Google

How and Why You Should Adopt and **Expose OSS Interfaces**

Case Study: *OpenTelemetry* & *Prometheus* in Google Cloud

Daniel Hrabovcak & Shishi Chen, Google

Nov 2023

Who Are We?

On The Stage

Daniel Hrabovcak

Shishi Chen

Who Are We?

On The Stage

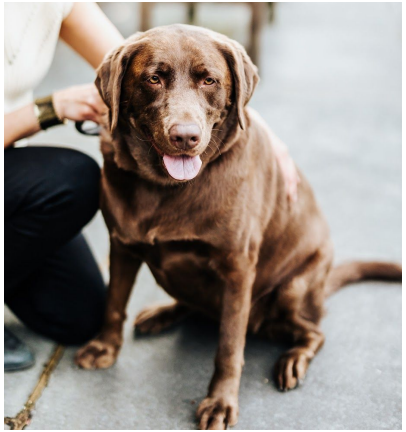
Daniel Hrabovcak

Shishi Chen

And Many More!

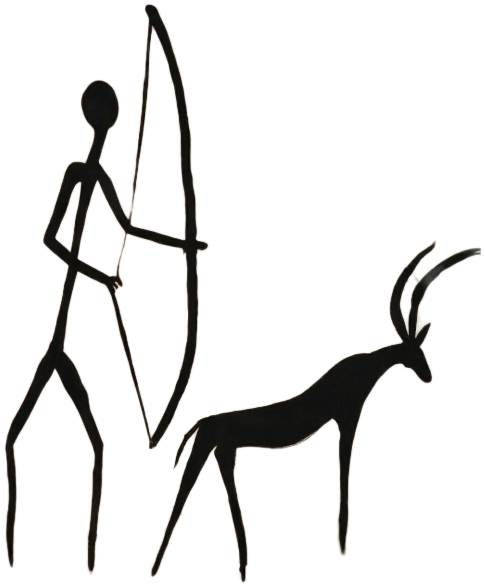
Who Are We?

... Many Many More!



Flashback to 2020...

Monitoring circa 2020…



Source: Duet AI “cave art stick figure”

Google Cloud Monitoring - Ingestion

Source: Duet AI “monarch”

Custom Push-based API (GCM API based on Stackdriver)

- Documentation can only do so much!

Storage: Monarch (Google’s Planet-Scale Time Series Database)

- https://research.google/pubs/pub50652/

Already a high barrier to entry!

Source: Duet AI “brick wall”

Google Cloud Monitoring - Query

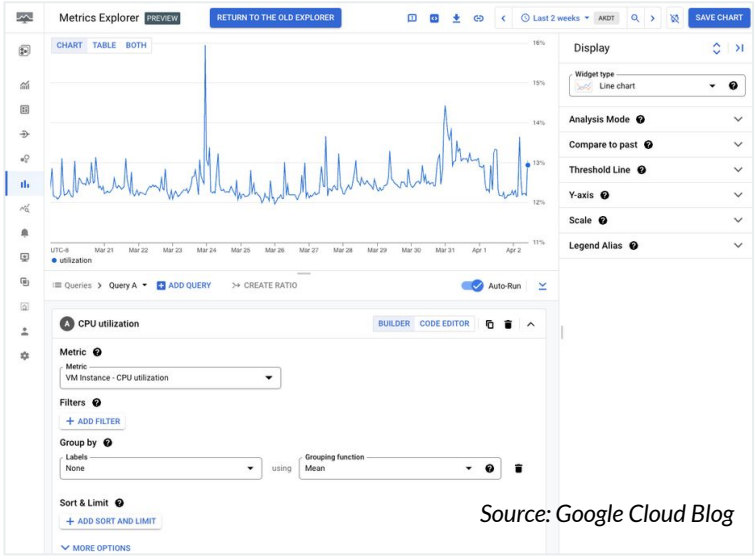
Source: Duet AI “bag of question marks”

Custom Query Language: MQL or UI builder

Visualization: Custom Dashboards (Google Cloud Console)

- Dashboards for common tools are constantly reinvented.

Making a high barrier higher…



Source: Google Cloud Blog

But... Why?

Source: Duet AI "stop sign"

We were not alone! At the time, this made sense to us and other providers.

However, users are not happy when they are siloed.

A Different Universe:
What was the industry doing?

Prometheus - Rise to the Top

Prometheus is an “open-source systems monitoring and alerting toolkit”.

Source: Duet AI “prometheus statue fire”

And rapidly growing in popularity!

2016	2017	2018	2019	2020	2022	2023
2.8k	16k	54k	242k	571k	774k	868k

Source: GrafanaLabs, PromCon 2023, Prometheus adoption stats

Prometheus - Everyone is doing it! (or has it)

Integration: 100s (!!!) of exporters for all your third-party tools and libraries.

Stackoverflow: 6k+ questions

Mailing List: 9k email threads

Talks: Dozens (PromCon, KubeCon)

GitHub Repo: 50k Stars

Avoid vendor lock-in!

Source: Duet AI "smiley money eyes"

Prometheus - Ingestion

Simply expose a metrics HTTP endpoint (i.e. `/metrics`) on your application (pull model)

Example: Prometheus Exposition Format

```
# HELP http_requests_total The total number of HTTP requests.
# TYPE http_requests_total counter
http_requests_total{method="post",code="200"} 1027
http_requests_total{method="post",code="400"}    3
```

Benefits:

- Scrape settings are tweaked on the server (Prometheus).
- Client SDKs are radically simplified.

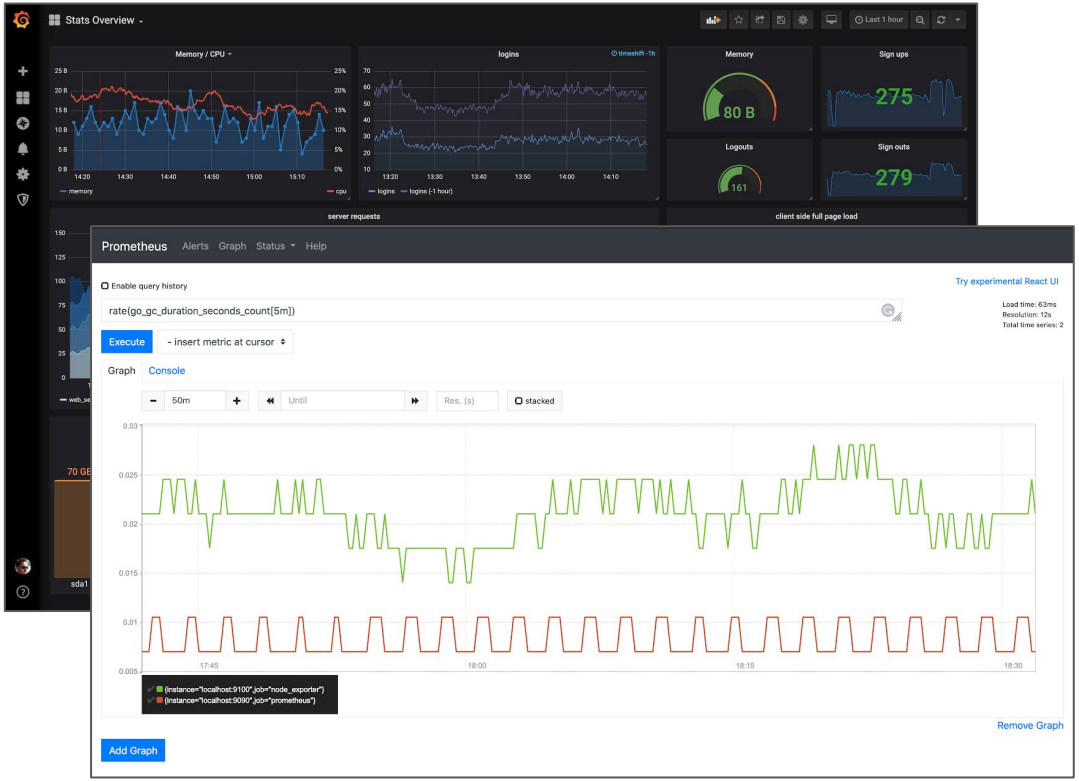
Prometheus - Query

Custom language: PromQL

Visualization: Various

Interoperability: Pre-built dashboards

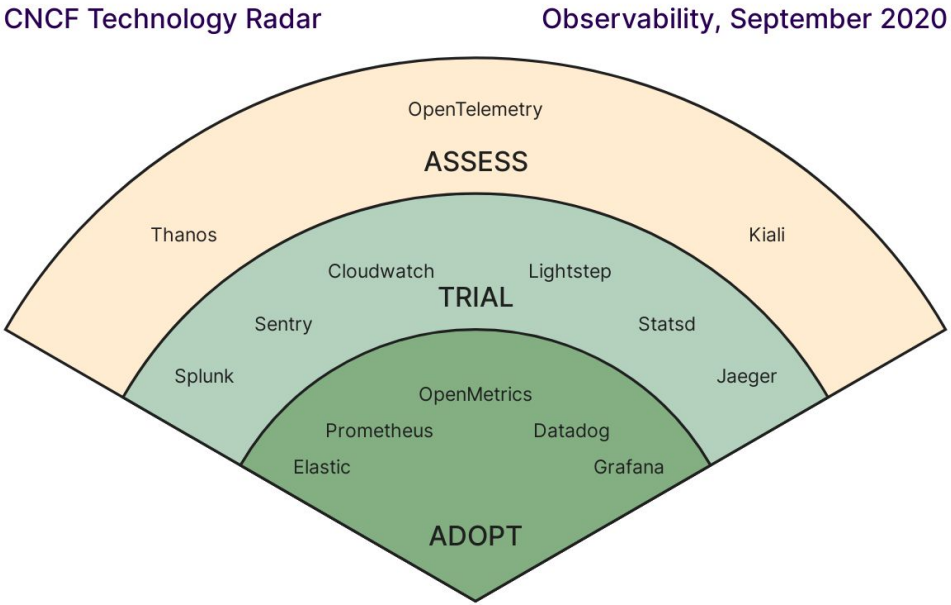
Alerting: Various, + SLO tools



Sources: grafana.com, prometheus.io

Prometheus empowers developers and saves them time!

OpenTelemetry - up and coming



Source: https://radar.cncf.io/2020-09-observability

OpenTelemetry - ingestion alternative

Source: Duet AI “open telemetry” in cyberpunk style

We want to adopt OSS standards!

Now what?

Integration Matrix

Component	Google Cloud Monitoring	
Ingestion	Custom API	Exposition Format
Storage	Monarch	Prometheus
Querying	MQL/UI	PromQL
Rules/Alerts	UI	PromQL
Dashboards	UI	Various

Adopting Prometheus Collection: The Question

Users are already exposing a metrics endpoint in Prometheus Exposition Format.

Idea: Can we let users use their existing applications

BUT Export to Cloud Monitoring instead?

Source: Duet AI "person thinking lightbulb"

Adopting Prometheus Collection: Options

Users are already exposing a metrics endpoint in Prometheus Exposition Format.

Idea: Can we let users use their existing applications

BUT Export to Cloud Monitoring instead?

Sure! But, do we:

1. Create a new parser/tool that emulates Prometheus.
2. Use Prometheus directly.
3. Fork Prometheus to extend it

Source: Duet AI "hammer"

Adopting Prometheus Collection: Options

Users are already exposing a metrics endpoint in Prometheus Exposition Format.

Idea: Can we let users use their existing applications

BUT Export to Cloud Monitoring instead?

Sure! But, do we:

1. Create a new parser/tool that emulates Prometheus.
2. Use Prometheus directly
3. Fork Prometheus to extend it.

Source: Duet AI "hammer"

Adopting Prometheus Collection: Fork Results

1. Easy Onboarding: Drop-in replacement binary.

2. Innovation: Scalable (stateless collection).

- KubeCon 2022 Detroit:
- Stateless Collectors For Stateful Data: Scaling Prometheus As a Node Agent

3. Everyone wins!

Forking is great and doesn't have to be forever!

Source: Duet AI "balloons"

One Step Further: A User's Journey

You're writing a new application in Kubernetes and want to use Prometheus.

What if it was just there by default when you create a new Kubernetes cluster?

Introducing Managed Service for Prometheus

Solution: Default-on in Google Cloud (no install required!)

To do this, we created our own, open source, scalable and secure operator.

- https://github.com/GoogleCloudPlatform/prometheus-engine

Integration Plan: After Collection

Component	Google Cloud Monitoring	
Ingestion	Custom API	Prometheus Expo Format
Storage	Monarch	Prometheus
Querying	MQL/UI	PromQL
Rules/Alerts	UI	PromQL
Dashboards	UI	Various

What about queries and storage?

Adopting PromQL: The Question

Users already have existing queries and dashboards defined with PromQL.

Idea: Can we let users use their existing application flows and lower barrier of entry?

Source: Duet AI "PromQL"

Adopting PromQL: The Options

We considered several implementation options:

- Running a Prometheus instance for each customer
- Using Thanos or Cortex
- Using Monarch

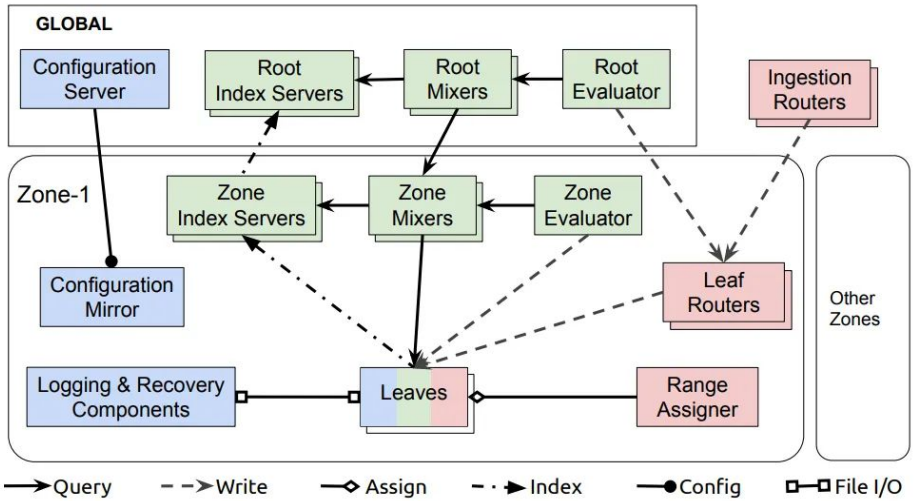
Source: Duet AI “Prometheus Thanos Cortex Monarch”

What is Monarch?

Google's monitoring system:

- stores trillions of timeseries in memory, quadrillions of points on disk
- ingests TBs of data / second
- monitors billions of resources

https://research.google/pubs/pub50652/



Source: Monarch paper

Adopting PromQL: Decision

Users already have existing queries and dashboards defined with PromQL.

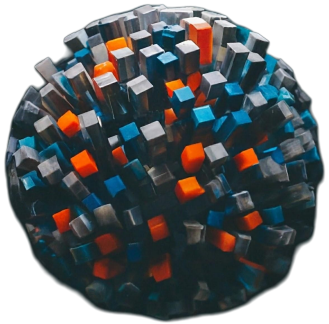
Idea: Can we let users use their existing application flows and lower barrier of entry?

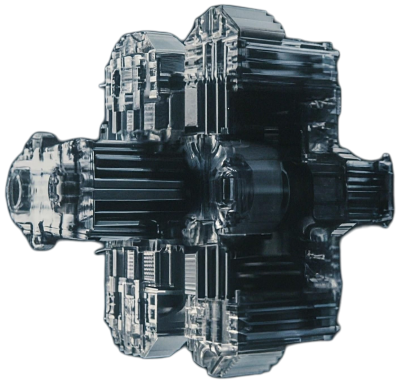
Solution: Implement the Prometheus Query API on top of the Monarch backend.

Adopting PromQL: The Challenges

The Pain Points:

1. **Mismatching data models between PromQL and Monarch**





Source: Duet AI "data model"

Challenge: Mismatching Data Models

Example:

- Prometheus is schemaless while Monarch has resource and metric descriptors

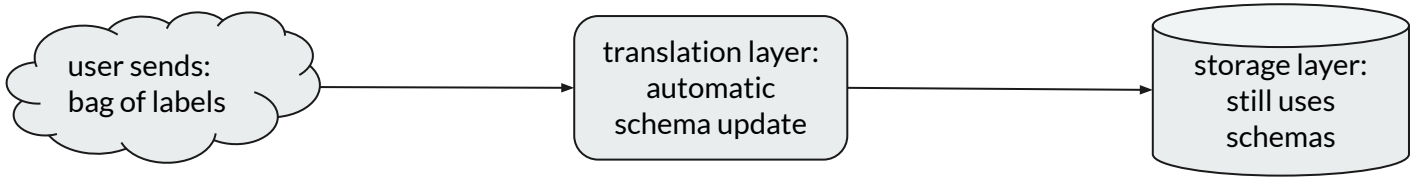
Challenge: Mismatching Data Models

Example:

- Prometheus is schemaless while Monarch has resource and metric descriptors

Improve with:

- Automatic handling of descriptors



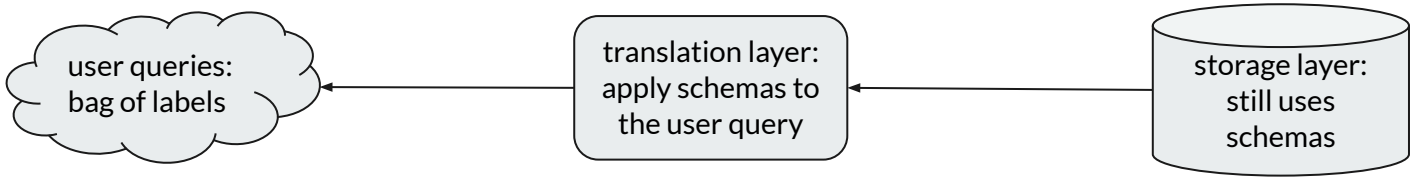
Challenge: Mismatching Data Models

Example:

- Prometheus is schemaless while Monarch has resource and metric descriptors

Improve with:

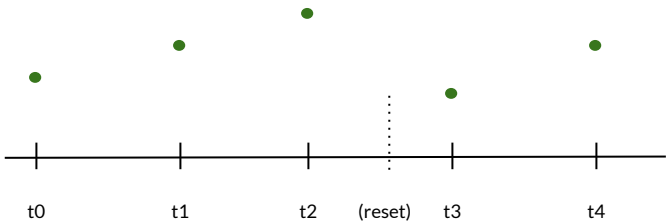
- Automatic translation of queries



Challenge: Mismatching Data Models

Example:

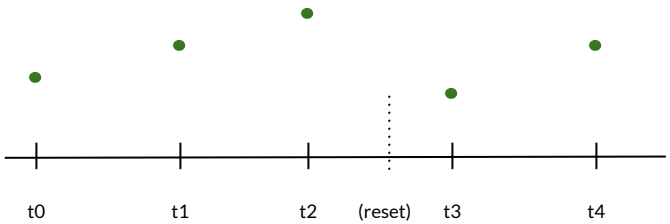
- Prometheus has single timestamp attached to each point, while Monarch represents counters as intervals over a start and end timestamp

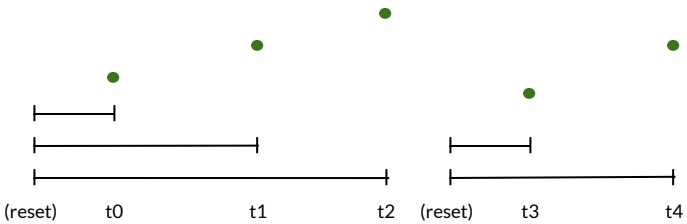


Challenge: Mismatching Data Models

Example:

- Prometheus has single timestamp attached to each point, while Monarch represents counters as intervals over a start and end timestamp





Challenge: Mismatching Data Models

Example:

- Prometheus has single timestamp attached to each point, while Monarch represents counters as intervals over a start and end timestamp

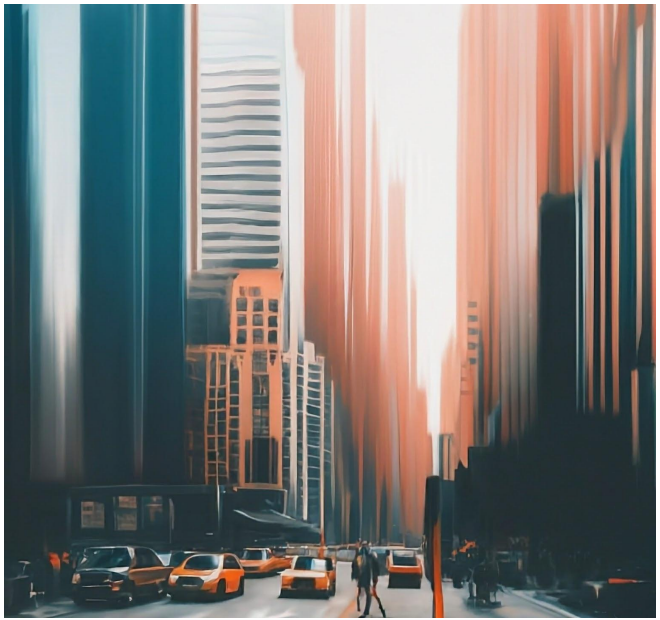
Improve with: Community involvement

- Created timestamp
  - PromCon EU 2023: When My Counter Restarted

Adopting PromQL: The Challenges

The Pain Points:

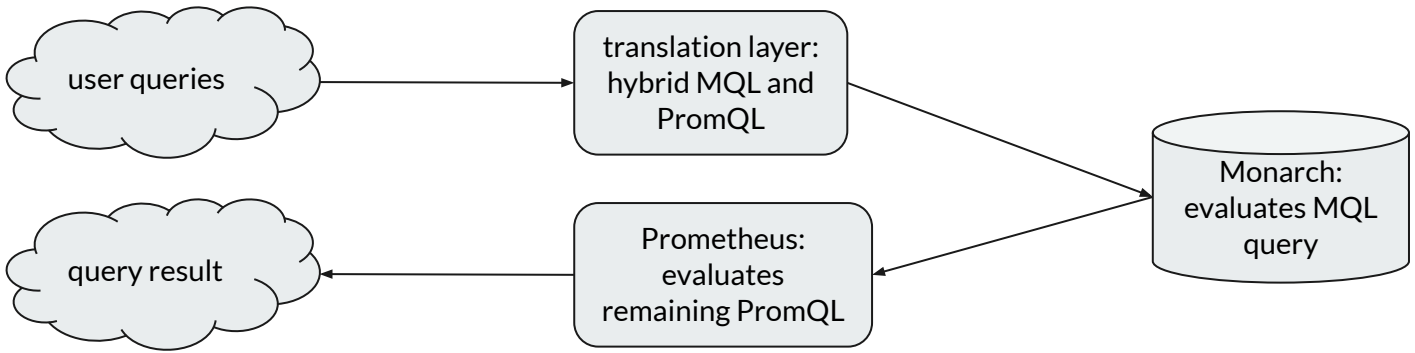
2. **Keeping the API up to date**



Source: Duet AI "modernization"

Challenge: Keeping the API up to date

Improve with: Hybrid implementation, running the OSS PromQL implementation alongside Monarch



Example Hybrid Query

Input:

```
topk(3,sum
by (app,
proc)
(rate(instan
ce_cpu_time_
ns[5m])))
```

Example Hybrid Query

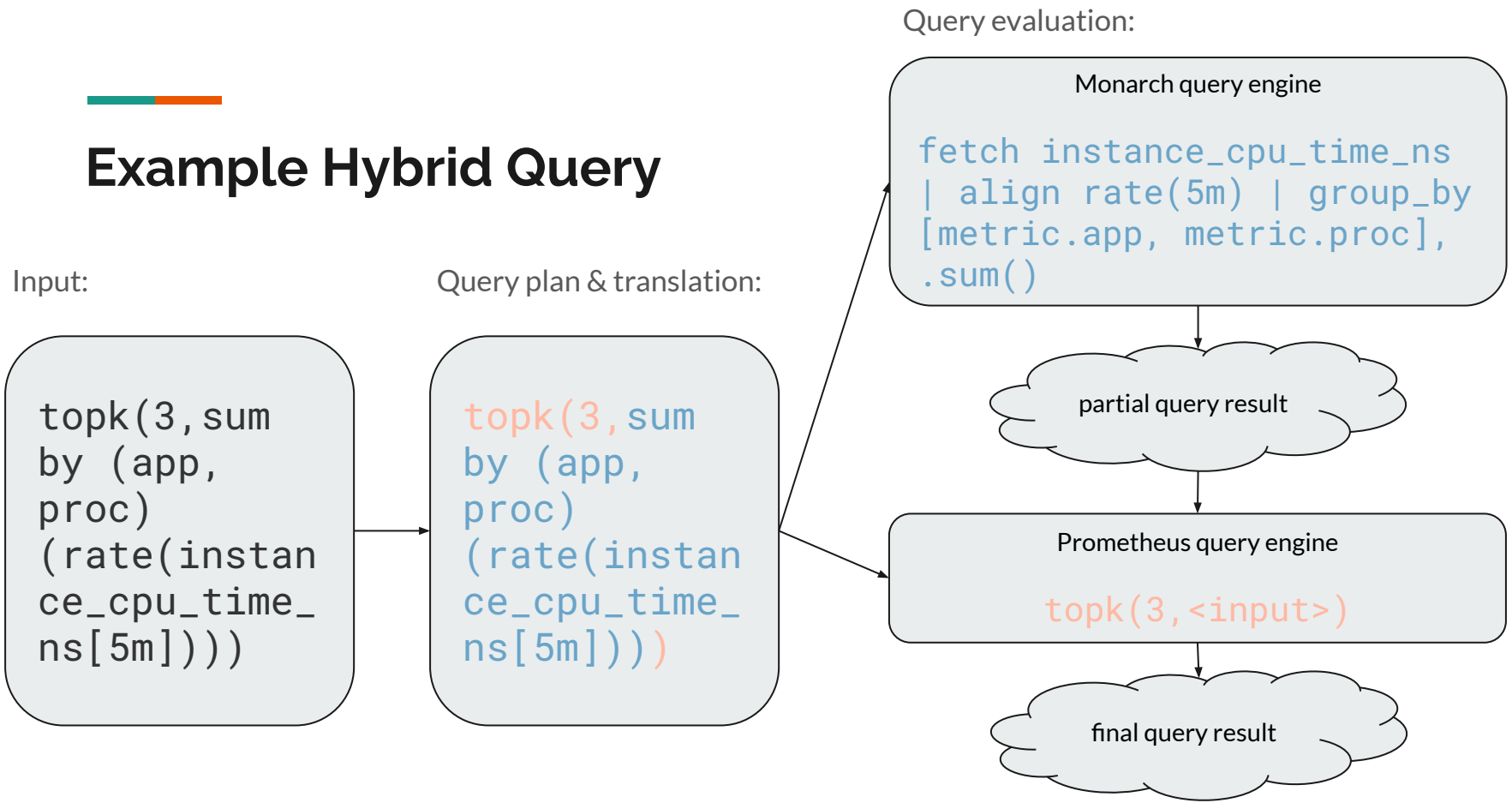
Input:

```
topk(3,sum
by (app,
proc)
(rate(instan
ce_cpu_time_
ns[5m])))
```

Query plan & translation:

```
topk(3,sum
by (app,
proc)
(rate(instan
ce_cpu_time_
ns[5m])))
```

Example Hybrid Query



Example Hybrid Query

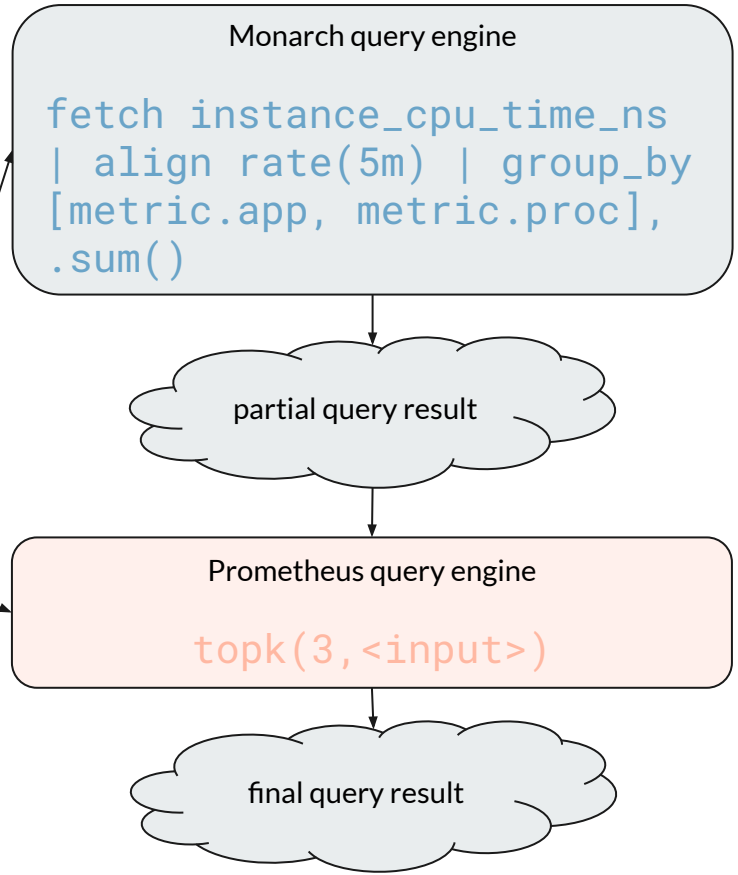
Input:

```
topk(3,sum
by (app,
proc)
(rate(instan
ce_cpu_time_
ns[5m])))
```

Query plan & translation:

```
topk(3,sum
by (app,
proc)
(rate(instan
ce_cpu_time_
ns[5m])))
```

Query evaluation:



Adopting PromQL: Results

1. Easy Onboarding: Drop-in replacement Prometheus HTTP API.

2. Integrated: Single point of entry to access all kinds of metric data, including non-Prometheus Google Cloud metrics.

3. Scalable: Storage is Monarch, Google's multi-tenant monitoring database. Queries are partially evaluated by Monarch's distributed query evaluator.

4. Everyone wins!

Integration Plan: Query

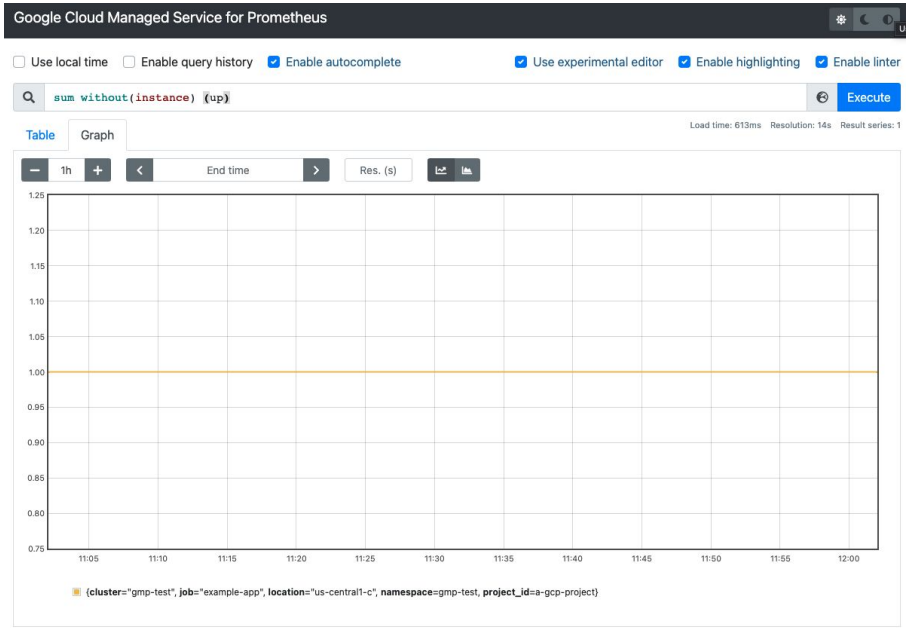
Component	Google Cloud Monitoring	
Ingestion	Custom API	Prometheus Expo Format
Storage	Monarch	Prometheus
Querying	MQL/UI	PromQL
Rules/Alerts	UI	PromQL
Dashboards	UI	Various

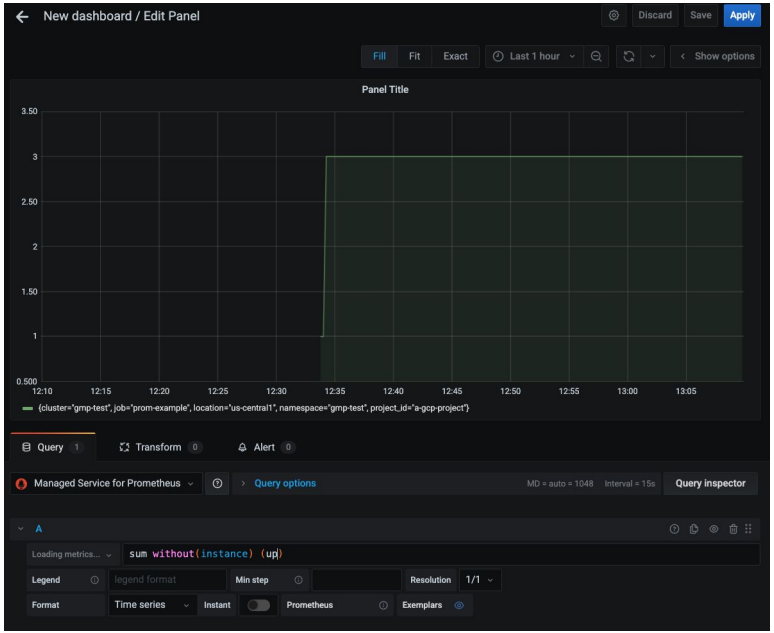
What about Alerting and Dashboards?

OSS SDKs works OOTB!

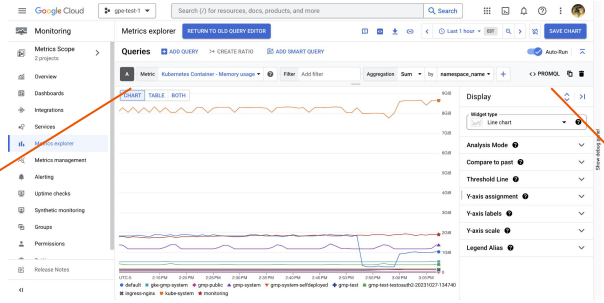
```
import "github.com/prometheus/client_golang/api"

client, err := api.NewClient(api.Config{
   Address: fmt.Sprintf("https://monitoring.googleapis.com/v1/projects/ %s/location/global/prometheus" ,
ProjectID),
   Client:  oauth2.NewClient(ctx, TokenSource),
})
```

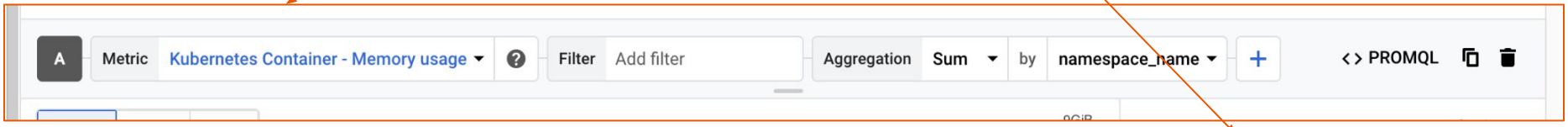
PromQL queries & dashboards work OOTB!

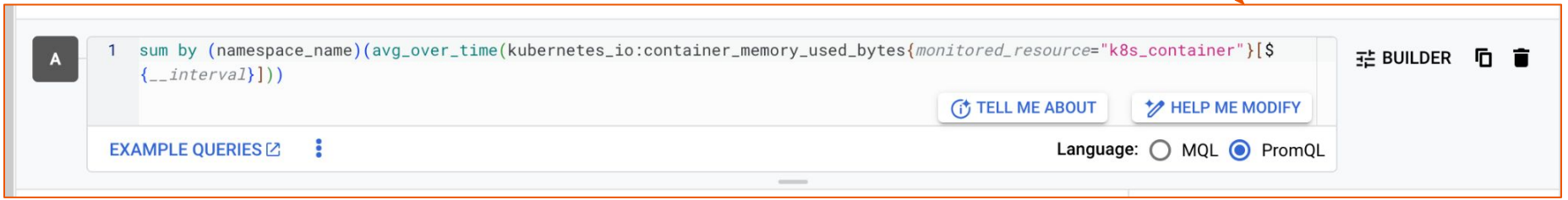




Instead of PromQL, use Google Cloud Monitoring!







Integration Plan: Result

Component	Google Cloud Monitoring	
Ingestion	Custom API	Prometheus Expo Format
Storage	Monarch	Prometheus
Querying	MQL/UI	PromQL
Rules/Alerts	UI	Prometheus API
Dashboards	UI	Prometheus API

Instead of Prometheus collection, use OpenTelemetry!

OpenTelemetry → GCM

1. OpenTelemetry Metrics Exporter
    - Uses Prometheus collector to scrape OpenTelemetry.
2. OpenTelemetry Collector
    - Uses OpenTelemetry to scrape Prometheus to GCM.
3. Future: OpenTelemetry uses Prometheus Remote Write (PRW)
    - PRW will eventually work with GCM API

Source: Duet AI "interoperability"

Integration Plan: Result

Component	Google Cloud Monitoring	
Ingestion	Custom API Prometheus Expo Format OpenTelemetry	
Storage	Monarch	Prometheus
Querying	MQL/UI	PromQL
Rules/Alerts	UI	Prometheus API
Dashboards	UI	Prometheus API

Let’s recap

Learnings for Vendors

1. Users are not happy when they are trapped in silos.

Learnings for Vendors

1. Users are not happy when they are trapped in silos.
2. OSS standards has enormous community momentum producing amazing tools and learning resources so why not reuse them?

Learnings for Vendors

1. Users are not happy when they are trapped in silos.
2. OSS standards has enormous community momentum producing amazing tools and learning resources so why not reuse them?
3. Adopt incrementally (if you can).

Learnings for Vendors

1. Users are not happy when they are trapped in silos.
2. OSS standards has enormous community momentum producing amazing tools and learning resources so why not reuse them?
3. Adopt incrementally (if you can).
4. You are not limited: Don't settle on OSS as-is, you can always improve and impact OSS yourself!

Learnings for Vendors

1. Users are not happy when they are trapped in silos.
2. OSS standards has enormous community momentum producing amazing tools and learning resources so why not reuse them?
3. Adopt incrementally (if you can).
4. You are not limited: Don't settle on OSS as-is, you can always improve and impact OSS yourself!
5. Contributing to OSS fosters innovation and accelerates progress in the OSS space.

Thanks! Questions?

Daniel Hrabovcak & Shishi Chen, Google

Resources:

- https://github.com/GoogleCloudPlatform/prometheus
- https://github.com/GoogleCloudPlatform/alertmanager
- https://github.com/GoogleCloudPlatform/prometheus-engine

Archived Slides - not presented

Agenda

- Presenters Intro
- WHY “Fail” GCM 3y ago: Waking up in weird spot with custom APIs “Here we are”
  - Define what “failing” is.
  - Ask Lee if he has any data points to strengthen our claim.
  - Users not being able to reuse artifacts done by the community or by your previous OSS usage (or it’s hard) e.g. recording rules, dashboards, alerts, probes
    - https://github.com/Vince-Cercury/Grafana-Dashboards/blob/master/etcd-prometheus-dashboard.json
  - Users not being able to reuse their instrumentation (!), which hard to change/improve
  - No community driven support documentation (open source is so good at this)
  - Adding different client code to handle custom API is one thing, but often the data model is completely incompatible (cannot do the same thing across vendor vs OSS)
    - OSS standards are usually organically grown and simplified (because this is what the user wants), so custom API are typically worse, and more difficult to use.
  - User Lock-in
  - Users cannot use common knowledge (e.g new hire onboarding, or motivation to learn details)
- WHAT Are there any OSS standards -> YES
  - PromQL for reads, OTEL for push, Prom for pull
- HOW We learned on mistakes and guess what, providers can (and should adopt OSS standards)!
  - Case Study:
    - Adopting PromQL to Monarch
    - Adopting PromQL in UI
    - Adopting OpenTelemetry collection
    - Adopting Prometheus collection
  - Contribute back - don’t just use it!
    - created timestamp, etc
  - Not only PromQL but really Prometheus APIs
  - Ways we innovated?
- Summary
  - Acknowledge problem
  - Contribute back

Examples

can we get some customer quotes from lee for examples

or stats etc

https://docs.google.com/document/d/1cV0MrCNsFzZEZLW814uo9XXGwCPiUjIp1mK-r53hVh8/edit

https://docs.google.com/document/d/1ZzMo71ivOSbhNo1ISg3Zw-T5w_HRDhGzoBRmMRaPErI/edit#heading=h.dze7ntggs6wp

https://docs.google.com/document/d/1LNh-ki9aG4WJh8xCjtFVvhYVdslFdakEX1tdshWWEw4/edit#heading=h.h99bpsg2joha

https://docs.google.com/presentation/d/1WTd10h4wTMYMBGY3CTUkzkerK_oryRo8J91Qb2npcsw/edit?resourcekey=0-XkXZiqMelxtYv1kDFLEsnw#slide=id.gcf961b00e2_0_10

What can we adopt?

Ingestion: People are building apps. Prometheus text expo format is open.

This centralizes data (within Google Cloud)

Reuse Prometheus libraries?

OpenTelemetry for push

Query: People are using PromQL

People are using the Prometheus HTTP API.

Centralizes logic (outside of Google Cloud)

Adopting PromQL: The Challenges

The Pain Points:

Mismatching data models between PromQL and Monarch

Examples:

2. Prometheus represents all data as double-valued points, while Monarch has more complex types such as distributions

Adopting PromQL: The Challenges

The Pain Points:

Mismatching data models between PromQL and Monarch

Examples:

2. Prometheus represents all data as double-valued points, while Monarch has more complex types such as distributions

Improve with: Adoption of new features such as native histograms

(Case Study) Adopting OpenTelemetry

OpsAgent being an Otel Collector (conversion)

Maintaining and contributing OpenTelemetry pieces (e.g. leading Go Otel SDK)

Ingestion OTLP support development

Examples of things you can do

- “there are 50% more PromQL queries than MQL queries despite MQL being all in console (OIC)”
- Show reusing dashboards easily, for example

Don’t just adopt, but contribute back!

As we mentioned previously, contributing back has helped us overcome our pain points

- Created timestamp
- Prometheus scalability patterns (daemonSet)
- Remote write
- TAG Observability Blueprint
- Agent mode optimizations
- other examples

Can we do even better?

- We can always be more involved.
- Increased transparency?