
fastly

fastly

OBSERVING FASTLY'S NETWORK AT SCALE THANKS TO K8S AND THE STRIMZI OPERATOR

Agenda

- Introduction
- Project and platform overview
- Solution description
- Problems, issues and learnings
- Closure

INTRODUCTION

Fernando Crespo Grávalos

- Staff SREngineer @ **Fastly**
- Previously Senior SRE@**Tuenti**, **NAGRA**, **Bitnami**…
- *OSS contributions* (see tuenti secrets manager, instaclustr_exporter)
- *Medium blog posts*

Daniel Caballero Rodríguez

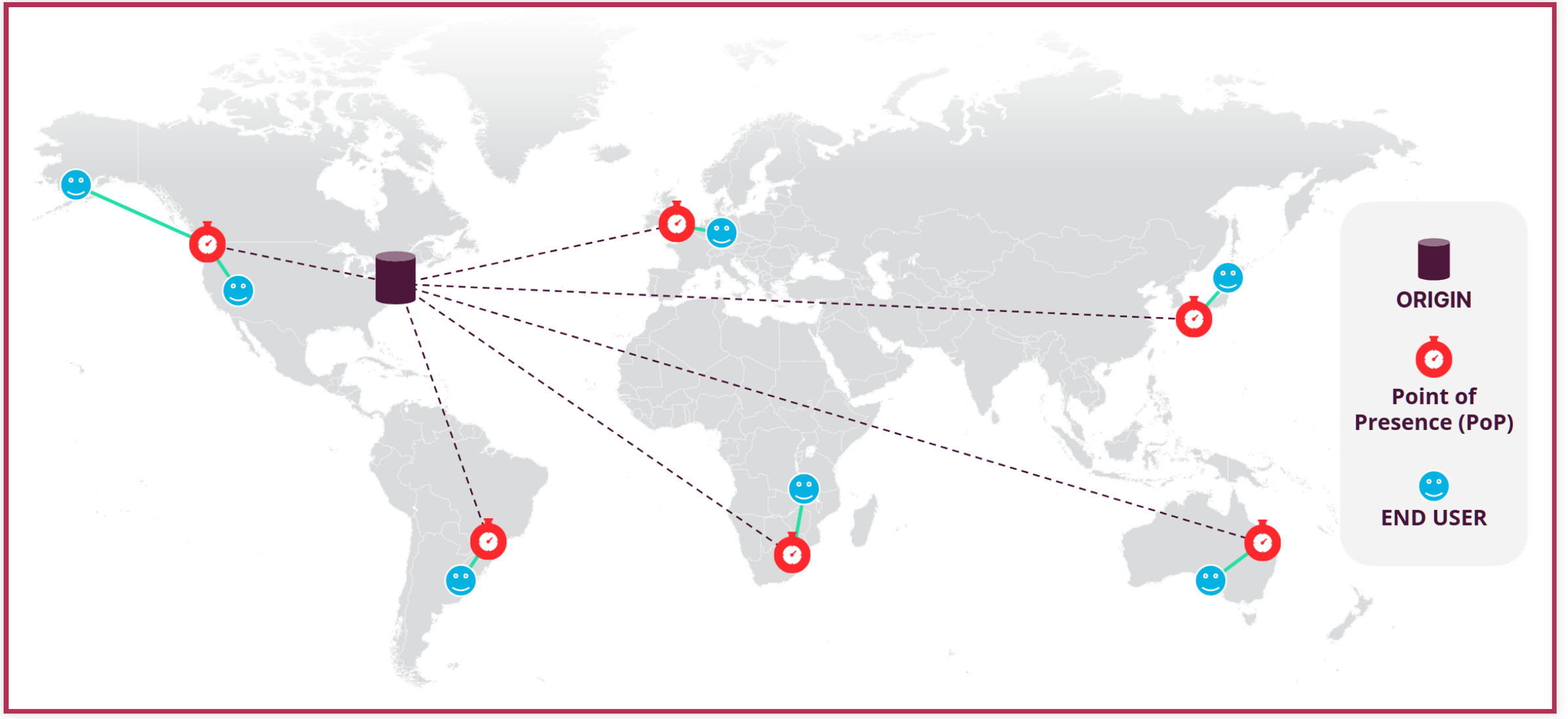
- Principal SREngineer @ **Fastly**
- Previously part time (Devops) lecturer, Devops@**Schibsted** (now Adevinta), **NTT**, **Oracle**…
- *Eventual OSS contributions* (see tcpgoon)

WHAT IS THIS TALK ABOUT?

How K8s and the Strimzi Kafka Operator helped implementing a scalable telemetry pipeline, core for traffic management automation


fastly

OUR BUSINESS



WHAT MAKES US UNIQUE

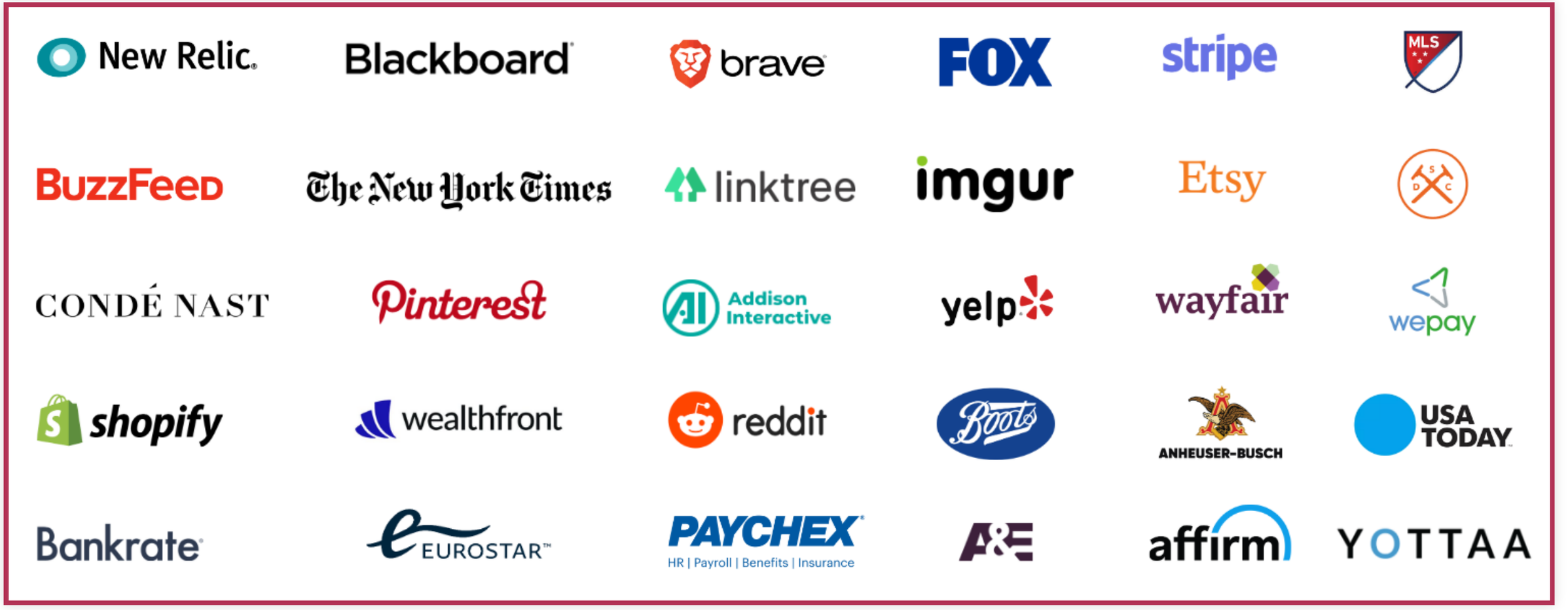
Software Defined Network — **Programmable Edge** — **Customer Empowerment Philosophy**

Edge Advantages

- Better performance
- Better UX
- More agility
- CI/CD integration
- Instant scale
- Greater control
- Heightened security
- Real-time visibility

- Reduce TCO
- Lower demand on origin
- Reduce time to deploy

OUR CUSTOMERS ACROSS INDUSTRIES



FASTLY AT A GLANCE

800B+

Requests served daily[1]

192 Tbps

Edge capacity[2]

95%+

Customer satisfaction score[3]

2,800+

Customers[4]

$354.3M

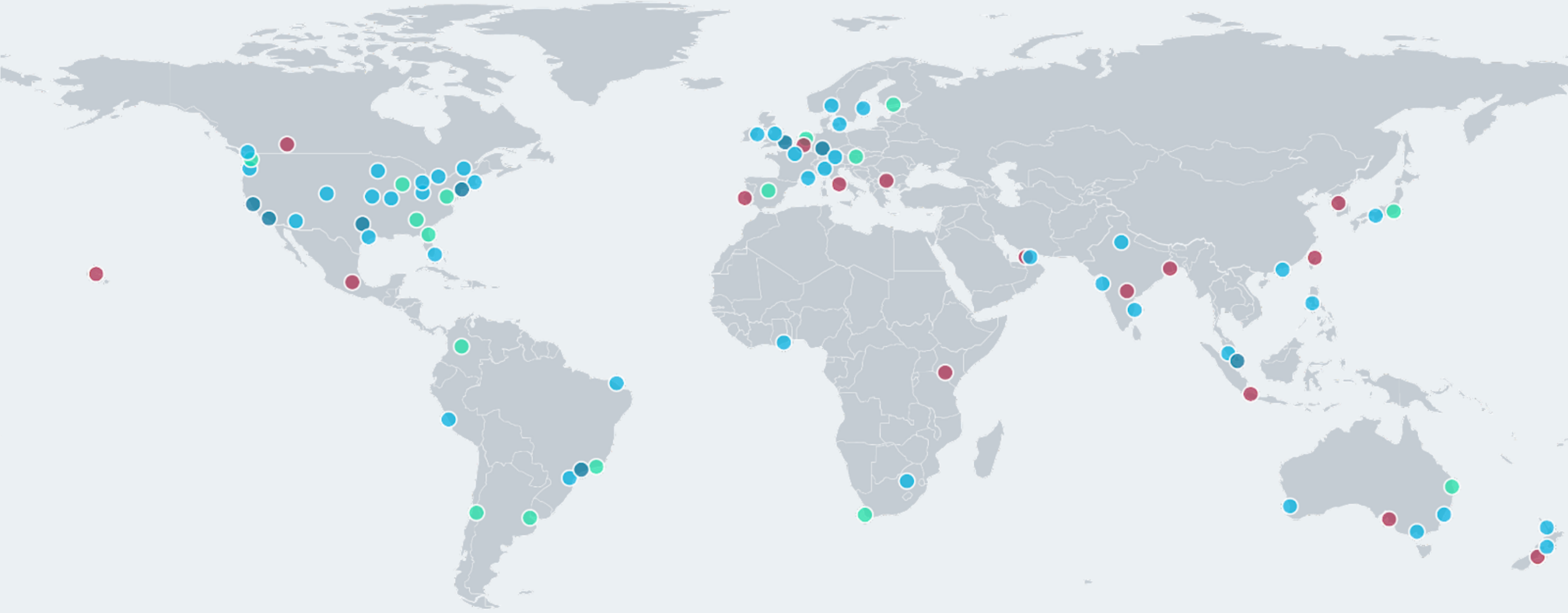
2021 Revenue

99%

Annual Revenue Retention[5]

1. As of December 30, 2020
2. As of January, 2022
3. As of February 1, 2022
4. As of February 16, 2022
5. As of February 16,, 2022

Fastly Points-of-Presence (POPs)



Current POP | Current multi-POP | Planned new POP | Planned POP expansion

Updated Dec 2021

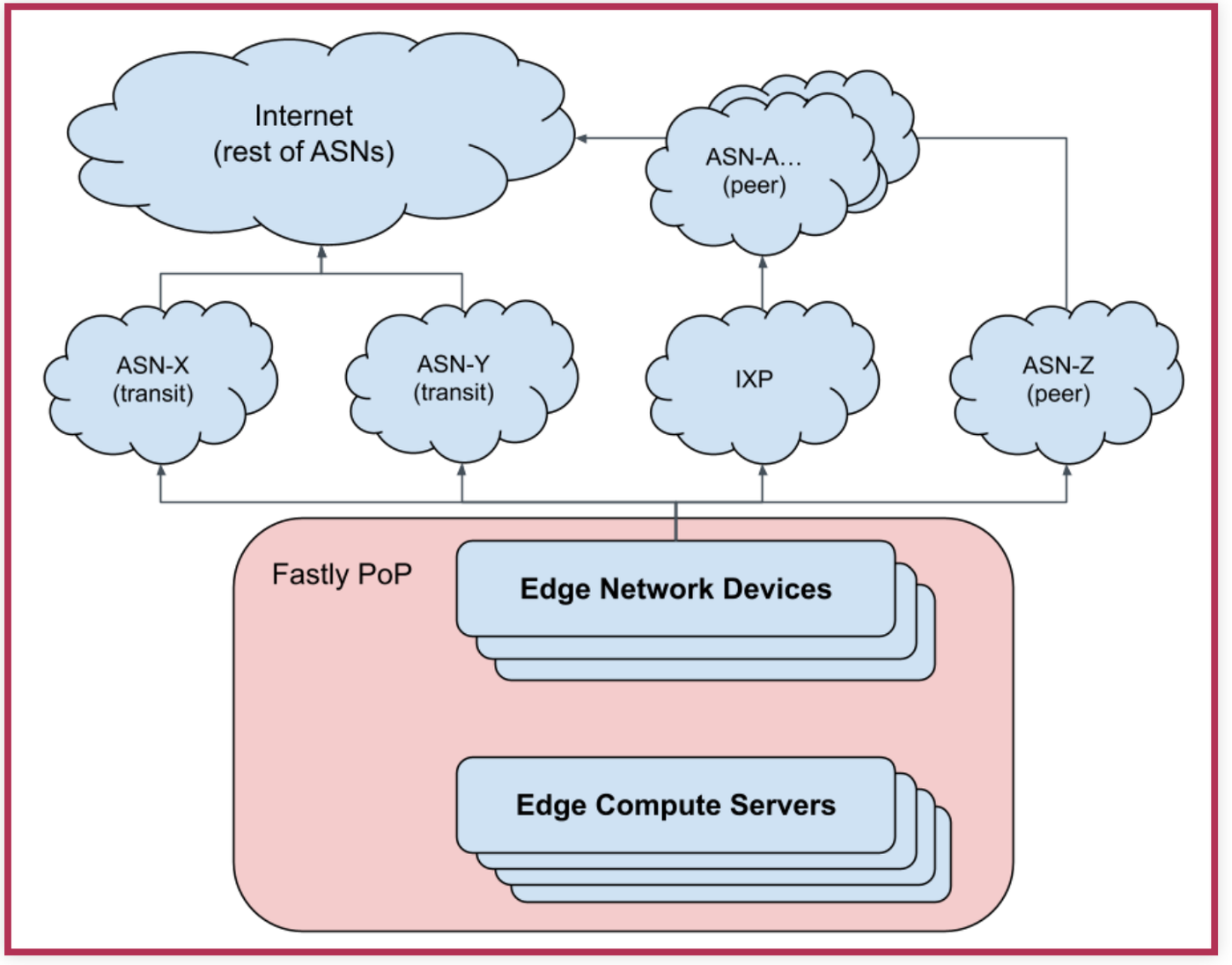
BUT THERE IS MORE

We have a control plane

PROJECT AND PLATFORM OVERVIEW

THE PROJECT: AUTOPILOT AND ITS TELEMETRY PIPELINE

CONTEXT: FASTLY AND EGRESS TRAFFIC ENGINEERING



Automation in this problem domain is **not a new concept…**

How network automation helps Fastly support the world's biggest live-streaming moments

Ryan Landry
Vice President, Technical Operations

NETWORK PERFORMANCE REAL-TIME INSIGHTS STREAMING

Published March 10, 2020

One of the keys to a clean, clear live-streaming experience is properly managing network congestion — something our platform performs mostly automatically, around the clock, and with reduced human intervention. As part of Fox Sports' live event streaming team for Super Bowl LIV, we were able to watch this automation

Want to continue the conversation?

Schedule time with an expert

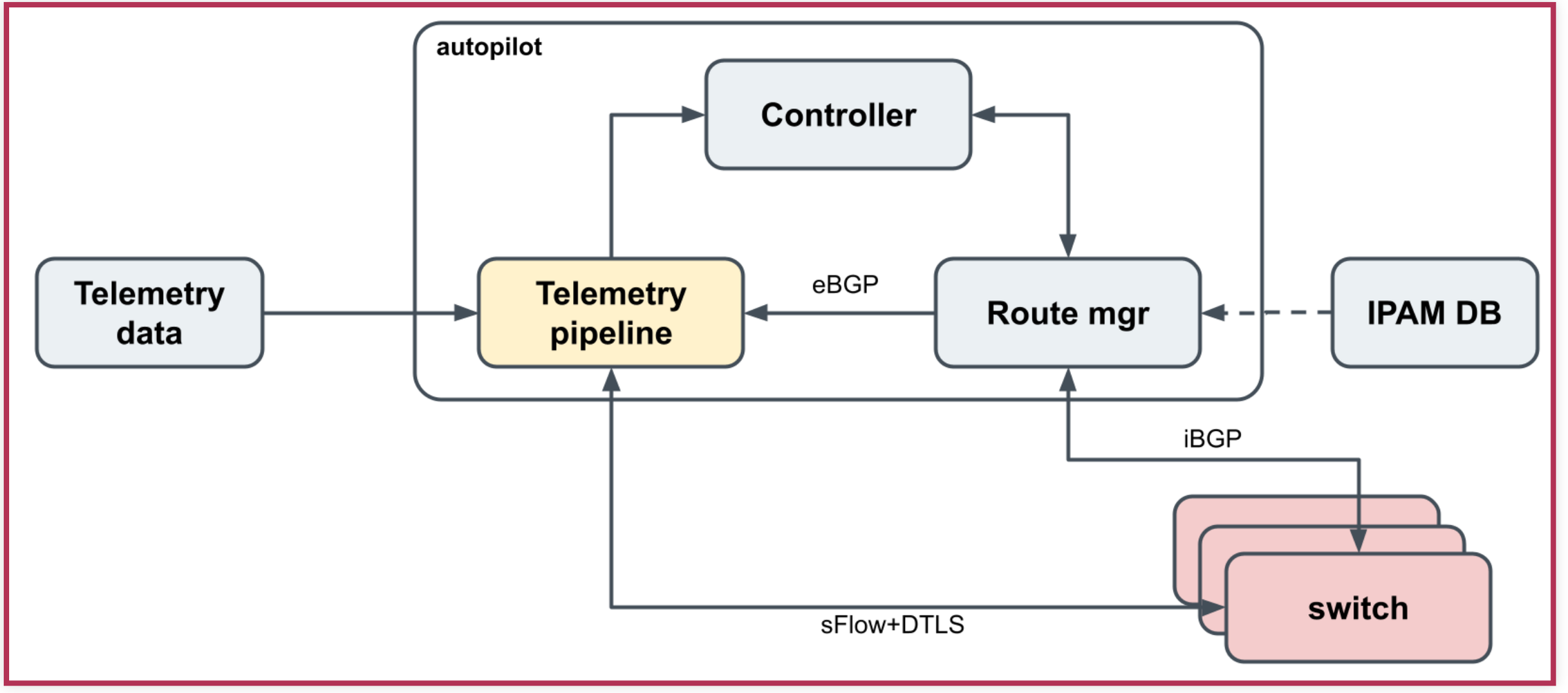
https://www.fastly.com/blog/network-automation-helps-support-worlds-biggest-live-streaming-moments

AUTOPILOT AS THE LAST ITERATION

- Developed and maintained by the *Network Control and Optimization* (**NCO**) team
- Initially around 6 people…
  - across 4 different countries
  - and different profiles (Software Engineers, Data Engineers, Network Engineers, SRE)

Note: Tech decisions need to factor team skills and size

AUTOPILOT AT HIGH LEVEL



THE TELEMETRY PIPELINE

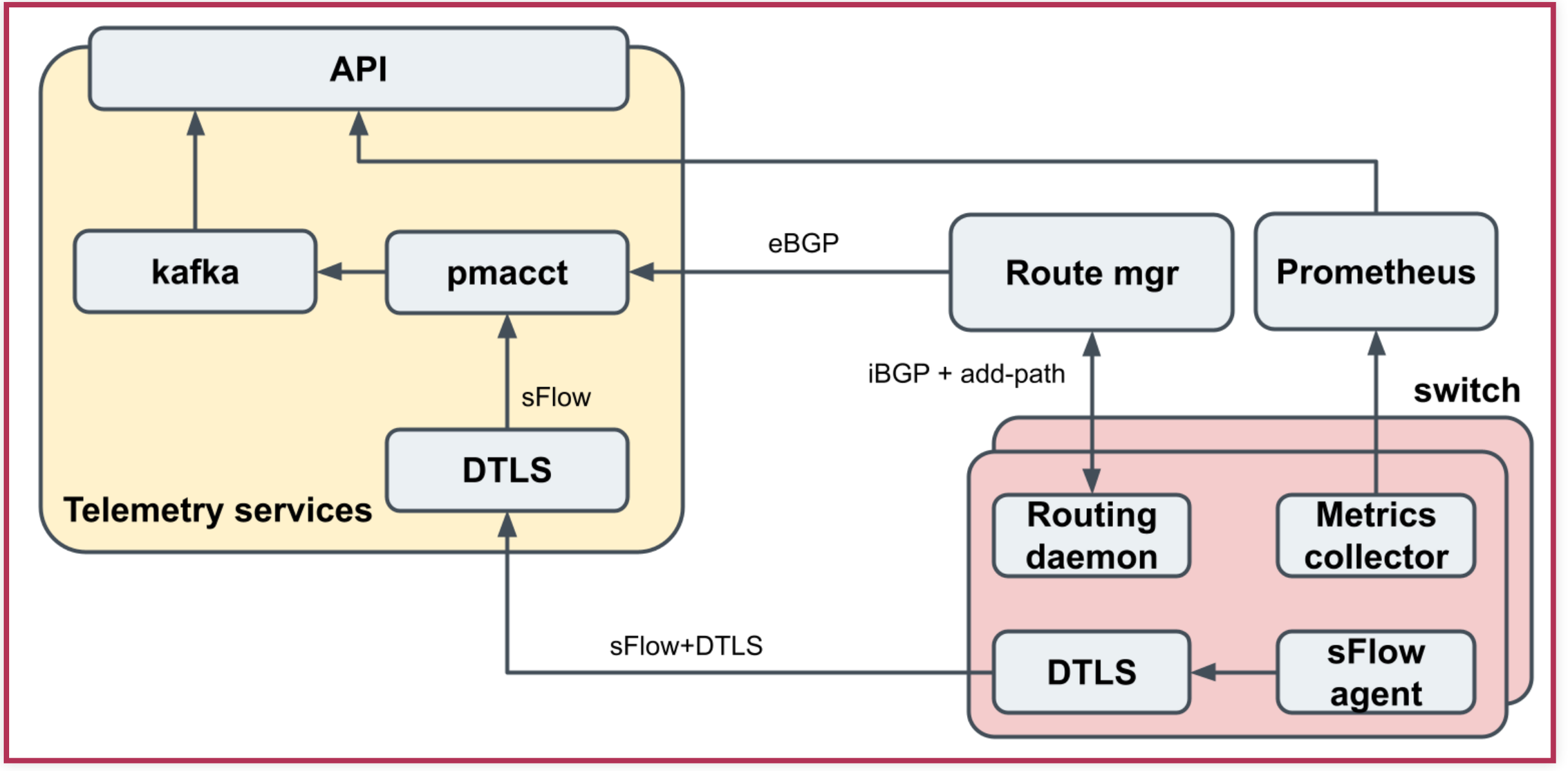
We need to process (lots of) telemetry data…

- Systems metrics
- Flow data (sampling packet headers from our fleet)
- Performance data
- Synthetic traffic / connectivity checks results

… and Autopilot is not the only use case:

- Our NetOps team also gets PoPs traffic insights
- Capacity planning
- New DDoS solutions
- Blog posts :)
- Others we may not even anticipate yet

PIPELINE DESIGN OVERVIEW



IMPLEMENTATION REQUIREMENTS

IMPLEMENTATION REQUIREMENTS

To process telemetry and flow data at **high scale**

IMPLEMENTATION REQUIREMENTS

To process telemetry and flow data at **high scale**

Cloud-agnostic / multi-cloud / multi-region

IMPLEMENTATION REQUIREMENTS

To process telemetry and flow data at **high scale**

Cloud-agnostic / multi-cloud / multi-region

Runtime that let's us deploy/manage apps **outside our fleet**

We did not have a proper standard

- Very well known patterns to deploy components to our fleet / data-plane…
- … the control plane was a different story
- NCO had dedicated GKE clusters for specific workloads
- And Kafka@Fastly was just in experimentation phase
  - There was actually opposition to introduce Kafka in the project scope

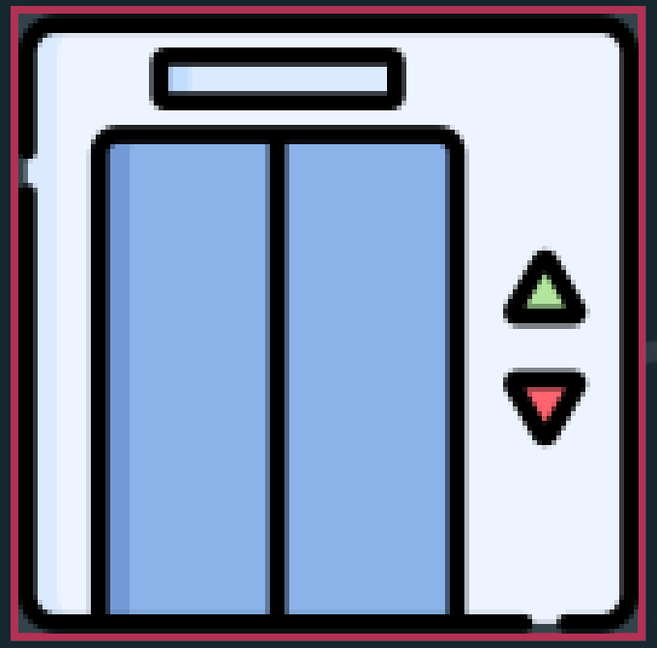
K8S@FASTLY CIRCA 2020

- Very fragmented infrastructure
- Mostly individual initiatives
- No standard way of creating k8s clusters or to deploy applications
- Engineers not necessarily versed on cluster administration
- Heavily used for CI

K8S TEAM

- Initially: New Zealand (2), Canada (1), Spain (1)
- Goals:
  - Build a k8s shared platform
  - Standarize infrastructure and deployments
  - Scale cross-regions, cross-clouds
  - **Elevate** developers capabilities to deploy and scale faster
- The latter brought the project name: **Elevation**

ELEVATION

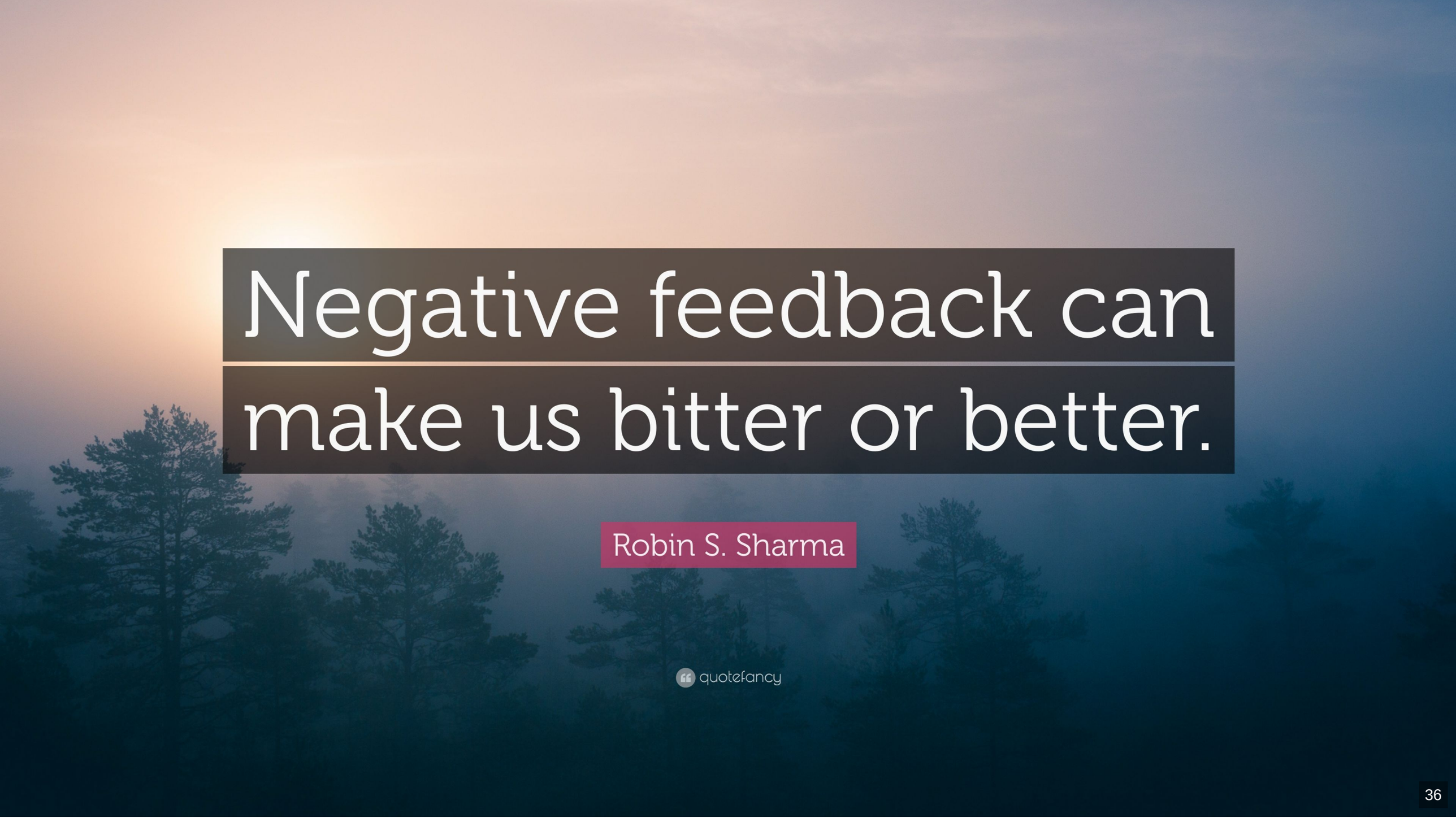


ELEVATION 1.0

- 3 Clusters: dev, staging, prod
- 1 region, but designed to span across regions
- 1 cloud, stack designed to be cloud agnostic:
  - Auth with our IdP, not tied to the Cloud IAM
  - Same deployment mechanism: Helm Charts via FluxCD (GitOps)
  - Harbor as Container/Helm Charts registry
  - Vault for secrets management
  - Nginx Ingress
  - Cert-Manager to issue certs from Lets Encrypt
  - Observability: Prometheus/Grafana, FluentD/Splunk
  - Service Mesh with Linkerd

ELEVATION 1.0 FEEDBACK

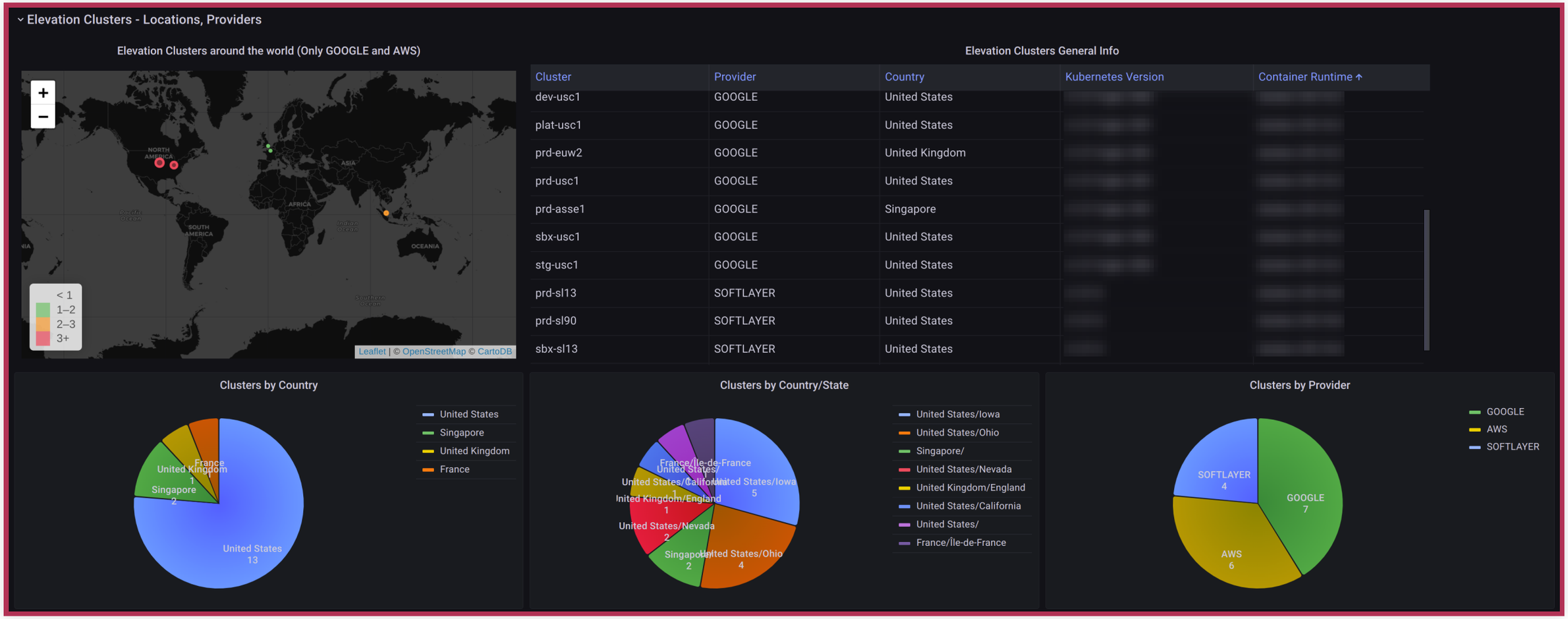
- Multi-Region, Multi-Cloud
- Reduce the onboarding overhead
- In-Cluster Kafka support
- Steep learning curve
- Seamless integration with our IdP for applications
- Improve service mesh observability


Negative feedback can make us bitter or better.
Robin S. Sharma
quotefancy

ELEVATION 2.0 - K8S @FASTLY TODAY

2 new team members: US, UK

More Cloud Providers and Regions



Onboarding Improvements

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
[...]
  rules:
  - generate:
      apiVersion: vault.fastly.com/v1beta1
      data:
        spec:
          mountInput:
            options:
              version: "2"
            type: kv
          path: '{{request.object.metadata.name}}'
      kind: SecretEngine
      name: '{{request.object.metadata.name}}'
      namespace: vault
    match:
      any:
      - resources:
          kinds:
          - v1/Namespace
          selector:
            matchLabels:
              gen-vault-kv-store: enabled
    name: vault-kv-store
```

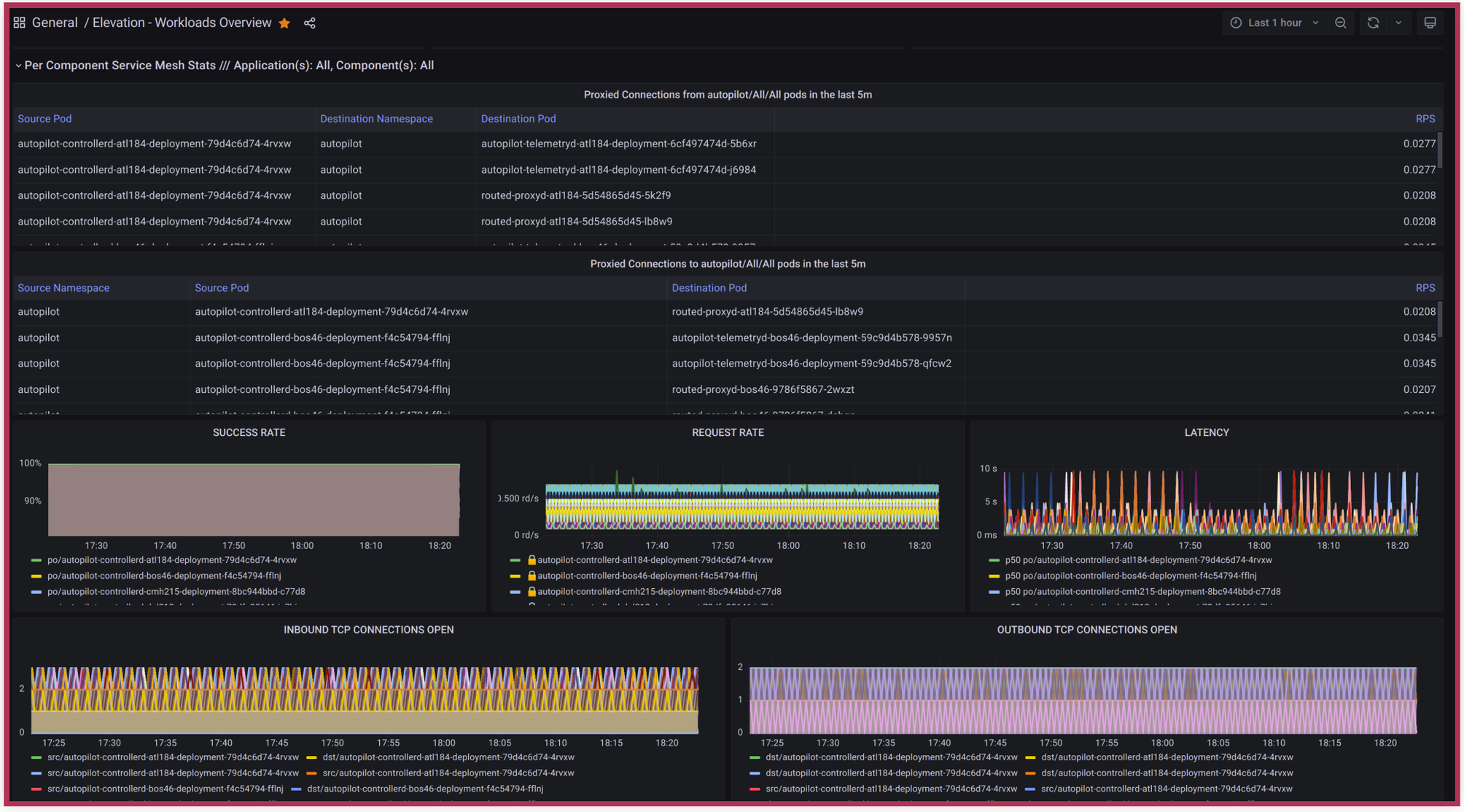
Idp auth as a Service

```
nginx.ingress.kubernetes.io/auth-signin: https://vouch.int.usc1.dev.k8s.example.com/login?url=$scheme://$http_host$request_uri&vouch-failcou
nginx.ingress.kubernetes.io/auth-url: https://vouch.int.usc1.dev.k8s.example.com/validate
nginx.ingress.kubernetes.io/auth-response-headers: X-Vouch-User
nginx.ingress.kubernetes.io/auth-snippet: |
  # these return values are used by the @error401 call
  auth_request_set $auth_resp_jwt $upstream_http_x_vouch_jwt;
  auth_request_set $auth_resp_err $upstream_http_x_vouch_err;
  auth_request_set $auth_resp_failcount $upstream_http_x_vouch_failcount;
```

Abstractions

```
app:
  name: foo
  components:
  - name: example
    containers:
    - name: web
      image:
        repo: fastly/foo-example
        tag: v1.21.0
      ports:
      - number: 8080
        protocol: HTTP
        expose: true
        ingress:
          http:
            routes:
            - host: foo.example.com
              prefix: /
```

OBSERVABILITY IMPROVEMENTS



Deployments status

Helm Charts - Deployment Status

chart	release	version	status
routed-proxyd	routed-proxyd	0.0.9	Deployed
flow-proxy	flow-proxy	0.0.18	Deployed
telemetry-api	telemetry-api	0.2.11	Deployed
controllerd	controllerd	0.2.8	Deployed
telemetry-sfac…	telemetry-sfacctd	0.0.20	Deployed
telemetry-kafka	telemetry-kafka	1.0.0	Deployed

Flux - Reconciliation Status

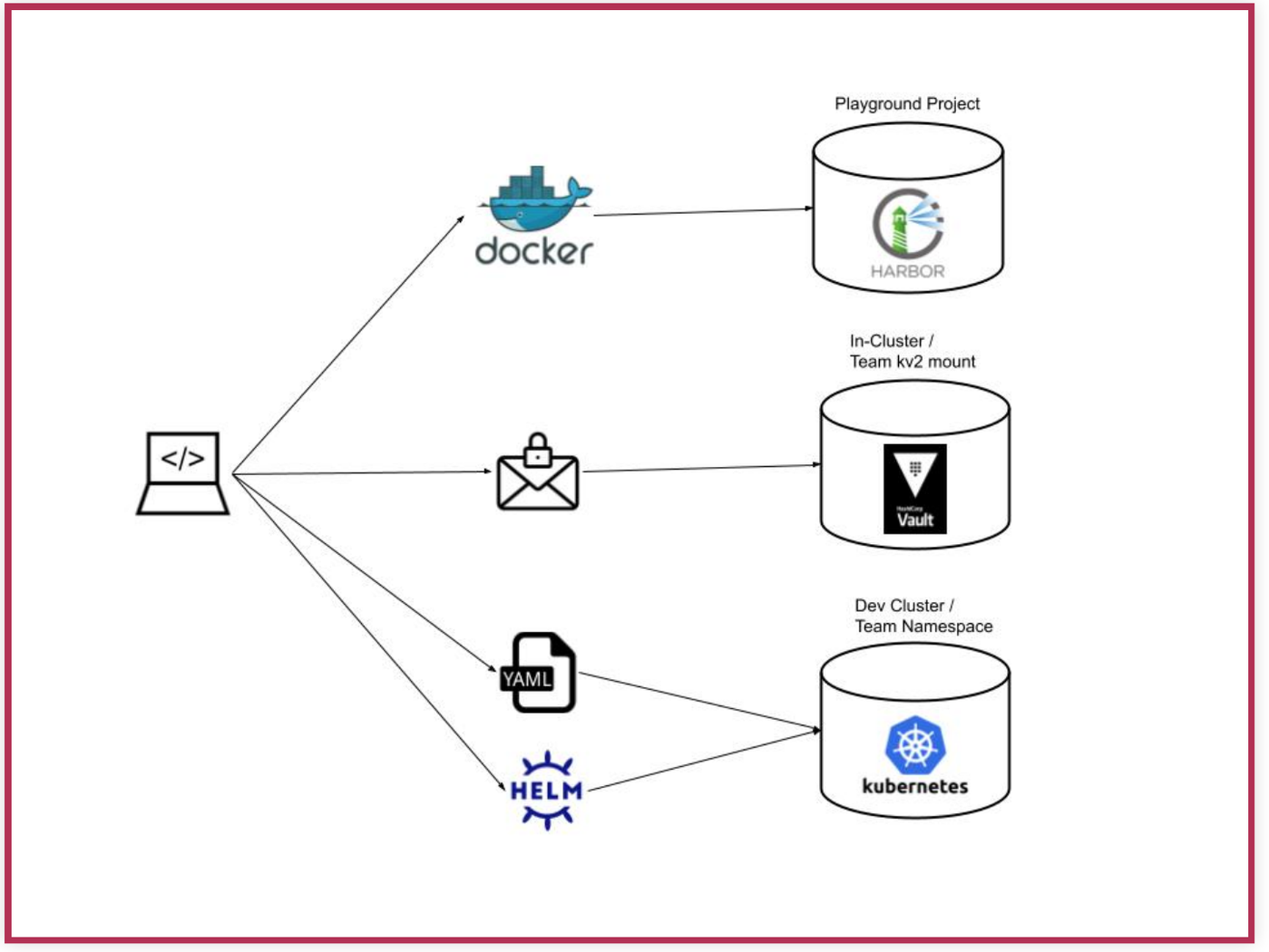
name	ready?	suspended?
controllerd-prd-usc1	Yes	No
routed-proxy-prd-usc1	Yes	No
telemetry-api-prd-usc1	Yes	No
telemetry-flow-proxy-prd-usc1	Yes	No
telemetry-kafka-prd-usc1	Yes	No
telemetry-sfacctd-prd-usc1	Yes	No

In-Cluster Kafka support

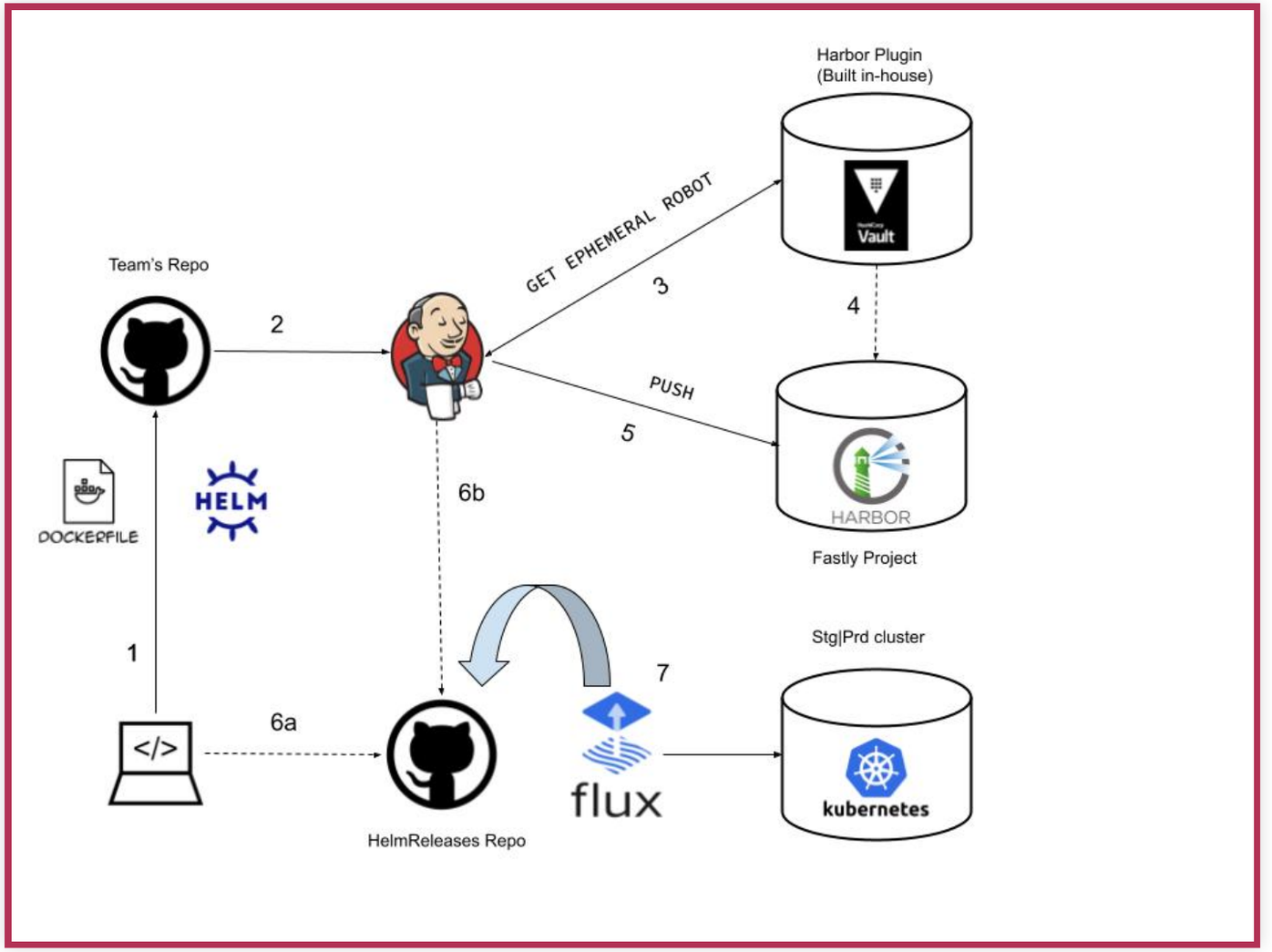
DEPLOYING ON ELEVATION: FROM DEV TO PROD

Explore, Build, Deploy

EXPLORE



BUILD AND DEPLOY




THE GOOD
THE BAD
AND THE UGLY

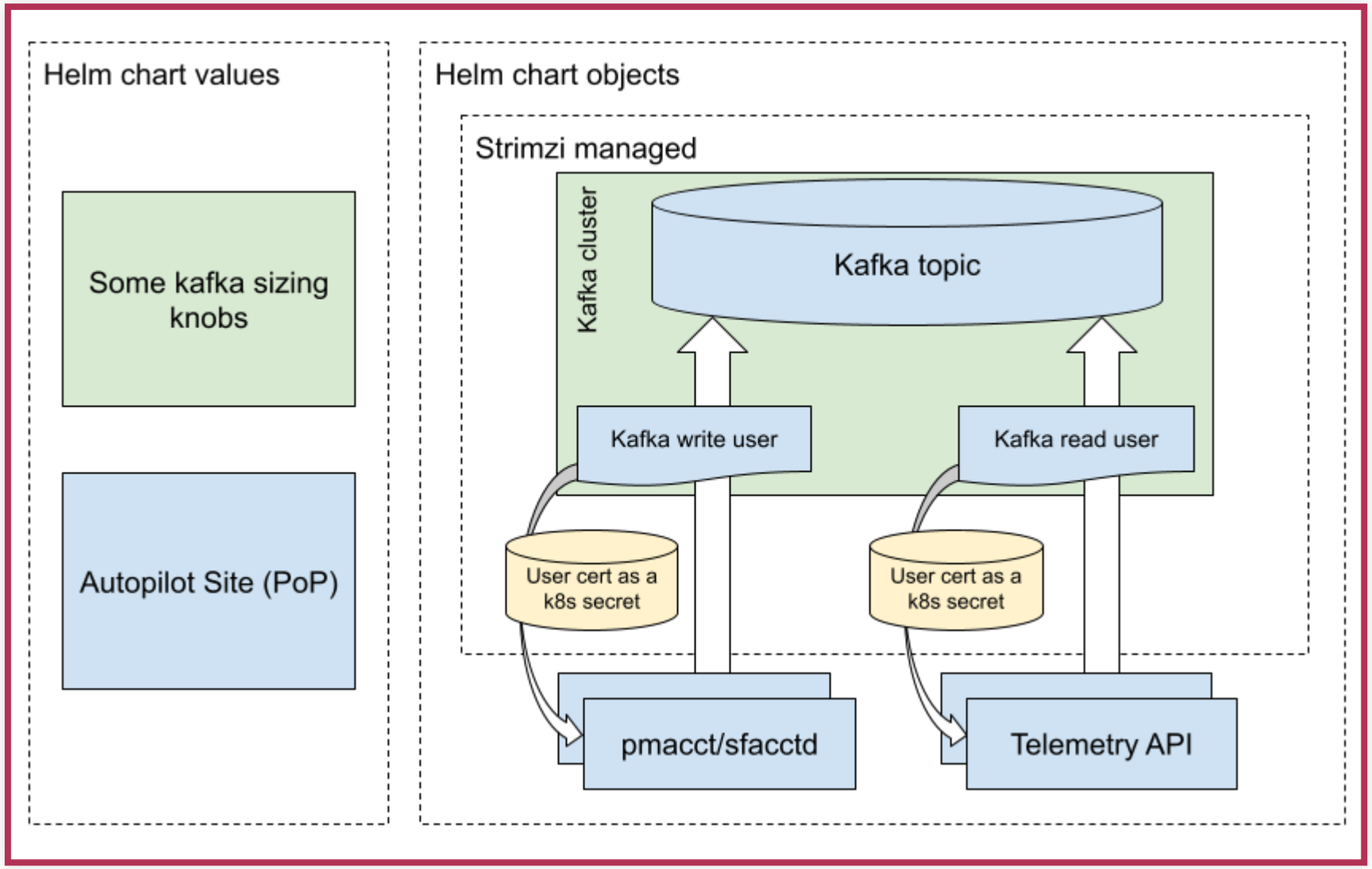
THE GOOD

```yaml
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 3
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
    config:
      offsets.topic.replication.factor: 1
      transaction.state.log.replication.factor: 1
      transaction.state.log.min.isr: 1
      default.replication.factor: 1
      min.insync.replicas: 1
  zookeeper:
    replicas: 3
    storage:
      type: persistent-claim
      size: 100Gi
  entityOperator:
    topicOperator: {}
    userOperator: {}
```

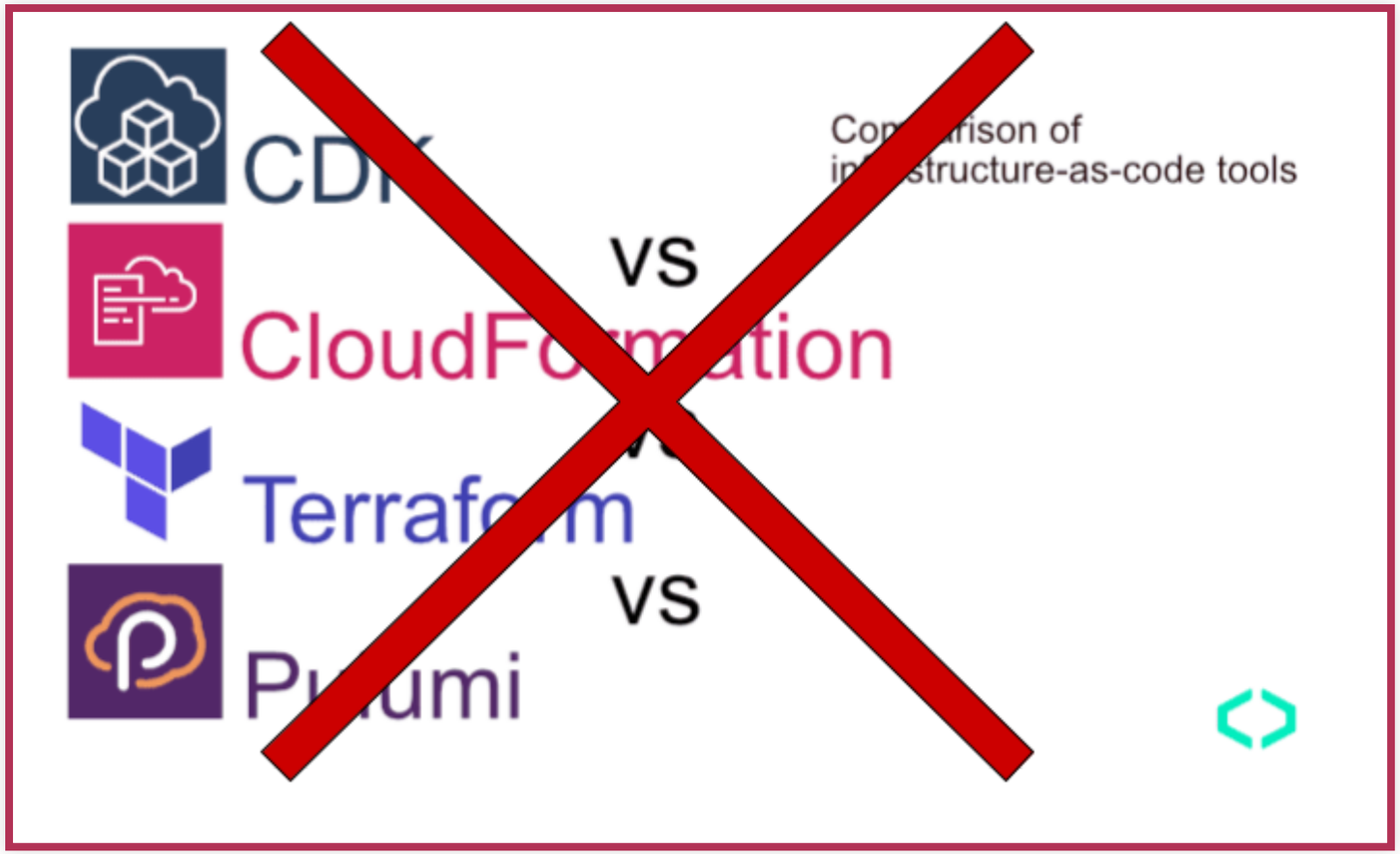
OUR ABSTRACTION

```yaml
spec:
  releaseName: telemetry-kafka
  interval: 5m
  chart:
    spec:
      chart: telemetry-kafka
      version: 1.0.0
      sourceRef:
        kind: HelmRepository
        name: fastly-charts
        namespace: platform-flux
  values:
    kafka:
      replicas: 5
      volumesSize: 50Gi
      resources:
        requests:
          memory: 10Gi
          cpu: 500m
        limits:
          memory: 16Gi
          cpu: '4'
    zookeeper:
      replicas: 5
      volumesSize: 20Gi
    autopilotSites:
    - PoP-ID1
    - PoP-ID2
    - PoP-ID3
    - PoP-ID4
```

Kafka cluster deployment and configuration is integrated



So no other IaC solutions or tooling are required



Not having a problem is better than the best of the solutions

We could even infer Kafka cluster sizing from the # of PoPs we are enabling

PORTABLE

```
$ cp -a prd-usc1/autopilot prd-awsuse1/
$ ls prd-awsuse1/autopilot/
controllerd.yaml  telemetry-api.yaml    telemetry-sfacctd.yaml
flow-proxy.yaml   telemetry-kafka.yaml
$ git commit -am "deploy all the things!" && git push
```

Moving services across clusters with a relatively low effort

```
$ ls -d */autopilot | cat
dev-awsuse2/autopilot
prd-asse1/autopilot
prd-awsapse1/autopilot
prd-awseuw3/autopilot
prd-awsuse2/autopilot
prd-euw2/autopilot
prd-usc1/autopilot
stg-awsuse2/autopilot
```

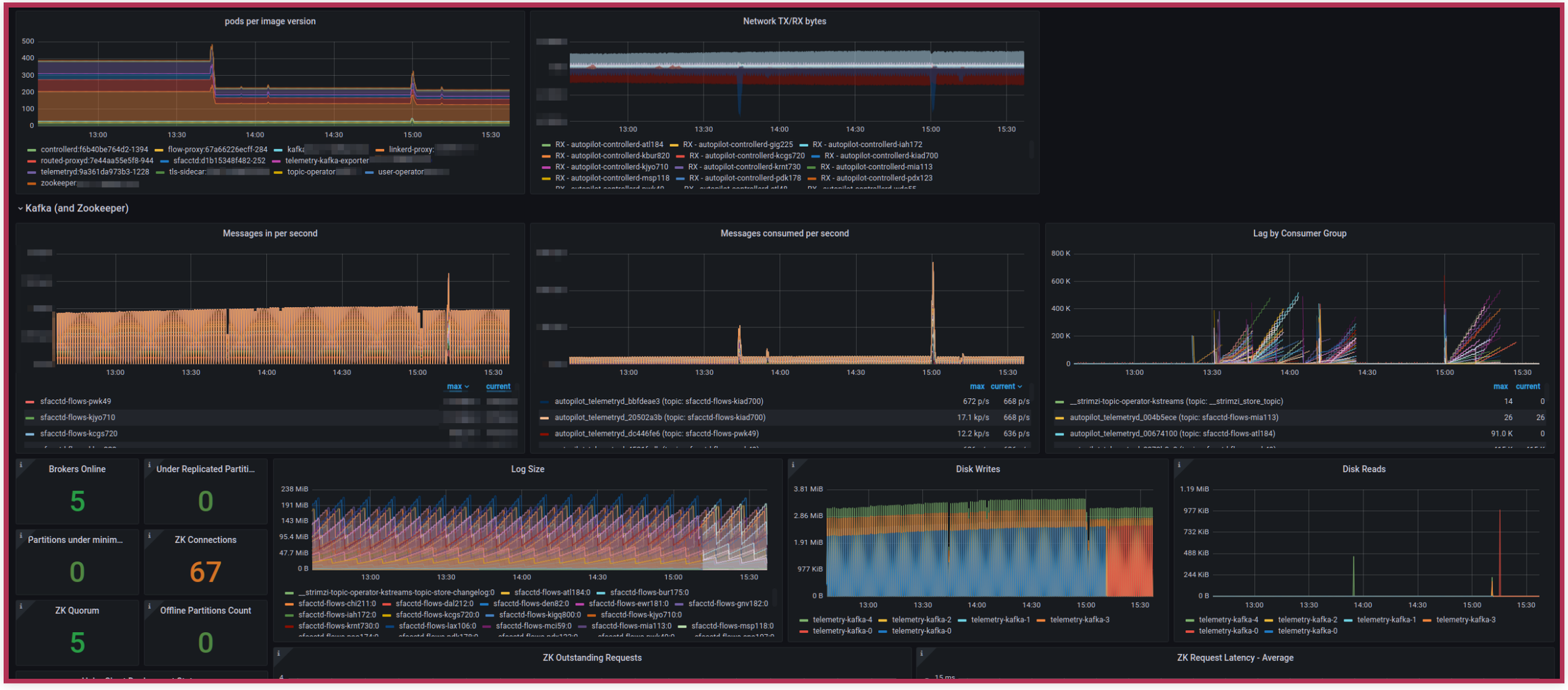
PLUGABLE

SAME TOOLS FOR DAY 2 OPERATIONS

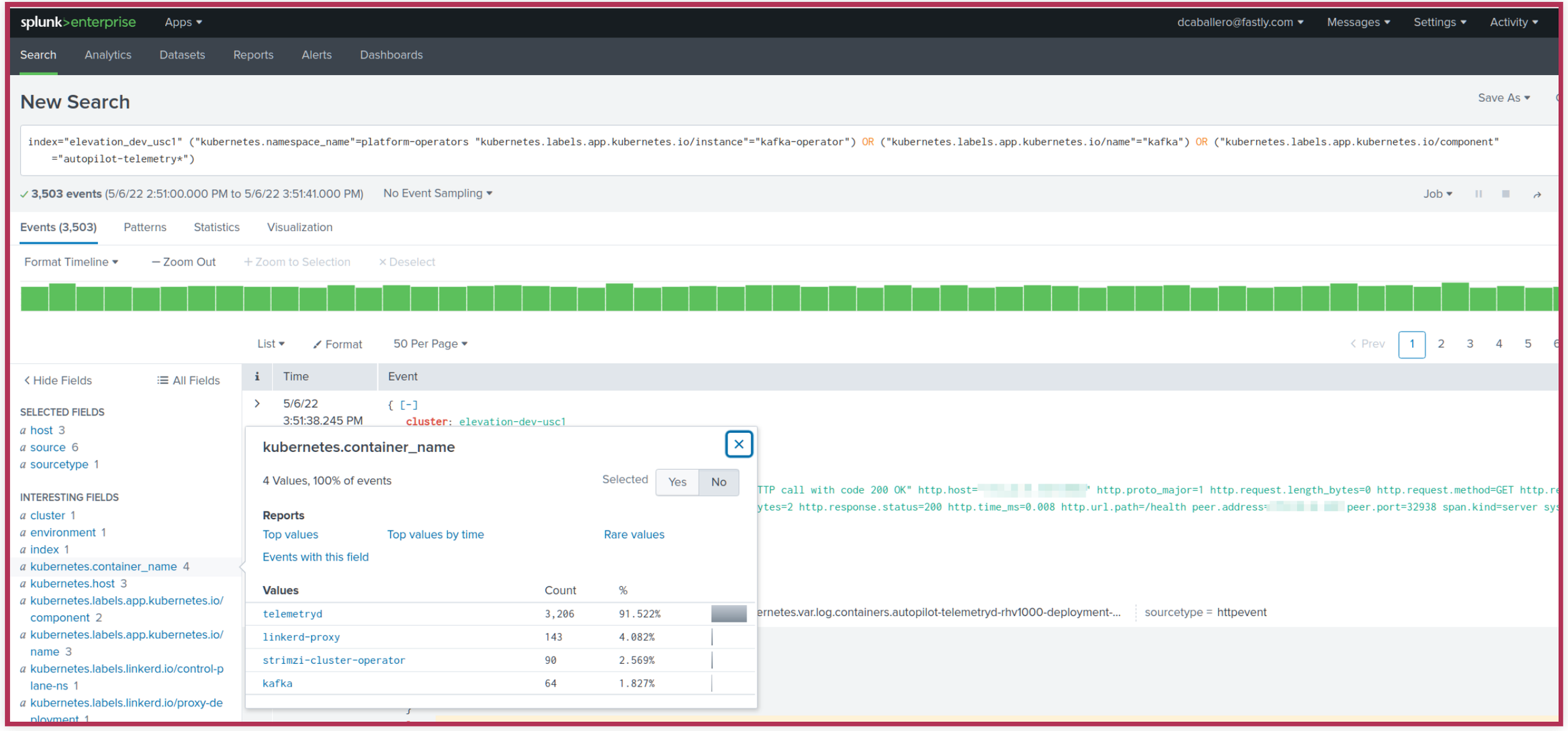
```
$ kubectl get kafkatopics
NAME                                          CLUSTER     PARTITIONS   REPLICATION FACTOR   READY
consumer-offsets---84e7a678d08f4bd226872e5cd telemetry   50           3                    True
sfacctd-flows-PoP-ID1                         telemetry   1            3                    True
sfacctd-flows-PoP-ID2                         telemetry   1            3                    True
sfacctd-flows-PoP-ID3                         telemetry   1            3                    True
sfacctd-flows-PoP-ID4                         telemetry   1            3                    True
```

```
# Other Kafka operations also mapped to K8s primitives!
$ kubectl annotate statefulset cluster-name-kafka strimzi.io/manual-rolling-update=true
$ kubectl annotate statefulset cluster-name-zookeeper strimzi.io/manual-rolling-update=true
```

CONSOLIDATED DASHBOARDS



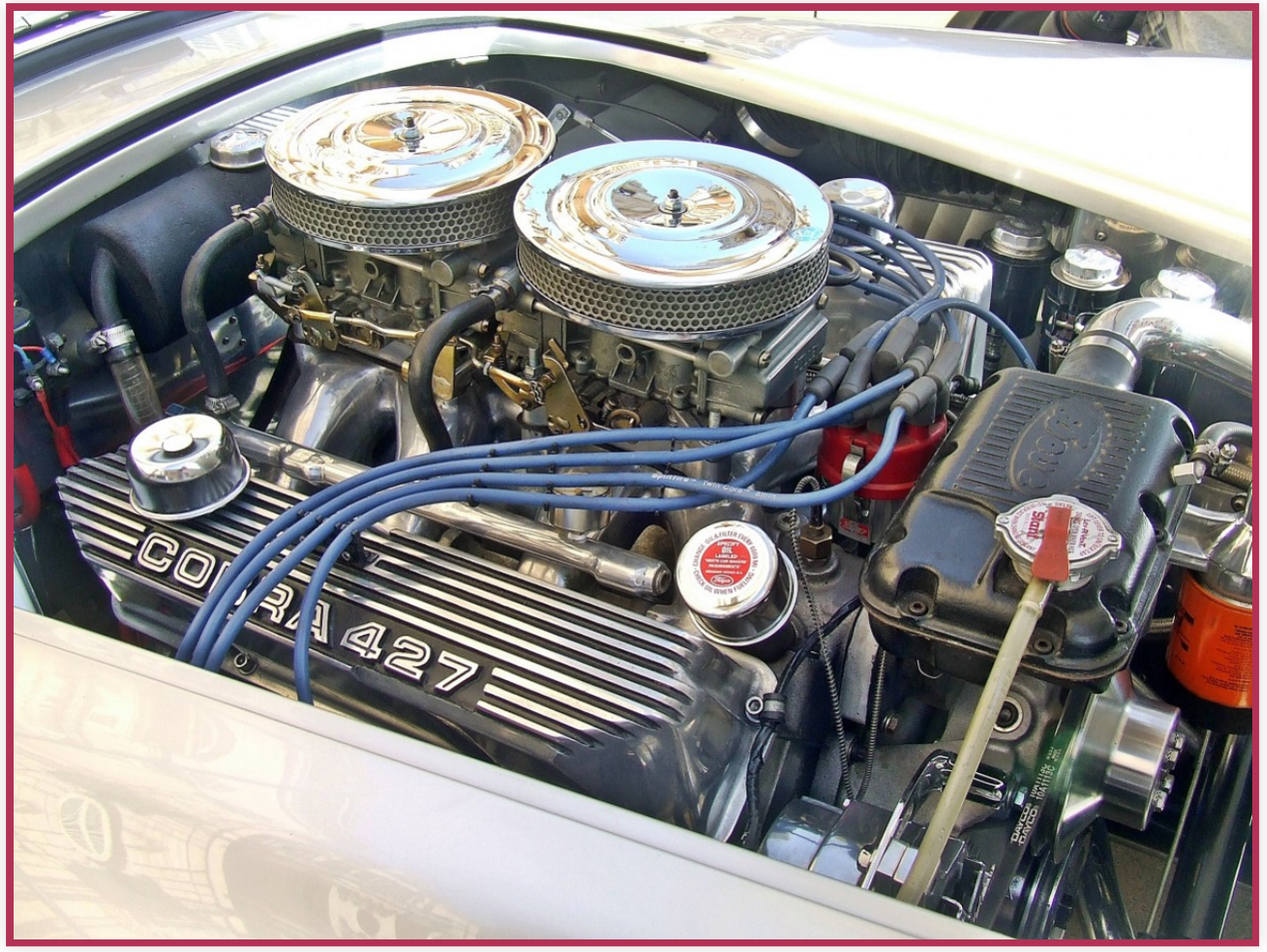
EASY EVENT CORRELATION



So Kafka, and its management, is not special

THE BAD

Keeping the operator up-to-date is an ongoing maintenance cost



KAFKA OPERATOR VS KAFKA CLUSTER OWNERSHIP

- The current ownership model requires strong collaboration between the team owning the operator and the team owning the Kafka cluster
- This might not scale well with many teams adopting this pattern

THE UGLY

Multi-cloud, service mesh, UDP, BGP…

Multi-cloud, service mesh, UDP, BGP…

UDP FLOW: STALE CONNTRACK ENTRY

- Autopilot Telemetry API receives a constant UDP flow from Fastly's switches
- We noticed that pods restarted and scheduled on the same node, will provoke UDP packets to be blackholed
- My teammate Danny Kulchinsky discovered a bug in Kube-Proxy:
  - Not flushing the relevant conntrack entry belonging to the LB ingress IP, when 0 -> 1+ endpoints
  - Flow will hit conntrack before IPTABLES nat gets applied
  - Conntrack entry will have the LB external IP not the new pod IP

AWS LB SG RULES HARD LIMIT

- AWS LB controller, creates execessive amount of NLB SG rules
  - 1 inbound rule to node's SG for client traffic per allowed client source IP
  - 1 rule for each LB subnet in the VPC for healthchecks
- ALBs, ELBs (deprecated) not an option: TCP/UDP, preserve src IP

AWS LB SG RULES HARD LIMIT (SOLUTION)

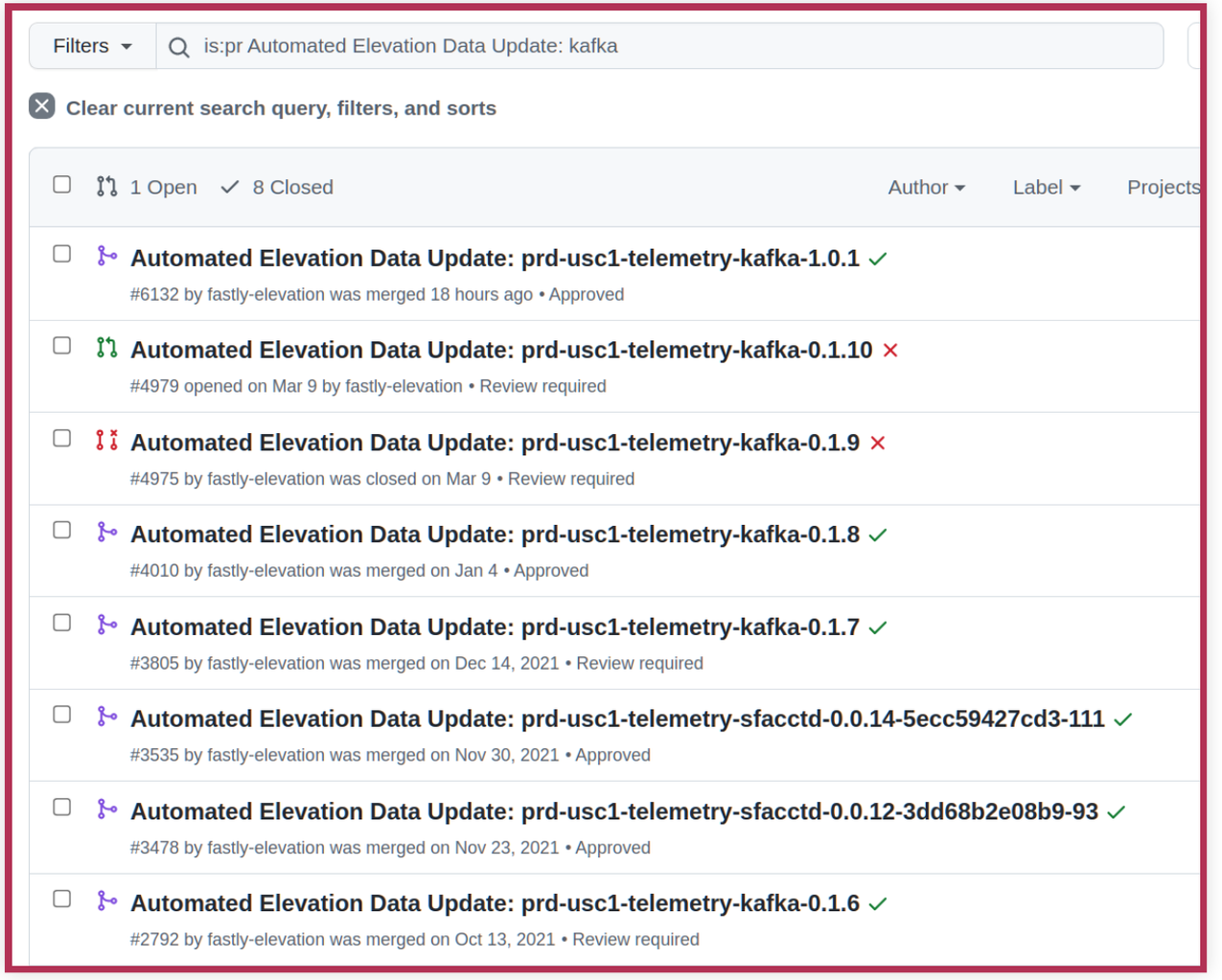
- Disable SG rules creation
- Block/Allow traffic based on Calico GNPs, dynamically generated with Kyverno
- Cleanup GNPs with custom Service Controller

IMPACT

Elevation and the Strimzi Operator had a very positive impact in Autopilot's Telemetry Pipeline development

- Even with a small team with other responsibilities…
- First version of the pipeline was developed and deployed in **4 weeks**

No significant changes were required to scale from 1 -> 10 -> ~ 100 sites



Everyone manages kafka!

master **elevation-data** / workloads / prd-usc1 / autopilot / **telemetry-kafka.yaml**

floatingstatic NCO-4384 - Decommission bur175, bwi50, chi211, jax209 autopilot compo… ···

10 contributors

69 lines (69 sloc) | 1.09 KB

```
1  apiVersion: helm.toolkit.fluxcd.io/v2beta1
2  kind: HelmRelease
3  metadata:
4    name: telemetry-kafka-prd-usc1
```

And do you remember this?

… and Autopilot is not the only use case:

- Our NetOps team also gets PoPs traffic insights
- Capacity planning
- New DDoS solutions
- Blog posts :)
- Others we may not even anticipate yet

It is becoming real

CLOSURE

SUMMARY

SUMMARY

Fastly needs an scalable and multi-* platform to run network automation

SUMMARY

Fastly needs an scalable and multi-* platform to run network automation

K8s helps Fastly to run some control plane services

SUMMARY

Fastly needs an scalable and multi-* platform to run network automation

K8s helps Fastly to run some control plane services

K8s + (kafka-)operators + elevation setup was a success for the autopilot-telemetry use case

QUESTIONS?

Thank You!


fastly