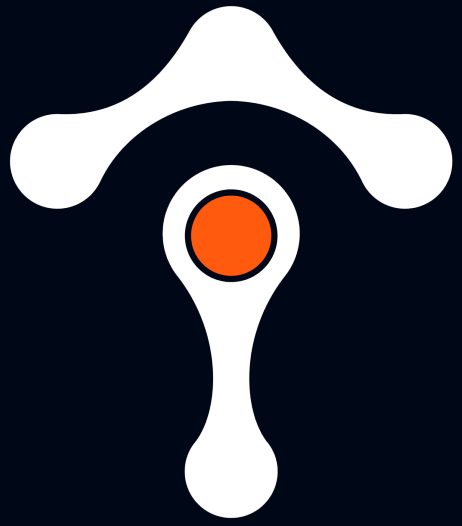


tetrate

THE ENTERPRISE SERVICE MESH COMPANY

IstioCon

WELCOME

Eitan Suez

Peter Jausovec

Meet the workshop instructors

ISTIO 0 TO 60

- Eitan Suez
- Peter Jausovec

About this workshop

ISTIO 0 TO 60

- Get up and running quickly with Istio
- Prerequisites
  - Basic understanding of Kubernetes, Docker, and Linux command line.

Logistics

ISTIO 0 TO 60

- Duration: 2.5 hours
  - 15 minute break half-way through the training
- Communication support through slack channel
  - https://istio.slack.com/
  - channel #istiocon-workshop-tetrate

Learn by doing

APPROACH

- *Lab-driven* training
- Minimize the use of slides
- We request your active participation
  - Please ask questions

Schedule

Lab environments

Configure access to your lab environment

What problems does Istio address?

A high-level introduction and overview of Istio

Install Istio

Get your cluster installed and configured with Istio

Sidecar injection and the app under test

Deploy a simple application to the mesh, and expose it with Ingress

Observability, Security, Traffic shifting

Three labs that cover the essential cross-cutting concerns that Istio addresses

Summary

Workshop labs

https://tetratelabs.github.io/istio-0to60/

Environments

IN A NUTSHELL

- BYOK (Bring Your Own Kubernetes) - if possible
- Fallback:
  - We have a few Kubernetes clusters provisioned in GCP that we can give you access to, "while supplies last."
  - Let us know via slack if you need a K8S cluster and we'll assign you a cluster.
  - The first lab contains instructions for accessing your cluster on GCP.
- We will begin with a lab to get your environment setup

LAB

Setup your environment

https://tetratelabs.github.io/istio-0to60/environment/

Meet Istio

PROBLEM

IT shift to a modern distributed architecture has left enterprises unable to monitor, connect, manage, and secure their services in a consistent way.

Problem

MODERN DISTRIBUTED ARCHITECTURE

- Container based services
- Deployed into dynamic environments
- Composed via the network

Problem

MONITOR

- Understand what’s actually happening in your deployment through basic tools:
  - Metrics
  - Logs
  - Tracing

Problem

CONNECT

- Get network out of the application
  - **Service discovery**
  - **Resiliency**

    Retries, outlier detection, circuit breaking, timeouts, etc.
  - **Load balancing**

    Client side

Problem

MANAGE

- Control which requests are allowed, and how and where they flow
  - **Fine-grained traffic control**

    > L7, not L4! Route by headers, destination or source, etc.
  - **Policy on requests**

    > Authentication, rate limiting, arbitrary policy based on L7 metadata

Problem

SECURE

- Elevate security out of the network
  - **Service-to-service authentication**
  - **Workload identity (L7)**

Service Mesh

WHAT IS SERVICE MESH

- Service mesh moves these facets out of the application for better division of labor and...
  - **Consistency across fleets**
  - **Centralized control**
  - **Ease of change**

    > Update configurations without redeployment

What is Istio?

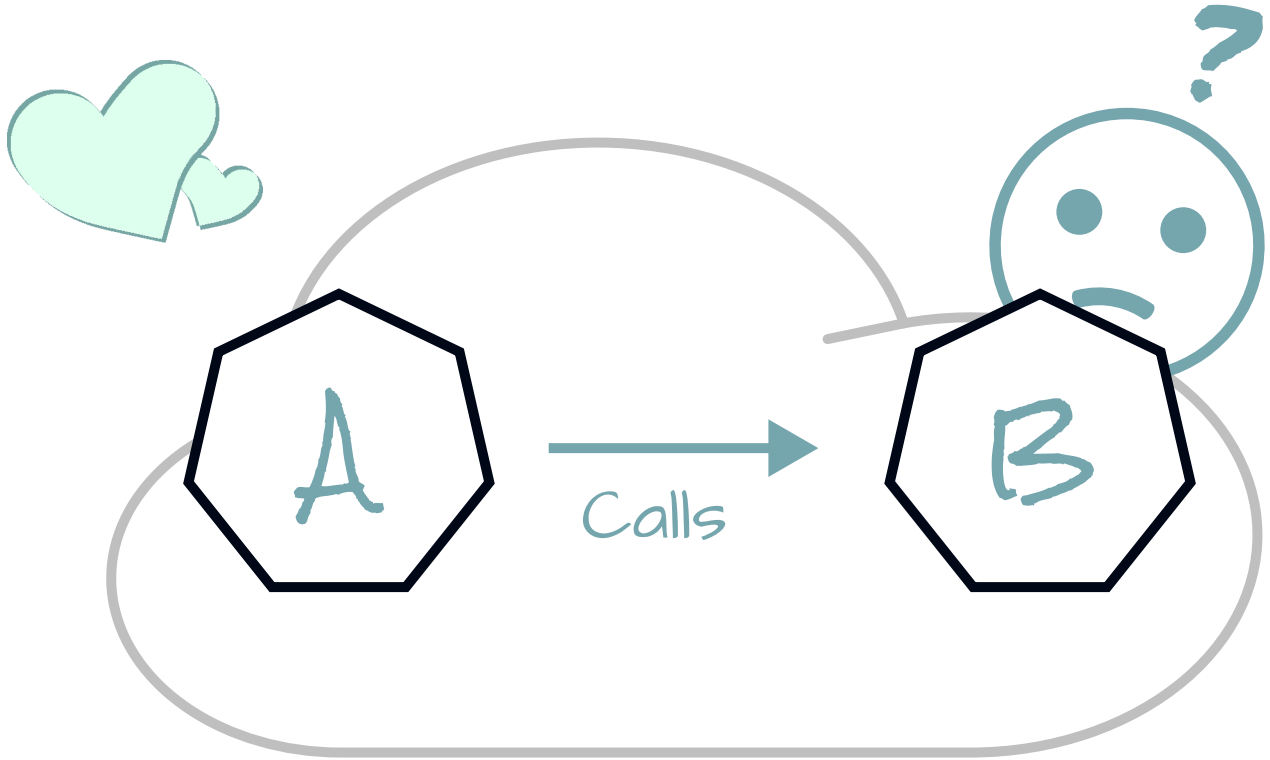
Istio is a platform to **monitor**, **secure**, **connect** and **manage** services consistently

LAB

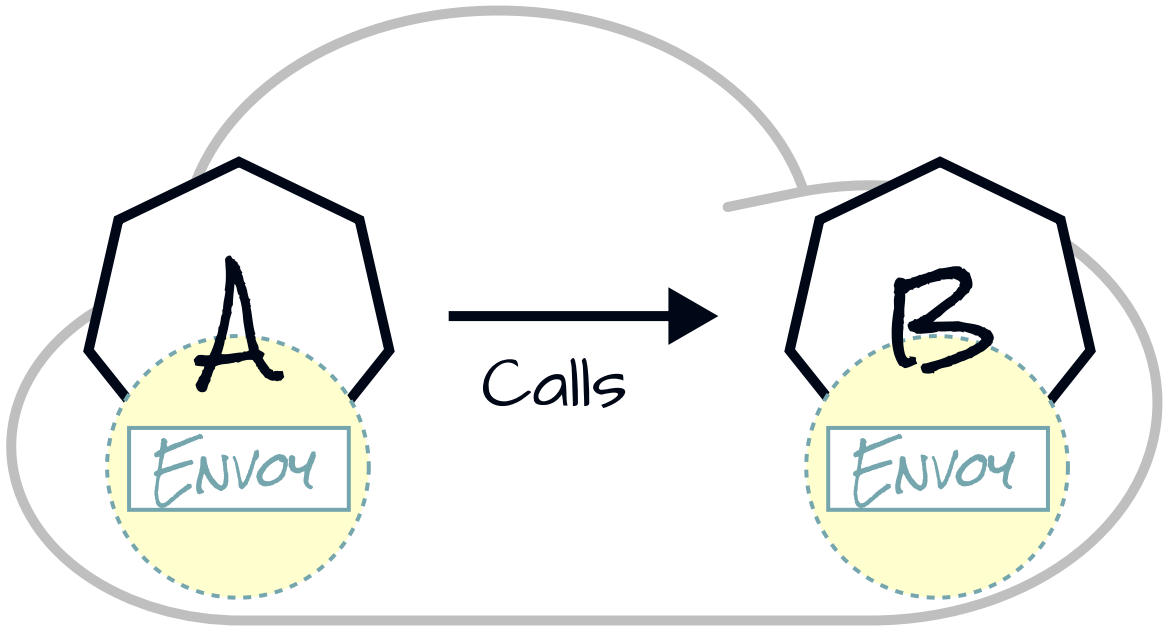
Installing Istio

https://tetratelabs.github.io/istio-0to60/install/

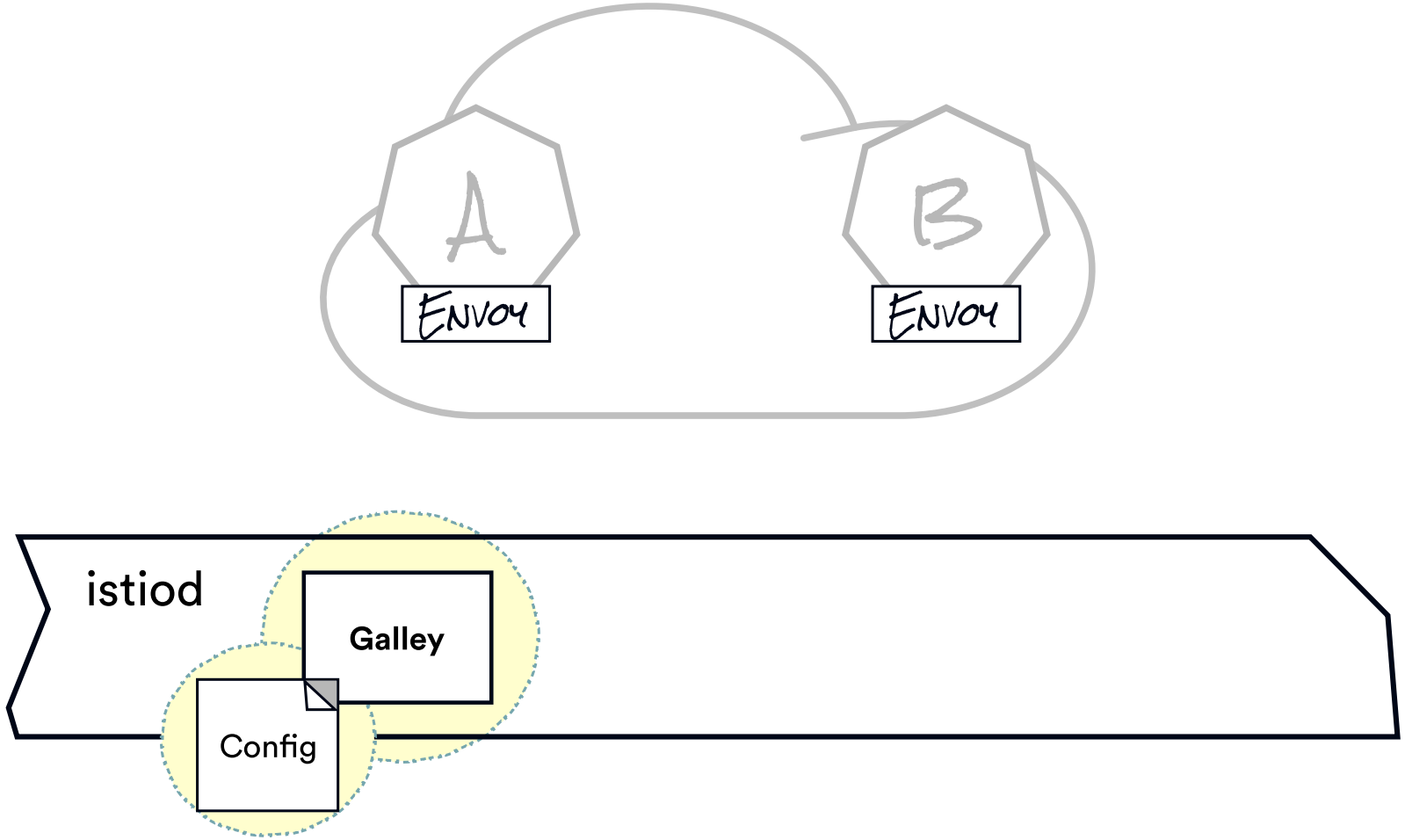
Istio Architecture



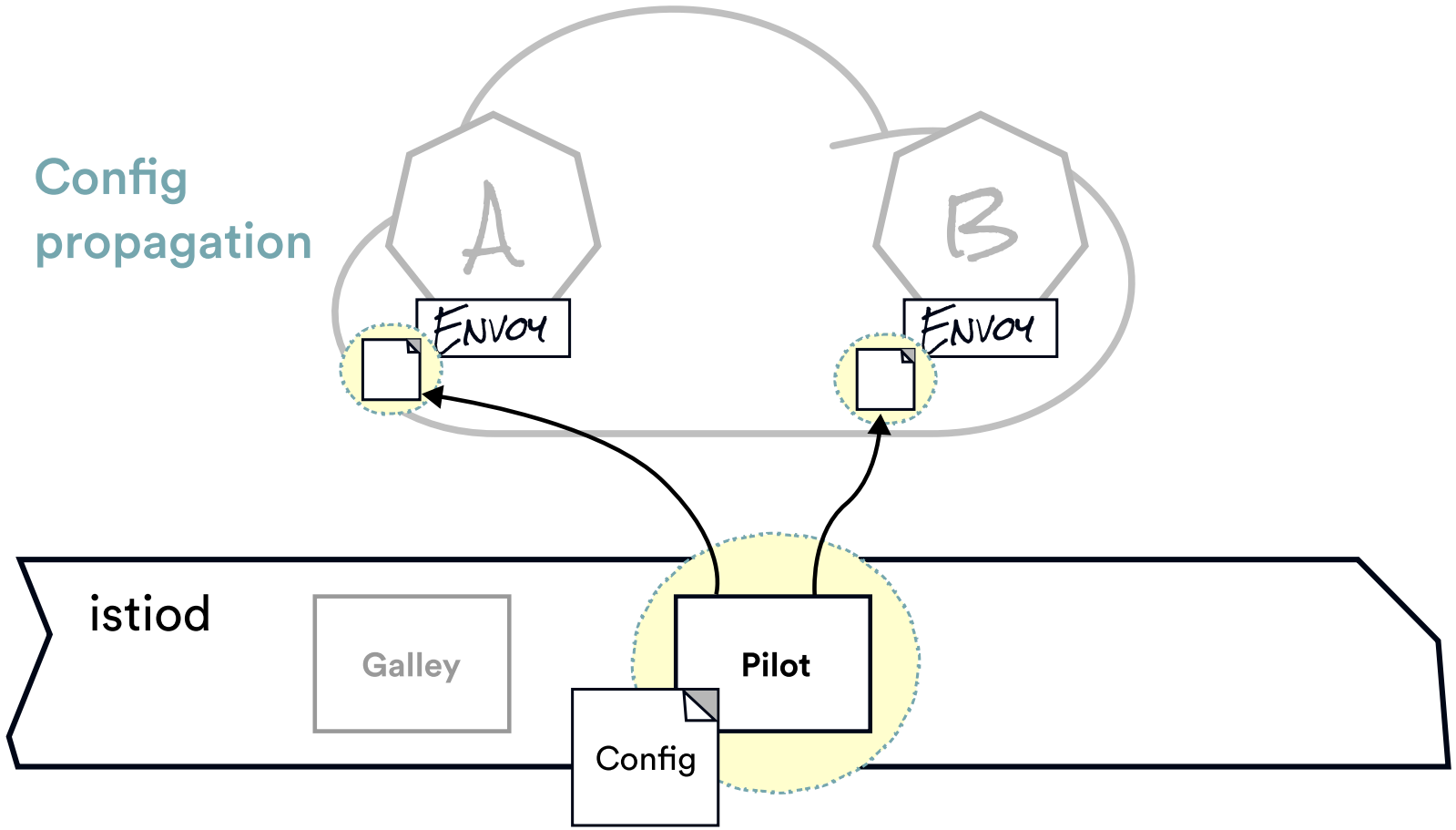
Story as old as time:
Service A meets service B...



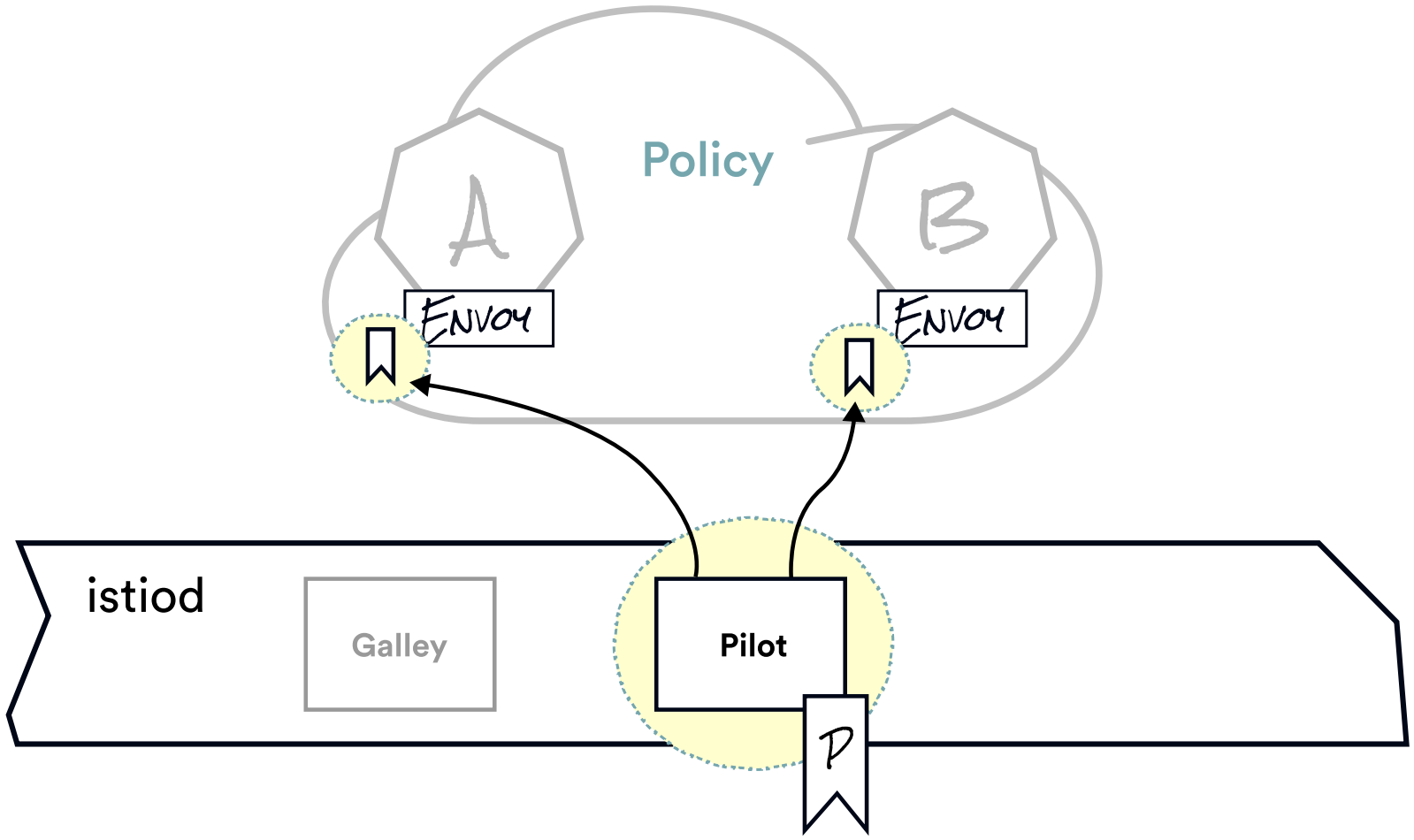
Deploy a proxy (Envoy) beside your application ("sidecar deployment")



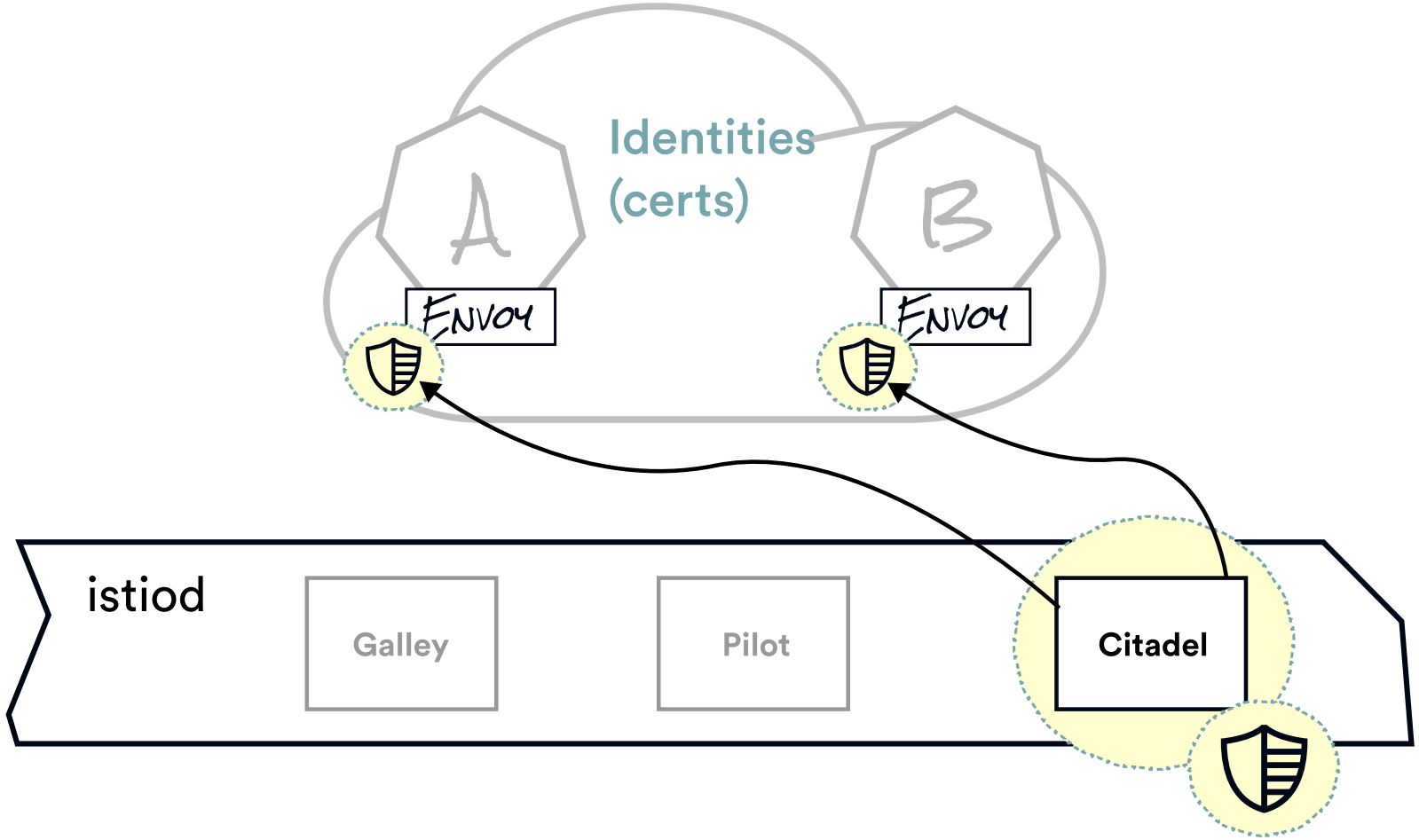
First logical component is Galley, which is responsible for validating incoming config.



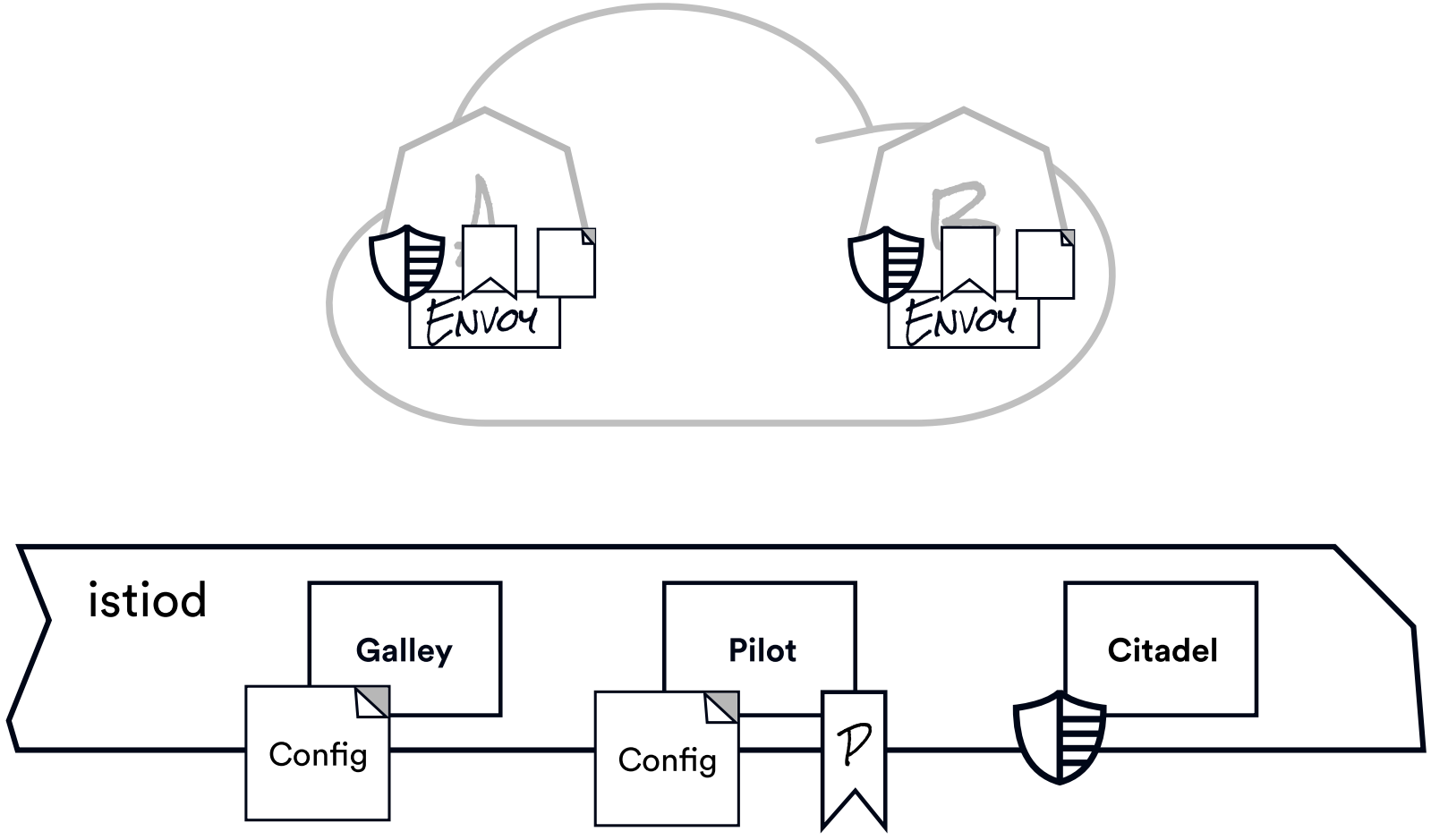
Pilot distributes the validated networking configuration to each Envoy



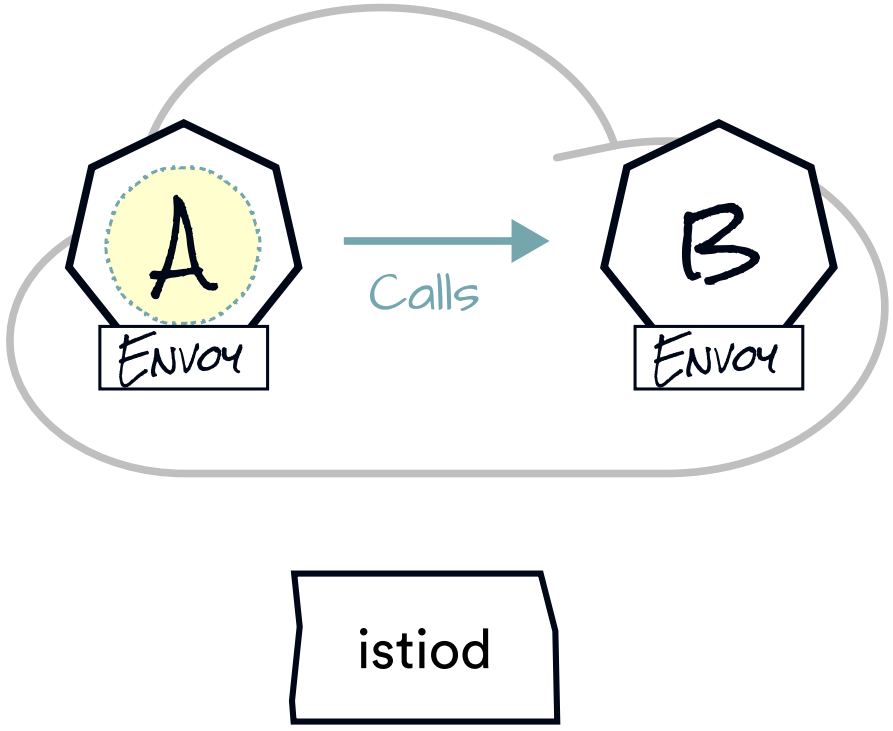
...and Pilot also distributes policy



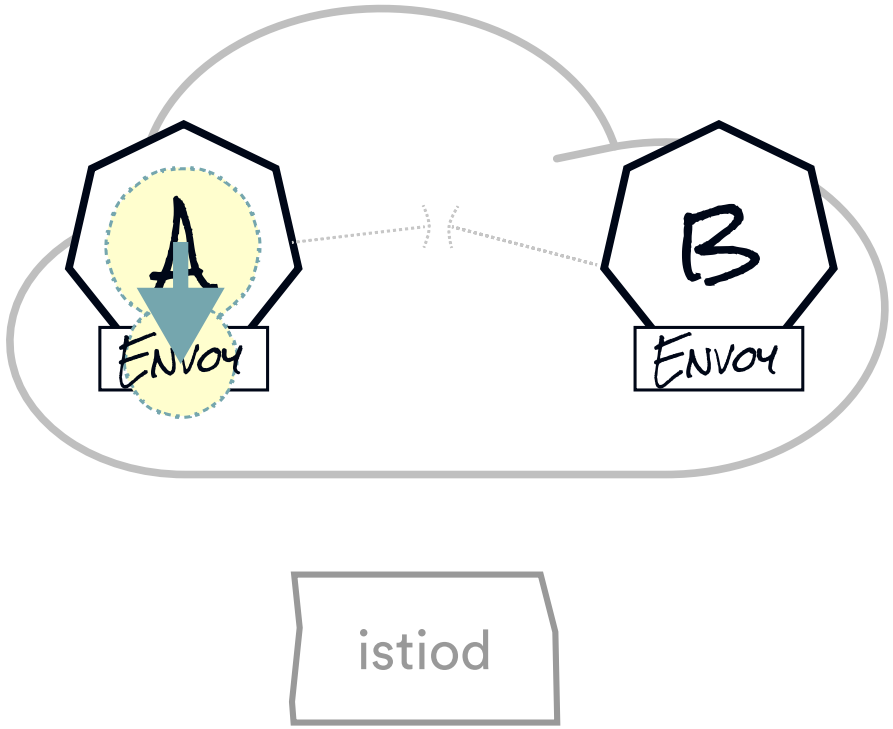
Citadel assigns SPIFFE identities to enable secure communication



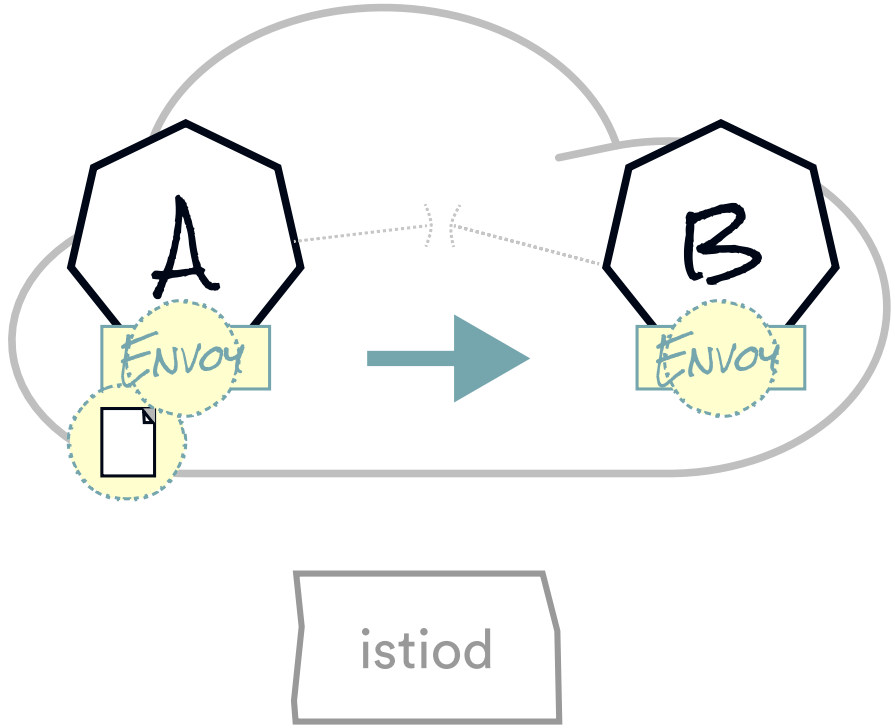
Control plane - Istiod



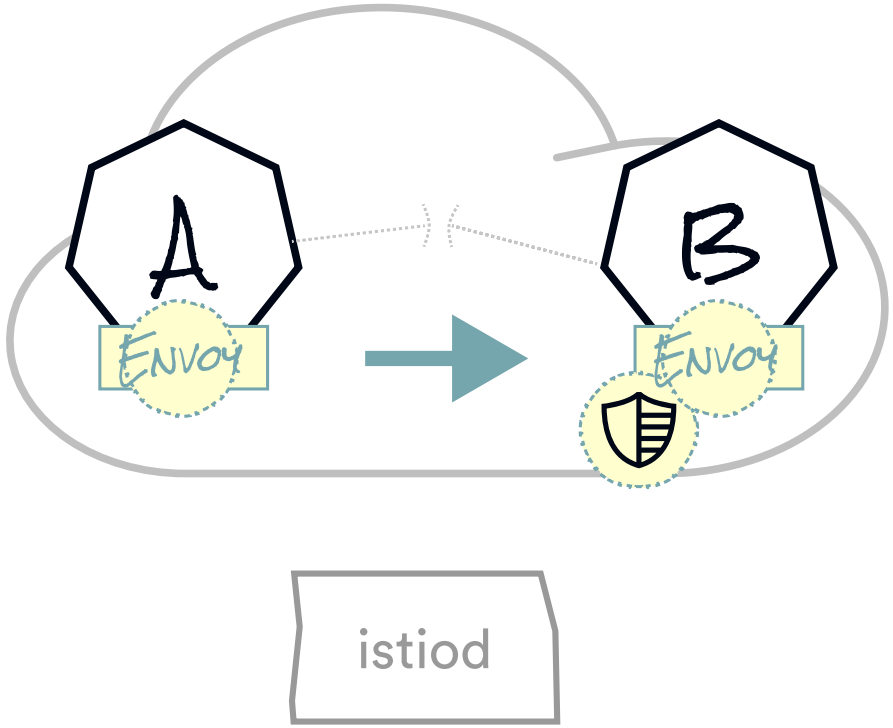
Now, let's track that call



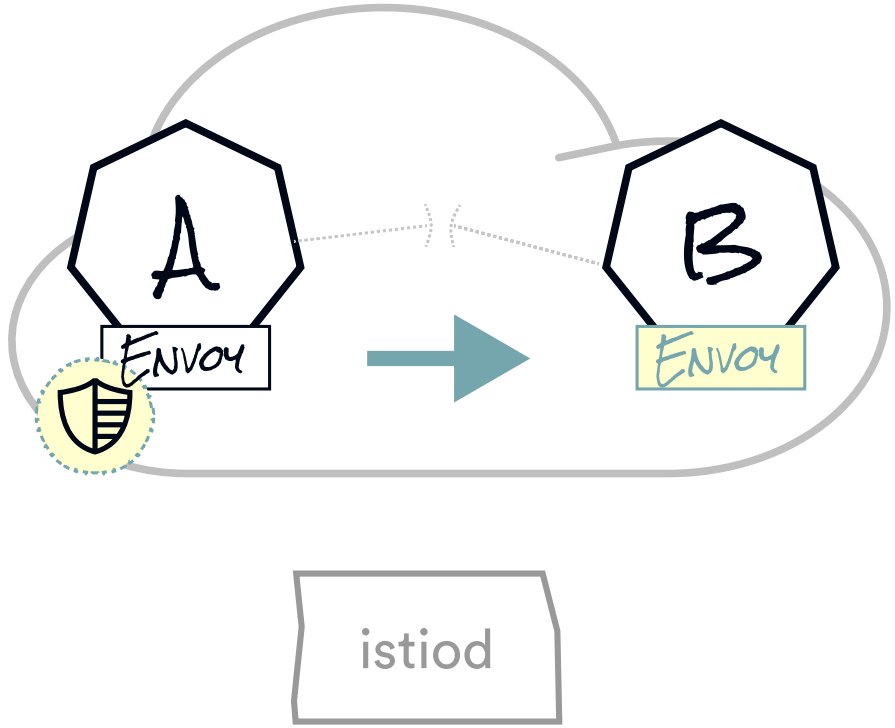
Envoy intercepts it



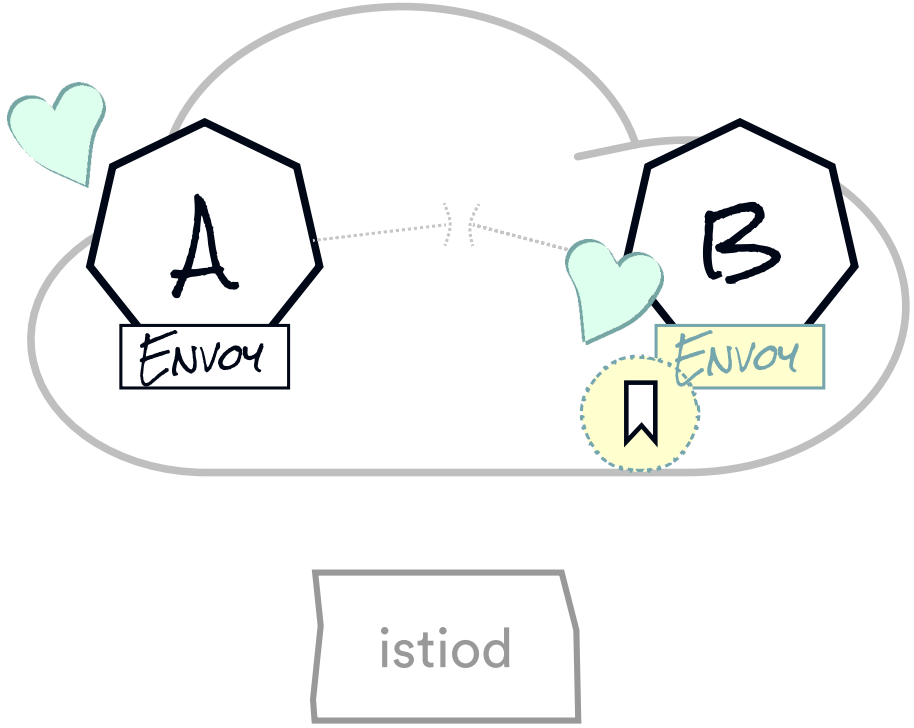
Uses the configuration to pick a new destination



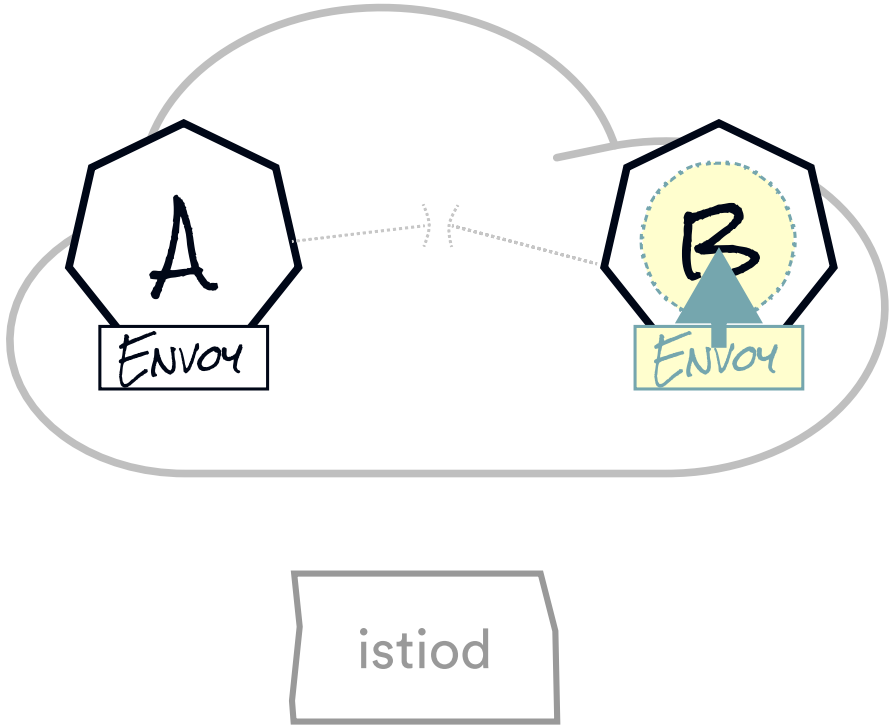
Verifies the destination's identity



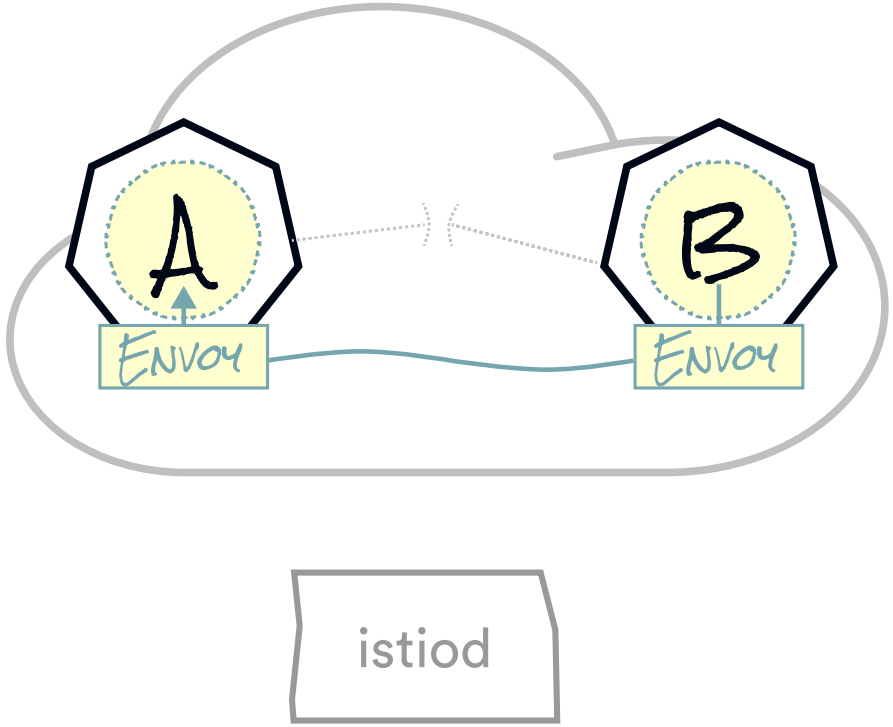
The receiving Envoy checks the sender's identity



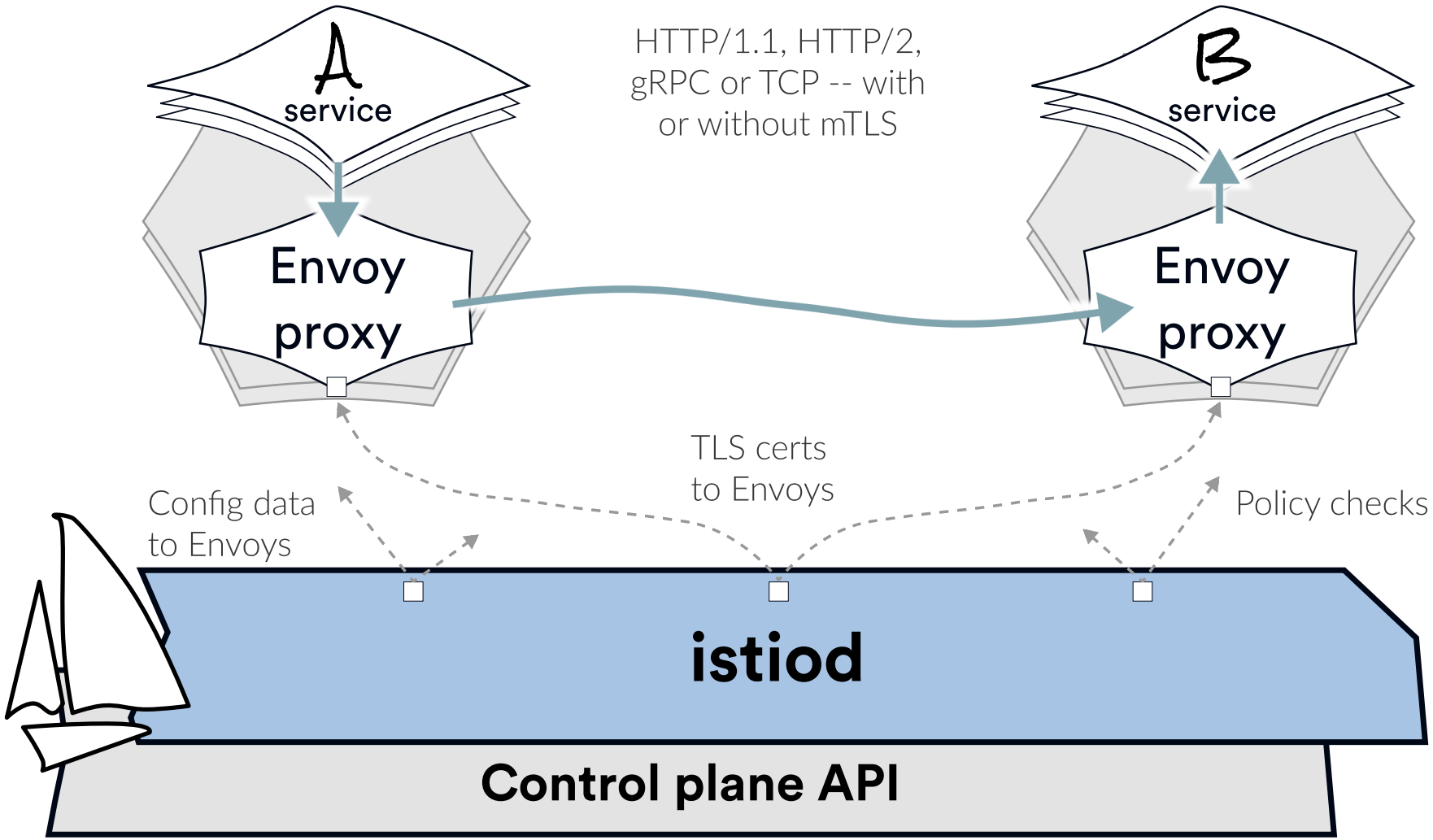
The receiving Envoy checks policy



Envoy hands the request to B



B answers



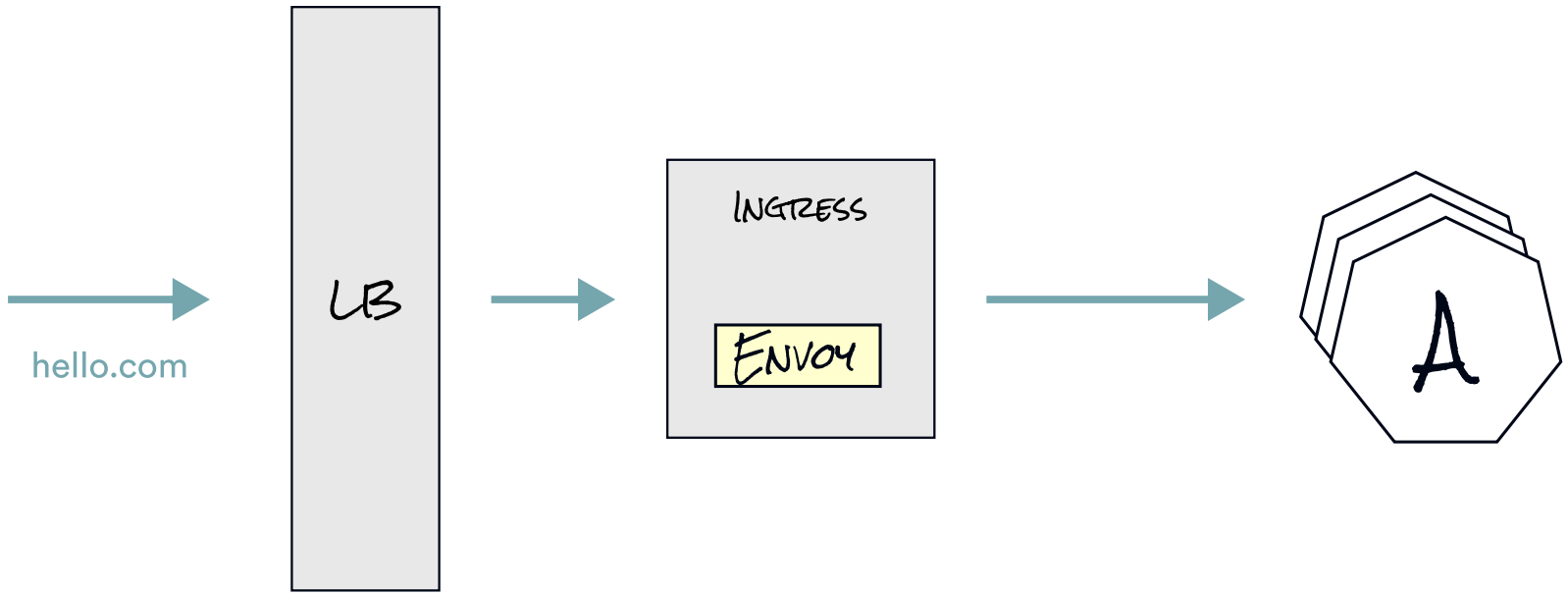
Istiod - Control plane Manager, handling

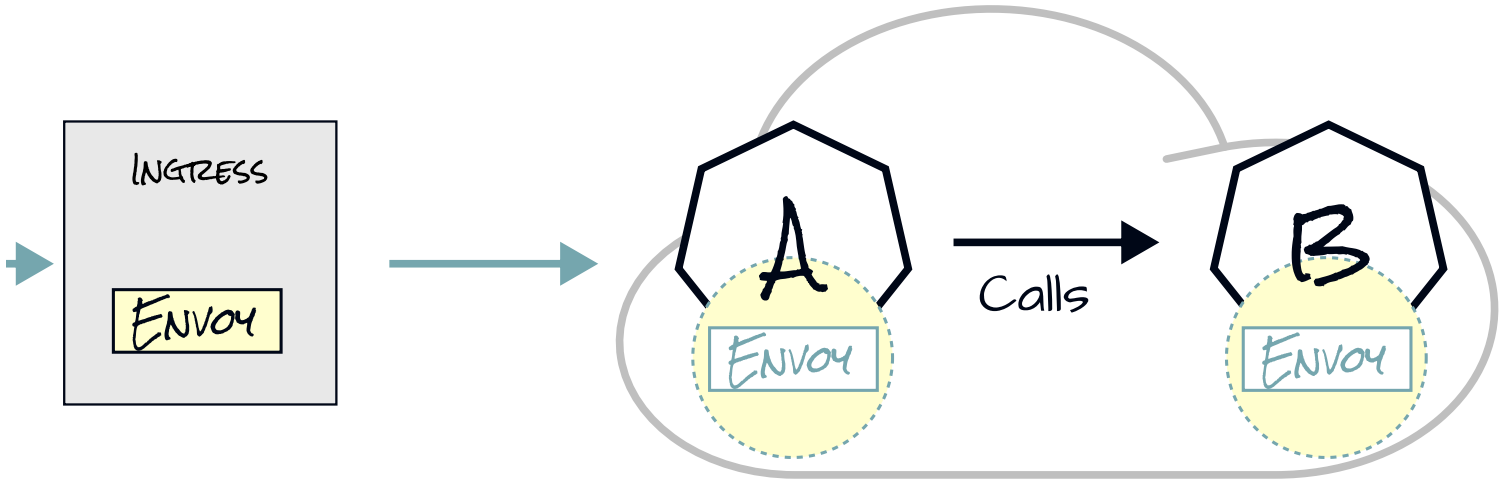
LAB

The application

https://tetratelabs.github.io/istio-0to60/the-app/

Ingress


hello.com
LB
INGRESS
ENVOY
A


INGRESS
ENVOY
A
ENVOY
Calls
B
ENVOY

What can we say about Ingress?

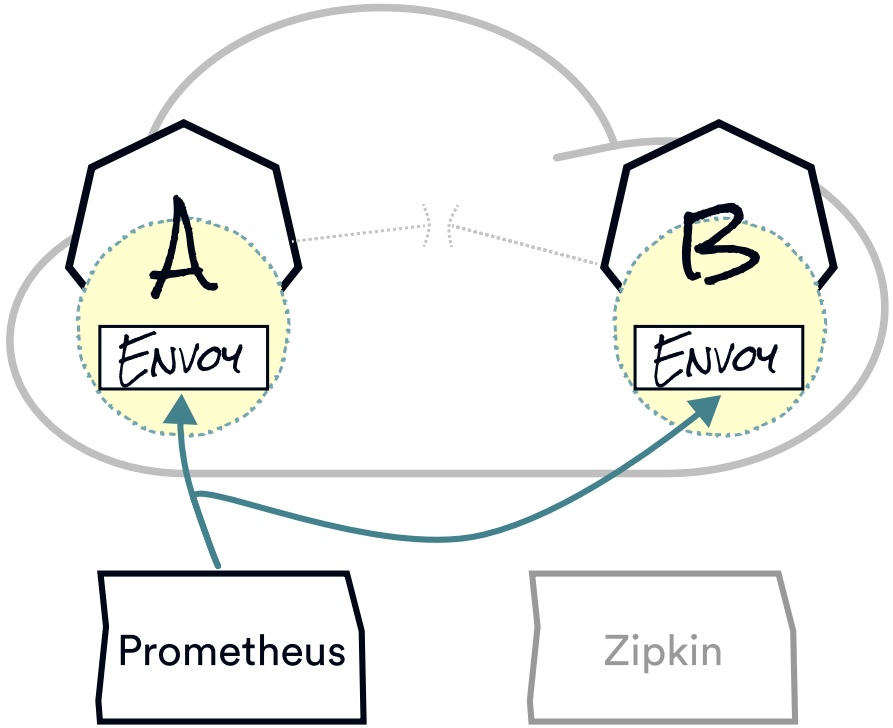
1. Ingress Pod runs in `istio-system` namespace
2. Configured logically in Kubernetes with the `Gateway` custom resource
3. Routing configured separately with `VirtualService` custom resource

LAB

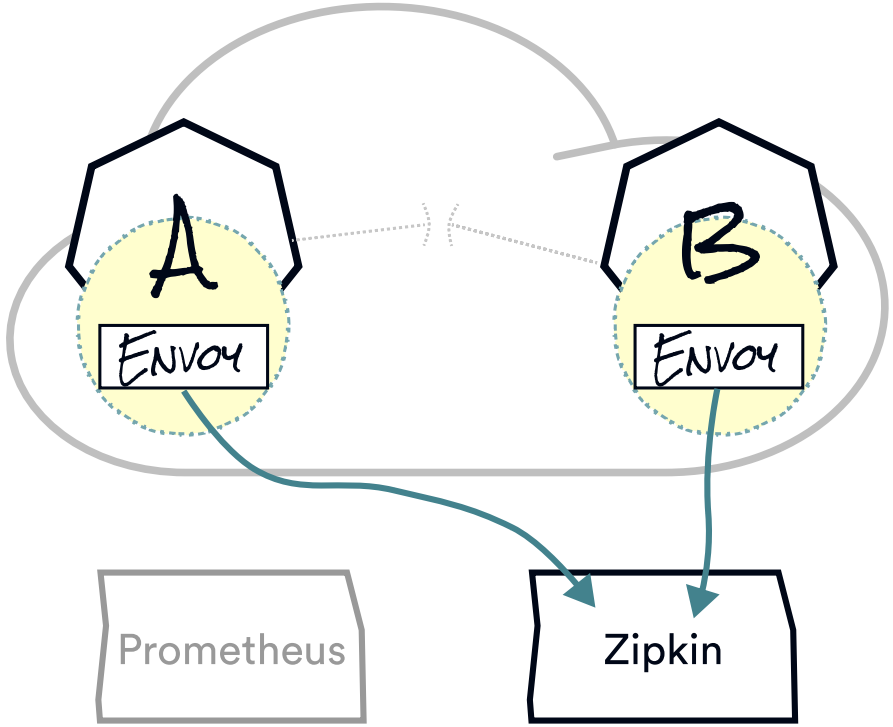
Ingress gateway

https://tetratelabs.github.io/istio-0to60/ingress/

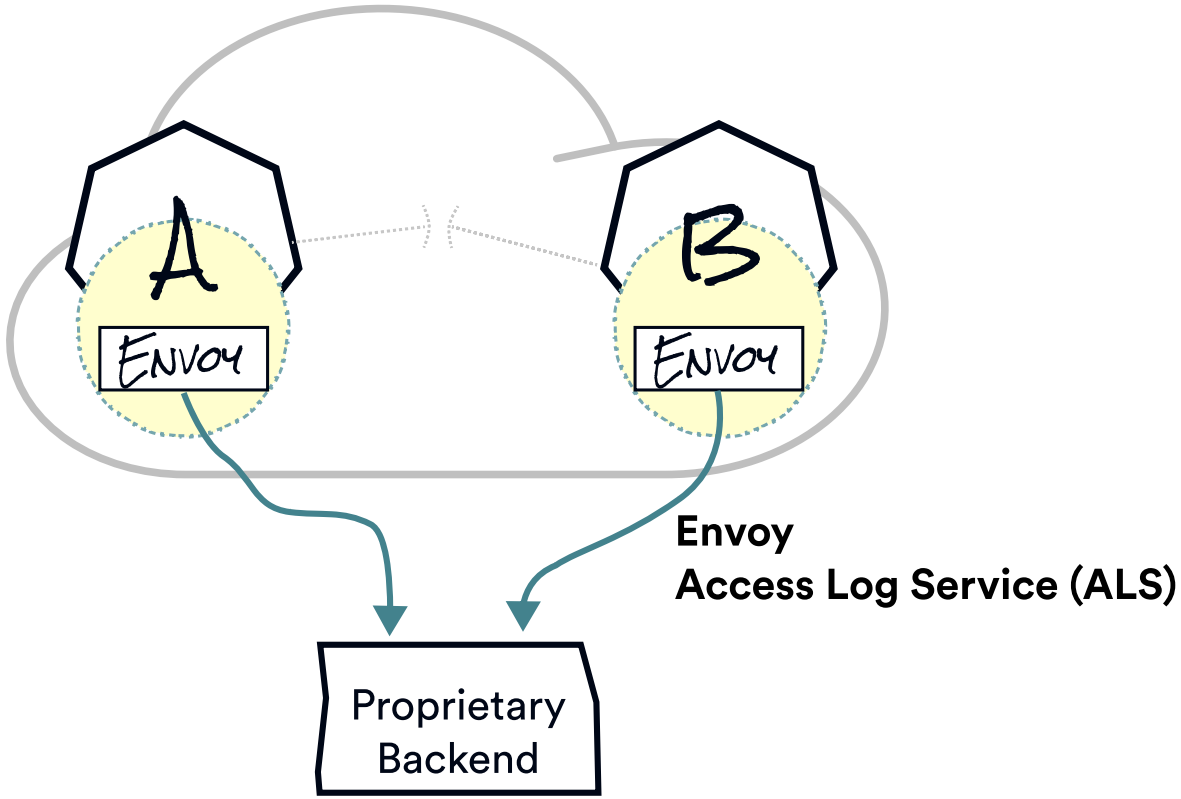
Observability



Metrics to Prometheus



Traces to Zipkin



Whatever you want via the Access Log Service

LAB

Observability

https://tetratelabs.github.io/istio-0to60/dashboards/

Secure your environment

Identifying workloads

AUTHENTICATION (AUTHN)

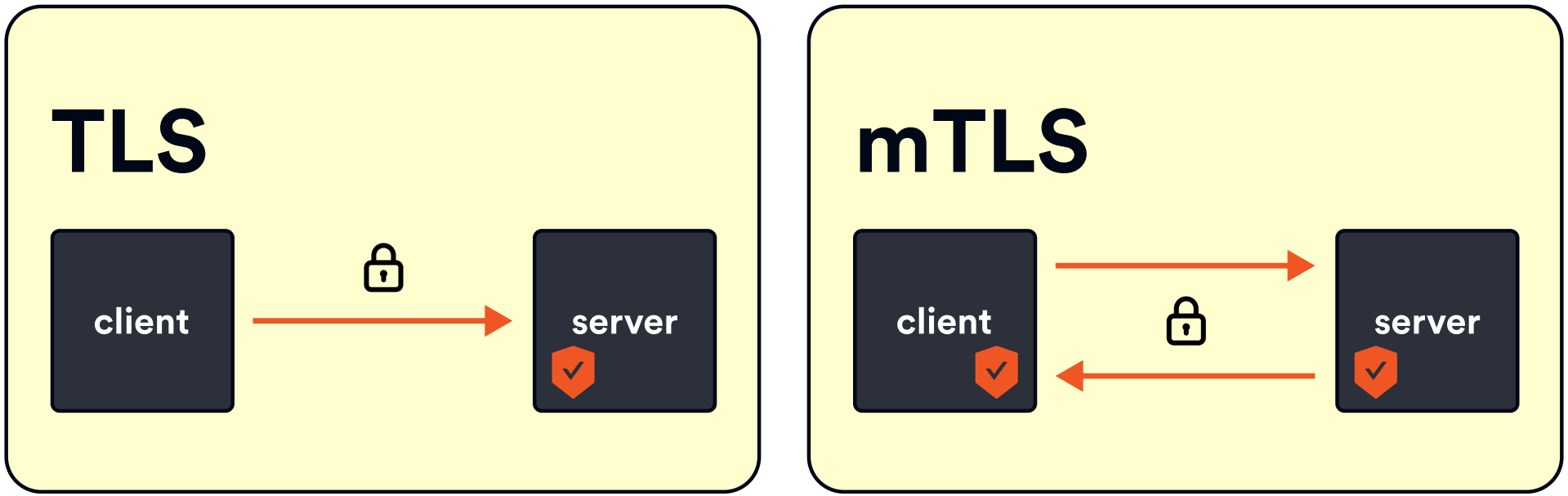
- Authn - all about the principal
- Each workload is assigned a unique identity that it uses to communicate with other workloads
  - Kubernetes = Istio uses service accounts

SPIFFE overview

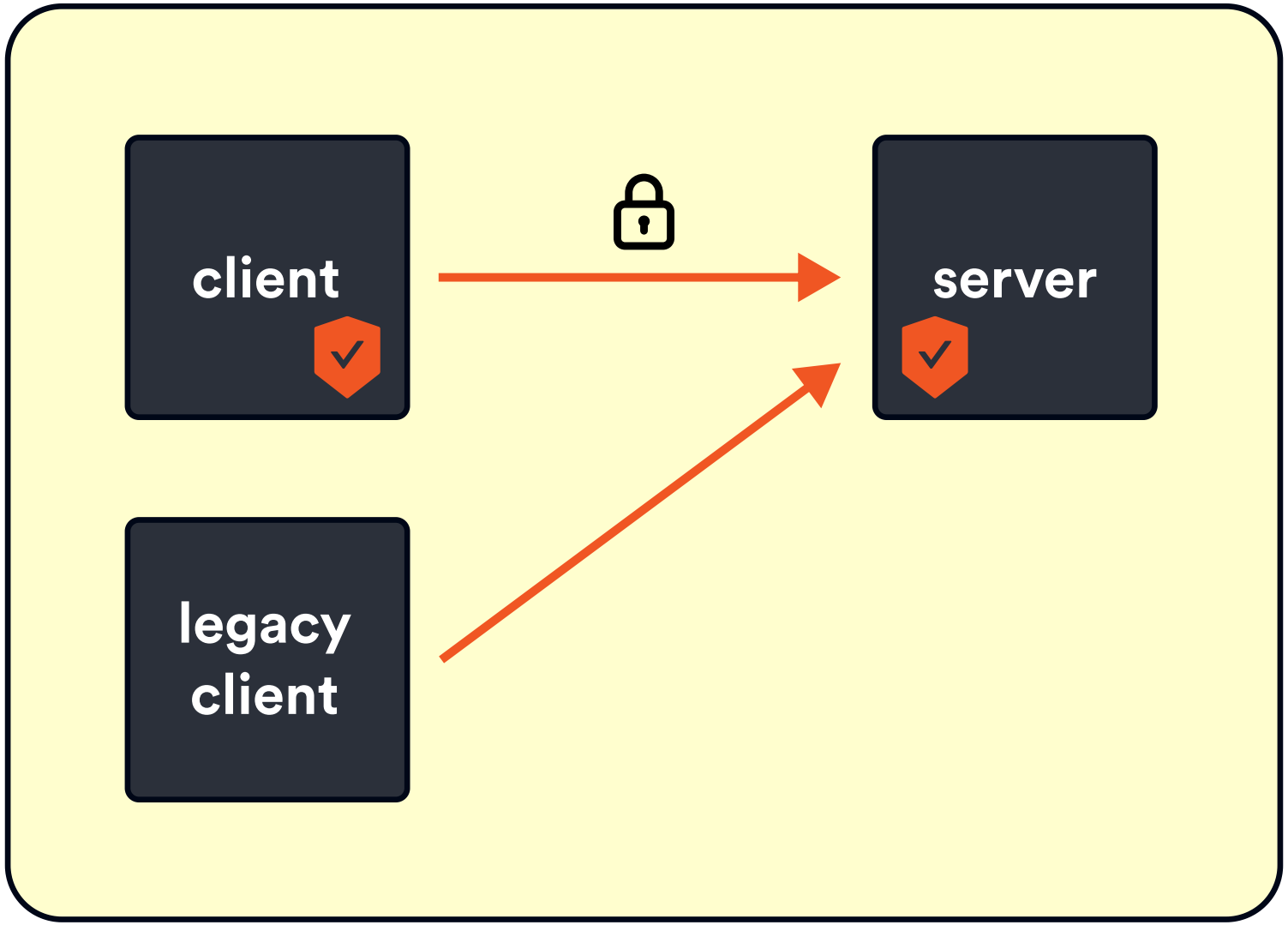
SECURE PRODUCTION IDENTITY FRAMEWORK FOR EVERYONE

- X.509 certificate (from SA) + SPIFFE spec = **IDENTITY**
- SPIFFE is a spec that describes:
  - A **naming scheme** for workload identities
  - `spiffe://cluster.local/ns/default/sa/my-sa`
  - How to **encode those names** into a X.509 certificate
  - How a client **Validates an X.509** certificate to authenticate the SPIFFE identity inside of it

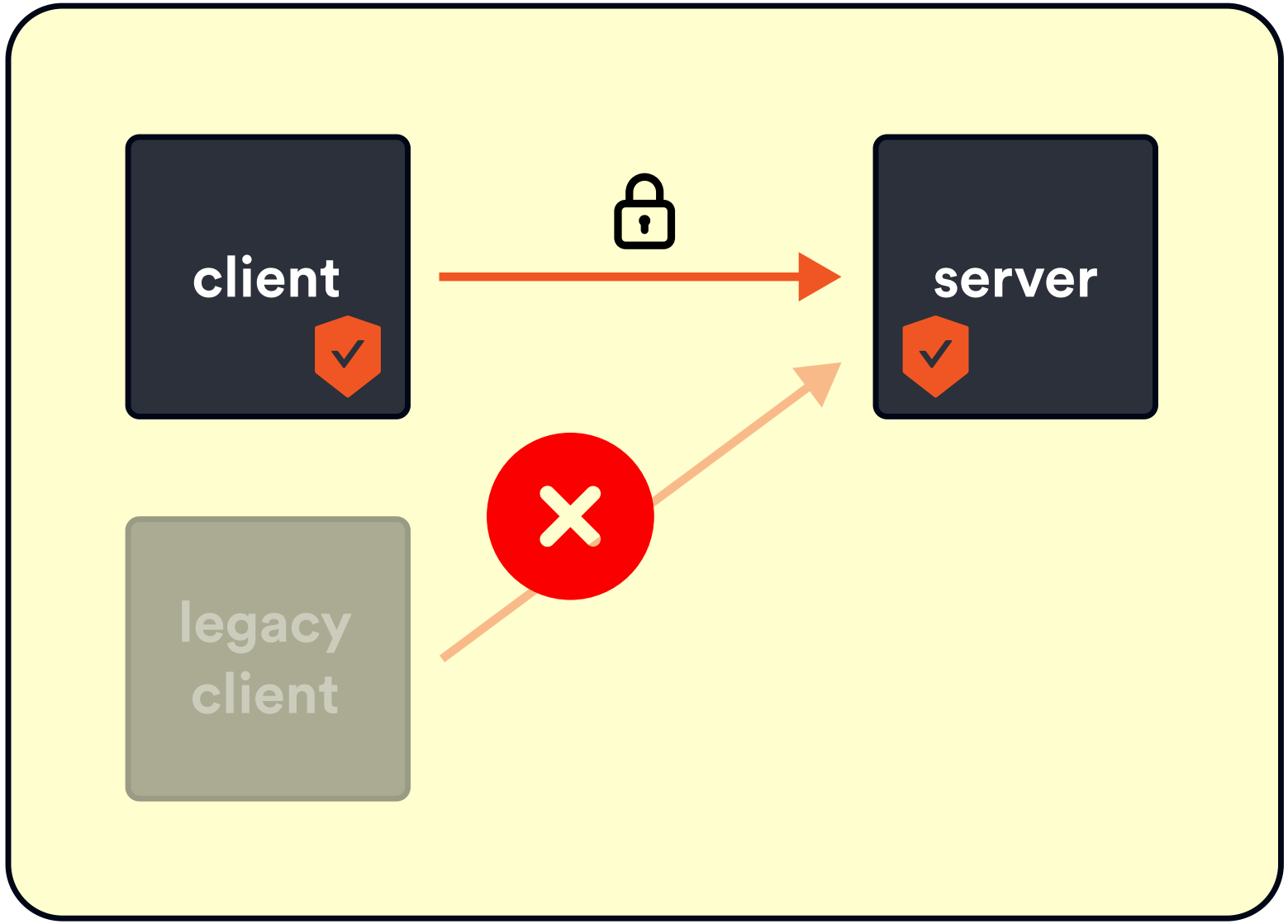
Mutual TLS (mTLS)



Permissive



Strict



Peer authentication

SERVICE-TO-SERVICE COMMUNICATION

- Controls communication between services
  - `PERMISSIVE` (default)
  - `STRICT`
- Mesh, namespace, workload, and port level

Peer authentication

NAMESPACE LEVEL

```yaml
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: foo
spec:
  mtls:
    mode: STRICT
```

Peer authentication

WORKLOAD LEVEL

```yaml
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: foo
spec:
  selector:
    matchLabels:
      app: prod
  mtls:
    mode: STRICT
```

Peer authentication

PORT LEVEL

```yaml
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: foo
spec:
  mtls:
    mode: STRICT
  portLevelMtls:
    5000:
      mode: DISABLE
```

What about users?

Request authentication

USER AUTHENTICATION

- Uses JWT tokens
- Mesh/namespace/workload scope
- Also at ingress level:
  - `forwardOriginalToken`

```yaml
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: httpbin
  namespace: default
spec:
  selector:
    matchLabels:
      app: httpbin
  jwtRules:
  - issuer: "issuer-foo"
    jwksUri: "someuri"
```

JWT authentication filter

- Authn enforced by the filter
- Doesn't deny requests without JWT tokens (*allowMissing*)
  - Used together with AuthorizationPolicy

```
name: envoy.filters.http.jwt_authn
typedConfig:
  providers:
    origins-0:
      issuer: testing@secure.istio.io
      localJwks:
        inlineString: '...'
      payloadInMetadata: testing@secure.istio.io
  rules:
  - match:
      prefix: "/"
    requires:
      requiresAny:
        requirements:
        - providerName: origins-0
        - allowMissing: {}
```

JWT authentication filter

WHEN ARE THE REQUESTS APPROVED/DENIED

DENIED

- Mismatching issuers
- Token expired
- Invalid audience (if provided)
- Invalid signature

APPROVED

- Valid JWT
- No JWT
  - Use AuthorizationPolicy

Authorization (authz)

CAN A PRINCIPAL PERFORM AN ACTION?

- Can user **A** send a **GET request** to path **/hello** on service **B**?
- Authn without authz (and vice-versa) is useless
- Control authenticated principals with **AuthorizationPolicy**

Authorization policy

- Make use of identities extracted from:
  - **Peer**Authentication -> *principals* (service/peer)
  - **Request**Authentication —> *requestPrincipals* (users)

Example

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: require-jwt
  namespace: default
spec:
  selector:
    matchLabels:
      app: prod
  rules:
  - from:
    - source:
        requestPrincipals: ["*"]
```

Authorization policy

RULES "FROM" FIELD

- source identities
- namespaces
- principals
- IP blocks and remote IP blocks

```
rules:
- from:
  - source:
      principals: ["cluster.local/ns/default/sa/workload"]
  - source:
      namespaces: ["prod"]
  - source:
      requestPrincipals: ["tetrate.io/peterj"]
```

Authorization policy

RULES "TO" FIELD

- hosts
- ports
- methods
- paths

```
rules:
- from:
  - ...
  to:
  - operation:
      methods: ["DELETE"]
      paths: ["/logs*"]
  - operation:
      methods: ["GET"]
      paths: ["/data"]
  - operation:
      hosts: ["request.host"]
      ports: ["3000", "5000"]
```

Authorization policy

"WHEN" FIELD (CONDITIONS)

- Keys and values (or notValues)
- Request attributes:
  - `request.headers`
  - `source.ip`, `remote.ip`
  - `source.namespace | principal`,
  - ...

```
rules:
- from:
  to:
  when:
    - key: request.auth.claims[iss]
      values: ["https://accounts.google.com"]
    - key: request.headers[my-header]
      values: ["some-value"]
    - key: source.namespace
      value: ["foo"]
    ...
```

Authorization policy

"ACTION" FIELD

- CUSTOM
- DENY
- ALLOW
- AUDIT

```
spec:
  action: DENY
  rules:
  - from:
    to:
    when:
    ...
```

Recap

- Services → PeerAuthentication
- Users → RequestAuthentication
- Access control rules → AuthorizationPolicy
  - From, To, When

LAB

Security

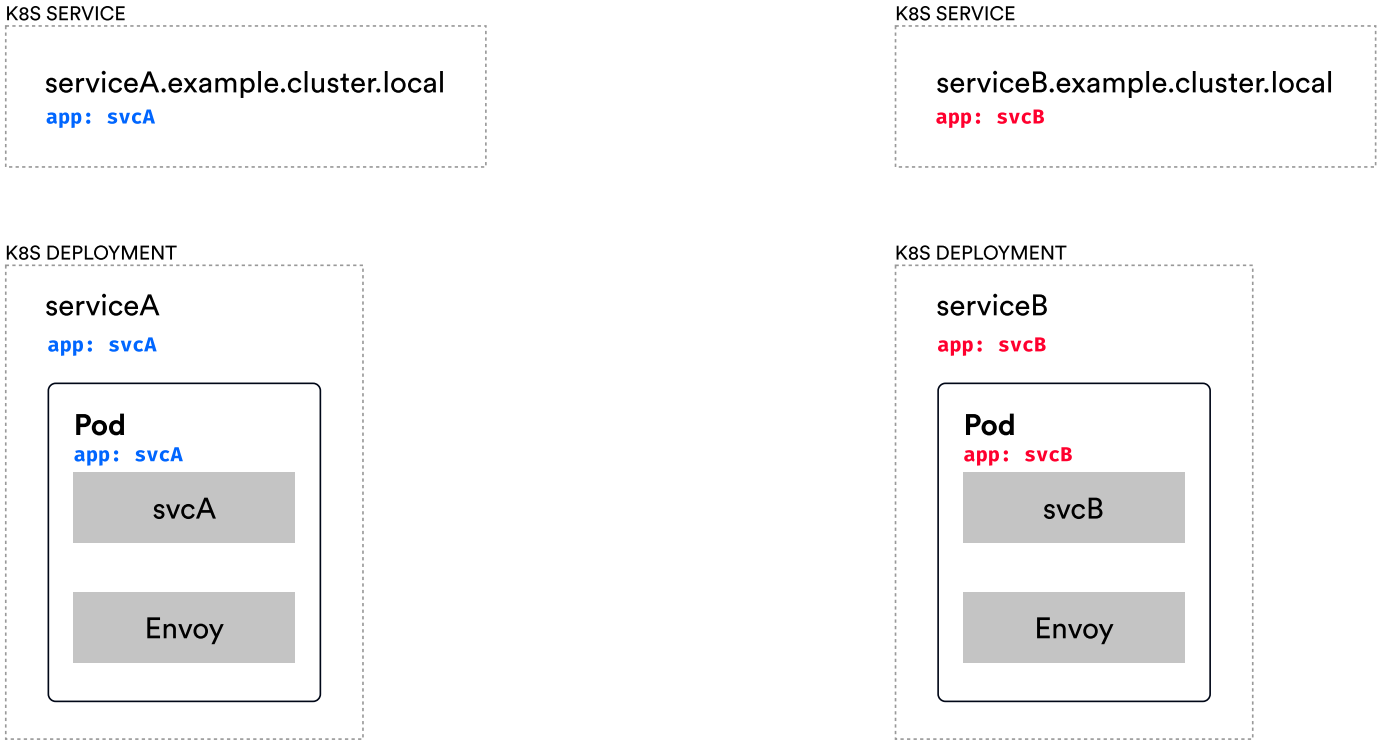
https://tetratelabs.github.io/istio-0to60/security/

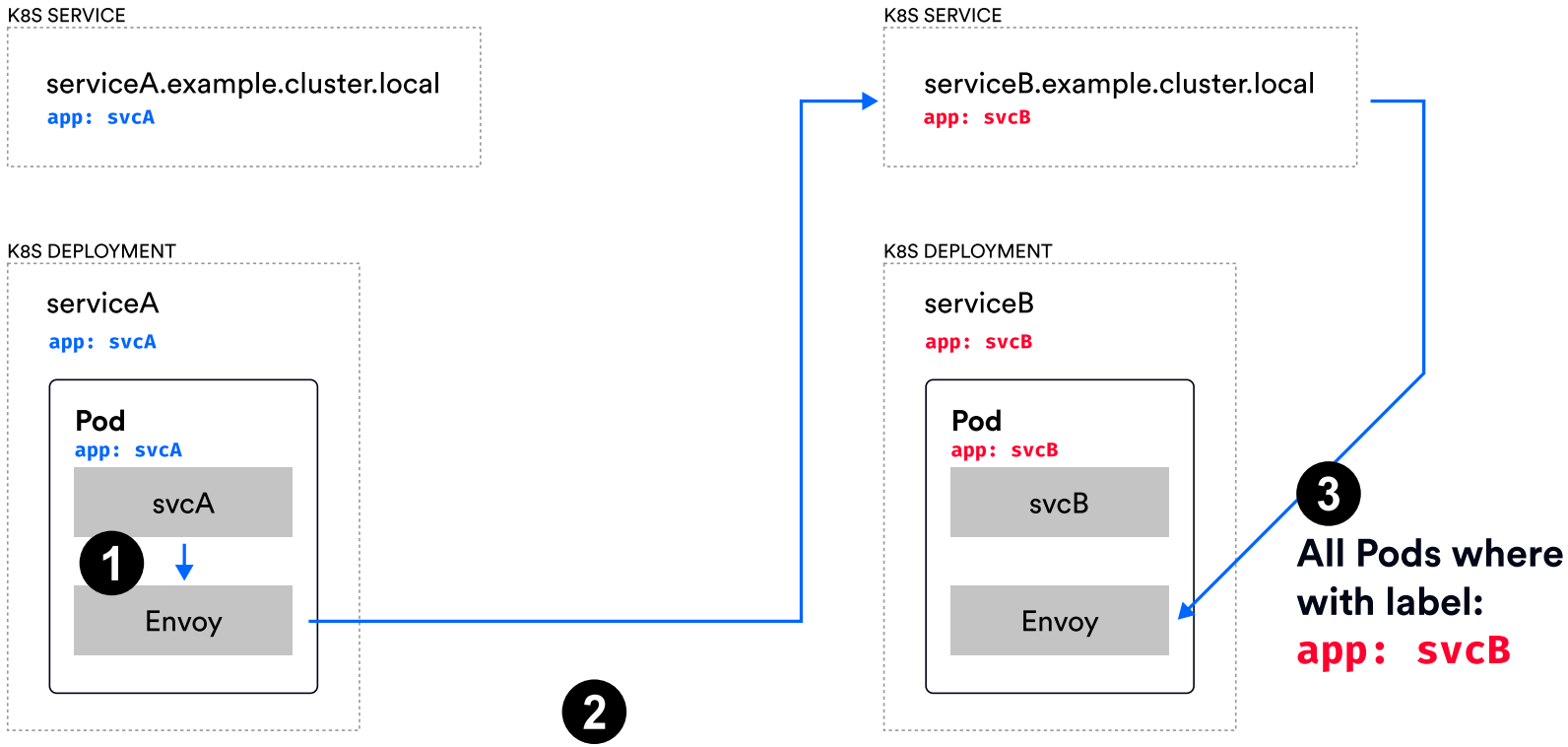
Istio Traffic Management

CUSTOM RESOURCES (1/2)

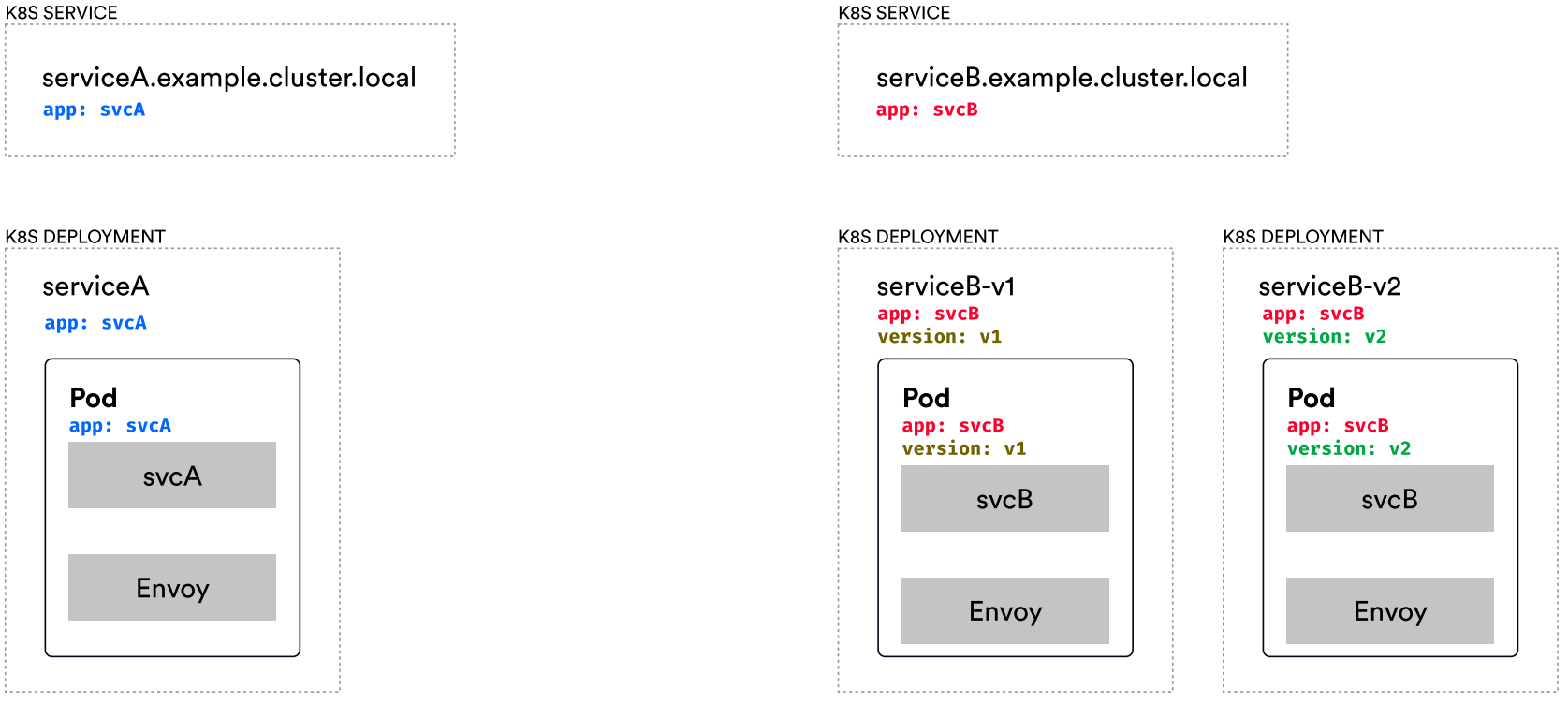
- **Virtual service**
  - Configure routing rules for each service
- **Destination rule**
  - Configure how to reach the target endpoint, applied after routing decision has been made

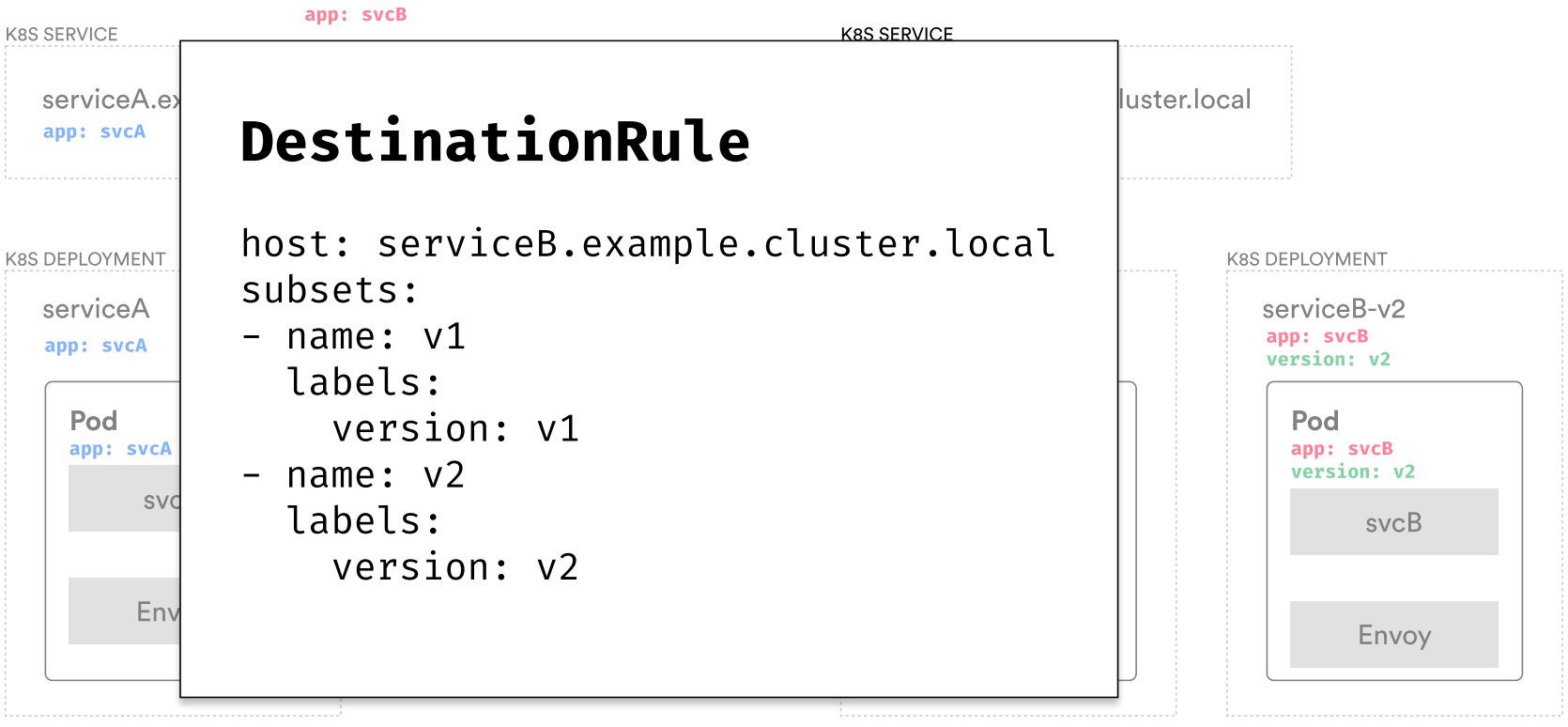
How to route the request?


K8S SERVICE
serviceA.example.cluster.local
app: svcA
K8S SERVICE
serviceB.example.cluster.local
app: svcB
K8S DEPLOYMENT
serviceA
app: svcA
Pod
app: svcA
svcA
Envoy
K8S DEPLOYMENT
serviceB
app: svcB
Pod
app: svcB
svcB
Envoy



http://serviceb.example.cluster.local


K8S SERVICE
serviceA.example.cluster.local
app: svcA
K8S SERVICE
serviceB.example.cluster.local
app: svcB
K8S DEPLOYMENT
serviceA
app: svcA
Pod
app: svcA
svcA
Envoy
K8S DEPLOYMENT
serviceB-v1
app: svcB
version: v1
Pod
app: svcB
version: v1
svcB
Envoy
K8S DEPLOYMENT
serviceB-v2
app: svcB
version: v2
Pod
app: svcB
version: v2
svcB
Envoy


app: svcB
K8S SERVICE
serviceA.ex
app: svcA
K8S SERVICE
luster.local
DestinationRule
host: serviceB.example.cluster.local
subsets:
- name: v1
labels:
version: v1
- name: v2
labels:
version: v2
K8S DEPLOYMENT
serviceA
app: svcA
Pod
app: svcA
svc
Env
K8S DEPLOYMENT
serviceB-v2
app: svcB
version: v2
Pod
app: svcB
version: v2
svcB
Envoy

app: svcB

K8S SERVICE

VirtualService

```
hosts:
 - serviceB.example.cluster.local
http:
- route:
  - destination:
      host: serviceB.example ...
      subset: v1
    weight: 70
  - destination:
      host: serviceB.example ...
      subset: v2
    weight: 30
```

K8S SERVICE

DestinationRule

```
host: serviceB.example.cluster.local
subsets:
- name: v1
  labels:
    version: v1
- name: v2
  labels:
    version: v2
```

DestinationRule & virtualService


70% to subset v1
30% to subset v2

K8S SERVICE

serviceA.example.cluster.local
app: svcA

K8S SERVICE

serviceB.example.cluster.local
app: svcB

K8S DEPLOYMENT

serviceA
app: svcA

Pod
app: svcA

svcA

Envoy

K8S DEPLOYMENT

serviceB-v1
app: svcB
version: v1

Pod
app: svcB
version: v1

svcB

Envoy

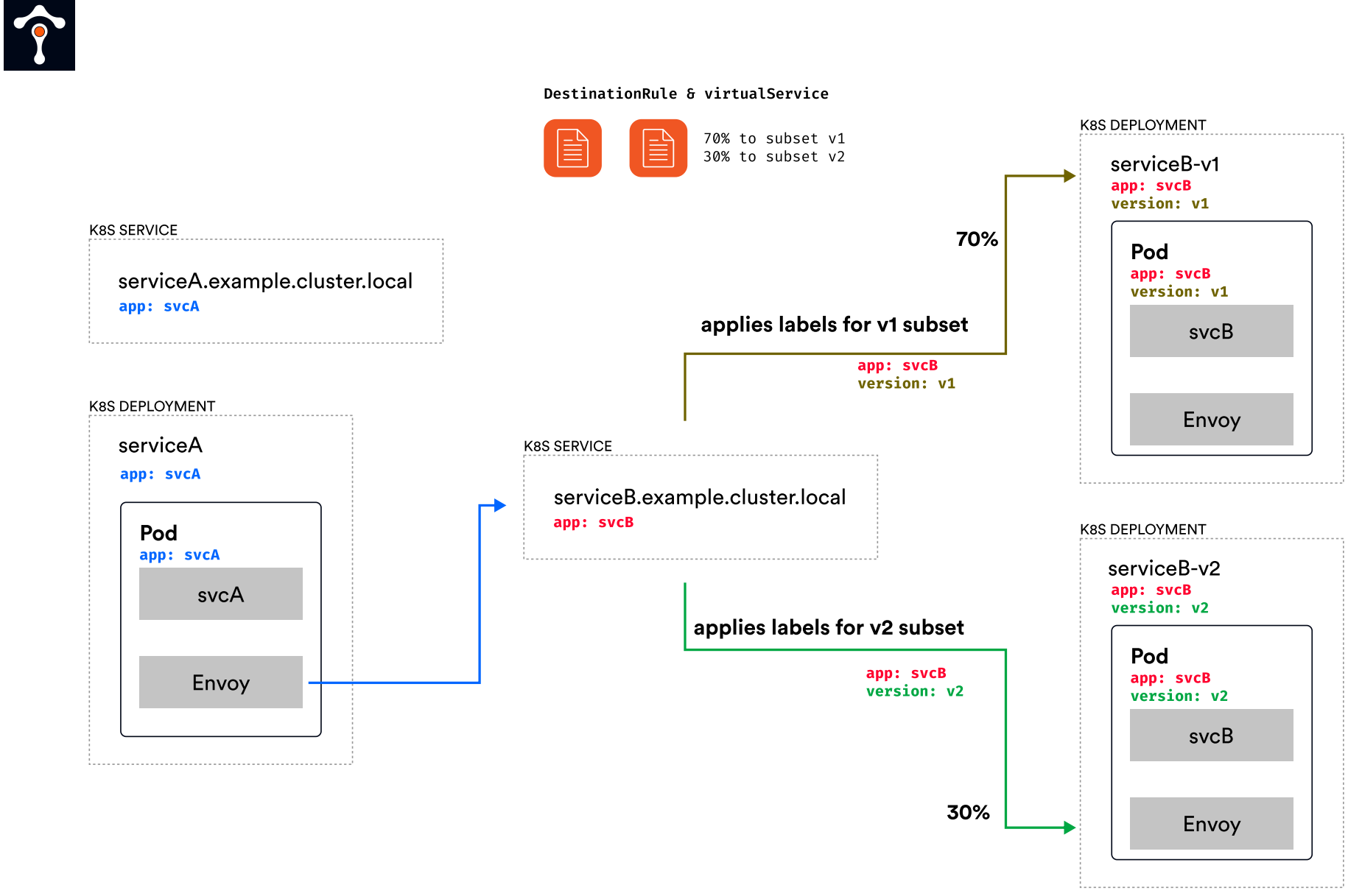
K8S DEPLOYMENT

serviceB-v2
app: svcB
version: v2

Pod
app: svcB
version: v2

svcB

Envoy


DestinationRule & virtualService
70% to subset v1
30% to subset v2
K8S SERVICE
serviceA.example.cluster.local
app: svcA
K8S DEPLOYMENT
serviceA
app: svcA
Pod
app: svcA
svcA
Envoy
K8S SERVICE
serviceB.example.cluster.local
app: svcB
70%
applies labels for v1 subset
app: svcB
version: v1
K8S DEPLOYMENT
serviceB-v1
app: svcB
version: v1
Pod
app: svcB
version: v1
svcB
Envoy
applies labels for v2 subset
app: svcB
version: v2
30%
K8S DEPLOYMENT
serviceB-v2
app: svcB
version: v2
Pod
app: svcB
version: v2
svcB
Envoy

DestinationRule

POLICIES APPLIED TO TRAFFIC FOR A SPECIFIC SERVICE

- Subsets = represent different service versions
- Traffic policies:
  - Load balancer settings (`ROUND_ROBIN`, `LEAST_CONN`, `RANDOM`, `PASSTHROUGH`)
  - Connection pool settings (TCP and HTTP)
  - Outlier detection
  - TLS

Connection pool settings

CONTROL THE VOLUME OF CONNECTIONS

- Applied to TCP and/or HTTP connections
- Timeouts
- Max connections/requests
- Max retries

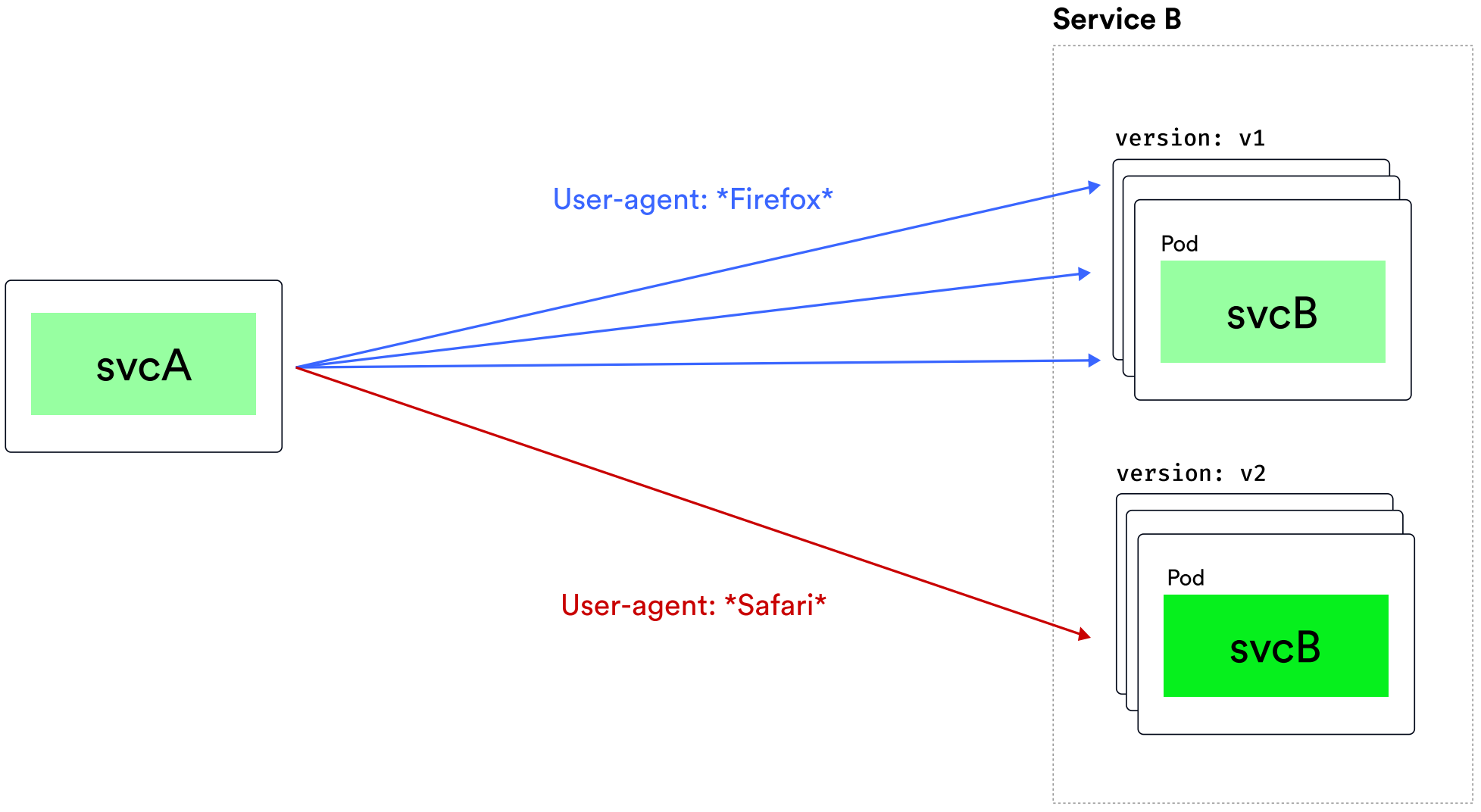
Outlier detection

HOW TO EJECT UNHEALTHY HOSTS

- When to eject unhealthy hosts?
  - e.g. `consecutive5xxErrors`, `consecutiveGatewayErrors`
- How long to eject them for?
  - `baseEjectionTime`
- How many hosts can be ejected?
  - `maxEjectionPercent`
- When to enable ejection?
  - `minHealthyPercent`

VirtualService

- Routing rules (TCP, HTTP, non-terminated TLS/HTTPS traffic)
  - Match & route
  - Redirect
  - Rewrite
  - Mirroring
  - Cors, Timeouts, retries, and fault injection
- Header manipulation


Service B
version: v1
User-agent: *Firefox*
svcA
Pod
svcB
version: v2
Pod
svcB
User-agent: *Safari*

Match on headers

```
hosts:
  - svcB.example.cluster.local
http:
- match:
  - headers:
      user-agent:
        regex: ".*Firefox.*"
  route:
  - destination:
      host: svcB.example.cluster.local
      subset: v1
- route:
  - destination:
      host: svcB.example.cluster.local
      subset: v2
```

AND semantics

```
hosts:
  - svcB.example.cluster.local
http:
- match:
  - headers:
      x-debug:
        exact: dev
    uri:
      prefix: /api/debug
  route:
  - destination:
      host: svcB.example.cluster.local
      subset: v1
- route:
  - destination:
      host: svcB.example.cluster.local
      subset: v2
```

OR semantics

```
hosts:
  - svcB.example.cluster.local
http:
- match:
  - headers:
      x-debug:
        exact: dev
  - uri:
      prefix: /api/debug
  route:
  - destination:
      host: svcB.example.cluster.local
      subset: v1
- route:
  - destination:
      host: svcB.example.cluster.local
      subset: v2
```

Timeout, retries

```
hosts:
  - svcB.example.cluster.local
http:
- route:
  - destination:
      host: svcB.example.cluster.local
      subset: v1
    weight: 30
    timeout: 5s
  - destination:
      host: svcB.example.cluster.local
      subset: v2
    weight: 70
    timeout: 0.5s
  retries:
    attempts: 3
    perTryTimeout: 2s
    retryOn: connect-failure
```

LAB

Traffic shifting

https://tetratelabs.github.io/istio-0to60/traffic-shifting/