

From Paths to Dimensions: How we Modernized Database Observability at Scale

Ioannis Androulidakis, SRE, Booking.com
Maris Elsins, Principal Engineer, Pythian

Observability Summit Europe
Mon 5th Oct 2026, Prague Czechia

Booking.com Pythian

About us

- ❑ Ioannis Androulidakis
 - ❑ Site Reliability Engineer at [Booking.com](https://www.booking.com)
 - ❑ Operating Systems, Storage, Observability, OSS
-



- ❑ Maris Elsins
- ❑ Principal Engineer at [Pythian](https://pythian.com)
- ❑ MySQL, Oracle, DB Reliability Engineering





Today's Journey

1. MySQL at Booking.com
2. Metrics Migration: Motivation, Challenges and Plan
3. Onboarding Producers
4. Migrating Consumers
5. Lessons Learned
6. Q/A



Today's Journey

1. **MySQL at Booking.com**
2. Metrics Migration: Motivation, Challenges and Plan
3. Onboarding Producers
4. Migrating Consumers
5. Lessons Learned
6. Q/A

At **Booking.com** we run MySQL at scale

>300

MySQL Clusters

>15K

MySQL Hosts

>100M

Queries Per Second (QPS)

Key Services

Frontend, Reservations,
Payments, Customer Support,
Governance

Main Relational Datastore

Single primary for WRITES, scale-out
replicas for READS

Usually we choose availability over
consistency ^[1]



Ownership Model

✗ We **do NOT own**: schemas, tables, data

✓ We **DO own**: database infrastructure

Multi-cloud

Our database fleet spans across multiple platforms:

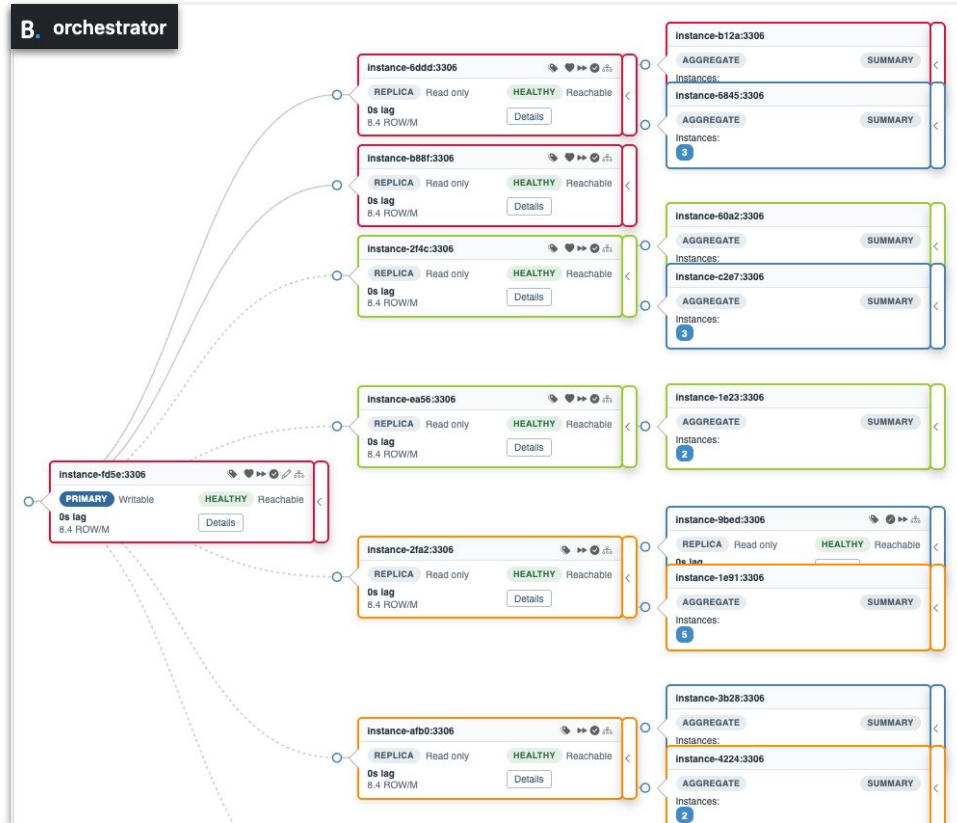
- Bare-Metal (ServerDB)
- Private Cloud (BC2)
- Public Cloud (AWS)
 - Self-managed (Amazon EC2)
 - Managed (Amazon Aurora, RDS)

Multi-region

Always deploy in **> 2 regions** for redundancy

[1] CAP & PACELC Theorems: https://en.wikipedia.org/wiki/CAP_theorem / https://en.wikipedia.org/wiki/PACELC_theorem

MySQL Cluster Topology



Running on Graphite for 13+ years

Our on-prem Graphite installation has served well over the years, scaling to huge numbers for database metrics

```
commit 70a8ed30df8c42cc37e244d9228538c250d3df9c
Date:   Wed Feb 6 15:08:19 2013 +0100

diff --git a/DBA/comm/graphite.py b/DBA/comm/graphite.py
new file mode 100644
```



- [1] <https://github.com/grobjan/carbon-c-relay>
- [2] <https://github.com/go-graphite/gorelka>
- [3] <https://github.com/bookingcom/nanotube>
- [4] <https://grafana.com/blog/how-booking-com-handles-millions-of-metrics-per-second-with-graphite>

Sunsetting Graphite

Booking.com has chosen to replace Graphite with a new, centralized **OpenTelemetry-based** observability pipeline.



Issues with Graphite:

1. **Not actively maintained in 2026** → no security updates, no new features
2. **No support for multi tenancy** → unclear ownership
3. **No native integration with OTEL** → architecture clashing, complex configuration
4. **Limited** data labeling, storage, and querying capabilities

Thread



Jul 28th, 2025 at 11:38 AM

Hi everyone!

We're starting the initial phase of Graphite deprecation and transition to Mimir, our new metrics solution. As part of this strategic upgrade, we are actively seeking **early adopter teams** to work together with us on pilot migrations.

Enter Grafana Mimir

1. 100% Prometheus-compatible
2. Multiple metric types, e.g., built-in histograms, summaries, counters
3. Multi-tenant (avoid noisy neighbors, clear usage, easier budgeting)
4. Better support, large and active community (libraries, exporters, etc.)
5. Clear retention and storage (no age downsampling or inactivity removal, resolution relies on the metric)
6. Flat hierarchy with explicit data dimensions:

`prod.frontend.web-01.http_requests.200`

vs

`http_requests_total{environment="prod", app="frontend", instance="web-01", status="200"}`



<https://prometheus.io/docs/introduction/comparison/#prometheus-vs-graphite>



Today's Journey

1. MySQL at Booking.com
2. **Metrics Migration: Motivation, Challenges and Plan**
3. Onboarding Producers
4. Migrating Consumers
5. Lessons Learned
6. Q/A

Why this migration was difficult

- At **20M** time series “lift and shift” was not an option. Run both Graphite and Mimir for a certain period of time → complexity, technical debt
- **Decade-old** integrations and dependencies are hard to overcome (DBA automation, fleet management, metric aggregation, dashboards, alerts all rely on Graphite) → extended scope, multiple failure points
- Graphite and Mimir follow **different** approaches → learning curve, shift in mental model (beyond code)
- **Cross-team** dependencies:
 - We rely on tooling/infra owned by the Observability team
 - Other teams rely on MySQL metrics owned by us
 - Agree on expectations, timeline and ownership

• ~20.3 million metrics total.

mysql.* - 16,409,125

Subtree	Metrics
mysql.monitoring.production	13,003,732
mysql.monitoring.dqs	1,925,956
mysql.monitoring.corporate	812,227
mysql.capacity_planning	643,587
mysql.minutely	22,354
mysql.monitoring.ten_minutely	1,264
mysql.monitoring.bacula	5

aggregations.*.mysql.* - 3,883,179

Bucket	Metrics
secondly	1,382,812
minutely	1,323,475
hourly	1,127,172
daily	45,222
five_minutely	4,498

Summary by Claude + Graphite MCP server

A Progressive Approach

Phase 0 – Assessment

Set the **scope** and **timeline** for the migration

Phase Details

- Explore Graphite MySQL namespaces
- Familiarize with OTEL and its multi-tenancy model
- Identify producers and consumers
- Design integration (code, SDK, Puppet)

Phase 1 – Double Write

Implement **parallel** OTEL-based metrics **stack**

Phase Details

- Introduce new OTEL metrics to replace Graphite ones
- Map Graphite paths to OTEL attributes
- Configure and deploy OTEL collector across the MySQL fleet
- Enable double write across all producers

Phase 2 – Migrate Reads

Switch all **consumers** from Graphite to Mimir

Phase Details

- Implement routing layer for choosing metrics backend
- Compare Graphite and PromQL queries
- Gradually migrate consumers based on business criticality
- Notify teams

Phase 3 – Grace Period

Run the two stacks in parallel: **compare** and **evaluate**

Phase Details

- Monitor automation, dashboards and alerts
- Monitor Mimir tenant size, retention, rate limiting, etc.
- Verify overall stability

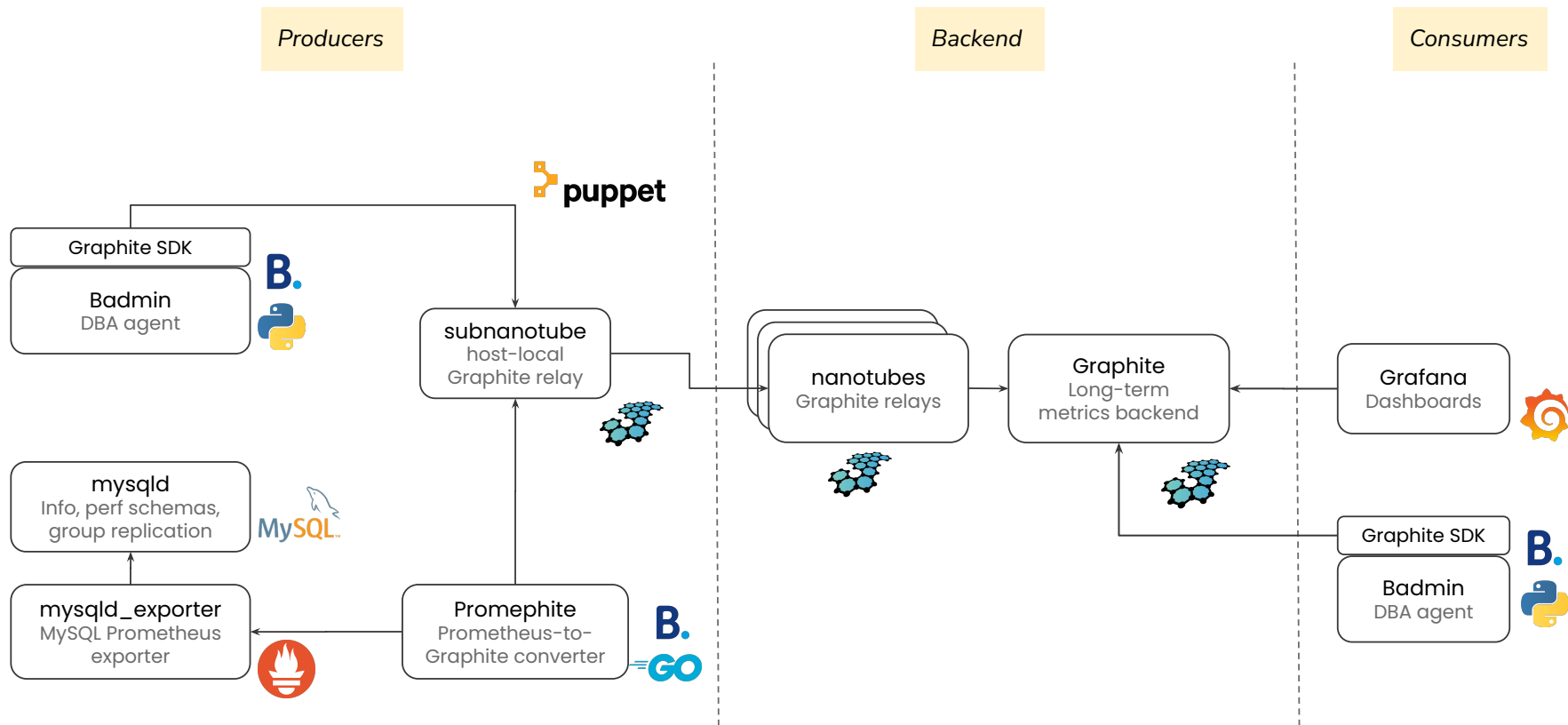
Phase 4 – Cleanup (ongoing)

Remove unused config, components, metrics

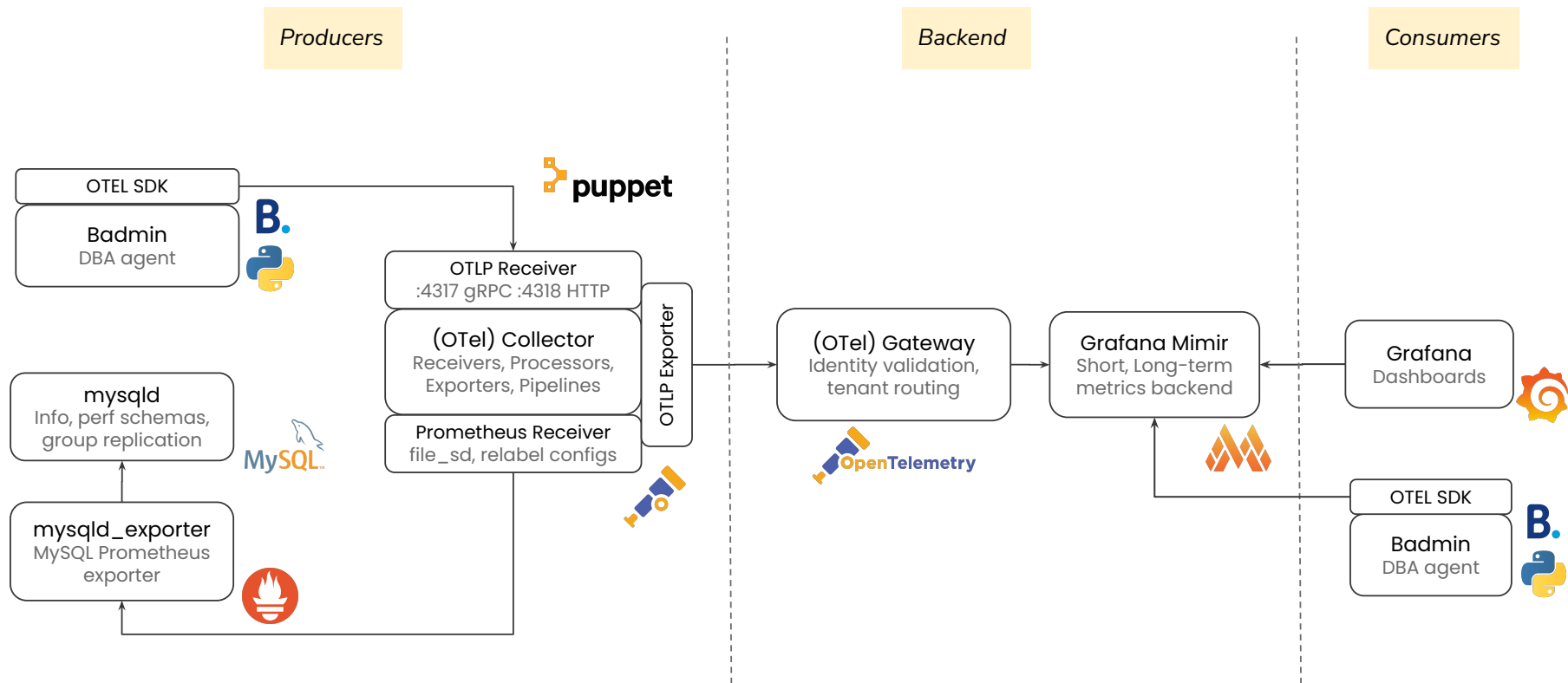
Phase Details

- Backfill historical metrics
- Remove double write logic from producers
- Remove routing logic from consumers

Graphite-based stack



OTEL-based stack





Today's Journey

1. MySQL at Booking.com
2. Metrics Migration: Motivation, Challenges and Plan
3. **Onboarding Producers**
4. Migrating Consumers
5. Lessons Learned
6. Q/A

Producers: Pushing metrics from Badmin

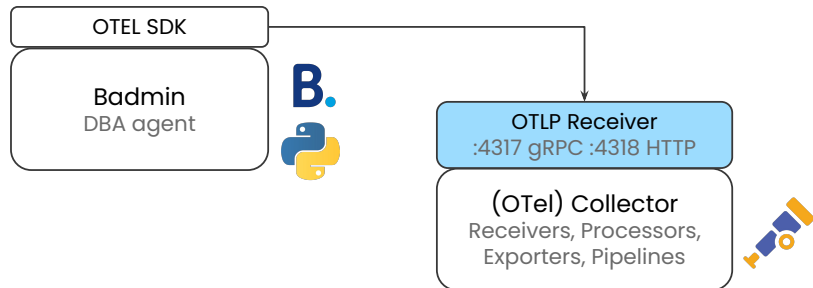
OTEL PYTHON SDK

OTLPMetrics() Core Class

- Export periodically to local OTLP Receiver
- Integrate with Service Directory → Mimir Tenant routing
- Set data temporality (delta vs cumulative)
- Context-manageable class to ensure flush on exit
- **Wrapper methods:** `set_gauge()`, `add_counter()`, `record_histogram()`
- **Upstream libraries:** `opentelemetry-sdk`, `opentelemetry-exporter-otlp-proto-http`

DOUBLE-WRITE STRATEGY

- 01 Identify all locations where `Graphite.push()` is used
- 02 Call `OTLPMetrics()` to double-write metrics to Mimir
- 03 Simplifies detection of legacy Graphite calls later



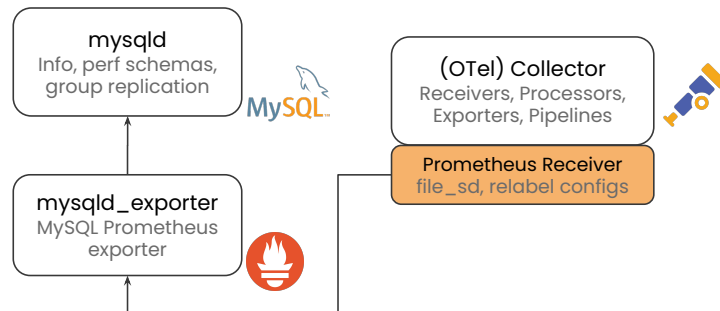
```
# Send metrics to OTEL
with OTLPMetrics() as otel:
    otel.set_gauge(
        meter_name=OtelMeters.BACKUP.value,
        gauge_name="backup.bacula.sd.volume.bytes",
        description="Disk space on the Bacula SD Volume",
        value=value,
        attributes={OtelAttributes.STATE.value: state},
    )

# Send metrics to Graphite
self.graphite.push(metric_name, value)
```

Producers: Pulling metrics from mysql_exporter

OTEL PROMETHEUS RECEIVER

- Use Puppet to configure scraping across the fleet
- Service discovery + custom attributes via `file_sd_configs` (refreshed by a systemd service)
- `$metrics_to_keep` retains 63 of ~970 metrics provided
- No need for metrics translation



```
otel::config::prometheus { 'mysql_exporter':  
  sd_config_type      => 'file',  
  file_sd_configs     =>  
  [ '/run/otelcol/dba_custom_attributes.json' ],  
  metric_relabel_configs => [  
    {  
      source_labels => [ '__name__' ],  
      regex         => $metrics_to_keep.join('|'),  
      action        => 'keep',  
    },  
  ],  
}
```

```
$ cat /run/otelcol/dba_custom_attributes.json  
[  
  {  
    "labels": {  
      "announcing_in_zookeeper": "True",  
      "is_primary": "False",  
      "pools": "testdb-misc",  
      "sdb_status": "live",  
      ...  
    },  
    "targets": [  
      "localhost:..."  
    ]  
  }  
]
```

The Cardinality Trap

With ~15K hosts, setting the `host_name` label on all metrics can cause a time-series explosion. Ingestion limits are applied to every Mimir tenant by default.

rate limiting → **missing metrics / broken consumers**

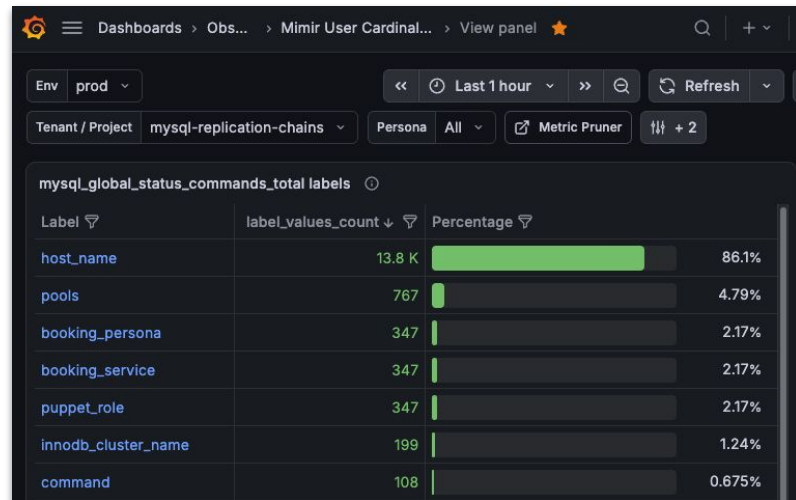
Cardinality Explosion

Case 1: Database Schema Overhead:

```
mysql_info_schema_table_rows{host_name=, table=},  
mysql_info_schema_table_size{host_name=, table=}
```

Case 2: High-Dimensional Zero Counters:

```
mysql_global_status_commands_total{host_name=, command=}
```



Cardinality Control – How?

- **Fine-Tune Ingestion Limits** upward to match real-world scale – defaults are too restrictive.
 - increase rate limits for OTEL processors (`metric_limits`)
 - increase series limits for Mimir (`max_global_series_per_user`)
- **Use OTEL Filter Processors:** Introduce custom [OTEL filter processors](#) to drop unused time series before storage.
 - `filter/drop_zeros` – to skip 0-values for specified metrics series
 - `filter/drop_unused` – to skip pushing additional listed metrics to Mimir

```
$drop_zero_metrics = [  
  'mysql_global_status_commands_total',  
  'mysql_global_status_performance_schema_lost_total',  
  'mysql_global_status_connection_errors_total',  
  'mysql_info_schema_table_rows',  
  'mysql_info_schema_table_size',  
]  
  
$filter_drop_zeros_conditions = $drop_zero_metrics.map  
|$metric| {  
  "- metric.name == \"${metric}\" and value_double ==  
  0.0"  
}.join("\n      ")  
  
otel::collector::config::fragment::processor {  
  'filter_drop_zeros':  
    content => @"EOF"  
    filter/drop_zeros:  
      metrics:  
        datapoint:  
          ${filter_drop_zeros_conditions}  
    | EOF  
}
```



Today's Journey

1. MySQL at Booking.com
2. Metrics Migration: Motivation, Challenges and Plan
3. Onboarding Producers
- 4. Migrating Consumers**
5. Lessons Learned
6. Q/A

Consumers: Reading metrics from Badmin

MIMIR CLIENT

Querying Mimir is simple - `Mimir()`

- A simple API client we built to interact with Mimir (`GET /api/v1/query_range`)
- Sets HTTP headers for routing to the specified Mimir tenant
- parses raw API response into typed objects

RISKS & CHALLENGES

Switching Reads from Graphite is Risky

- Possible metrics **quality issues**
- Possible **bugs** in new PromQL queries
- Even correct queries can render **different results** (handling time, missing values, metric types)

```
>>> check=MySQLUserConnectionsCountCheck()
>>> check._get_user_connections_nth_percentile_from_graphite(username, 95)
189
>>> check._get_user_connections_nth_percentile_from_mimir(username, 95)
225
```

Graphite query

```
nPercentile(
    removeEmptySeries(
        mysql.monitoring.production.$persona.*.
        host.*.*.ten_secondly.user_connections.
        $username.count
    ), 95
)
```

Mimir query

```
quantile_over_time(0.95,
    mysql_info_schema_processlist_processes_by_user{
        booking_persona="$persona",
        mysql_user="$username"
    }[24h:$__interval]
)
```

Consumers: Reading metrics from Badmin

OUR APPROACH

Safe Transition to Reading from Mimir

- **reversible:** supports both backends, simple to switch between them
- **gradual:** low → medium → high criticality clusters
- **traceable:** logs show backend used; debug logs show queries & data

QUERY ROUTER

MetricsQueryRouter router for metrics queries

- python module name as input
- executes a Graphite or Mimir query
- routing is based on the name/criticality of the DB cluster
- configuration is stored in the DB
- no deployment for a configuration change

```
def get_previous_target_sizes(
    self,
    start_time_days_ago: int,
    end_time_days_ago: int,
) -> list:
    role = self.sdb_role
    router = MetricsQueryRouter(
        module=__name__,
        role=role,
        graphite_fn=self._get_previous_target_sizes_from_graphite,
        mimir_fn=self._get_previous_target_sizes_from_mimir,
    )
    return router.query(start_time_days_ago, end_time_days_ago)
```

Consumers: Updating the Grafana Dashboards

INVENTORY AUDIT

[illegible]

MIGRATION CHALLENGE

- **EOY 2026** is the deadline for Graphite
- Unknown volume
- Unknown complexity of the work
- Competing priorities
- Only dashboards owned by DBEs in scope

FIRST ACTIONS

- Identify the folders we use to store dashboards
- list all dashboards
- 181 dashboards found

Finding the dashboards we really use

METHODOLOGY

- Use AI (Grafana MCP) to **extract queries** from each dashboard
- **Filter and count** the Graphite queries (~2000)
- Leverage dashboard **view counts**
- **Manual inspection** to reduce the scope of work.

SCOPE ANALYSIS

- **MIGRATE = 32 dashboards**
(1,082 graphite queries)
- NO GRAPHITE QUERIES = 73
- NOT IN USE = 38
- OBSOLETE = 38

Top 10 Dashboards - Last 30 Days

FolderName	Name	dashboardUid	Views ↓
DQSV2			1723
DQSV2			775
DBE			500
DBE			416
DBE			359
DQSV2			311

PRIORITIZATION APPROACH

Start with the most important and complex queries:
order by (Views * Graphite queries) desc

Preparing for dashboards migration

Define Standards

(same look-and-feel)

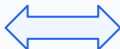
- **Switchable DS** – Prod / non-Prod
- **Variables** (label queries / ordering)
- **Data representation** – i.e. always “per second”, always `rate()`
- Dealing with **sparse values** (`last_over_time`)
- **Similar visuals** (legend, axis, etc)

Dashboard logistics

- Both versions available simultaneously
- Create and modify a **WIP copy**
- Side-by-side comparison
- Backup the original
- Publish WIP as new version (preserve the URL)
- Add banners, keep the backup for fallback
- Remove the WIP

This dashboard uses Mimir data sources

The backup of the old Graphite-based dashboard is available [here](#)



Warning

This is an outdated version of the dashboard that's based on Graphite data sources.

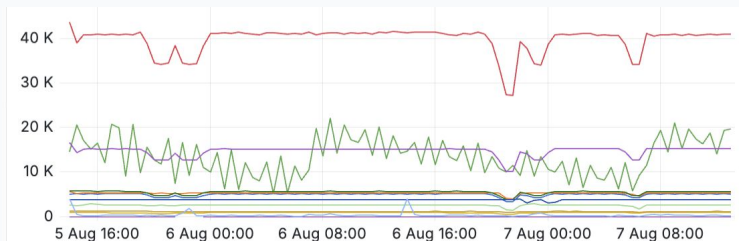
Use the new Mimir-based dashboard [here](#)

Updating dashboards manually (40% migrated)

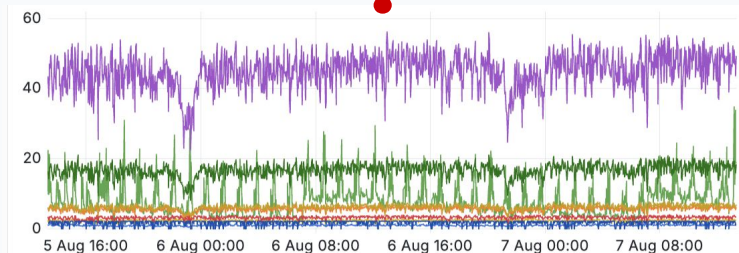
Manual Updates

- Very **slow process**
- Address panels **1-by-1**
- Steep **learning curve** to write queries correctly
 - data differences (granularity, sparsity)
 - data from multiple mimic tenants
 - bugs in Graphite queries (units, rate calculations)
- **AI help is limited**
 - Too much **manual inputs**
 - **Context size** limitation
 - **inconsistent results** (queries don't work, or get wrong data)
- Results are **not** always a **100% match**

VERIFICATION CHALLENGES



?



Using Observability team's AI skills (60% migrated)

AI skills:

- **survey:** Graphite metrics inventory
- **dashboard2mimir:** migrate a dashboard

DBEs used the dashboard migration skill extensively:

- DBEs were the **early adopters** and provided feedback to improve them
- **Generate metric mappings** from the manually converted dashboards
- The new dashboard standards provided as **operator context**
- **Visual side by side verification** was still required
- Provided metrics mappings to other teams consuming mysql metrics

~85%

Manual Work Reduction





Today's Journey

1. MySQL at Booking.com
2. Metrics Migration: Motivation, Challenges and Plan
3. Onboarding Producers
4. Migrating Consumers
- 5. Lessons Learned**
6. Q/A

Lessons Learned

1. Plan early and involve the right people
2. **Stable != Finished**: don't try to solve all problems at once (migrate, rewrite, prune)
3. **Accept discrepancies** when switching systems
4. At scale, **cardinality** and **rate limiting** become real
5. Agents/MCP servers/Skills take significant burden from you – **don't trust them blindly** (yet)



Questions?

Share your feedback



From Paths to Dimensions: How We Modernized Database
Observability at Scale

Backfilling the data to Mimir

Importing Old Data is Difficult

- Remote Write of old data to Mimir is not allowed
- Complicated & slow backfilling process
- Recording rules don't apply to backfilled data
- Aggregation is not possible during backfilling

Our Chosen Approach

- Identify essential metrics to store for 10y:
 - Number of hosts & pool sizing
 - Volume sizes / FS usage & backup sizes
- Set up recording rules to aggregate & record to **long-term-10y** tenant – preserves incoming data
- Run backfilling to retain full history

01. Create Mimir to **Mimir 10y recording rules**

02. **Pre-aggregate** data in **Graphite** to match the new recording rules for long term storage

03. Use purpose-built tooling to **export** data from Graphite **to OpenMetrics** files

04. Use **promtool** to **create TSDB blocks** from openmetrics files

05. **Split the blocks** into smaller 24h blocks to satisfy Mimir's compactor validation rules

06. Use **mimirtool** to **upload the blocks**

How to preserve old data?

Keeping Old Data

- Old data is useful: internal auditing, team forecasting, YoY growth inspection, capacity planning
- Metrics: host counts, space usage, backup sizes, pool sizing, etc.
- Graphite: up to 10 years for select namespaces
- Mimir: 400 days for all metrics by default

Retention Challenges

Extending default retention is inefficient:

- Query Performance & Execution Timeouts
- Cost Escalation (Storage, IOPS & egress)
- Metadata Synchronization & Compactor Bottlenecks
- Store-Gateway Index & RAM Footprint

10y Tenant Solution

Dedicated long-term tenant provided:

- Cannot be written to directly; accessible only via recording rules
- Restricted to aggregated or small-volume metrics
- Serves as a separate data source for dashboards

