# Building and operating service mesh at mid-size company

Taiki Ono, Cookpad Inc

cookpad

# Agenda

- Background
- Problems
- Introducing and operations
- Key results
- Next challenges

# Backgroud

# Our business

- "Make everyday cooking fun!"
- Originally started in Japan in 1997
- Operate in over 23 languages, 68 countries
- World largest recipe sharing site: **cookpad.com**

# Our scale and organization structure

- 90M monthly average user
- ~200 product developers
- ~100 production services
- 3 platform team members
    - 1 for service mesh dev

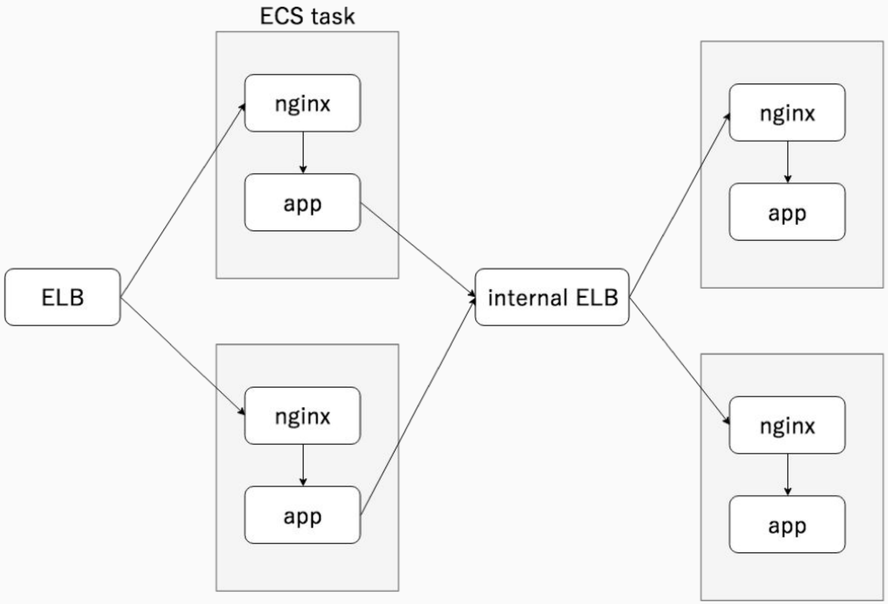
Each product team owns their products but all of operations are still owned by central SRE team

SRE team

Product team

# Technology stack

- Ruby on Rails for both web frontend and backend apps
- Python for ML apps
- Go, Rust for backend apps
- Java for new apps
- Other languages for internal apps



Frontend app

Backend app

# Problems

# Operational problems

- Decrease in system reliability
- Hard to troubleshoot and debug distributed services
    - Increase of time detect root causes of incidents
    - Capacity planing

# Library approach solutions

- github.com/cookpad/expeditor
  - Ruby library inspired by Netflix's Hystrix
  - Parallel executions, timeouts, retries, cirbuit breakers
- github.com/cookpad/aws-xray
  - Ruby library for distributed tracing using AWS's X-Ray service

# GoPythonJava apps?

- Limitation of library model approach
    - Save resources for product development
- Controlling library versions is hard in a large organization
- Planning to develop our proxy and mixed with consul-template

# "Lyft's Envoy: Experiences Operating a Large Service Mesh"

at SRECON America 2017 (March)

# Introducing our service mesh

# Timeline

- Early 2017: making plan
- Late 2017: building MVP
- Early 2018: generally available

# In-house control-plane

- Early 2017: no Istio
- We are using Amazon ECS
- Not to use full features of Envoy
  - Resiliency and observability parts only
- Small start with in-house control-plane, but planned to migrate to future managed services.

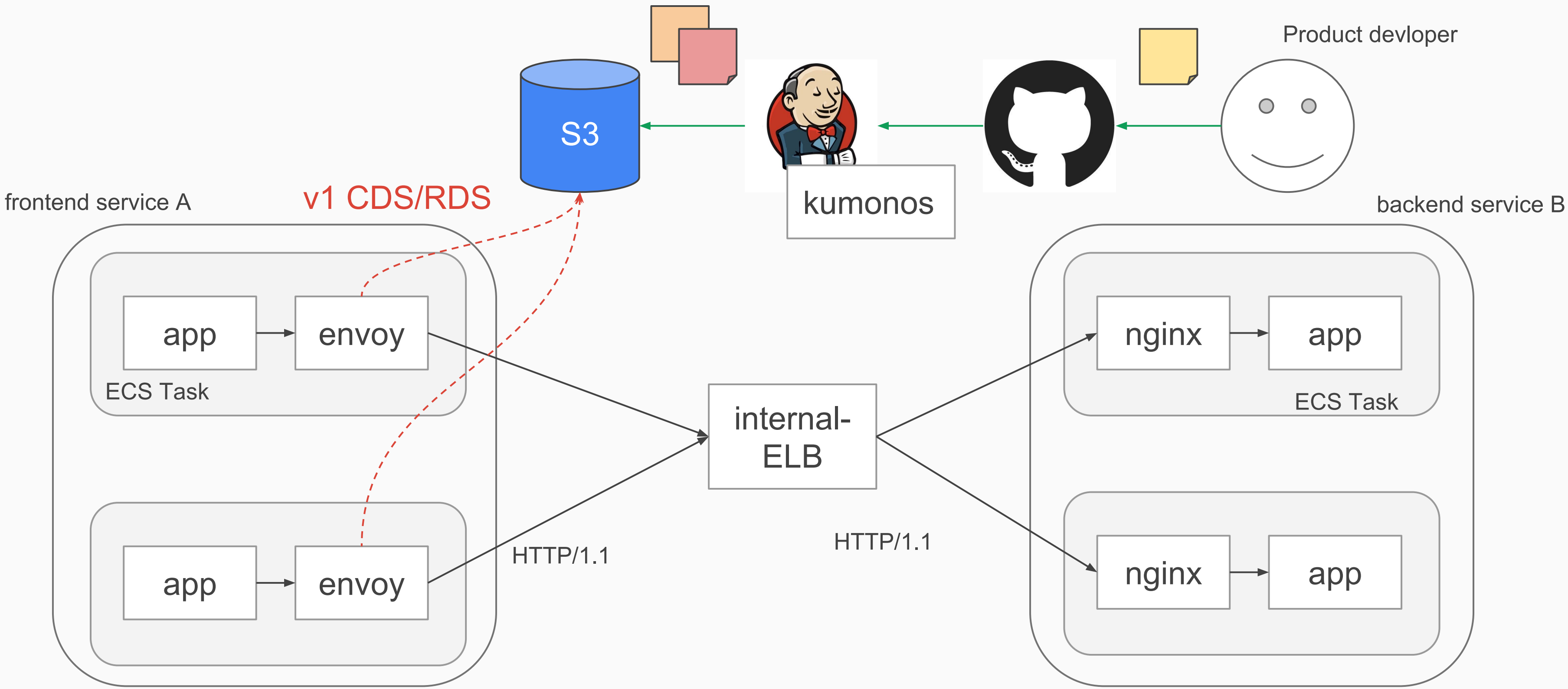
# Considerations

- Everyone can view and manage resiliency settings
  - Centrally managed
  - GitOps with code reviews
- All metrics should go into Prometheus
- Low operation cost
  - Less components, use of managed services

# Our service mesh components

- **kumonos** (github.com/cookpad/kumonos)
  - v1 xDS response generator
- **sds** (github.com/cookpad/sds)
  - Fork of github.com/lyft/discovery to allow multiple service instances on the same IP address
  - Implements v2 EDS API
- **itacho** (github.com/cookpad/itacho)
  - v2 xDS response generator (CLI tool)
  - v2 xDS REST HTTP POST-GET translation server
    - GitHub#4526 “REST xDS API with HTTP GET requests”

# v1 ELB based with v1 xDS



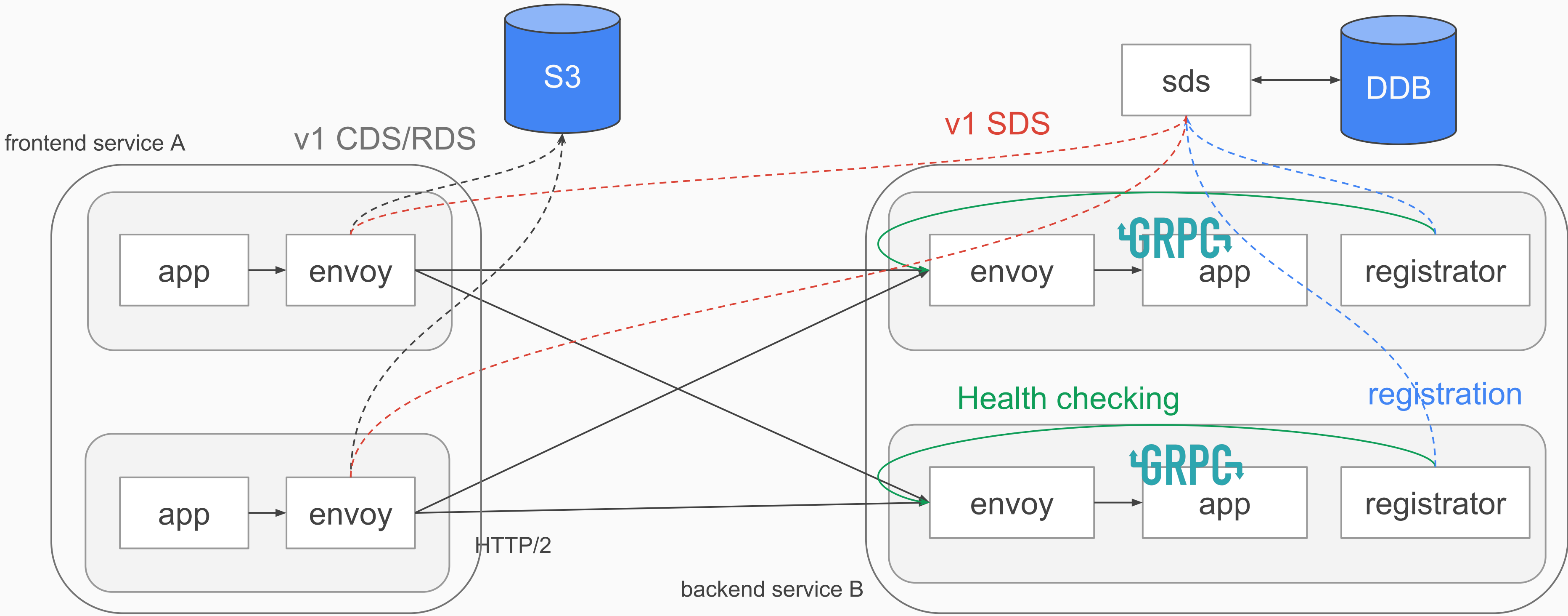
# Configuration file

- Single Jsonnet file represents single service configuration
  - 1 service - N upstream dependencies
- Route config
  - Retry, timeouts for paths, domains
  - Auto retry with GET,HEAD routes
- Cluster config
  - DNS name of internal ELB
  - Port, TLS, connect timeout
  - Circuit breaker settings

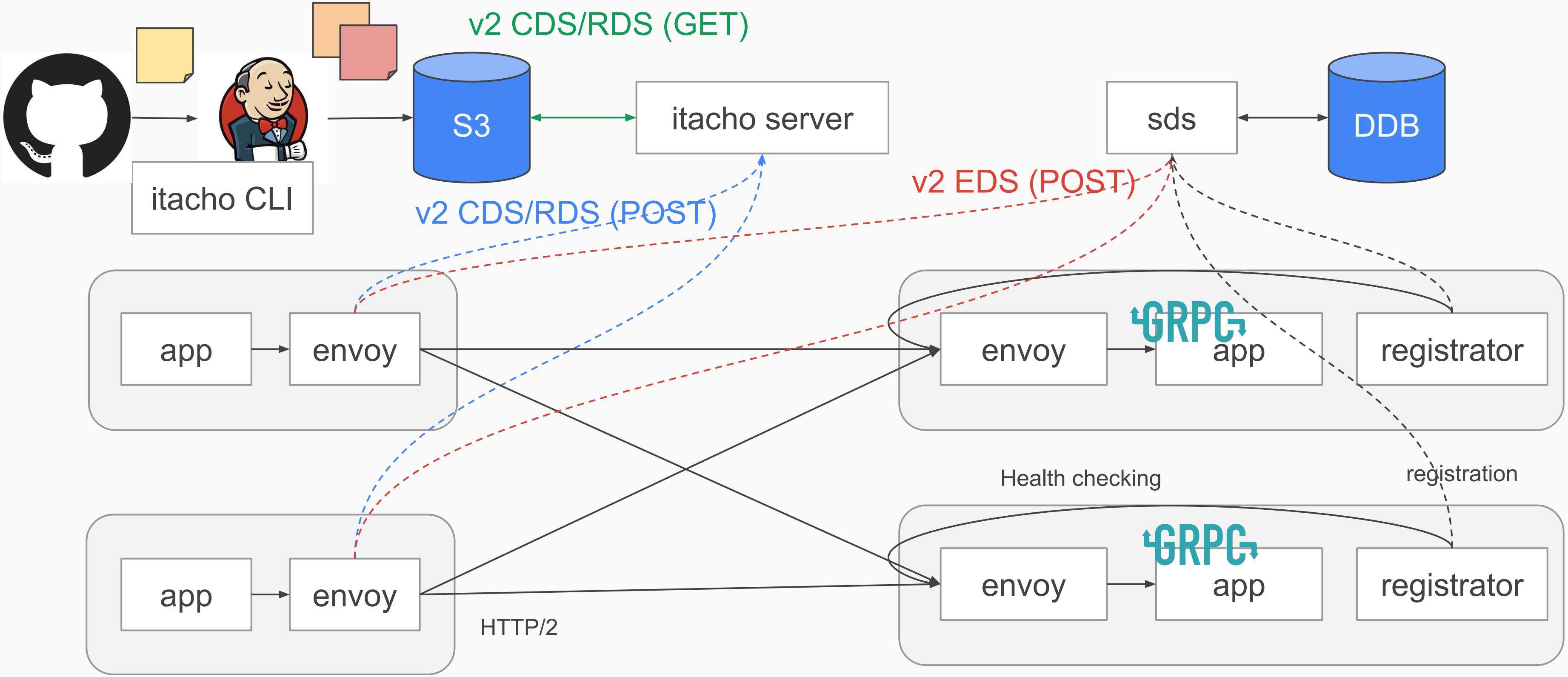
```
local circuit_breaker = import 'circuit_breaker.libsonnet';
local routes = import 'routes.libsonnet';

{
    version: 1,
    dependencies: [
        {
            name: "user",
            cluster_name: "user-development",
            lb: "user-service.example.com:80",
            tls: false,
            connect_timeout_ms: 250,
            circuit_breaker: circuit_breaker.default,
            routes: [routes.default],
        },
    ],
}
```

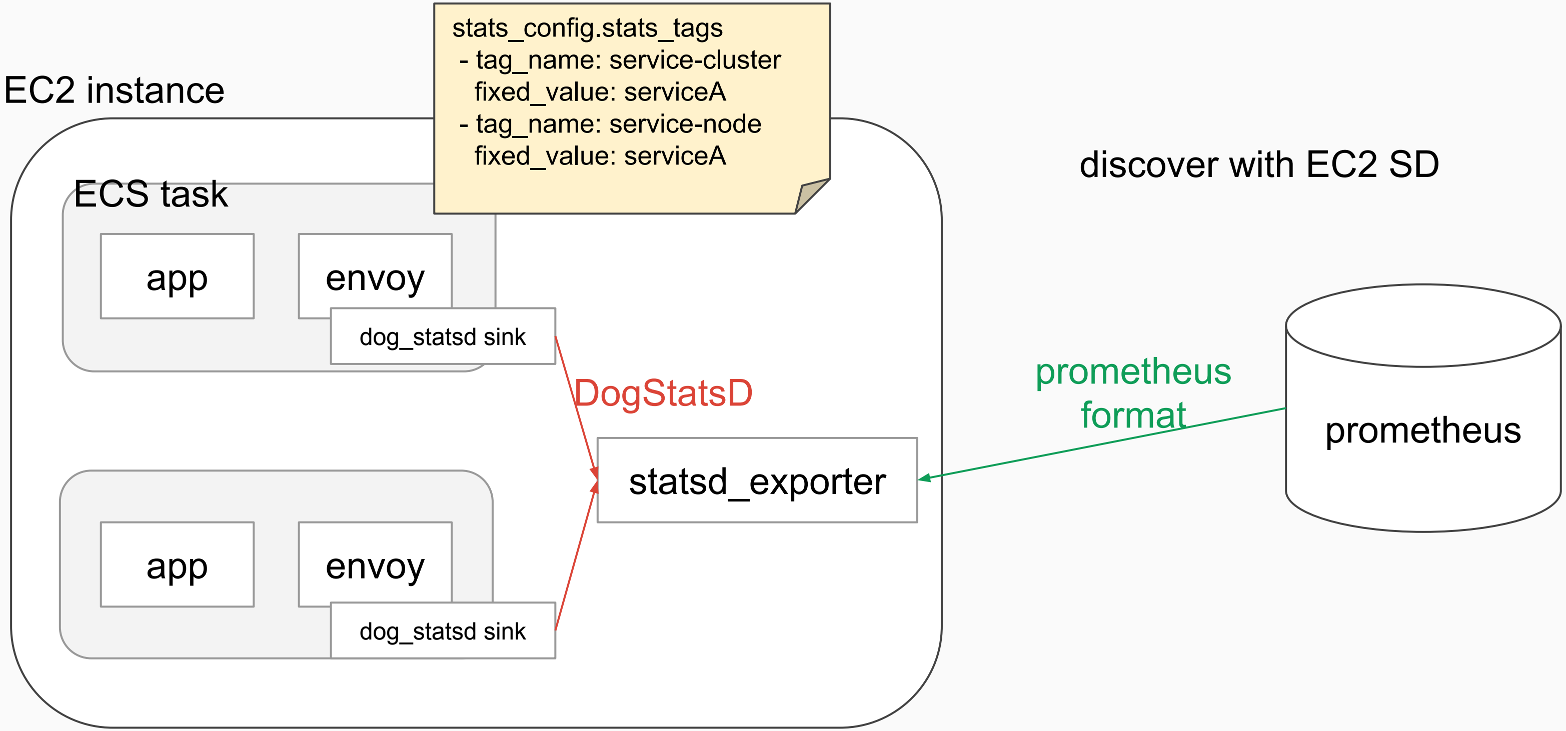
# v1.1 with v1 SDS for backend gRPC apps



# v2 with v2 xDS



# Sending metrics



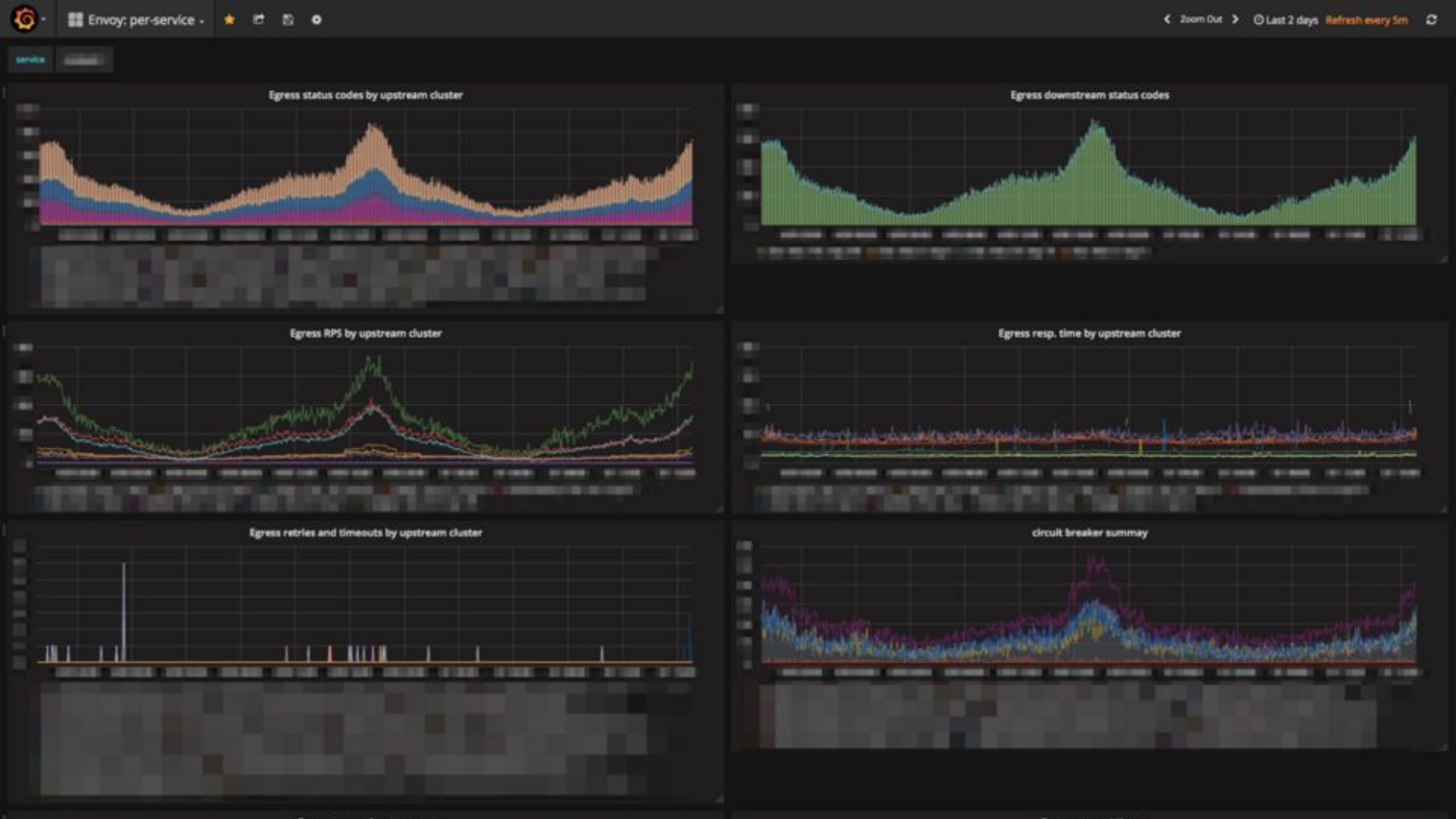
```
stats_config.stats_tags
 - tag_name: service-cluster
   fixed_value: serviceA
 - tag_name: service-node
   fixed_value: serviceA
```

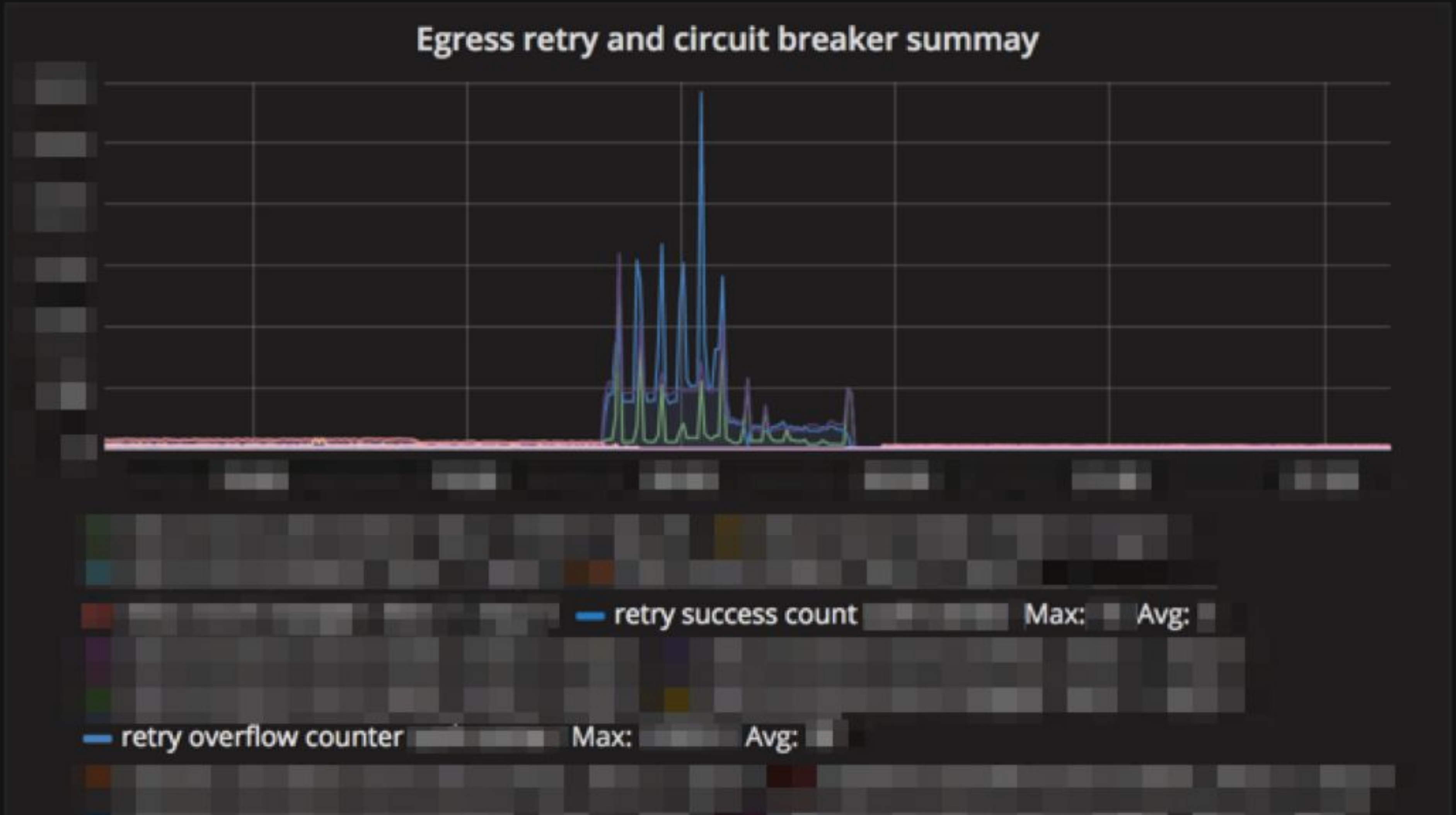
discover with EC2 SD

# Operation side

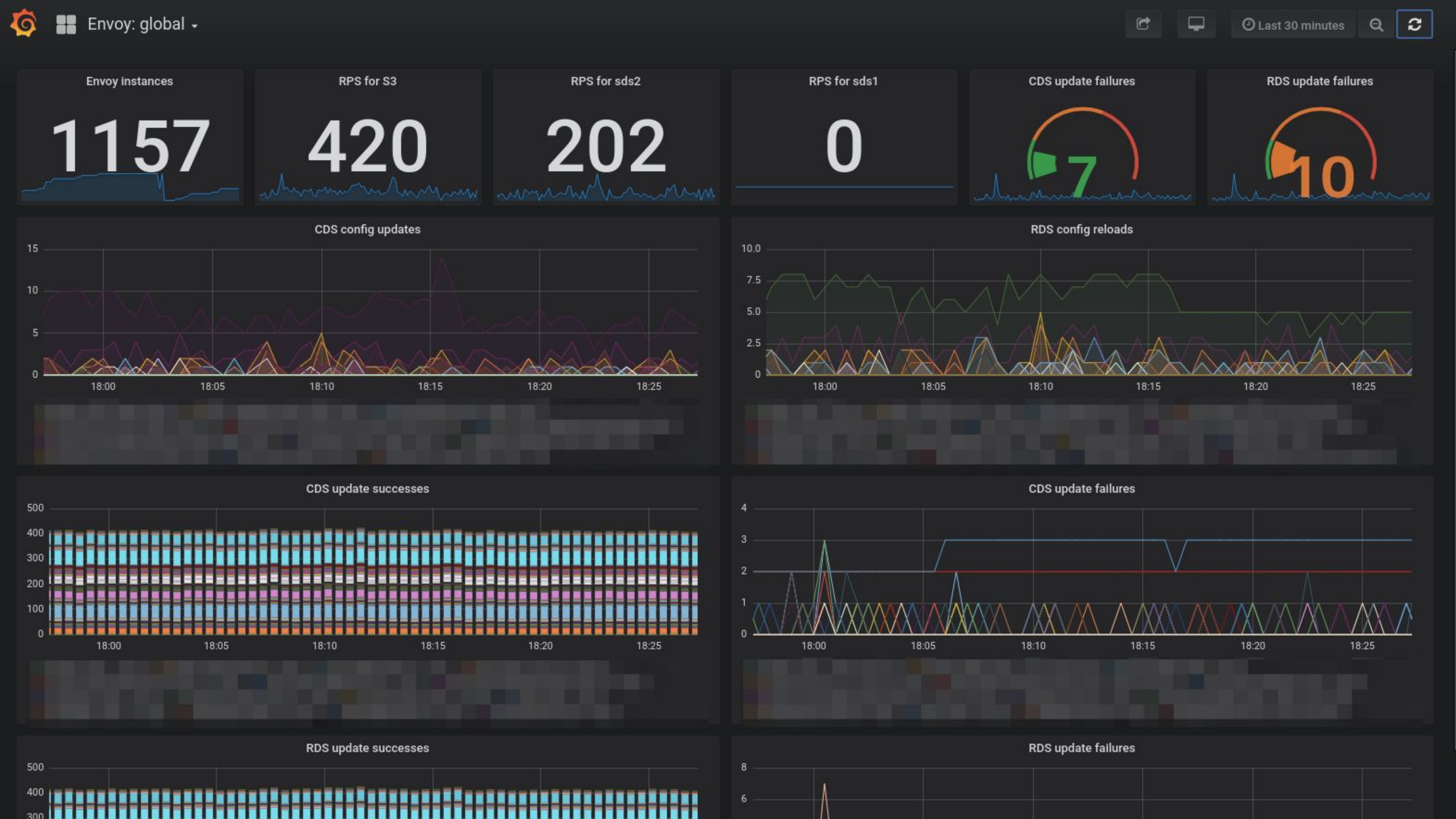
# Dashboards

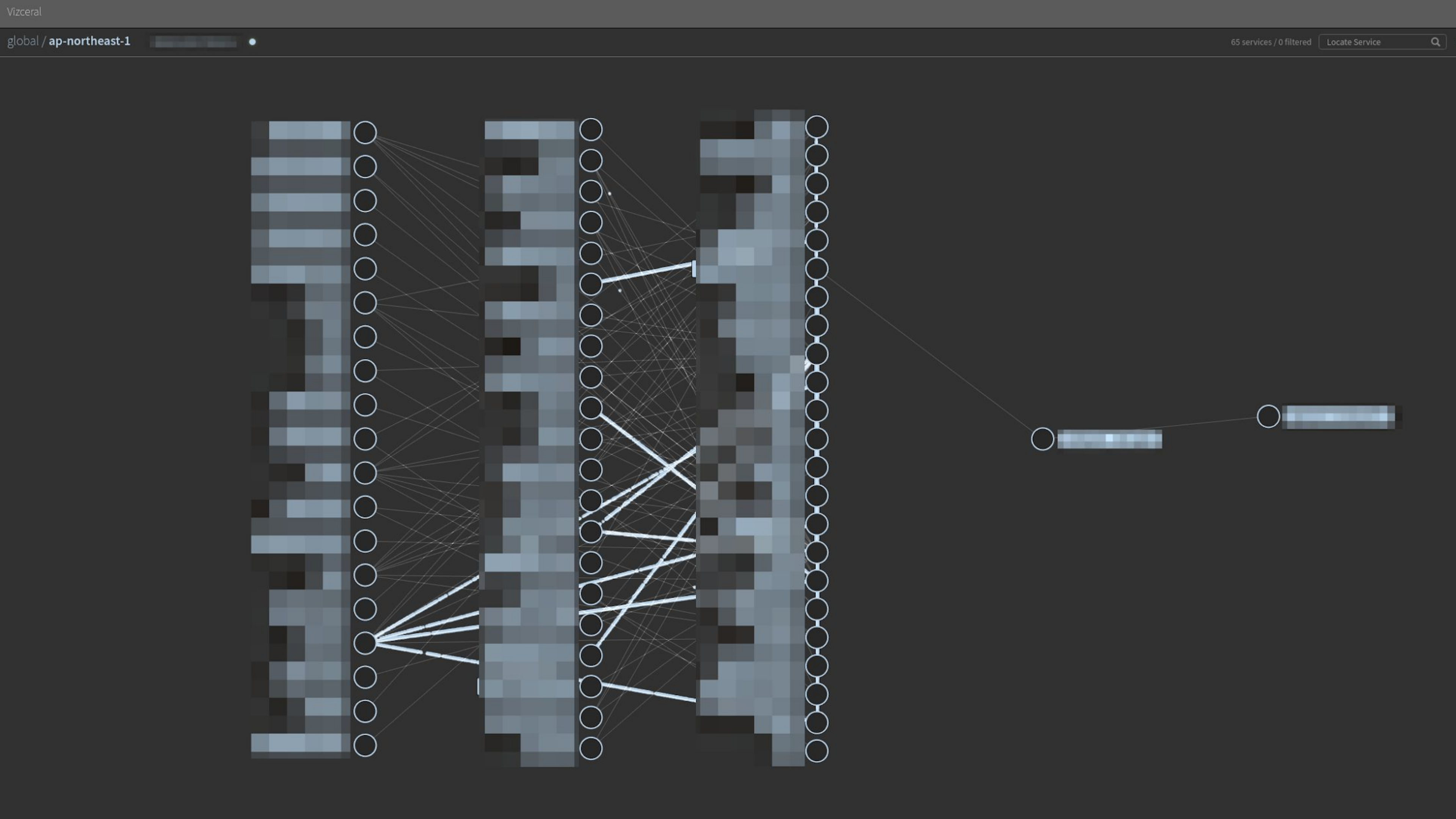
- Grafana
  - Per service (1 downstream - N upstreams)
  - Per service-to-service (1 downstream - 1 upstream)
  - Envoy instances
- Netflix’s Vizceral
  - github.com/nghialv/promviz
  - github.com/mjhd-devlion/promviz-front


Envoy: per-service
Zoom Out
Last 2 days
Refresh every 5m
service
Egress status codes by upstream cluster
Egress downstream status codes
Egress RPS by upstream cluster
Egress resp. time by upstream cluster
Egress retries and timeouts by upstream cluster
circuit breaker summay



Egress retry and circuit breaker summay

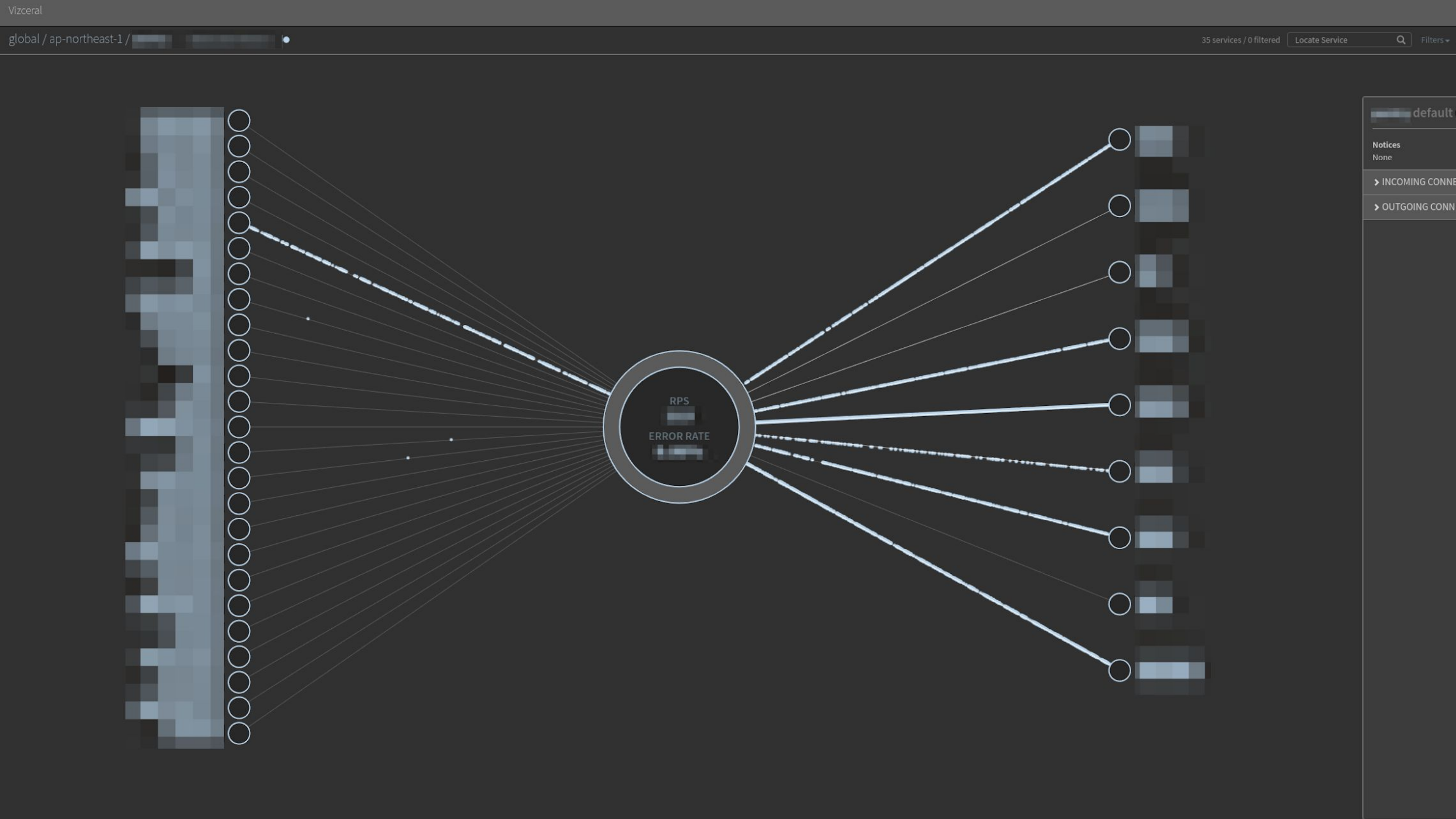

Envoy: global
Last 30 minutes
Envoy instances
1157
RPS for S3
420
RPS for sds2
202
RPS for sds1
0
CDS update failures
7
RDS update failures
10
CDS config updates
15
10
5
0
18:00
18:05
18:10
18:15
18:20
18:25
[redacted]
RDS config reloads
10.0
7.5
5.0
2.5
0
18:00
18:05
18:10
18:15
18:20
18:25
[redacted]
CDS update successes
500
400
300
200
100
0
18:00
18:05
18:10
18:15
18:20
18:25
[redacted]
CDS update failures
4
3
2
1
0
18:00
18:05
18:10
18:15
18:20
18:25
[redacted]
RDS update successes
500
400
300
RDS update failures
8
6



Vizceral

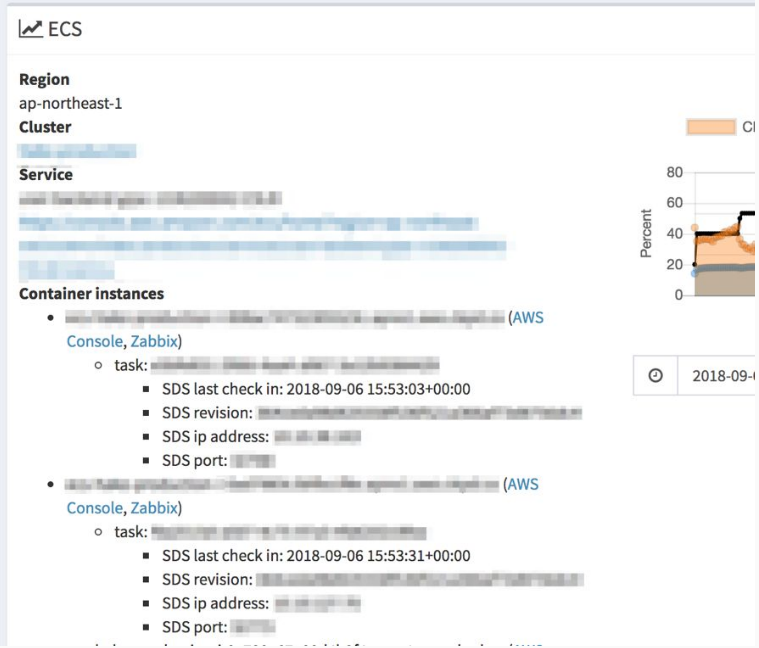
global / ap-northeast-1 [redacted]

65 services / 0 filtered


Vizceral
global / ap-northeast-1 / [redacted]
35 services / 0 filtered
Locate Service
Filters ▾
[redacted] default
Notices
None
› INCOMING CONNE
› OUTGOING CONN
RPS
[redacted]
ERROR RATE
[redacted]

# Integrate with in-house platform console

- Show service dependencies
- Link to service mesh config file
- Show SDS/EDS registration
- Link to Grafana dashboards
  - Per service dashboard
  - Service-to-service dashboard



ECS

**Region**

ap-northeast-1

**Cluster**

[redacted]

**Service**

[redacted]

**Container instances**

- [redacted] (AWS Console, Zabbix)
  - task: [redacted]
    - SDS last check in: 2018-09-06 15:53:03+00:00
    - SDS revision: [redacted]
    - SDS ip address: [redacted]
    - SDS port: [redacted]
- [redacted] (AWS Console, Zabbix)
  - task: [redacted]
    - SDS last check in: 2018-09-06 15:53:31+00:00
    - SDS revision: [redacted]
    - SDS ip address: [redacted]
    - SDS port: [redacted]

# Envoy on EC2 instances

- Build and distribute as a in-house deb package
  - Setup instances with configuration management tool like Chef: github.com/itamae-kitchen/itamae
- Manage Envoy process as a systemd service
- Using hot-restarter.py
  - Generate starter script for each host role

# Wait initial xDS fetching

- Sidecar Envoy containers need a few seconds to be up
  - Background jobs are service-in quickly
  - ECS does not have an API to wait the initializing phase
- Wrapper command-line tool
  - github.com/cookpad/wait-side-car
  - Wait until an upstream health check succeed
- Probably move to GitHub#4405

# The hard points

- Limitation of ECS and its API
  - Without ELB integration, we need to manage lots of things on deployments.
  - We needed AWS Cloud Map (actually we made almost the same thing in our environment).

# Key results

# Observability

- Both SRE and product team have confidence in what’s happened in service-to-service communication area
  - Visualization of how resiliency  mechanism is working
  - Decrease of time to detect root causes around service communication issues
- Necessary to encourage collaboration between multiple product teams

# Failure recovery

- Be able to configure proper resiliency setting values with fine-grained metrics
- Eliminates temporal burst of errors from backend services
- Fault isolation: not yet remarkable result

# Continuous development of app platform

- Improve application platform without product application deployment
- Increase velocity of platform development team

# Next challenges

# Next challenges

- **Fault injection platform**
- Distributed tracing
- Auth{z, n}
- More flexibility on traffic control
  - Envoy on edge proxies?
- **Migration to managed services**

# Q&A

- Twitter hashtag #EnvoyCon, @taiki45
- Published this slide at: envoyconna18.sched.com