Running Istio in TVLK Data Production

Imre Nagi
@imrenagi
Software Engineer @TVLK DATA

About Me?

Imre Nagi

@imrenagi

- Software Engineer @traveloka data
- ex - Software Engineer Intern @eBay & @CERN
- Indonesia Docker Community Leader
- One of Jakarta Kubernetes Organizer team member

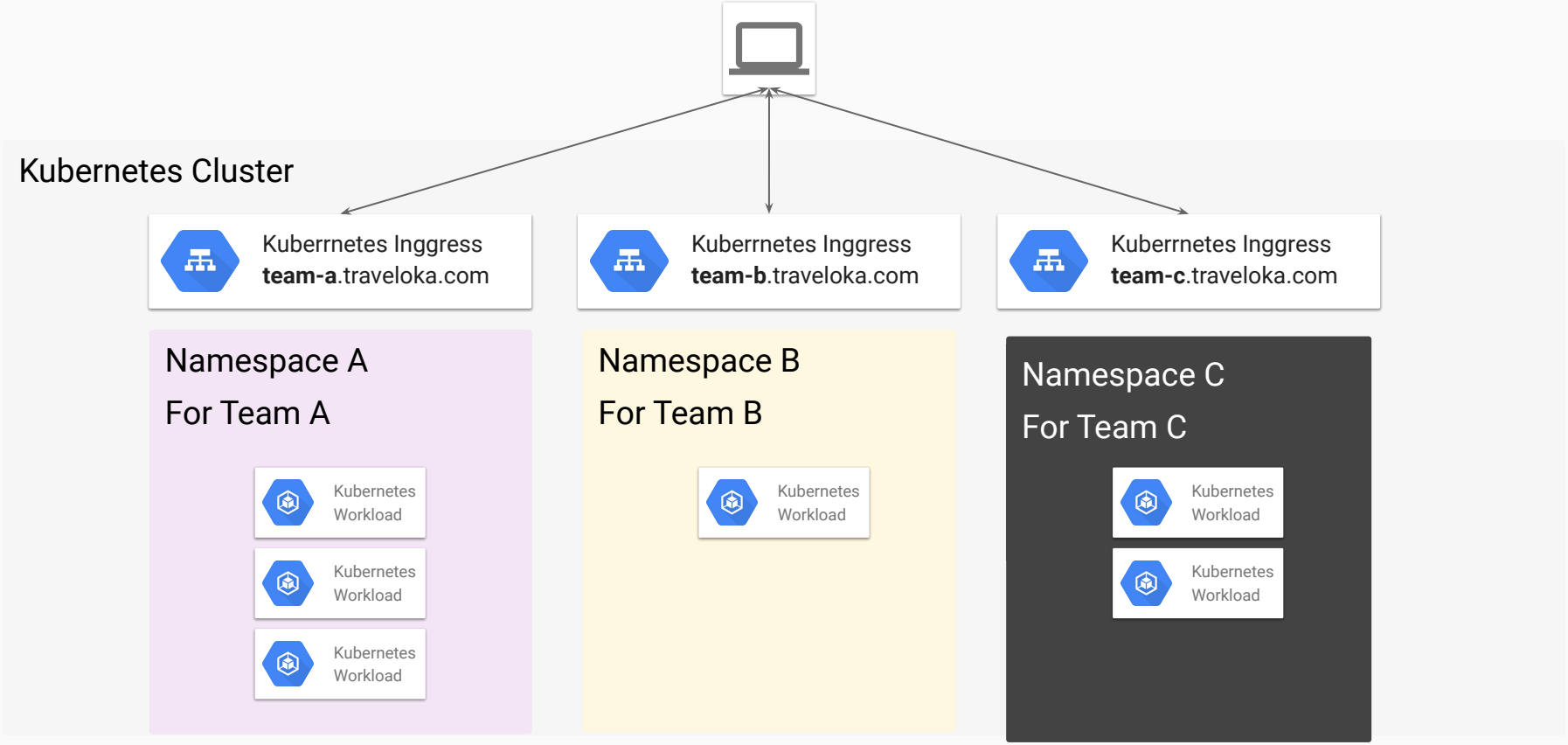
Agenda

- Kubernetes In Traveloka
- Istio
- Running Istio @traveloka
- Lesson learned

Kubernetes in TVLK

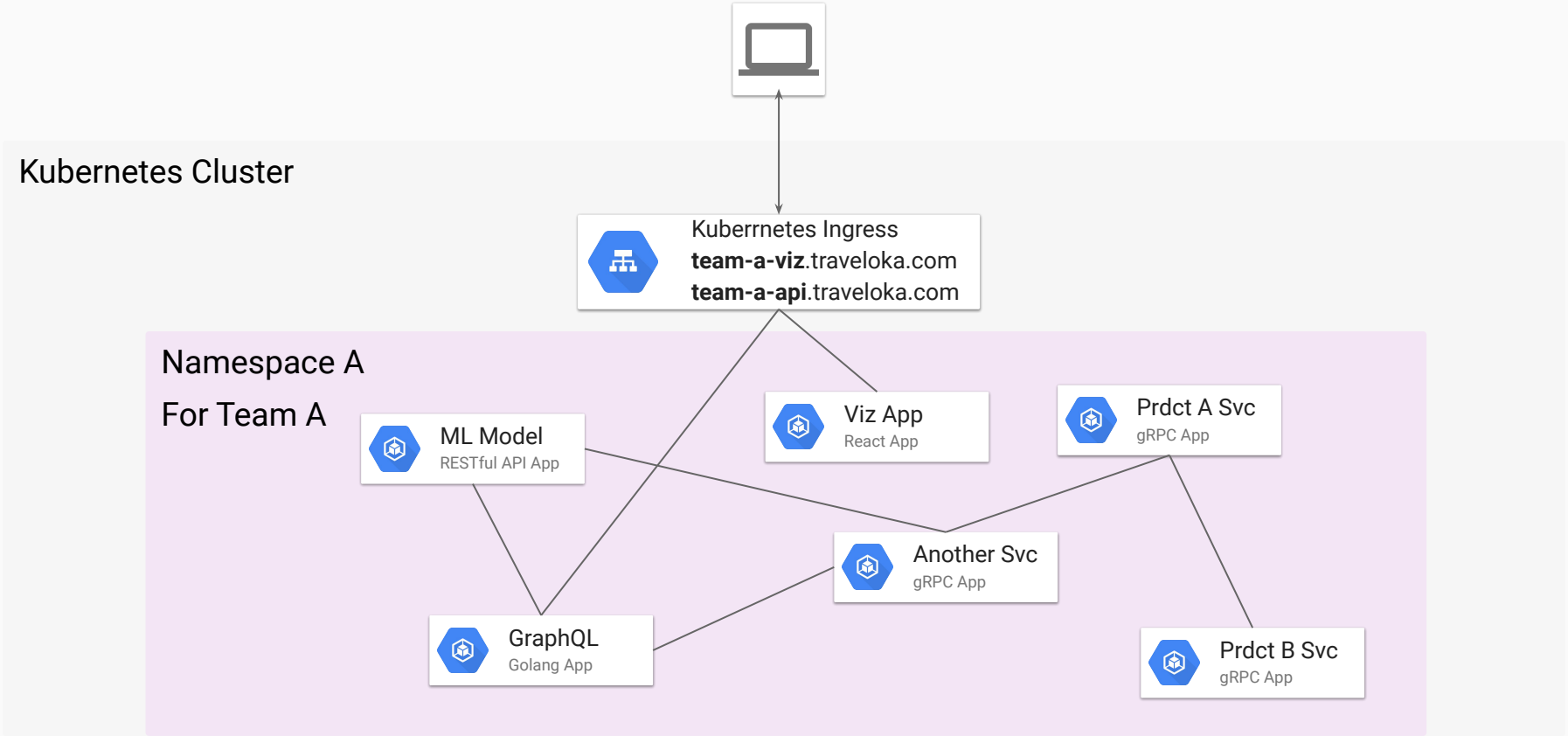
Kubernetes, Long story short

- About 2 years in Data Team Production
- Multiple teams in TVLK Data are using Kubernetes to run multiple type of loads.
  - API,
  - Visualization Dashboard,
  - GPU and Machine Learning
- At least, handles tens of thousands of request per second
- Cost Optimization (up to 60%)
  - Node and pod autoscaling
  - Better CPU utilizations

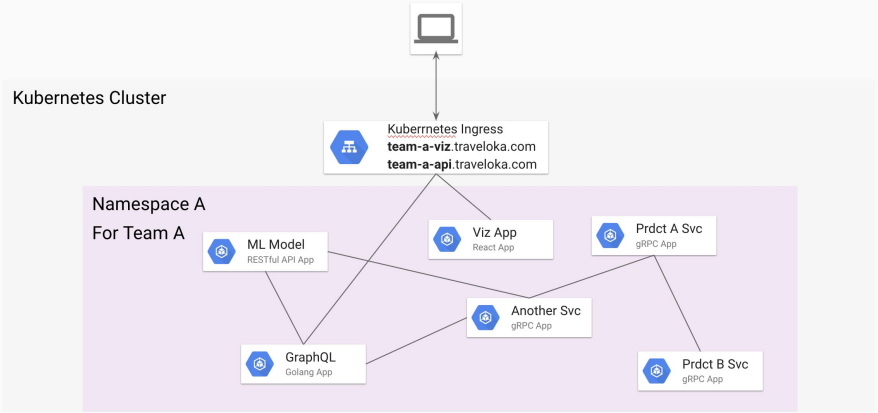


* Similar configuration between prod & staging cluster

“Everything is just working fine until one team decided to start decoupling their applications into multiple smaller services.”


Kubernetes Cluster
Kuberrnetes Ingress
team-a-viz.traveloka.com
team-a-api.traveloka.com
Namespace A
For Team A
ML Model
RESTful API App
Viz App
React App
Prdct A Svc
gRPC App
Another Svc
gRPC App
GraphQL
Golang App
Prdct B Svc
gRPC App

Managing Service Mesh is Freakin' Complicated



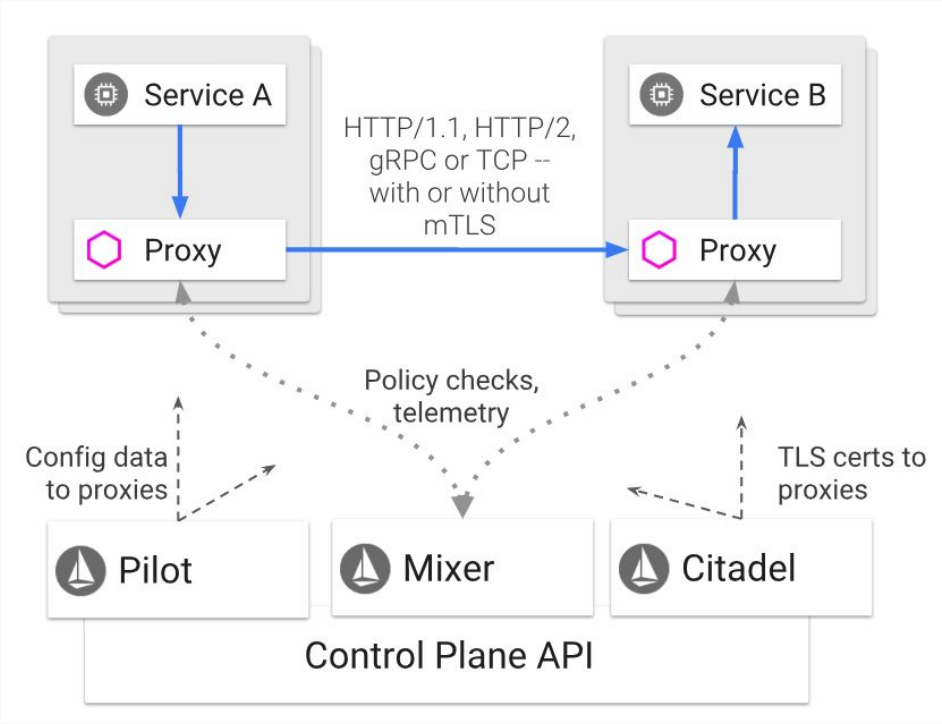
- Securing inter-service communication
- Unstandardized monitoring approach
  - Different monitoring tools used by different programming lang
  - Need better way to monitor RPC/API call between services (RED metrics, and Distributed Tracing)
- gRPC Load balancing doesn't work out of the box..

Istio to simplify service mesh in microservice

Why istio?

Istio features

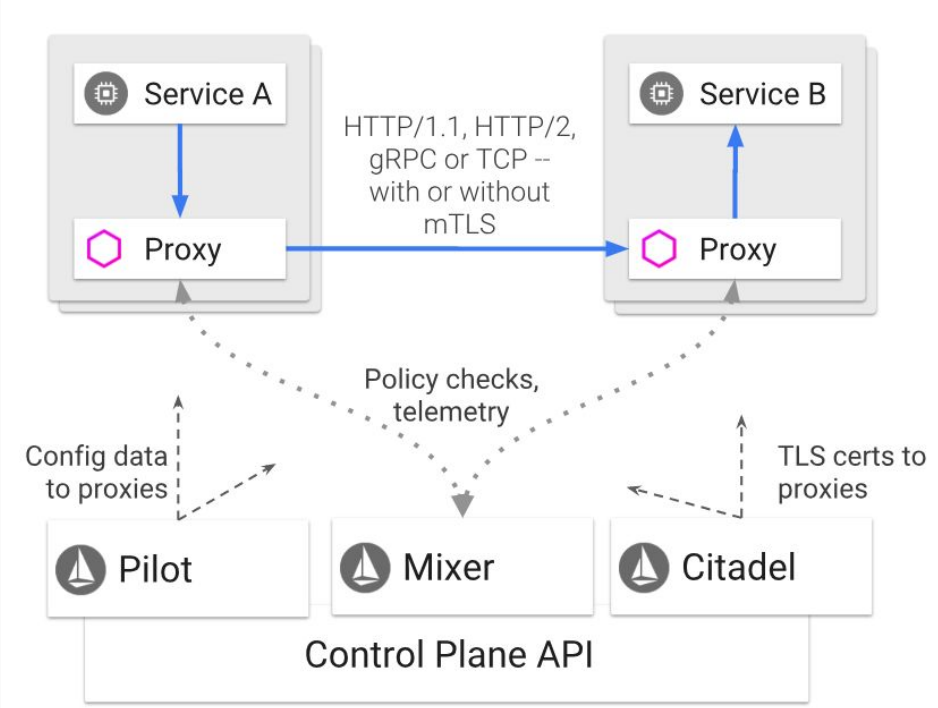
- Traffic Management
- Security
- Monitoring



Istio architecture

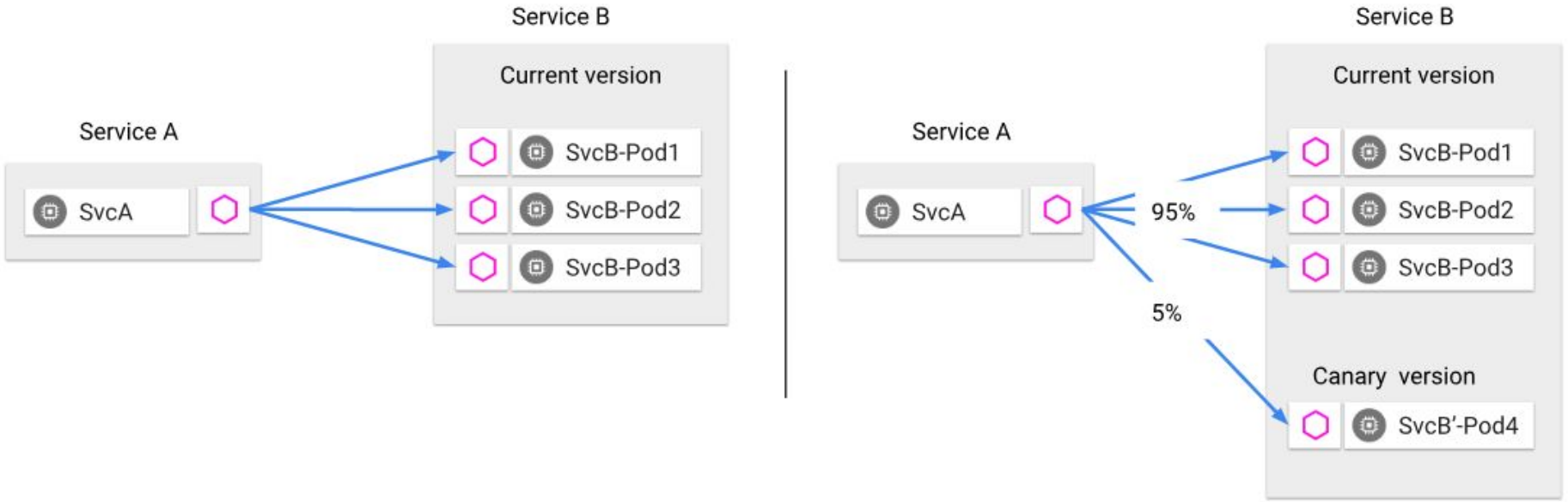
Istio Traffic Management

- Intelligent routing
  - A/B tests
  - Canary deployments
- Resiliency
  - timeouts,
  - retries,
  - circuit breakers,
  - etc.



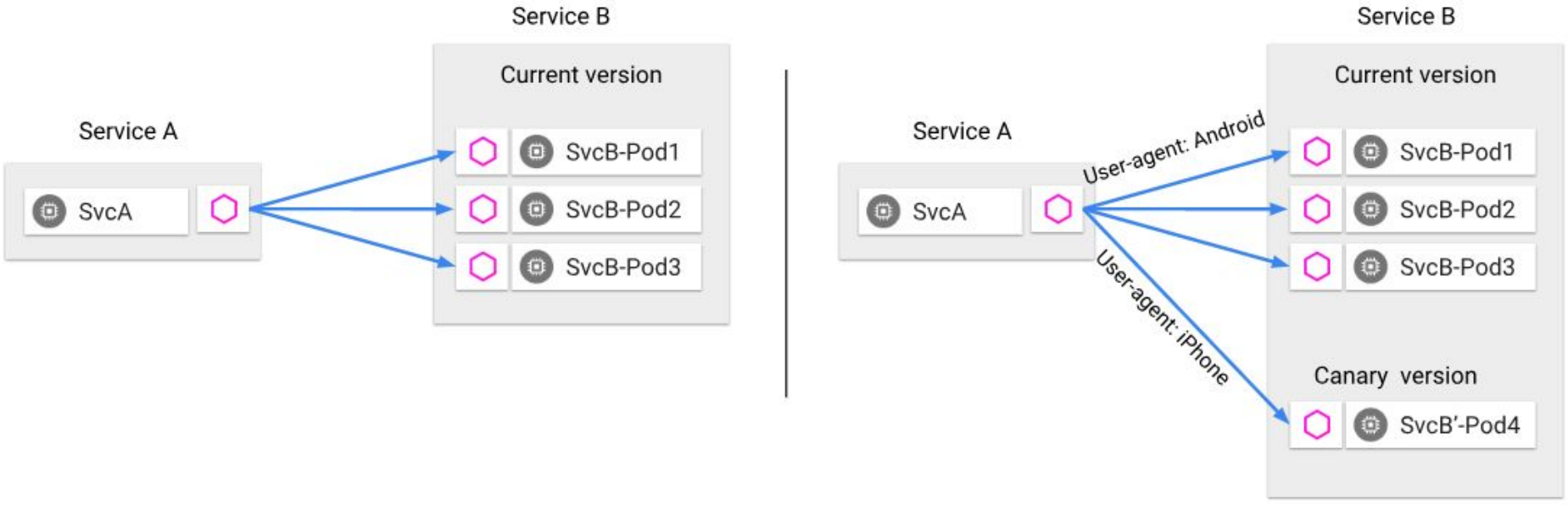
Istio architecture

Splitting Traffic Based on Traffic Proportion



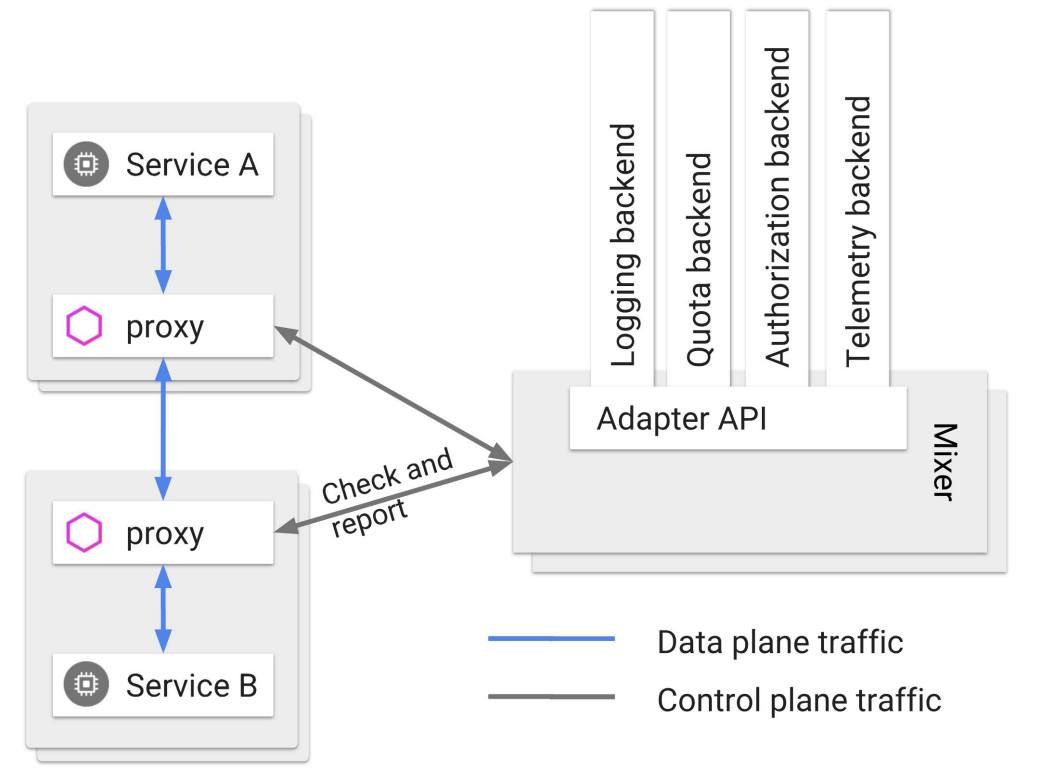
Traffic splitting decoupled from infrastructure scaling - proportion of traffic routed to a version is independent of number of instances supporting the version

Splitting Traffic Based on It's Request Content



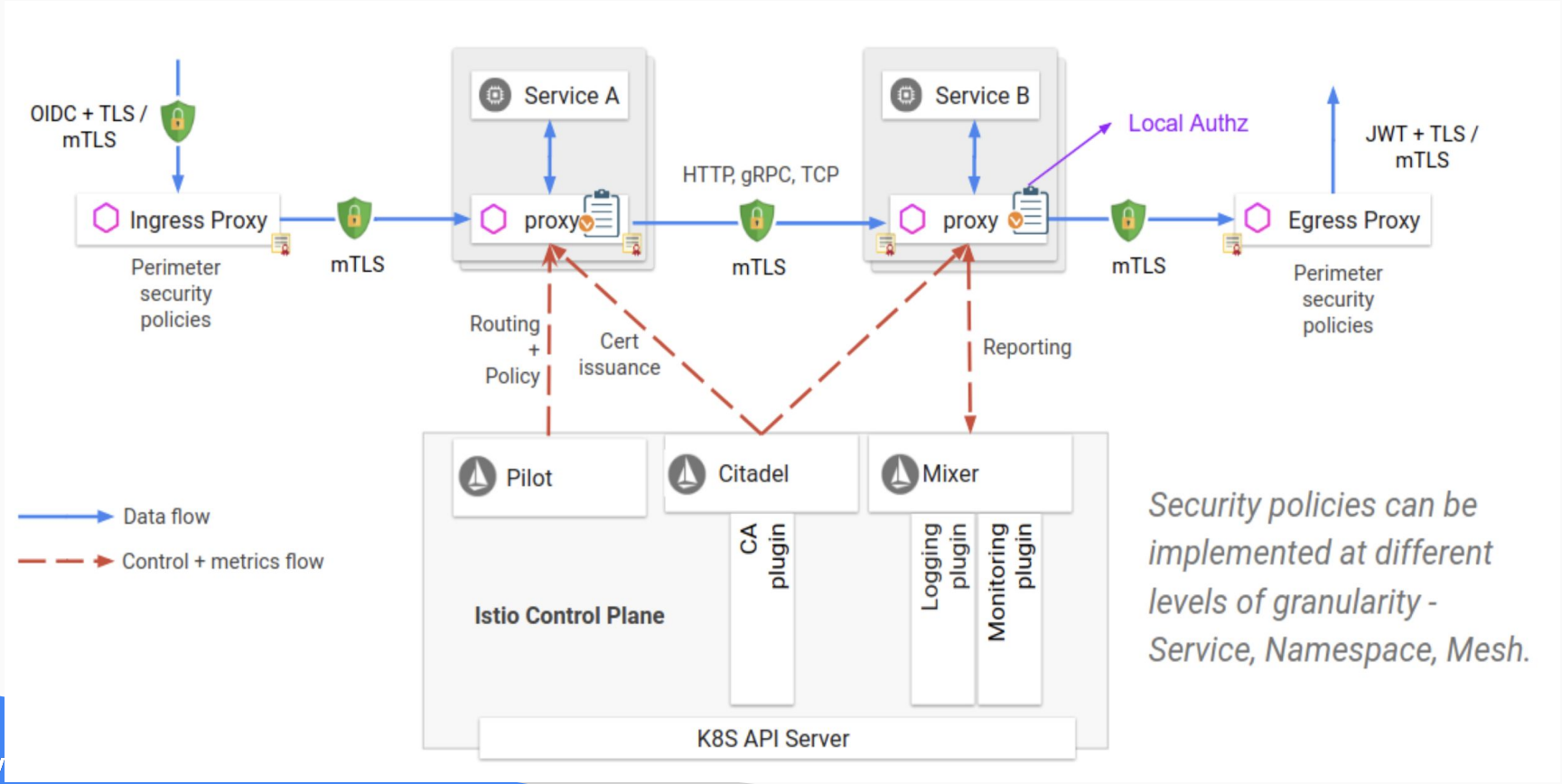
Content-based traffic steering - The content of a request can be used to determine the destination of a request

Istio Mixer for Collecting Metrics



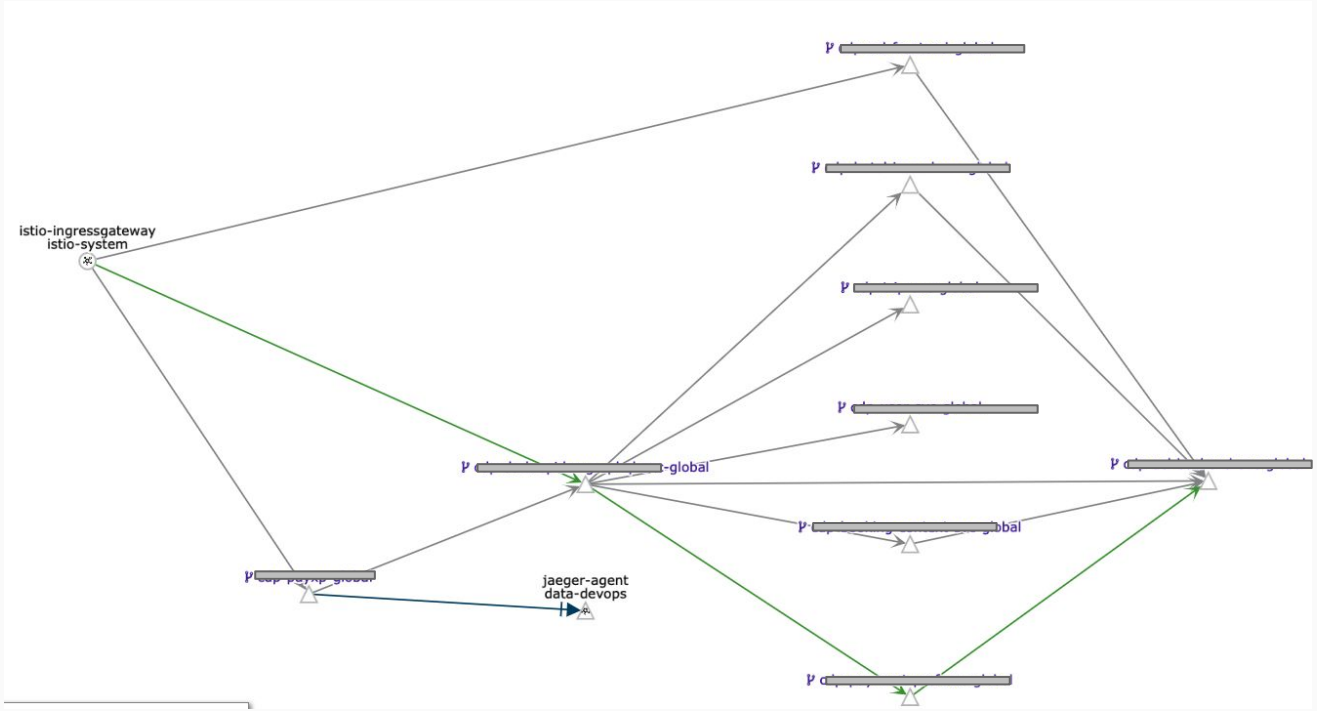
- Mixer is platform independent component
- Enforces access control and usage policies across the mesh.
- Collect telemetry data from envoy
- No vendor lock-in. This enables istio to interface with variety of backend

Istio Citadel for Cert Issuance



Running Istio @ Traveloka Data

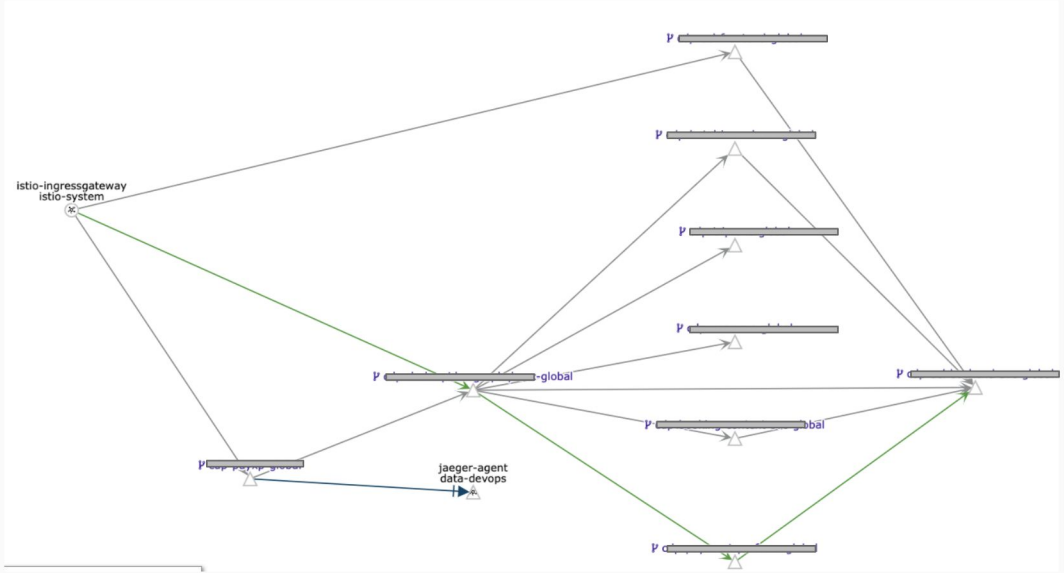
App that is running in istio?



- Mostly Golang + gRPC
- Some RESTful API
- GraphQL server
- Some visualizations & dashboards

Issues in Monitoring the Mesh (Before Istio)

- Unstandardized monitoring metrics.
  - Each service owner has his/her own approach in collecting the metrics with Datadog (latency, request count, etc)
- Hard to separate metrics produced by different version (canary and stable)
- Hard to see the client side metrics



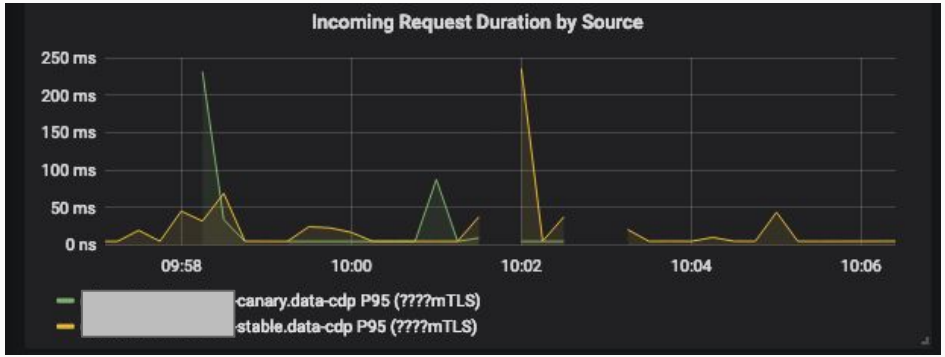
Monitoring Dashboard (With Istio)

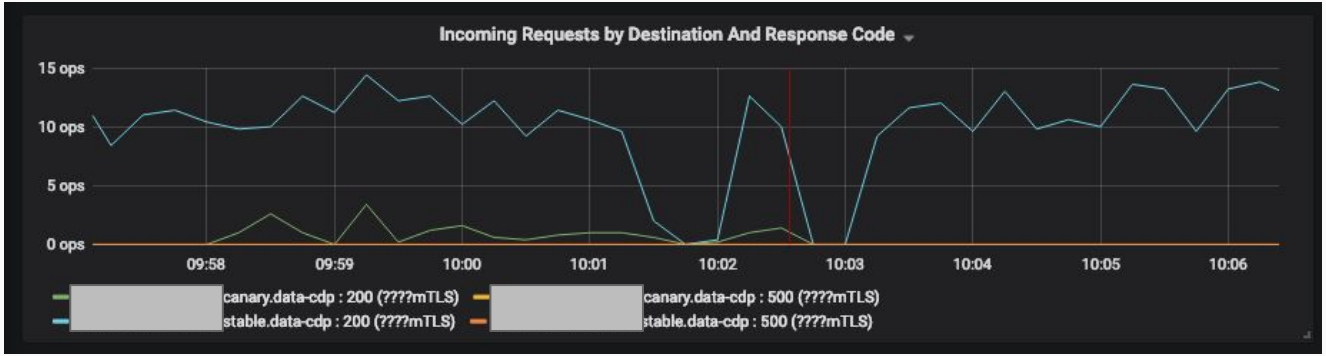
- RED* (**R**ate, **E**rror, **D**uration) metrics are collected by Istio Mixer and stored in Prometheus
- Those metrics are visualized in Grafana
  - Supported query by regex
  - Plug and Play some reusable dashboards

-

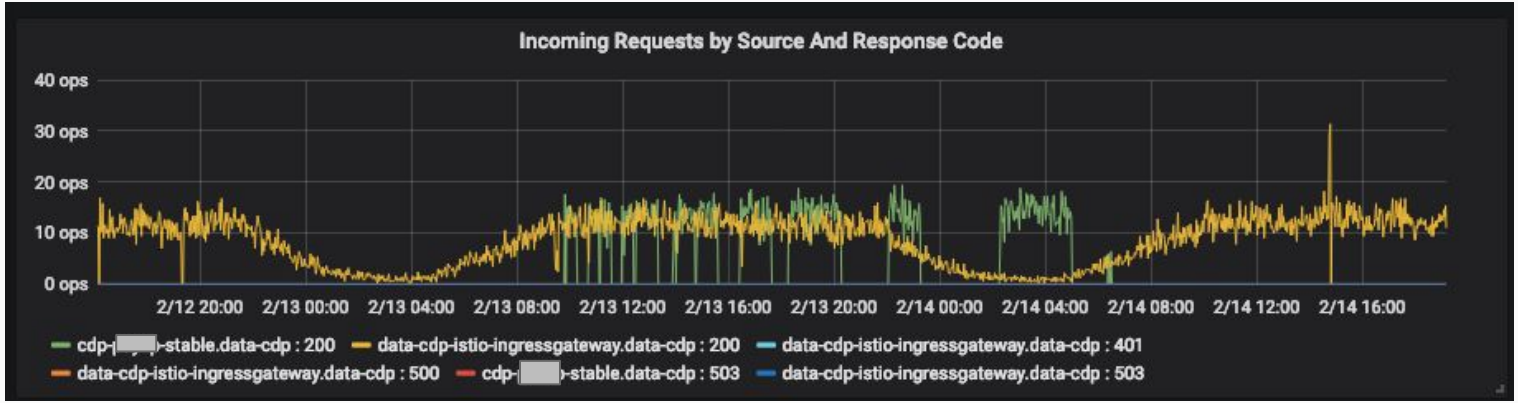
*https://www.weave.works/blog/the-red-method-key-metrics-for-microservices-architecture/

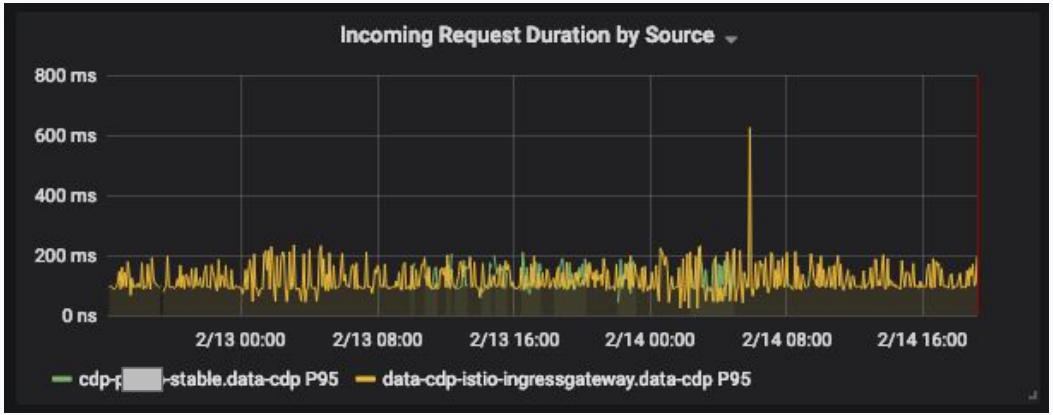
Stable & Canary monitoring



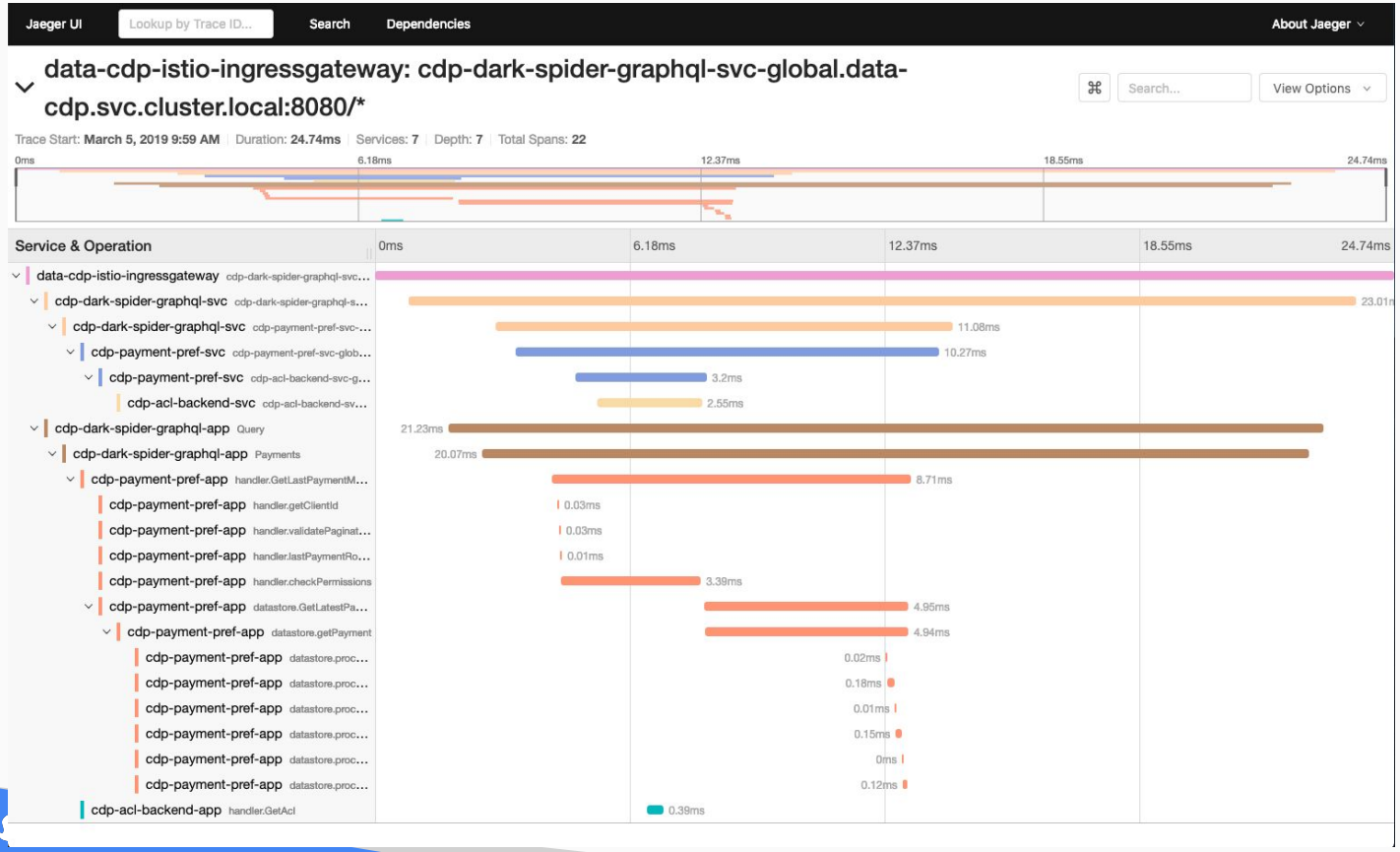


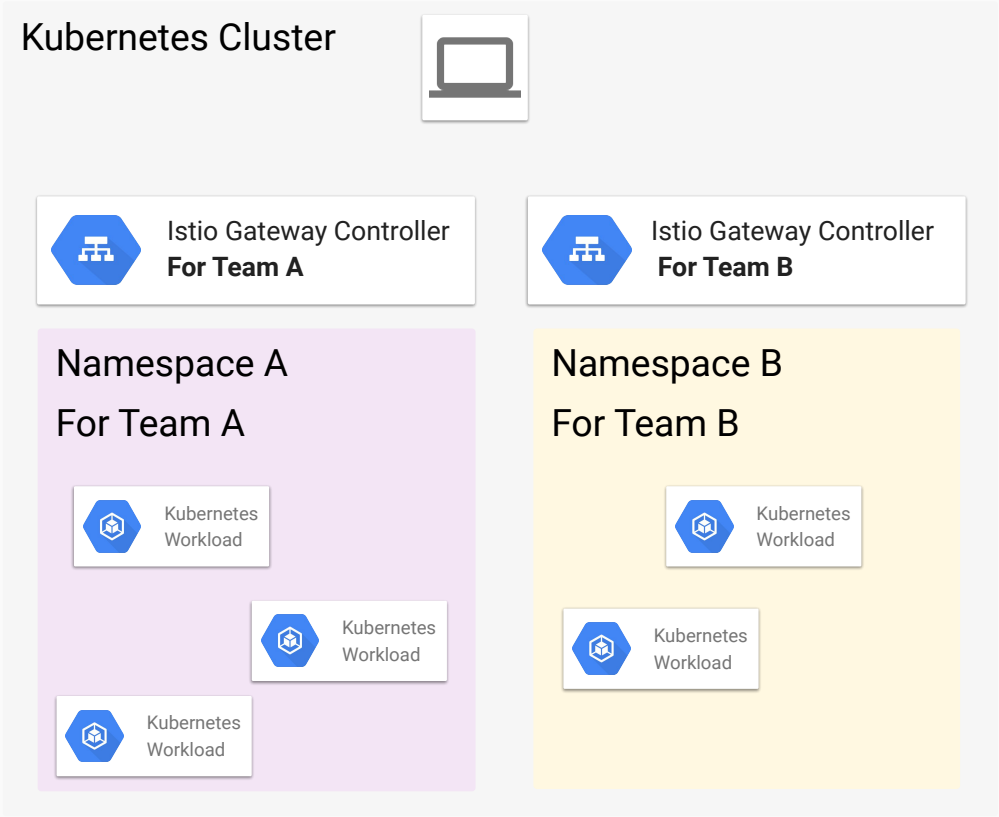
Client side metrics





Distributed Tracing with Jaeger





Dedicated Istio Gateway for Each Team

Benefits:

- Decentralized
- Failure isolation
- Cost Isolation

DevOps Helm Chart Template

- Expose relevant istio's configuration related to the traffic management, security policy, etc to service developer
- Reduced developer effort to maintain all of their istio and kubernetes manifests, Increased productivity.
- Split into two helm-chart
  - Application helm chart: app/pod deployment
  - Infra helm chart: infra, service routing, and security setting

```
---
# default values supplied for templates/* files
app:
  istioProxy:
    enabled: true
  name: example-app
  track: stable
  env: stg
  baseImageName: gcr.io/[redacted]/java-maven-app
  imageTag: c1448e9
  containerPort: 8080
  # see <protocol> supported here: https://istio.io/docs/setup/kubernetes/spec-requirements/
  containerPortProtocol: http

  # define volumes in the pod
  volumes:
  - name: example-app-secret-volume
    secret:
      secretName: example-app-secrets
      # set optional true if the secret is not that important
      optional: true
```

```
# mount specific volume to app's container
volumeMounts:
- name: example-app-secret-volume
  readOnly: true
  mountPath: /etc/example-app/secrets

# the app's container resource spec
resources:
  requests:
    cpu: "0.5"
    memory: "1G"
  limits:
    cpu: "1"
    memory: "2G"

# autoscale configuration
autoscale:
  enabled: true
  maxReplica: 10
  minReplica: 1
  targetCPUUtilizationPercentage: 60
```

Expose Service Through Istio Ingress Gateway

This configuration below is normally needed to expose your service to public.

```
infra:
  ingressGateway:
    enabled: true
    httpsRedirect: true
    selector:
      app: istio-ingressgateway # use other selector if you want to have a different gateway
    hosts:
    - "foo.tvlk-data.com"
```

Protect Service Access Within The Same Cluster with mTLS (Mutual TLS)

Normally, you only use TLS Certs managed by Istio Citadel, the Istio component that manage (e.g: renew and deploy) certificates automatically for your services.

```
infra:
# this trafficPolicy is a configuration that applied to <app-name>-global service communication.
  # it tells the client which uses istio-proxy, to use this trafficPolicy when communicating with our <app-n
  # see https://istio.io/docs/reference/config/istio.networking.v1alpha3/#TrafficPolicy to check the availab
  trafficPolicy:
    tls:
      # see: https://istio.io/docs/reference/config/istio.networking.v1alpha3/#TLSSettings-TLSmode
      # `mode` possible values: DISABLE, SIMPLE, MUTUAL, ISTIO_MUTUAL
      mode: ISTIO_MUTUAL

  # servicePolicy: is authentication policy enforcement when communicating with the `target` service in our
  # technically speaking, the `servicePolicy` field is a `spec` field in istio's Policy object, see: https:/
  servicePolicy:
    # targets: a list of target service in a shortname.
    targets:
    - name: [redacted]-svc-global
    peers:
    - mtls:
        # mtls set to STRICT in production
        mode: PERMISSIVE # possible values are PERMISSIVE, STRICT
```

Split Traffic to Different Track of Service

Given there are 2 track of services: `stable` and `canary` have been deployed, you can split the traffic based on weight (with total of 100 percent). Here's how you can do that

```
infra:
  trafficRoute:
    weight:
      stable: 95
      canary: 5
    canaryCondition:
      match:
      - headers:
          end-user:
            exact: apratama
          key:
            exact: agung
```

Traffic Mirroring/Shadowing to Different Version/Track of Service

Given there are 2 track of services: `stable` and `canary` have been deployed, you can mirror the current `stable` traffic to the `canary` as well, without impact end-user. The response from `canary` service isn't delivered to the upstream (end-user), so you can think of it as fire-and-forget. The only response delivered to the upstream is from `stable`. Here's how to do that:

```yaml
infra:
  trafficRoute:
    weight:
      stable: 100
      canary: 0
    mirror:
      enabled: true
      track: canary
```

Continuous Integration & Delivery

Asking Image Release Input	Initialization	Checkout	Pull Image	Create Kube Secret	Canary Deployment to Prod Cluster	Canary to Stable Promote Verification	Stable Deployment to Prod Cluster
29ms	24s	16s	7s	18s	5min 20s	28ms	2min 7s
28ms (paused for 35s)	23s	15s	6s	20s	6min 44s (paused for 38s)	29ms (paused for 2min 38s)	1min 52s

- Decide which image to deploy
- Ask how much traffic goes to canary?
- Verify the canary work as expected?
- Prod deployment

Deploy w/ Makefile

```
stage("Stable Deployment to Prod Cluster") {
    echo "Deploy ${appImage} as stable deployment to cluster"

    sh "make deploy-app TRACK=stable ${commonDeployParams} ARGS=\"--set app.autoscale.enabled=true --timeout ${deployWaitInSeconds} --force\""

    echo "Setup Infra around application"
    sh """
    make deploy-infra ENV=${environment} NAMESPACE=${appNamespace} \
    ARGS=\"--set infra.trafficRoute.weight.canary=0 --set infra.trafficRoute.weight.stable=100 --timeout ${deployWaitInSeconds} --force\"
    """

    echo "Cleaning up canary deployment"
    sh """
    sleep 10
    make delete APP_NAME=${appName} TYPE=canary
    """
}
```

Lesson Learned

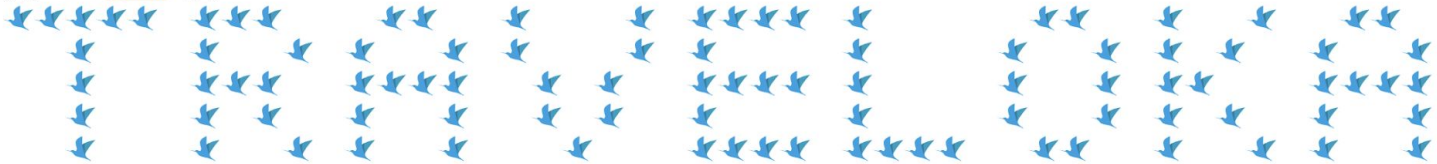
Lesson learned

- FYI, we are running OSS Istio in GKE.
- At least, our PoC can handle 800 RPS, enough for now
- Some Issues we found
  - Different architectured used in our staging and prod made incident post mortem became harder when we got issue in one of those env. (Hard to reproduce the issue)
  - gRPC mirroring doesn't work as expected in istio.
  - Authentication for human user interaction doesn’t exists in Istio.
- Always run enough load test to:
  - Fail faster
  - Tune the autoscaling config
  - Tune monitoring resiliency

traveloka

Question?

23:18 **ngegas** APP @Imre ngegas:



@Imre ngegas:

send email to: imre.nagi@traveloka.com for referral 😛