Google Cloud
SUMMIT

What is Istio ?

Abdellfetah SGHIOUAR

Strategic Cloud Engineer @Google Stockholm
Twitter: boredabdel@

Today we're going to cover the why, what, and how of Istio

Why Istio ?

Google Cloud

Containers?

Great!

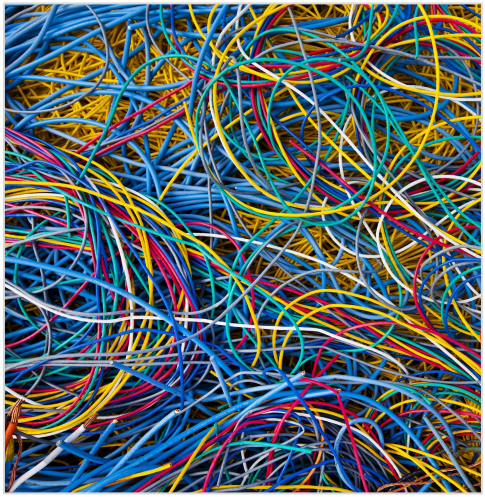
Google Cloud

Kubernetes?

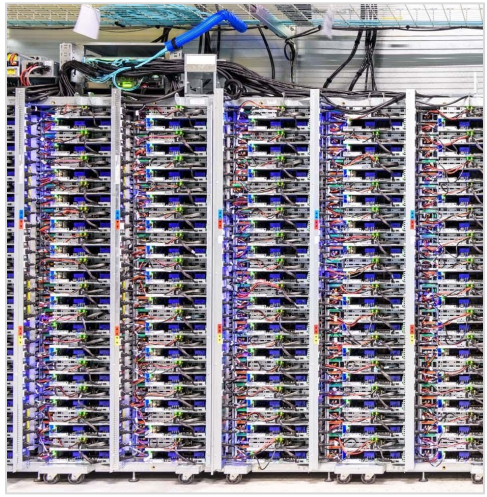
Also Great!

Google Cloud

But microservices deployments are hard



How do you go from a bunch of services...



...to a well organized and functioning deployment?

Google Cloud

What makes microservices difficult?

Canary releases require infrastructure scaling

Don't know if you're talking to the right service

Digging into high latency drivers takes work

Can't control traffic using L7 attributes

Can't control who accesses what service

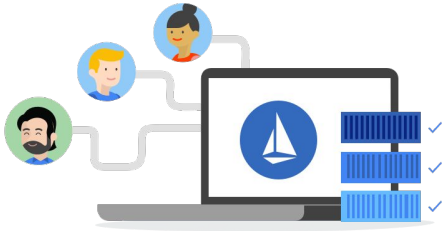
Can't track requests from start to finish

What is Istio?

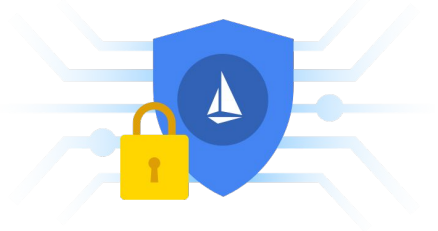
Google Cloud

How does Istio help?

Uniform observability



Operational agility



Policy driven security

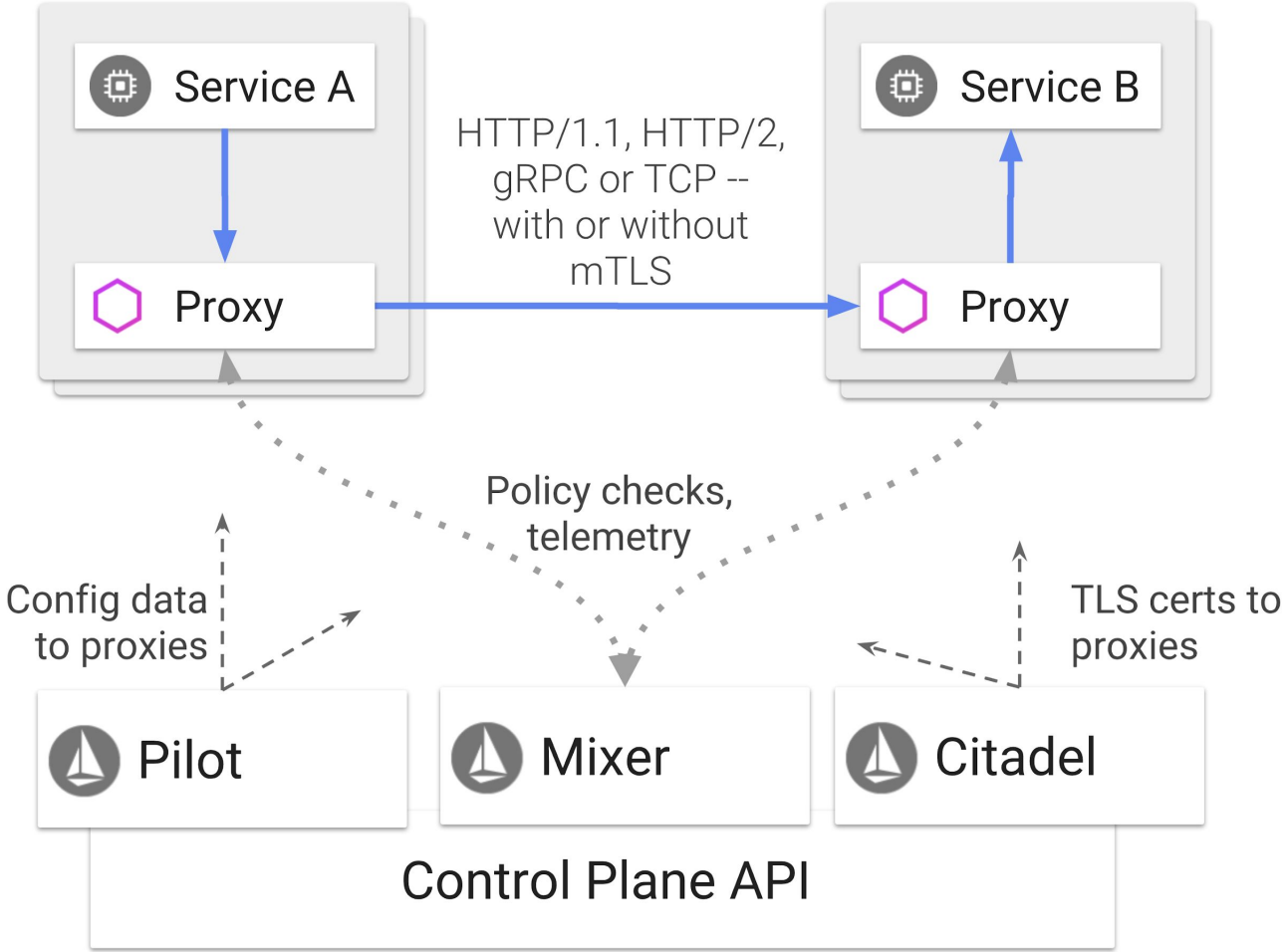
How does Istio help?

Examine **everything** happening with your services with little to no instrumentation

Manage the flow of traffic **into**, **out of**, and **within** your complex deployments

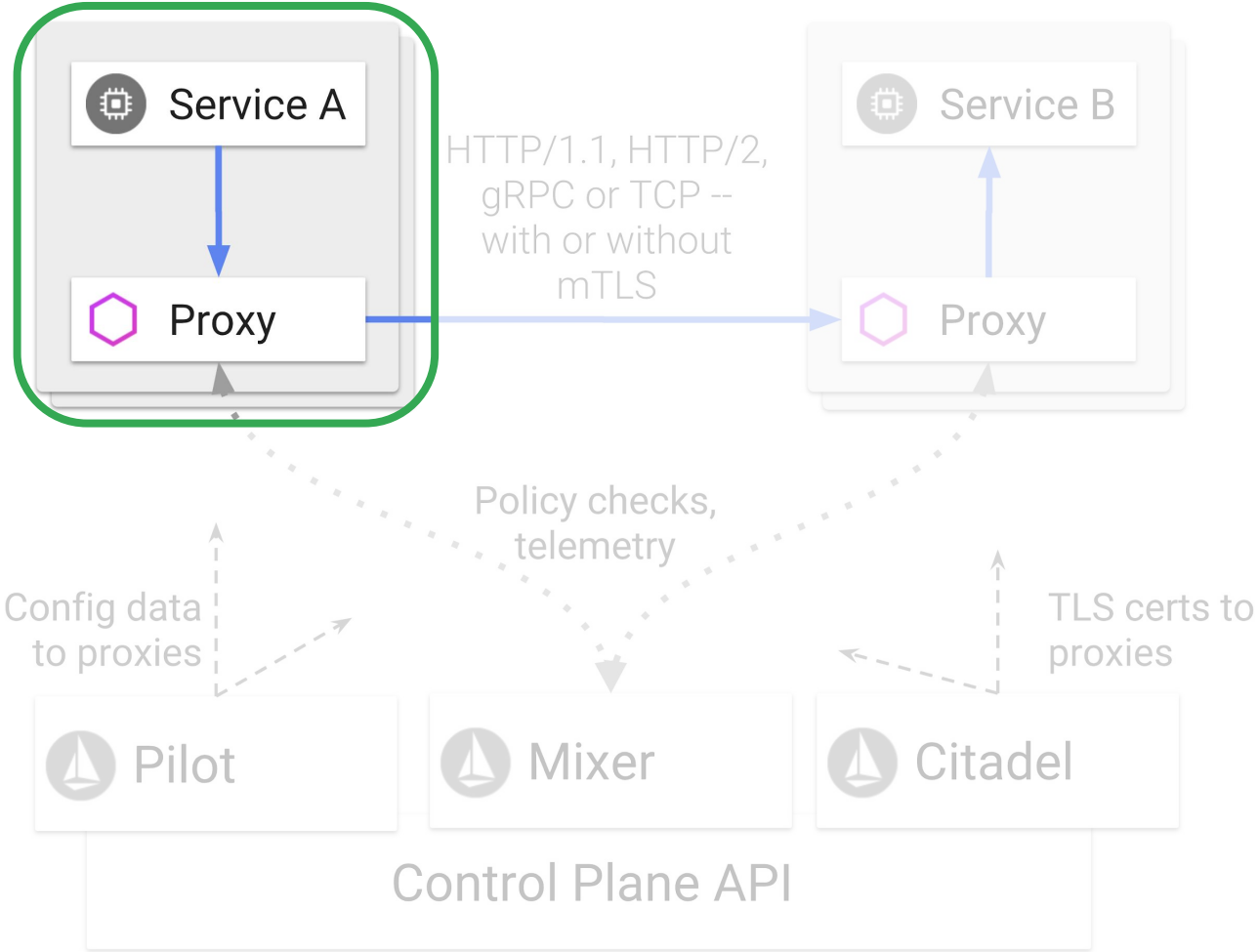
Secure **access** and **communications** between some or all services

Istio at a glance



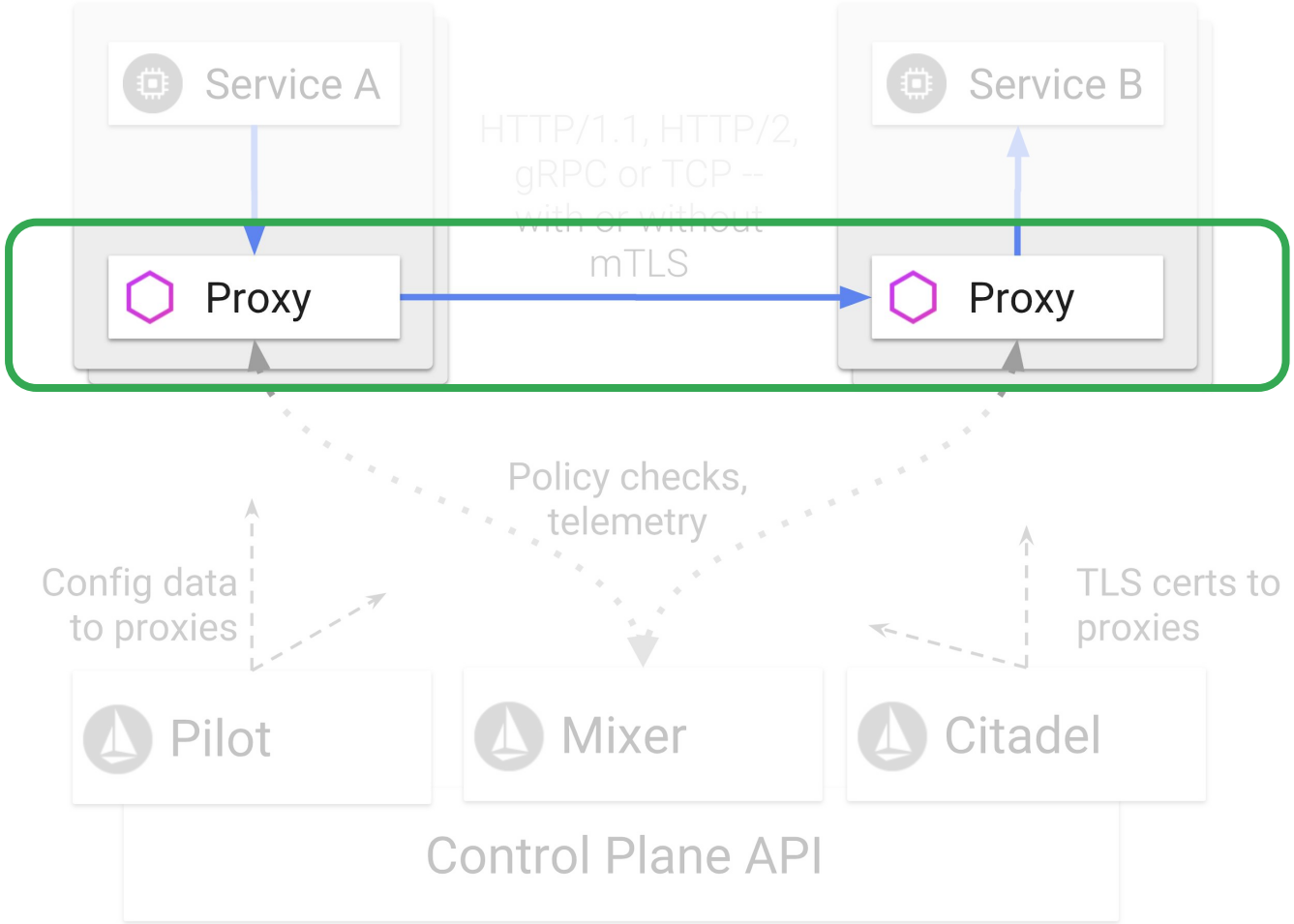
Google Cloud

Istio sidecar proxy



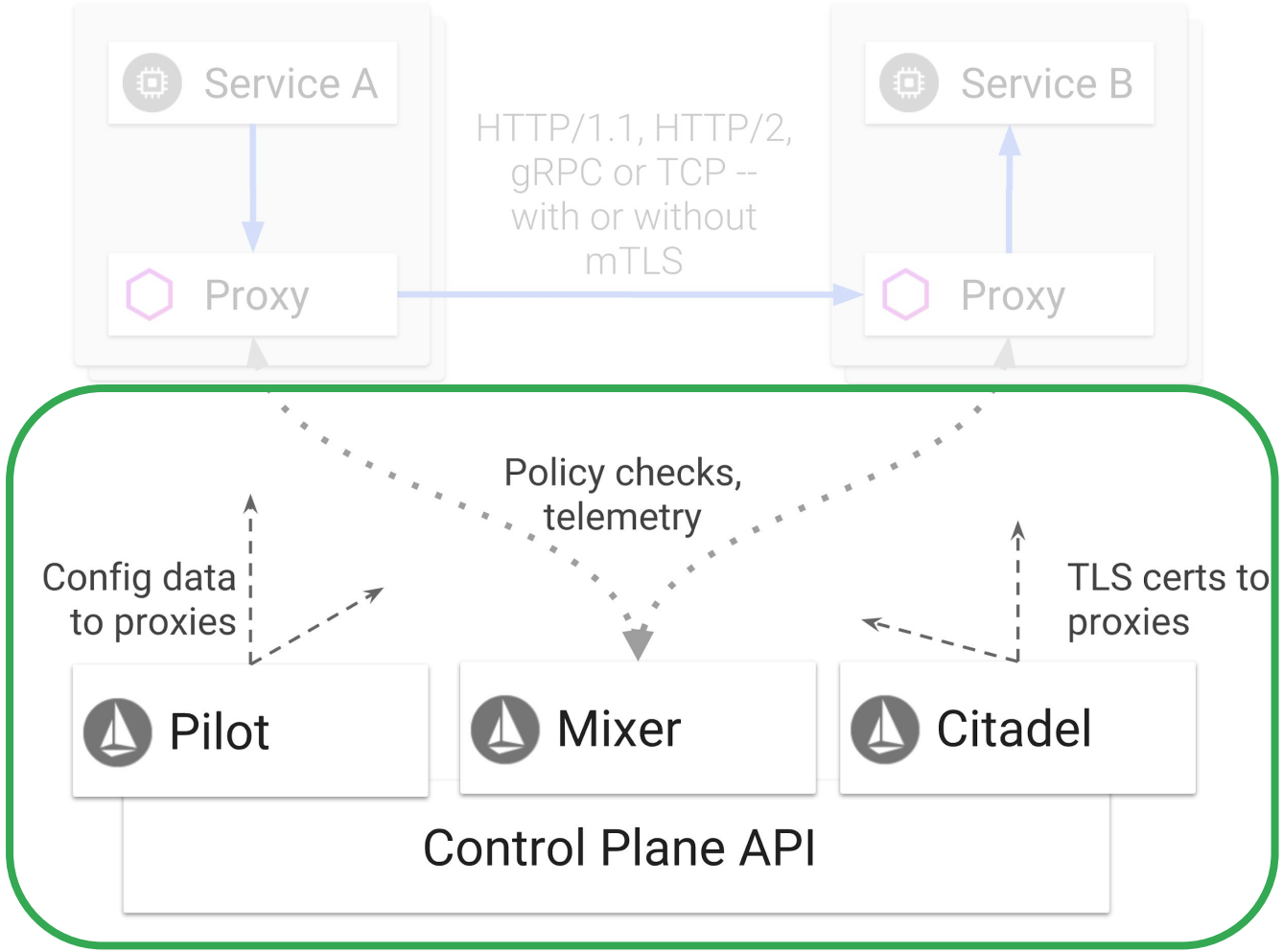
Google Cloud

Istio data plane



Google Cloud

Istio control plane



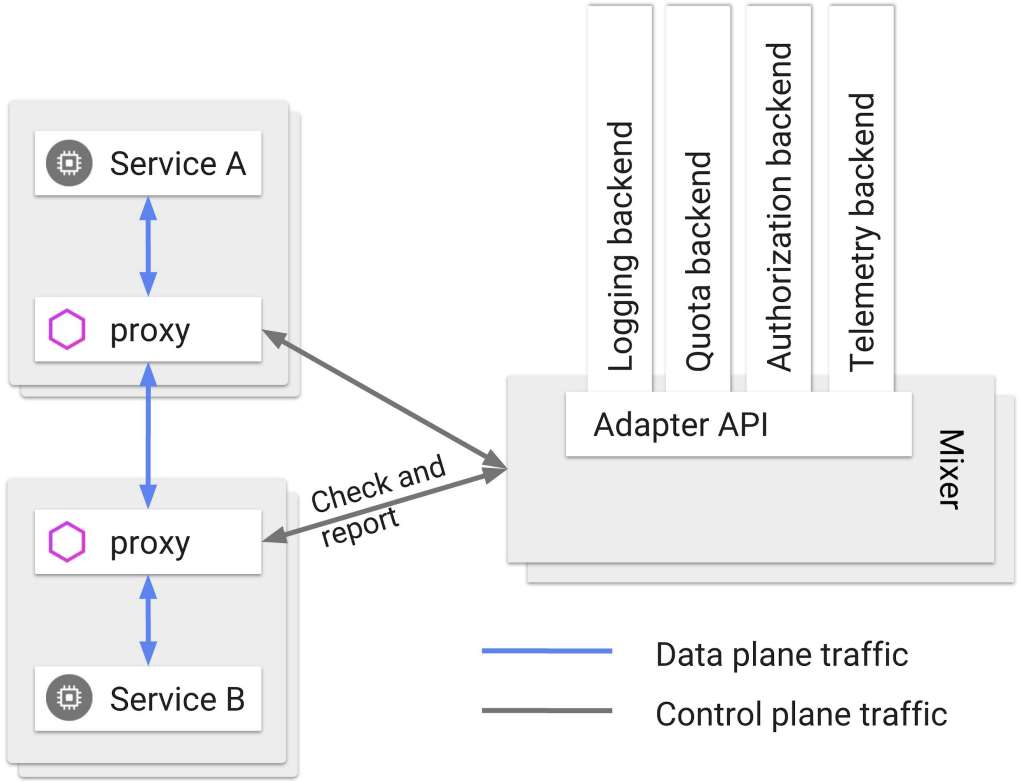
Google Cloud

What does Istio do?

Google Cloud

Observing services

Get automatic tracing, monitoring, and logging of all your services.



Google Cloud

Connecting services

Using `VirtualService`, `DestinationRule`, `Gateway`, and `ServiceEntry` objects, Istio helps you with:

Traffic splitting

Traffic steering

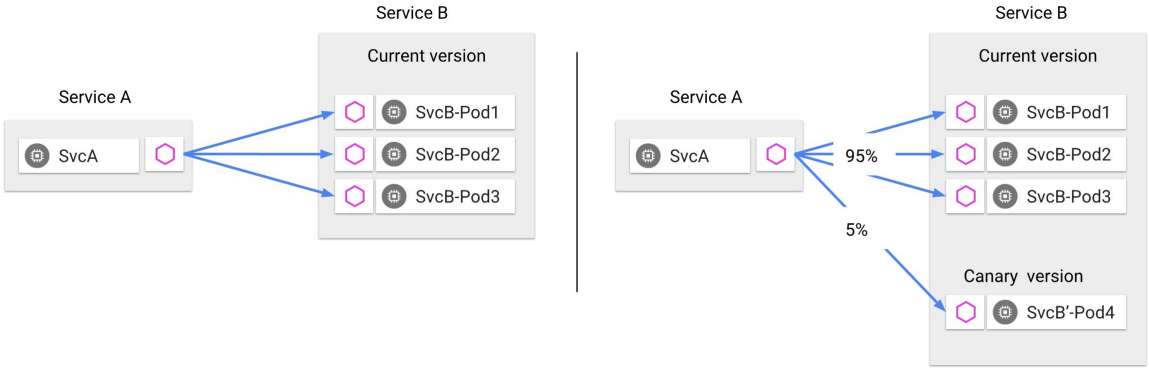
Fault injection

Circuit breaking

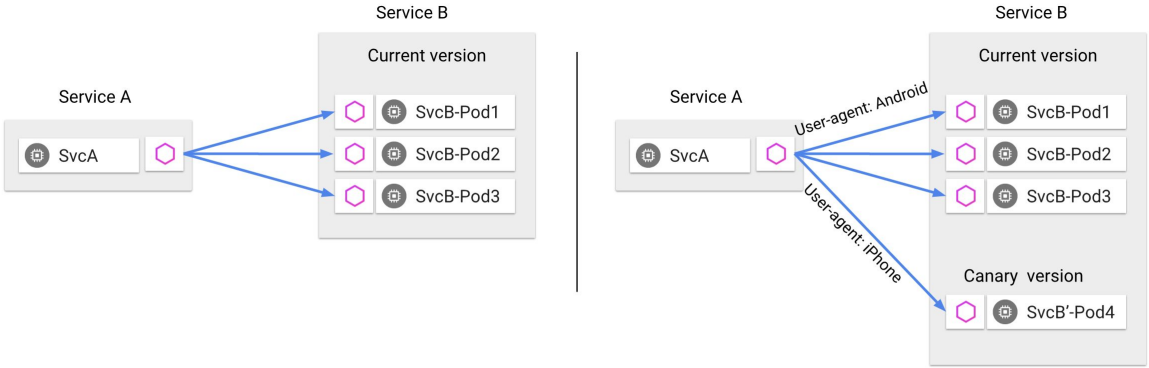
Egress control

Connecting services

Traffic splitting and traffic steering



Traffic splitting decoupled from infrastructure scaling - proportion of traffic routed to a version is independent of number of instances supporting the version

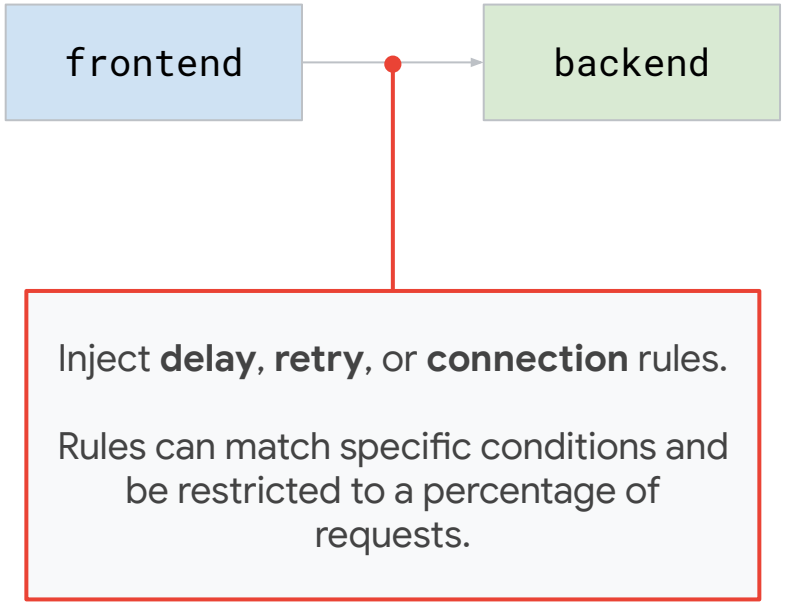


Content-based traffic steering - The content of a request can be used to determine the destination of a request

Google Cloud

Connecting services

Fault injection and circuit breaking



Securing services

Automatically secure your services through managed authentication, authorization, and encryption of communication between services.

Traffic encryption

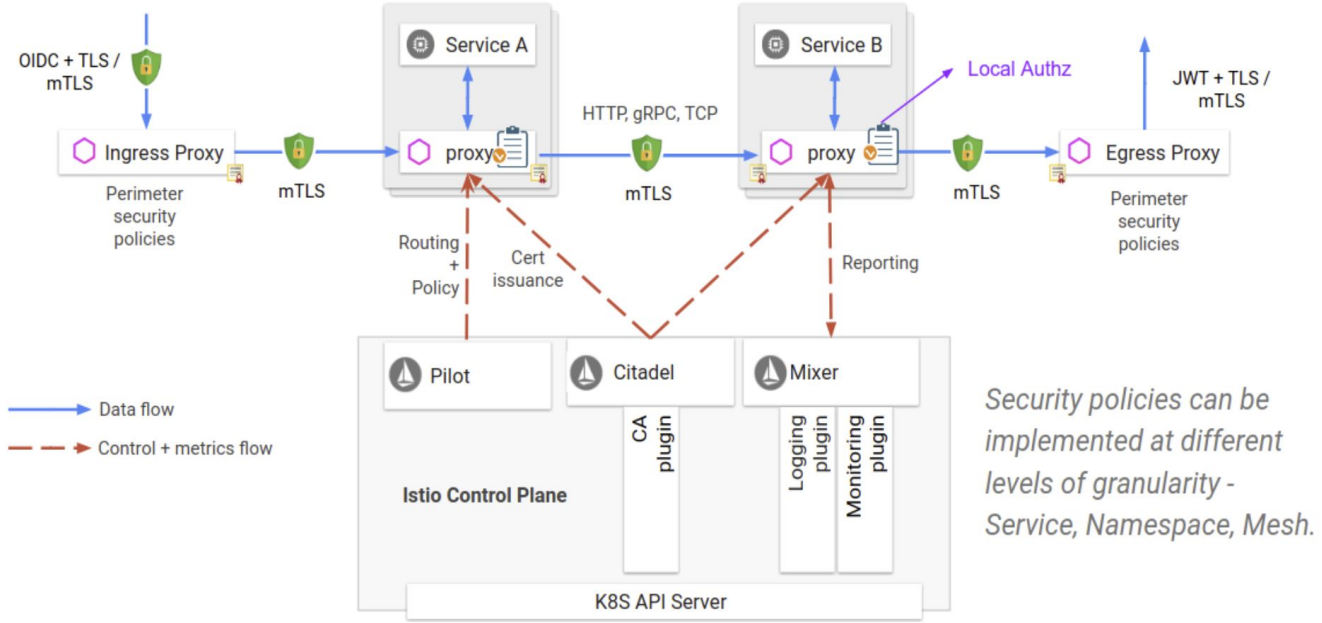
Service auth

Auditing controls

Access policies

Controlling and securing services

Apply broad or fine-grained security policies to some or all of your workloads



Istio Security Architecture

Security policies can be implemented at different levels of granularity - Service, Namespace, Mesh.

Google Cloud

How do you put Istio to work?

4

Google Cloud

Including the sidecar proxy in your Pods

```
# manually injecting the sidecar proxy
kubectl apply -f <(istioctl kube-inject -f deployment.yaml)
```

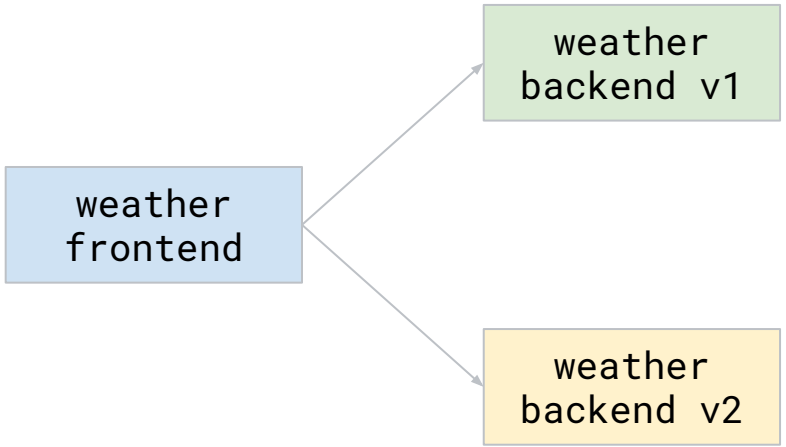
```
# auto-injecting the sidecar proxy
kubectl label ns default istio-injection=enabled
```

Traffic management

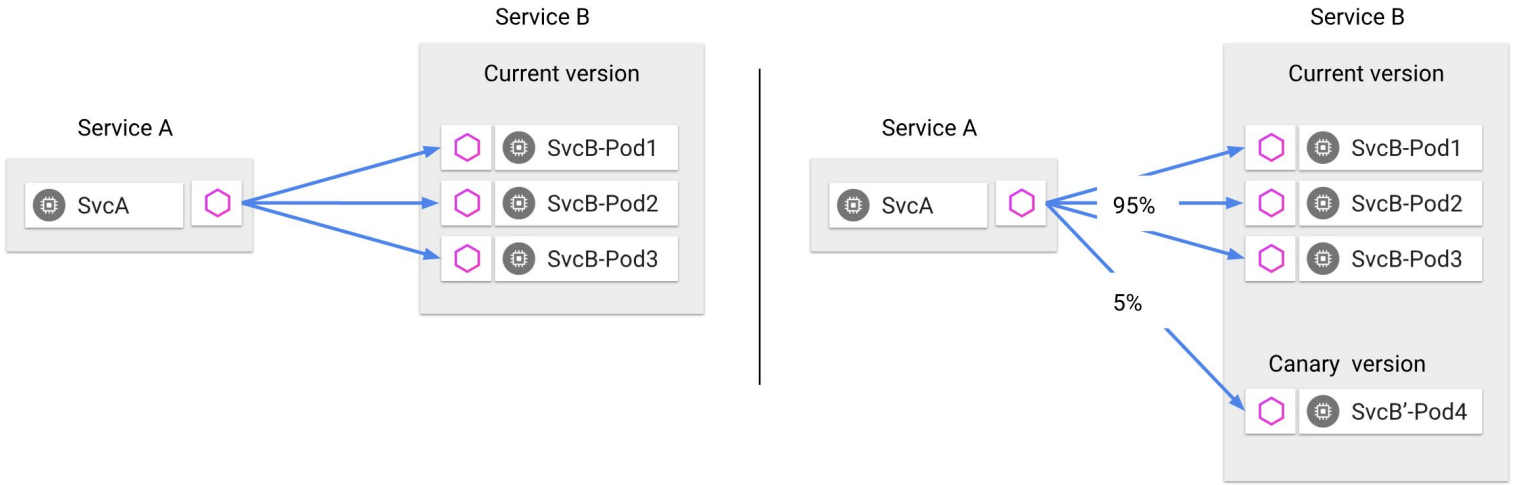
Google Cloud

Simple deployment

This is a simple `frontend`/`backend` deployment. We want to canary test `v2` of the backend. Each backend provides slightly different functionality.



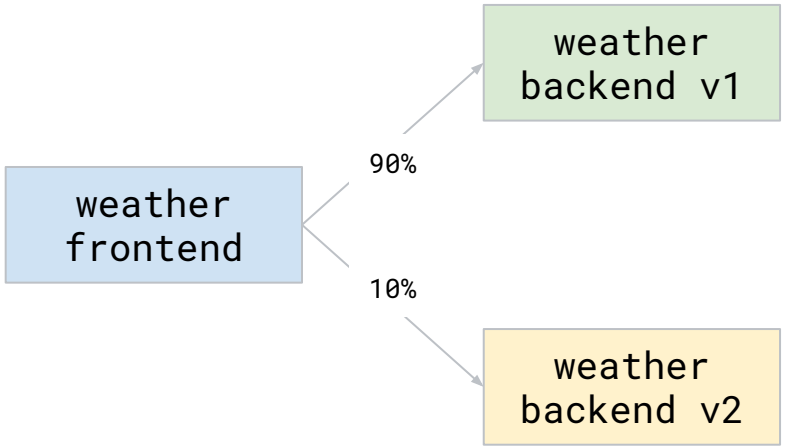
Recap: Traffic splitting



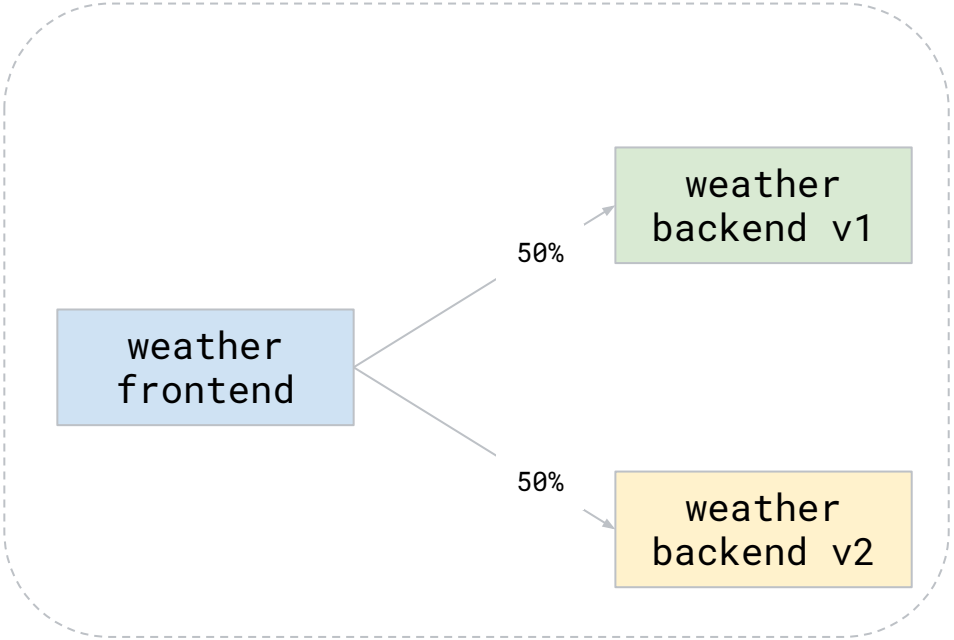
Traffic splitting decoupled from infrastructure scaling - proportion of traffic routed to a version is independent of number of instances supporting the version

Distributing traffic

Without sidecar proxies, how can we use Istio's traffic distribution? The `frontend` should hit `v1` 90% of the time, and `v2` 10% of the time.



Default Kubernetes round-robin routing



Traffic policy for backend deployment

Create a `DestinationRule` policy for `weather-backend` - in this case, creating named subsets to send traffic to

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: weather-backend-destination
spec:
  host: weather-backend
  subsets:
  - name: single
    labels:
      version: single
  - name: multiple
    labels:
      version: multiple
```

Routing traffic from ingress

Create routing rules using a `VirtualService`, telling the `Gateway` object where/how to distribute traffic

Google Cloud

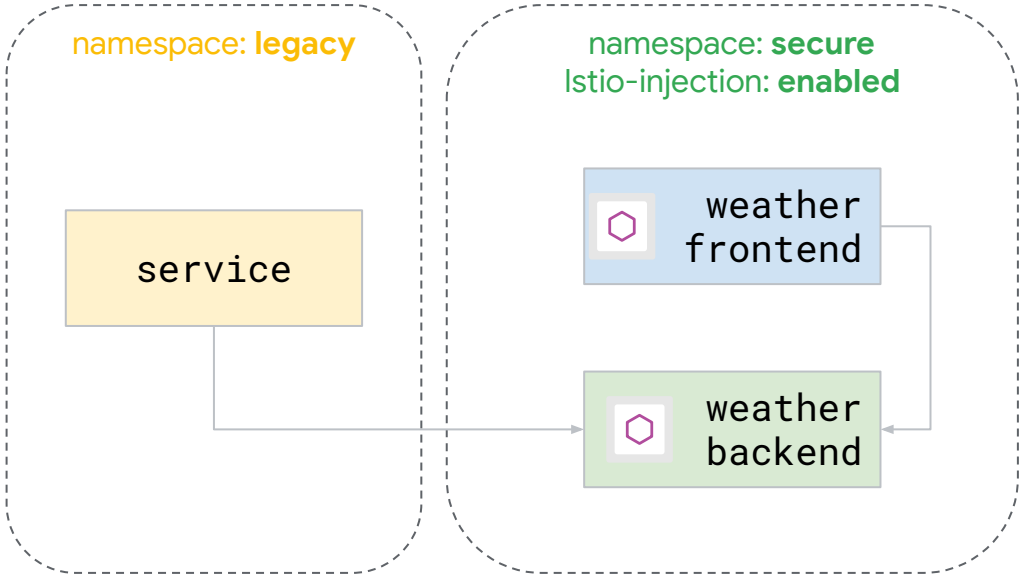
```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: weather-backend-service
spec:
  hosts:
  - "*"
  gateways:
  - weather-backend-gateway
  http:
  - match:
    - uri:
        exact: /api/weather
    route:
    - destination:
        host: weather-backend
        subset: single
        port:
          number: 5000
      weight: 90
    - destination:
        host: weather-backend
        subset: multiple
        port:
          number: 5000
      weight: 10
```

Securing services

Google Cloud

Securing a subset of services

How do you use Istio to slowly deploy mTLS across services, while also keeping legacy clients in mind?



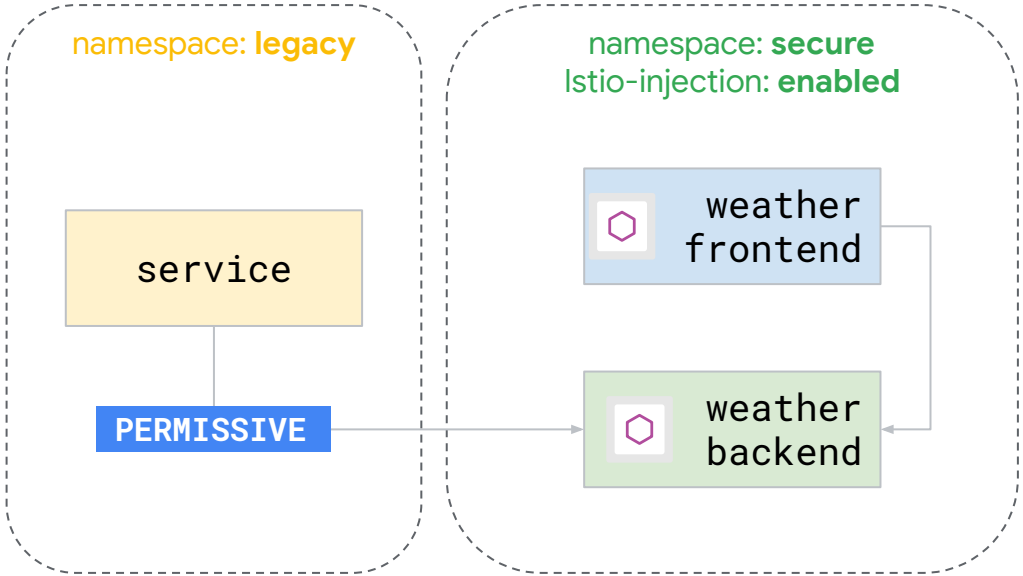
Google Cloud

Rolling out mTLS

1. Deploy Istio with `PERMISSIVE` mTLS settings
2. Apply `Policy` for Service A with `PERMISSIVE` mode
3. Apply `DestinationRule` for Service A with `MUTUAL` mode
    a. Services **with** `istio-proxy` encrypt traffic using mTLS
    b. Services **without** `istio-proxy` don't encrypt traffic
4. When ready, apply `Policy` for Service A with `STRICT` mode

Apply `Policy` with `PERMISSIVE` mode

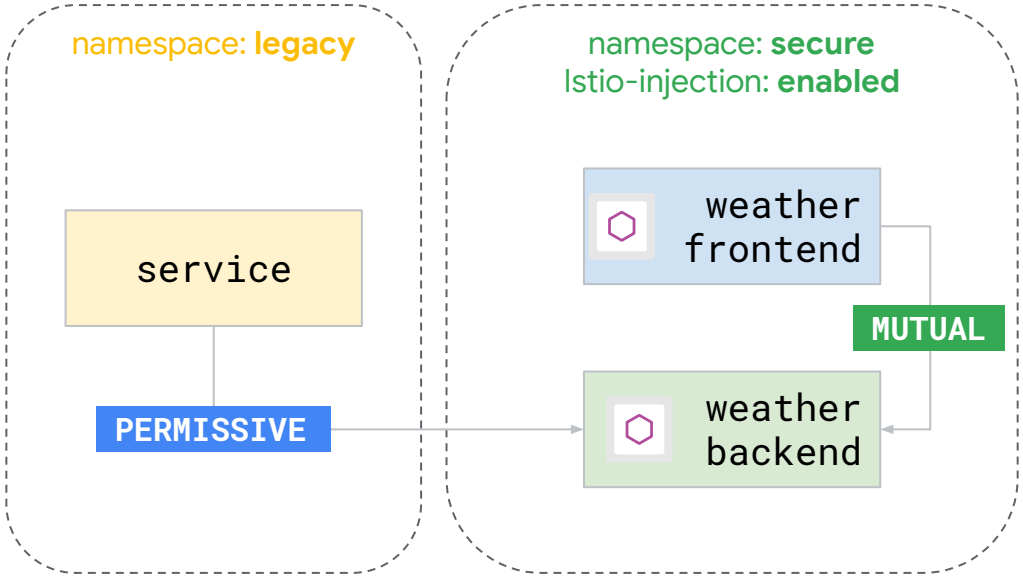
```yaml
apiVersion: auth.istio.io/v1alpha1
kind: Policy
metadata:
  name: mtls-backend
  namespace: secure
spec:
  targets:
  - name: weather-backend
  peers:
  - mtls:
      mode: PERMISSIVE
```



Google Cloud

Apply `DestinationRule` with `MUTUAL` mode

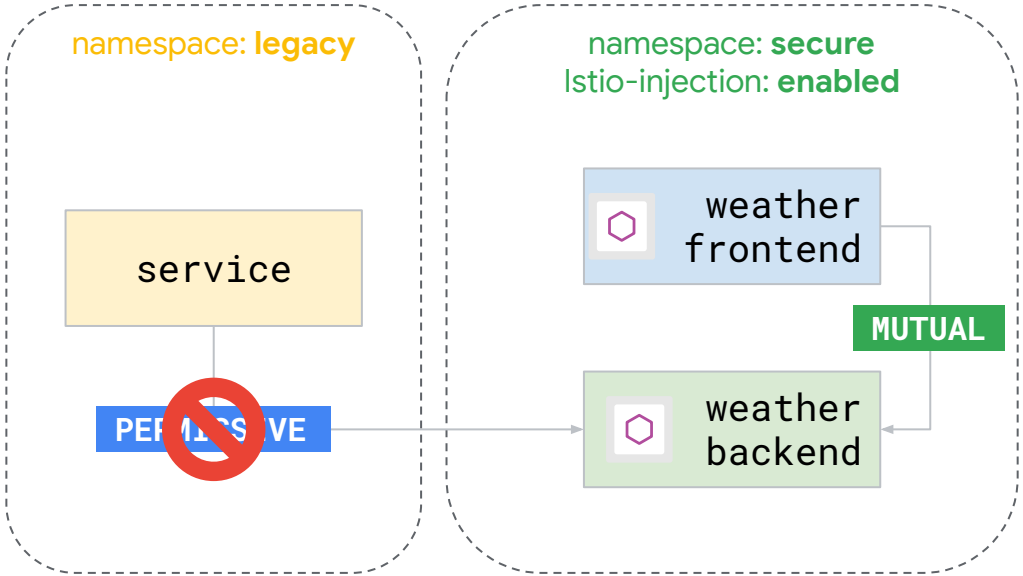
```yaml
apiVersion: net.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: mtls-mutual
spec:
  host: weather-backend.secure
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
```



Google Cloud

Apply `Policy` with `STRICT` mode

```yaml
apiVersion: auth.istio.io/v1alpha1
kind: Policy
metadata:
  name: mtls-backend
  namespace: secure
spec:
  targets:
  - name: weather-backend
  peers:
  - mtls:
      mode: STRICT
```



Google Cloud

Gathering telemetry

Google Cloud

Telemetry for free

Without any instrumentation, Istio captures a predefined set of **metrics**, request **traces**, and service **logs** - and forwards them to the configured adapters.

Istio built-in metrics

`Request Count`
Incremented for every request handled

`Request Duration`
Duration of the request

`Request Size`
Size of the HTTP request body

`Response Size`
Size of the HTTP response body

`TCP Byte Sent`
Total bytes sent during response

`TCP Byte Received`
Total bytes received during request

Where do you go from here ?

Google Cloud

Some light reading to get you started

Service Mesh Era blog posts (conceptual and practical intros)

Incremental Istio: Traffic Management

mTLS Migration

Istio Concepts

github.com/crscmnky/weatherinfo

github.com/crcsmnky/next19-incremental-istio

Thank you!