ORDINA

Service Meshes with Istio

Agenda

Introduction

Istio Overview

Hands-on: Istio Setup

Hands-on: Demo Application

But first...

https://github.com/tomverelst/istio-workshop

- Install kubectl
- Download Istio 1.0.0
- Setup minikube v0.28+

Installation details in README.md

Distributed world

"The trends of containerization, microservices and hybrid/multi-cloud deployments have created more distributed applications than ever.

Developers, devops and secops need modern tools to secure, manage and monitor distributed applications."

Dan Ciruli - Istio Product Manager

"Everybody got all fired up about Kubernetes and microservices and then were like ‘Oh, s--t, what’s going on?’

Some early adopter

Popular Open Source Tools

Eureka Service discovery

Ribbon Client Side LB

Hystrix Circuit Breaker

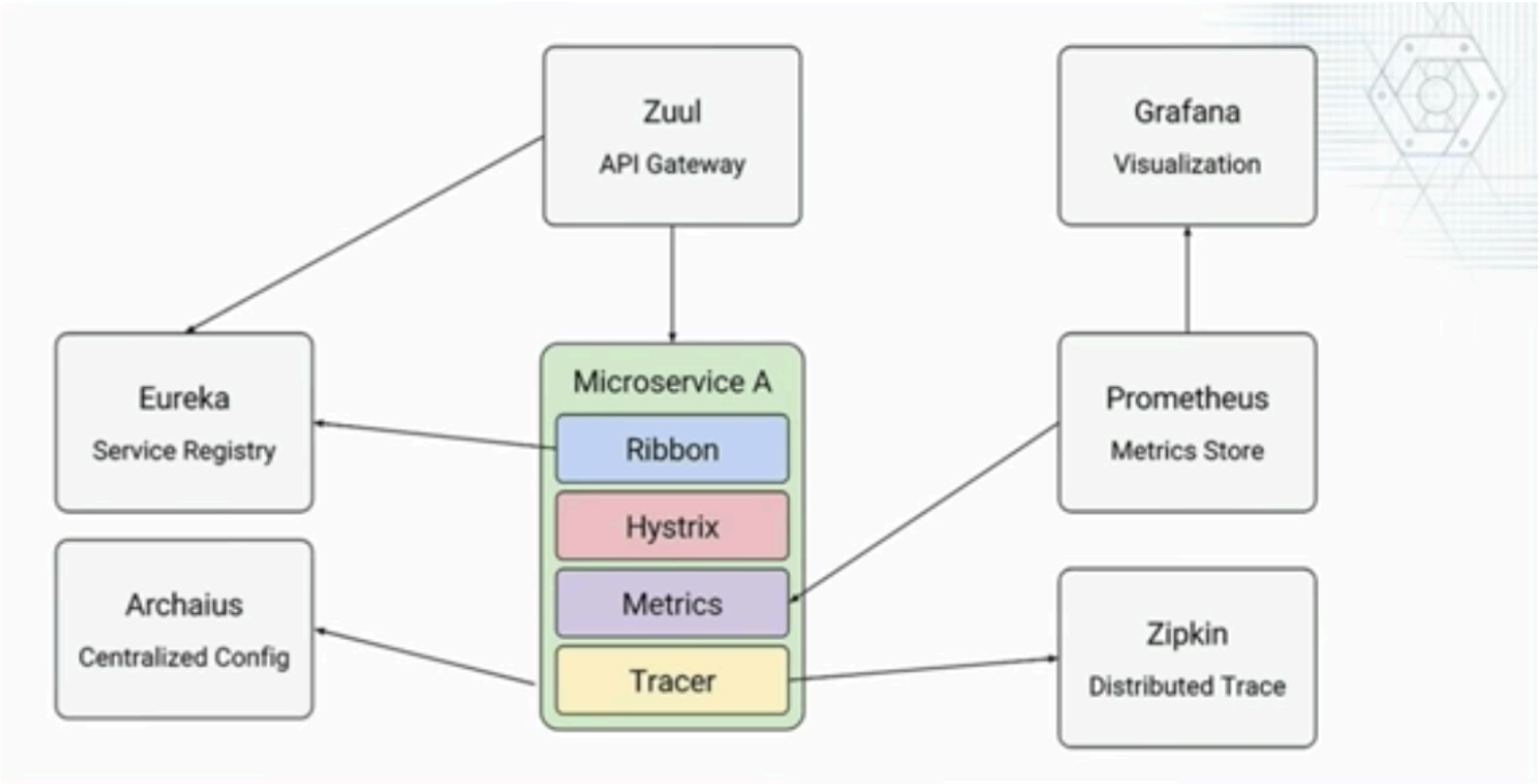
Zipkin Distributed Tracing

Prometheus Monitoring

Grafana Data Visualisation

Chaos Monkey Chaos Engineering

Zuul API Gateway


Zuul
API Gateway
Grafana
Visualization
Eureka
Service Registry
Microservice A
Ribbon
Hystrix
Metrics
Tracer
Prometheus
Metrics Store
Archaius
Centralized Config
Zipkin
Distributed Trace

Easy when...

Spring Boot

Single stack

Multiple stacks
Multiple frameworks
Polyglot
Legacy applications

Let Microservice A talk to Microservice B

Should be as simple as making an HTTP call
Platform should take care of non-functional requirements

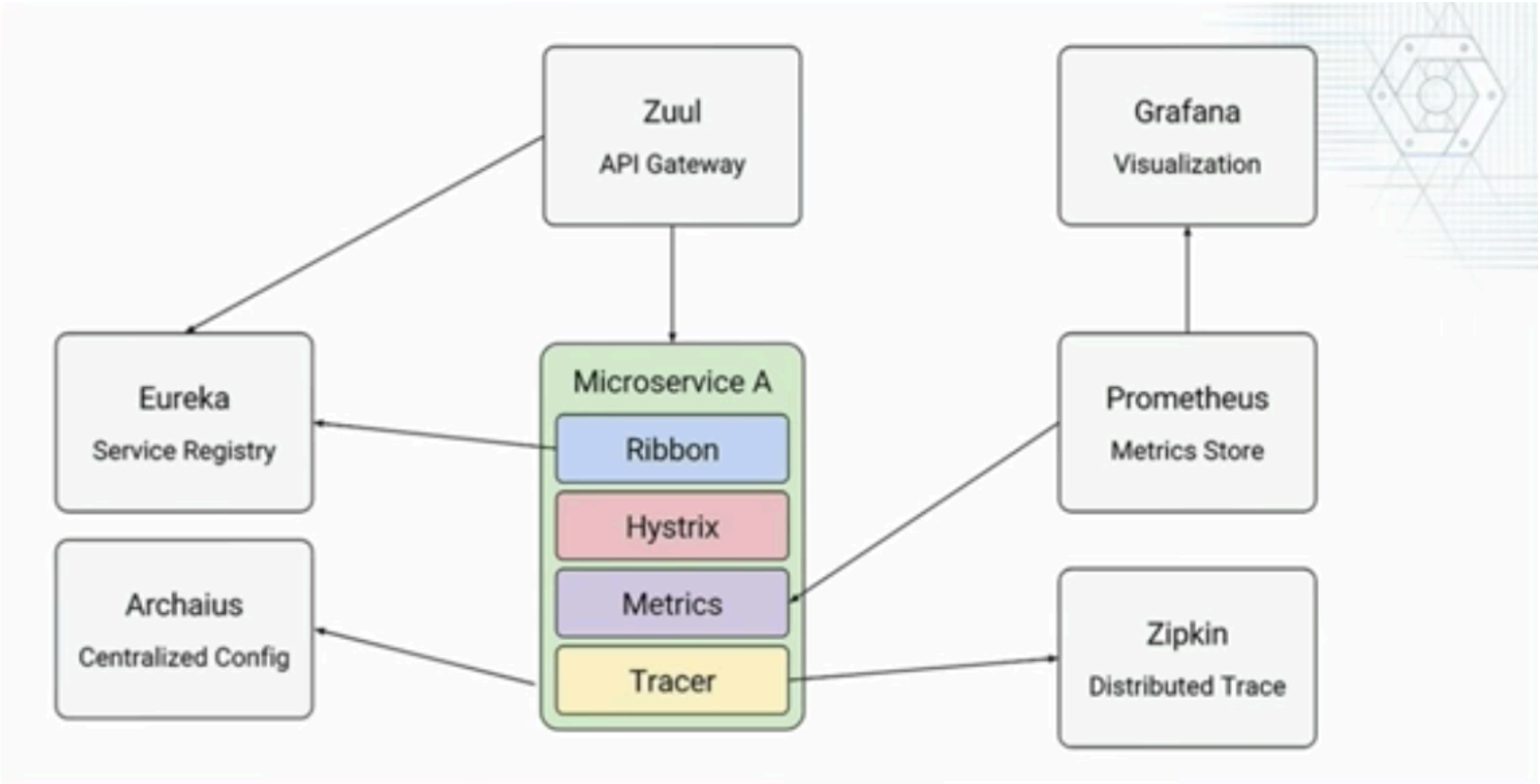
What is a service mesh?

A service mesh provides a **transparent** and **language-independent** way to flexibly and easily **automate** application network functions: control, configure, monitor application-level requests, ensure resilience, routing, observability, fault-injection and more.

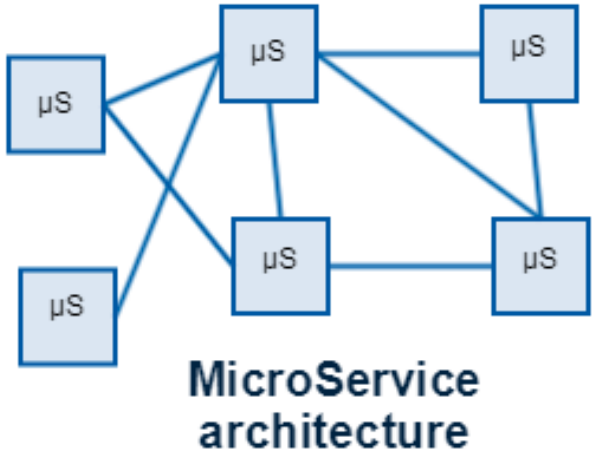
Separate **infrastructure** from **application**

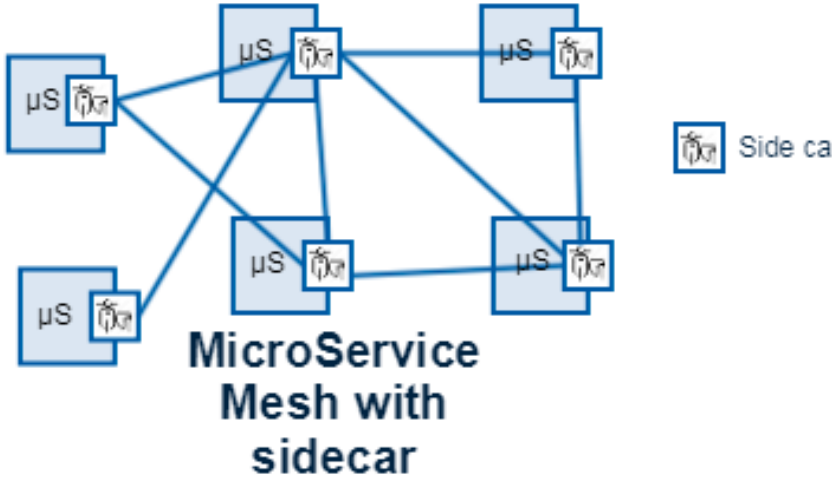
"Making microservices micro"

Ray Tsang - Google Cloud Developer Advocate


Zuul
API Gateway
Grafana
Visualization
Eureka
Service Registry
Microservice A
Ribbon
Hystrix
Metrics
Tracer
Prometheus
Metrics Store
Archaius
Centralized Config
Zipkin
Distributed Trace

- Deployed as a sidecar in a Kubernetes pod





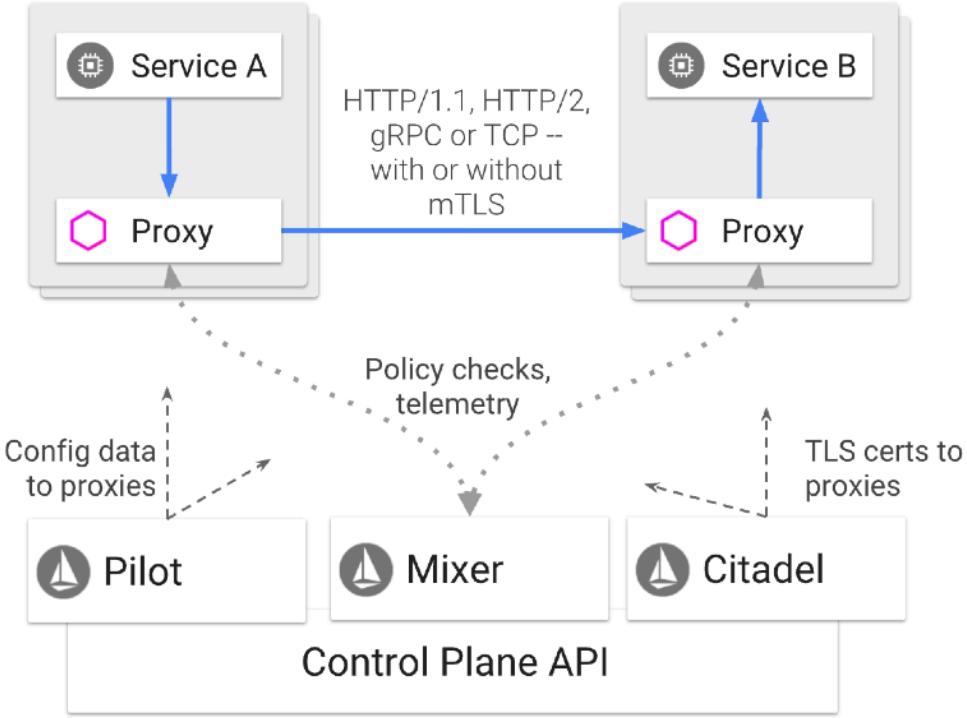
Istio Overview

Connect, secure and monitor services in hybrid environments

ORDINA ICT voor mensen

Istio: An open services platform to manage service interactions across container and VM-based workloads

Architecture



- **Pilot**: Control plane to configure and push service communication policies
- **Envoy**: Network proxy to intercept communication and apply policies
- **Mixer**: Policy enforcement with a flexible plugin model for providers for a policy
- **Citadel**: Service-to-service auth[n,z] using mutual TLS, with built in identity and credential management

Envoy: High performance proxy

- **C++ L4/L& Proxy**
  - 2M requests/s
- **API driven config updates**:
  - No restarts required
- Zone-aware load balancing with failover
- Supports HTTP/2 & gRPC
- Health checks, circuit breakers, retry budgets, timeouts, fault injection, …

Pilot: Configuring the data plane

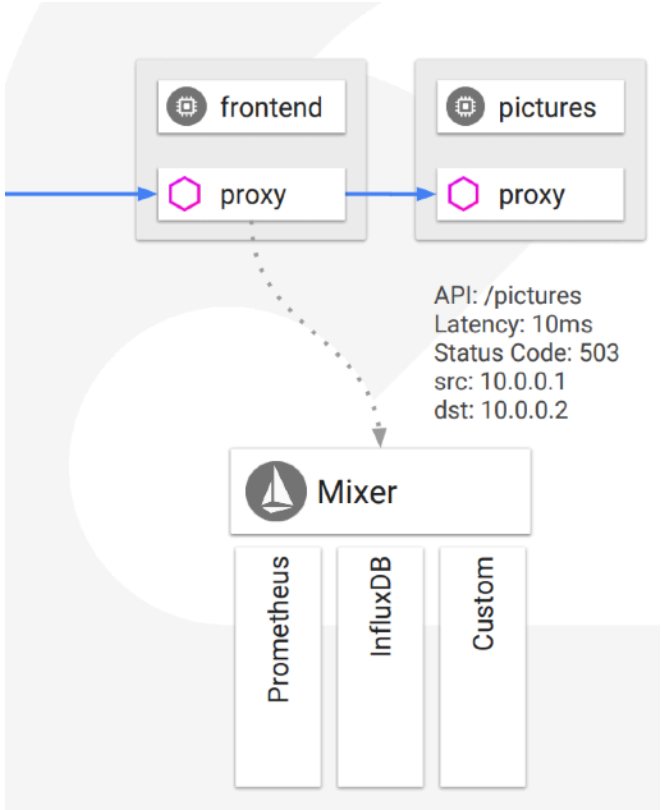
- **Observes service topology**:
  - Services, pods and ingress rules

- **Routing rules**:
  - Reads high-level rules and converts them to Envoy config

- **Configures Envoy side-cars**
  - Intelligent traffic management (A/B, canary testing, ...)
  - Resiliency (circuit breakers, timeouts, retries)
  - Service discovery

Mixer: Operator control plane

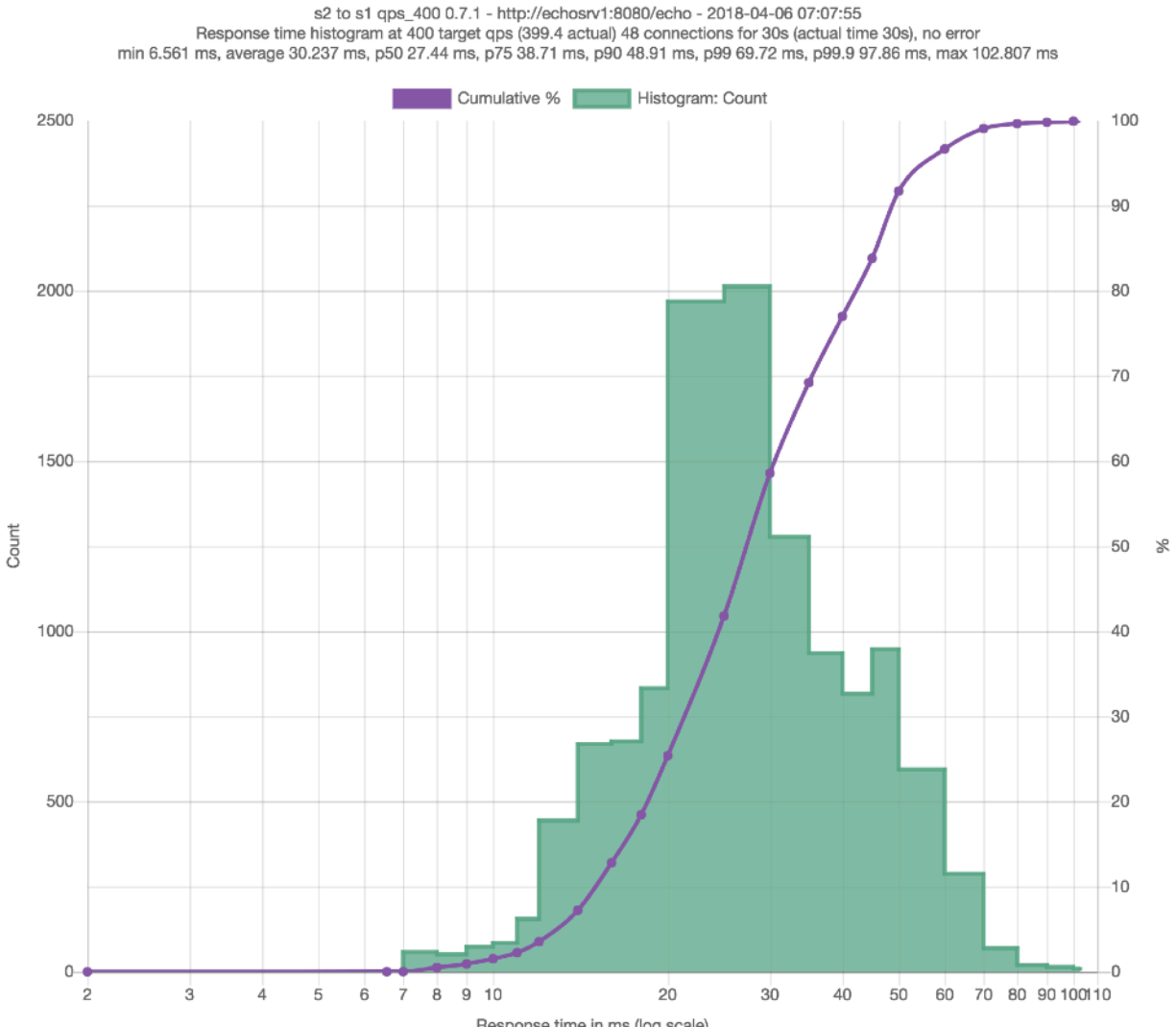
- Called by Envoy to gate traffic and report telemetry
  - Response caching in Envoy for performance
- Pluggable components
  - Monitoring
  - Logging
  - Policy



Performance

Service to service

No cache

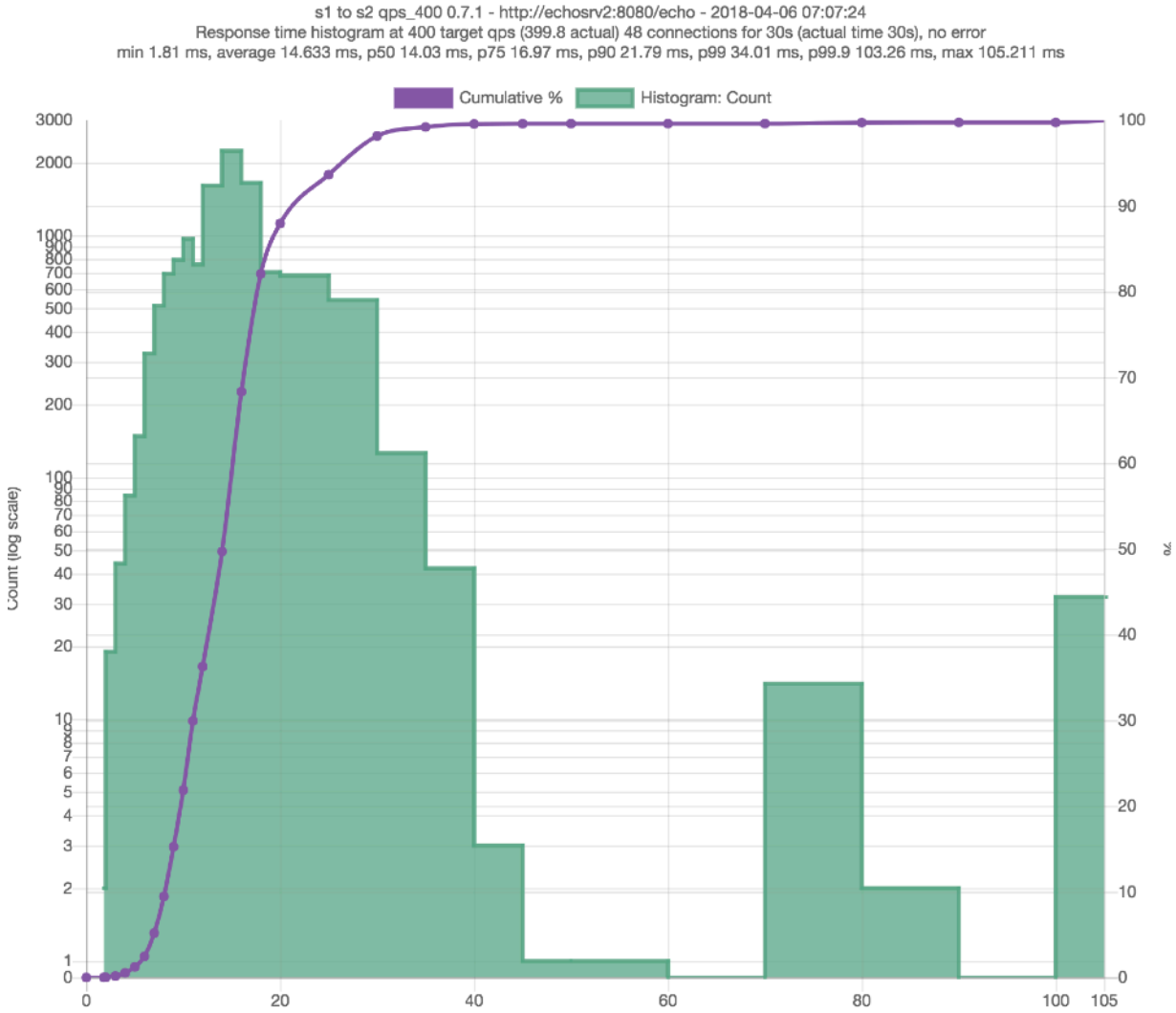


ORDINA ICT voor mensen

Performance

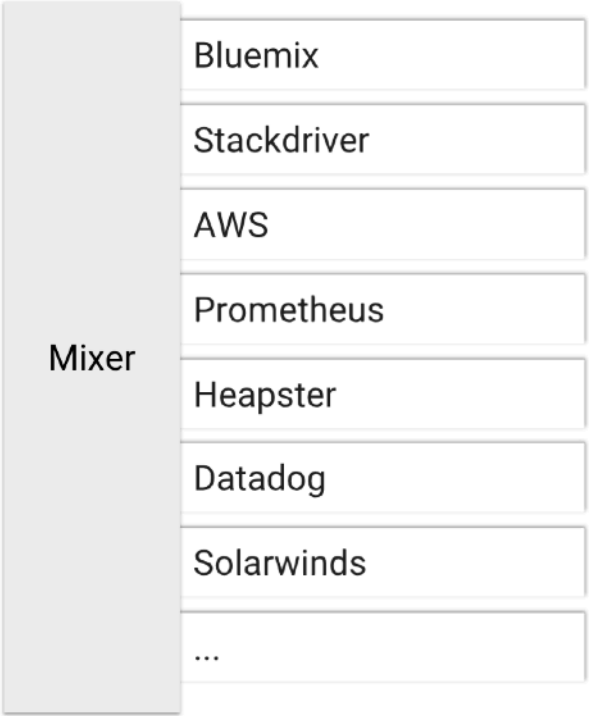
Service to service

Cached policies



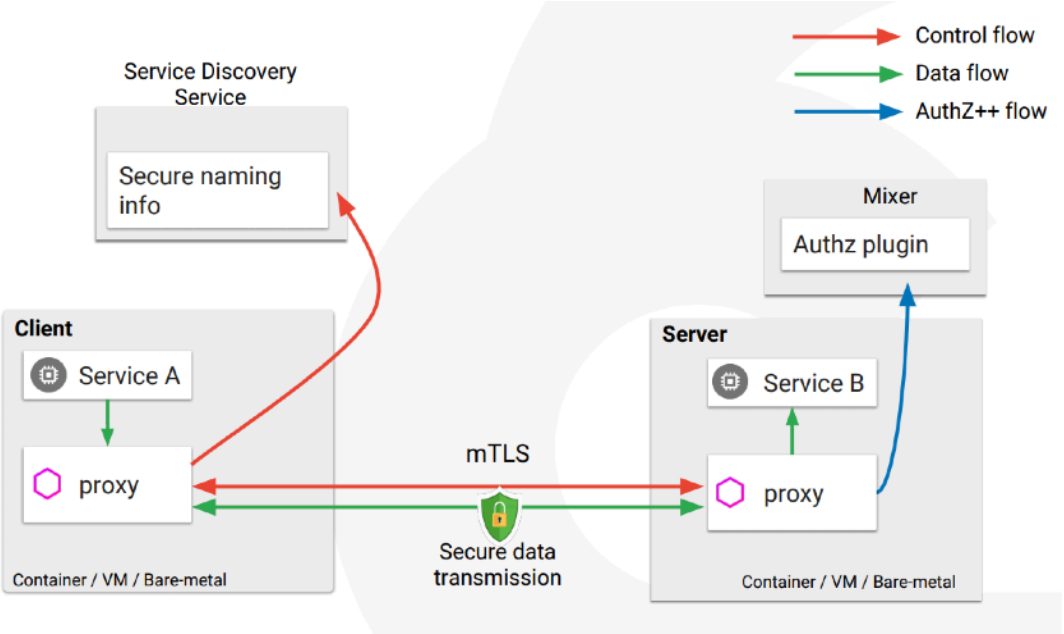
ORDINA ICT voor mensen

Mixer Adapters

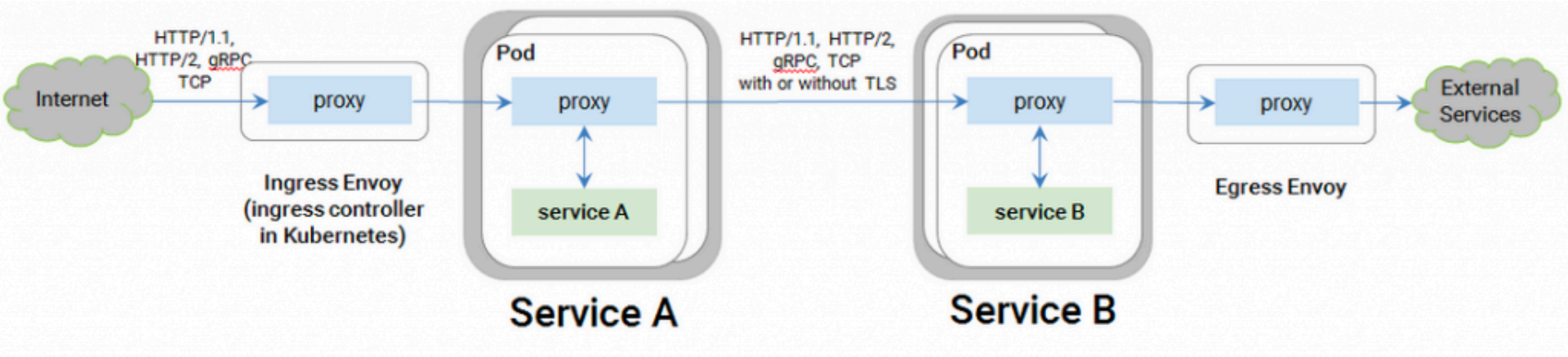


Citadel: Certificate management and more

- Certificate management
- Service-level Authorization
- Security analytics, ACL, …



Istio Request Flow



High-level features

Traffic Control
Observability
Tracing
Resiliency
Security
Policy Enforcement

Zero code changes
(excl. forwarding tracing headers)

WENGER

Let’s try it

If you have not done it yet…

https://github.com/tomverelst/istio-workshop

- Install kubectl
- Download Istio 1.0.0 (optionally)
- Setup minikube v0.28+

Installation details in README.md

Setup

- Setup Kubernetes cluster (minikube)
- Install the Istio Custom Resource Definitions (CRD)
- Install Istio components
- Enable automatic sidecar injection

Setup minikube

Start minikube with some options and ingress enabled

```
$ minikube addons enable ingress
$ minikube start --memory=8192 --cpus=4 \
  —kubernetes-version=v1.10.0 \
    --extra-config=controller-manager.cluster-signing-cert-
file="/var/lib/localkube/certs/ca.crt" \
    --extra-config=controller-manager.cluster-signing-key-
file="/var/lib/localkube/certs/ca.key" \
    --vm-driver=`your_vm_driver_choice`
```

Install Custom Resource Definitions

```
$ kubectl apply -f istio/install/kubernetes/helm/
istio/templates/crds.yaml
```

Install Istio

```
$ kubectl apply -f istio/install/kubernetes/istio-demo-auth.yaml
```

This will install all Istio components. Could take a while!

Adding a service to the mesh

Istio config must be applied to Kubernetes resources

Manual injection using istioctl

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

Automatic injection

- Requires the Istio sidecar injector to be running
- Namespace must be labeled

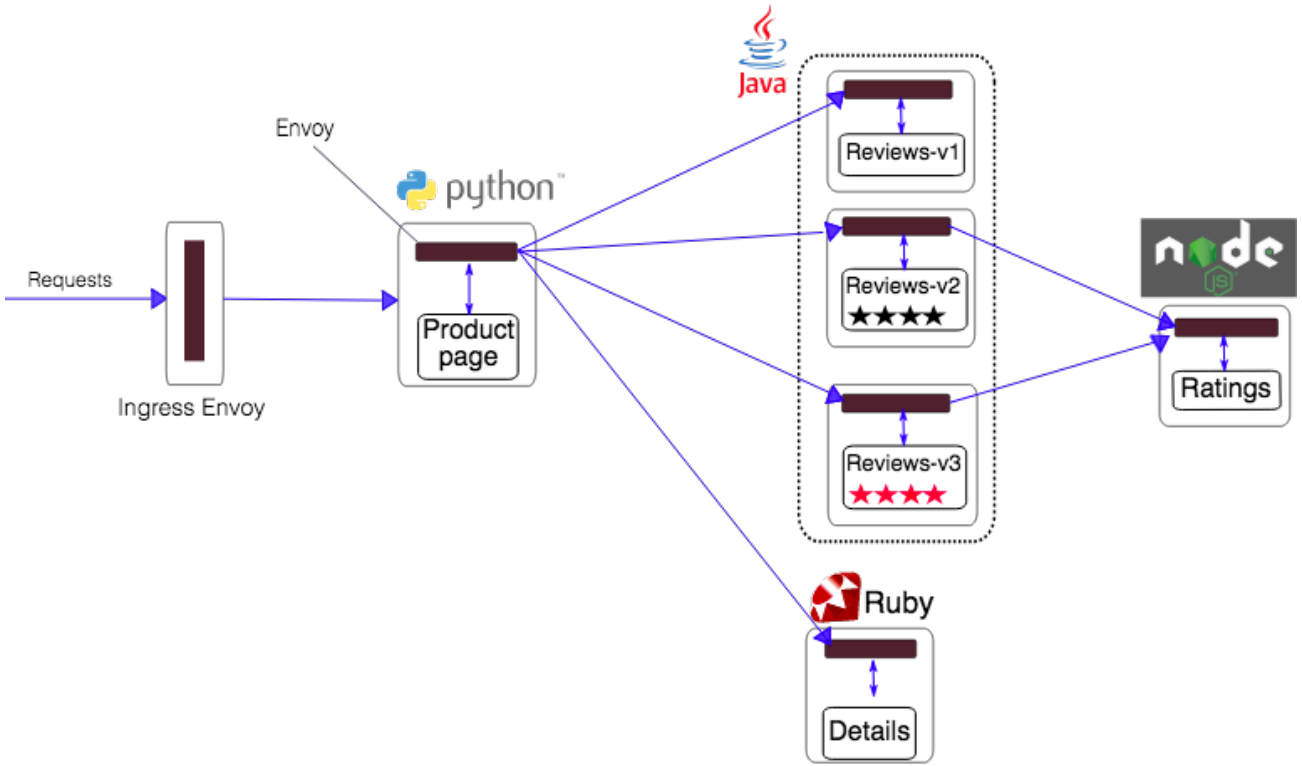
```
$ kubectl label namespace default istio-injection=enabled
```

Enable automatic injection

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

This will cause the istio-sidecar-injector admission controller to automatically inject sidecars

Bookinfo sample application



Bookinfo sample application

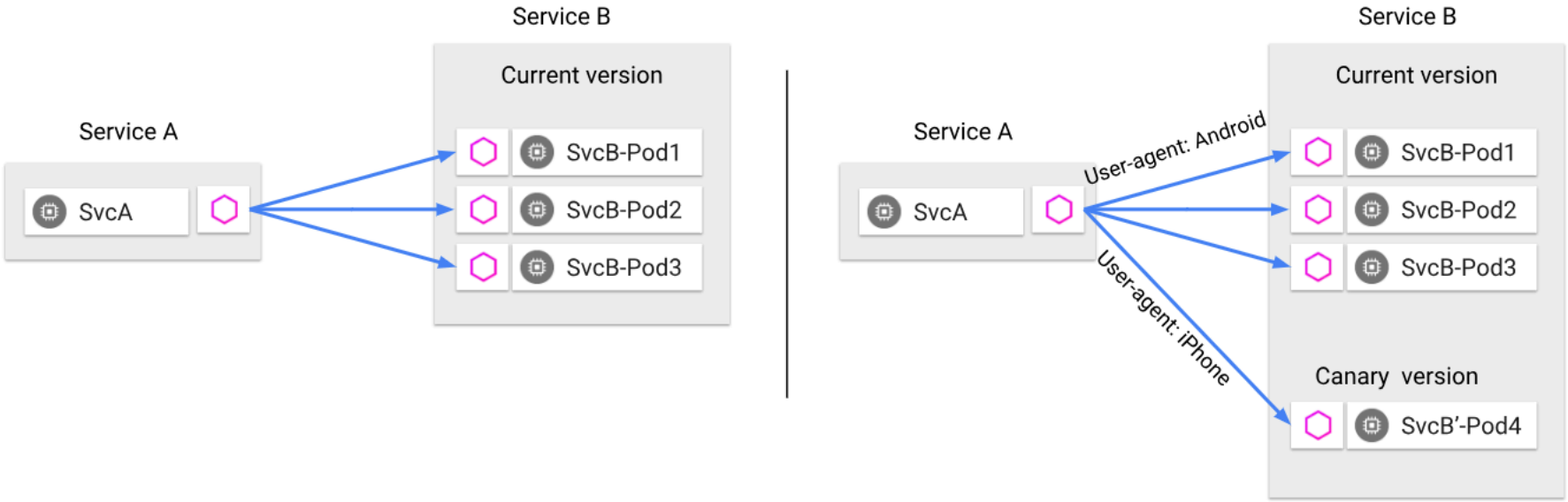
- Set up application
- Tracing & Monitoring
- Request routing (content-based)
- Fault injection: delays and aborts
- Traffic shifting (%-based)

Traffic Management

There are four traffic management configuration resources in Istio: **VirtualService**, **DestinationRule**, **ServiceEntry**, and **Gateway**:

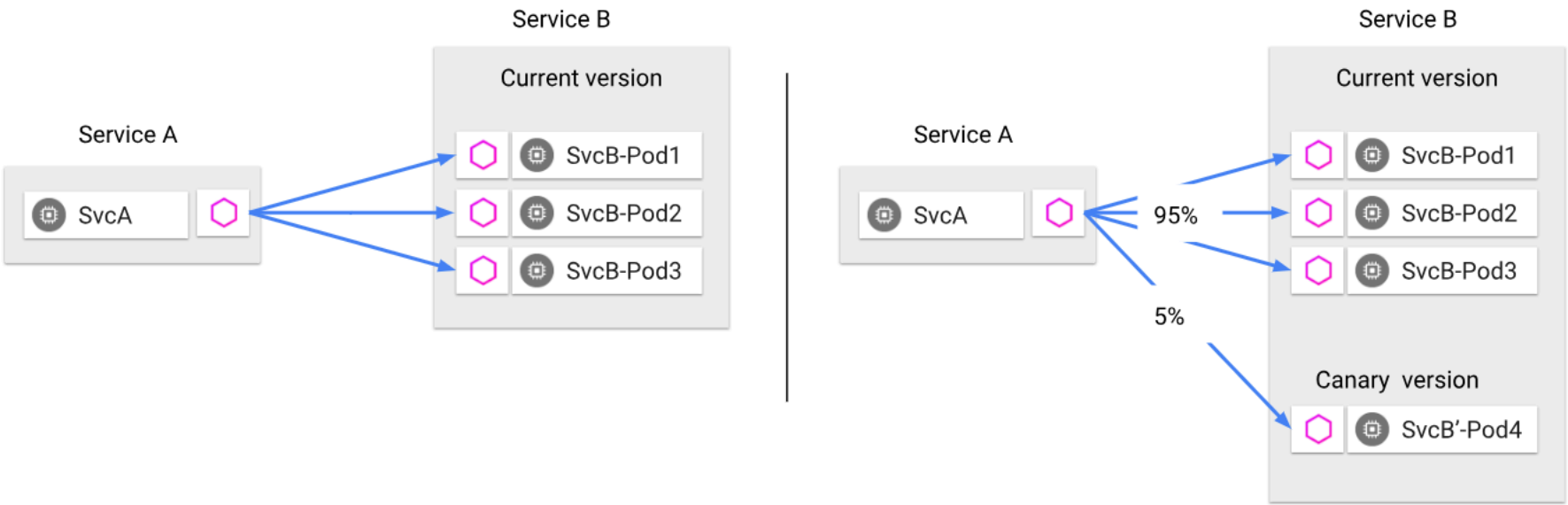
- A **VirtualService** defines the rules that control how requests for a service are routed within an Istio service mesh.
- A **DestinationRule** configures the set of policies to be applied to a request after VirtualService routing has occurred.
- A **ServiceEntry** is commonly used to enable requests 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.

Content-based steering



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

Traffic % splitting



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

Traffic Management

- Define available **subsets** (versions) first in **destination rules**

- Define how requests flow using a **virtual service**

More features

- **Traffic Management**
  - Setting request timeouts
  - Control Ingress Traffic
  - Control Egress Traffic
  - Circuit Breaking
  - Mirroring Traffic
  - Health Checking

More features

- **Policies**
  - Rate Limiting
  - Denials
  - White/Black Listing
- **Telemetry**
  - Collect metrics and logs
- **Multi-cluster service meshes**
- **...**

Conclusion

- Service meshes offer a lot of solutions out of the box, but you do not have to use everything

- Shifts operational complexity from the application-level to the infrastructure level. Makes microservices micro!

Thanks!