K8s Service-Mesh Introduction

Wen Zhenglin
2018-11-28

概览

- 什么是Service-mesh
- Service-mesh的主要解决方案
- Istio的介绍及示例
- Linkerd简要介绍
- 总结

什么是service-mesh

Service mesh is a network of microservices that make up such applications and the interactions between them.

As a service mesh grows in size and complexity, it can become harder to understand and manage

Its requirements can include discovery, load balancing, failure recovery, metrics, and monitoring.

A service mesh also often has more complex operational requirements, like A/B testing, canary releases, rate limiting, access control, and end-to-end authentication.

Service-mesh解决方案

1. Istio
2. Linkerd

Istio

Istio介绍

Istio is a sidecar container implementation of the features and functions needed when creating and managing microservices. Monitoring, tracing, circuit breakers, routing, load balancing, fault injection, retries, timeouts, mirroring, access control, rate limiting, and more, are all a part of this.

Istio let’s you get these benefits with no changes to your source code.

Istio的特性

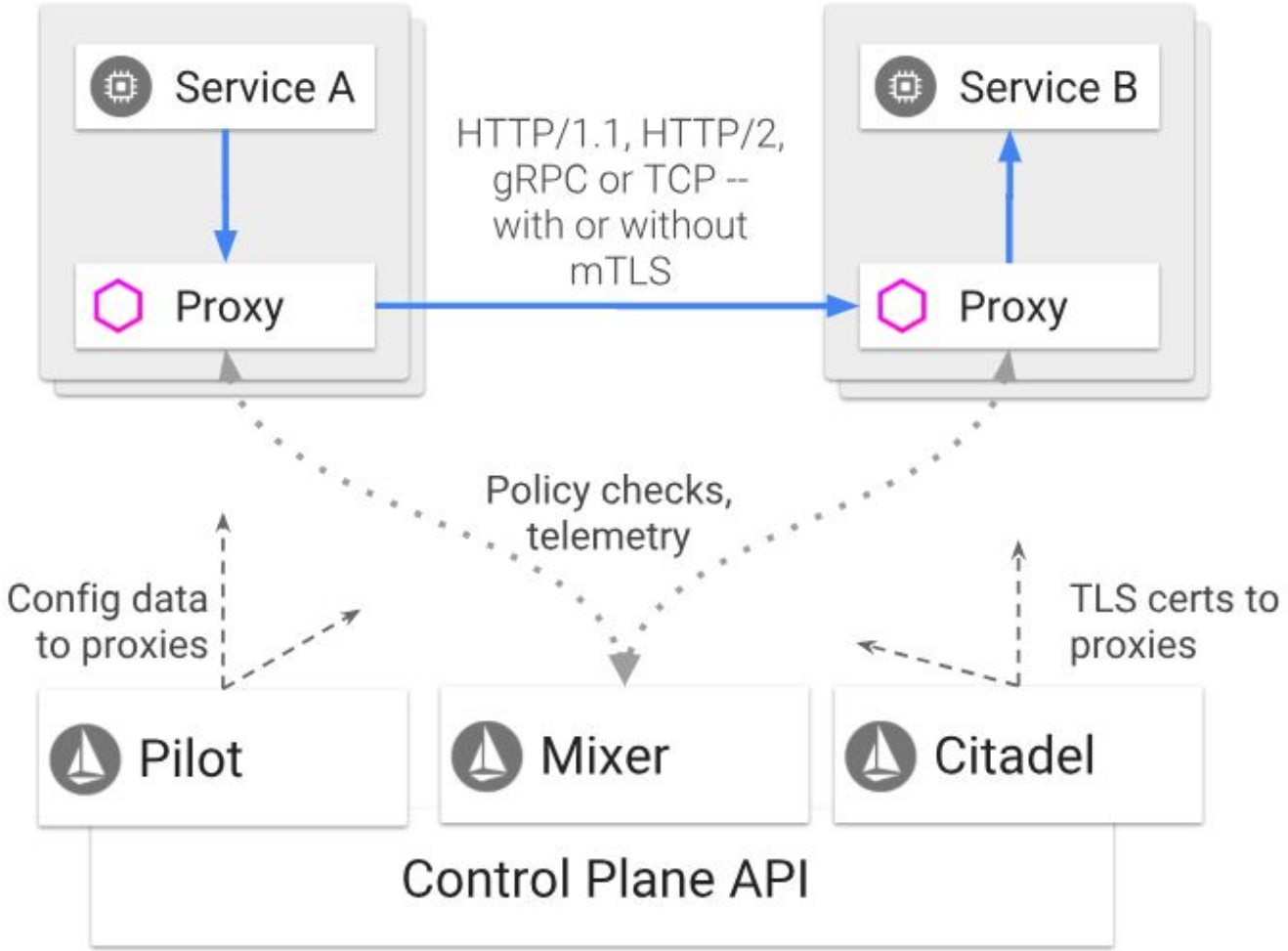
- Automatic load balancing for HTTP, gRPC, WebSocket, and TCP traffic.
- Fine-grained control of traffic behavior with rich routing rules, retries, failovers, and fault injection.
- A pluggable policy layer and configuration API supporting access controls, rate limits and quotas.
- Automatic metrics, logs, and traces for all traffic within a cluster, including cluster ingress and egress.
- Secure service-to-service communication in a cluster with strong identity-based authentication and authorization.

Istio arch

An Istio service mesh is logically split into a data plane and a control plane.

- The data plane is composed of a set of intelligent proxies (Envoy) deployed as sidecars. These proxies mediate and control all network communication between microservices along with Mixer, a general-purpose policy and telemetry hub.
- The control plane manages and configures the proxies to route traffic. Additionally, the control plane configures Mixers to enforce policies and collect telemetry.

Istio arch


Service A
Service B
HTTP/1.1, HTTP/2,
gRPC or TCP --
with or without
mTLS
Proxy
Proxy
Policy checks,
telemetry
Config data
to proxies
TLS certs to
proxies
Pilot
Mixer
Citadel
Control Plane API

Envoy(proxy)提供的特性

- Dynamic service discovery
- Load balancing
- TLS termination
- HTTP/2 and gRPC proxies
- Circuit breakers
- Health checks
- Staged rollouts with %-based traffic split
- Fault injection
- Rich metrics

Rule configuration

新的四种K8s配置资源：

- A VirtualService defines the rules that control how requests for a service are routed within an Istio service mesh.
- A DestinationRule configures the set of policies to be applied to a request after VirtualService routing has occurred.
- A ServiceEntry is commonly used to enable requests to services outside of an Istio service mesh.
- A Gateway configures a load balancer for HTTP/TCP traffic, most commonly operating at the edge of the mesh to enable ingress traffic for an application.

Virtual service

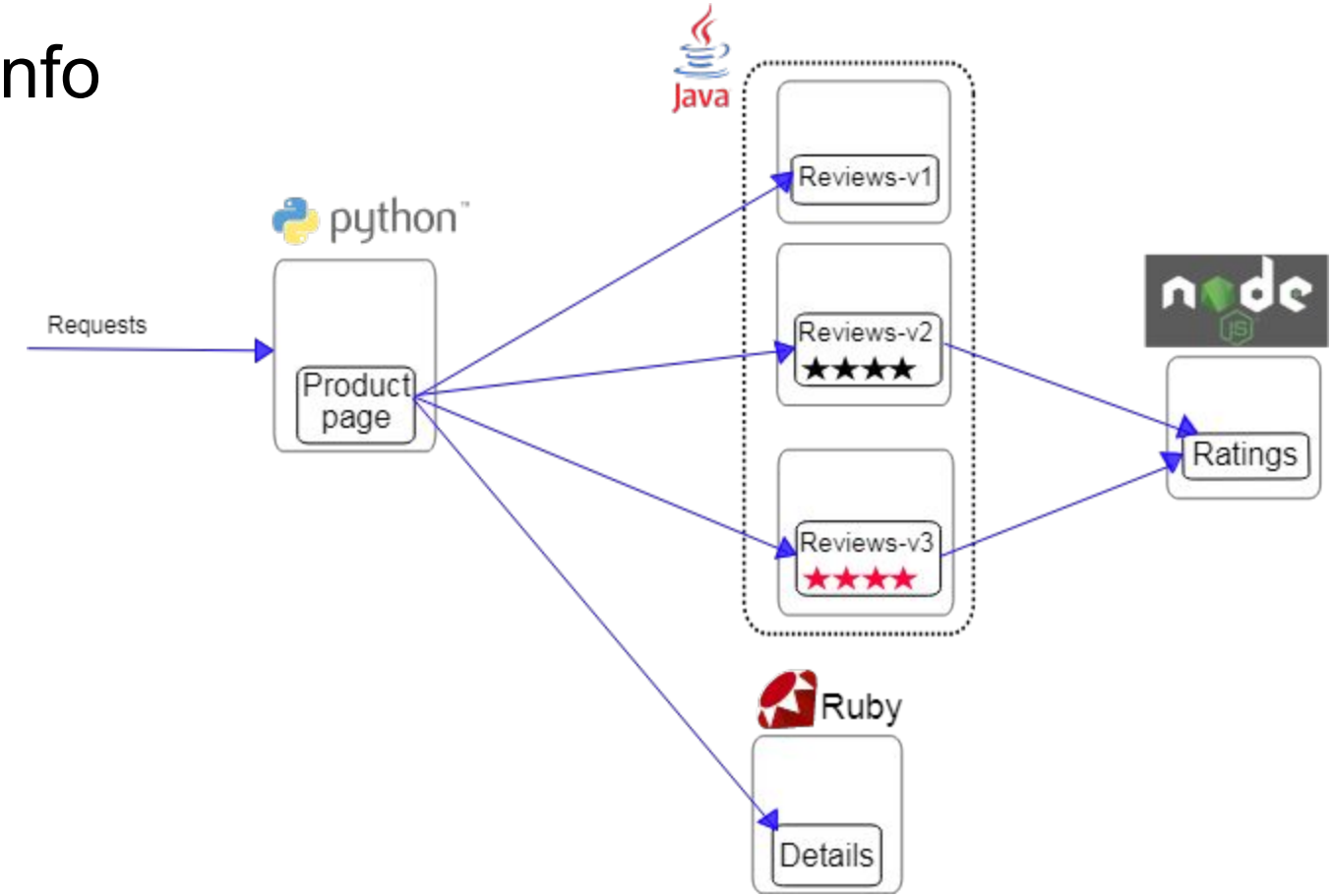
```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
```

Destination Rule

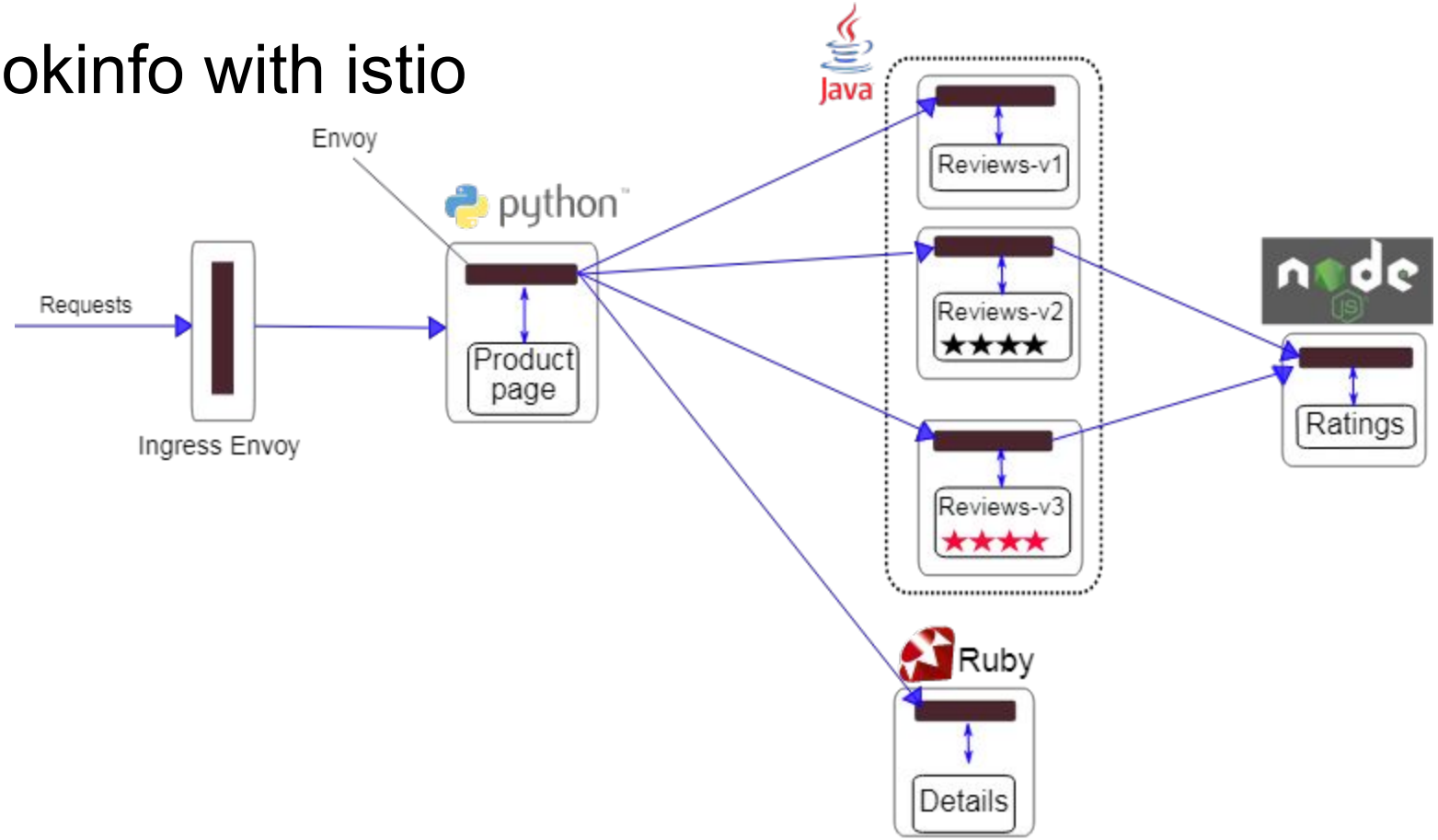
```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: reviews
spec:
  host: reviews
  trafficPolicy:
    loadBalancer:
      simple: RANDOM
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

Demo usage

Bookinfo


Java
Reviews-v1
python
Requests
Reviews-v2
Product page
Ratings
Reviews-v3
Ruby
Details

Bookinfo with istio



Splitting traffic between versions

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
      weight: 75
    - destination:
        host: reviews
        subset: v2
      weight: 25
```

Timeouts

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
    - ratings
  http:
  - route:
    - destination:
        host: ratings
        subset: v1
    timeout: 10s
```

Timeouts and retries

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
    - ratings
  http:
  - route:
    - destination:
        host: ratings
        subset: v1
    retries:
      attempts: 3
      perTryTimeout: 2s
```

Injecting faults (10% and 5 second delay)

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - fault:
      delay:
        percent: 10
        fixedDelay: 5s
    route:
    - destination:
        host: ratings
        subset: v1
```

Injecting faults (10% and http 400 code)

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - fault:
      abort:
        percent: 10
        httpStatus: 400
    route:
    - destination:
        host: ratings
        subset: v1
```

Injecting faults (review-v2 -> rating-v1 delay & abort)

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - match:
    - sourceLabels:
        app: reviews
        version: v2
    fault:
      delay:
        fixedDelay: 5s
      abort:
        percent: 10
        httpStatus: 400
    route:
    - destination:
        host: ratings
        subset: v1
```

Policies based on subsets

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: reviews
spec:
  host: reviews
  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
```

Circuit breakers

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: reviews
spec:
  host: reviews
  subsets:
  - name: v1
    labels:
      version: v1
    trafficPolicy:
      connectionPool:
        tcp:
          maxConnections: 100
```

Service entries

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bar-foo-ext-svc
spec:
  hosts:
    - bar.foo.com
  http:
  - route:
    - destination:
        host: bar.foo.com
    timeout: 10s
```

Gateway (ingress)

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gateway
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

Gateway (bind virtual service)

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bookinfo
spec:
  hosts:
  - "*"
  gateways:
  - bookinfo-gateway
  http:
  - match:
    - uri:
        exact: /productpage
    - uri:
        prefix: /api/v1/products
    route:
    - destination:
        host: productpage
        port:
          number: 9080
```

开启方式

手工注入sidecar (proxy)

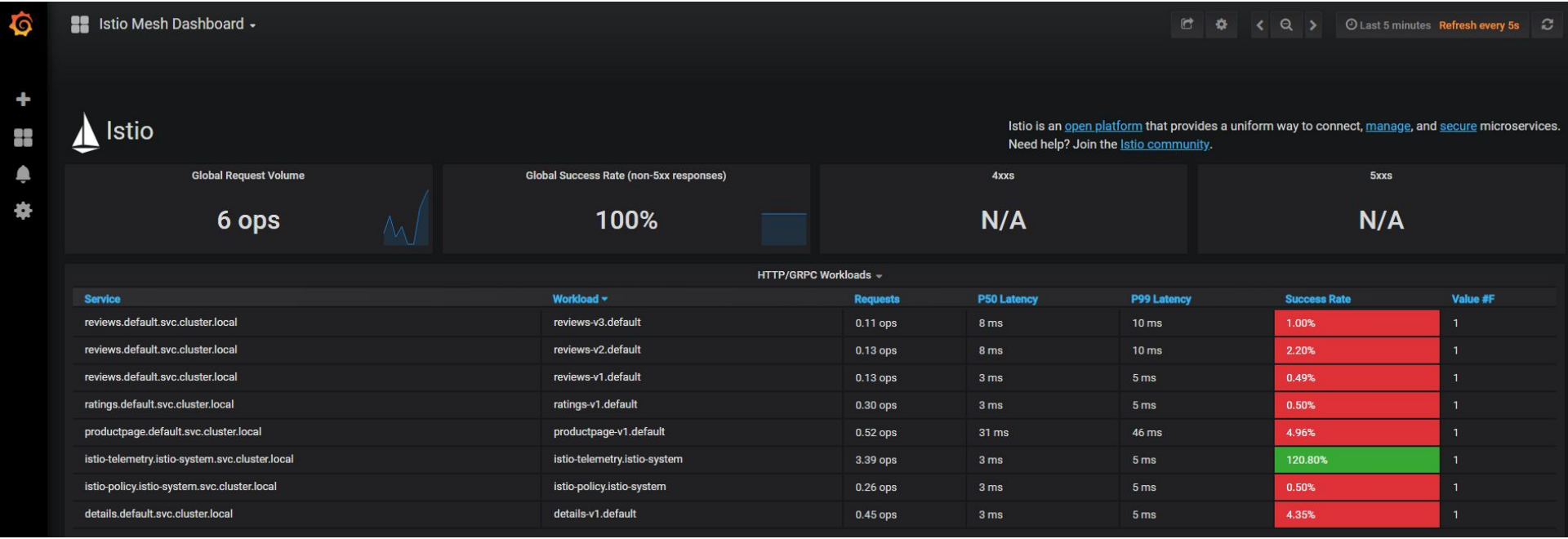
```
istioctl kube-inject -f samples/sleep/sleep.yaml | kubectl apply -f -
```

自动注入sidecar (proxy)

```
kubectl label namespace default istio-injection=enabled
```

```
kubectl apply -f samples/sleep/sleep.yaml
```

Istio grafana



Linkerd

Linkerd介绍

为K8s量身定制的轻量级服务网格方案

Actionable service metrics

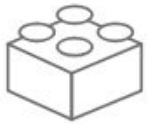
Best-in-class observability allows you to monitor *golden metrics* —success rate, request volume, and latency—for every service.

Deep runtime diagnostics

Get a comprehensive suite of diagnostic tools, including automatic service dependency maps and live traffic samples.

Installs in seconds with zero config

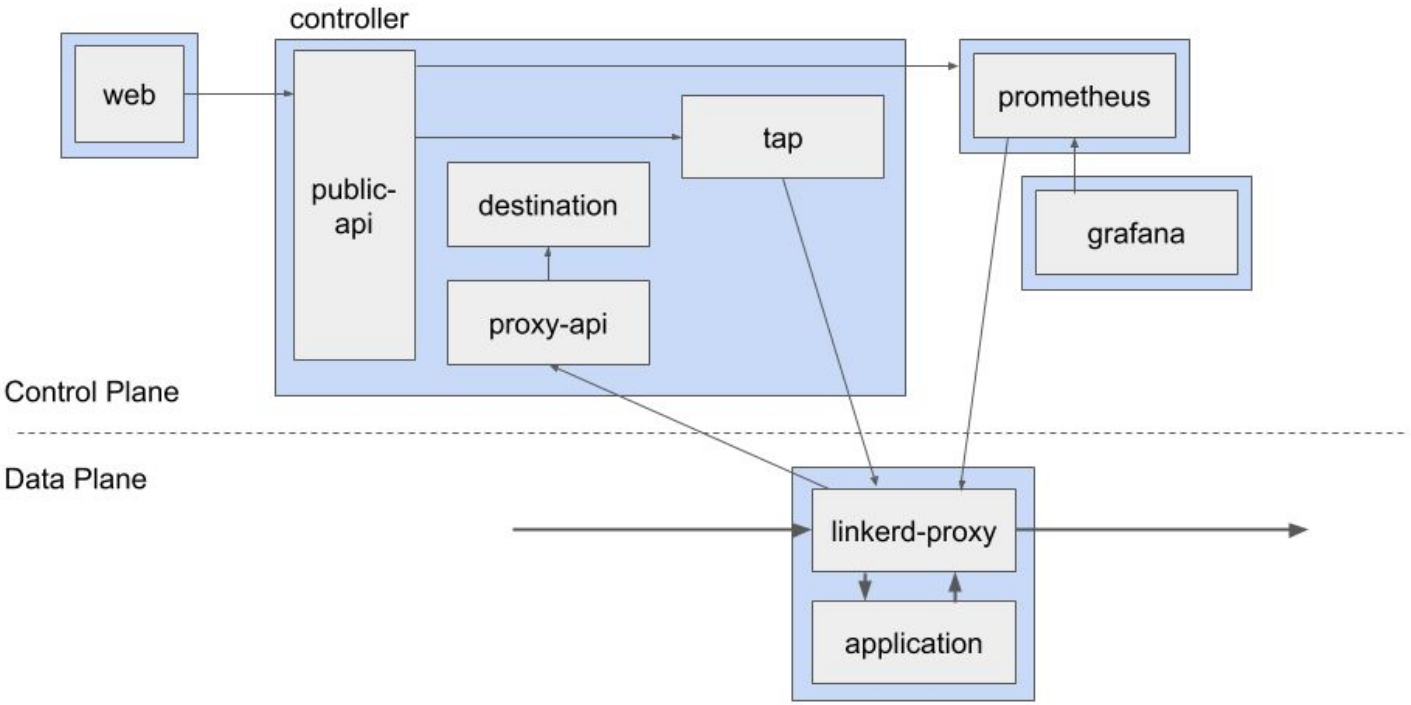
Linkerd's control plane installs into a single namespace, and services can be safely added to the mesh, one at a time.



Simple, minimalist design

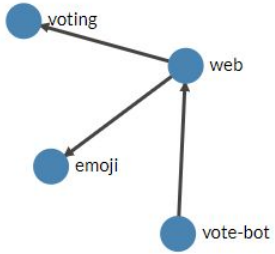
No complex APIs or configuration. For most applications, Linkerd will "just work" out of the box.

Linkerd arch



Linkerd UI

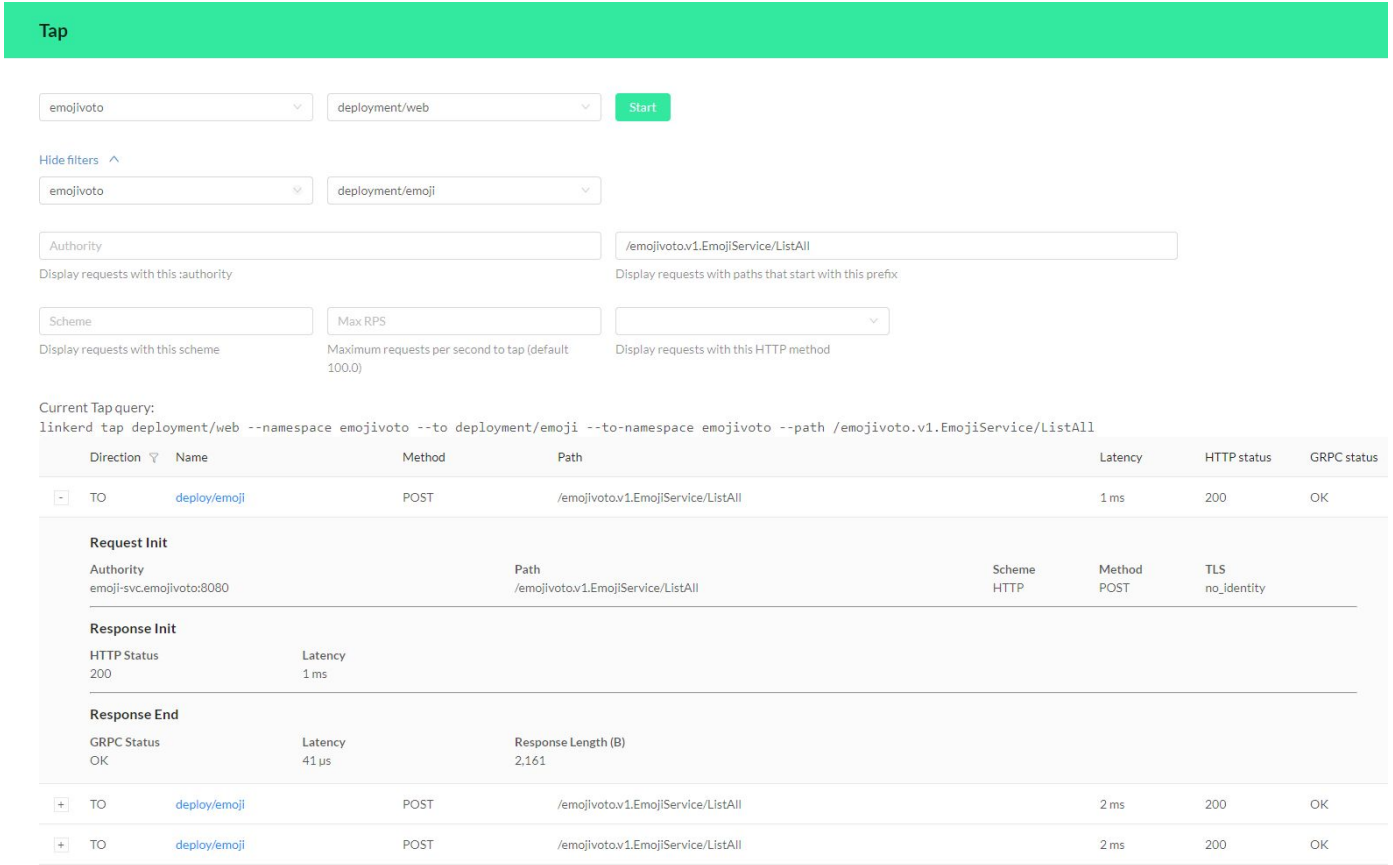
Namespace > emojivoto



Deployments

DEPLOYMENT	MESHED	SR	RPS	P50	P95	P99	TLS	DASH
emoji	1/1	100.00%	2	1 ms	1 ms	1 ms	0.0%	
vote-bot	1/1	---	---	---	---	---	---	
voting	1/1	81.67%	1	1 ms	1 ms	1 ms	0.0%	
web	1/1	90.83%	2	2 ms	3 ms	4 ms	0.0%	

Linkerd UI (cont.)



Linkerd与Istio的对比

Linkerd的也是通过sidecar来中转流量，以提供相应的附件功能，两者都是对业务无侵入的方式实现

Linkerd在监控方面做的更强大点，可观察具体的访问请求

Linkerd在其它方面目前尚未提供太多功能（如流量管理方面）

总结

Istio在服务管理上提供了更多功能(如流量新旧版本分离，A/B测试等)

Linkerd在监控方面做的更强大(具体到查看每个访问的状态)。

通常两者对同一个应用不能同时使用，由于Istio的复杂性，及特殊业务场景的应用要求，初期可以考虑使Istio对某一应用选择性的支持，Linkerd的应用也须作针对性的选择(如对监控要求较高的)。

K8s部署service-mesh相关信息

http://issue.qianbao-inc.com/SRE/k8s/src/branch/master/service-mesh/istio

http://issue.qianbao-inc.com/SRE/k8s/src/branch/master/service-mesh/linkerd2

Reference

https://istio.io/

https://linkerd.io

https://opensource.com/article/18/9/what-istio

Thank you !