Resources and Transactions

a fundamental duality in observability

QCon Plus

Track: “Observability and Understandability in Production”

May 18, 2021

Ben Sigelman
Co-founder and CEO, Lightstep
Co-creator of OpenTelemetry and OpenTracing
Co-creator of Dapper and Monarch at Google

Lightstep

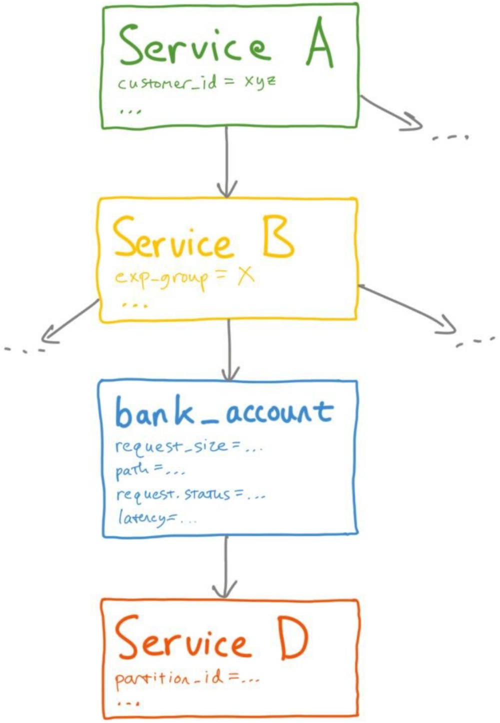
Part I

Transactions

Lightstep

What are “Transactions?”

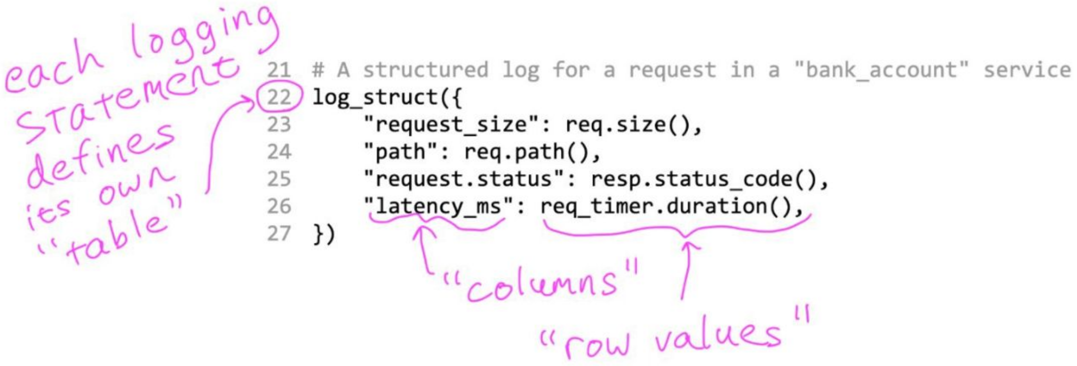
- The things in your app that **actually “do something”** for the end-user
- These days, they’re distributed
- Can be described at *many* granularities… For example:
  - End-user workflows in mobile apps, etc
  - HTTP requests in microservices
  - Local function calls
  - CPU instructions



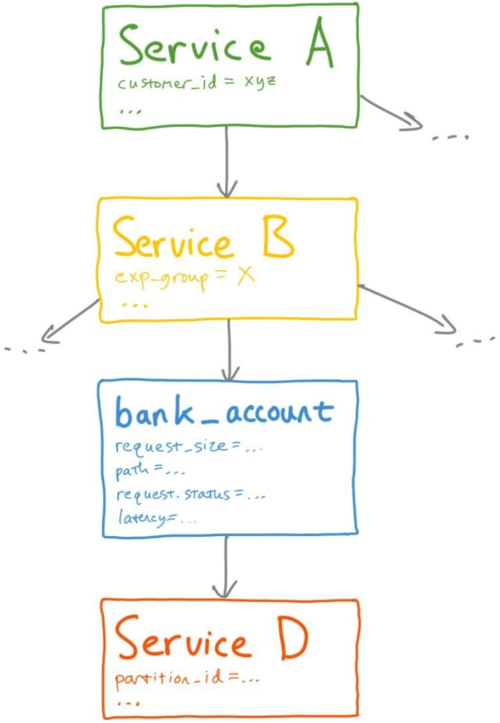
What are “Transactions?”

Telemetry: tracing and structured logs, with key:value attributes.

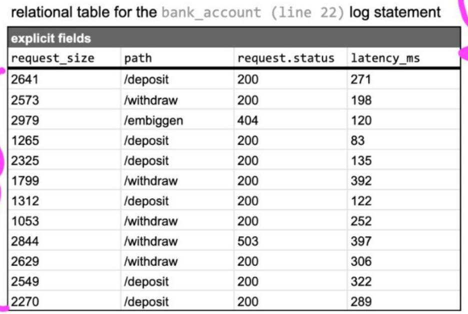
… and **tracing will eventually replace logging.**



```
# A structured log for a request in a "bank_account" service
log_struct({
    "request_size": req.size(),
    "path": req.path(),
    "request.status": resp.status_code(),
    "latency_ms": req_timer.duration(),
})
```



Tracing is just a JOIN across transactions!

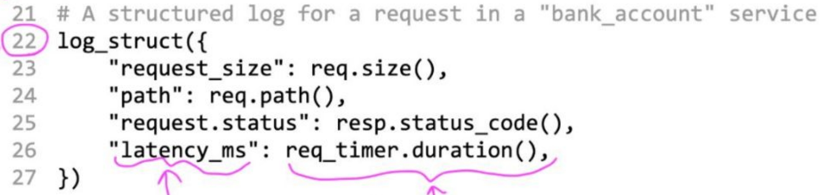


relational table for the `bank_account (line 22)` log statement

explicit fields			
request_size	path	request.status	latency_ms
2641	/deposit	200	271
2573	/withdraw	200	198
2979	/embiggen	404	120
1265	/deposit	200	83
2325	/deposit	200	135
1799	/withdraw	200	392
1312	/deposit	200	122
1053	/withdraw	200	252
2844	/withdraw	503	397
2629	/withdraw	200	306
2549	/deposit	200	322
2270	/deposit	200	289

fields from the log statement

values from the log statement



```
21  # A structured log for a request in a "bank_account" service
22  log_struct({
23      "request_size": req.size(),
24      "path": req.path(),
25      "request.status": resp.status_code(),
26      "latency_ms": req_timer.duration(),
27  })
```

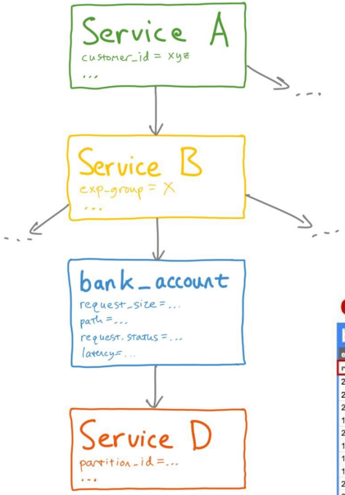
each logging statement defines its own "table"

"columns"

"row values"

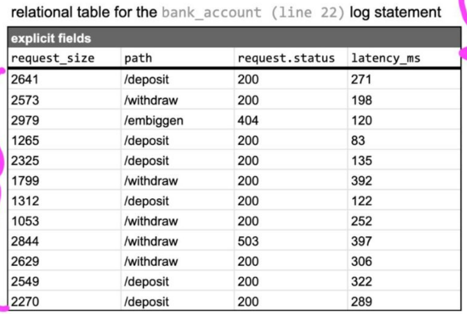
Tracing is just a JOIN across transactions!

Implementing "JOIN" via distributed Tracing



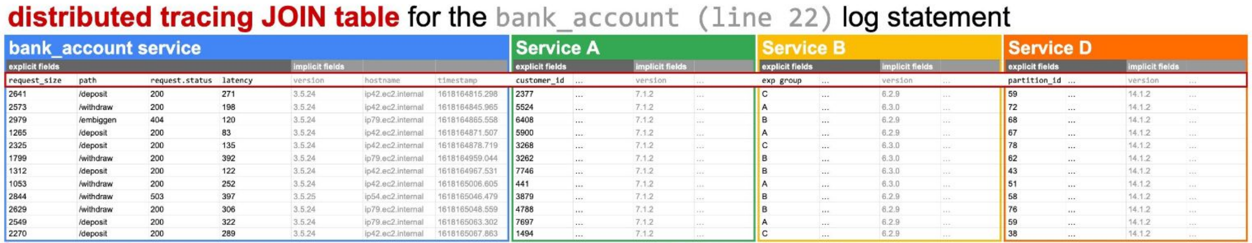
values from the log statement

fields from the log statement



relational table for the bank_account (line 22) log statement

explicit fields			
request_size	path	request.status	latency_ms
2641	/deposit	200	271
2573	/withdraw	200	198
2979	/embiggen	404	120
1265	/deposit	200	83
2325	/deposit	200	135
1799	/withdraw	200	392
1312	/deposit	200	122
1053	/withdraw	200	252
2844	/withdraw	503	397
2629	/withdraw	200	306
2549	/deposit	200	322
2270	/deposit	200	289



distributed tracing JOIN table for the bank_account (line 22) log statement

bank_account service						
explicit fields				implicit fields		
request_size	path	request.status	latency	version	hostname	timestamp
2641	/deposit	200	271	3.5.24	ip42.ec2.internal	1618164815.298
2573	/withdraw	200	198	3.5.24	ip42.ec2.internal	1618164845.965
2979	/embiggen	404	120	3.5.24	ip79.ec2.internal	1618164865.558
1265	/deposit	200	83	3.5.24	ip42.ec2.internal	1618164871.507
2325	/deposit	200	135	3.5.24	ip42.ec2.internal	1618164878.719
1799	/withdraw	200	392	3.5.24	ip79.ec2.internal	1618164959.044
1312	/deposit	200	122	3.5.24	ip42.ec2.internal	1618164967.531
1053	/withdraw	200	252	3.5.24	ip42.ec2.internal	1618165006.605
2844	/withdraw	503	397	3.5.25	ip54.ec2.internal	1618165046.479
2629	/withdraw	200	306	3.5.24	ip79.ec2.internal	1618165048.559
2549	/deposit	200	322	3.5.24	ip79.ec2.internal	1618165063.302
2270	/deposit	200	289	3.5.24	ip42.ec2.internal	1618165067.863

Service A				Service B				Service D			
explicit fields		implicit fields		explicit fields		implicit fields		explicit fields		implicit fields	
customer_id	...	version	...	exp group	...	version	...	partition_id	...	version	...
2377	...	7.1.2	...	C	...	6.2.9	...	59	...	14.1.2	...
5524	...	7.1.2	...	A	...	6.3.0	...	72	...	14.1.2	...
6408	...	7.1.2	...	B	...	6.2.9	...	68	...	14.1.2	...
5900	...	7.1.2	...	A	...	6.2.9	...	67	...	14.1.2	...
3268	...	7.1.2	...	C	...	6.3.0	...	78	...	14.1.2	...
3262	...	7.1.2	...	B	...	6.3.0	...	62	...	14.1.2	...
7746	...	7.1.2	...	B	...	6.3.0	...	43	...	14.1.2	...
441	...	7.1.2	...	A	...	6.3.0	...	51	...	14.1.2	...
3879	...	7.1.2	...	B	...	6.2.9	...	58	...	14.1.2	...
4788	...	7.1.2	...	B	...	6.2.9	...	76	...	14.1.2	...
7697	...	7.1.2	...	A	...	6.2.9	...	59	...	14.1.2	...
1494	...	7.1.2	...	C	...	6.2.9	...	38	...	14.1.2	...

Tracing automates a system-wide JOIN!

Lightstep

Part II

Resources

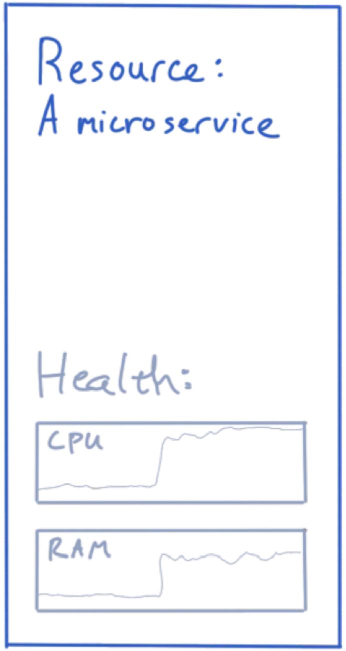
What are “Resources?”

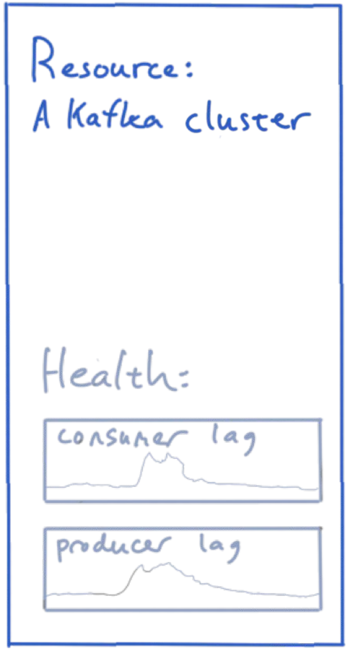
- **The things transactions *consume*** in order to do their work
  - Corollary: they are finite
- Resources **survive across transactions**
- Resources are **shared across transactions** (unless you have unlimited budget)
- Can also be described at *many* granularities… For example:
  - <Your AWS bill>
  - Kafka topic throughput
  - VM cpu usage or VM memory usage
  - Contention on a single mutex lock

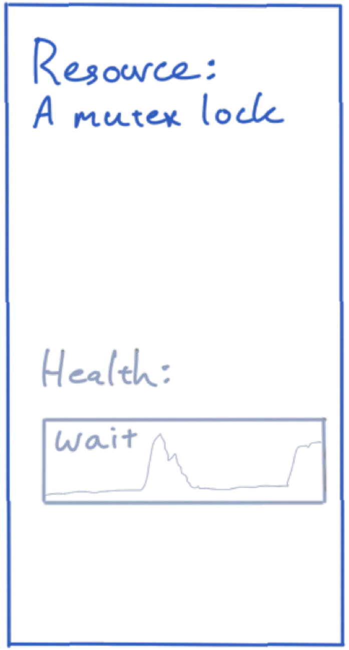
What are resources?

Resource:
A microservice
Resource:
A Kafka cluster
Resource:
A mutex lock

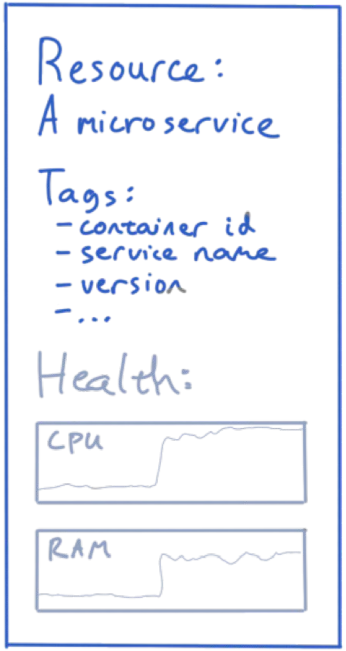
What are resources?





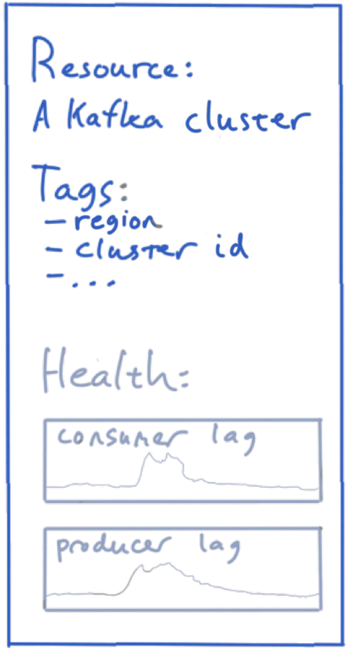


What are resources?



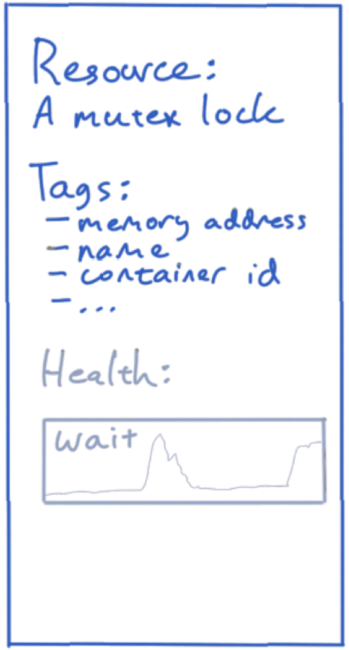
Resource:
A microservice

Tags:
- container id
- service name
- version
- ...



Resource:
A Kafka cluster

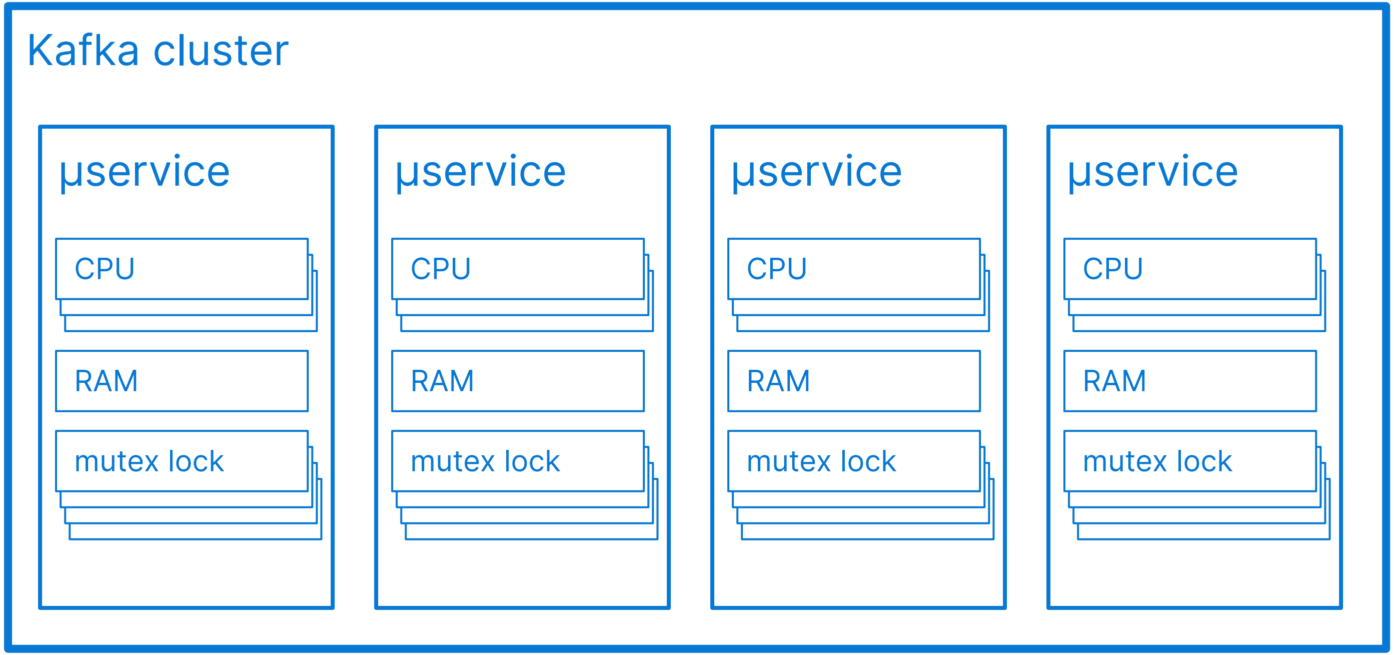
Tags:
- region
- cluster id
- ...



Resource:
A mutex lock

Tags:
- memory address
- name
- container id
- ...

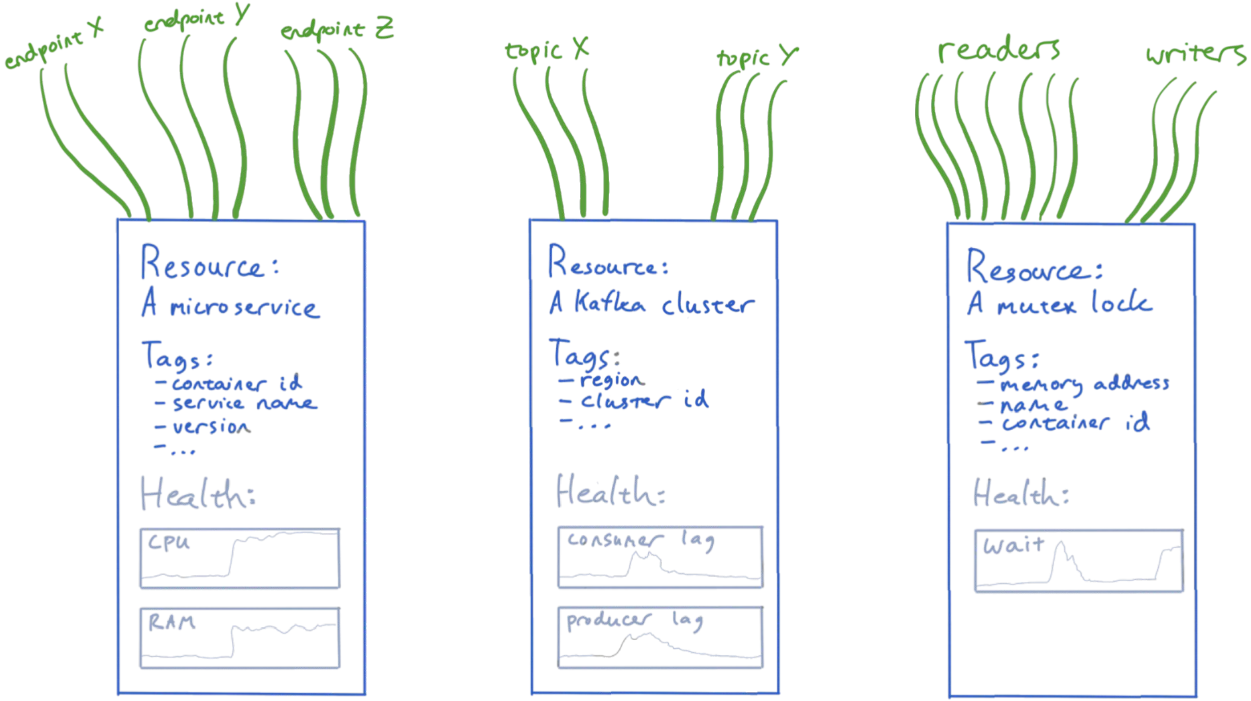
Resources are a hierarchy, too



Part III

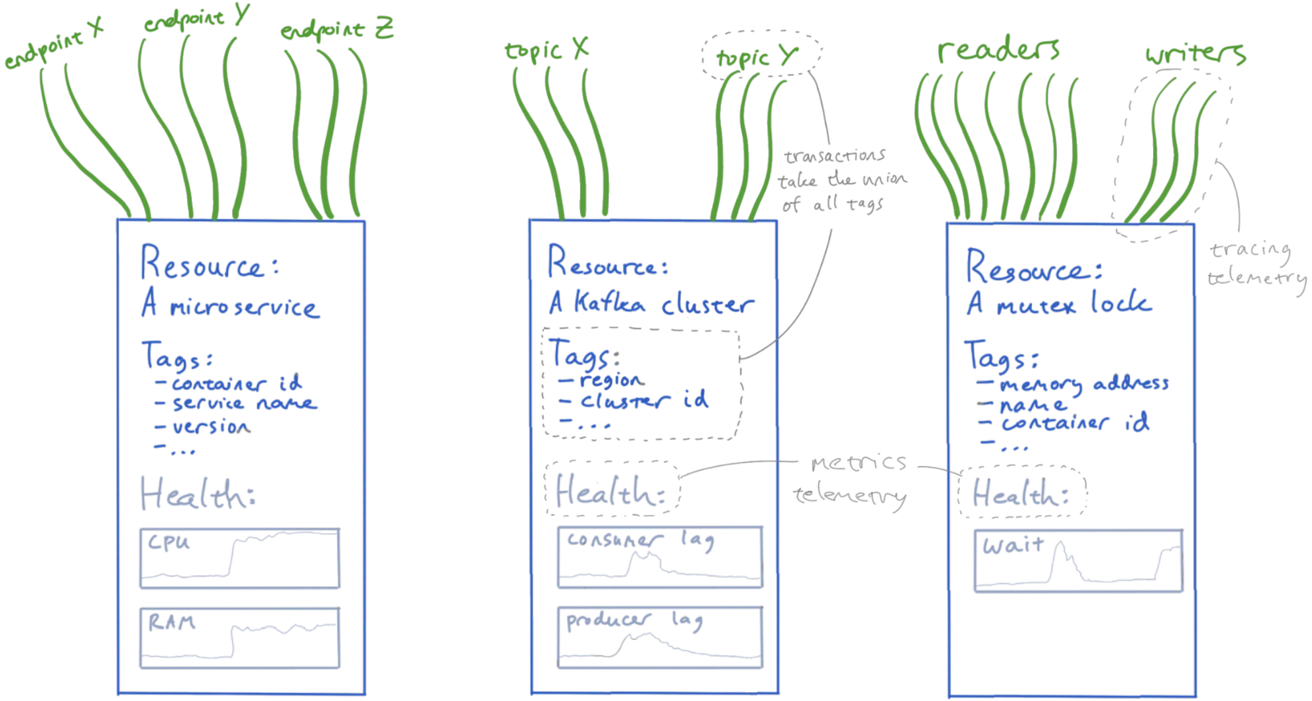
Co-dependency

Resources and transactions



Lightstep

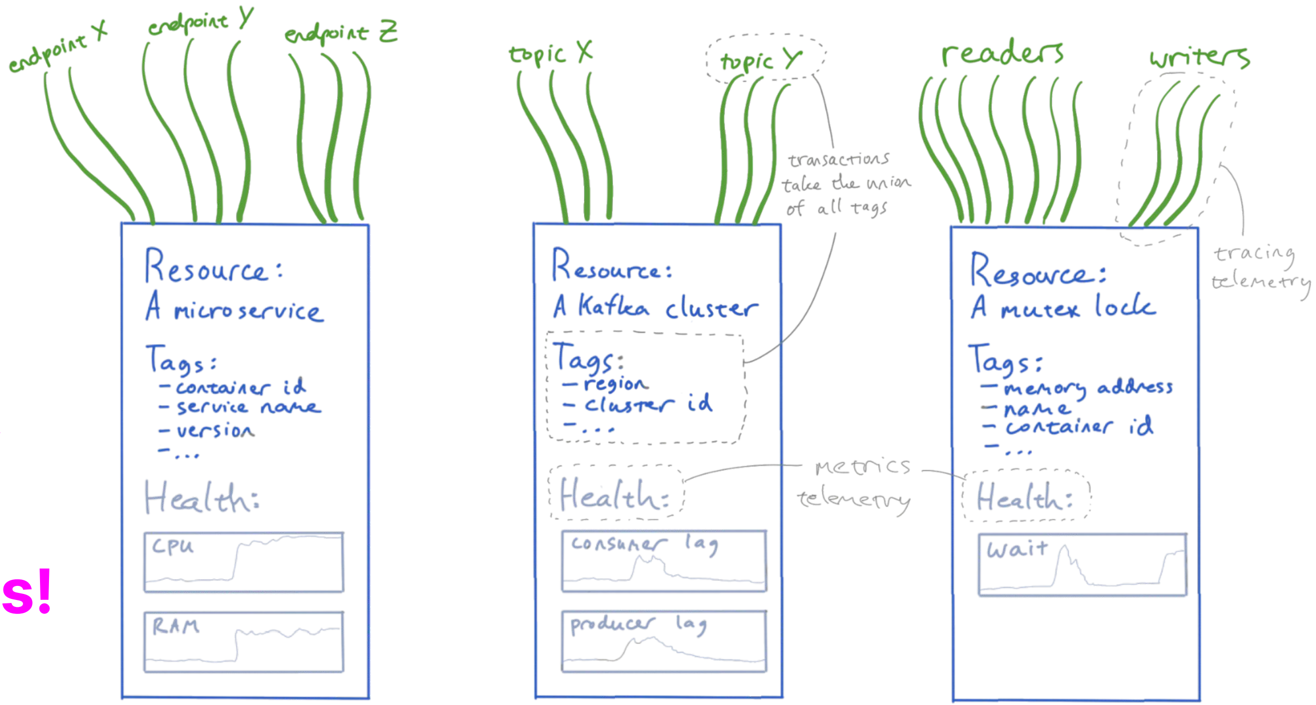
Resources and transactions



Lightstep

Resources and transactions are totally co-dependent

End-users only care about **Transactions!**



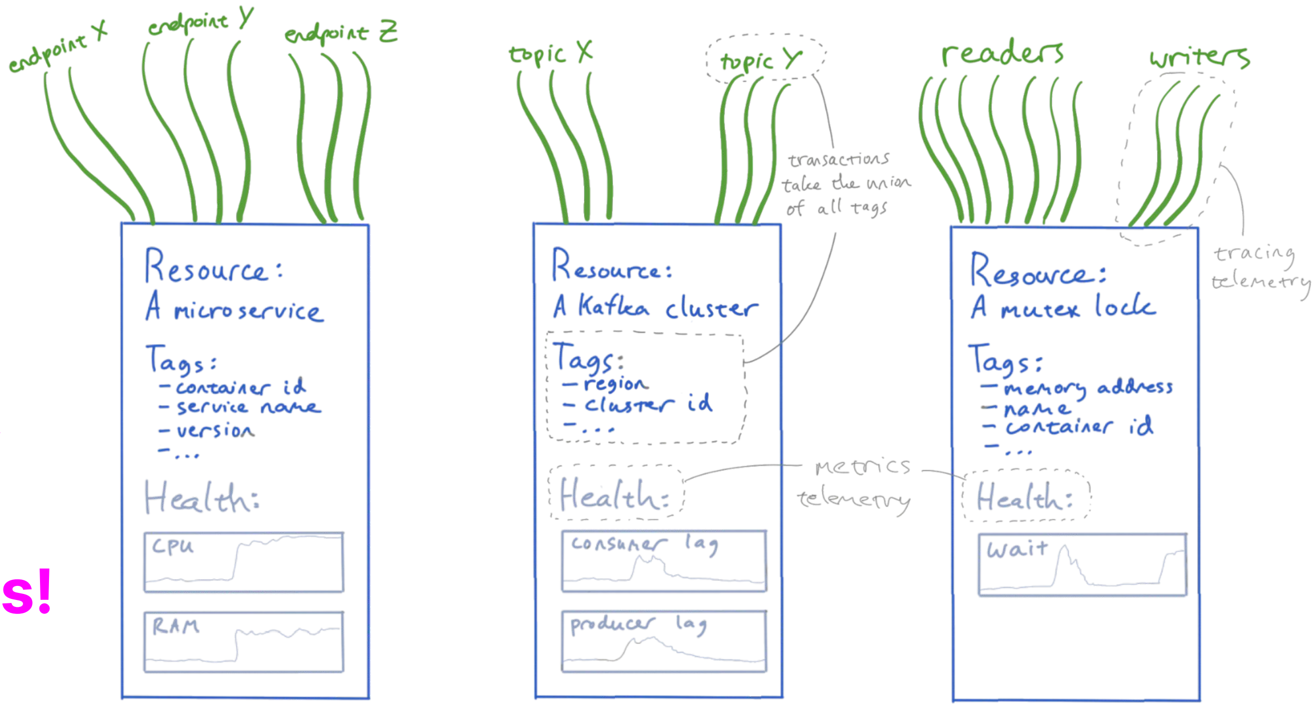
Operators only have control over **Resources!**

Resources and transactions are totally co-dependent

End-users and (dev)operators are totally co-dependent, too

End-users only care about **Transactions!**

Operators only have control over **Resources!**



“What’s a DevOps eng / SRE to do???”

- Must pivot between resources and transactions
- … and thus across telemetry types (metrics ↔ traces)
- … in the aggregate
- … via tags
- … automatically
- … and without becoming an observability tooling expert

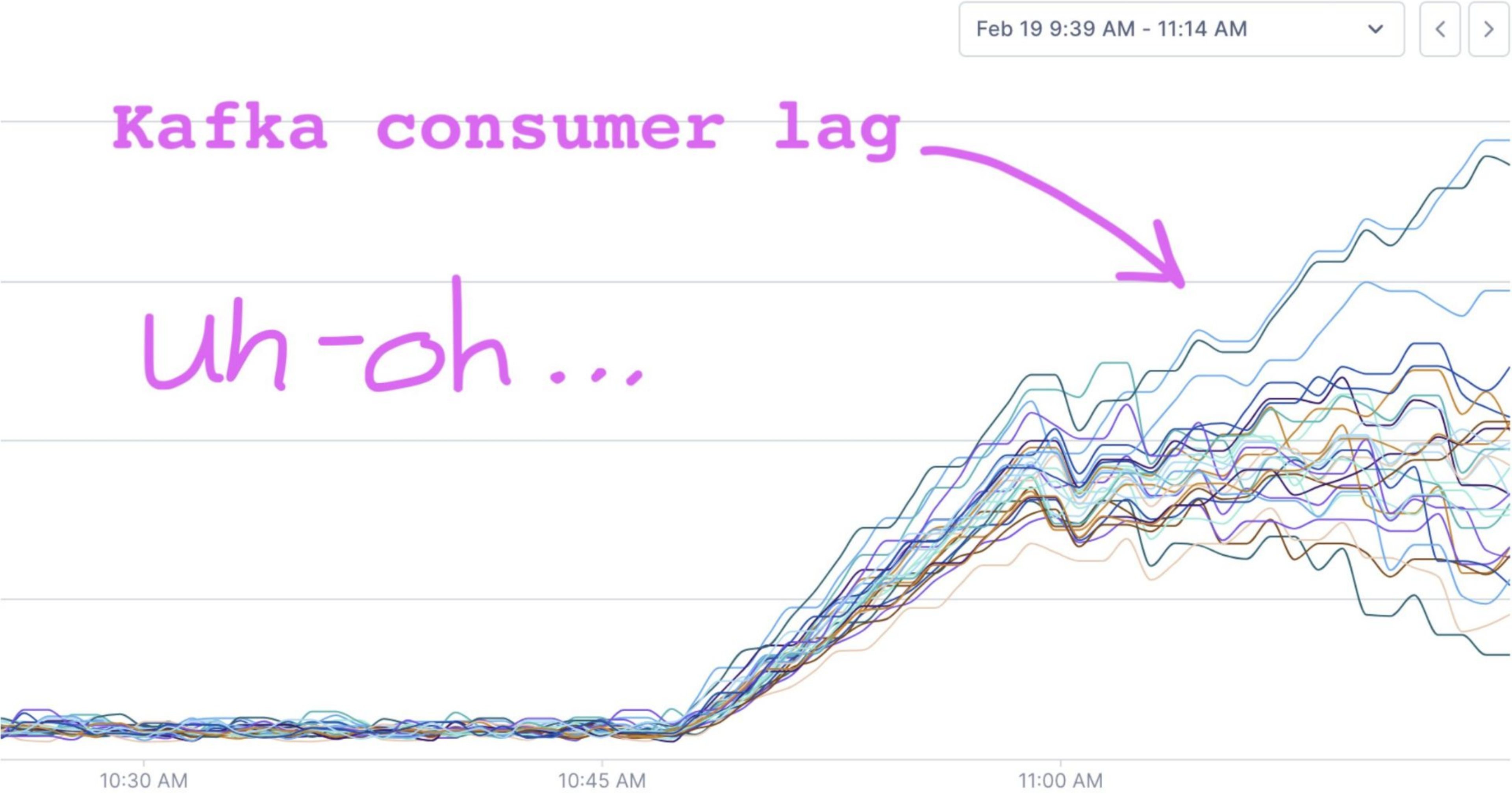
One helpful tool: **SLOs!**

- SLOs ("Service-Level Objectives") are a hot topic these days 😎
- SLOs are goals …
  - … about **a set of Transactions** (e.g., error ratios, latency, etc)
  - … scoped to **a set of Resources** (e.g., a microservice or DB)

In this way, **SLOs encompass Transactions, Resources, Operators, and End-Users.** No wonder they're so helpful for observability!

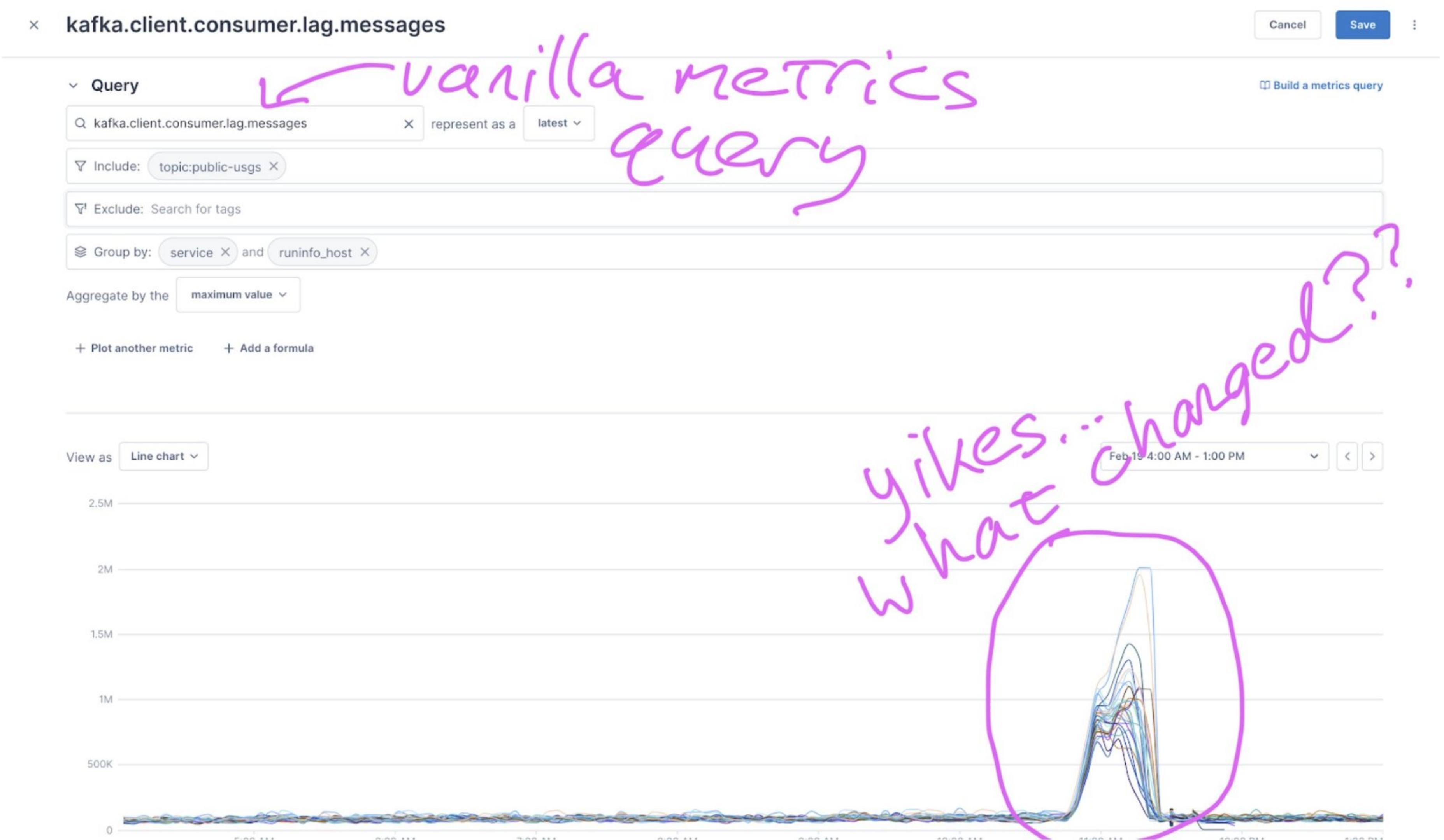
Part IV

What does this look like in practice?



Kafka consumer lag

Uh-oh...



× **kafka.client.consumer.lag.messages**

Cancel Save ⋮

⌄ **Query**

Build a metrics query

kafka.client.consumer.lag.messages ✕ represent as a latest ⌄

Include: topic:public-usgs ✕

Exclude: Search for tags

Group by: service ✕ and runinfo_host ✕

Aggregate by the maximum value ⌄

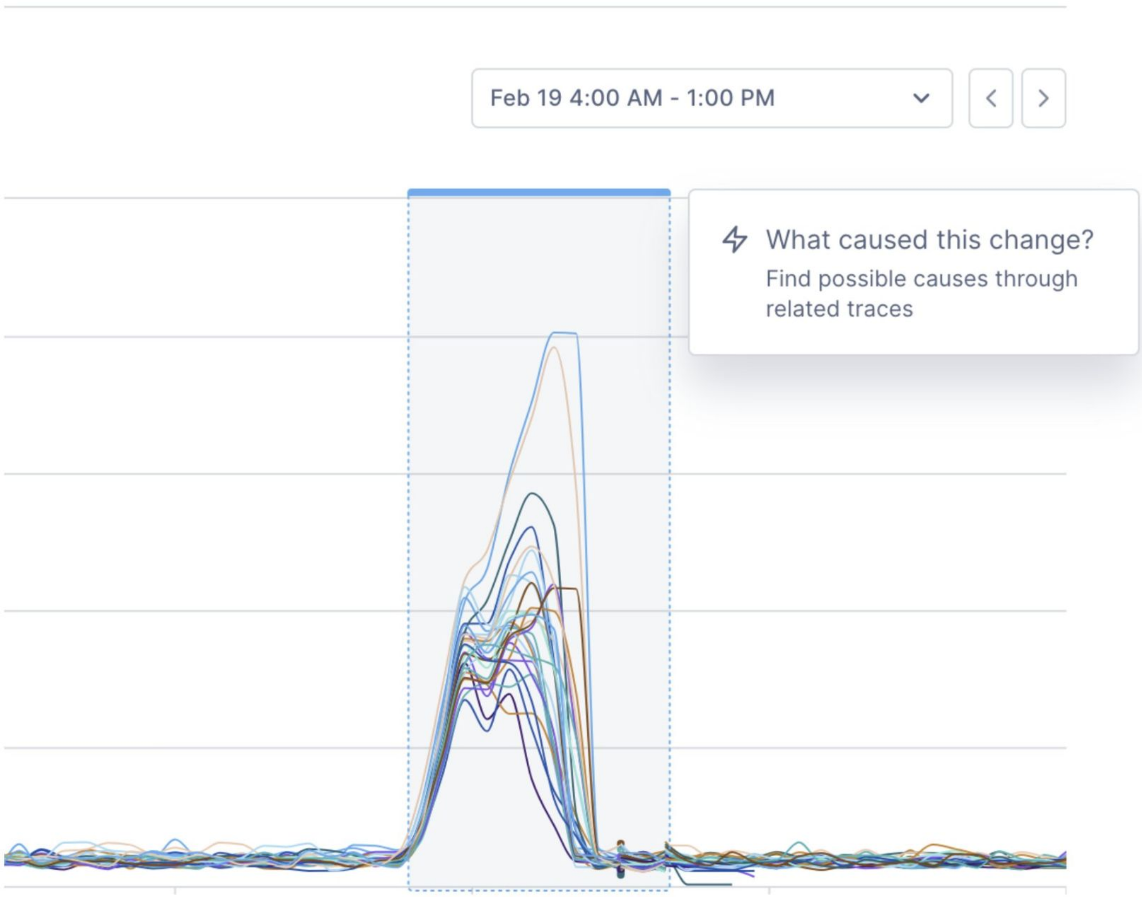
＋ Plot another metric ＋ Add a formula

View as Line chart ⌄

Feb 19 4:00 AM - 1:00 PM ⌄ ‹ ›

vanilla metrics query

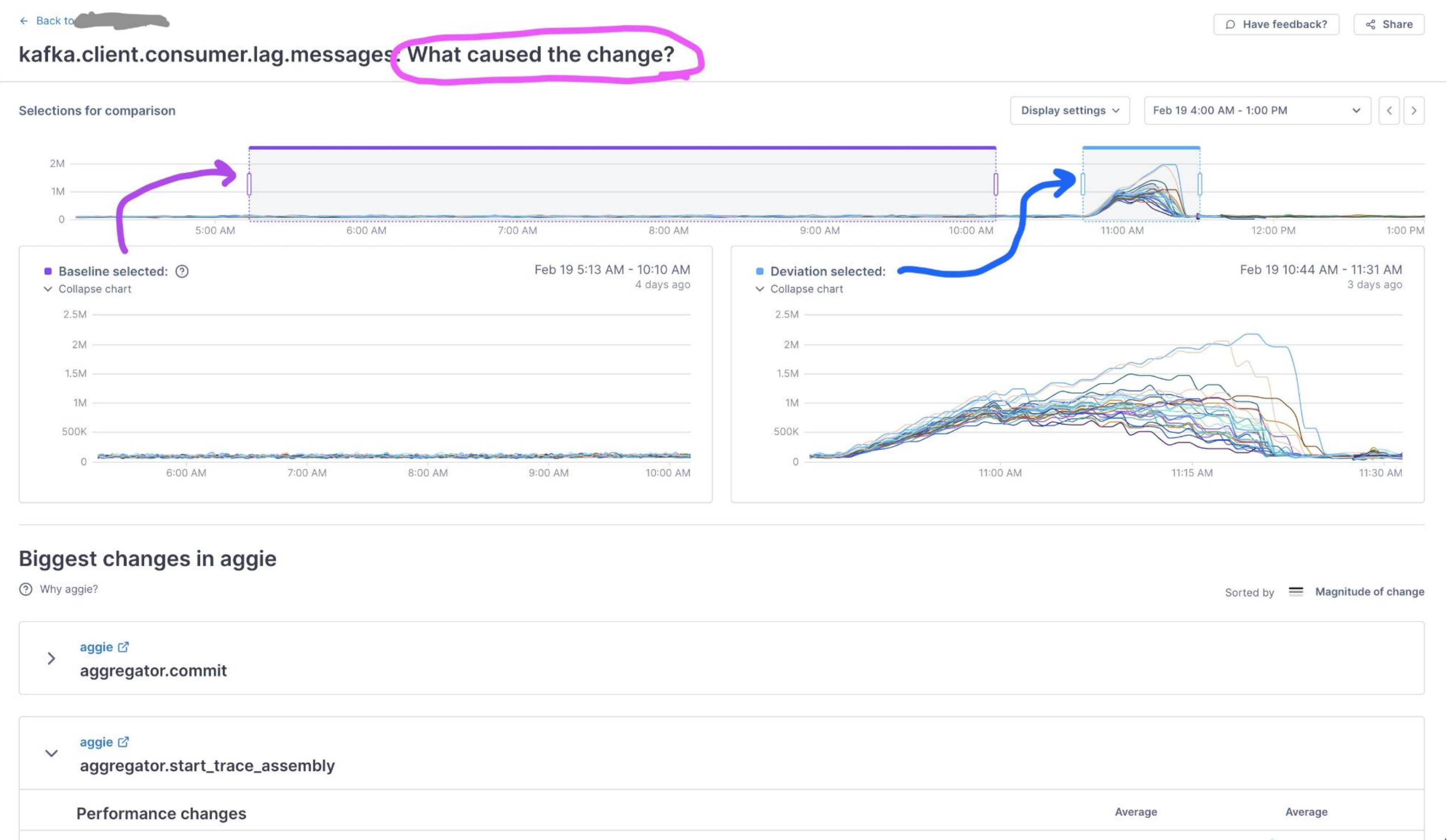
yikes... what changed?!



Feb 19 4:00 AM - 1:00 PM

What caused this change?

Find possible causes through related traces



← Back to [redacted]

kafka.client.consumer.lag.messages: What caused the change?

Selections for comparison

Display settings

Feb 19 4:00 AM - 1:00 PM

Baseline selected: Feb 19 5:13 AM - 10:10 AM, 4 days ago

Deviation selected: Feb 19 10:44 AM - 11:31 AM, 3 days ago

Biggest changes in aggie

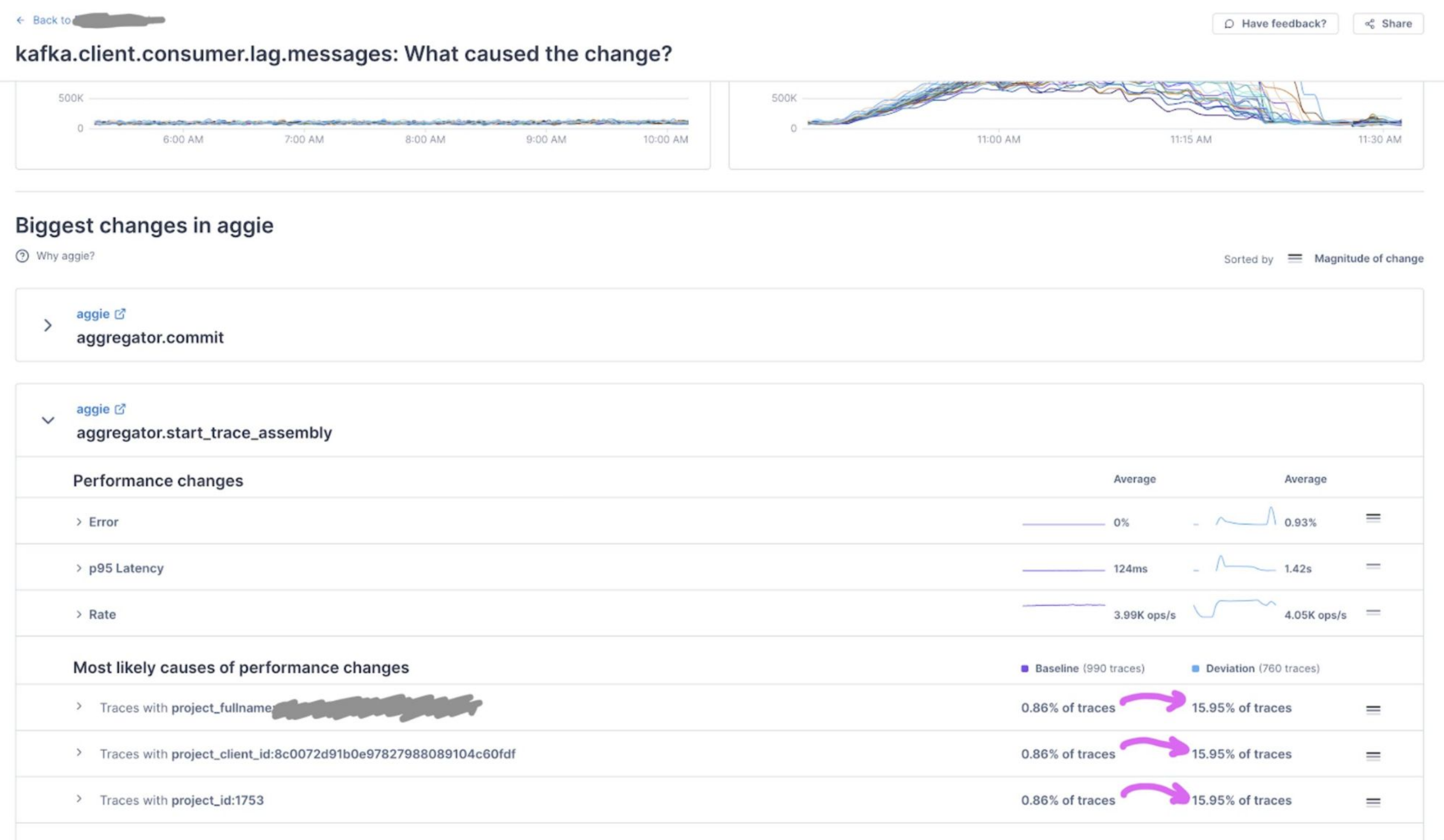
Why aggie?

Sorted by Magnitude of change

aggie
aggregator.commit

aggie
aggregator.start_trace_assembly

Performance changes



← Back to [redacted]

Have feedback? Share

kafka.client.consumer.lag.messages: What caused the change?

Biggest changes in aggie

Why aggie?

Sorted by Magnitude of change

aggie

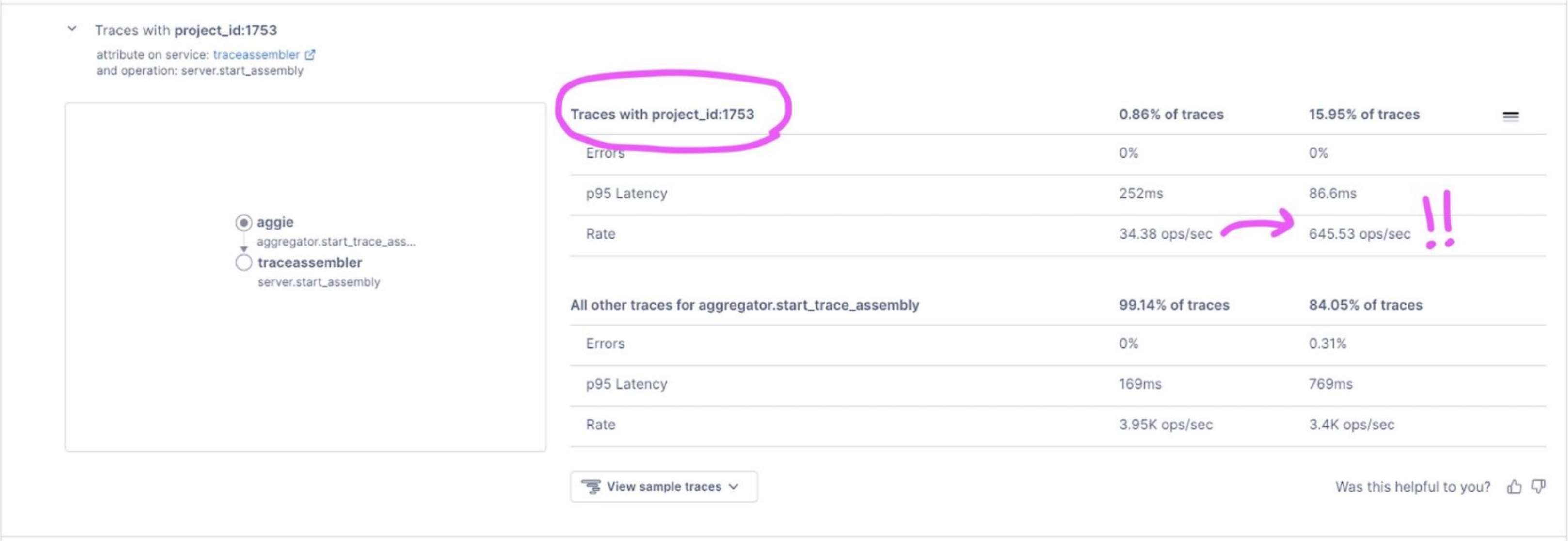
aggregator.commit

aggie

aggregator.start_trace_assembly

Performance changes	Average	Average
Error	0%	0.93%
p95 Latency	124ms	1.42s
Rate	3.99K ops/s	4.05K ops/s

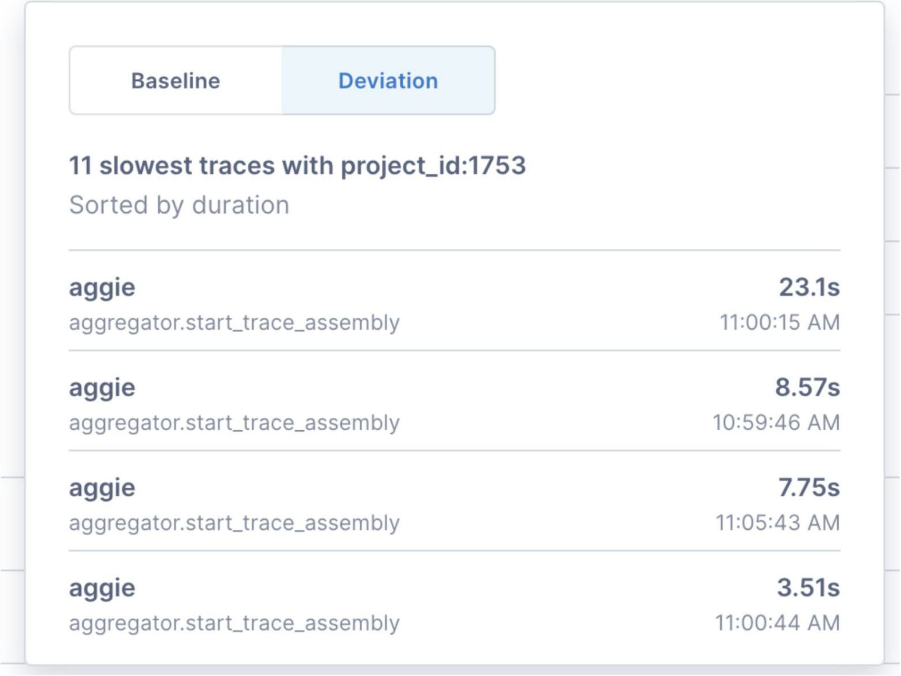
Most likely causes of performance changes	Baseline (990 traces)	Deviation (760 traces)
Traces with project_fullname:[redacted]	0.86% of traces	15.95% of traces
Traces with project_client_id:8c0072d91b0e97827988089104c60fdf	0.86% of traces	15.95% of traces
Traces with project_id:1753	0.86% of traces	15.95% of traces



Traces with **project_id:1753**

attribute on service: traceassembler
and operation: server.start_assembly

Traces with project_id:1753	0.86% of traces	15.95% of traces
Errors	0%	0%
p95 Latency	252ms	86.6ms
Rate	34.38 ops/sec →	645.53 ops/sec !!
All other traces for aggregator.start_trace_assembly	**99.14% of traces**	**84.05% of traces**
Errors	0%	0.31%
p95 Latency	169ms	769ms
Rate	3.95K ops/sec	3.4K ops/sec



11 slowest traces with project_id:1753

Sorted by duration

aggie aggregator.start_trace_assembly	23.1s 11:00:15 AM
aggie aggregator.start_trace_assembly	8.57s 10:59:46 AM
aggie aggregator.start_trace_assembly	7.75s 11:05:43 AM
aggie aggregator.start_trace_assembly	3.51s 11:00:44 AM

Part V

Summing up...

- Transactions **traverse systems** and **use Resources**
- Users don’t give a s**t about your Resources
- DevOps can’t **do** s**t about individual Transactions
- We *must* be able to join Resources and Transactions to address the most important question in observability:

“What caused that change?”