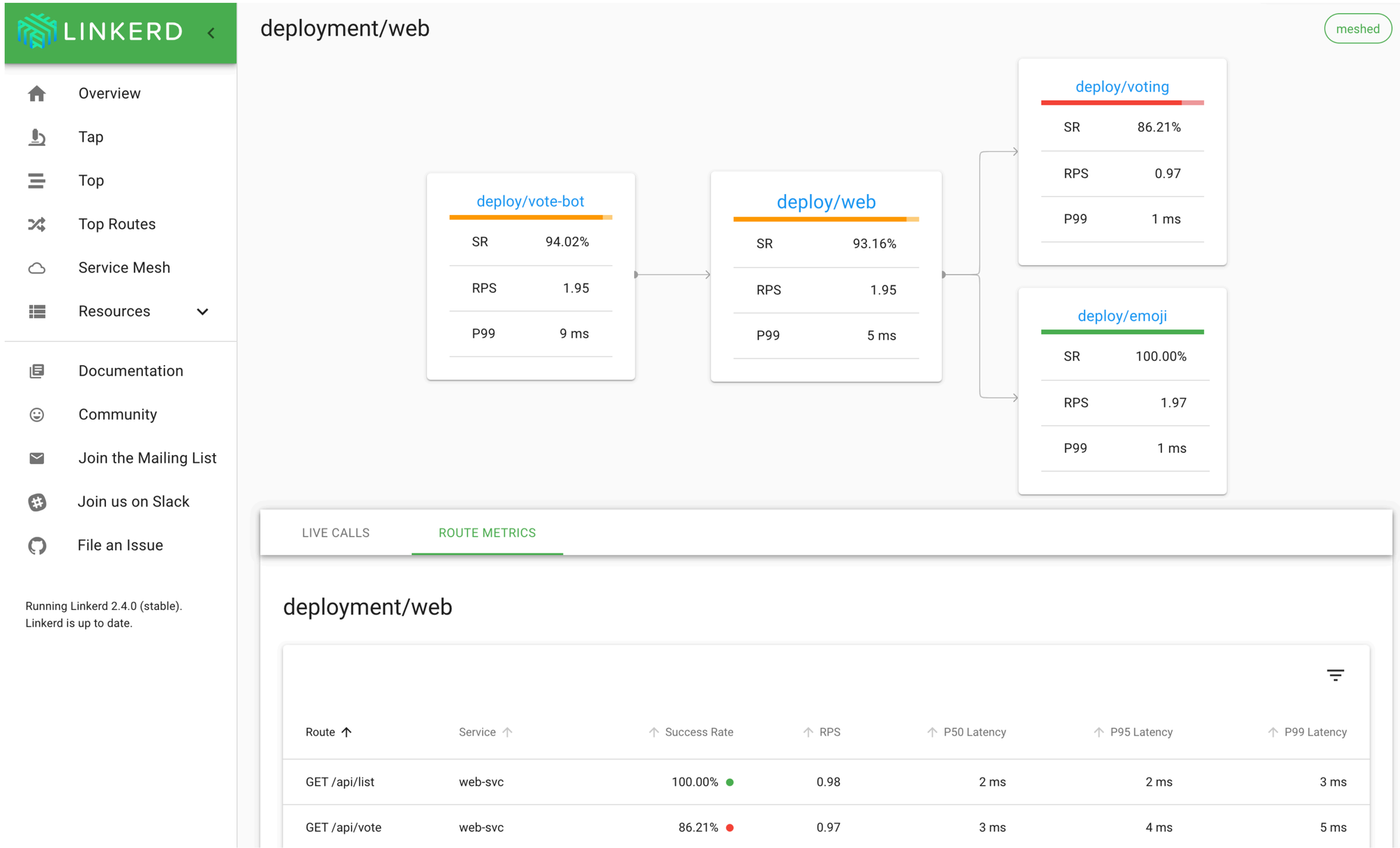
von **Buoyant**

optimiert auf **Usability** und **Performance**

Kubernetes only

Features

		Istio 1.3	Linkerd 2.6
Generell	**Plattform**	Verschiedene	Kubernetes
	Automatische Sidecar-Injektion	✓	✓
Monitoring	**Metriken**	✓	✓
	Tracing	✓	(✓)
	Dashboard mit Service Graph	✓	✓
Routing	**Load Balancing**	✓	✓
	Routing-Regeln	✓	✓
Resilienz	**Timeouts & Retrys**	✓	✓
	Circuit Breaker	✓	✘
Sicherheit	**Authentifizierung via mTLS**	✓	✓
	Autorisierung	✓	✘


LINKERD
Overview
Tap
Top
Top Routes
Service Mesh
Resources
Documentation
Community
Join the Mailing List
Join us on Slack
File an Issue
Running Linkerd 2.4.0 (stable).
Linkerd is up to date.
deployment/web
meshed
deploy/vote-bot
SR 94.02%
RPS 1.95
P99 9 ms
deploy/web
SR 93.16%
RPS 1.95
P99 5 ms
deploy/voting
SR 86.21%
RPS 0.97
P99 1 ms
deploy/emoji
SR 100.00%
RPS 1.97
P99 1 ms
LIVE CALLS
ROUTE METRICS
deployment/web
Route
Service
Success Rate
RPS
P50 Latency
P95 Latency
P99 Latency
GET /api/list
web-svc
100.00%
0.98
2 ms
2 ms
3 ms
GET /api/vote
web-svc
86.21%
0.97
3 ms
4 ms
5 ms

Performance & Ressourcen

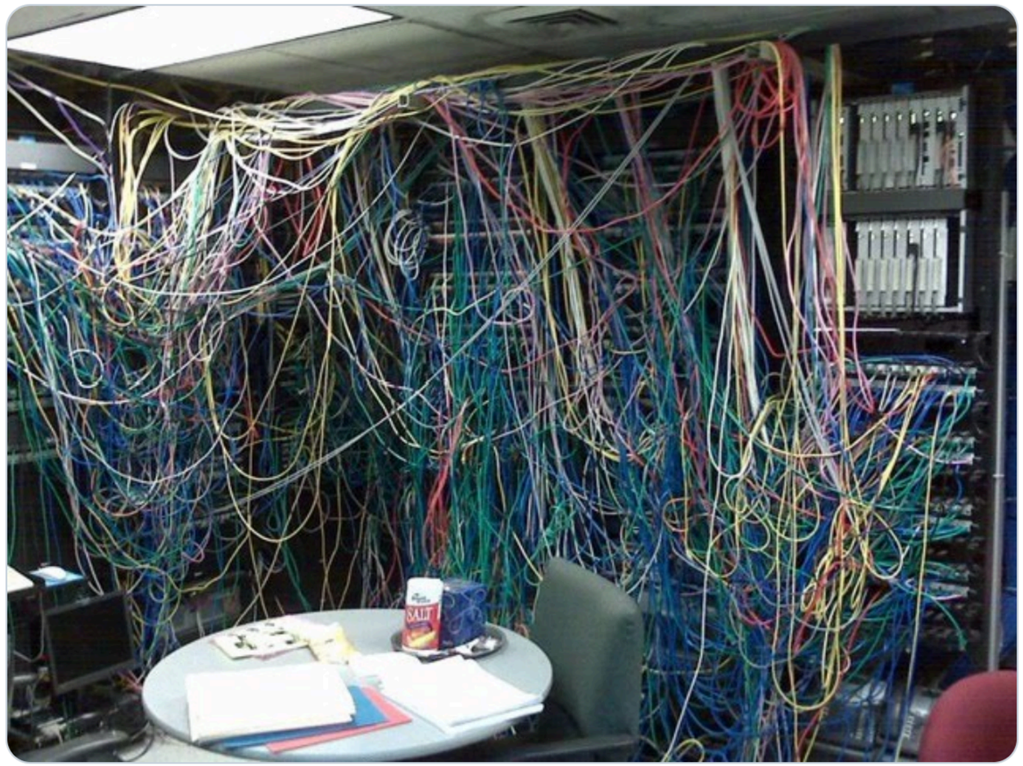
- **Latenz**
  - **Istio**: zusätzlich ca. 8ms Latenz - pro Aufruf zwischen Services!
  - **Linkerd 2**: ähnlich, mit weniger starken Schwankungen
- **Ressourcen**
  - Zusätzliche Container für Control Plane & jedes Sidecar
  - → Erhöhter CPU- & Arbeitsspeicher-Verbrauch

Aber: Abhängig von konkretem Setting
→ eigenen Benchmark machen!

Usability

Dave Strebel
@dave_strebel

“Finally got Istio into production”



2:21 AM · Sep 18, 2019 from Apple Valley, MN · Tweetbot for iOS

322 Retweets **1.2K** Likes

Service Mesh Fazit

+

Löst viele wesentliche Probleme von Microservices

... ohne Codeänderungen!

-

Eine weitere komplexe Technologie

Latenz- und Ressourcen-Overhead

Entscheidungshilfe

Mesh oder kein Mesh?

- **Service Mesh** wenn:
  - Großteil in Kubernetes läuft
  - Viele Microservices, viele synchrone Aufrufe
  - Viele ungelöste Probleme beim Monitoring, Routing, Resilienz und/oder Security

Istio oder Linkerd 2?

- **Istio nur** wenn Komplexität und Flexibilität nötig ist:
  - Teilsystem bleibt außerhalb von Kubernetes
  - Circuit Breaking und Autorisierung unverzichtbar
  - Ausreichend zeitliche und kognitive Kapazität im Team

Mehr Mesh

- Artikel Brauchen asynchroner Microservices und Self-Contained-Systems ein Service Mesh?
- Artikel Alle 11 Minuten verliebt sich ein Microservice in Linkerd
- Service Mesh Vergleich auf servicemesh.es
- Podcast zu Service Meshes
- Beispiel-Anwendung auf GitHub
- Tutorial von Linkerd
- Tutorial von Istio

Danke! Fragen?

Hanna Prinz
hanna.prinz@innoq.com
@HannaPrinz

INNOQ
www.innoq.com

Kostenloser Service Mesh Primer
gedruckt am INNOQ-Stand!

Oder auf leanpub.com/service-mesh-primer

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
Germany
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin
Germany
+49 2173 3366-0

Ludwigstr. 180E
63067 Offenbach
Germany
+49 2173 3366-0

Kreuzstr. 16
80331 München
Germany
+49 2173 3366-0

Hermannstrasse 13
20095 Hamburg
Germany
+49 2173 3366-0

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
+41 41 743 0116