SELA|DEVELOPER|PRACTICE

December 6-8, 2020

Leon Jalfon

Service Mesh Fundamentals

Agenda

- Workshop Introduction
- Understanding Service Mesh
- Service Mesh Fundamentals with Linkerd
- Service Mesh Fundamentals with Istio
- Service Mesh Interface (SMI)
- Summary

Workshop Objectives

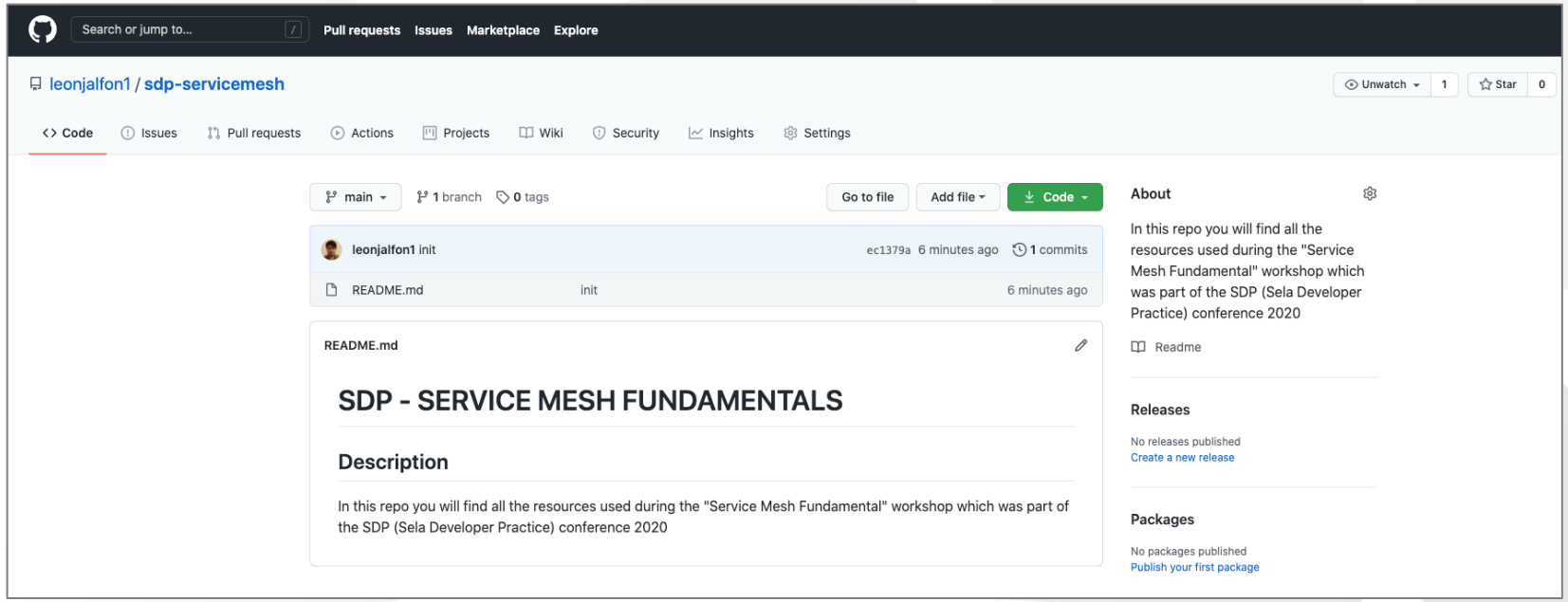
- Getting started with service mesh on Kubernetes
- Understand how service mesh works
- Learn Linkerd and Istio through hands-on labs
- And most importantly, let's have fun!

Knowledge Assumptions

- Linux knowledge
- Networking understanding
- Hands-on experience with Docker (or another container technology)
- Hands-on experience with Kubernetes (especially regarding networking)
- Experience managing Microservices on Kubernetes
- Kubernetes CRD's and sidecar pattern understanding

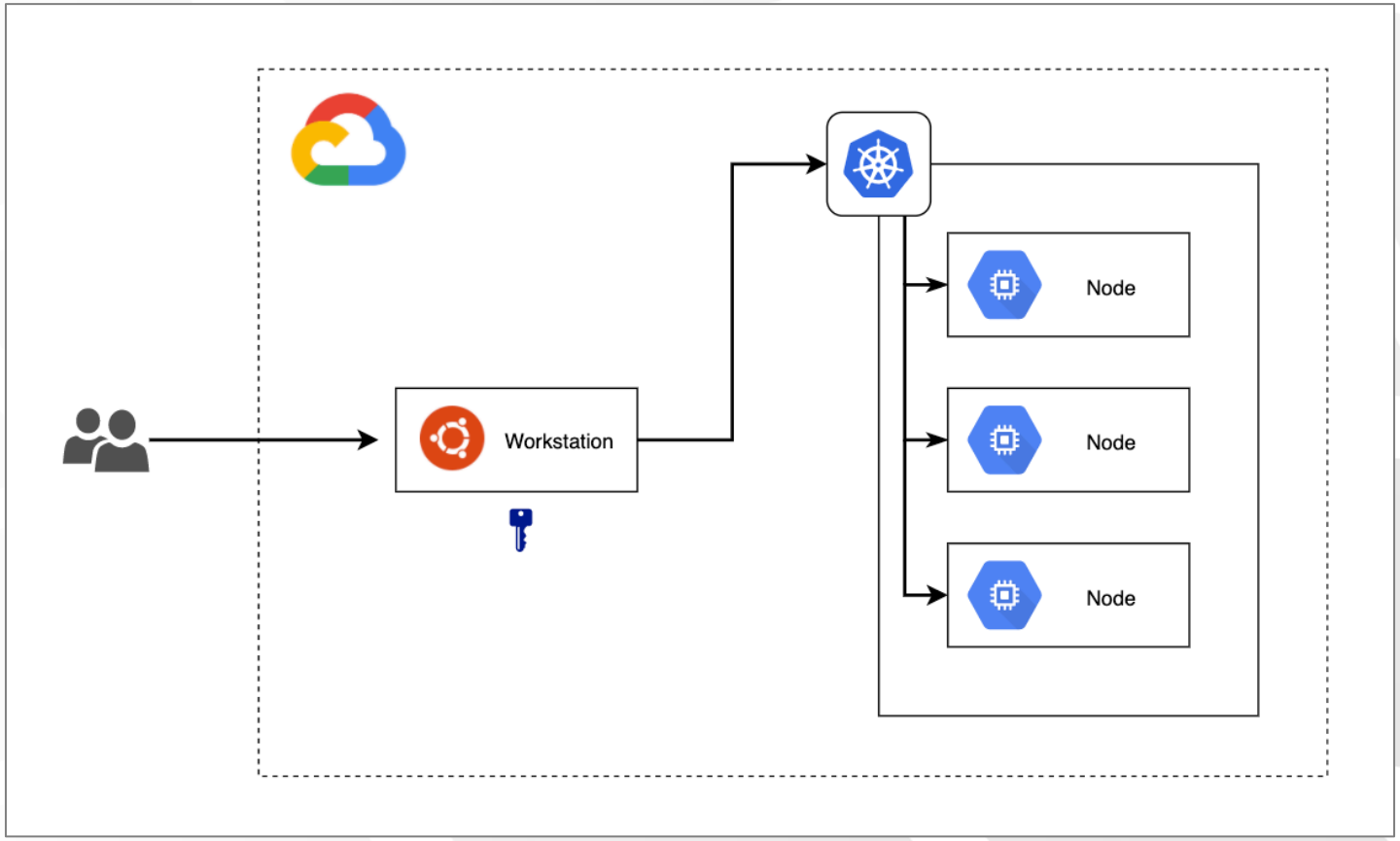
Workshop Resources

- You can find all the resources used during the workshop in my GitHub



https://github.com/leonjalfon1/sdp-servicemesh

Lab Infrastructure



Workshop Hand-On Labs - Linkerd

- LAB 01: Installing Linkerd
- LAB 02: Exploring Linkerd
- LAB 03: Debugging with Linkerd
- LAB 04: Validating Linkerd mTLS
- LAB 05: Deploying and Debugging the BookApp
- LAB 06: Working with Service Profiles

Workshop Hand-On Labs - Istio

- LAB 07: Installing Istio
- LAB 08: Exploring Istio
- LAB 09: Intelligent Routing with Istio
- LAB 10: Fault Injection with Istio
- LAB 11: Traffic Management with Istio
- LAB 12: Setup Istio Authorization


LETS GET STARTED
makeameme.org


SERVICE MESH
ASPEN MESH
Istio
Open
Service
Mesh
HashiCorp
Consul
LIGHTYEAR
maesh
LINKERD
SERVICE MESH EVERYWHERE
imgflip.com

What is a Service Mesh?

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help

Service mesh

From Wikipedia, the free encyclopedia

This article has multiple issues. Please help **improve it** or discuss these issues on the **talk page**. *(Learn how and when to remove these template messages)* [show]

In software architecture, a **service mesh** is a dedicated infrastructure layer for facilitating service-to-service communications between microservices, often using a sidecar proxy.

Having such a dedicated communication layer can provide a number of benefits, such as providing observability into communications, providing secure connections, or automating retries and backoff for failed requests.

Understanding Service Mesh

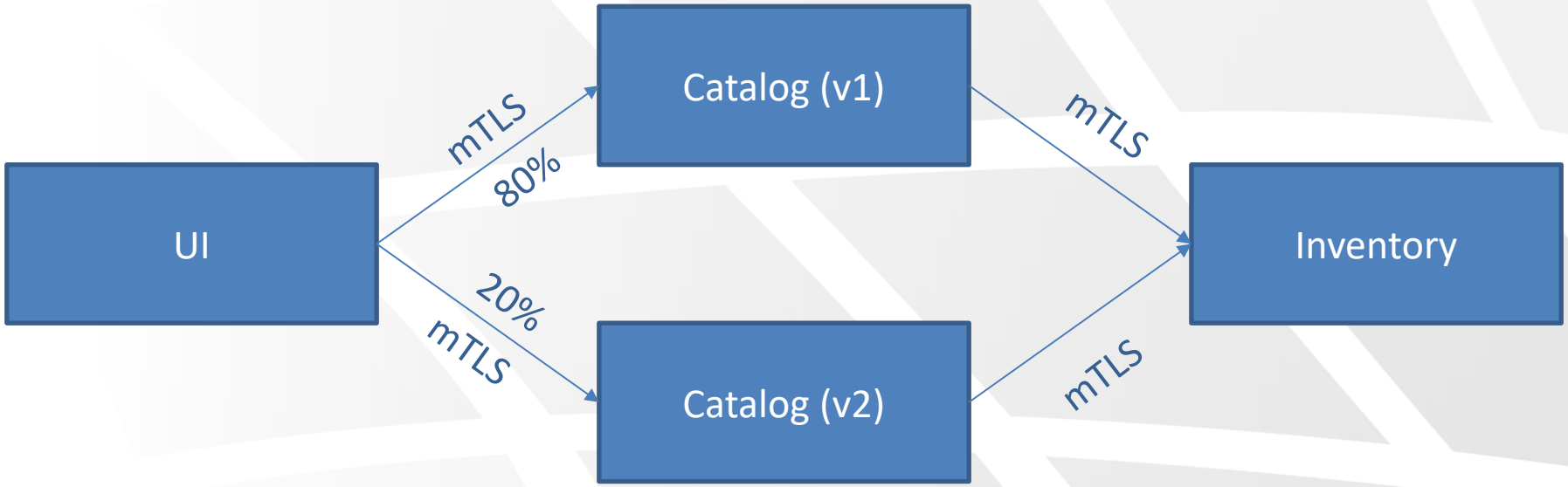
Common application without using a service mesh

Understanding Service Mesh

Improve Security → Automatic mutual TLS

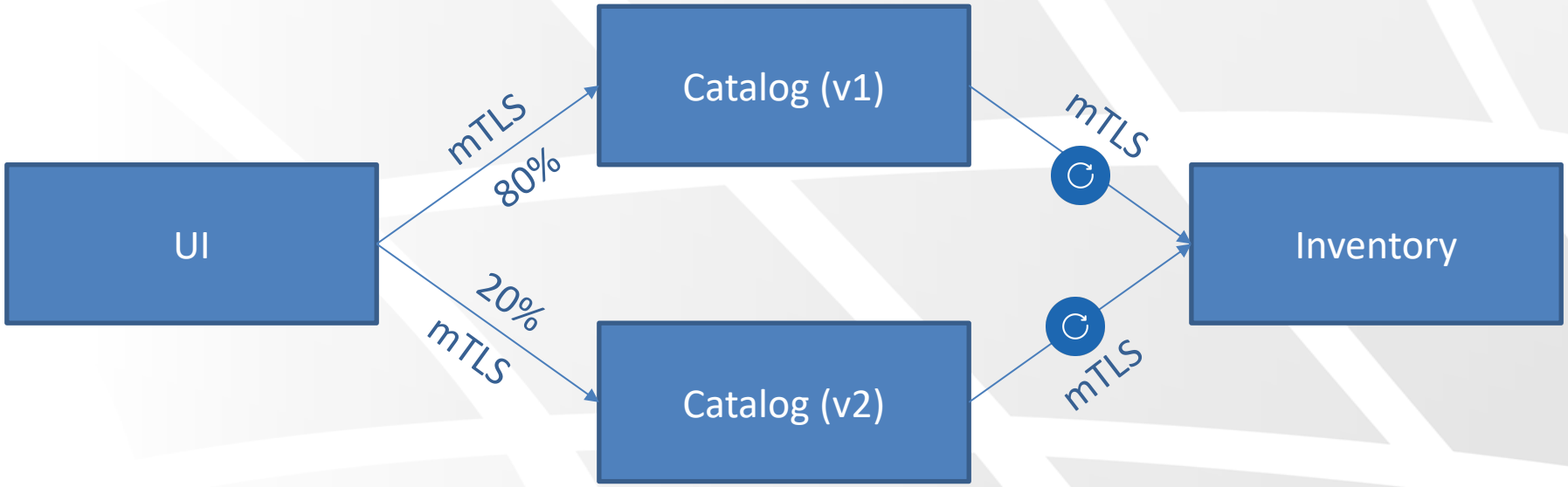
Understanding Service Mesh

Improve Networking → Intelligent Routing (canary deployments, blue/green, etc)



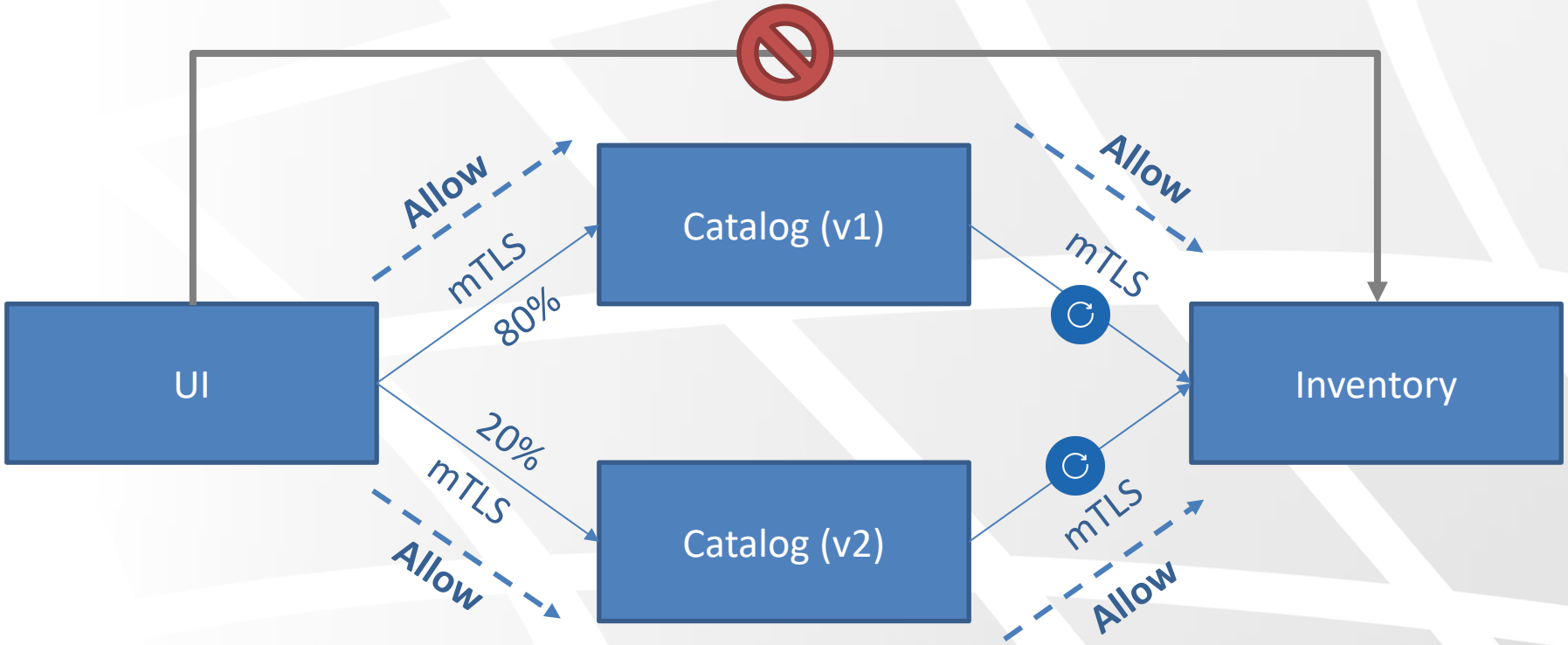
Understanding Service Mesh

Improve Networking → Network Policies (retries, timeouts, fault injections, etc)



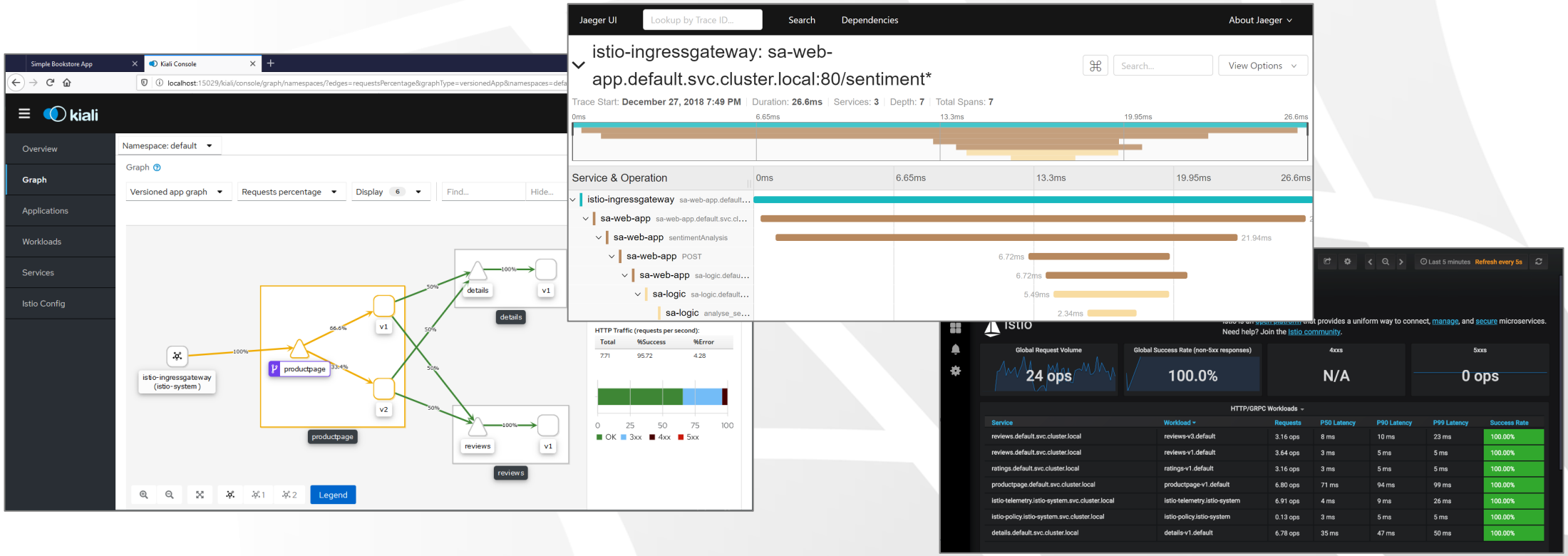
Understanding Service Mesh

Improve Control → Authorization Rules

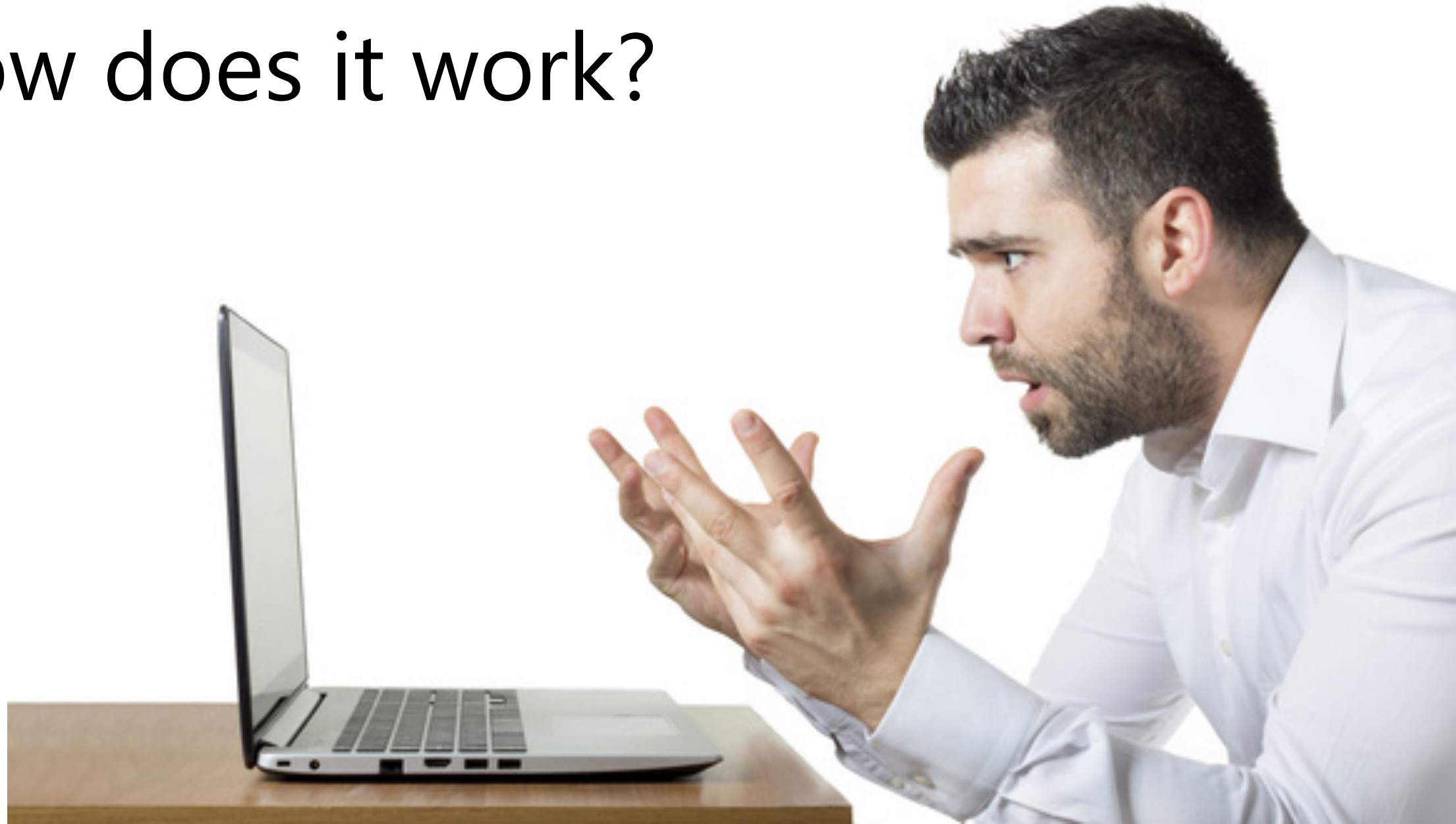


Understanding Service Mesh

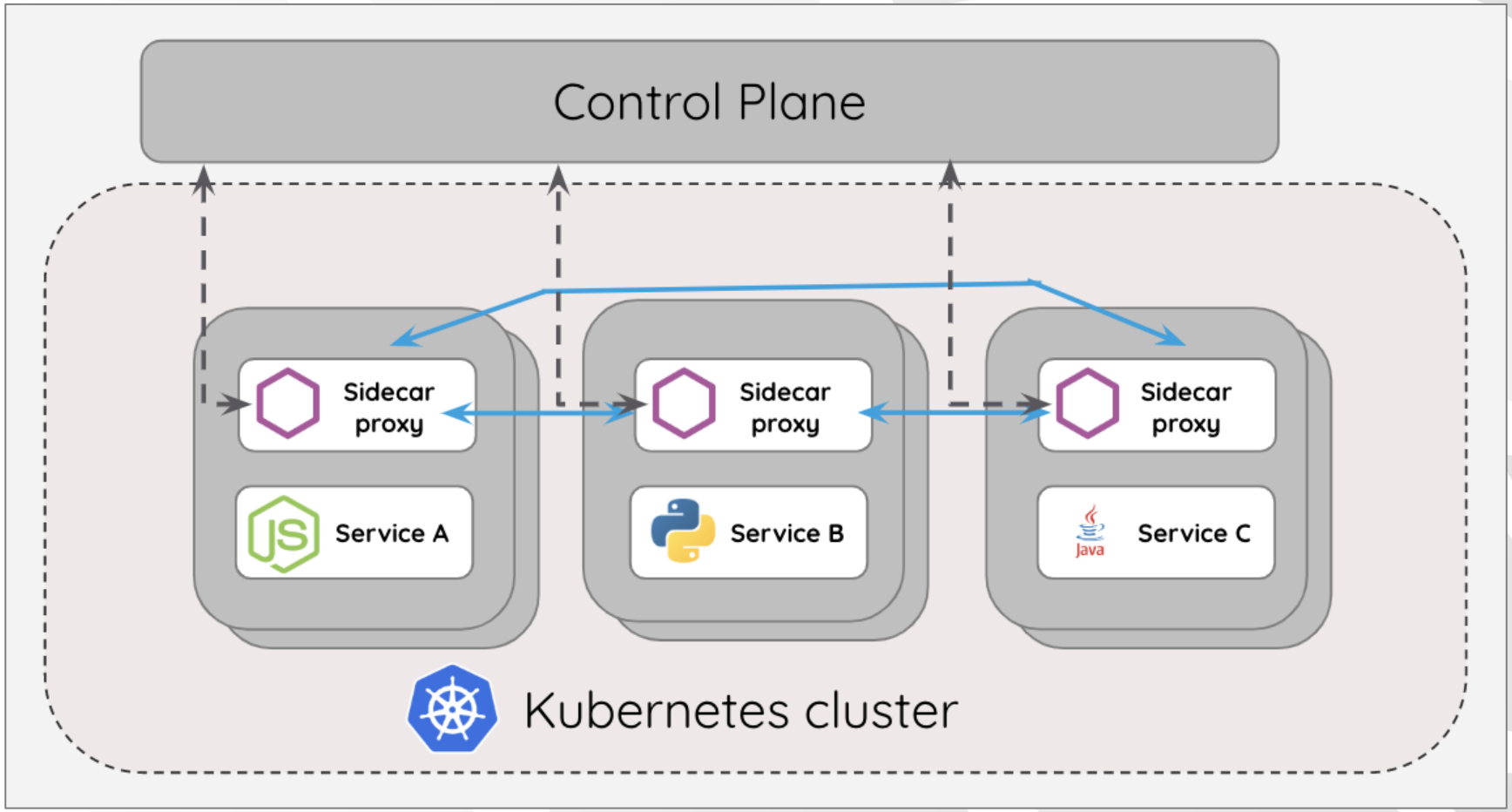
Observability→ Visibility, Monitoring, Tracing, etc



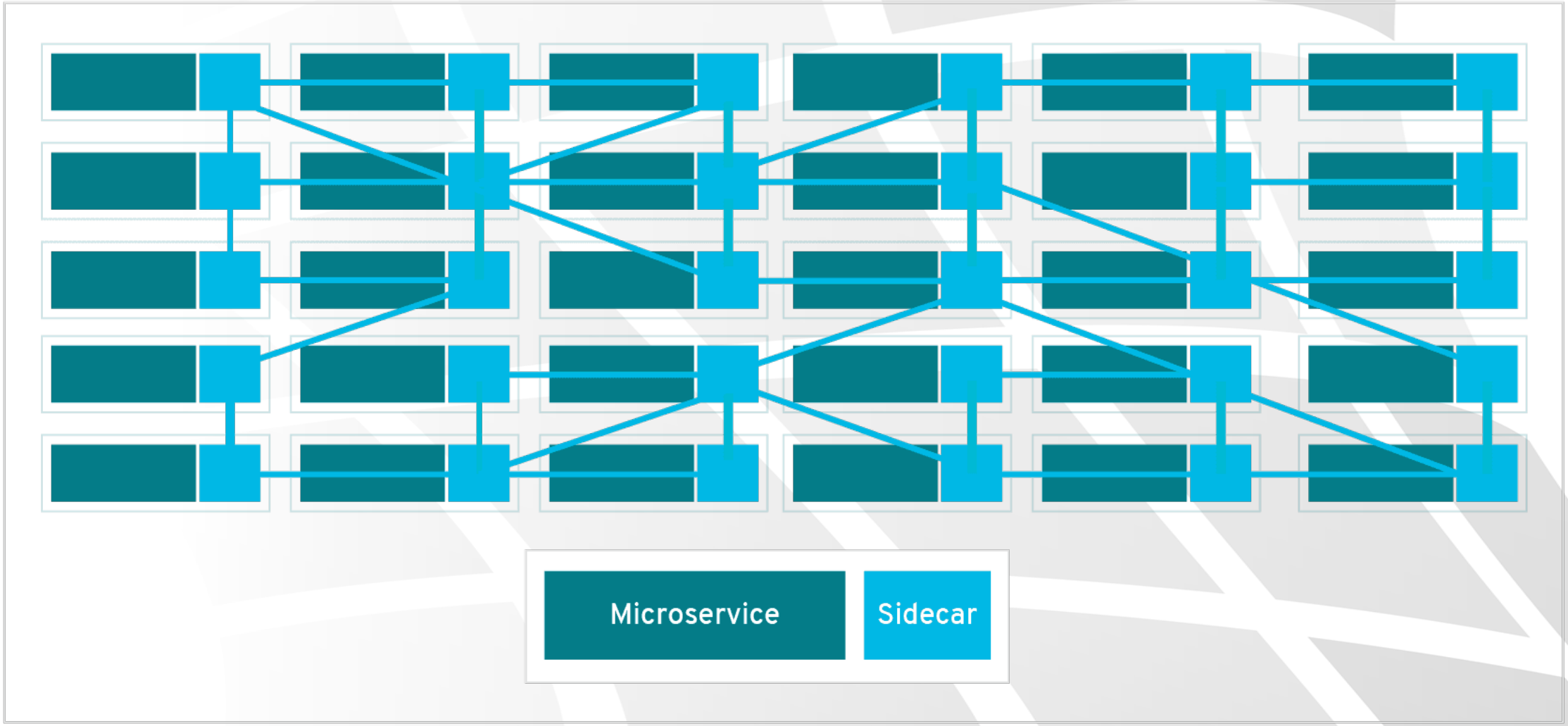
How does it work?



Understanding Service Mesh

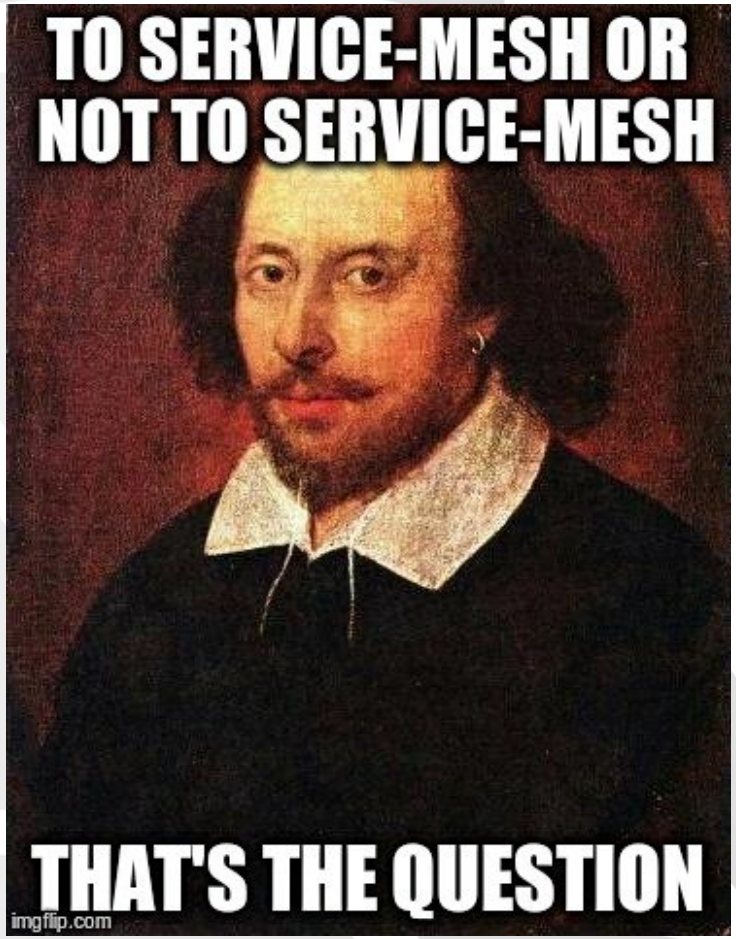


Understanding Service Mesh



Why use a Service Mesh?

- Automatic load balancing
- Fine-grained control of traffic
- Pluggable policy layer
- Access controls, rate limits and quotas.
- Service discovery
- Service monitoring with automatic metrics, logs and traces for all traffic.
- Secure service to service communication



Service Mesh Implementations

Istio

If you have heard about service mesh, you have probably heard about Istio too. Istio is by far the most popular service mesh because of its rich feature set and it Google's and IBM's support.

Linkerd 2

While Istio made the service mesh popular, Linkerd was the first service mesh and quite successful. Still, the developers decided to build a new version - Linkerd 2 - committed to usability, performance, and Kubernetes as the underlying platform.

Consul/Consul Connect

HashiCorp's Consul has been well known as a service discovery solution for a long time. Now that it has adopted the Envoy proxy and Sidecar pattern, Consul can serve as a service mesh for a variety of platforms like Kubernetes and VMs.

AWS App Mesh

Not long after the service mesh hype, AWS added its own service mesh for applications on AWS.

Traefik Mesh

As the name already reveals, Traefik Mesh (formerly Maesh) is the service mesh based the cloud-native API gateway Traefik.

Kuma

Similar to Traefik Mesh, Kuma is also a very new service mesh made by developers of an API gateway - Kong.

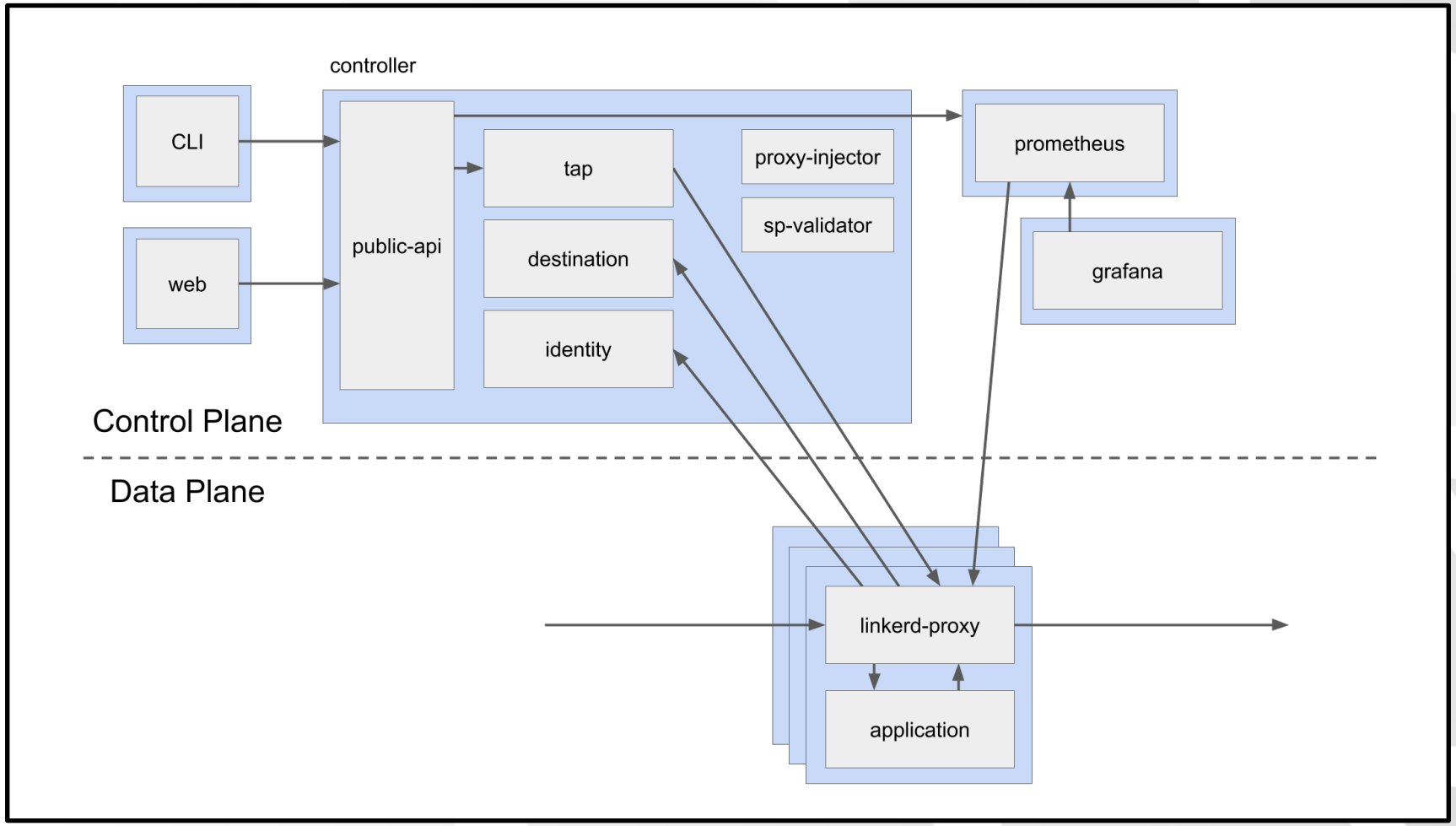
Open Service Mesh (OSM)

A new implementation by Microsoft, following common service mesh design principles like adopting envoy and implementing SMI spec.

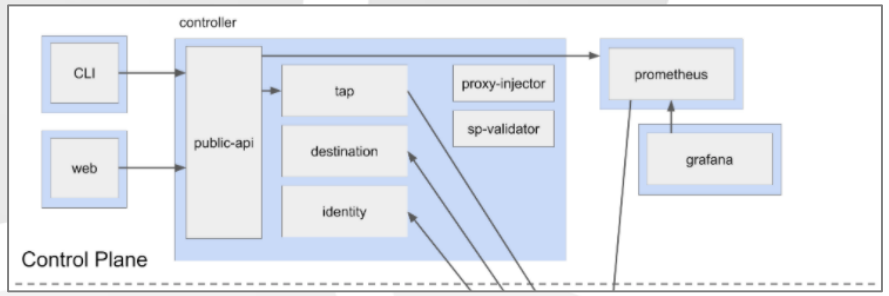
About Linkerd

- Linkerd is focused on simplicity, speed, and low resource usage
- Linkerd is built for security from the ground up (on-by-default mTLS, a data plane that is built in a memory-safe language, regular security audits, etc)
- Linkerd is committed to open governance and is hosted by a neutral foundation (CNCF)
- Use the linkerd2-proxy, which is built specifically for the service mesh sidecar use case, (smaller and faster than envoy-based service meshes)

Linkerd Architecture

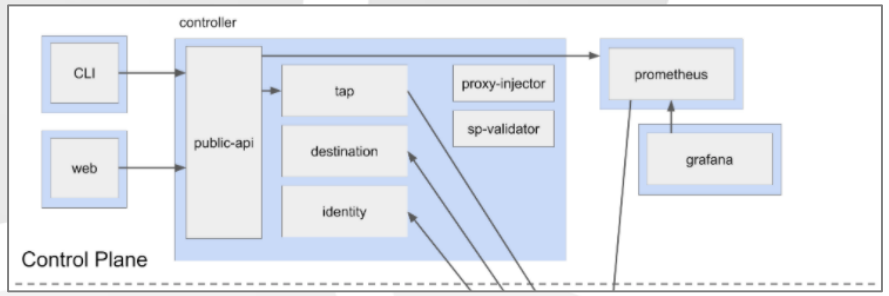


Architecture - Control Plane



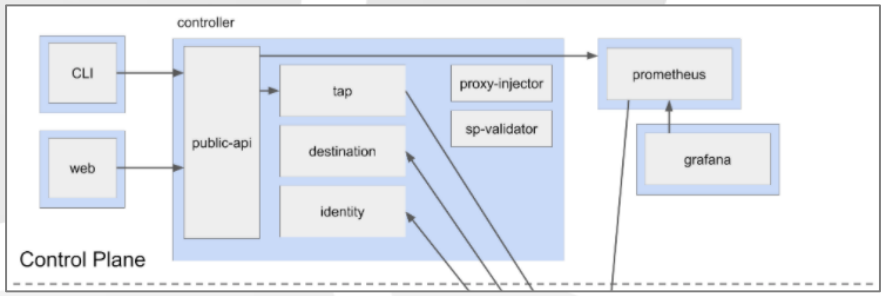
- **Web:** The web deployment provides the Linkerd dashboard. This does not require running Linkerd dashboard and can be exposed to others.
- **Public API:** The controller deployment consists of the public-api container that provides an API for the CLI and dashboard to interface with.
- **Tap:** The tap deployment receives requests from the CLI and dashboard to watch requests and responses in real time.

Architecture - Control Plane



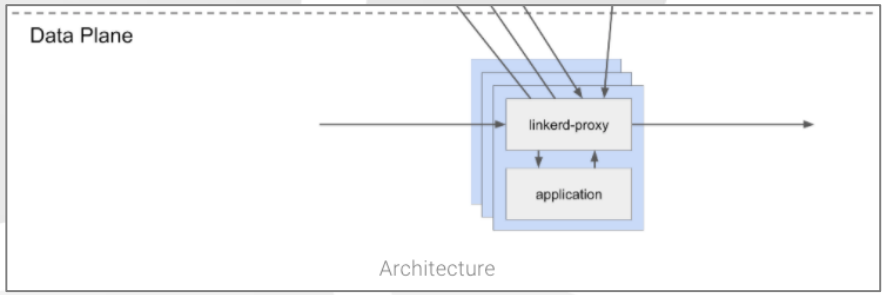
- **Destination:** Each proxy in the data plane uses this component to lookup where to send requests.
- **Identity:** Provides a Certificate Authority that accepts CSRs from proxies and returns certificates signed with the correct identity.
- **Proxy Injector:** The injector is an admission controller, which receives a webhook request every time a pod is created.

Architecture - Control Plane



- **Service Profile Validator:** The validator is also an admission controller, which validates new Service Profiles (Linkerd CRD) before they are saved.
- **Grafana:** Linkerd comes with many dashboards out of the box. The Grafana component is used to render and display these dashboards.
- **Prometheus:** Prometheus is a cloud native monitoring solution that is used to collect and store all of the Linkerd metrics.

Architecture - Data Plane



- **Proxy:** An ultralight transparent proxy written in Rust, the proxy is installed into each pod of a service and becomes part of the data plane. It receives all incoming traffic for a pod and intercepts all outgoing traffic via an initContainer that configures iptables to forward the traffic correctly. Because it is a sidecar and intercepts all the incoming and outgoing traffic for a service, there are no code changes required and it can even be added to a running service.

Linkerd CLI

The Linkerd CLI is the primary way to interact with Linkerd. It can install the control plane to your cluster, add the proxy to your service and provide detailed metrics for how your service is performing.

check	Check the Linkerd installation for potential problems
completion	Output shell completion code for the specified shell (bash or zsh)
dashboard	Open the Linkerd dashboard in a web browser
edges	Display connections between resources, and Linkerd proxy identities
endpoints	Introspect Linkerd's service discovery state
get	Display one or many mesh resources
inject	Add the Linkerd proxy to a Kubernetes config
install	Output Kubernetes configs to install Linkerd
install-cni	Output Kubernetes configs to install Linkerd CNI
install-sp	Output Kubernetes configs to install Linkerd Service Profiles

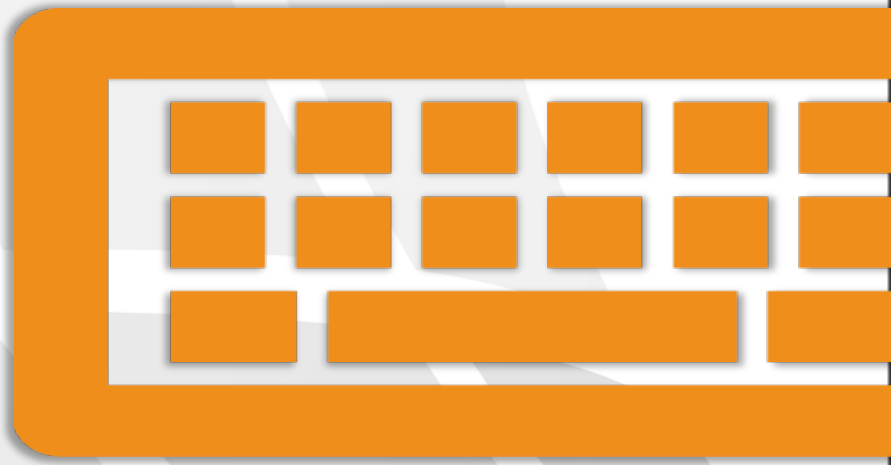
logs	Tail logs from containers in the Linkerd control plane
metrics	Fetch metrics directly from Linkerd proxies
profile	Output service profile config for Kubernetes
routes	Display route stats
stat	Display traffic stats about one or many resources
tap	Listen to a traffic stream
top	Display sorted information about live traffic
uninject	Remove the Linkerd proxy from a Kubernetes config
upgrade	Output Kubernetes configs to upgrade an existing Linkerd control plane
version	Print the client and server version information

Linkerd Installation

- Ensure you have access to a Kubernetes cluster running 1.13 or later
- Ensure you have a functioning kubectl command on your local machine
- Install the Linkerd CLI
- Validate your Kubernetes Cluster
- Install Linkerd onto the Cluster
- Validate the Installation

Lab 01: Installing Linkerd

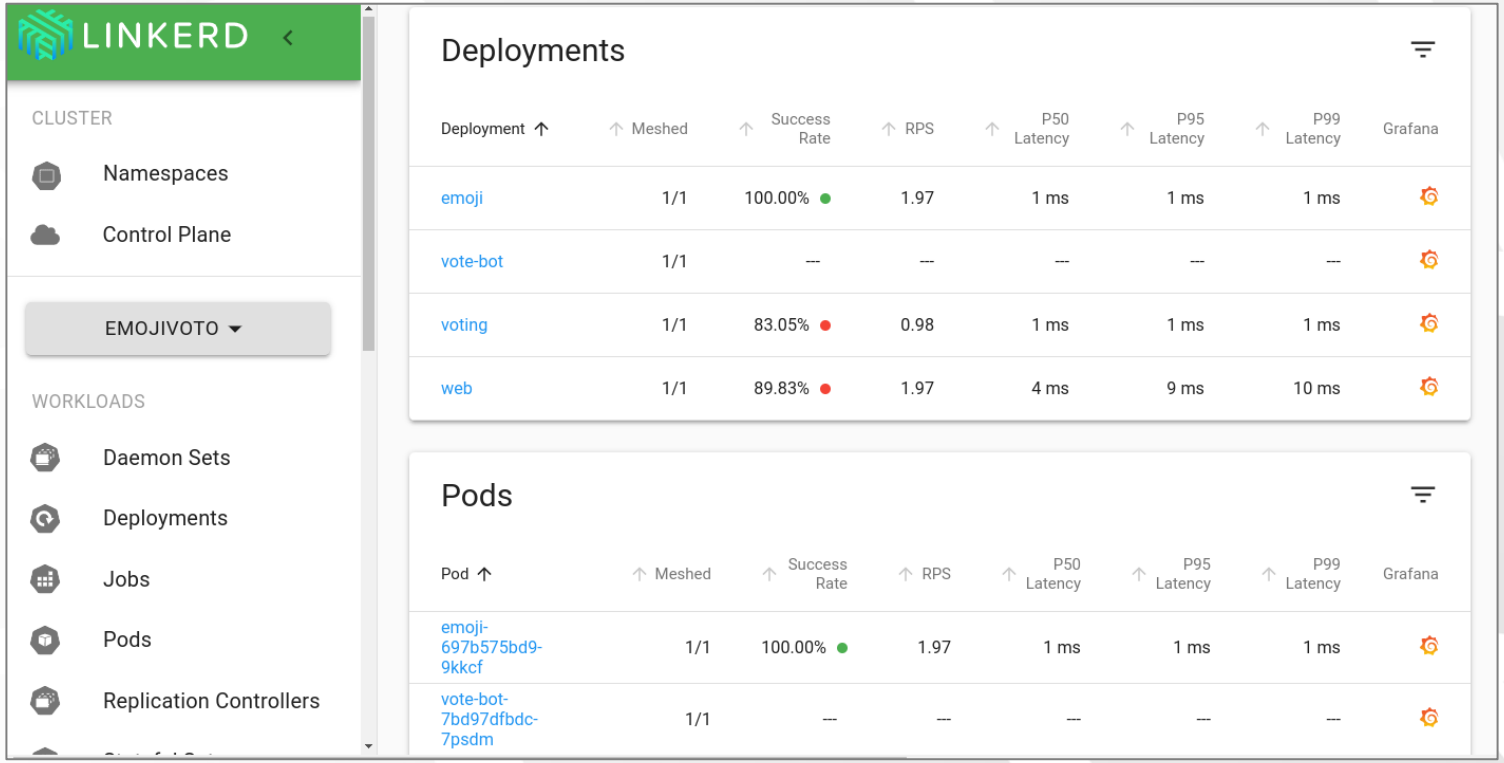
Lab



https://github.com/leonjalfon1/sdp-servicemesh/labs/01-installing-linkerd.md

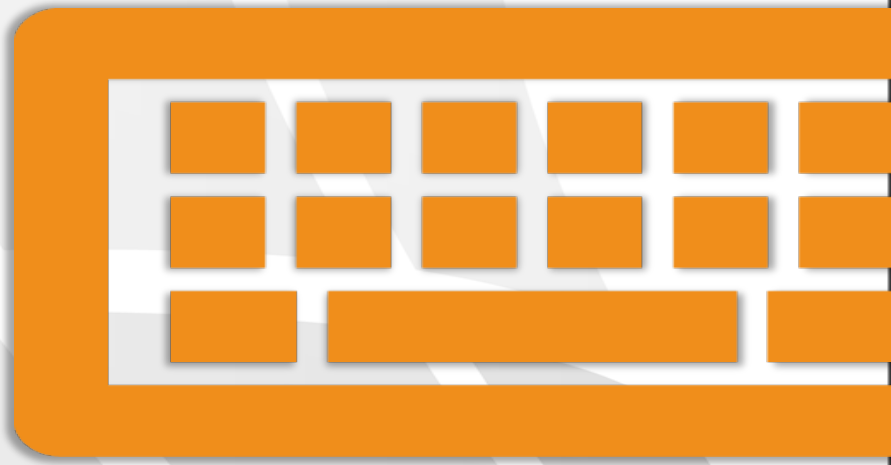
Linkerd Dashboard

The Linkerd dashboard provides a high-level view of what is happening with your services in real time.



Lab 02: Exploring Linkerd

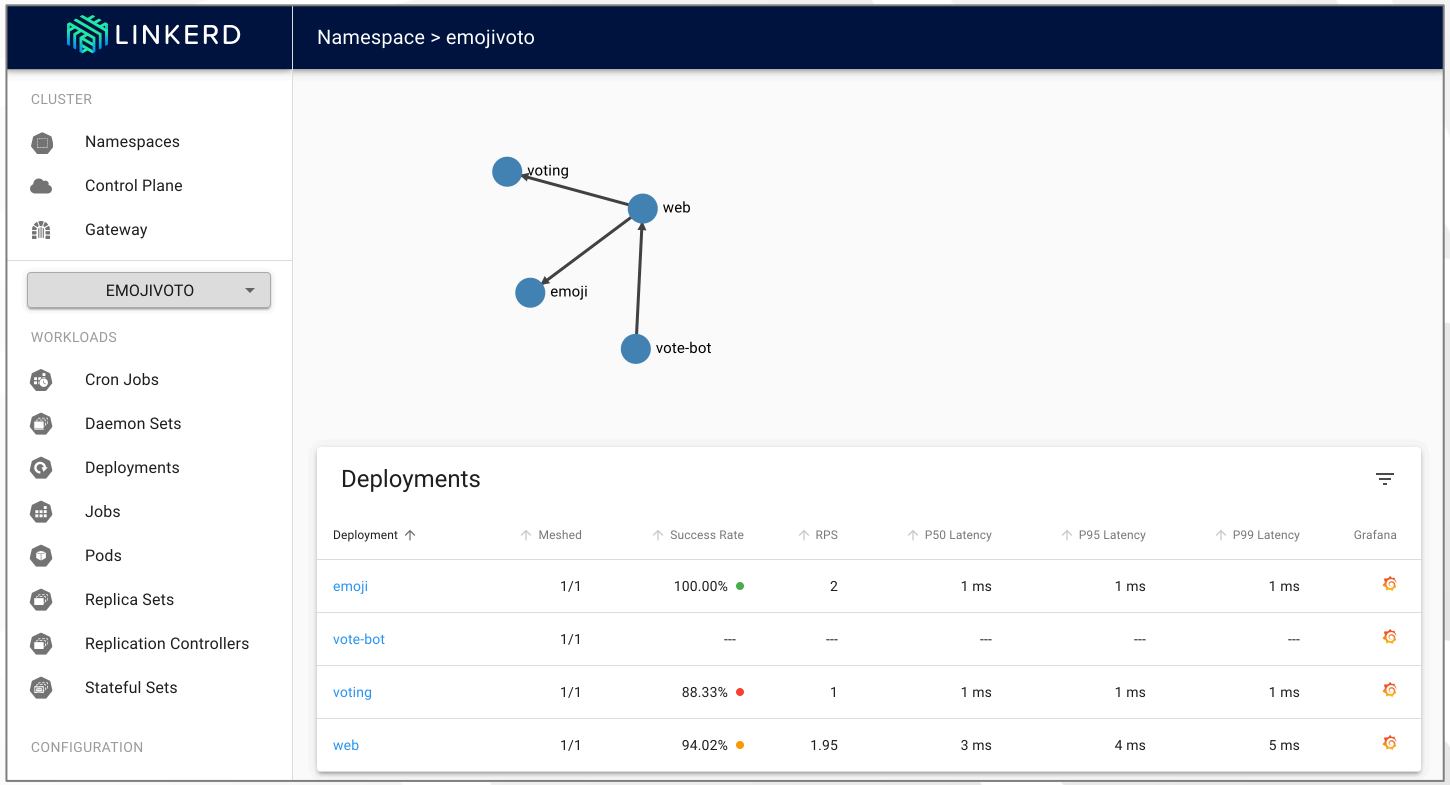
Lab



https://github.com/leonjalfon1/sdp-servicemesh/labs/02-exploring-linkerd.md

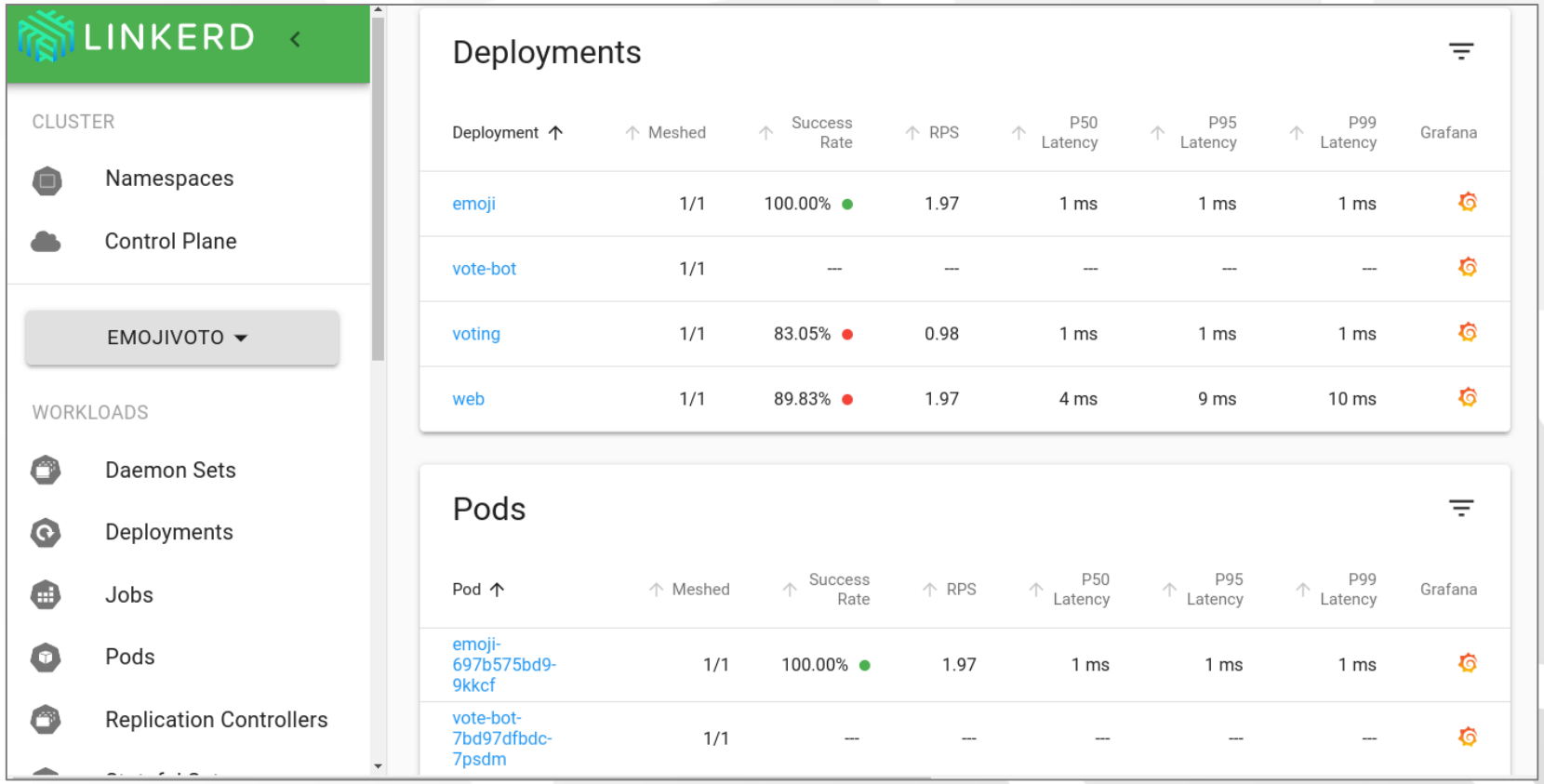
Linkerd Dashboard

The Linkerd dashboard provides a high-level view of what is happening with your services in real time.



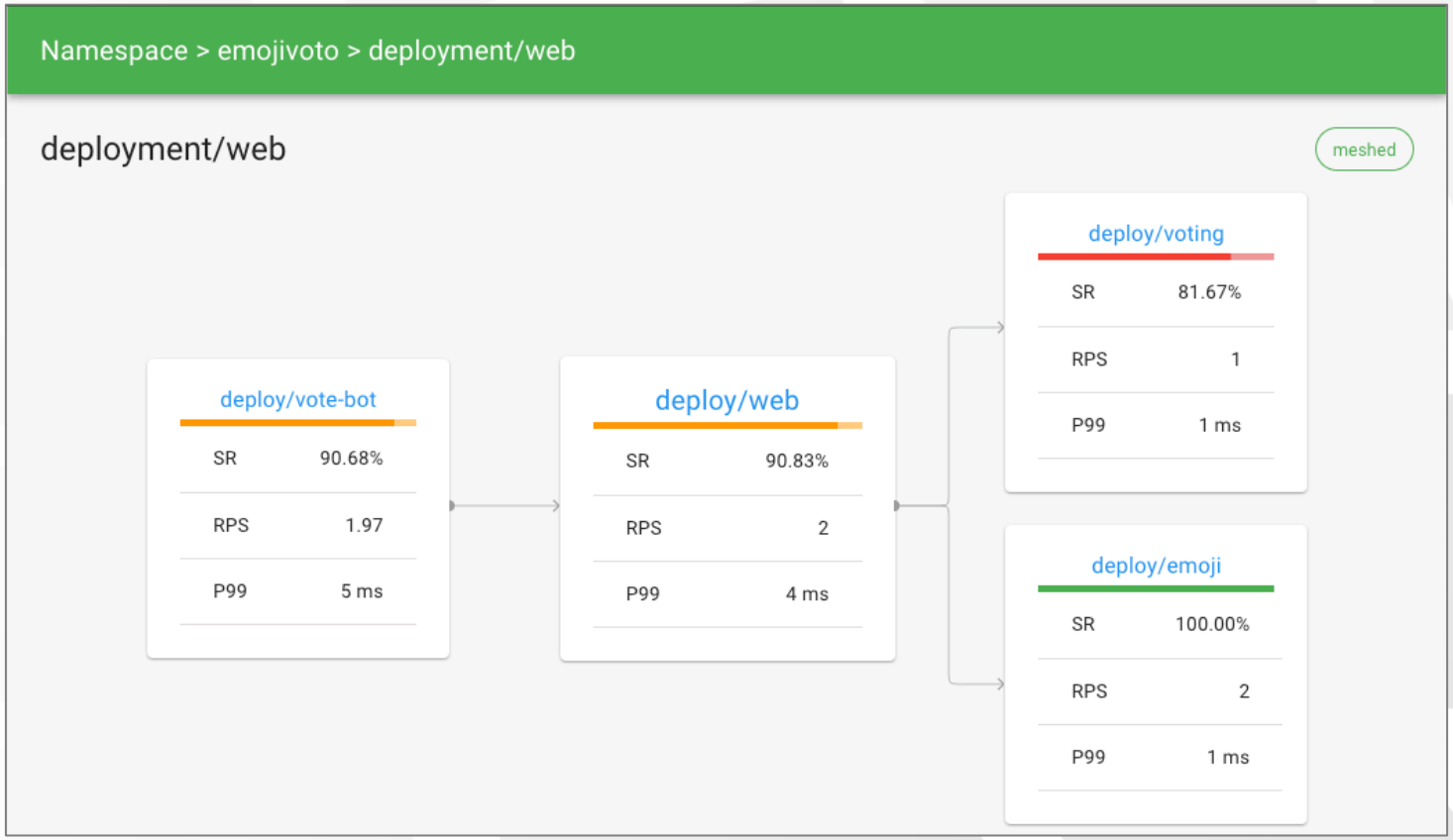
Linkerd Dashboard

Show the “golden” metrics for each deployment



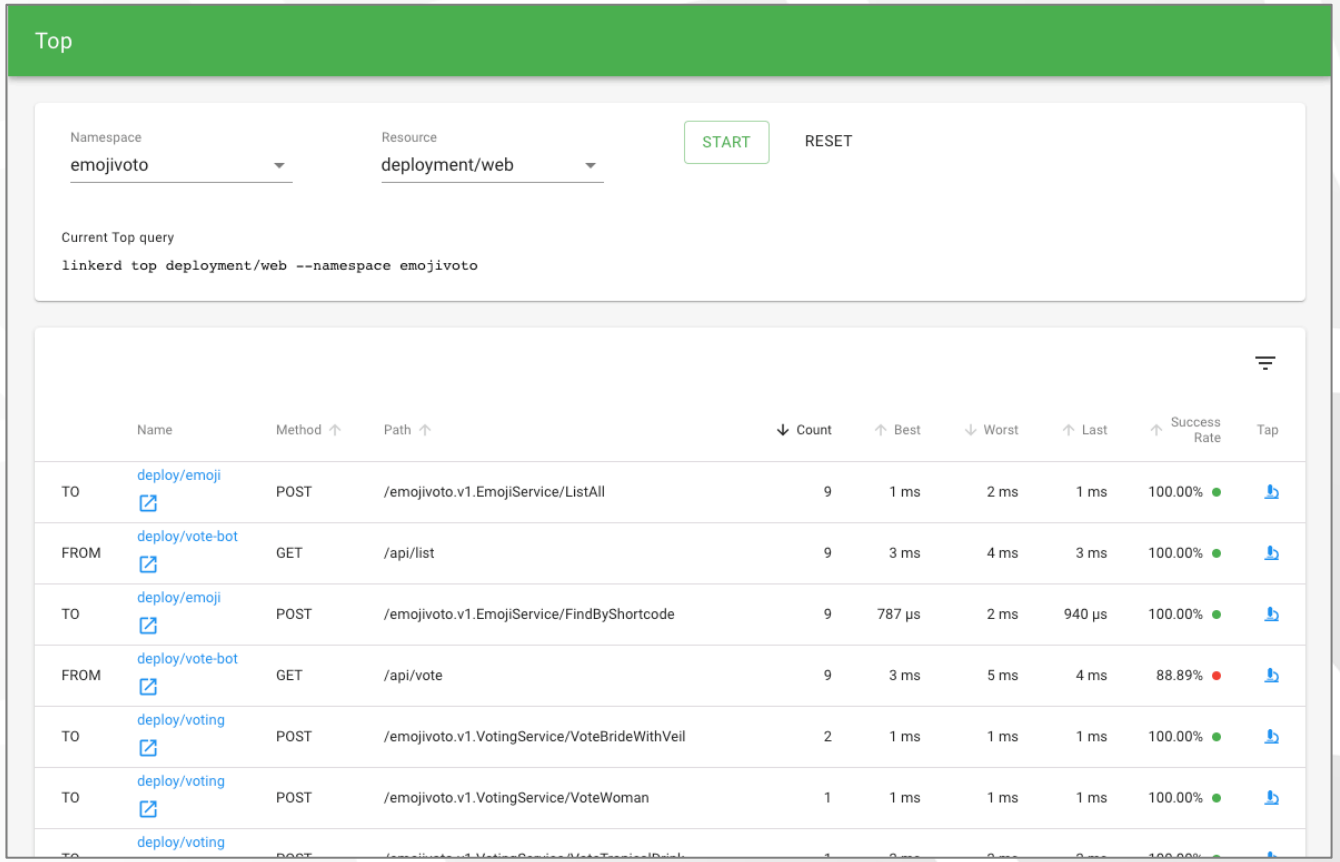
Linkerd Dashboard

Show deployment details (visualization)



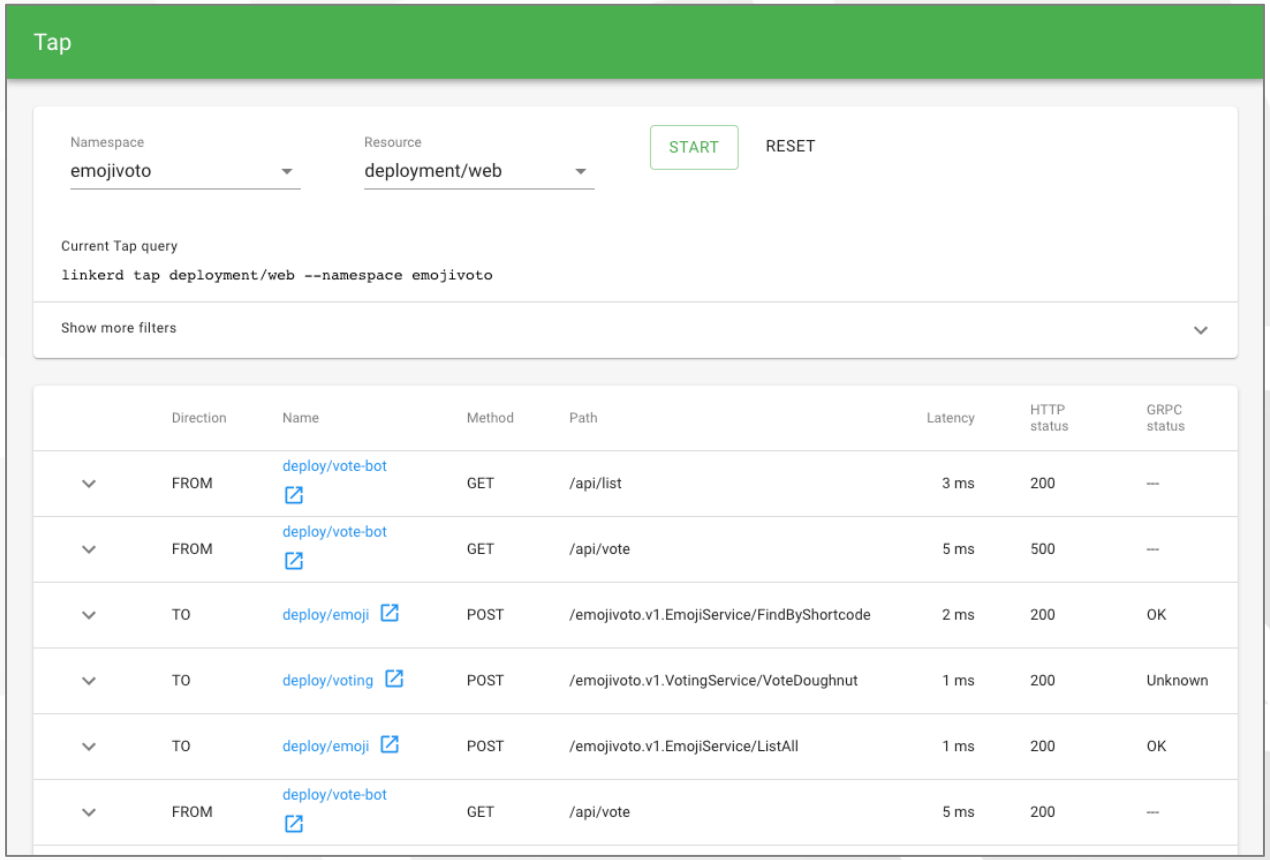
Linkerd Dashboard

Top: Display sorted information about live traffic on a per-route basis.



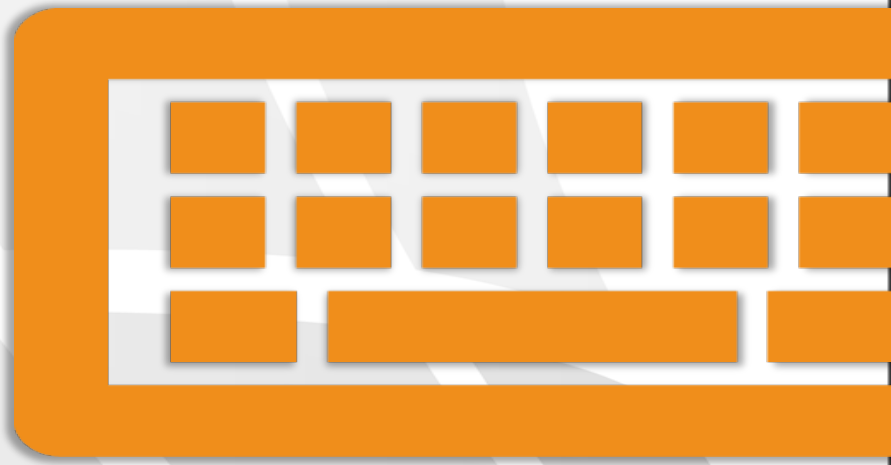
Linkerd Dashboard

Tap: Listen to a traffic stream for a resource



Lab 03: Debugging with Linkerd

Lab

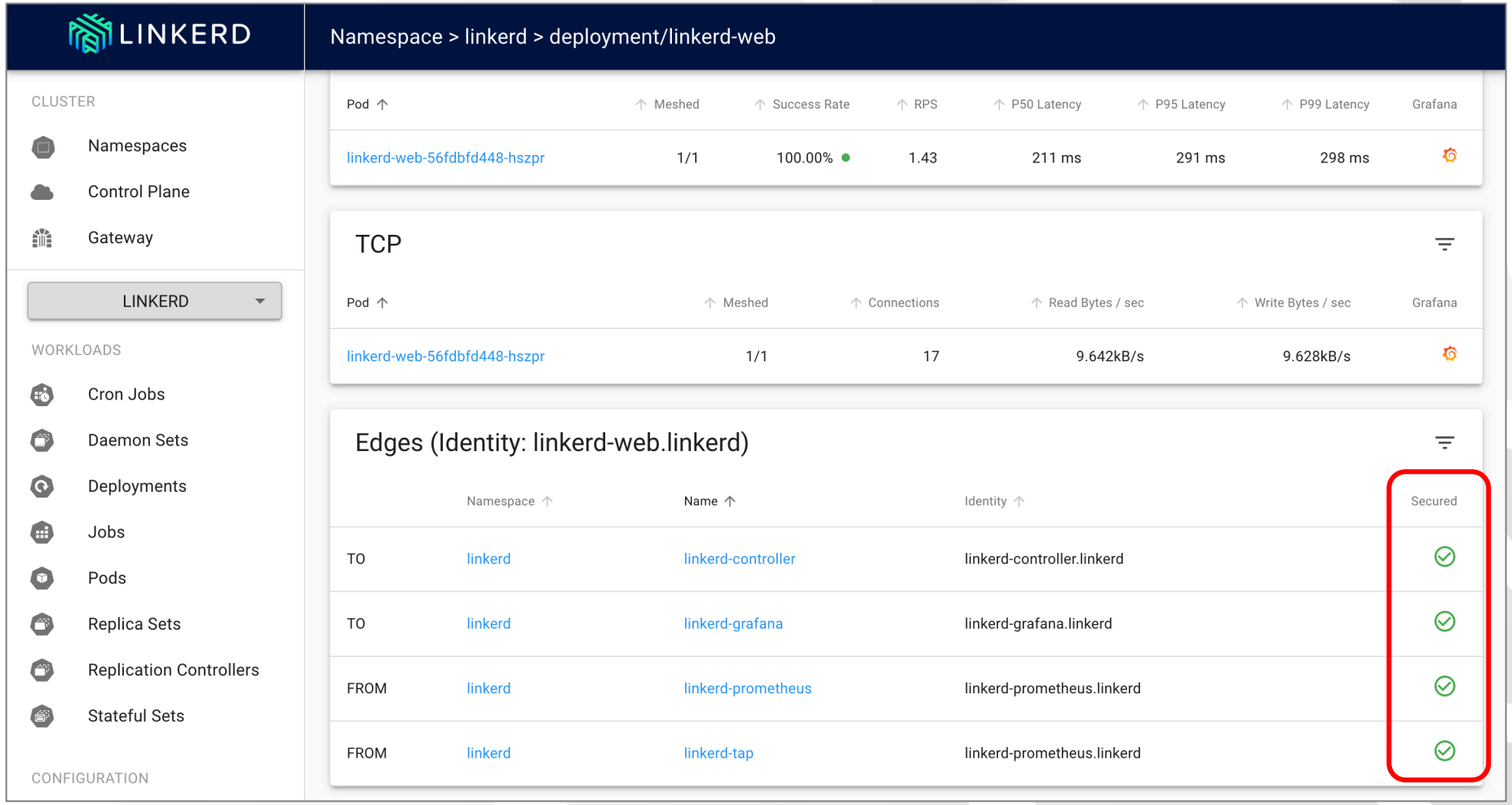


https://github.com/leonjalfon1/sdp-servicemesh/labs/03-debugging-with-linkerd.md

Automatic Mutual TLS

- By default, Linkerd automatically enables mutual Transport Layer Security (mTLS) for most TCP traffic between meshed pods, by establishing and authenticating secure, private TLS connections between Linkerd proxies.
- Because the Linkerd control plane also runs on the data plane, this means that communication between Linkerd's control plane components are also automatically secured via mTLS
- Not all traffic can be automatically mTLS'd, but it's easy to verify which traffic is (using linkerd tap)

Automatic Mutual TLS – Linkerd Edges

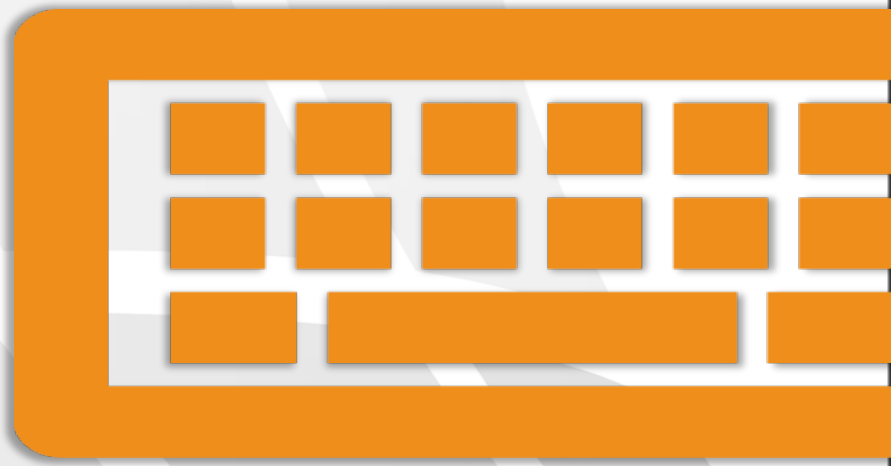


Automatic Mutual TLS - Considerations

- By default, the issuer certificate and key are not automatically rotated (expires after 365). You can set up automatic rotation with cert-manager
- Linkerd does not currently enforce mTLS
- Any unencrypted requests inside the mesh will be opportunistically upgraded to mTLS.
- Any requests originating from inside or outside the mesh will not be automatically mTLS'd by Linkerd (this will be addressed in a future Linkerd release)

Lab 04: Validating Linkerd mTLS

Lab



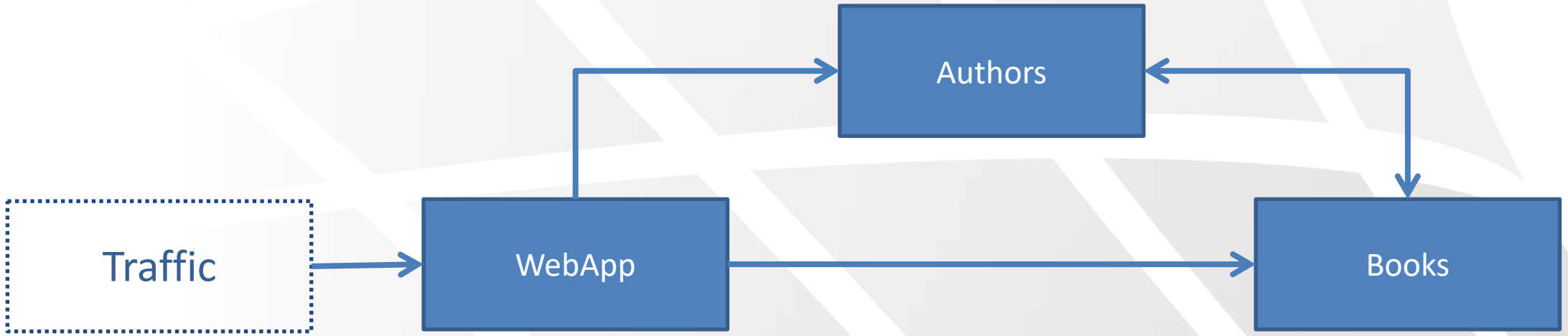
https://github.com/leonjalfon1/sdp-servicemesh/labs/04-validating-linkerd-mtls.md

BookApp Demo App

- Let's use the BookApp to learn the following topics:
  - Debug a network issue (a classic case of non-obvious, intermittent failure)
  - Configure Linkerd Service Profiles
  - Adding Retries
  - Adding Timeouts

BookApp - Architecture

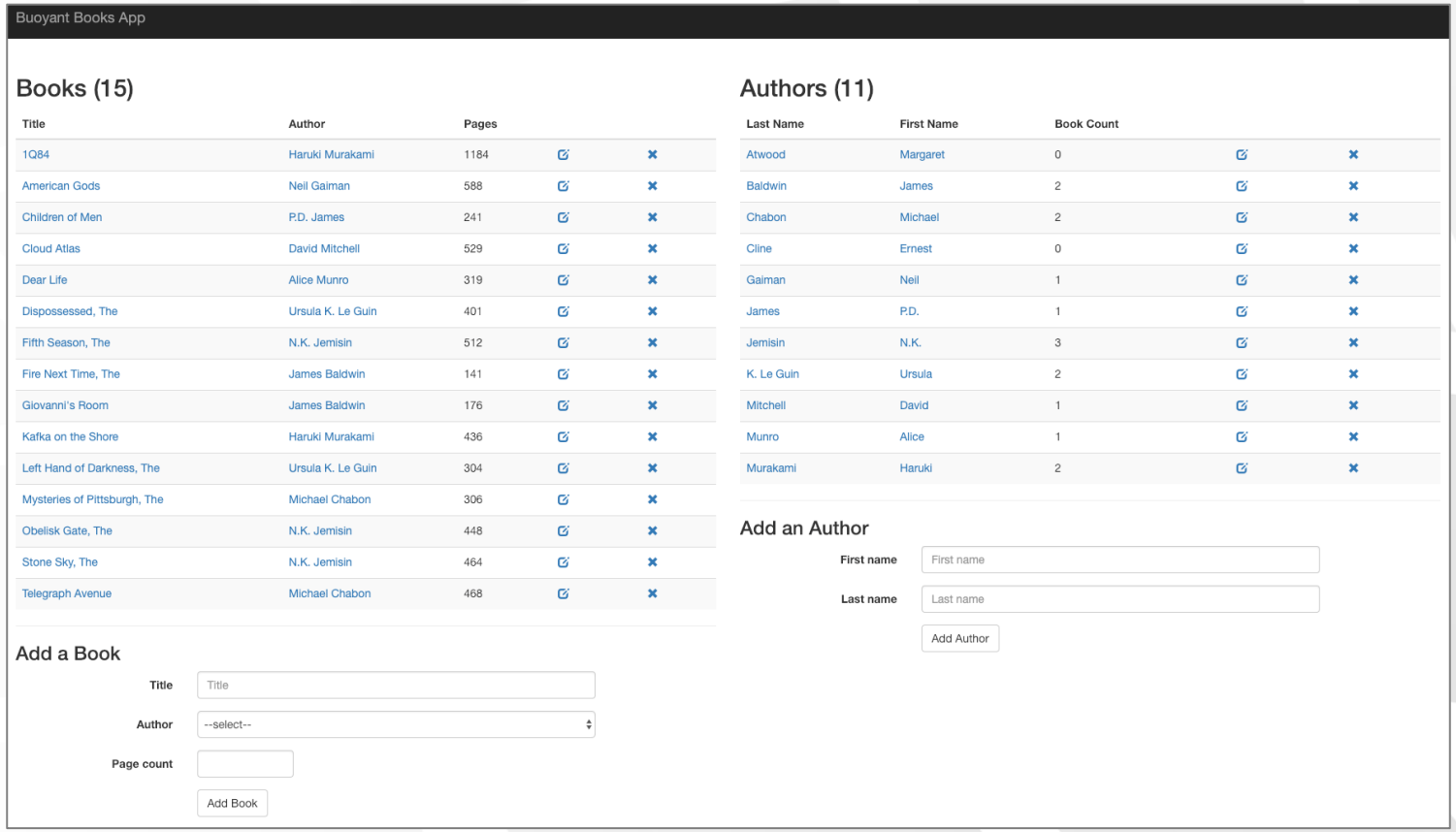
The BookApp is a Ruby application that helps you manage your bookshelf. It consists of multiple microservices and uses JSON over HTTP to communicate with the other services.



Note: for demo purposes, the app comes with a simple traffic generator

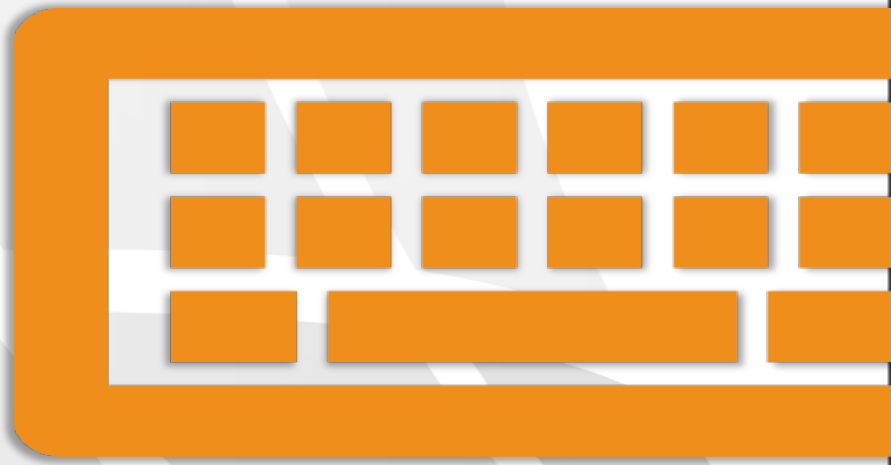
BookApp - Architecture

There is a bug in the app: if you click Add Book, it will fail 50% of the time



Lab 05: Deploying and Debugging the BookApp

Lab



https://github.com/leonjalfon1/sdp-servicemesh/labs/05-deploying-linkerd-bookapp.md

Services Profiles

- Is a custom Kubernetes resource (CRD) that can provide Linkerd additional information about a service.
- Service Profiles allows you to define a list of routes for the service
- Each route uses a regular expression to define which paths should match that route
- Defining a service profile enables Linkerd to report per-route metrics and also allows you to enable per-route features such as retries and timeouts

Services Profiles

```
apiVersion: linkerd.io/v1alpha2
kind: ServiceProfile
metadata:
  creationTimestamp: null
  name: webapp.booksapp.svc.cluster.local
  namespace: booksapp
spec:
  routes:
  - condition:
      method: GET
      pathRegex: /
    name: GET /
  - condition:
      method: POST
      pathRegex: /authors
    name: POST /authors
```

Services Profiles

- There are a couple different ways to use the linkerd profile command to create service profiles:
  - **Swagger:** from an OpenAPI (Swagger) spec for your service
  - **Protobuf:** from a protobuf format for your service
  - **Auto-Creation:** generate service profiles from watching live traffic (this is based off tap data)
  - **Template:** you can get a template that allows you to add routes manually

Services Profiles

- Before using a Service Profile:

```
linkerd -n booksapp tap deploy/webapp -o wide | grep req
```

```
req id=124:12 proxy=in  src=10.36.1.28:52398 dst=10.36.2.18:7000 tls=true :method=POST
:authority=webapp:7000 :path=/books src_res=deploy/traffic src_ns=booksapp dst_res=deploy/webapp
dst_ns=booksapp
```

- After using a Service Profile:

```
linkerd -n booksapp tap deploy/webapp -o wide | grep req
```

```
req id=0:1 proxy=in src=10.1.3.76:57152 dst=10.1.3.74:7000 tls=true :method=POST
:authority=webapp.default:7000 :path=/books/2878/edit src_res=deploy/traffic src_ns=booksapp
dst_res=deploy/webapp dst_ns=booksapp rt_route=POST /books/{id}/edit
```

Services Profiles - Retries

- To add a retry just add "isRetryable" to a specific route

```
spec:
  routes:
  - condition:
      method: HEAD
      pathRegex: /authors/[^/]*\.json
    name: HEAD /authors/{id}.json
    isRetryable: true ### ADD THIS LINE ###
```

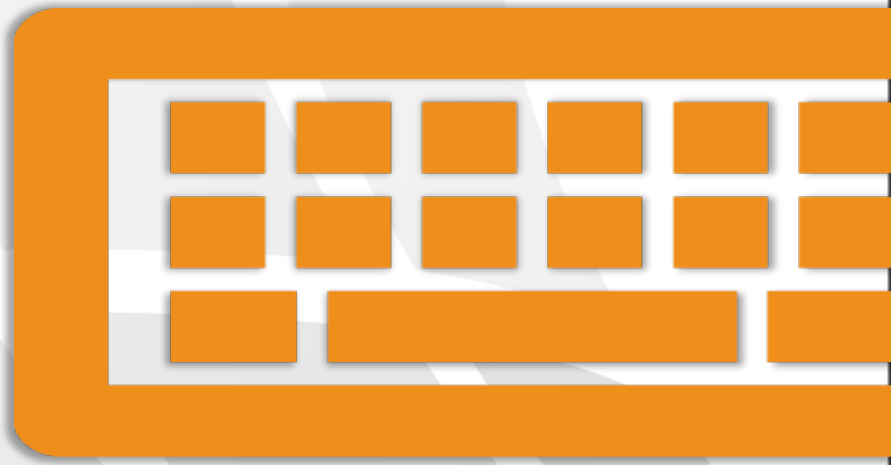
Services Profiles - Timeouts

- To add a retry just add "timeout" to a specific route

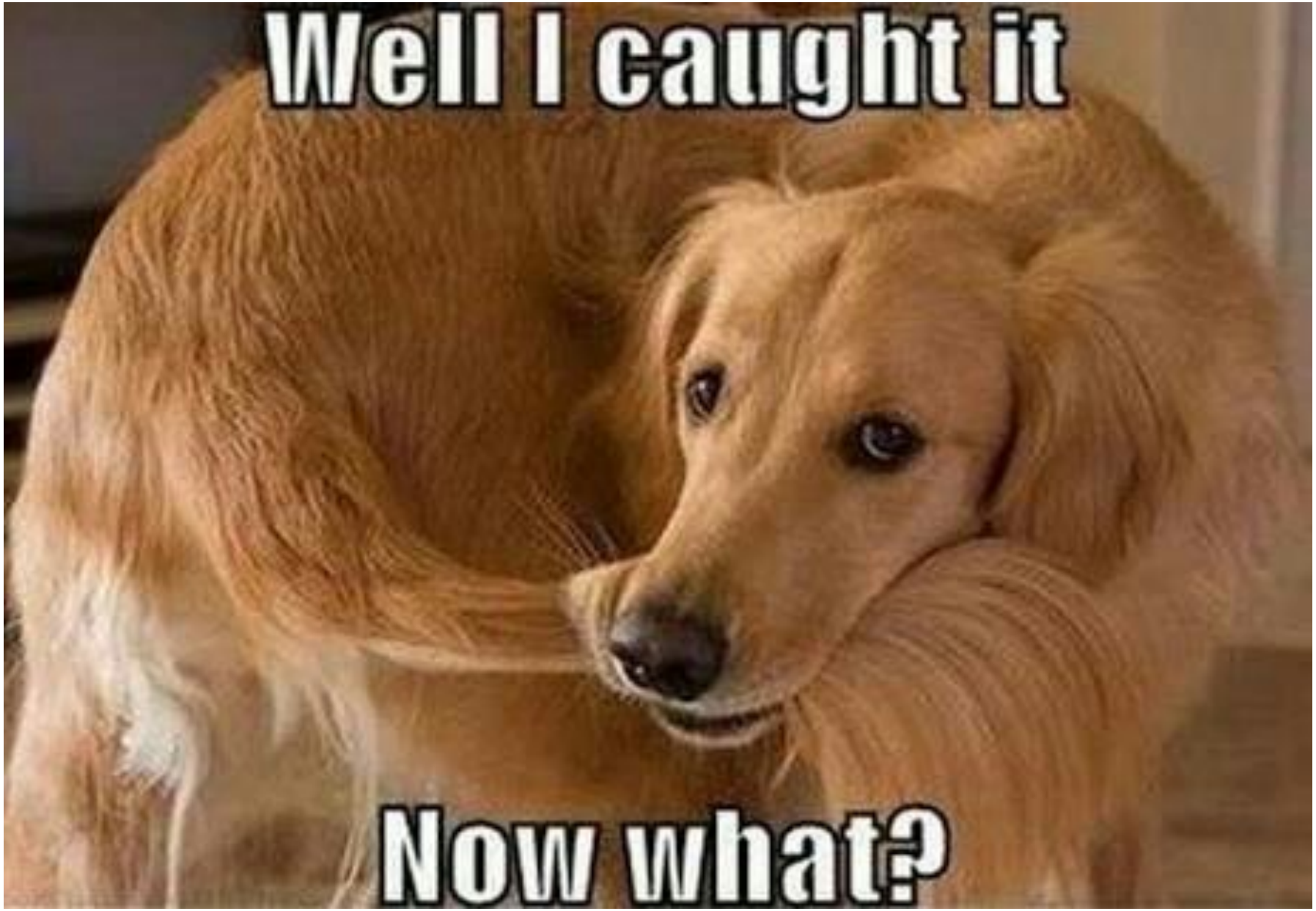
```
spec:
  routes:
  - condition:
      method: PUT
      pathRegex: /books/[^/]*\.json
    name: PUT /books/{id}.json
    timeout: 25ms ### ADD THIS LINE ###
```

Lab 06: Working with Service Profiles

Lab



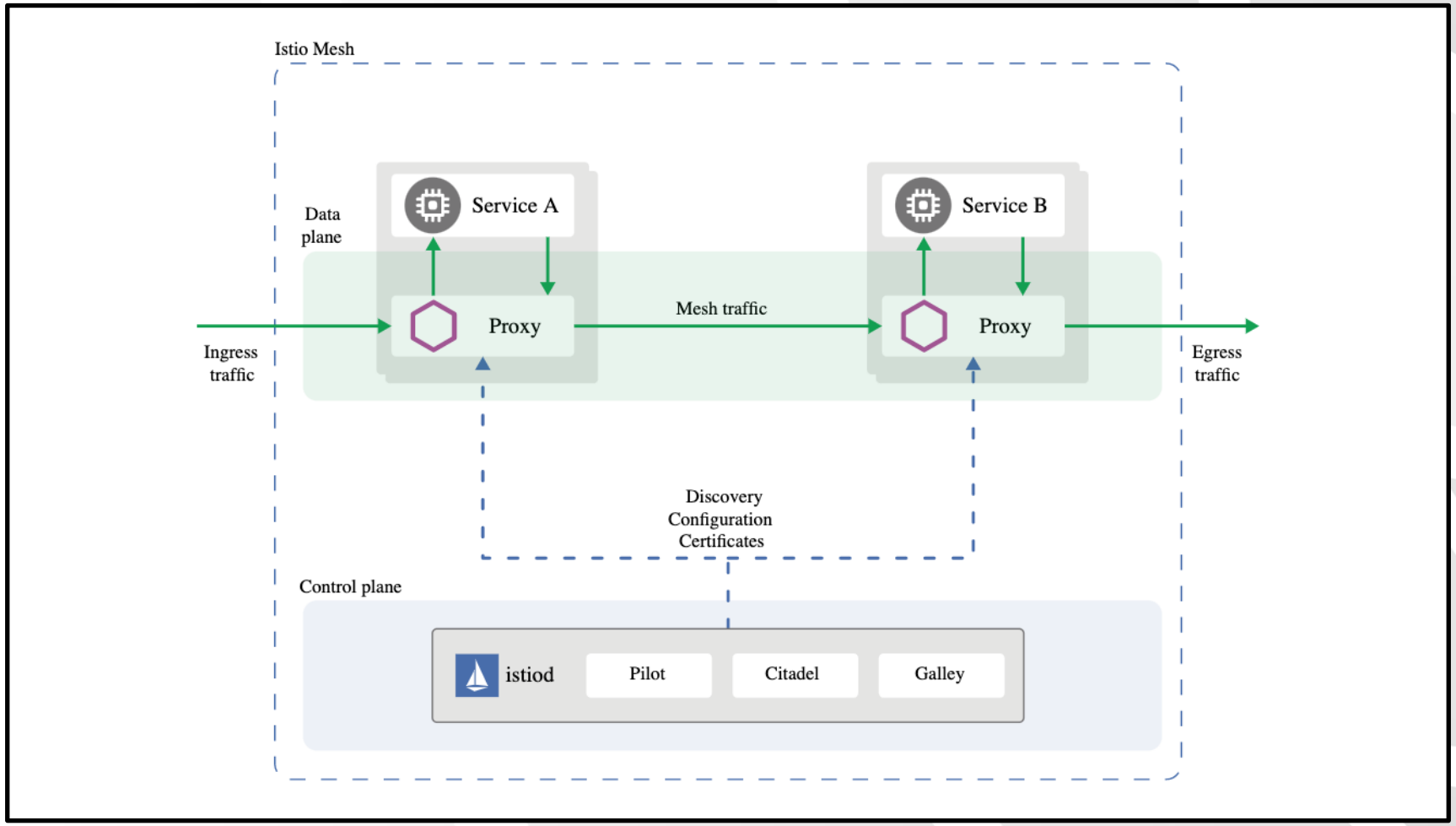
https://github.com/leonjalfon1/sdp-servicemesh/labs/06-linkerd-service-profiles.md


Well I caught it
Now what?

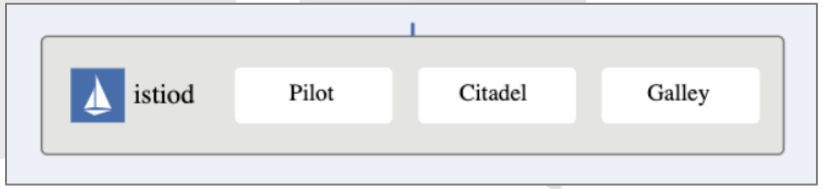
About Istio

- Initiative from Google, IBM and Lyft
- Istio is designed to be platform-independent, initially focused on K8S.
- Provides a uniform way to connect, manage, and secure microservices
- It supports managing traffic flows between services, enforcing access policies, and aggregating telemetry data, all without requiring changes to the microservice code
- Istio is totally open source and free

Istio Architecture

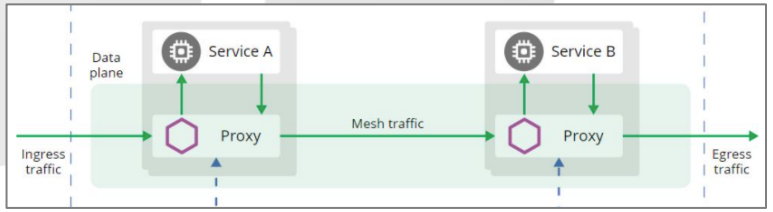


Architecture - Control Plane



- **Istiod:** Unifies functionality that Pilot, Galley, Citadel and the sidecar injector previously performed, into a single binary (introduced in Istio v1.5)
- **Pilot:** Provides service discovery, intelligent routing and fault tolerance
- **Citadel:** Provides reliable service-to-service and end-user authentication.
- **Galley:** Is responsible for validate the istio configuration.

Architecture - Data 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. They also collect and report telemetry on all mesh traffic.
- Envoy is a CNCF Graduate based L4/L7 proxy developed on C++ by Lyft. Use API Driven updates (without hot-reload) and is injected as sidecar.

Istioctl

Istio configuration command line utility for service operators to debug and diagnose their Istio mesh.

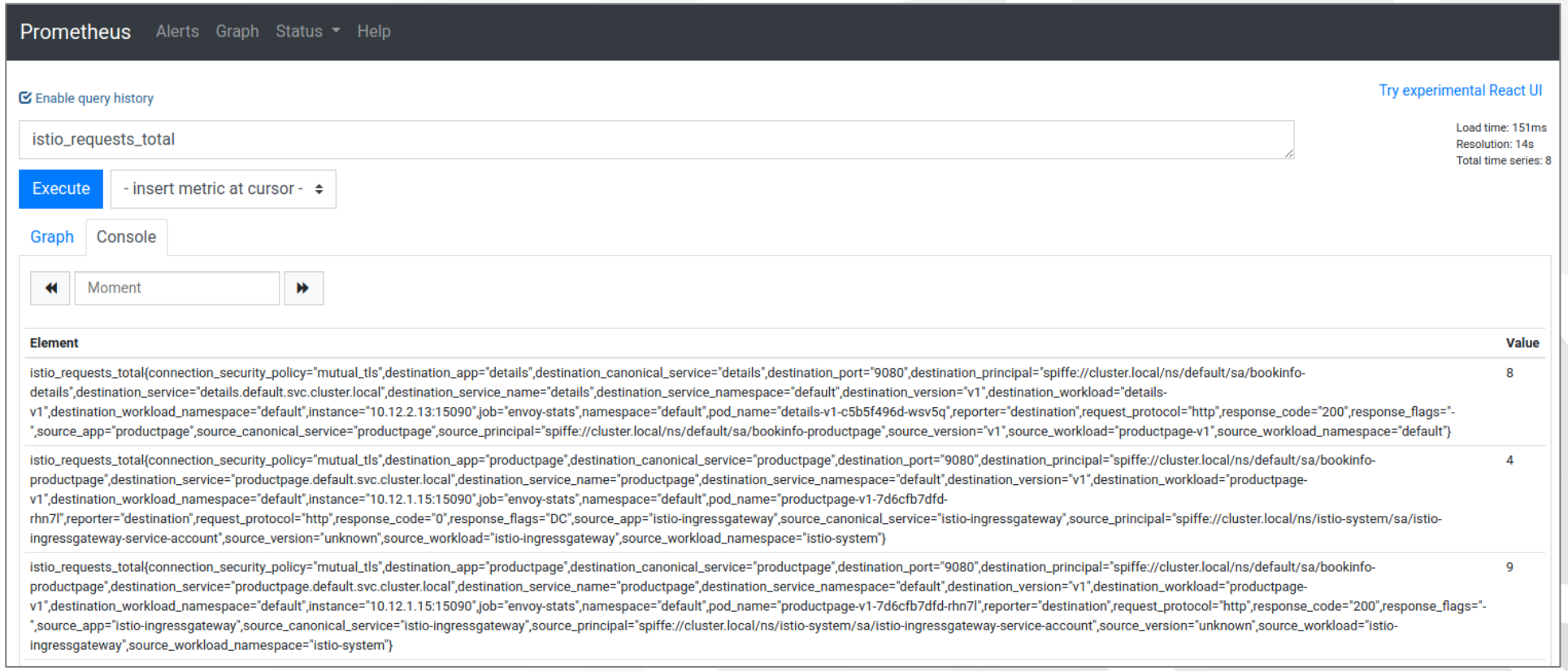
```
Istio configuration command line utility for service operators to
debug and diagnose their Istio mesh.

Usage:
  istioctl [command]

Available Commands:
  analyze         Analyze Istio configuration and print validation messages
  authz           (authz is experimental. Use `istioctl experimental authz`)
  bug-report      Cluster information and log capture support tool.
  convert-ingress Convert Ingress configuration into Istio VirtualService configuration [Deprecated, it will be removed in Istio 1.9]
  dashboard       Access to Istio web UIs
  deregister      De-registers a service instance [Deprecated, it will be removed in Istio 1.9]
  experimental    Experimental commands that may be modified or deprecated
  help            Help about any command
  install         Applies an Istio manifest, installing or reconfiguring Istio on a cluster.
  kube-inject     Inject Envoy sidecar into Kubernetes pod resources
  manifest        Commands related to Istio manifests
  operator        Commands related to Istio operator controller.
  profile         Commands related to Istio configuration profiles
  proxy-config    Retrieve information about proxy configuration from Envoy [kube only]
  proxy-status    Retrieves the synchronization status of each Envoy in the mesh [kube only]
  register        Registers a service instance (e.g. VM) joining the mesh [Deprecated, it will be removed in Istio 1.9]
  upgrade         Upgrade Istio control plane in-place
  validate        Validate Istio policy and rules files
  verify-install  Verifies Istio Installation Status
  version         Prints out build version information
```

Istio Installation

- **istioctl install:** The simplest and most qualified installation and management path with high security.
- **Istio Operator:** Simple installation path without istioctl binaries.
- **Istioctl Manifest Generate:** This method is suitable where strict auditing or augmentation of output manifests is needed.
- **Helm installation:** The Helm charts used in this guide are the same underlying charts used when installing Istio via Istioctl or the Operator.

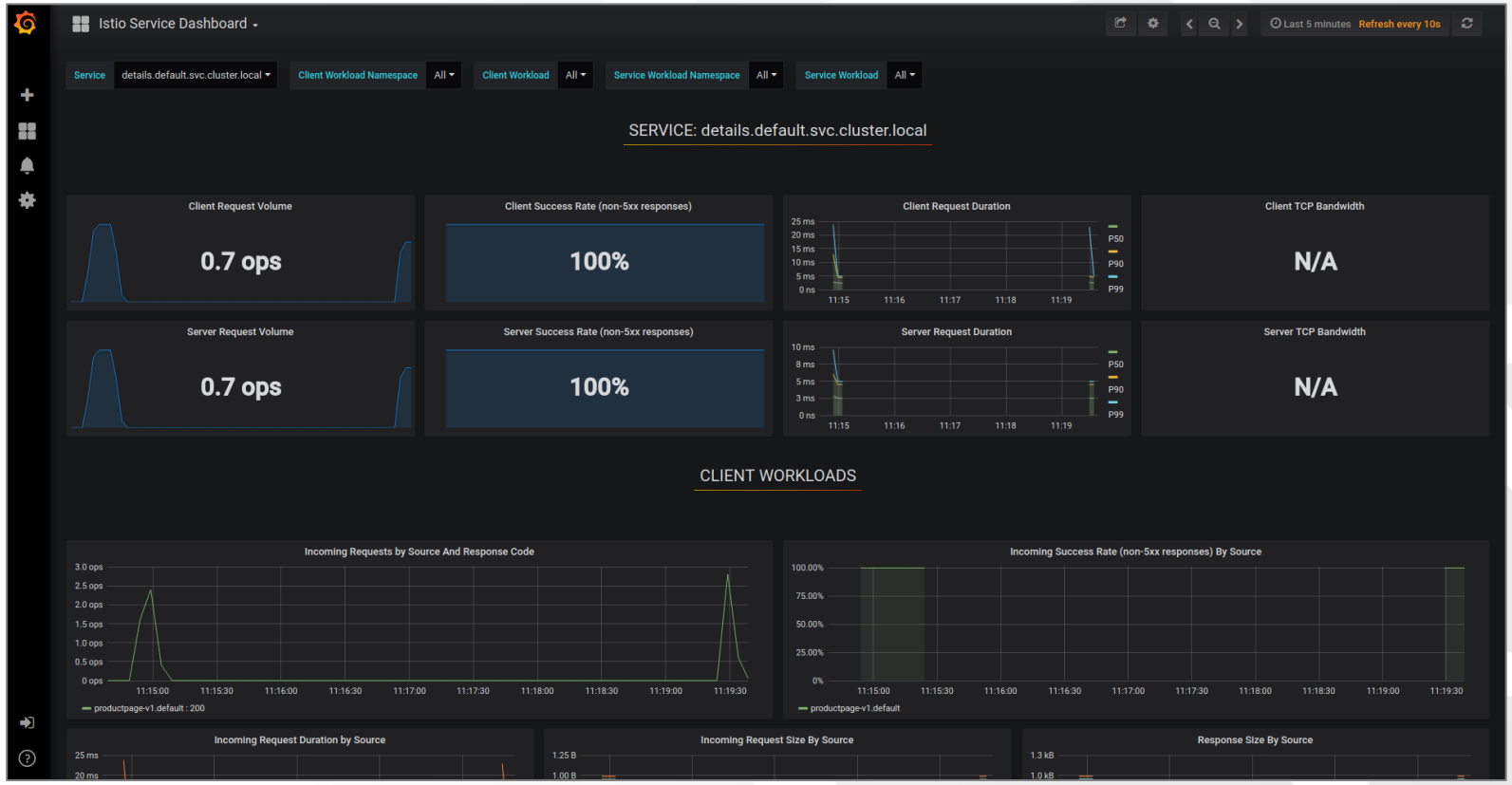
Istio Addons

Istio use Prometheus to store metrics



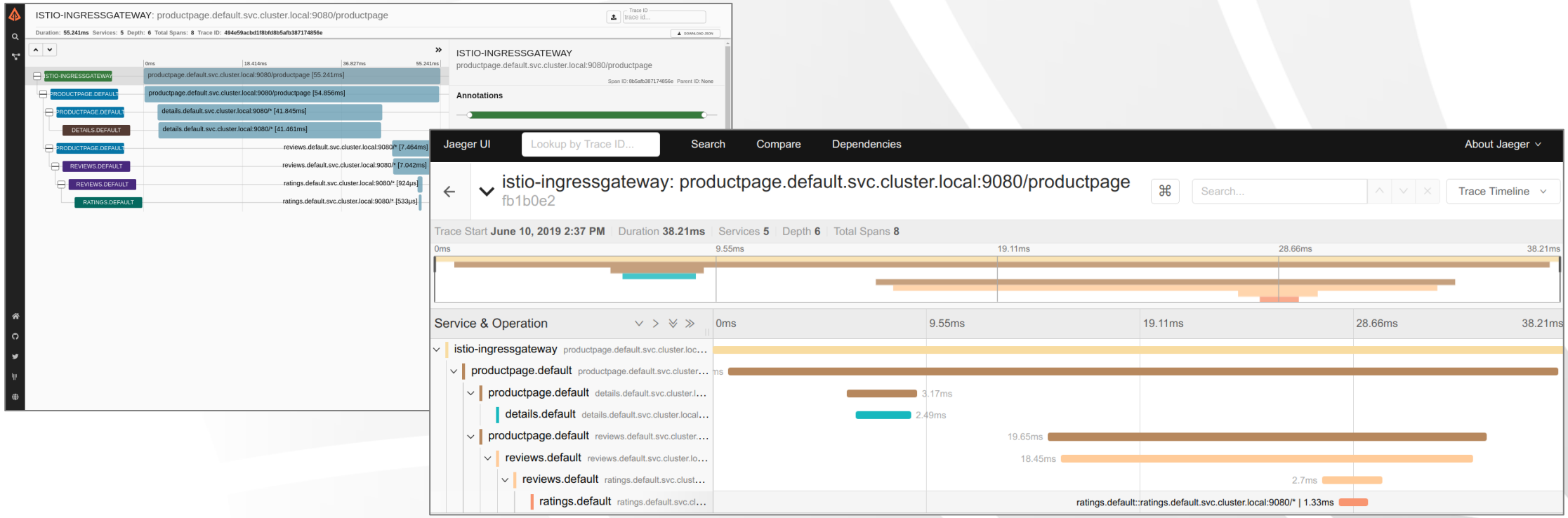
Istio Addons

Istio use Grafana for metrics visualization



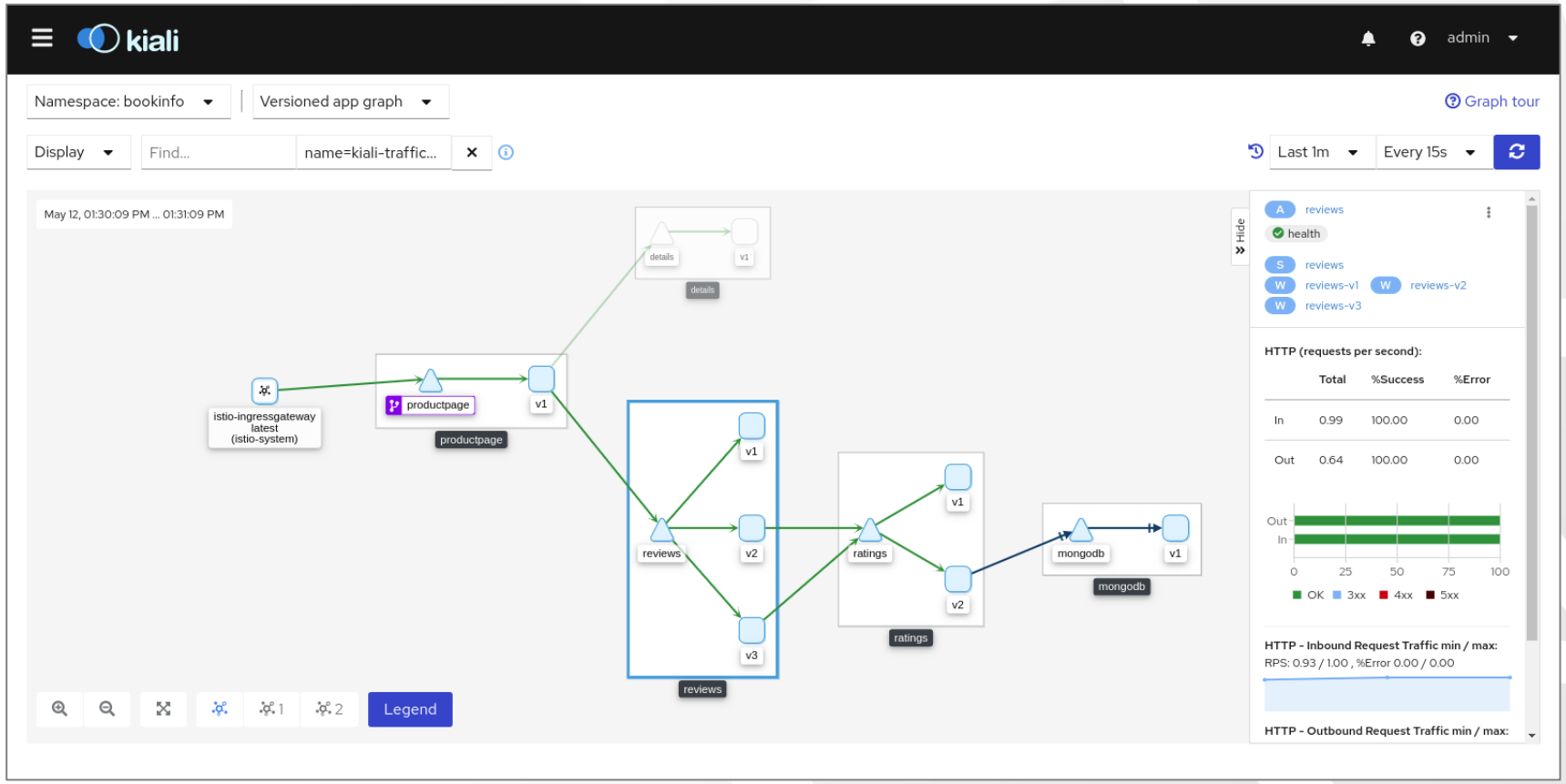
Istio Addons

Istio can use Jaeger or Zipkin for tracing visualization



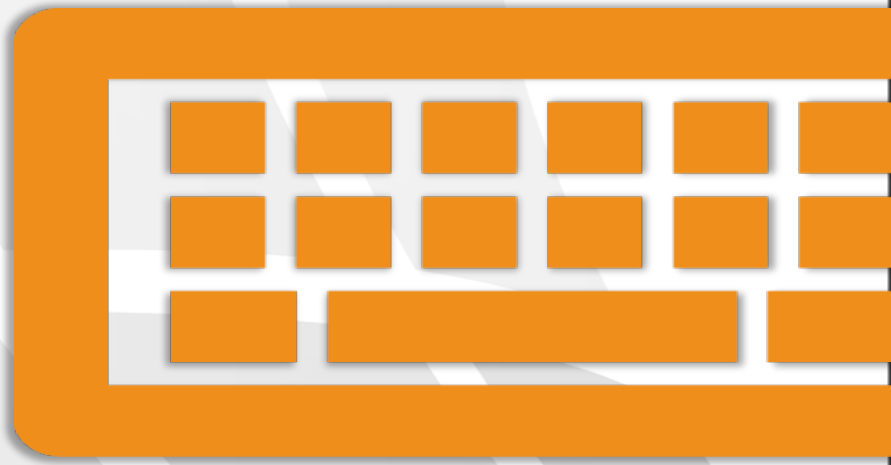
Istio Addons

Istio use Kiali for the mesh visualization



Lab 07: Installing Istio

Lab

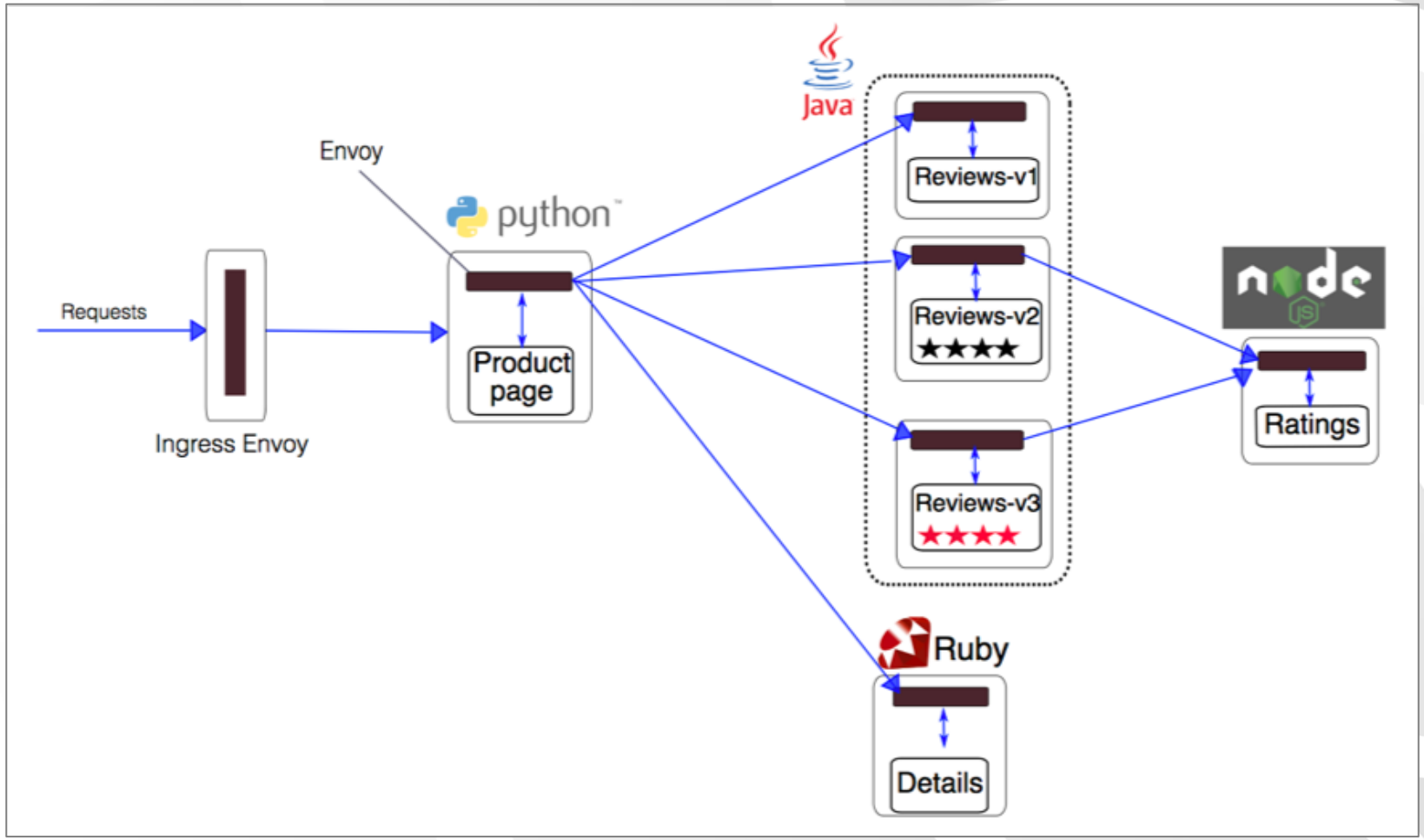


https://github.com/leonjalfon1/sdp-servicemesh/labs/07-installing-istio.md

BookInfo Demo App

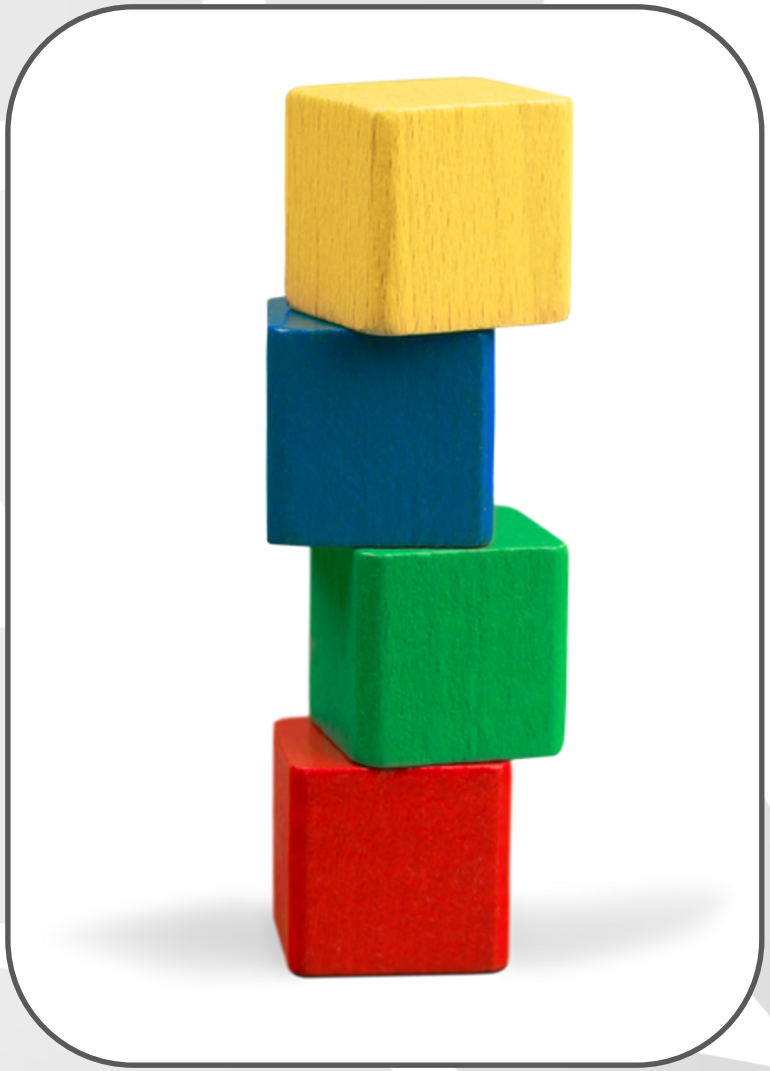
- Let's use the BookInfo application to learn the following topics:
  - Intelligent Routing
  - Fault inject (delay/failure)
  - Traffic Management
  - Authorization

BookInfo – Architecture



Istio Building Blocks

- Gateway
- VirtualService
- DestinationRule
- ServiceEntry
- Sidecar
- PeerAuthentication
- AuthorizationPolicy
- …



Gateway

- You use a gateway to manage inbound and outbound traffic for your mesh, letting you specify which traffic you want to enter or leave the mesh
- Gateway configurations are applied to standalone Envoy proxies that are running at the edge of the mesh, rather than sidecar Envoy proxies running alongside your service workloads
- Unlike Kubernetes Ingress Controllers, Istio gateways let you use the full power and flexibility of Istio's traffic routing

Gateway

```
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:
    - "*"
```

This gateway configuration lets HTTP traffic from any hostname into the mesh on port 80, but doesn't specify any routing for the traffic

VirtualService

- A virtual service lets you configure how requests are routed to a service within an Istio service mesh
- Each virtual service consists of a set of routing rules that are evaluated in order
- Istio match each given request to the virtual service to a specific real destination within the mesh
- Without virtual services, Envoy distributes traffic using round-robin load balancing between all service instances

VirtualService

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

This VirtualService route all the traffic coming from the “bookinfo-gateway” to the destination “reviews” with subset “v1”

DestinationRule

- You can think of virtual services as how you route your traffic to a given destination, and then you use destination rules to configure what happens to traffic for that destination
- Destination rules are applied after virtual service routing rules are evaluated, so they apply to the traffic's "real" destination
- Each subset is defined based on one or more labels
- By default, Istio uses a round-robin load balancing policy but you can specify another algorithm in destination rules

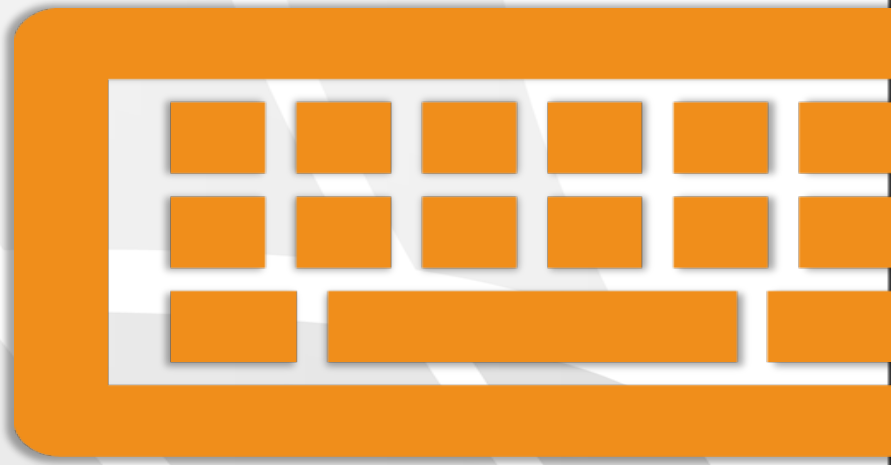
DestinationRule

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: reviews
spec:
  host: reviews
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
  - name: v3
    labels:
      version: v3
```

This destination rule define three subsets for the reviews service. For example, the destination “v1” will route the traffic to a Kubernetes service using the label ”version=v1”

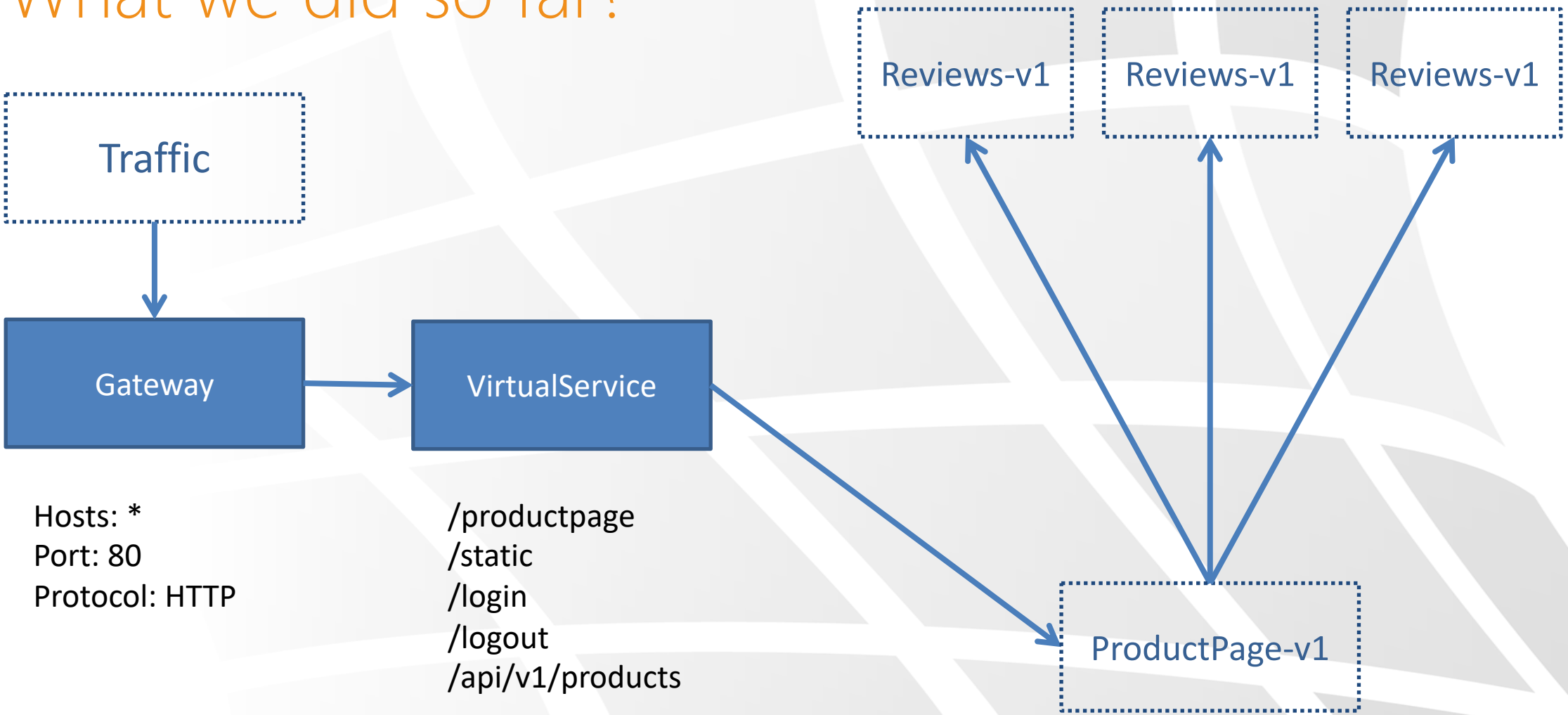
Lab 08: Exploring Istio

Lab

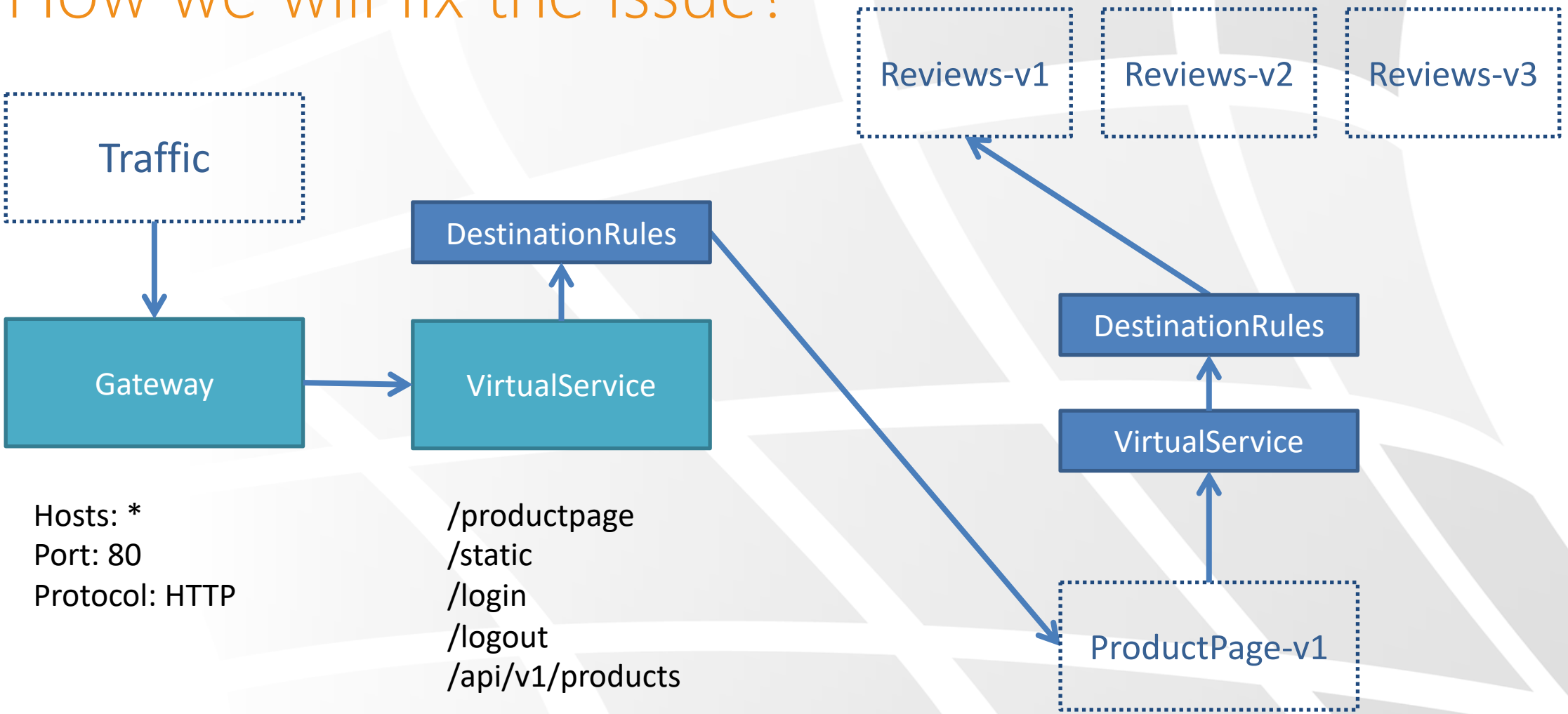


https://github.com/leonjalfon1/sdp-servicemesh/labs/08-exploring-istio.md

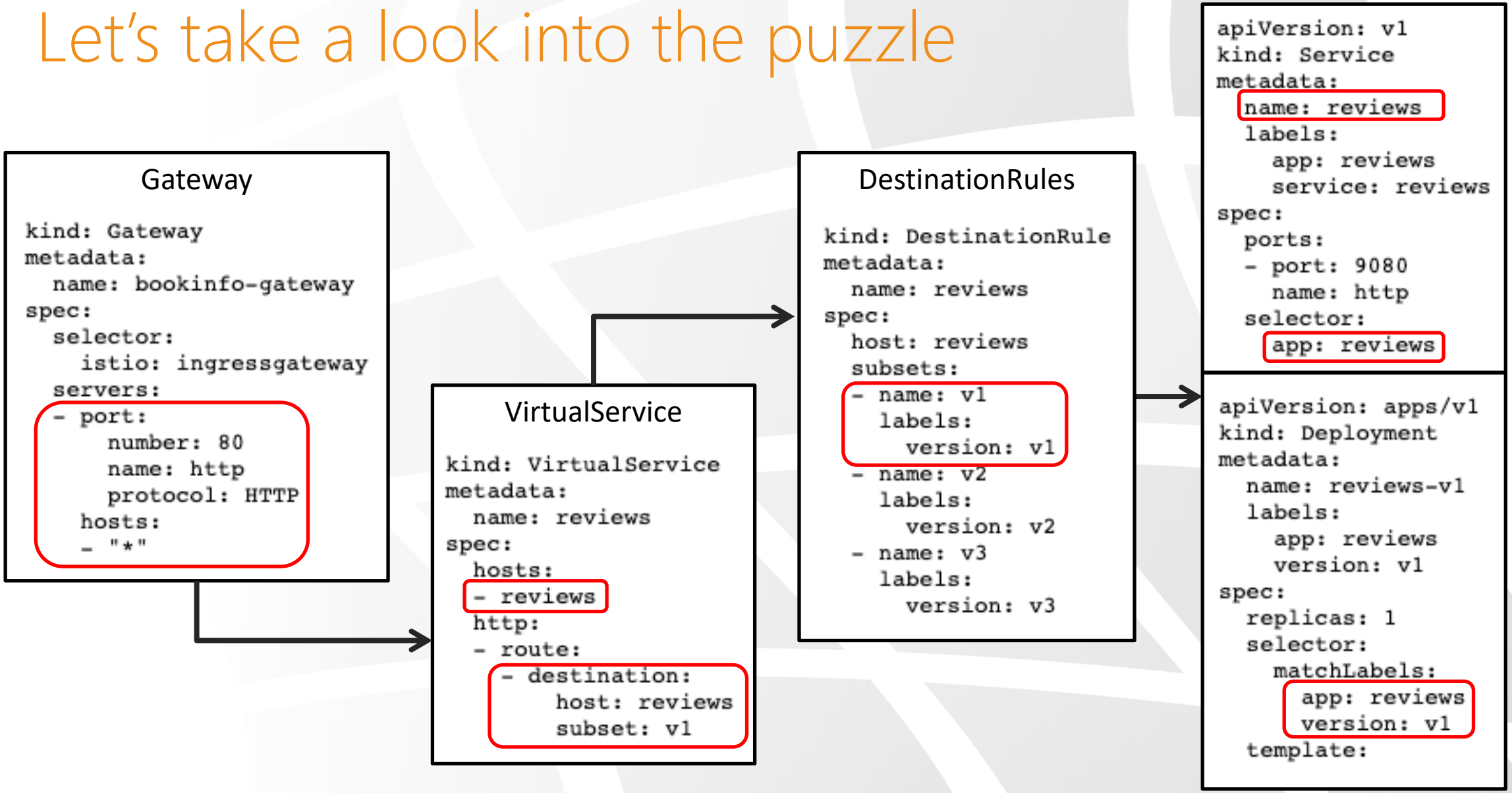
What we did so far?



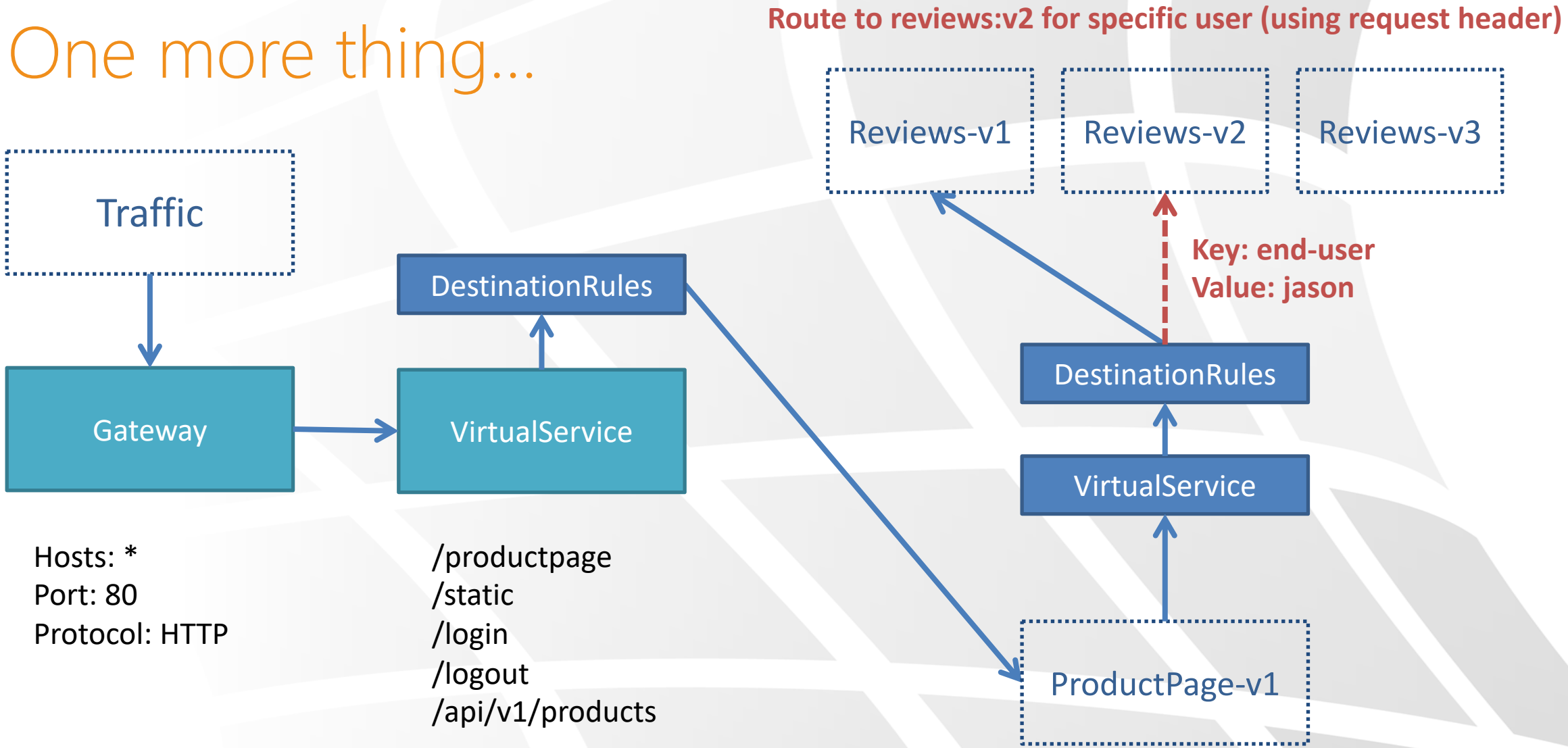
How we will fix the issue?



Let's take a look into the puzzle



One more thing…

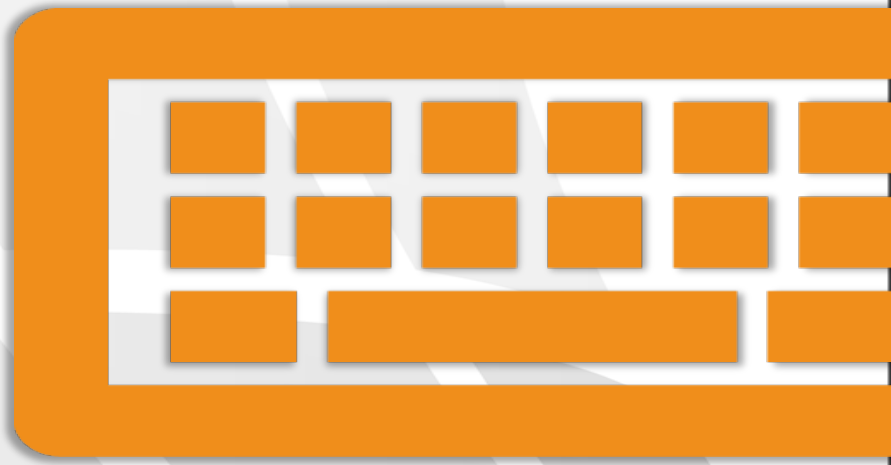


VirtualService – Match Rules

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
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
```

Lab 09: Intelligent Routing with Istio

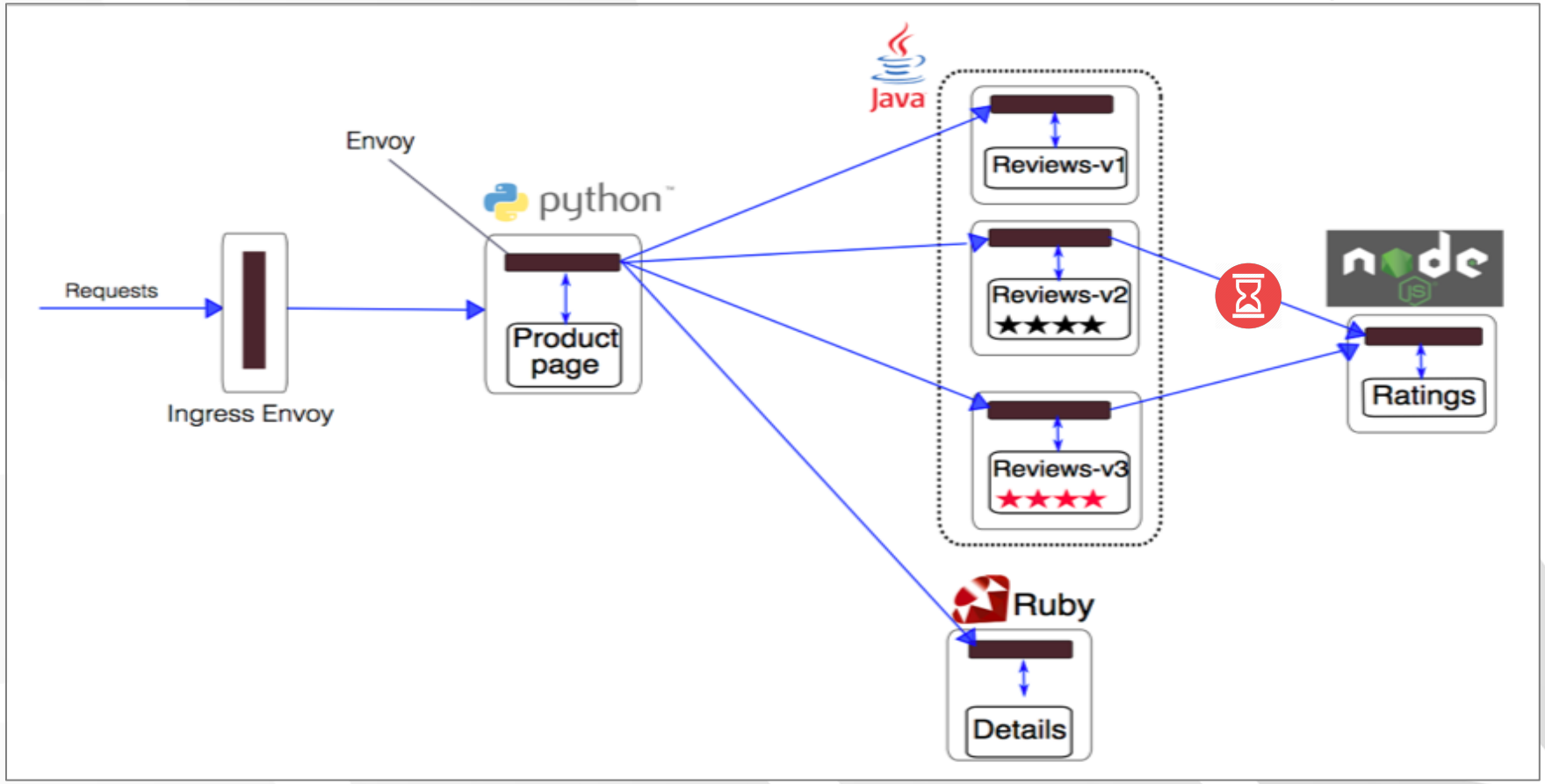
Lab



https://github.com/leonjalfon1/sdp-servicemesh/labs/09-intelligent-routing-istio.md

Delay Injection

- Delay

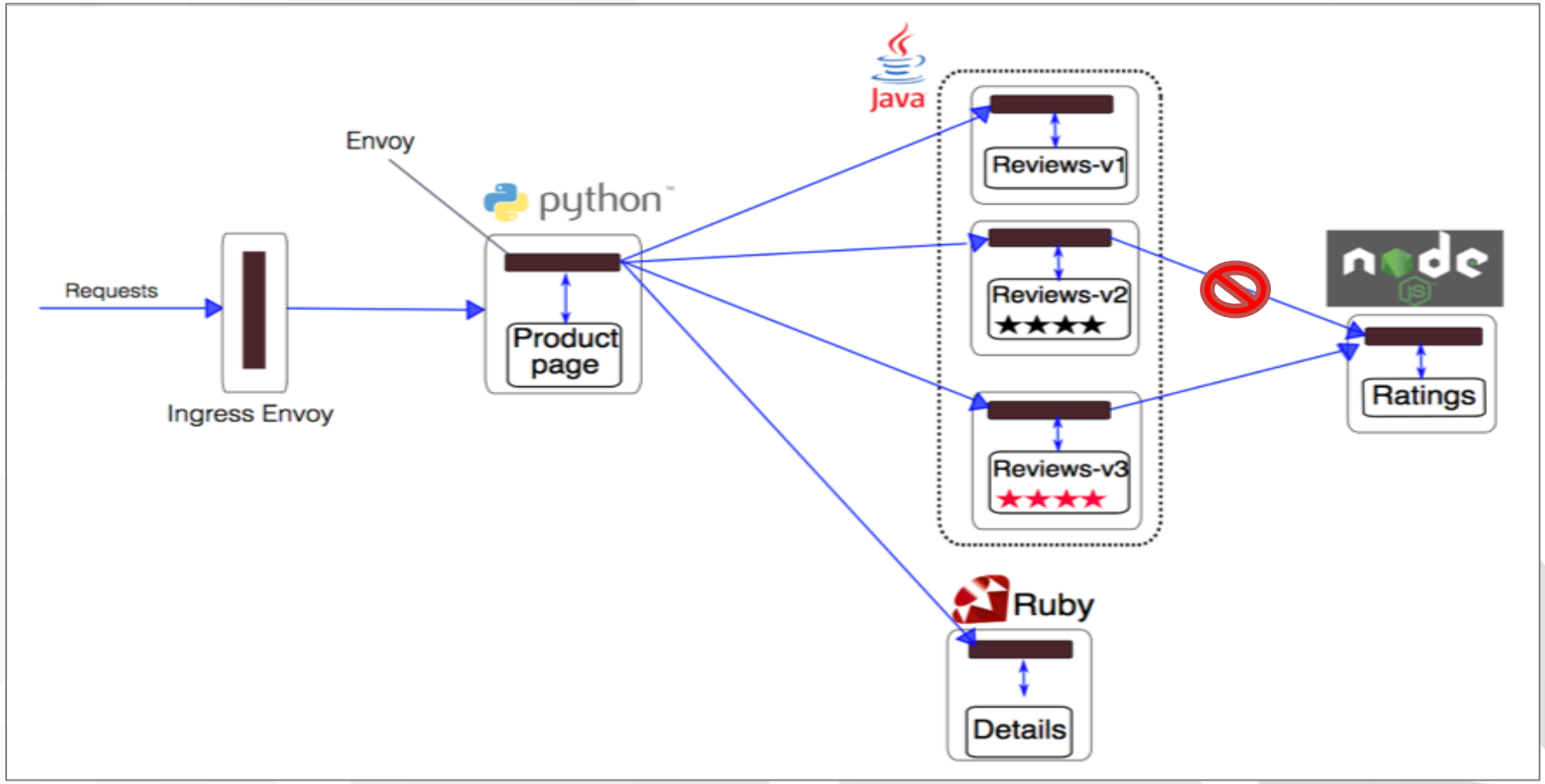


VirtualService – Delay Injection

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    fault:
      delay:
        percentage:
          value: 100.0
        fixedDelay: 7s
    route:
    - destination:
        host: ratings
        subset: v1
  - route:
    - destination:
        host: ratings
        subset: v1
```

Error Injection

- Error

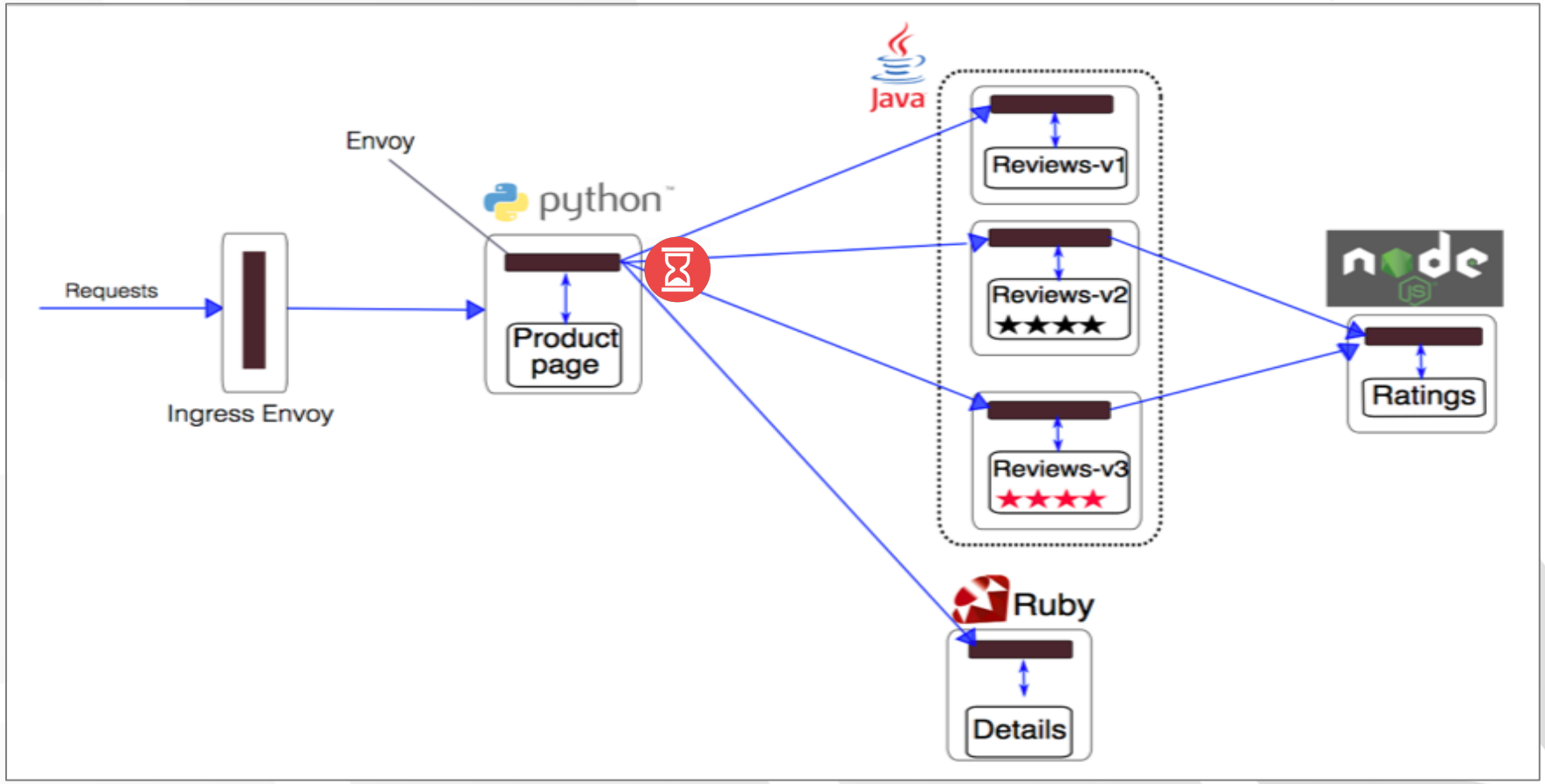


VirtualService – Error Injection

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    fault:
      abort:
        percentage:
          value: 100.0
        httpStatus: 500
    route:
    - destination:
        host: ratings
        subset: v1
  - route:
    - destination:
        host: ratings
        subset: v1
```

Requests Timeout

- Timeout

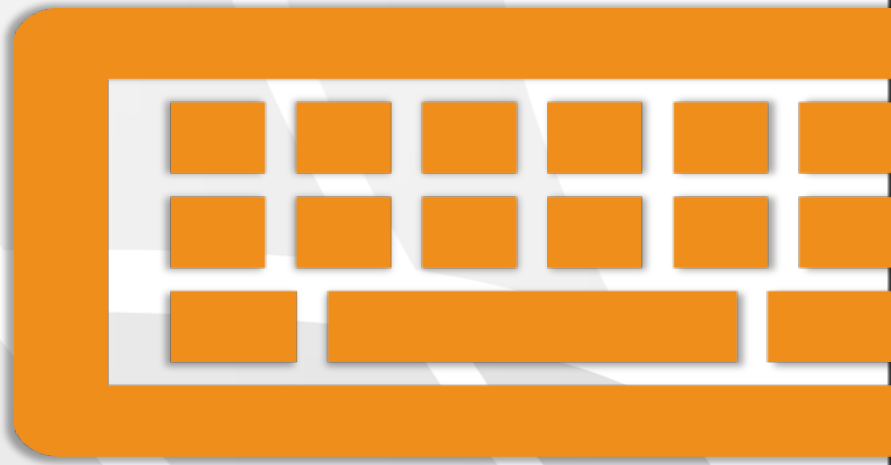


VirtualService – Requests Timeout

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v2
    timeout: 0.5s
```

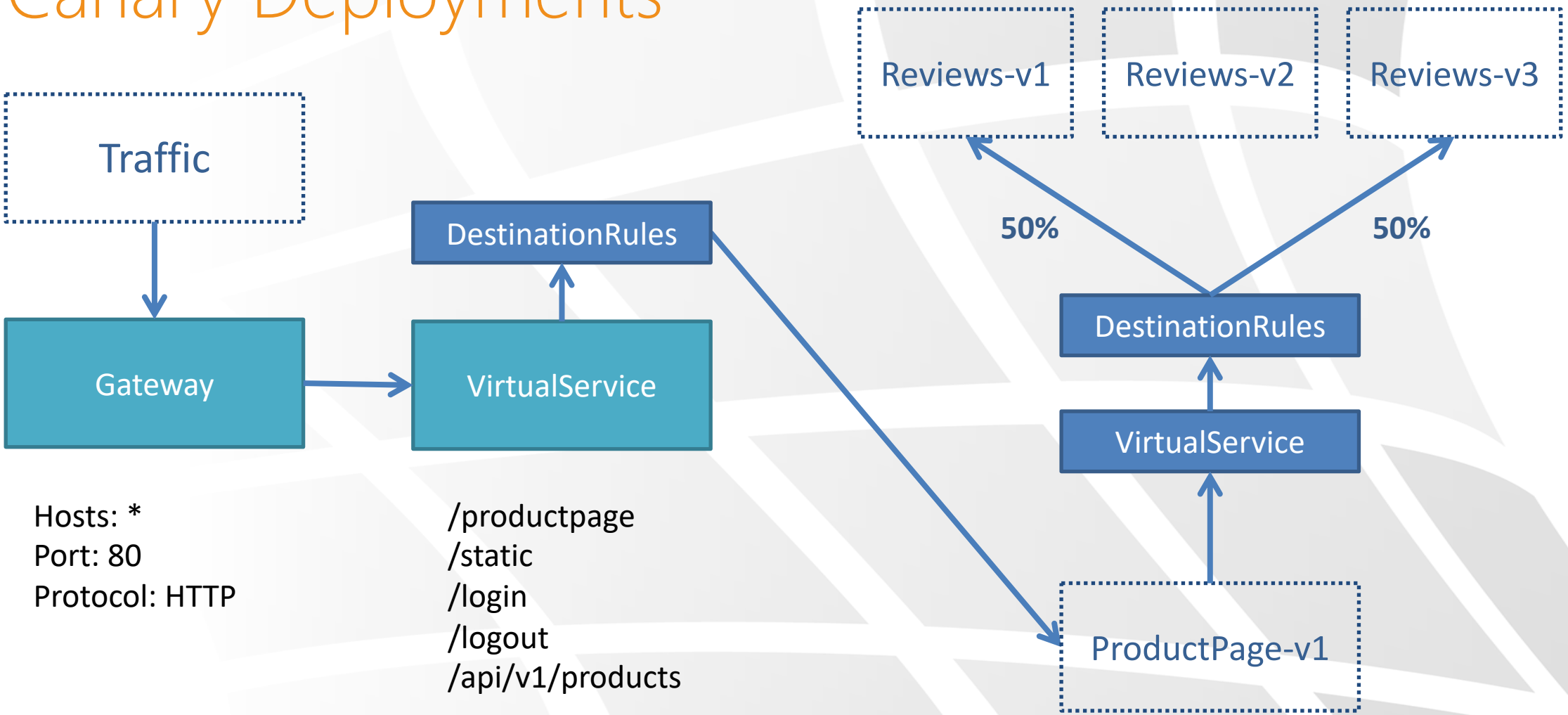
Lab 10: Fault Injection with Istio

Lab



https://github.com/leonjalfon1/sdp-servicemesh/labs/10-fault-injection-istio.md

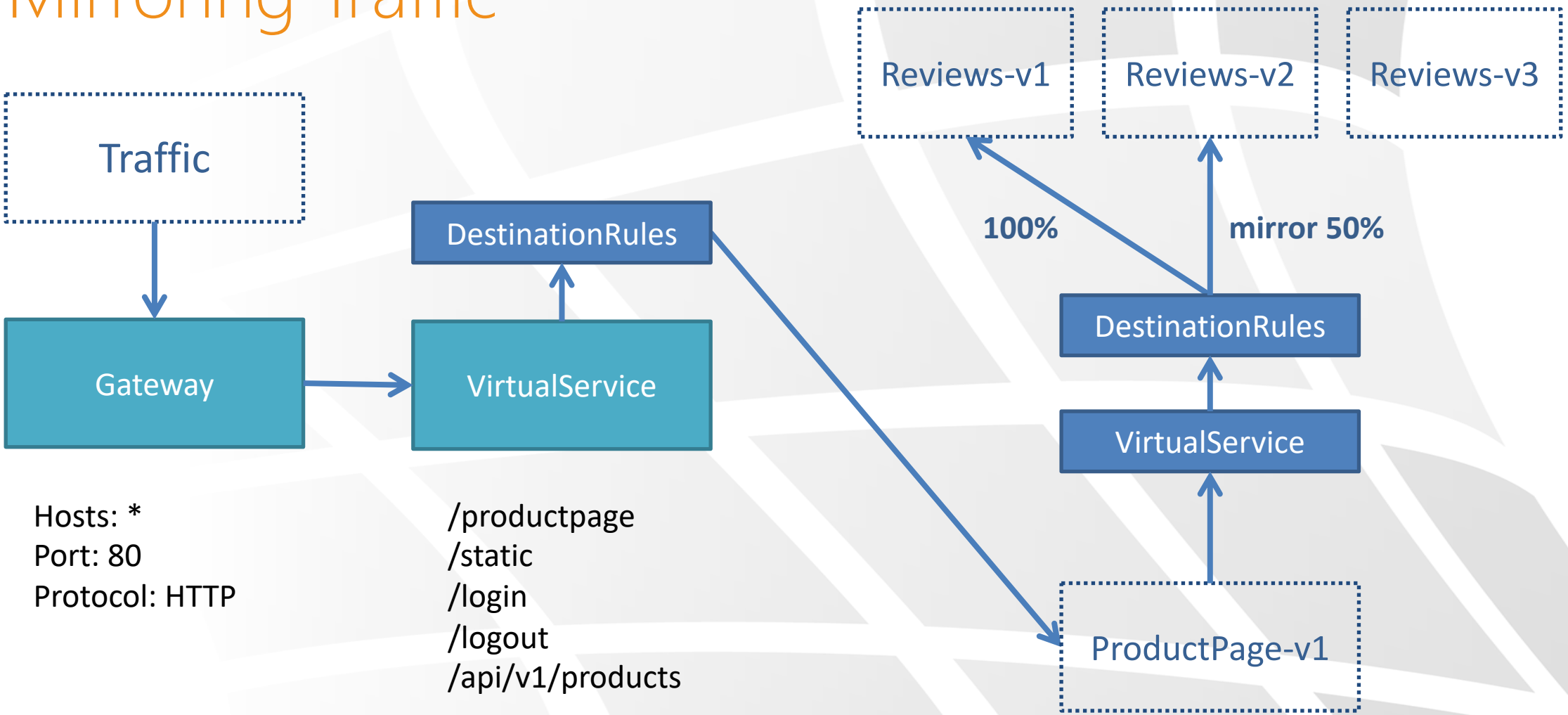
Canary Deployments



VirtualService – Traffic Shifting

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
      weight: 50
    - destination:
        host: reviews
        subset: v3
      weight: 50
```

Mirroring Traffic

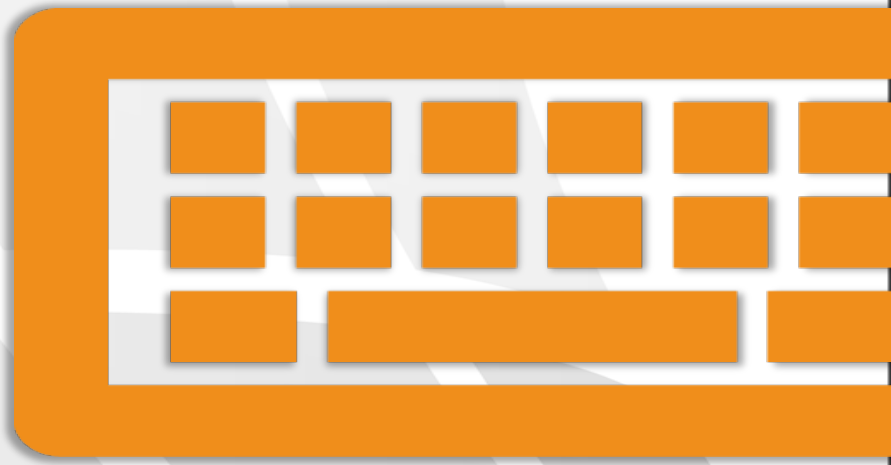


VirtualService – Mirroring Traffic

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
      weight: 100
    mirror:
      host: reviews
      subset: v2
    mirror_percent: 50
```

Lab 11: Traffic Management with Istio

Lab



https://github.com/leonjalfon1/sdp-servicemesh/labs/11-traffic-management-istio.md

Istio Authorization Benefits

- Workload-to-workload and end-user-to-workload authorization.
- A Simple API: it includes a single AuthorizationPolicy CRD, which is easy to use and maintain.
- Flexible semantics: operators can define custom conditions on Istio attributes, and use DENY and ALLOW actions.
- High performance: Istio authorization is enforced natively on Envoy.
- High compatibility: supports gRPC, HTTP, HTTPS and HTTP2 natively, as well as any plain TCP protocols.

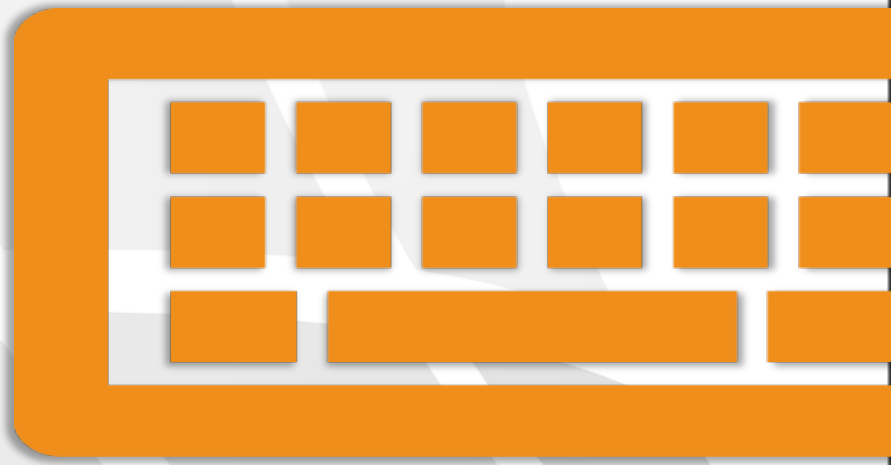
Authorization Policy (CRD)

- The selector field specifies the target of the policy
- The action field specifies whether to allow or deny the request
- The rules specify when to trigger the action

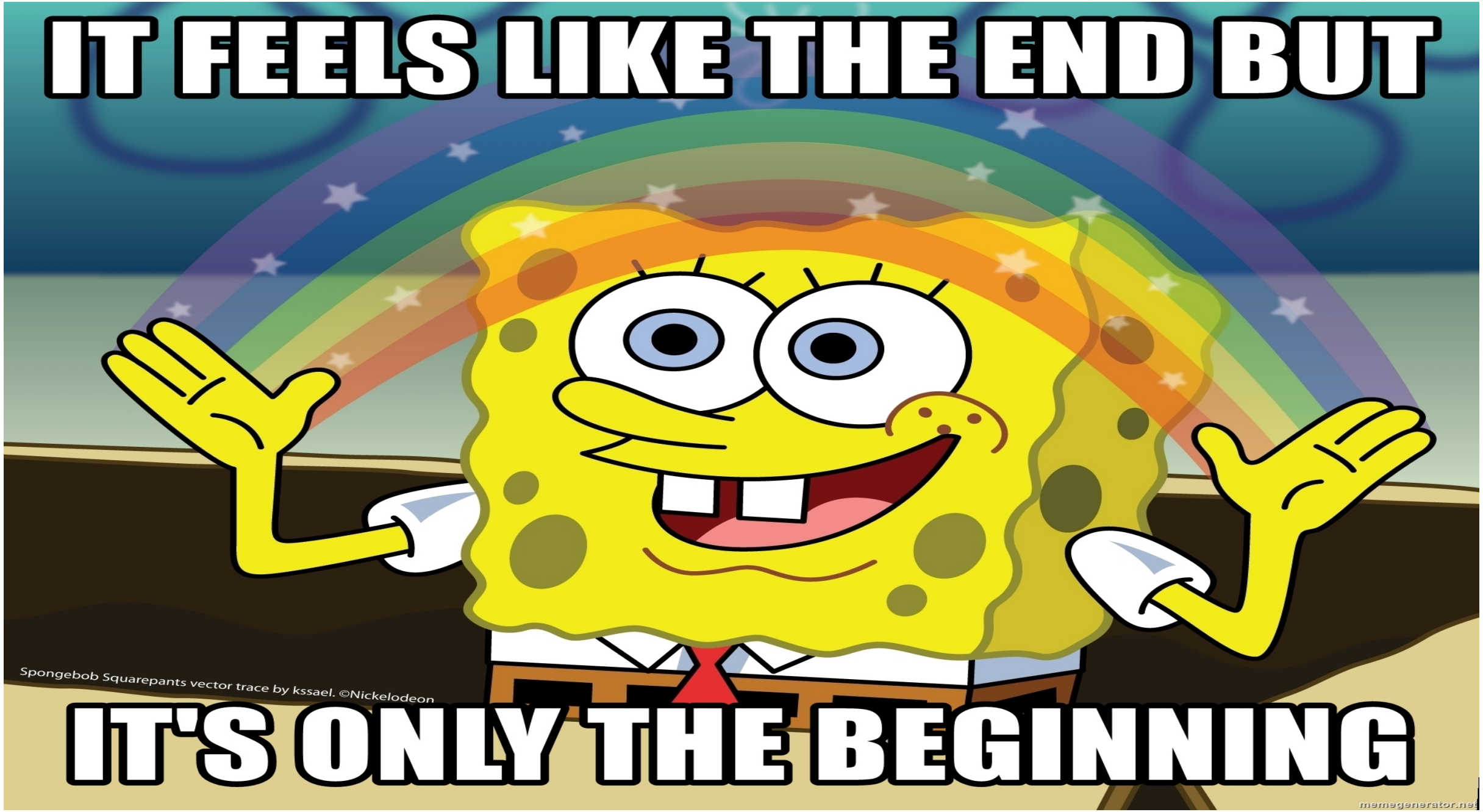
```
apiVersion: "security.istio.io/v1beta1"
kind: "AuthorizationPolicy"
metadata:
  name: "details-viewer"
  namespace: default
spec:
  selector:
    matchLabels:
      app: details
  rules:
  - from:
    - source:
        principals: ["cluster.local/ns/default/sa/bookinfo-productpage"]
    to:
    - operation:
        methods: ["GET"]
```

Lab 12: Setup Istio Authorization

Lab



https://github.com/leonjalfon1/sdp-servicemesh/labs/12-setup-istio-authorization.md


IT FEELS LIKE THE END BUT
Spongebob Squarepants vector trace by kssael. ©Nickelodeon
IT'S ONLY THE BEGINNING
memegenerator.net

The Service Mesh Interface (SMI)

- A standard interface for service meshes on Kubernetes
- A basic feature set for the most common service mesh use cases
- Flexibility to support new service mesh capabilities over time
- Space for the ecosystem to innovate with service mesh technology

What SMI covers?

- **Traffic policy** – apply policies like identity and transport encryption across services
- **Traffic telemetry** – capture key metrics like error rate and latency between services
- **Traffic management** – shift traffic between different services

Service Mesh Interface Ecosystem

SMI API's

- **Traffic Access Control:** Apply policies like identity and transport encryption across services
- **Traffic Split:** Shift traffic between different services.
- **Traffic Specs:** Describe traffic on a per-protocol basis.
- **Traffic Metrics:** Capture key metrics like error rate and latency between services.

TrafficSplit Example

Canary example:

```yaml
kind: TrafficSplit
metadata:
  name: canary
spec:
  # The root service that clients use to connect to the destination application.
  service: website
  # Services inside the namespace with their own selectors, endpoints and configuration.
  backends:
  - service: website-v1
    weight: 90
  - service: website-v2
    weight: 10
```

The above configuration will route 10% of the `website` incoming traffic to the `website-v1` service and 90% to `website-v2` service.

Ok, But Why SMI?

- The goal of the SMI API is to provide a common, portable set of service mesh APIs which a Kubernetes user can use in a provider agnostic manner.
- In this way people can define applications that use service mesh technology without tightly binding to any specific implementation.


WE DID IT!
WOOHOO
makeameme.org

What We Learned Today?

- Service Mesh (with Linkerd and Istio), including:
  - Installation
  - Observability
  - Debugging
  - Fault Injection
  - Intelligent Routing
  - Traffic Management
  - Authorization for HTTP traffic
- Service Mesh Interface

So, How to Choose a Service Mesh?

	Linkerd	Istio
Advantages	Linkerd 2 is designed to be non-invasive and is optimized for performance and usability. Therefore, it requires little time to adopt.	Istio can be adapted and extended like no other Mesh. Its many features are available for Kubernetes and other platforms.
Drawbacks	Linkerd 2 is deeply integrated with Kubernetes and cannot be expanded. Since Linkerd 2 does not rely on a third-party proxy, it cannot be extended easily.	Istio's flexibility can be overwhelming for teams who don't have the capacity for more complex technology. Also, Istio takes control of the ingress controller.
	[SIMPLE]	[POWERFUL]

Full comparission available here: https://servicemesh.es/

PEOPLE RUNNING
ISTIO ON K8S

PEOPLE RUNNING
LINKERD ON K8S

Thank you for your attention!

Service Mesh Fundamentals

Leon Jalfon

leonj@sela.co.il