

The Cisco Live! logo features the word "CISCO" in a dark blue, sans-serif font, followed by "Live!" in a dark blue, cursive script font. The background of the entire image is a vibrant, multi-colored abstract pattern of overlapping, wavy bands and a radial sunburst effect on the right side, transitioning from dark blue on the left to bright yellow and white in the center-right.

CISCO *Live!*

Let's go



The bridge to possible

# Unlock Intersight monitoring with OpenTelemetry

Chris Gascoigne, Principal Architect  
[@chrisgascoigne@hachyderm.io](mailto:chrisgascoigne@hachyderm.io)

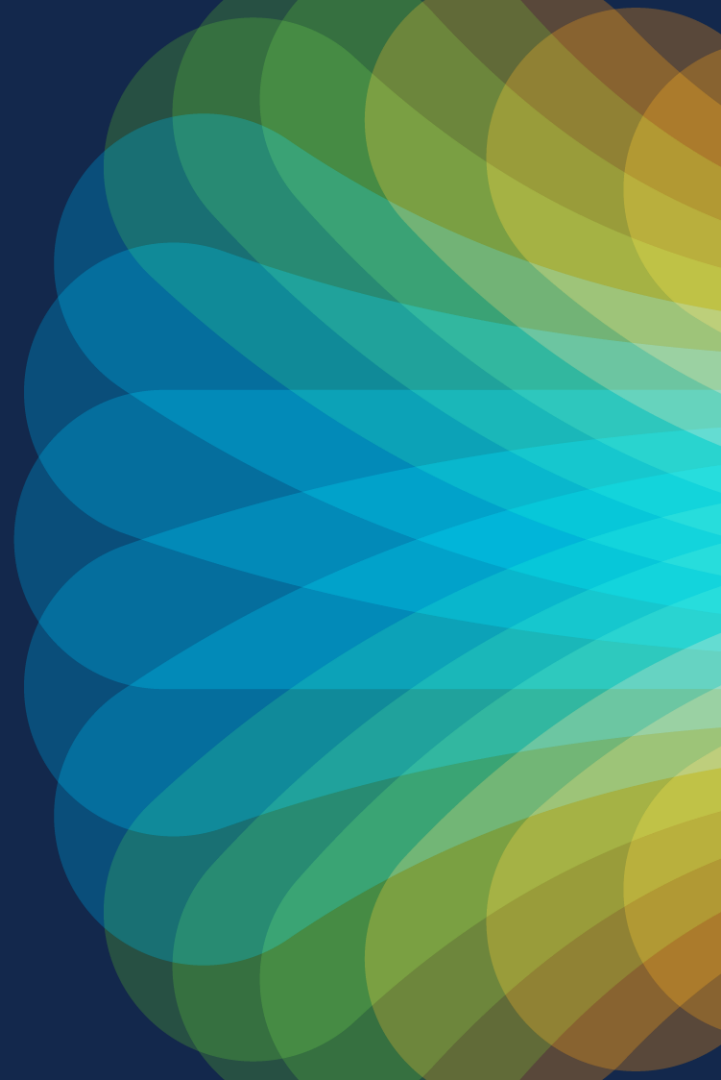
CISCO *Live!*

DEVNET-2118

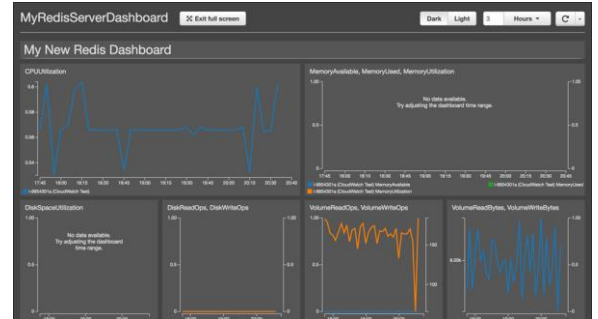
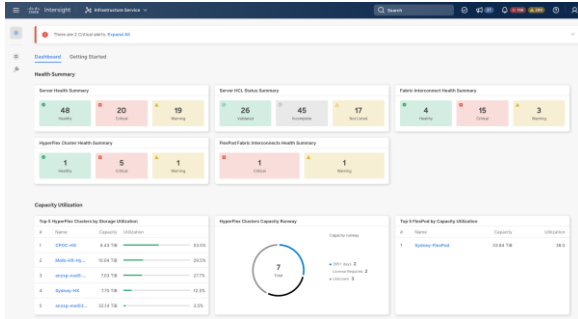
# Agenda

- The Challenge
- Enter OpenTelemetry
- Intersight and OpenTelemetry?
- Key Takeaways

# The Challenge



# Silo'd data stores and dashboards



Infrastructure team

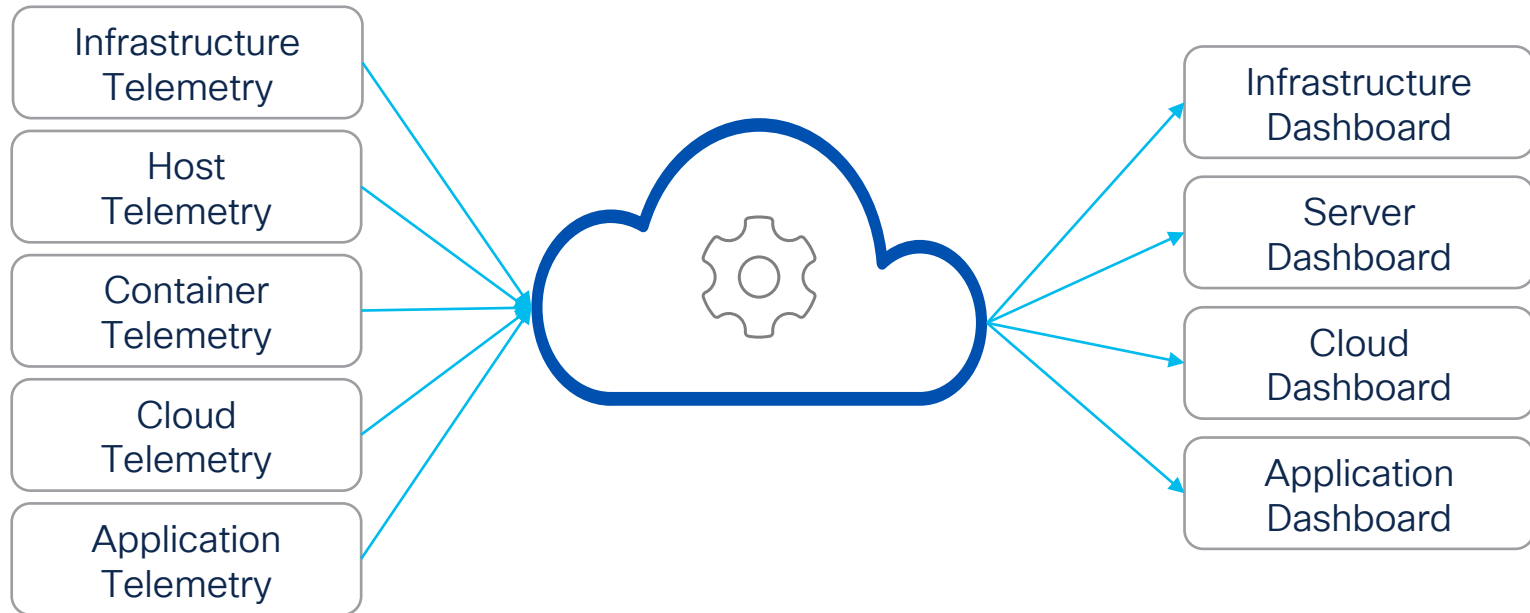


Virtualization / Systems team



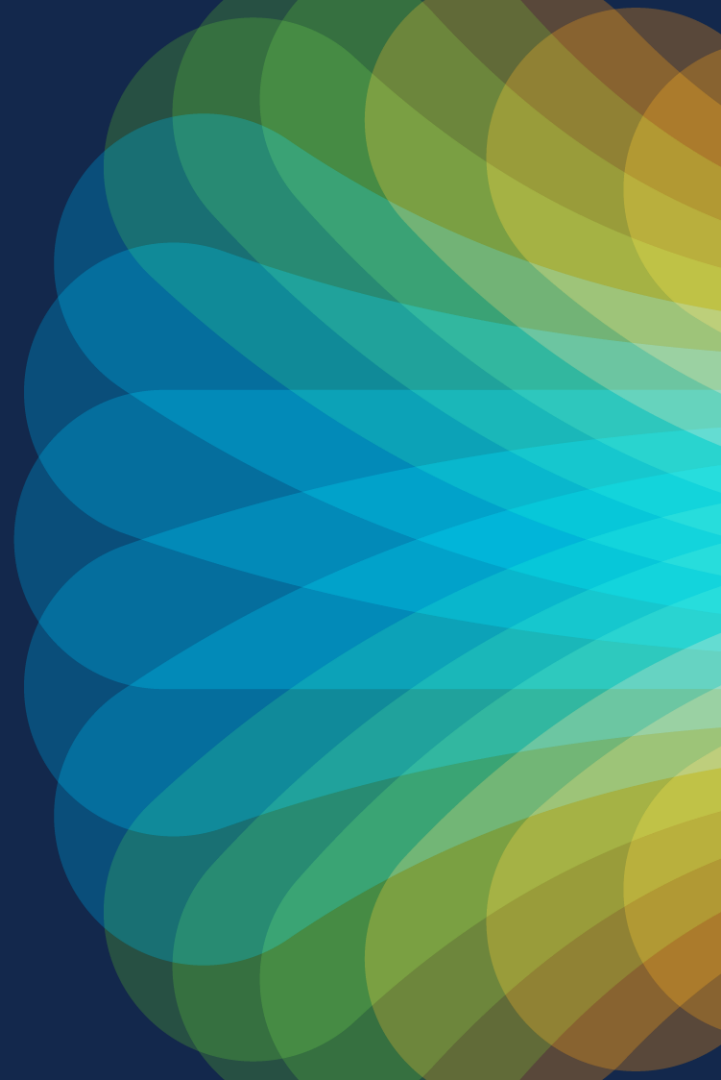
Cloud team

# What if there was a way to democratize that data?





# Enter OpenTelemetry



# What is OpenTelemetry?

*“OpenTelemetry is a collection of tools, APIs, and SDKs. Use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) to help you analyze your software’s performance and behavior.”*

# OpenTelemetry Signal Types

## Metrics

a set of measurements collected at regular intervals

## Logs

strings of structured or unstructured text with an associated timestamp

## Traces

chains of events (or transactions) between different components in an application

# Relevant OpenTelemetry Terminology

API

In the OpenTelemetry project, used to define how telemetry data is generated

SDK

A telemetry SDK implements the OpenTelemetry API in a specific language

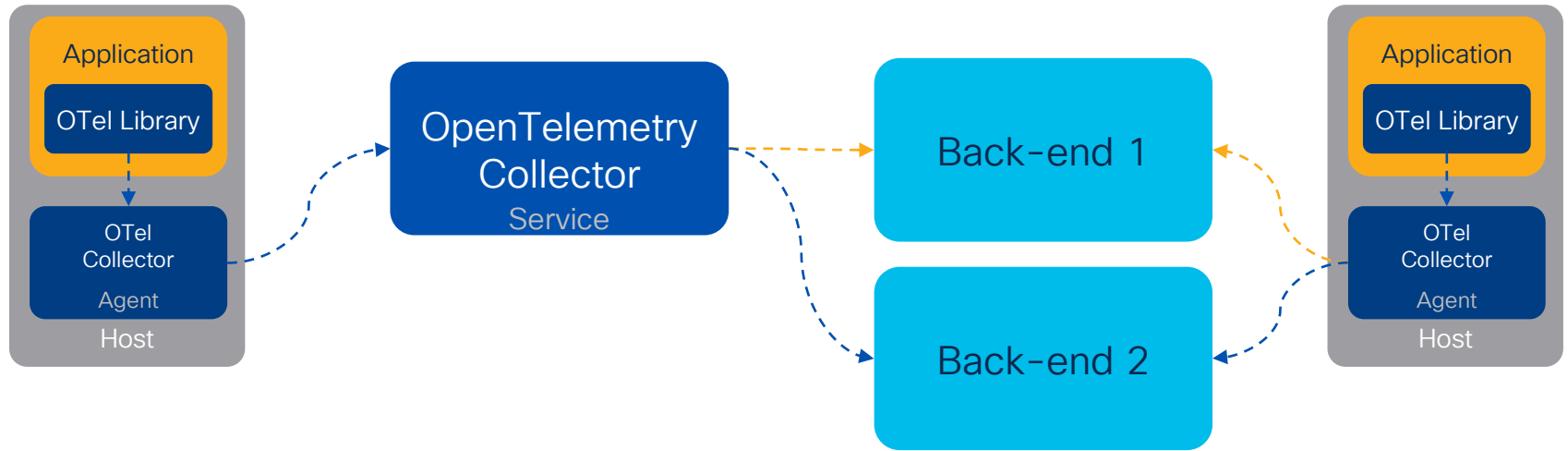
Semantic  
Conventions

Defines standard names and values of Metadata in order to provide vendor-agnostic telemetry data.

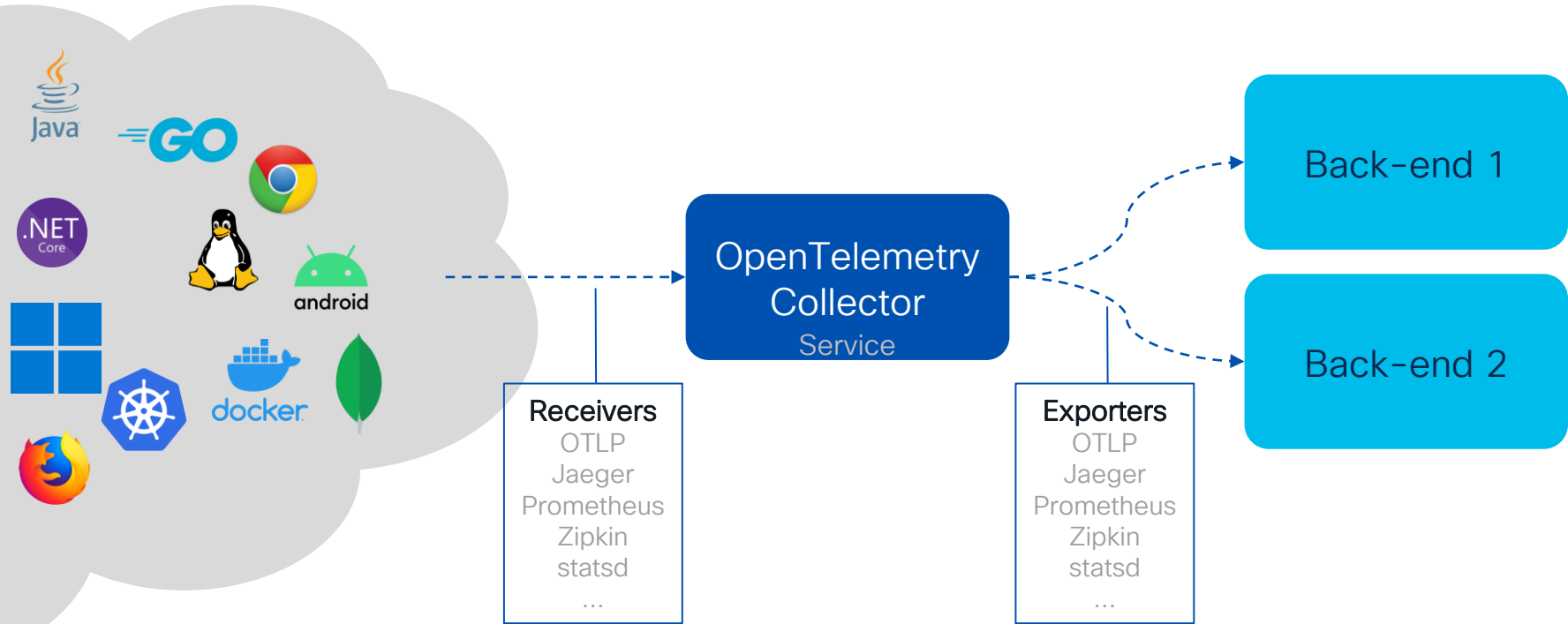
OTLP

OpenTelemetry Protocol (OTLP) defines the protocol used to exchange data between the client and the server over gRPC or HTTP

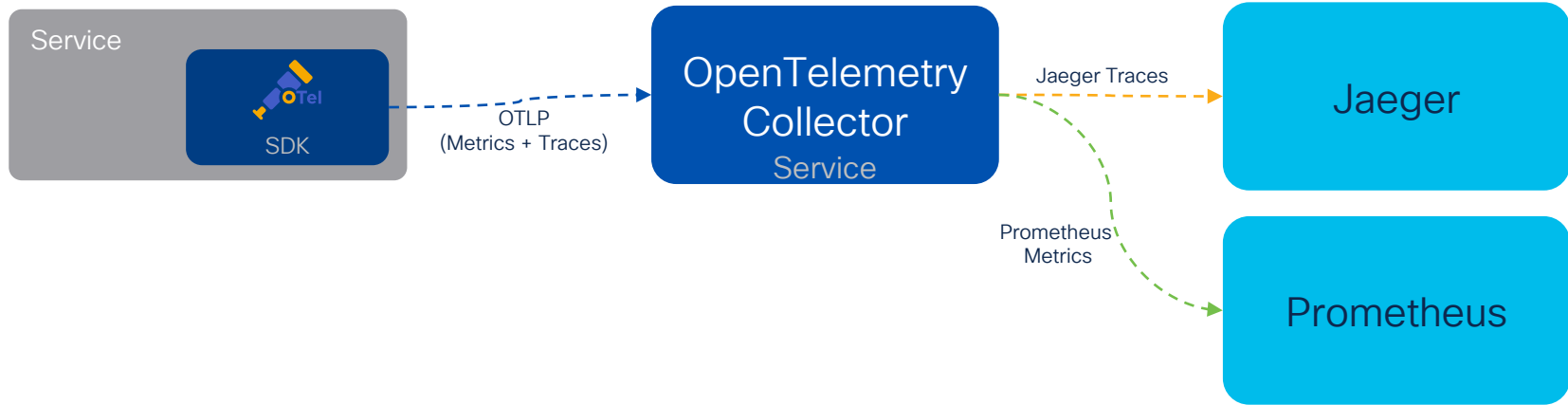
# OpenTelemetry reference Architecture



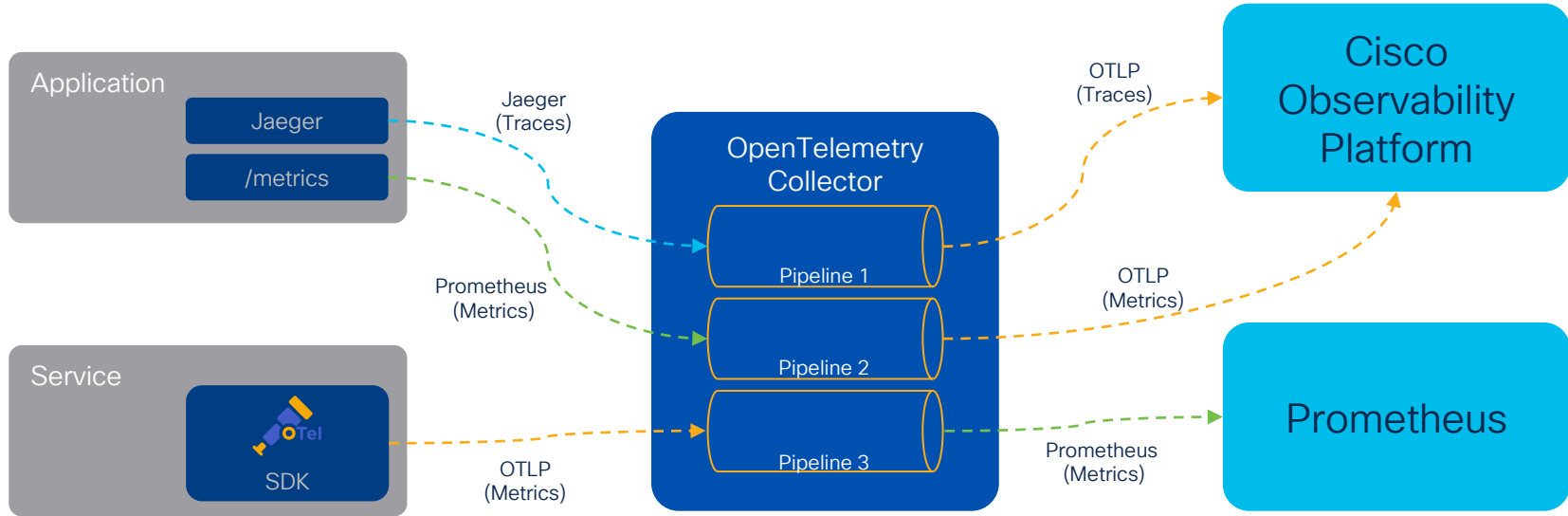
# OpenTelemetry Collector as protocol translator



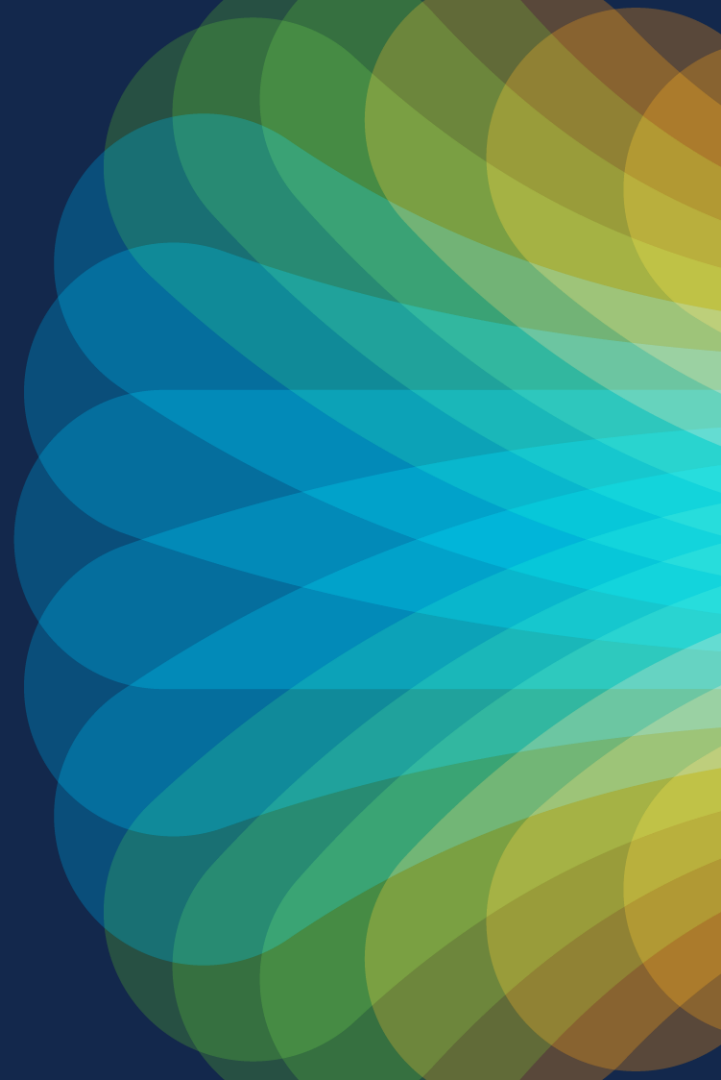
# Collector Use Cases (1)



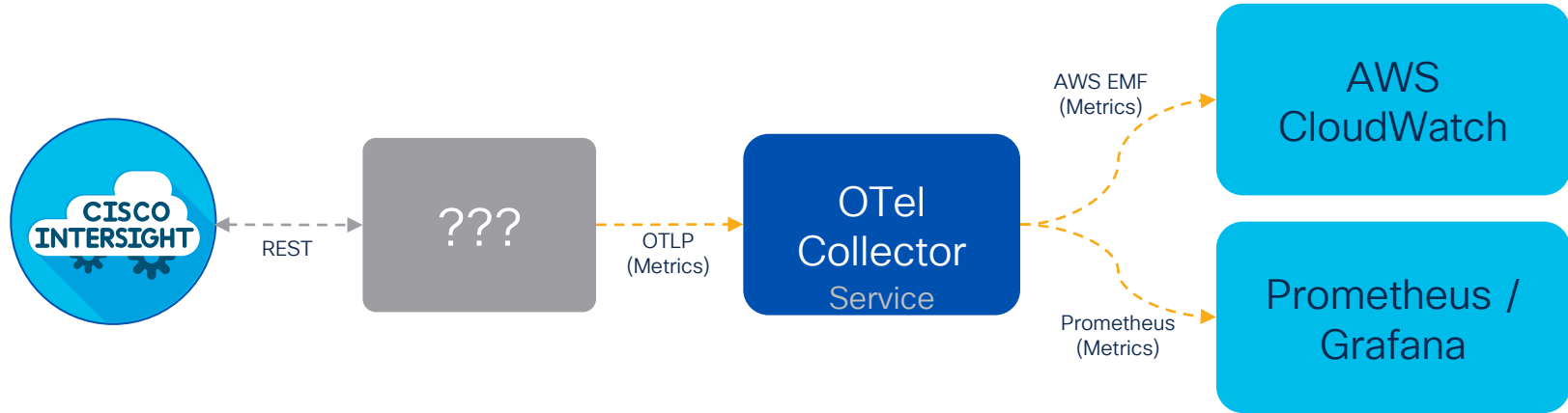
# Collector Use Cases (2)



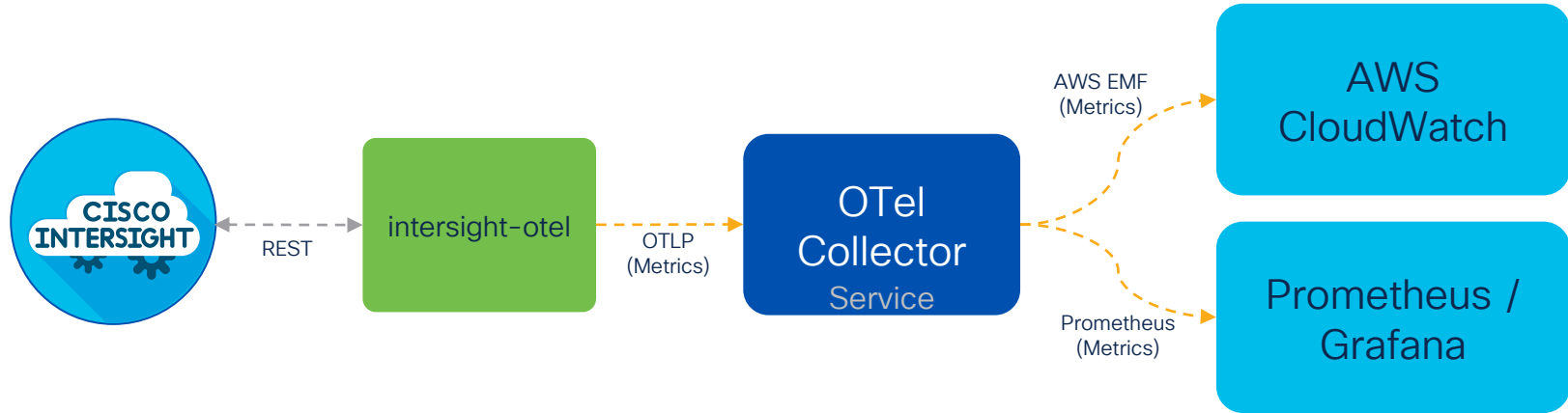
# Intersight and OpenTelemetry?

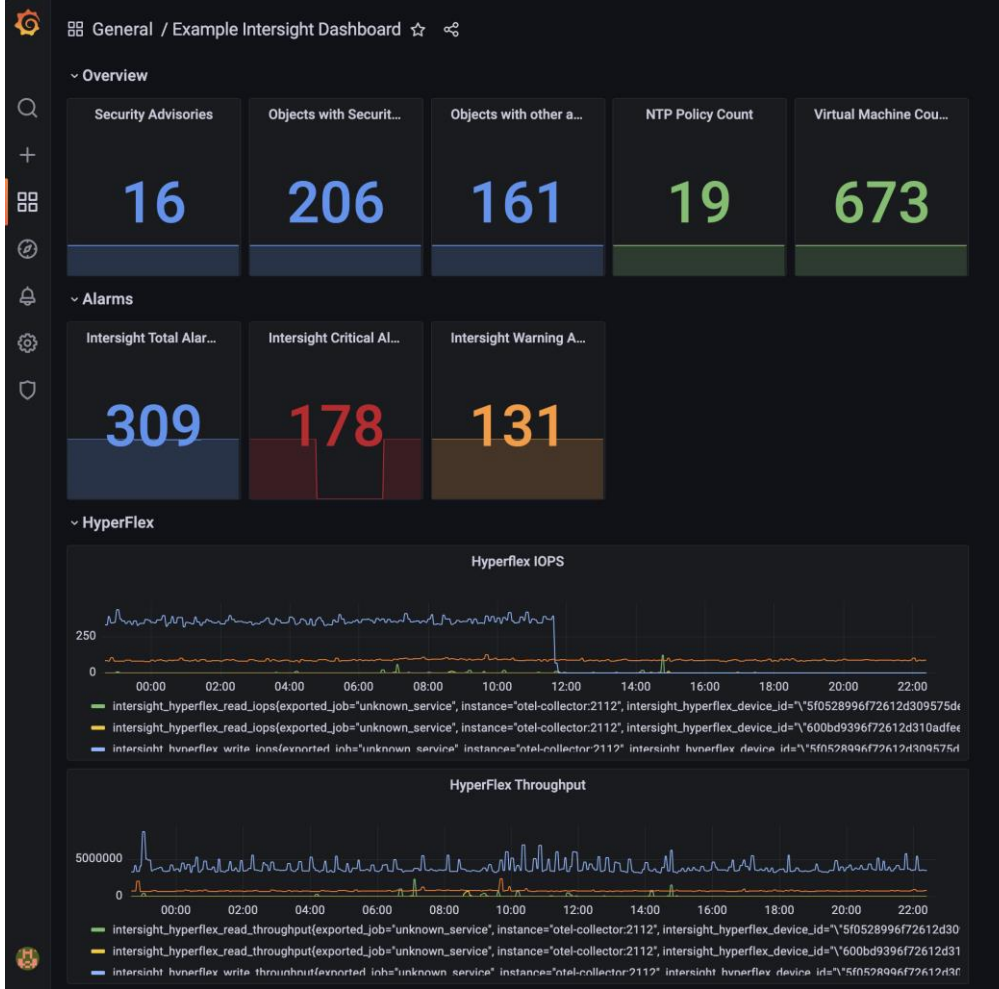


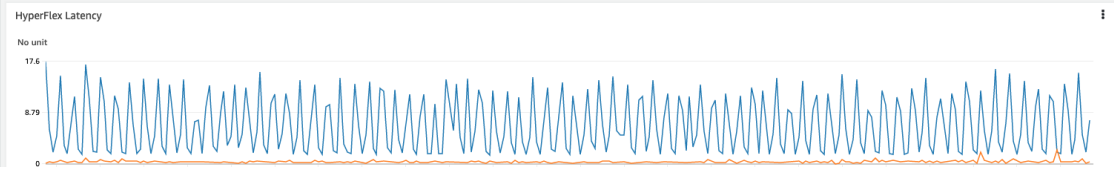
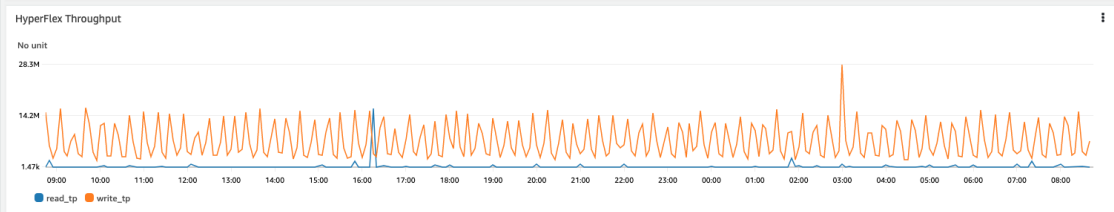
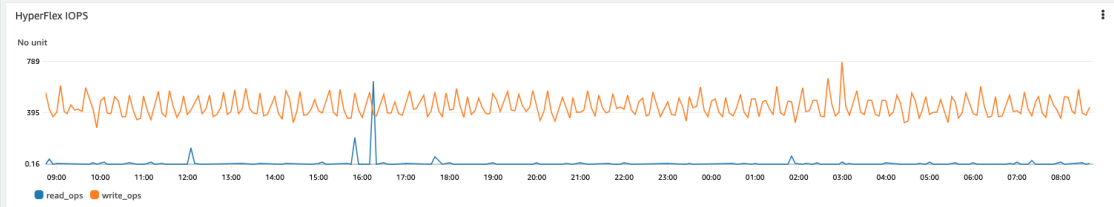
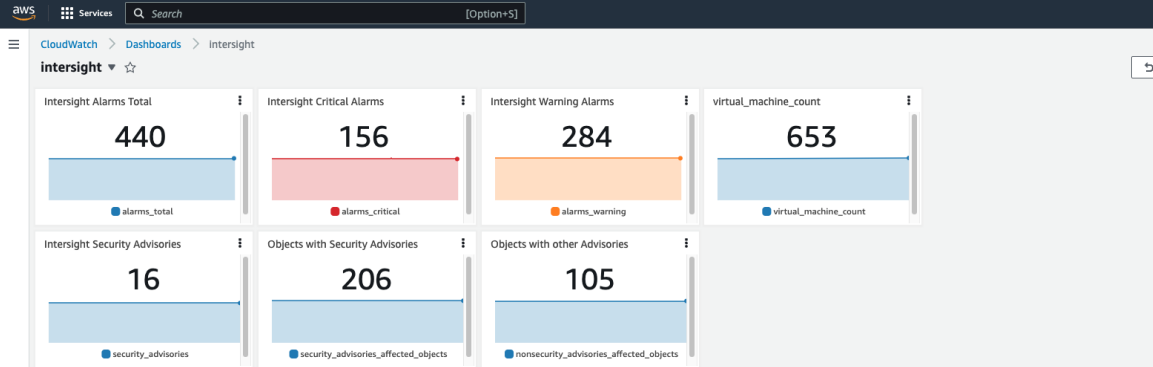
# Intersight and OpenTelemetry?



# Intersight and OpenTelemetry







- Observe
- Explore
- Configure
- FSO Platform Exchange
- Spacefleet
- Intersight

Hyperflex Cluster 1

"5f0528996f72612d30..."

Account 1

anz-sydney5



#### PROPERTIES

intersightdevnet.account.name  
anz-sydney5

intersightdevnet.hyperflex.device.id  
"5f0528996f72612d309575de"

---

**HX Read IOPs**  
5.58 IOPs

---

**HX Write IOPs**  
616.41 IOPs

---

**HX Read Latency**  
0.37 ms

---

**HX Write Latency**  
1.68 ms

---

**HX Read Throughput**  
242.04K bytes/sec

---

**HX Write Throughput**  
6.01M bytes/sec

# What is intersight-otel?

- Lightweight collector (written in Rust)
- Periodically polls Intersight API to retrieve metrics
- Extremely configurable – API requests and metrics generated are configuration not code
- Available as binaries for multiple platforms or container image to run on Docker, Kubernetes, AWS ECS, etc.
- <https://github.com/cgascoig/intersight-otel>

# Configuration overview

```
key_file = "/etc/ismetricskey/intersight-key.pem"
key_id = "0123456789abcdef01234567/0123456789abcdef01234567/0123456789abcdef01234567"
otel_collector_endpoint = "http://otel-collector:4317"

[[pollers]]
name = "virtual_machine_count"
api_query = "api/v1/virtualization/VirtualMachines?$count=true"
aggregator = "result_count"
interval = 300

[[pollers]]
name = "ntp_policy_count"
api_query = "api/v1/ntp/Policies?$count=true"
aggregator = "result_count"
interval = 300

[[tspollers]]
name = "hx_performance"
datasource = "hx"
dimensions = ["deviceId"]
filter = { ... }
aggregations = [...]
post_aggregations = [...]
interval = 300
```

## Global configuration

- Intersight auth
- Otel destination

## Regular Pollers

Call a regular Intersight API and extract a metric value

## TimeSeries Pollers

Query the Intersight “TimeSeries” API for metrics

# Regular Pollers in action

```
[[pollers]]
name = "virtual_machine_count"
api_query = "api/v1/virtualization/VirtualMachines?$count=true"
aggregator = "result_count"
interval = 300
```



intersight-otel

OTel Collector

```
GET https://intersight.com/
api/v1/virtualization/VirtualMachines?$count=true
```

```
{
  "ObjectType": "mo.DocumentCount",
  "Count": 658
}
```

```
Metric #4
Descriptor:
  -> Name: virtual_machine_count
  -> Description:
  -> Unit:
  -> DataType: Gauge
NumberDataPoints #0
StartTimestamp: 2023-05-05 05:33:15.411738108 +0000 UTC
Timestamp: 2023-05-10 08:04:15.41354285 +0000 UTC
Value: 658.000000
```

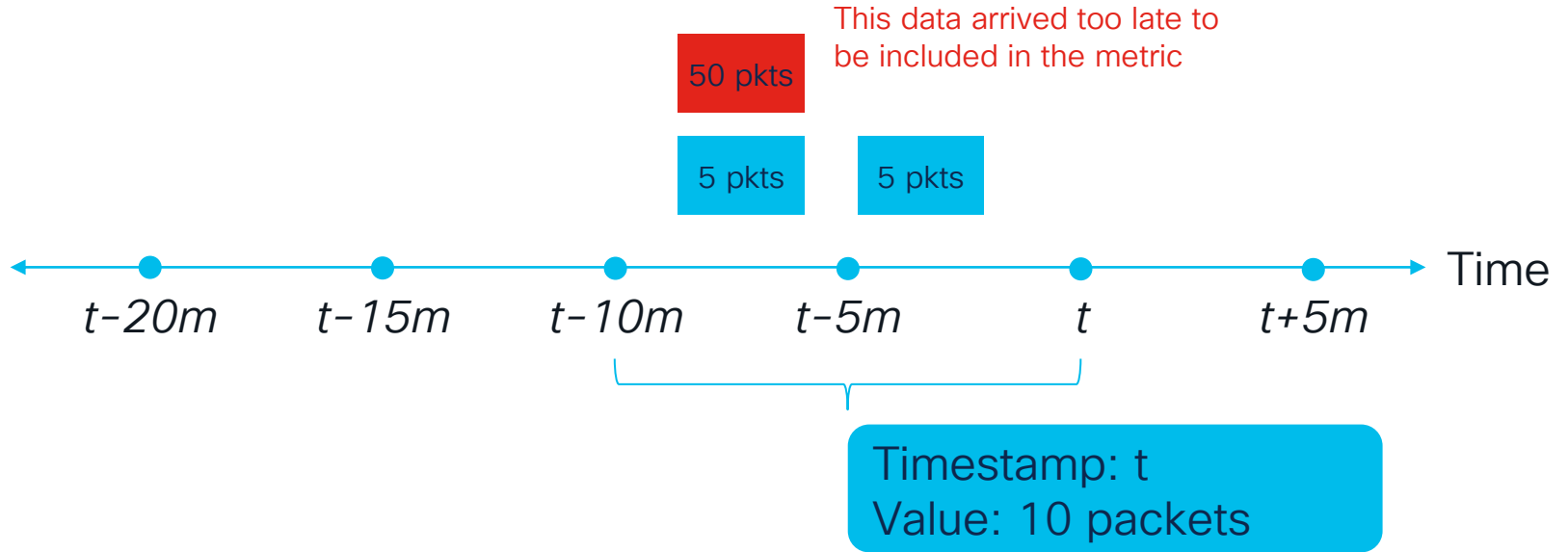
result\_count aggregator

# TimeSeries Pollers - API

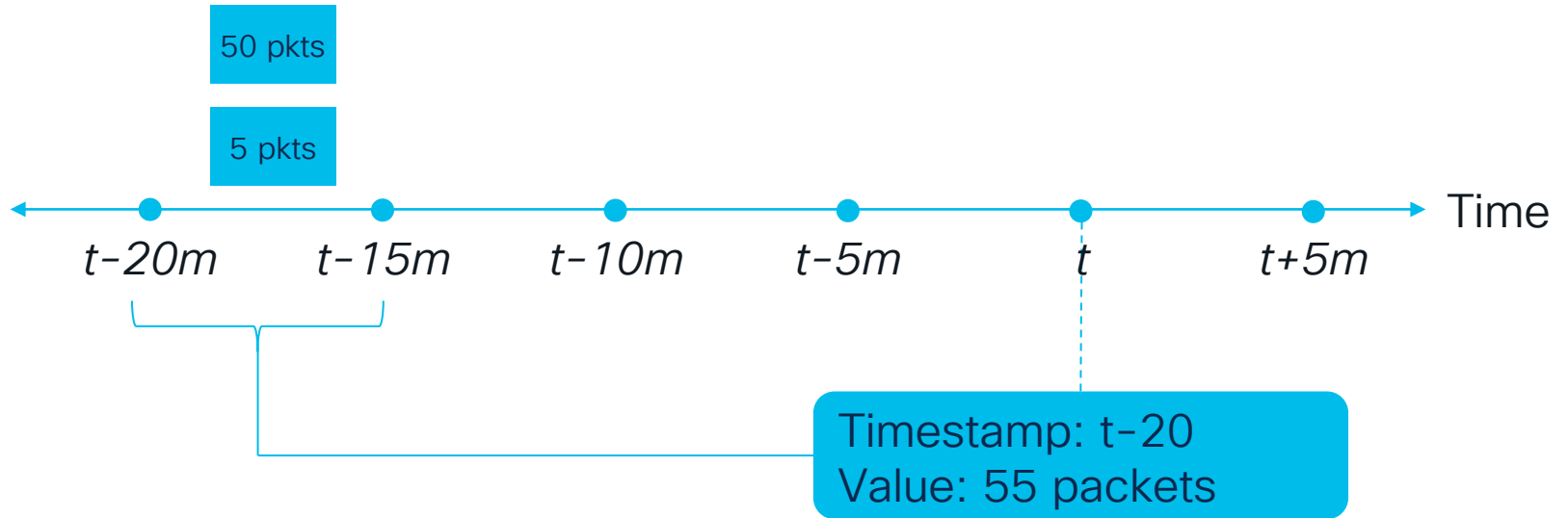
The screenshot shows the Cisco Intersight Developer Center interface. At the top, there is a navigation bar with the Cisco logo, 'Intersight Developer Center', and links for 'Guides', 'API Reference', 'Downloads', 'Code Repo', and 'Support'. A 'Sign In' button is also present. Below the navigation bar, the 'Service' dropdown is set to 'Intersight'. The 'API Reference v1.0.11-12203' section includes a search bar and a list of API endpoints. The selected endpoint is 'POST /api/v1/telemetry/TimeSeries', which is highlighted in a grey box with the text 'Perform a Druid TimeSeries request.'. To the right of the endpoint name, there is a 'REST Client' toggle switch. The main content area displays the 'Request Model' for this endpoint, listing various query parameters and their descriptions:

- `queryType`: string
- `dataSource`: object
- `descending`: boolean Whether to make descending ordered result. Default is false(ascending).
- `intervals`: array A JSON Object representing ISO-8601 Intervals. This defines the time ranges to run the query over.
- `granularity`: object
- `filter`: object
- `aggregations`: object Aggregation functions are used to summarize data in buckets. Summarization functions include counting rows, calculating the min/max/sum of metrics and retrieving the first/last value of metrics for each bucket. Additional summarization functions are available with extensions. If no aggregator is provided, the results will be empty for each bucket.
- `postAggregations`: object Post-aggregations are specifications of processing that should happen on aggregated values as they come out of Apache Druid. If you include a post aggregation as part of a query, make sure to include all aggregators the post-aggregator requires.
- `limit`: integer An integer that limits the number of results. The default is unlimited.
- `context`: object

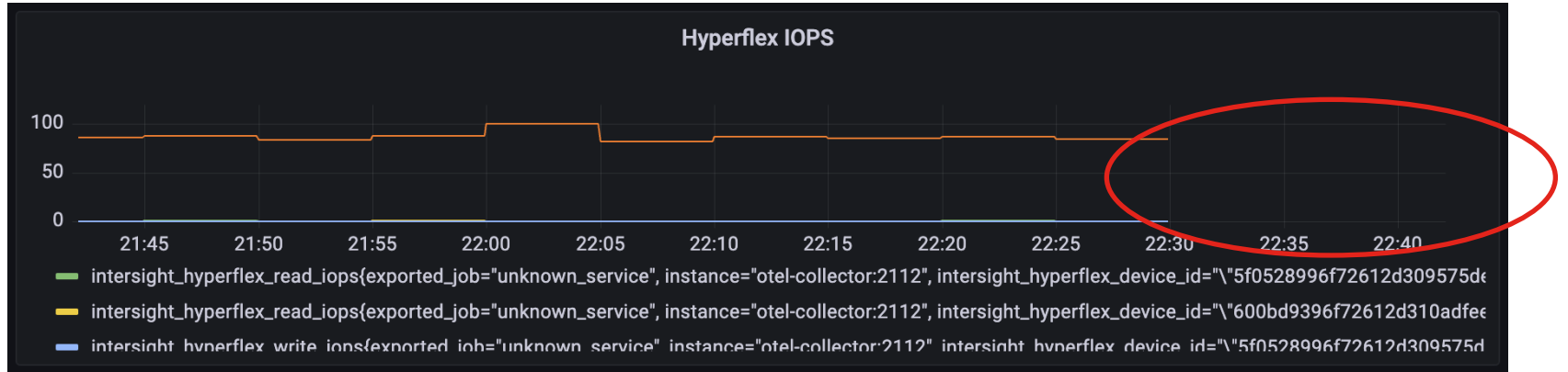
# TimeSeries Pollers – Naïve strategy



# TimeSeries Pollers - Actual strategy

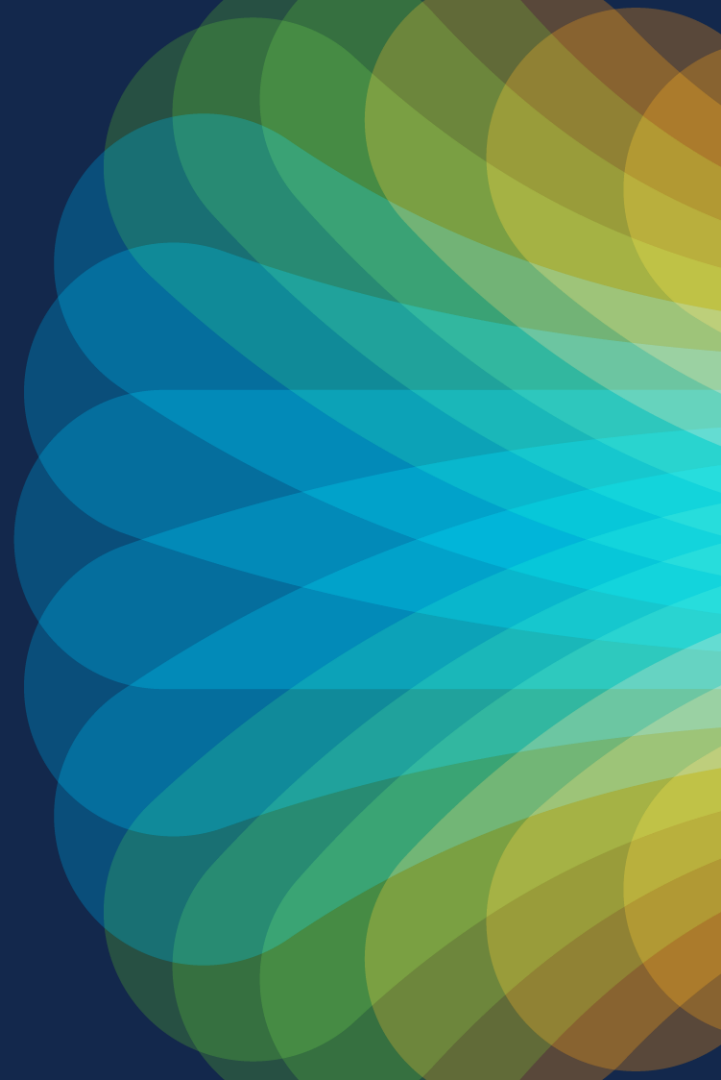


# TimeSeries Pollers



Demo time!

# Key Takeaways



# Key Takeaways

- OpenTelemetry provides a vendor-agnostic telemetry framework
- Once in OpenTelemetry we can do almost anything with that data
- Think of OpenTelemetry Collector as a protocol translator for telemetry
- *intersight-otel* can generate OpenTelemetry metrics from almost any Intersight API call you can think of
  - Please try it out
  - Give feedback
  - “Star” / “Watch” the GitHub repository for updates



The bridge to possible

Thank you

CISCO *Live!*

The Cisco Live! logo features the word "CISCO" in a bold, black, sans-serif font, followed by "Live!" in a black, cursive script font. The background of the entire image is a vibrant, multi-colored abstract pattern of overlapping, wavy bands in shades of red, orange, yellow, green, and blue, radiating from a bright white center on the right side.

CISCO *Live!*

Let's go