

Using Serverless & arm64 for Real-time Observability

Liz Fong-Jones
Field CTO, Honeycomb



Today

How serverless is useful for on-demand compute

How serverless is painful for on-demand compute

How to experiment with serverless in your environment

Deploying applications (serverless & serverful) on more cost-effective & sustainable arm64

What is Lambda for?

Let's talk use cases of serverless

What is ~~Lambda~~ for?

We'd like to optimize our custom datastore, Retriever

What is Retriever for?

It's a distributed column store for **real-time event aggregation**



What is ~~Retriever~~ for?

Real-time event aggregation for interactive querying over traces





Liz Fong-Jones

Field CTO, Honeycomb
AWS Community Hero

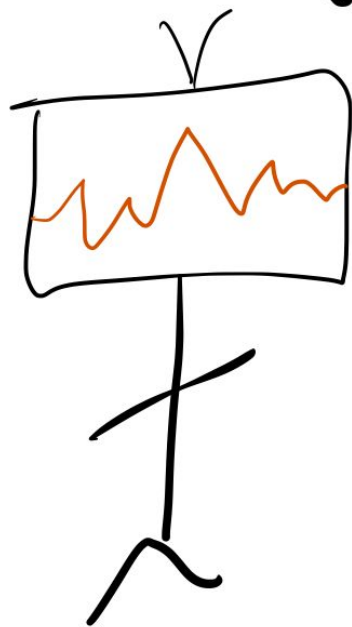


What is Honeycomb for?

Observability: finding out what is going on
(by querying traces!)



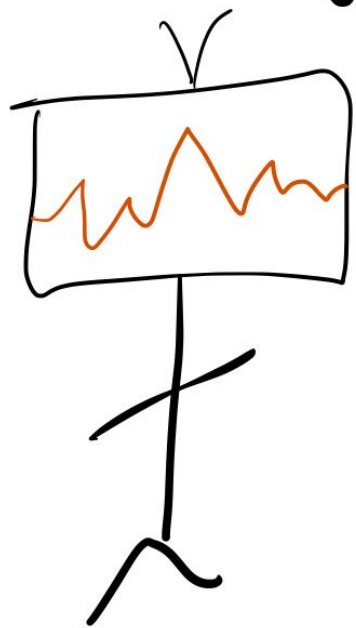
Monitoring



what do you
want to
watch for?

I'll count it up
and make
those graphs fast!

Monitoring



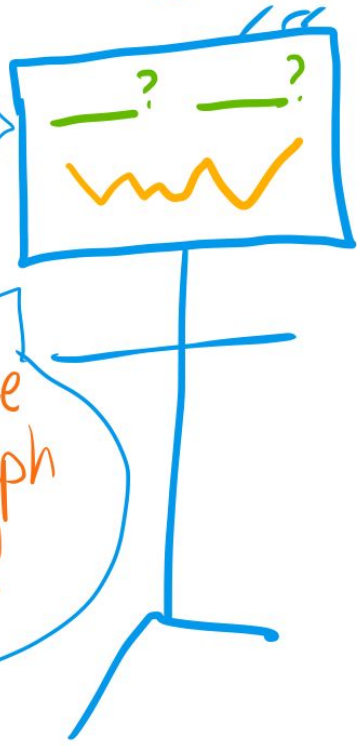
what do you want to watch for?

I'll count it up and make those graphs fast!

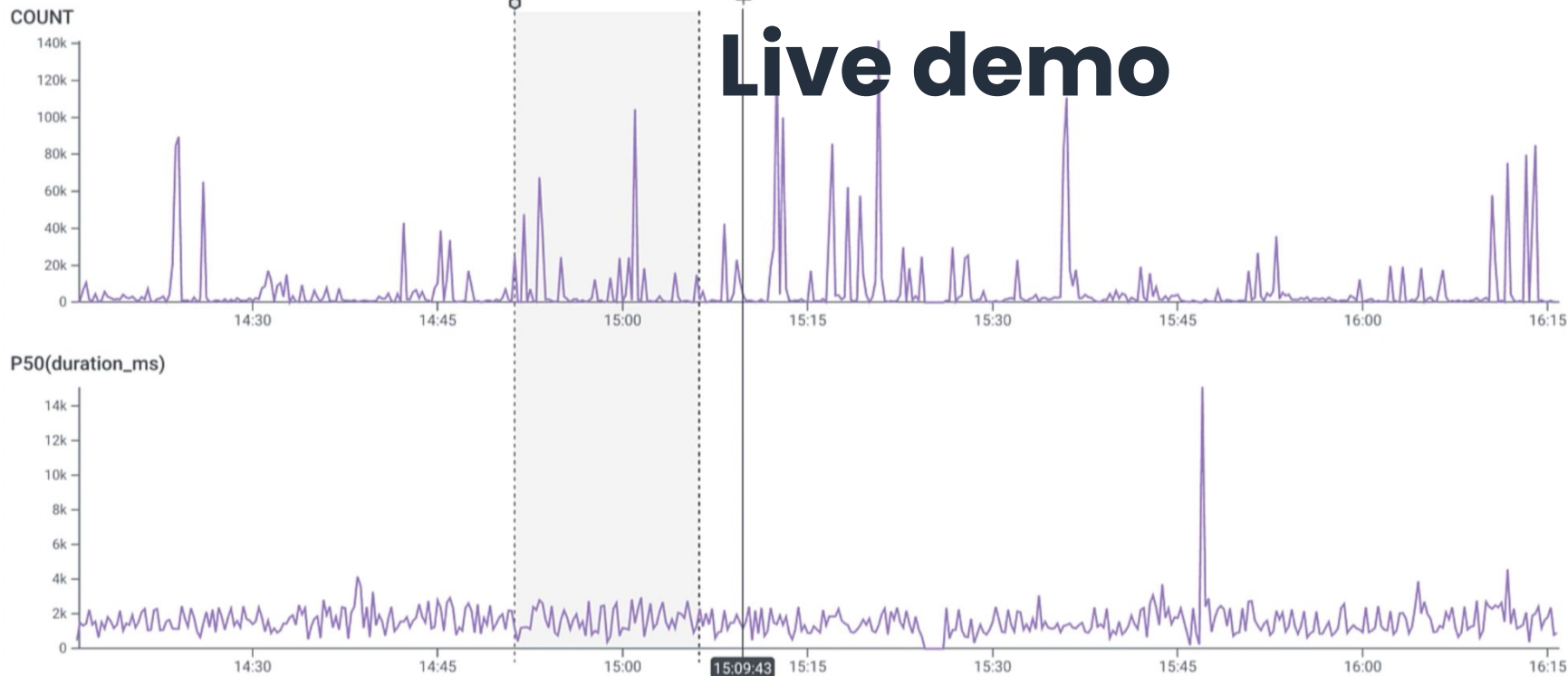
Observability

what do you want to see right now?

I'll make any graph fast!



Oct 7 2022, 2:15:57 PM – Oct 7 2022, 4:15:57 PM (Granularity: 15 sec)



name

COUNT



P50(duration_ms)



Invoke

3,382,181

1,601.72182

elapsed query time: 3.063187812s rows examined: 600,077,969 nodes reporting: 100%

Interactive investigation of production behavior

We run fast queries across any combination of fields.

DATA GENERATION

STREAMING INGEST

STORAGE + PROCESSING

VISUAL ANALYSIS LOOP

Logging Data



Amazon Relational Database Service (Amazon RDS)



AWS Elastic Load Balancing



AWS Elastic Beanstalk



Amazon Elastic Kubernetes Service (Amazon EKS)

Realtime Ingest
Auth and validation

Metrics Data

Host Metrics

App Metrics



Amazon CloudWatch

Prometheus

OpenTelemetry

Honeycomb API

Unpacked
Into
Columns

Trace Span Data

JavaScript

Go

Jaeger

Java

Ruby

Python

.NET

Front-End

OpenTelemetry

Refinery
*User-controlled
dynamic tail sampling*

Query Engine

Unlimited users can store thousands of dimensions at no additional cost and query any arbitrary combinations without pre-aggregation.

Proprietary distributed computing and parallelized processing returns query results in < 3 seconds across billions of rows of data.

Columnar Datastore

High-cardinality
column file

Segment with
time range



Amazon Simple Storage Service (Amazon S3)

Context
Context
Context
Context
Context
Context
Context
Context

Column
Column
Column
Column
Column
Column
Column
Column

Segment
Segment
Segment
Segment
Segment
Segment
Segment
Segment



AWS Lambda

ui.honeycomb.io

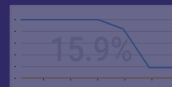
ui.honeycomb.io

ui.honeycomb.io

Events contain many fields and distinct values

Context
Context
Context
Context
Context
Context
Context
Context

"Wide events" packed with as much context as you need for debugging



Graph Rendering
Click through
visuals based on
granular data



Query Builder UI
Intuitive GUI with
multiple group-by



Query Result History
Shared team
intelligence



DATA GENERATION

STREAMING INGEST

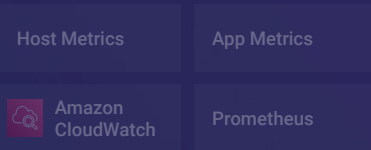
STORAGE + PROCESSING

VISUAL ANALYSIS LOOP

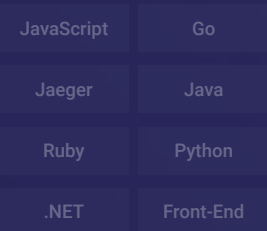
Logging Data



Metrics Data



Trace Span Data



Realtime Ingest

Auth and validation

Honeycomb API

Unpacked
Into
Columns

Refinery

User-controlled
dynamic tail sampling

Query Engine

Unlimited users can store thousands of dimensions at no additional cost and query any arbitrary combinations without pre-aggregation.

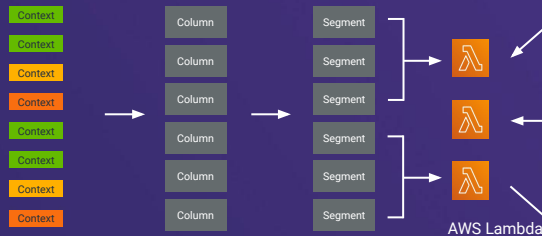
Proprietary distributed computing and parallelized processing returns query results in < 3 seconds across billions of rows of data.

Columnar Datastore

High-cardinality
column file

Segment with
time range

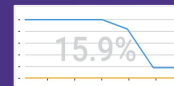
Amazon Simple Storage
Service (Amazon S3)



Events contain many
fields and distinct
values



"Wide events" packed with
as much context as you
need for debugging



Graph Rendering

Click through
visuals based on
granular data



Query Builder UI

Intuitive GUI with
multiple group-by



Query Result History

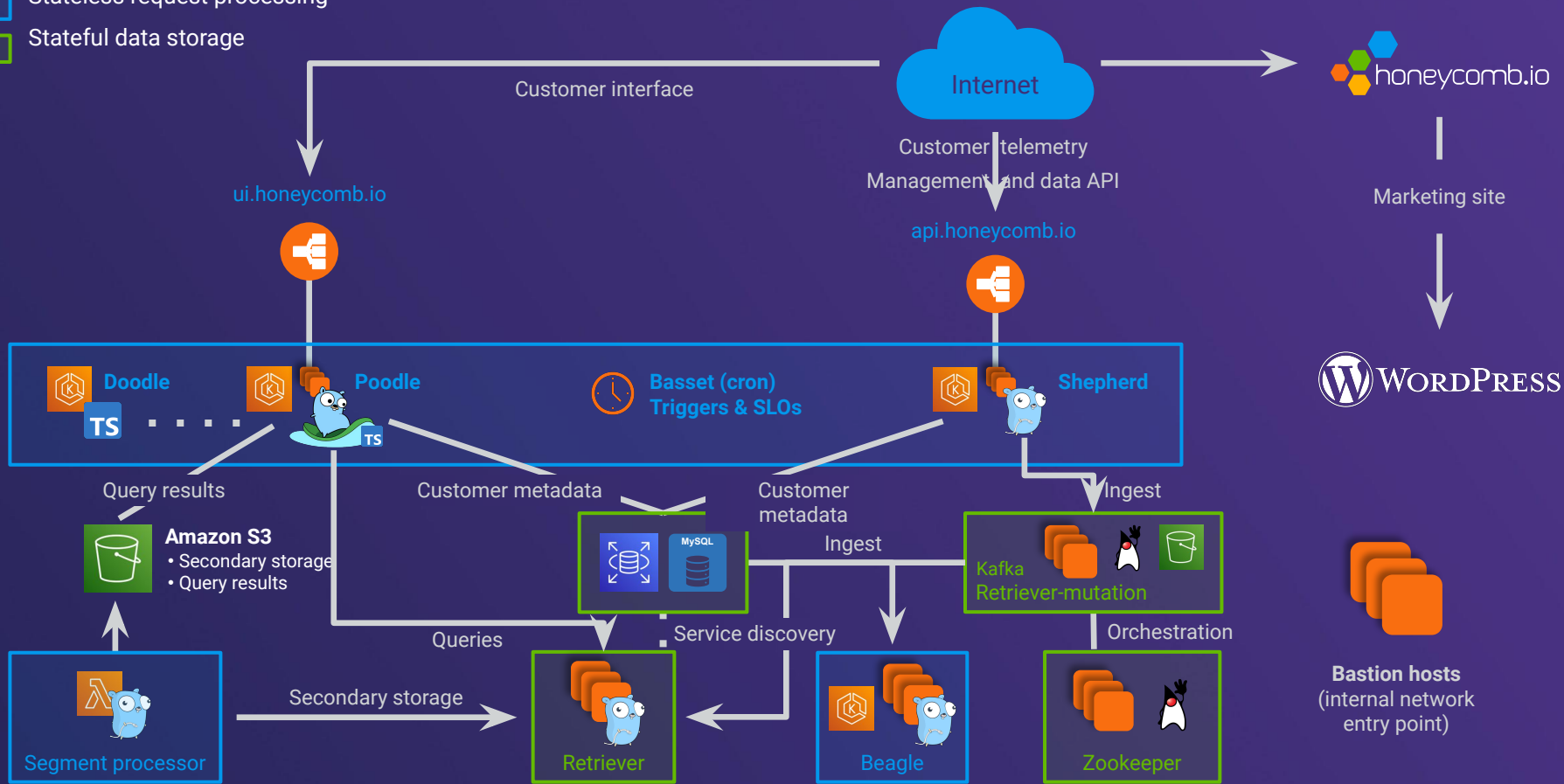
Shared team
intelligence



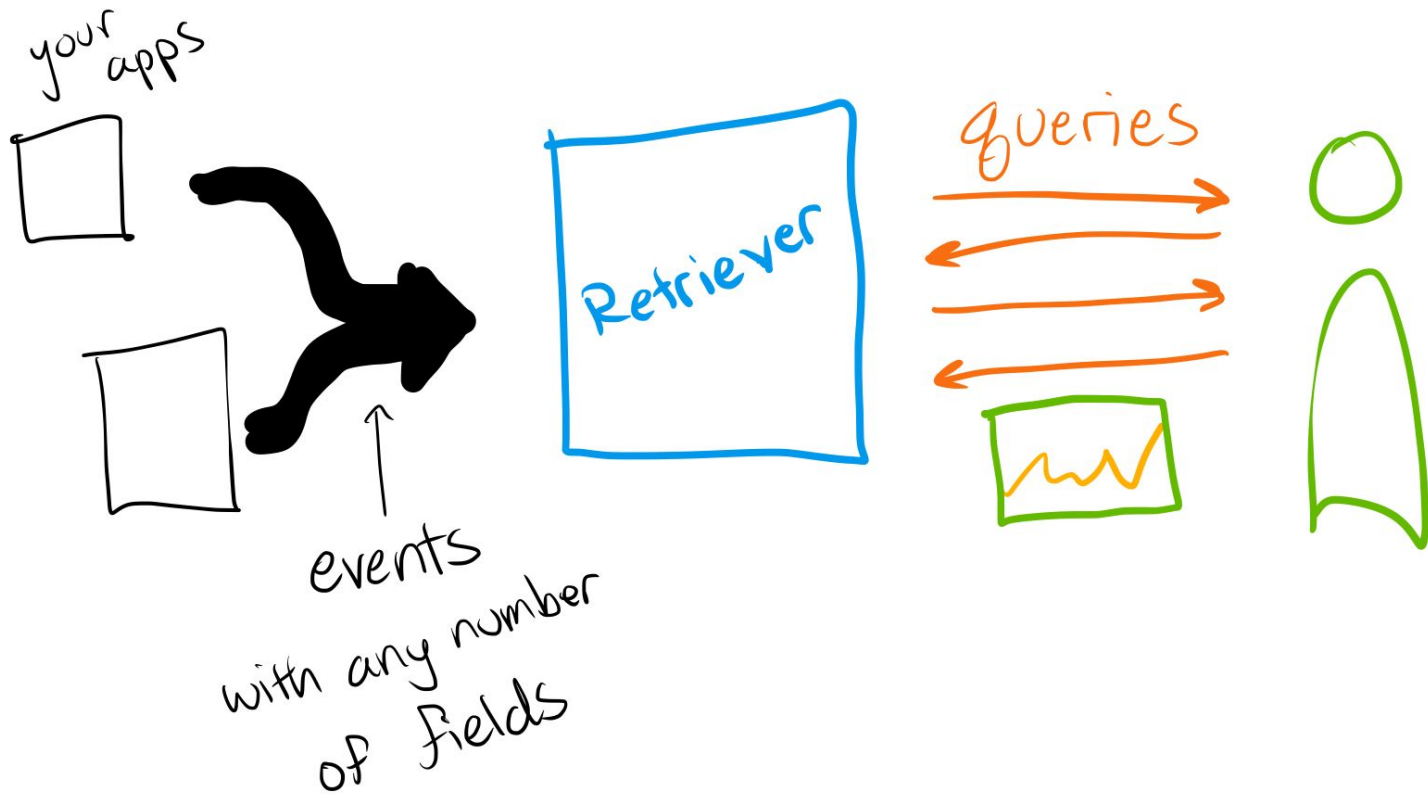
Emphasis: *interactive*.

100ms is fast. 1000ms is ok. 10sec is slow. 100sec is unacceptable.

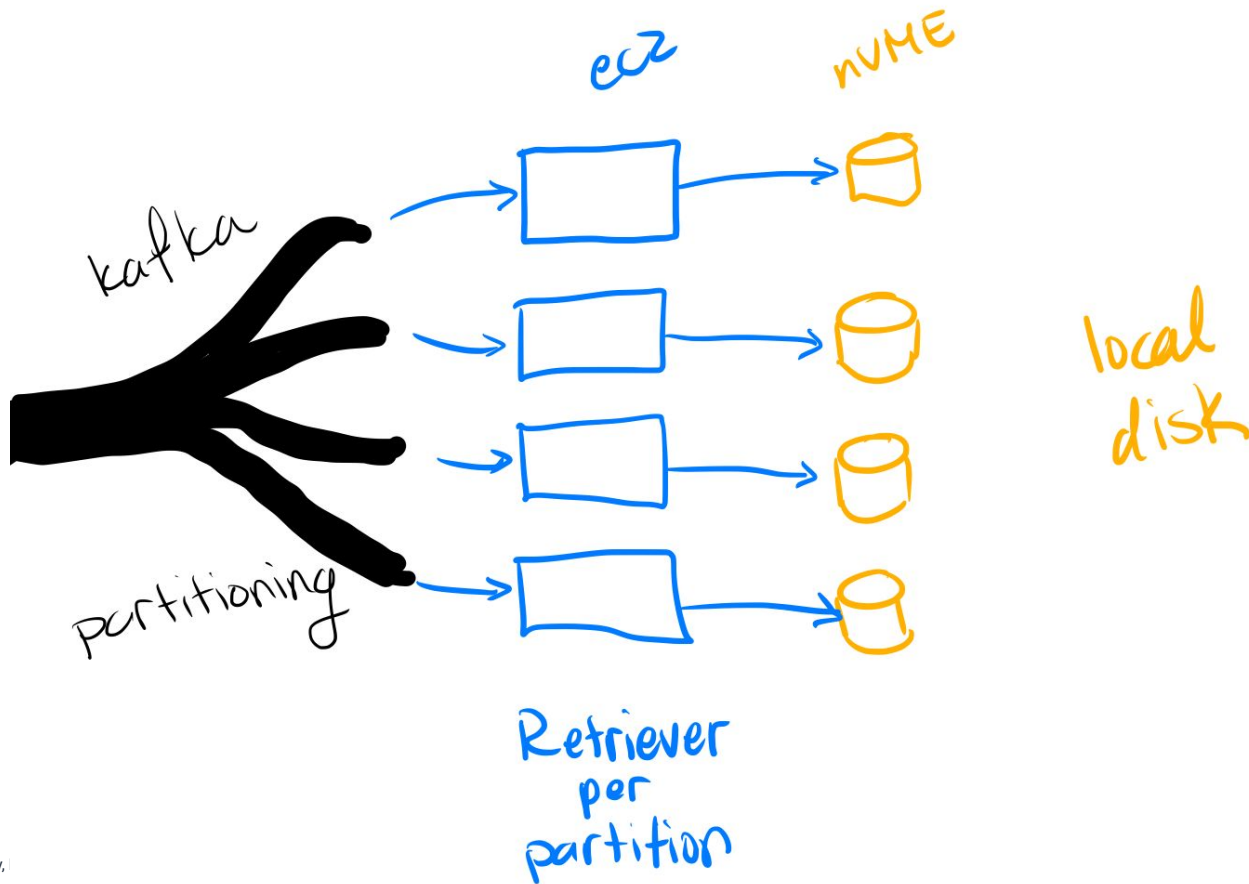
- Stateless request processing
- Stateful data storage



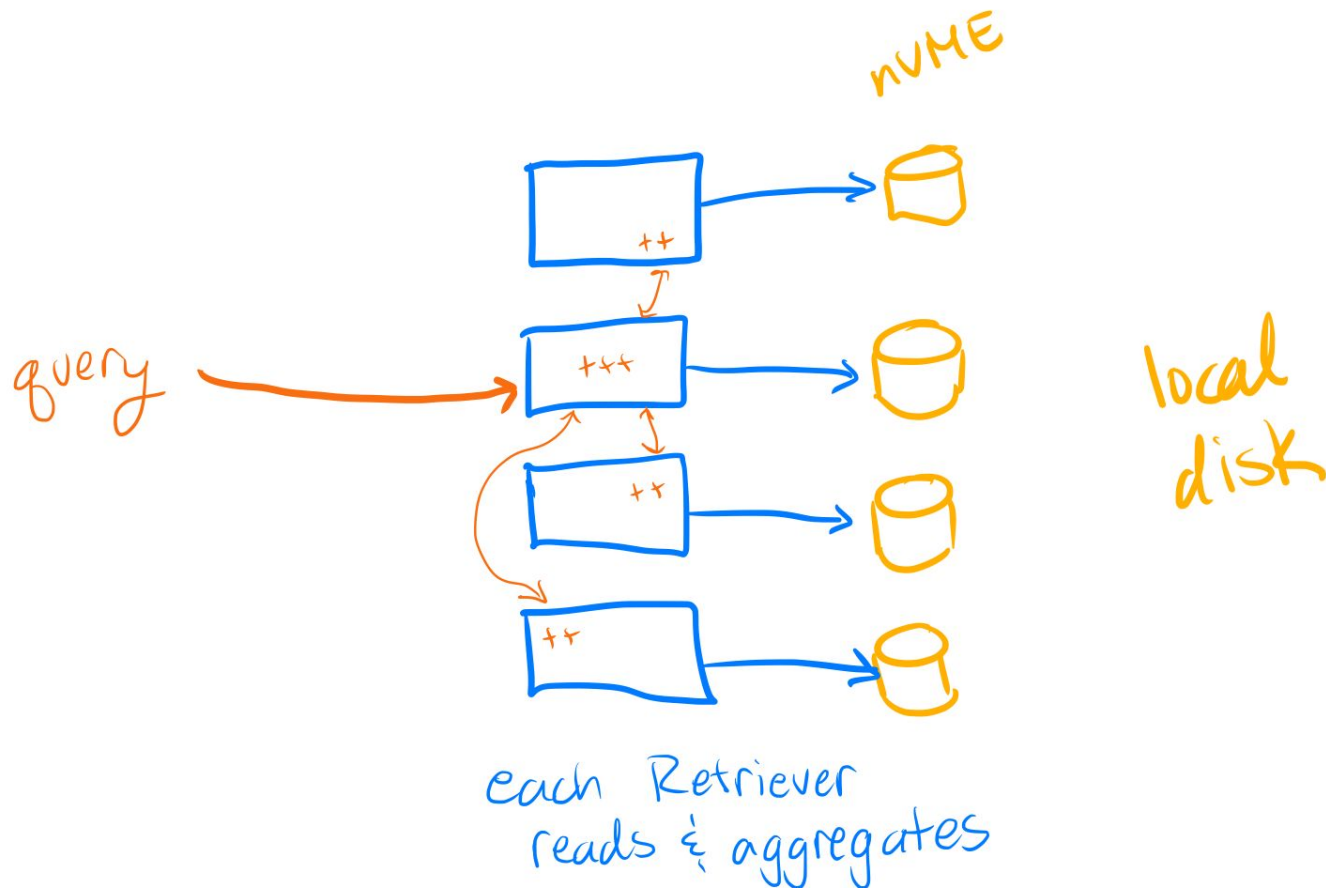
Retriever stores all your event data



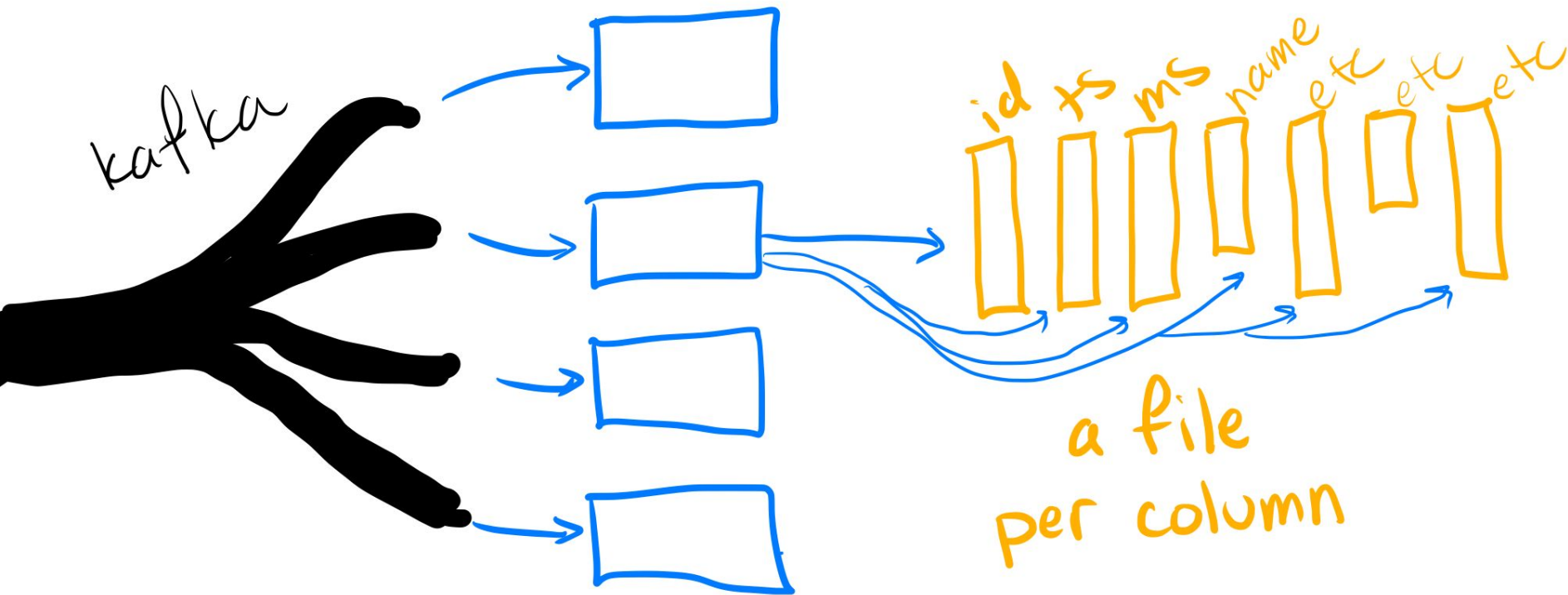
Retriever is a distributed datastore.



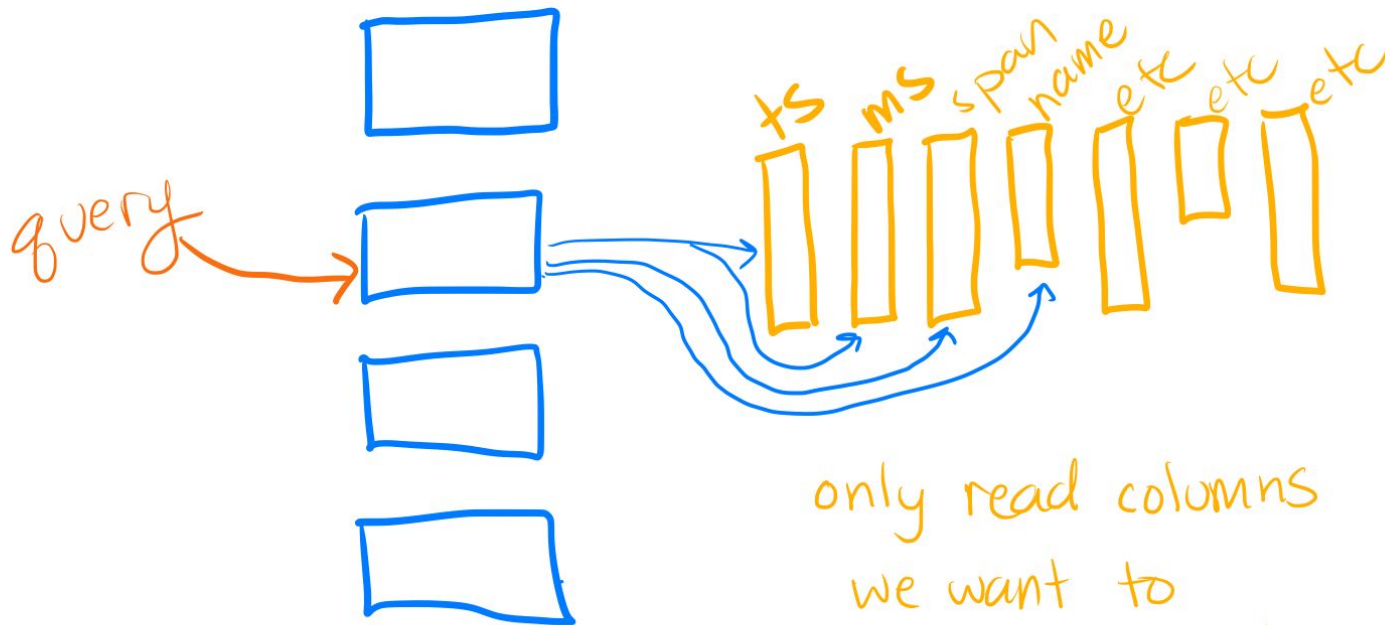
Retriever is a distributed datastore.



Retriever is a distributed column store.

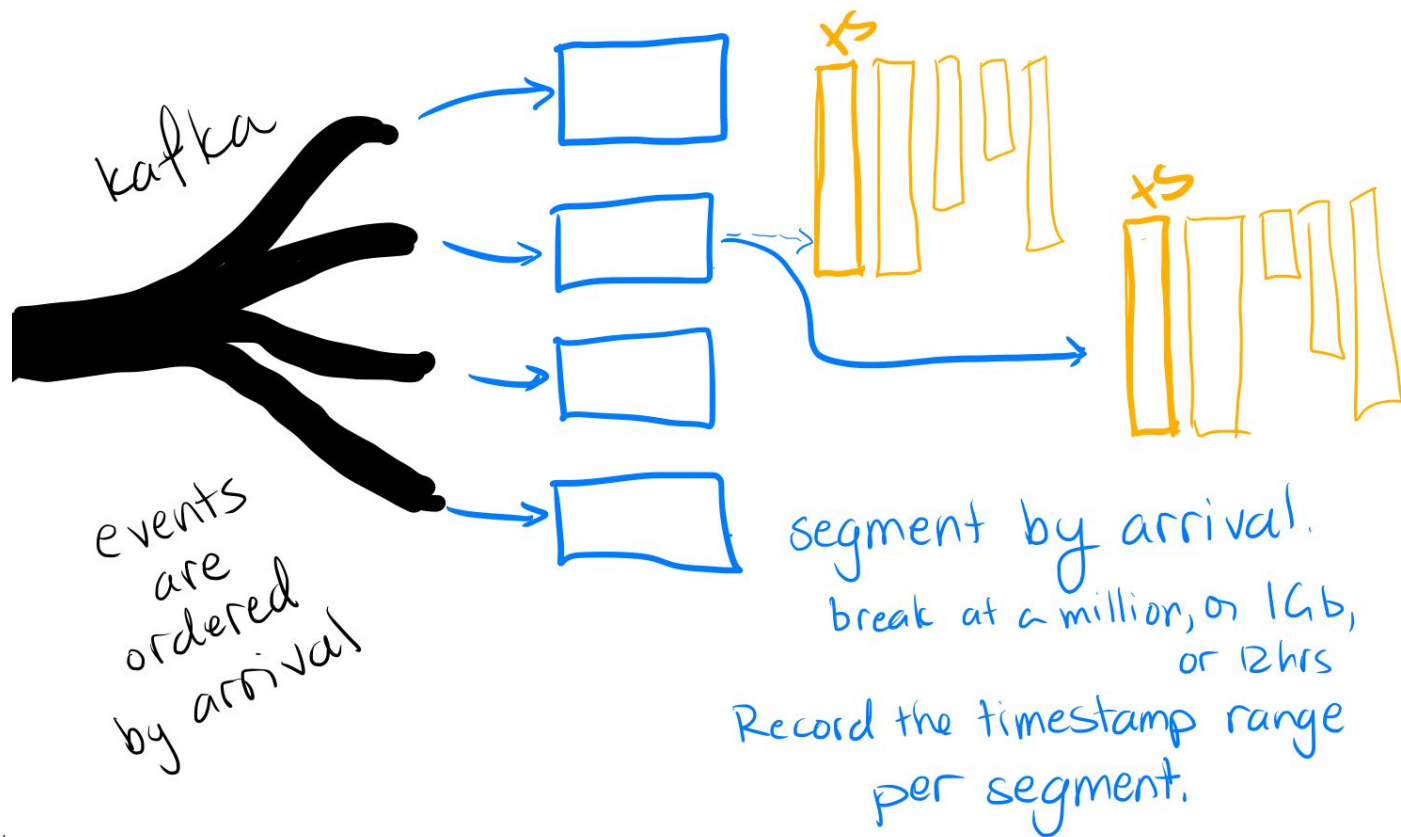


Retriever is a distributed column store.

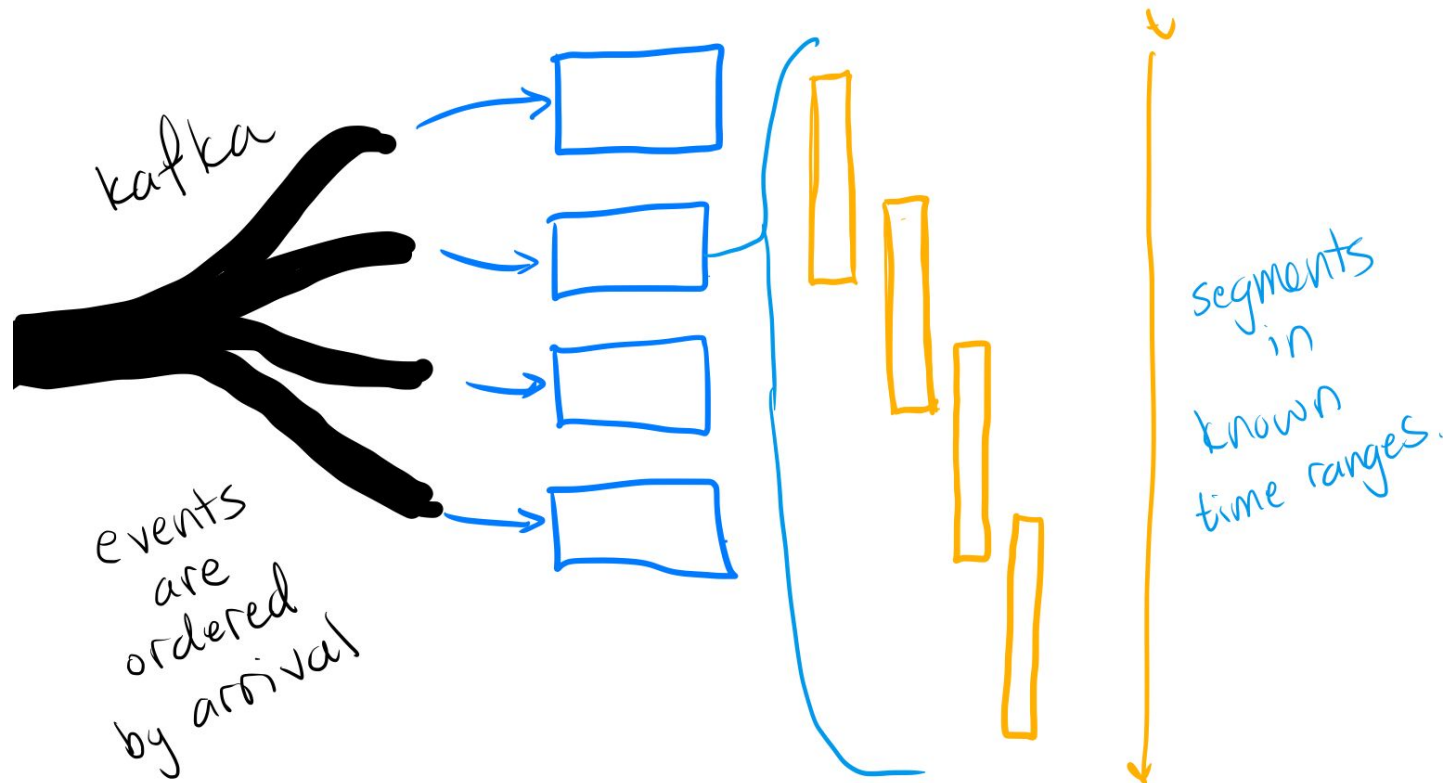


only read columns
we want to
filter, aggregate,
or group by.

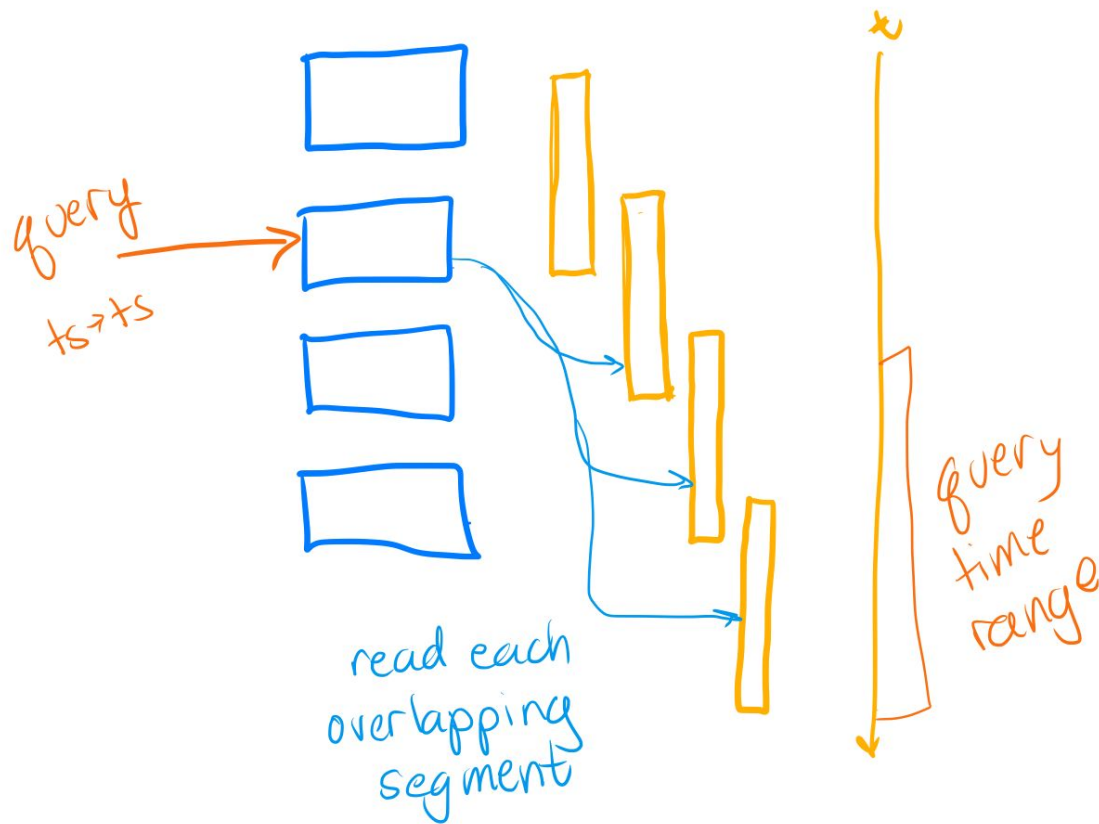
Retriever indexes segments by timestamp.



Retriever indexes segments by timestamp.



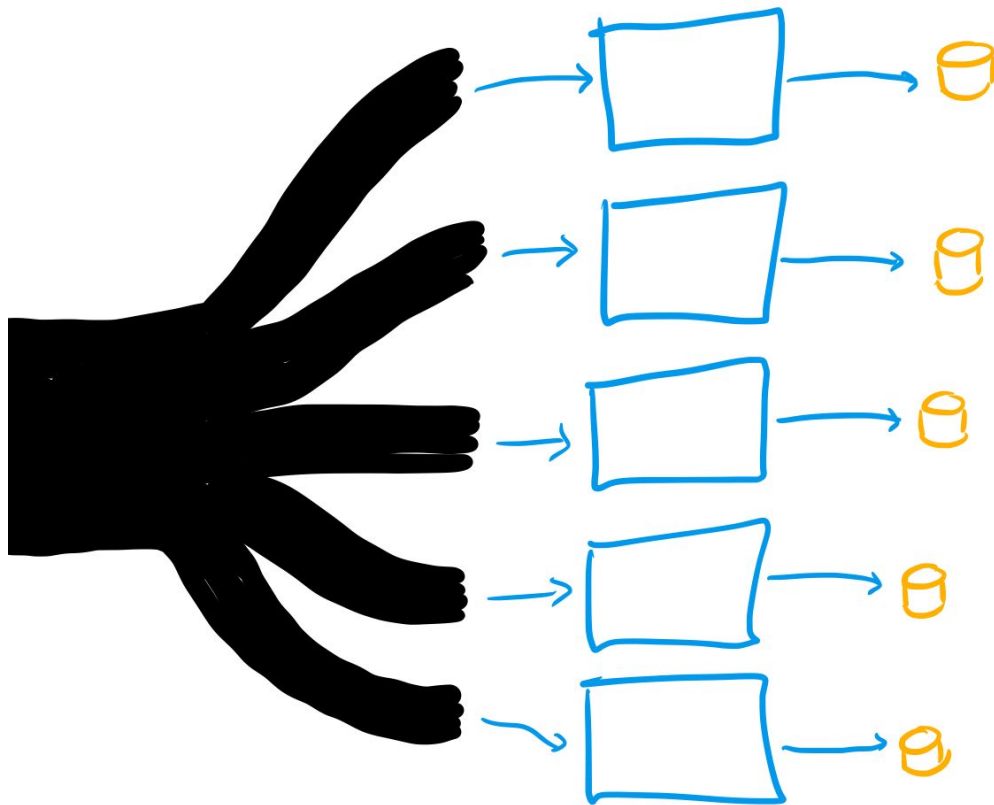
Retriever indexes segments by timestamp.



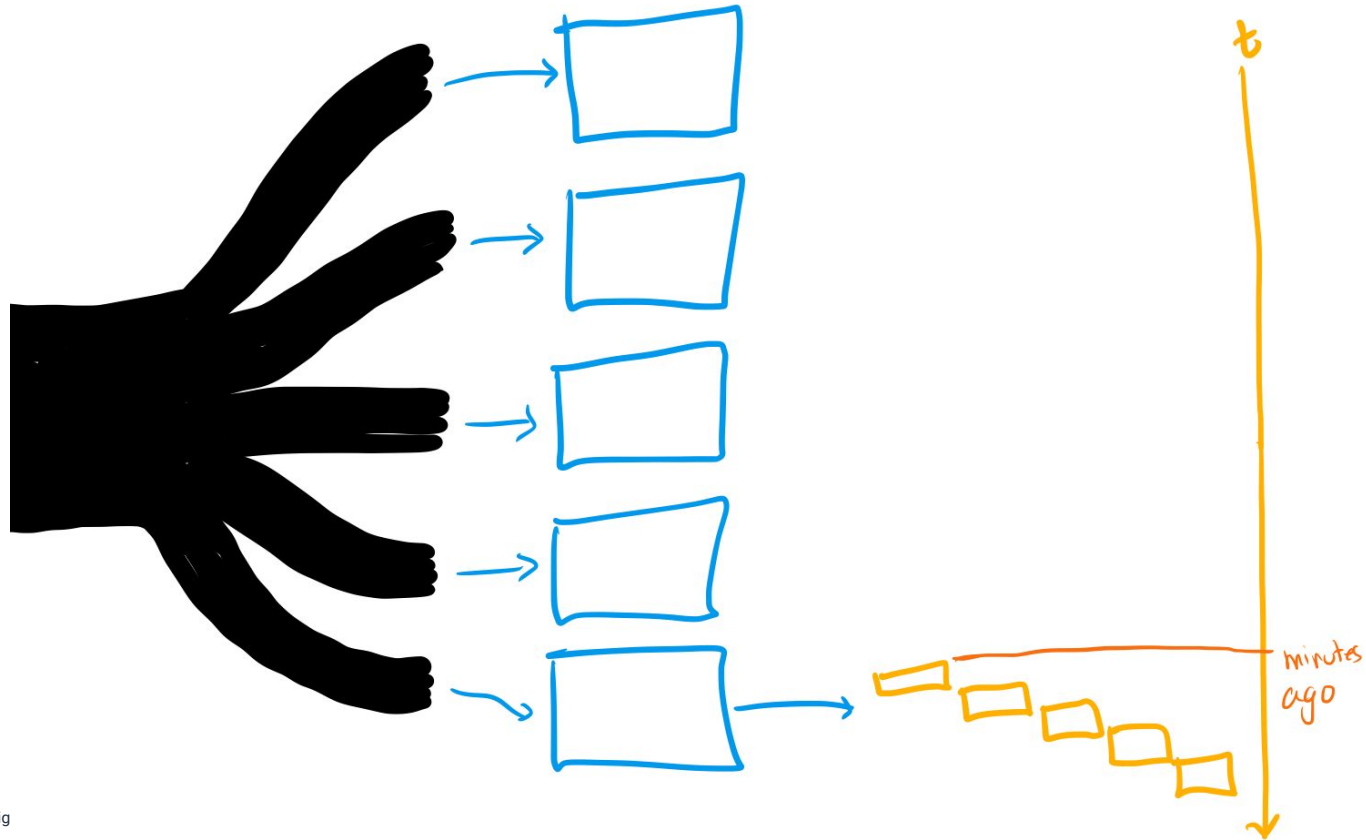
Dynamic aggregation of any fields across any time range

A custom datastore, carefully suited, continually optimized.

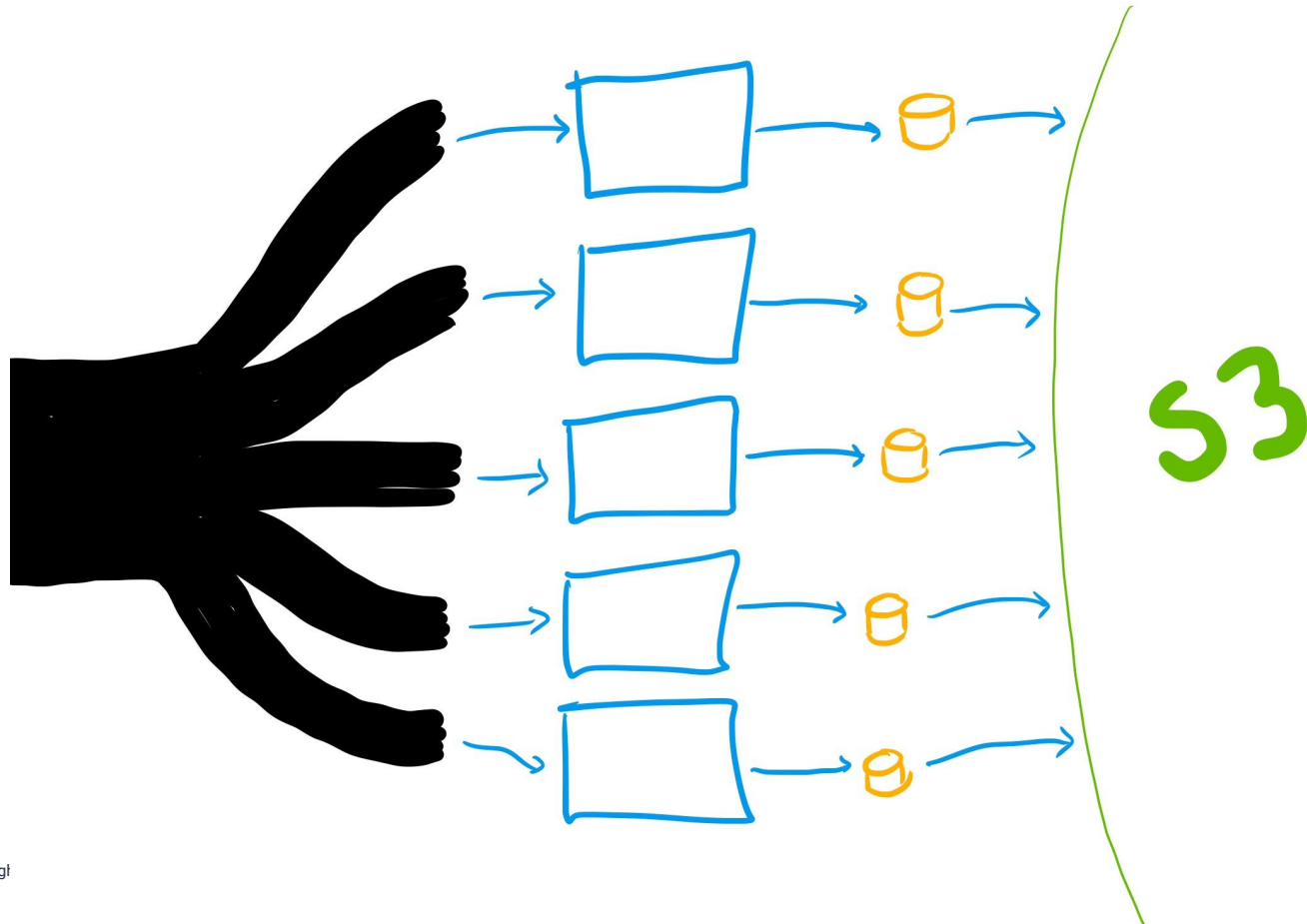
Bigger customers, more data coming in.



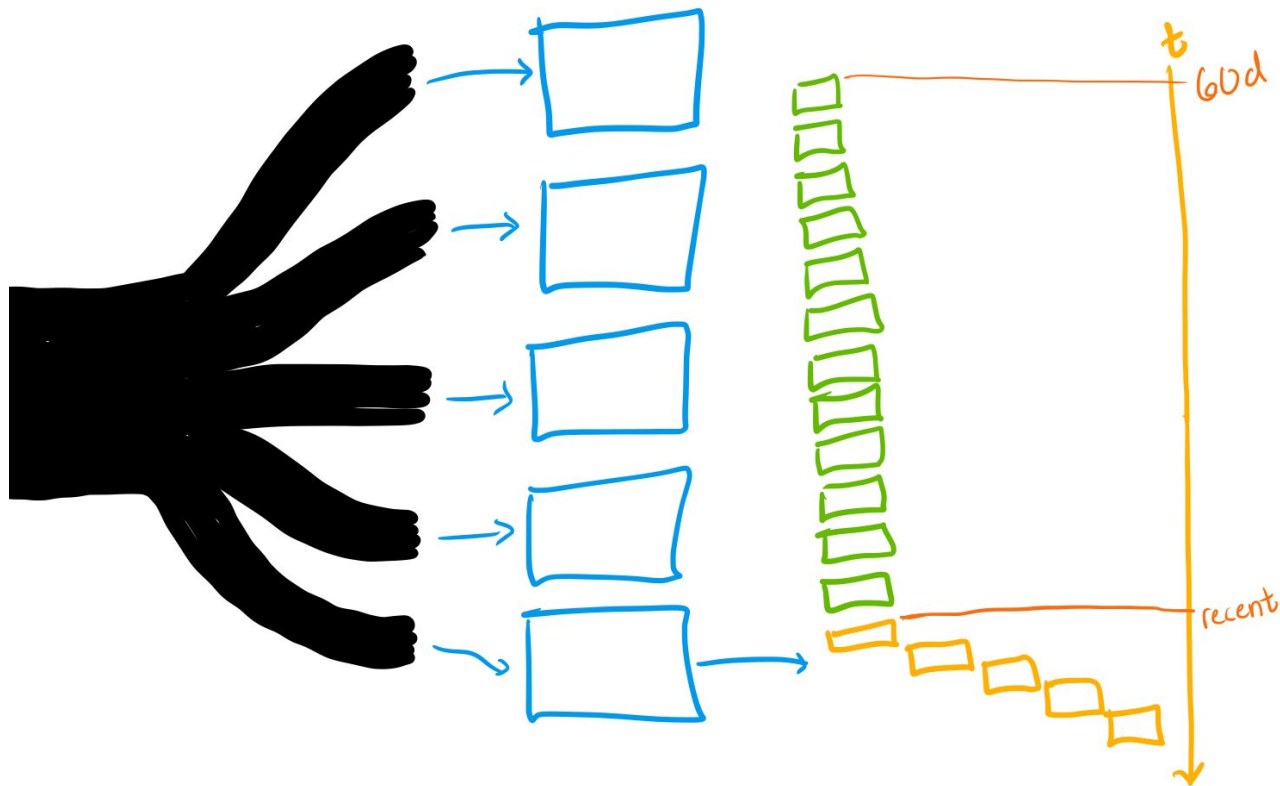
Segments hold a smaller time range.



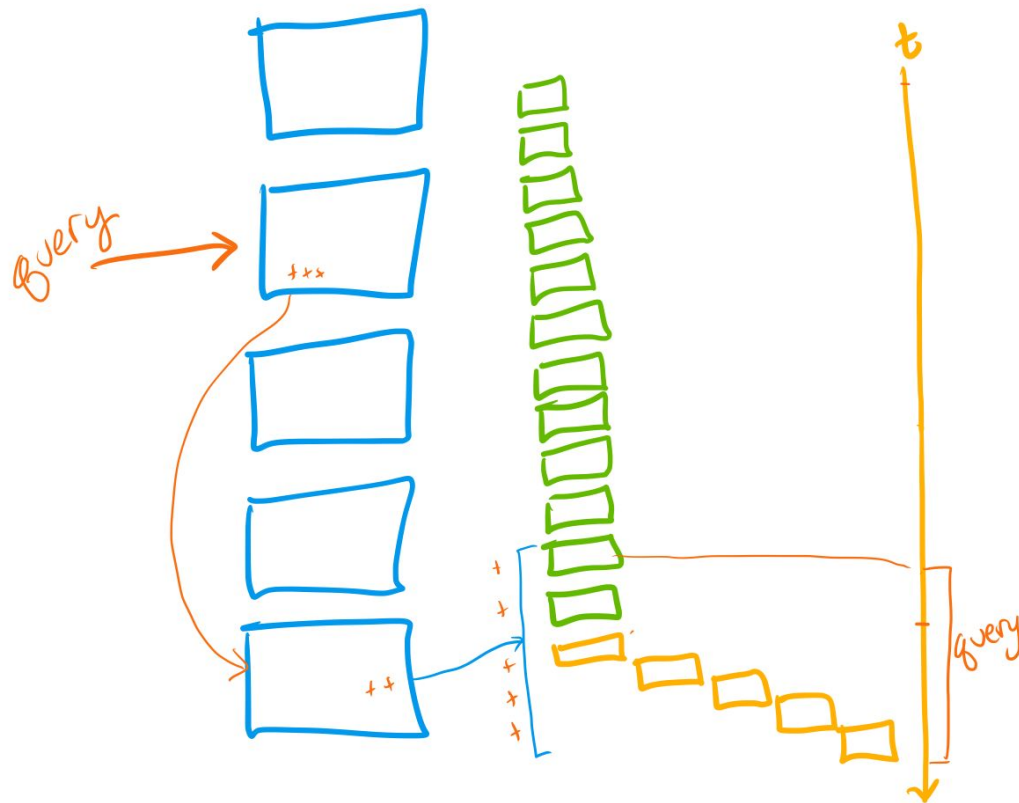
Solution: MOAR storage



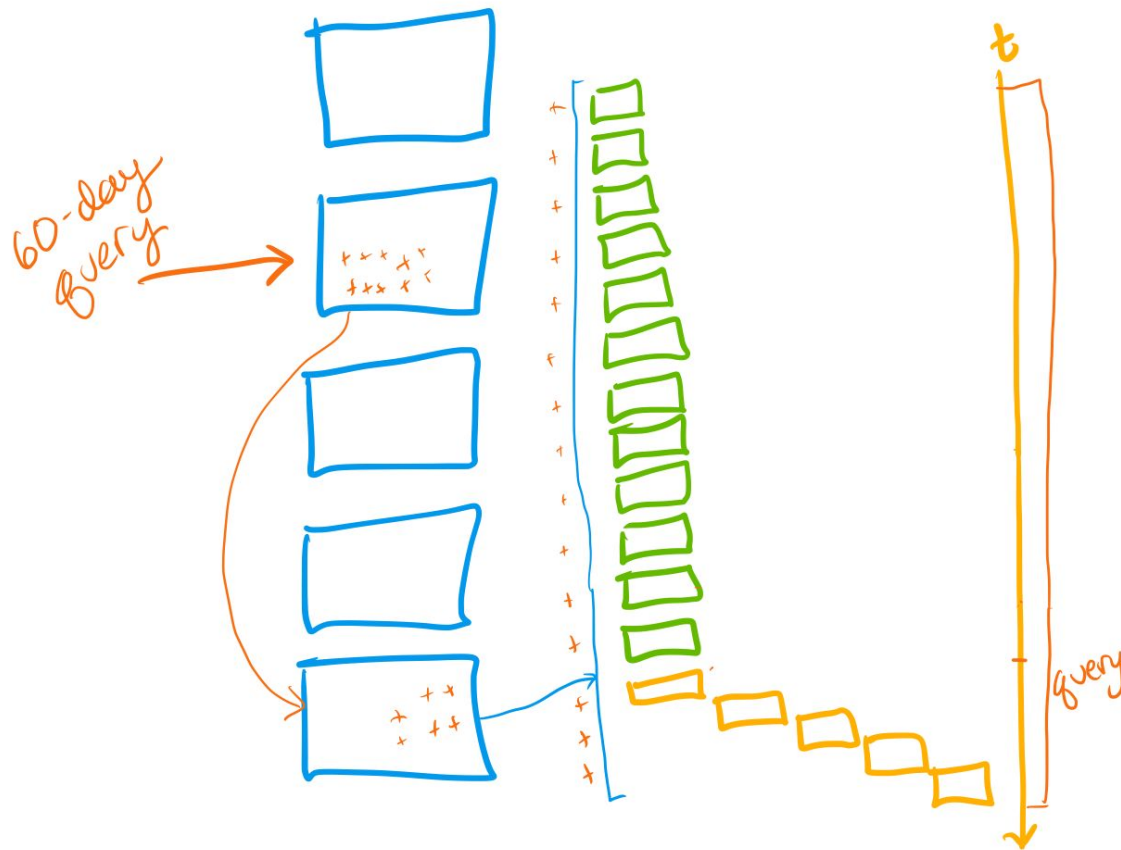
Now we can keep data for a fixed time range!



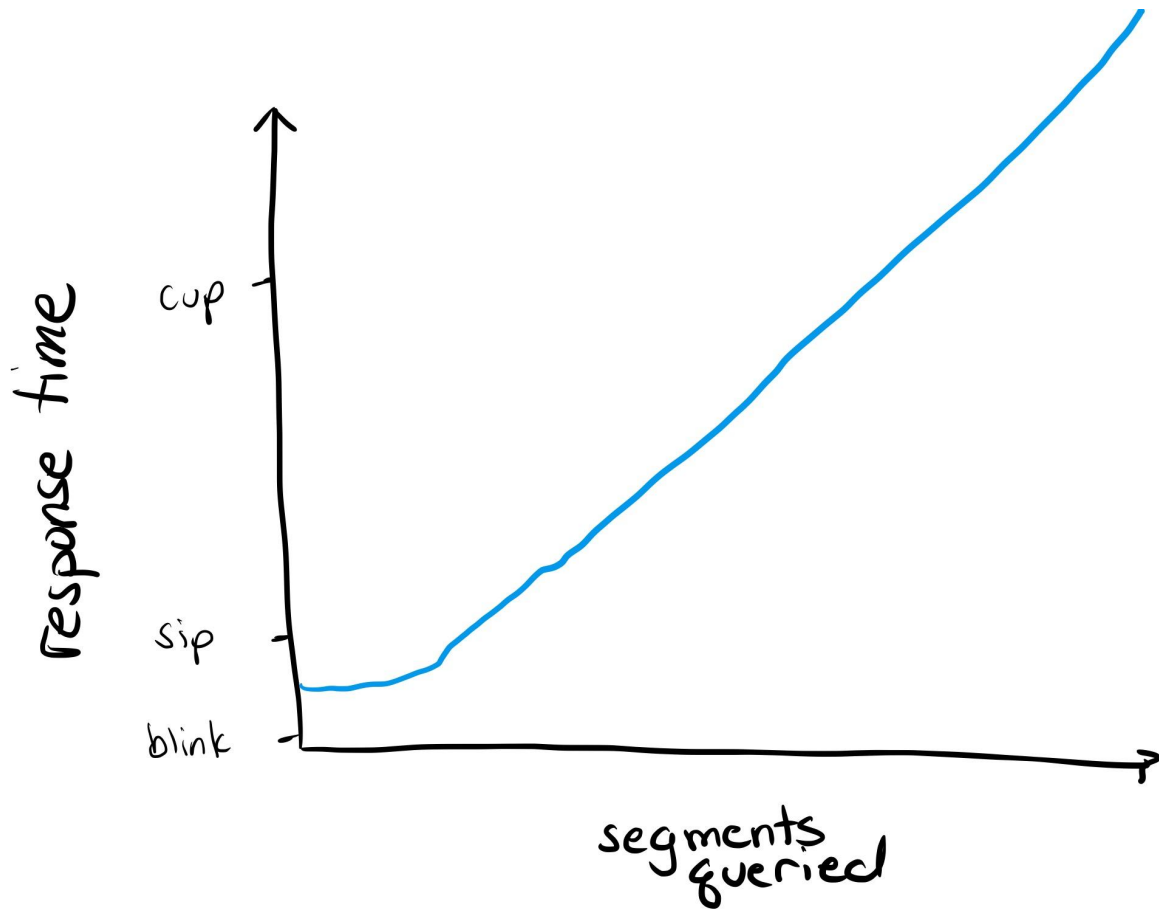
Retrievers grab data back from S3 at need.



Now people can run queries over 60 days 🤖

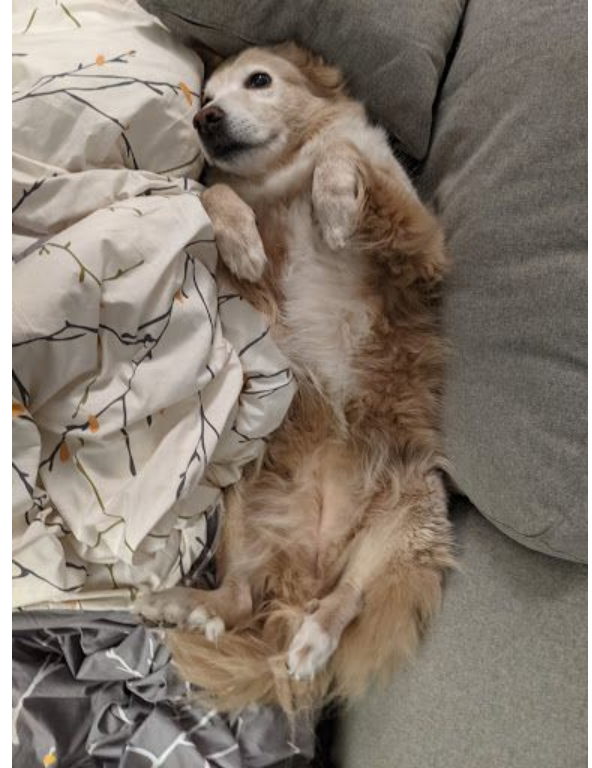


Now people can run queries over 60 days 🤔



**“ Lots more compute to play
with, pretty please!
but only if I want to play!**

Retrievers



MOAR compute, on demand.

VISUALIZE

CONCURRENCY

WHERE

service_name = lambda
name = Invoke

GROUP BY

None; don't segment

...

Run Query

Run 2 minutes
ago

+ ORDER BY

+ LIMIT

+ HAVING

Results BubbleUp Metrics Traces Raw Data

☐ Compare to

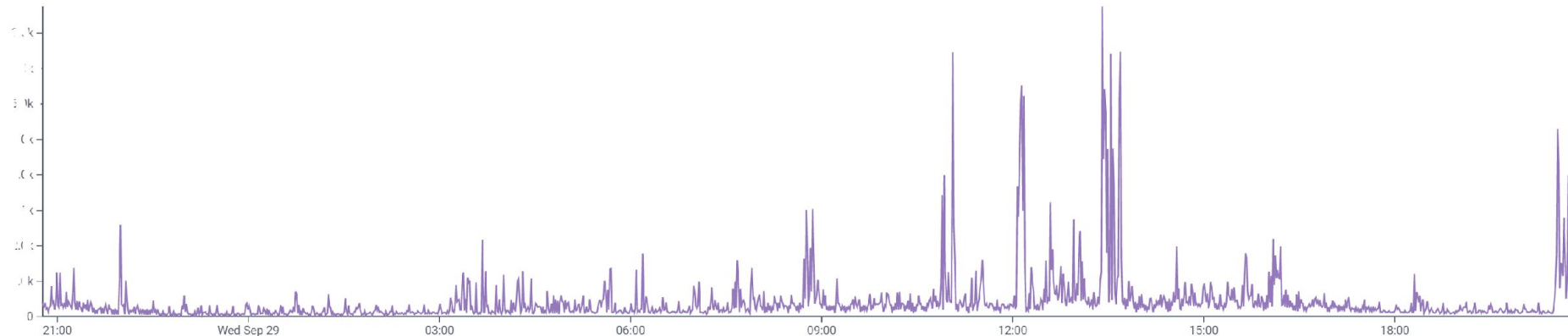
1 day prior



Graph Settings

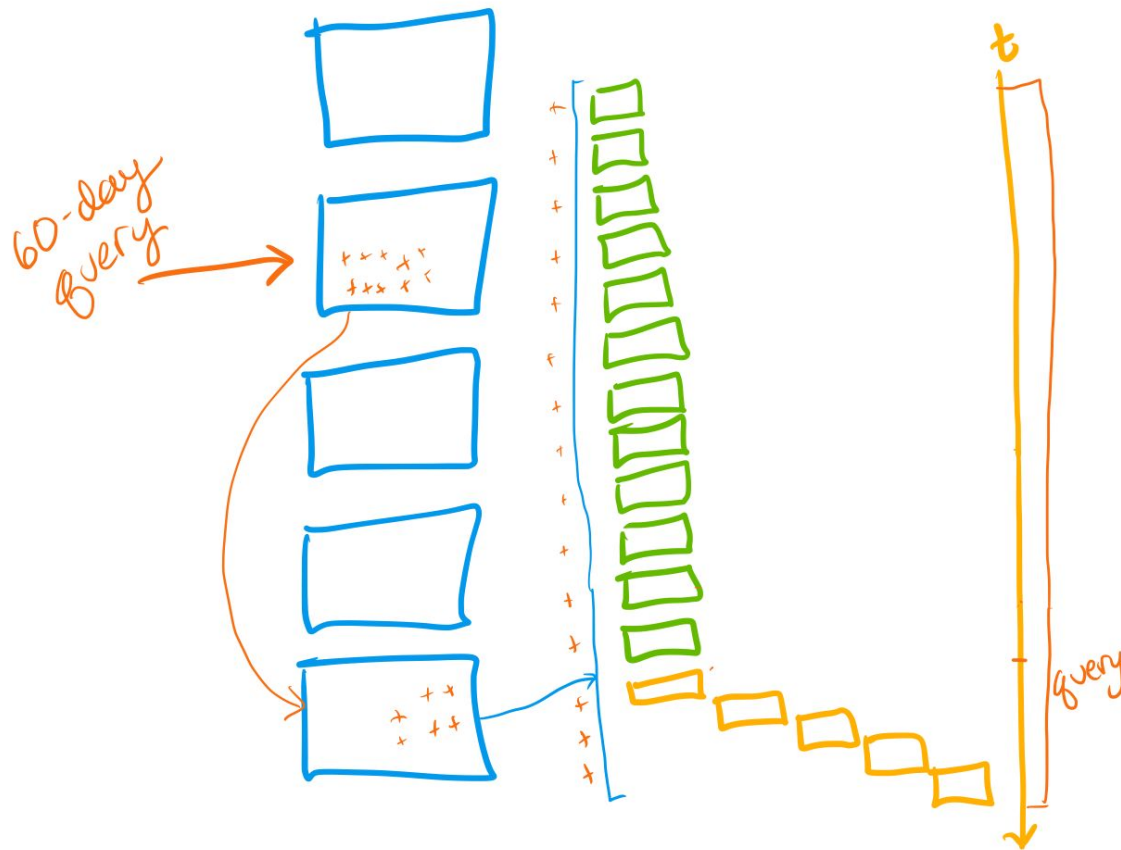
Sep 28 2021, 8:46 PM – Sep 29 2021, 8:46 PM (Granularity: 1 min)

CONCURRENCY

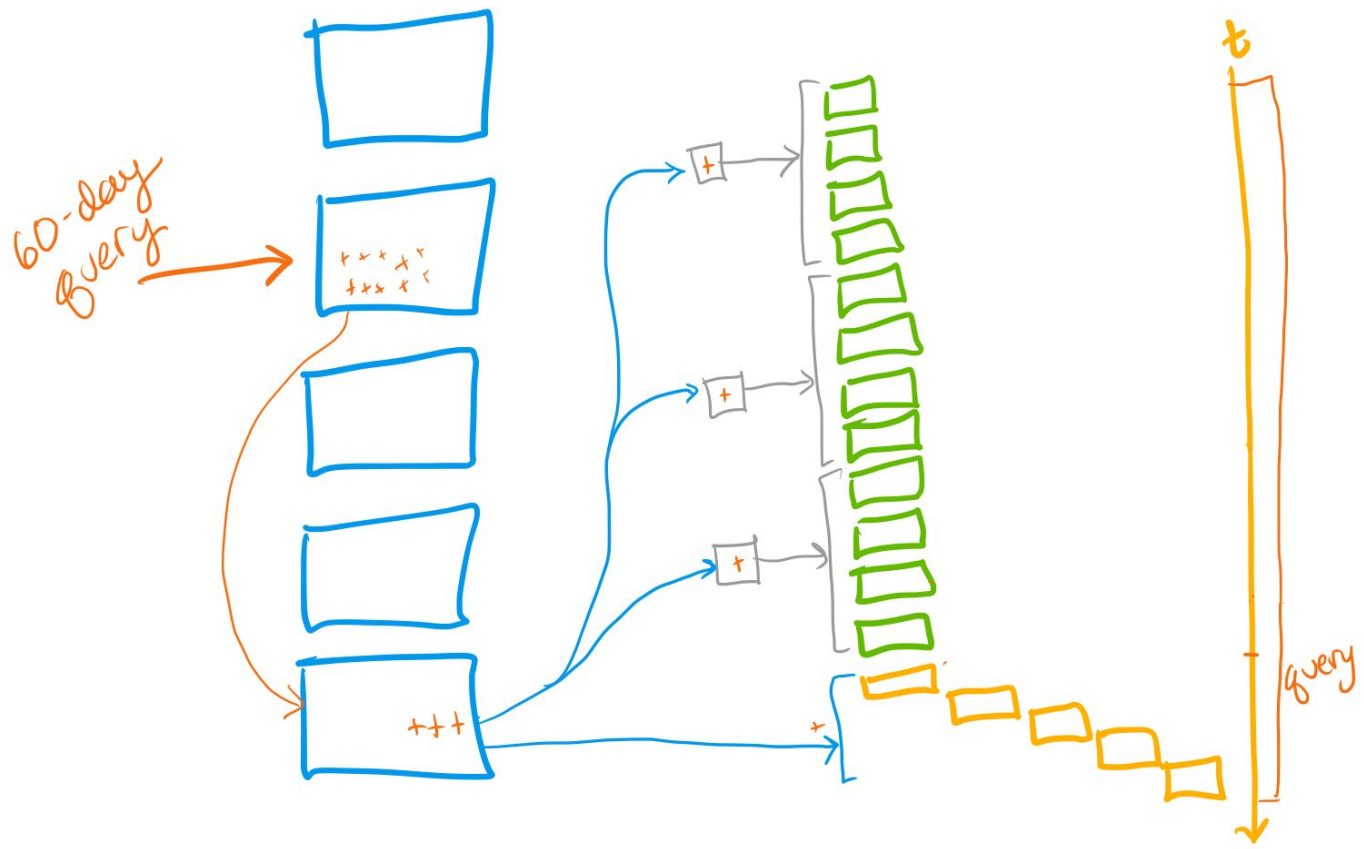


@iiznigrey at #GOLDORD

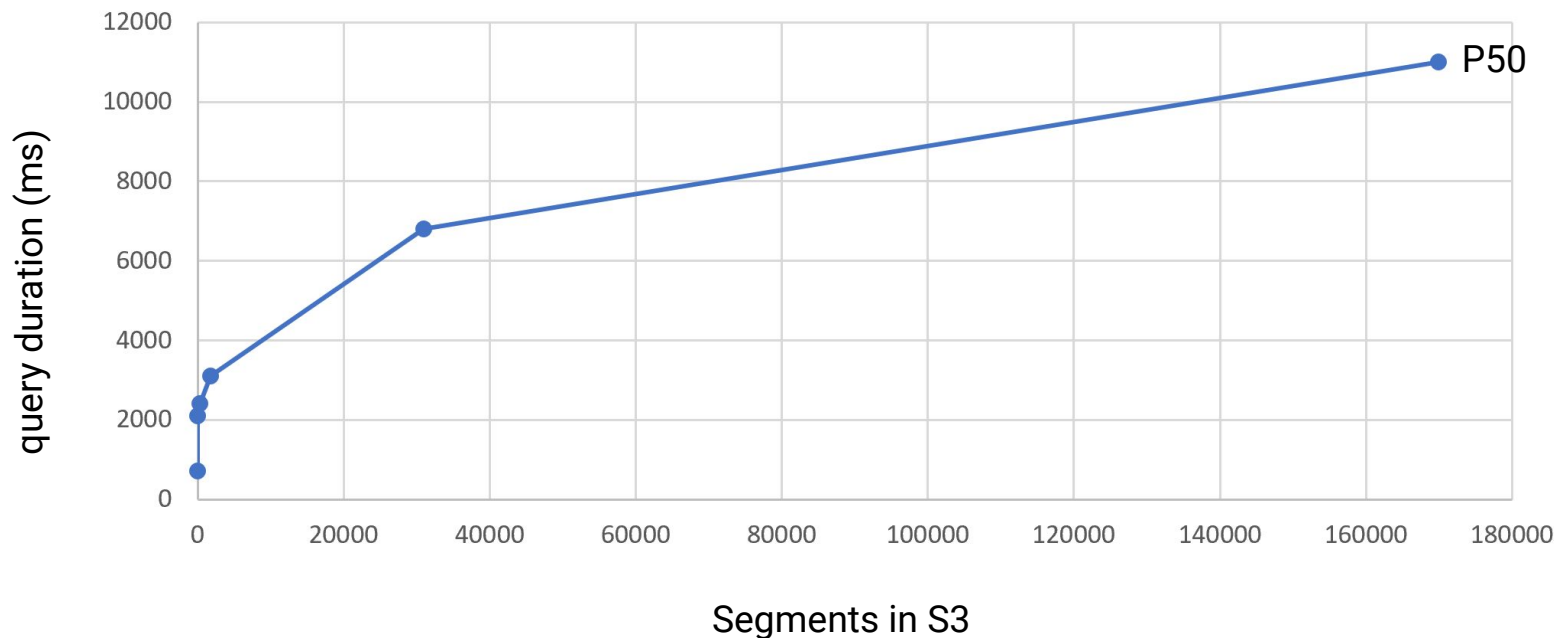
Problem: too much data for one retriever...



Solution: more compute, on demand.



Increase in query time is sublinear



Buy compute in ~~100ms~~ 1ms units

Compute scales with time range, so response time doesn't have to.

Lambda scales* up our compute

50ms

median* startup
time

90%

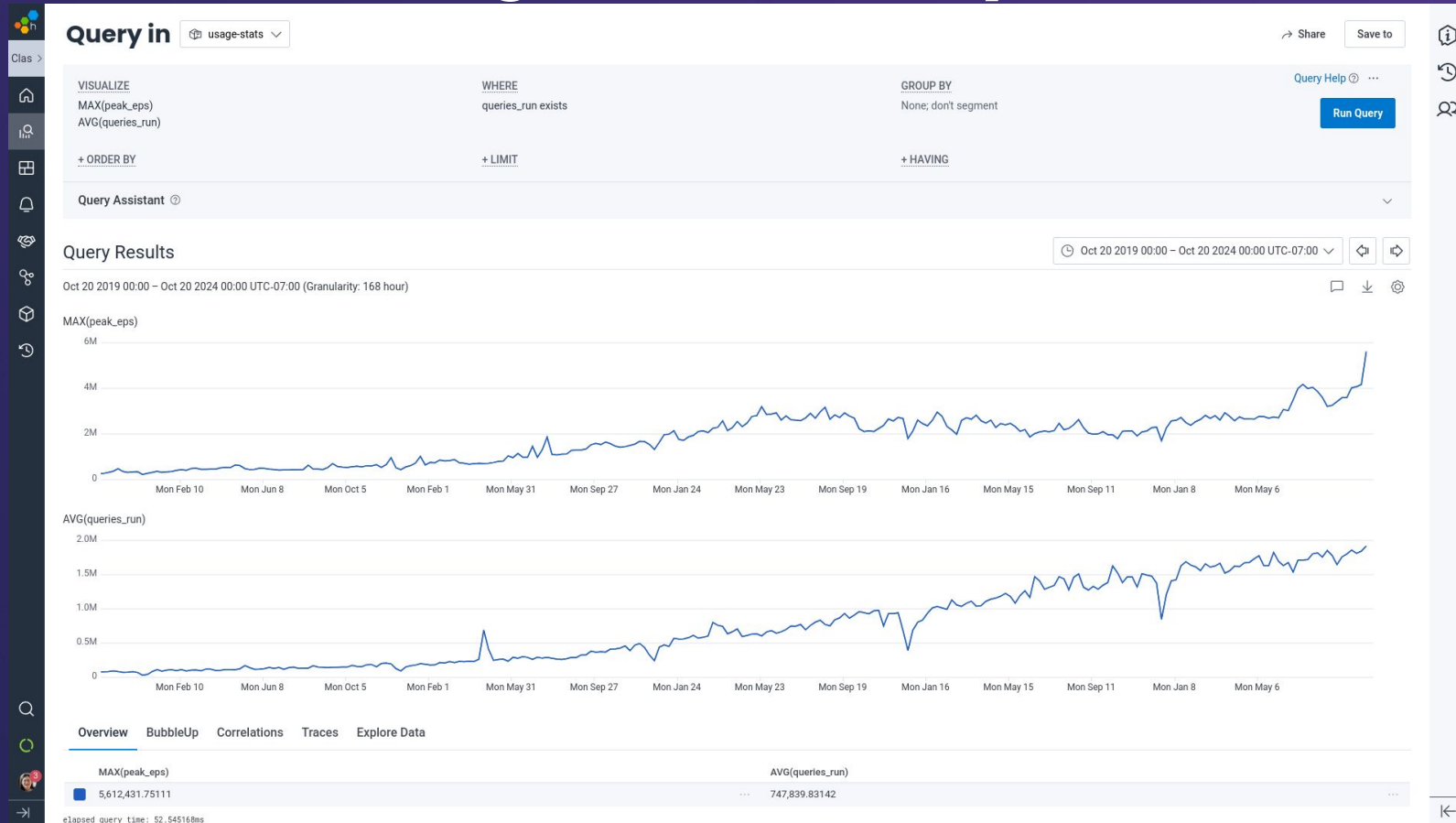
of ours return*
within 2.5s

3-4x

as expensive*
as EC2



20x growth in five years



Considerations

Lambda *is* on-demand compute, but they didn't build it for this.

Lambda **scales** up our compute

50ms

median startup
time

90%

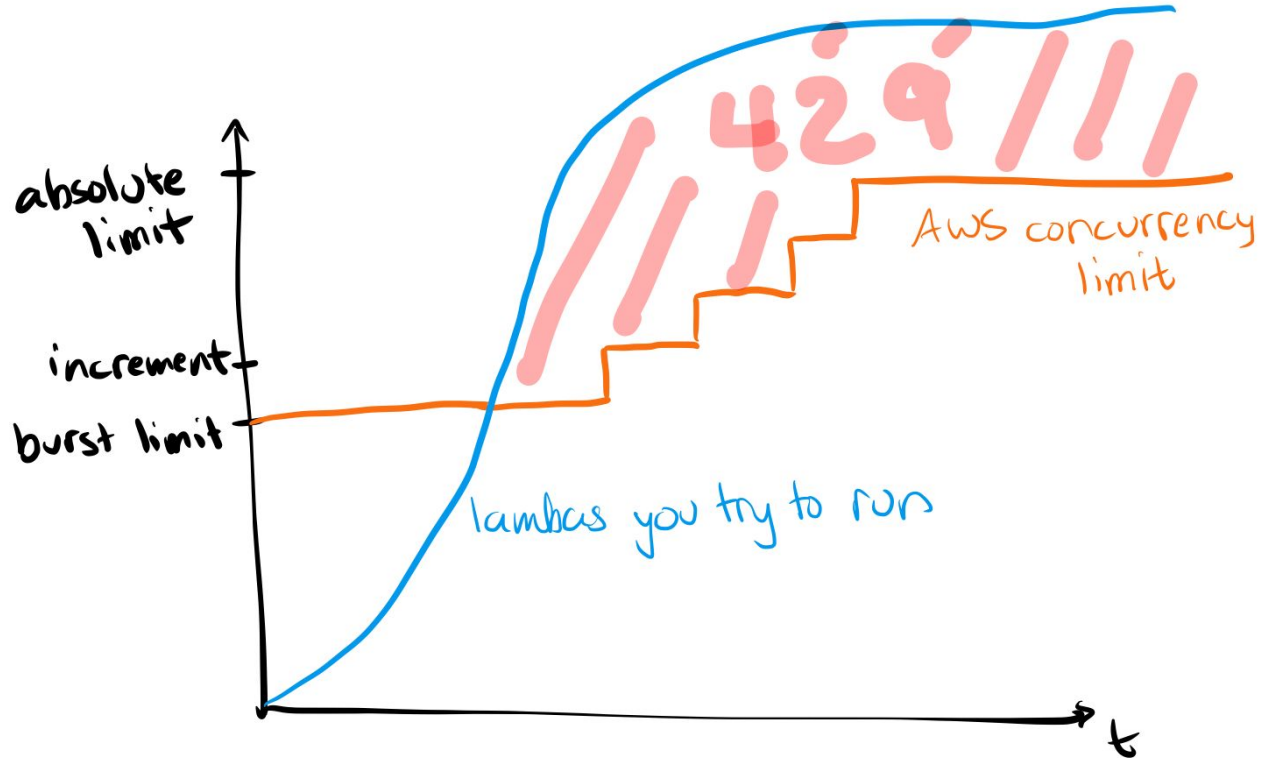
of ours return
within 2.5s

3-4x

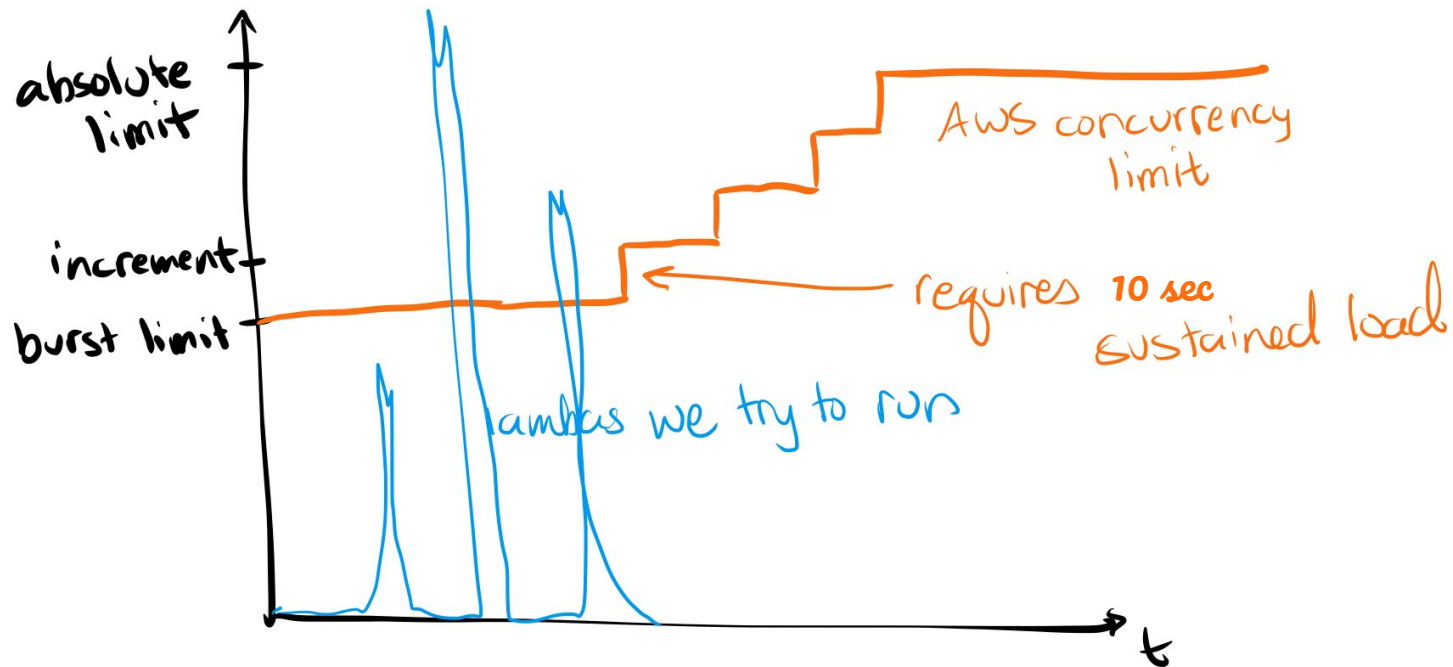
as expensive as
EC2



Lambda scales... within limits



Lambda scales... within limits



Observability helps: concurrency

VISUALIZE

CONCURRENCY

WHERE

service_name = lambda
name = Invoke

GROUP BY

None; don't segment

...

Run Query

Run 2 minutes
ago

+ ORDER BY

+ LIMIT

+ HAVING

Results BubbleUp Metrics Traces Raw Data

☐ Compare to

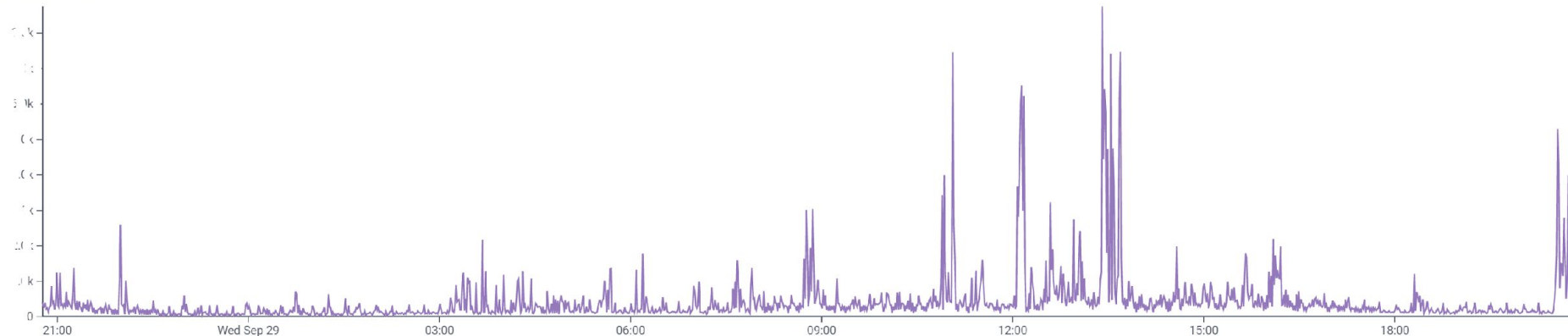
1 day prior



Graph Settings

Sep 28 2021, 8:46 PM – Sep 29 2021, 8:46 PM (Granularity: 1 min)

CONCURRENCY



@iiznigrey at #GOTOORD

Lambda scales... within limits

Study your limits:

<https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html>

Change the SDK retry parameters

Observability helps 😊

Talk to your account reps



Lambda scales up our compute

50ms

median startup
time

90%

of ours return
within 2.5s

3-4x

as expensive as
EC2



Functions start up... when they do

retriever-traces Queries | Honeycomb | retriever-traces Trace | Honeycomb |

https://ui-dogfood.honeycomb.io/prod/datasets/retriever-traces/result/fMmrnrkk9BV/trace/kDJ6kSoZaTz

← Trace 4a4cc66ecb68cf734239d25674ee5a3b at 2021-09-29 20:31:59

Search spans

Fields

Rerun

lambda > main

Distribution of span duration ?

HEATMAP(duration_ms)

Fields

Filter fields and values in span

Timestamp
2021-09-30T01:31:59.04967907Z

duration_ms
13.236605

global_availability_zone
"

global_build_depth
11117

global_build_id
352504

global_commit_hash
66b8e1fad87b42b900c54f2805a77518066c004

global_env
production

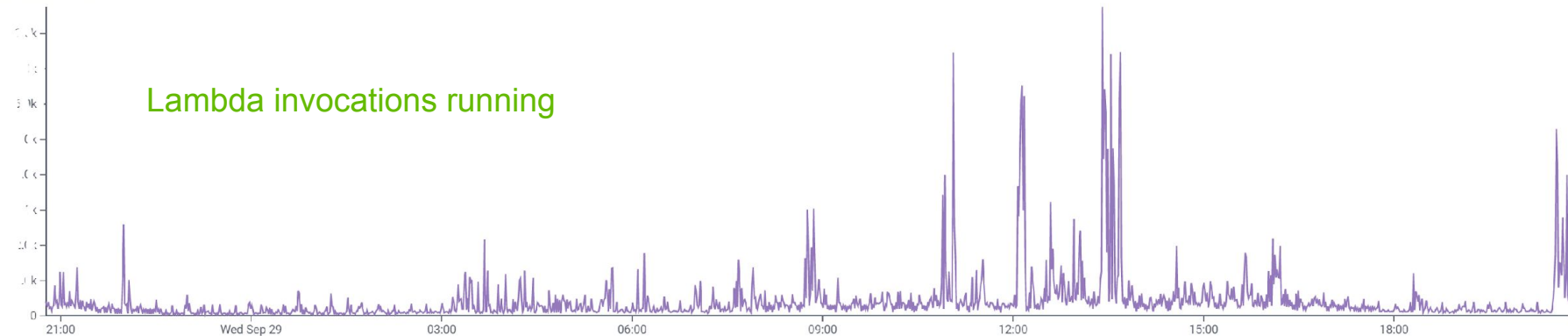
global_infra_type
aws_instance

global_instance_type

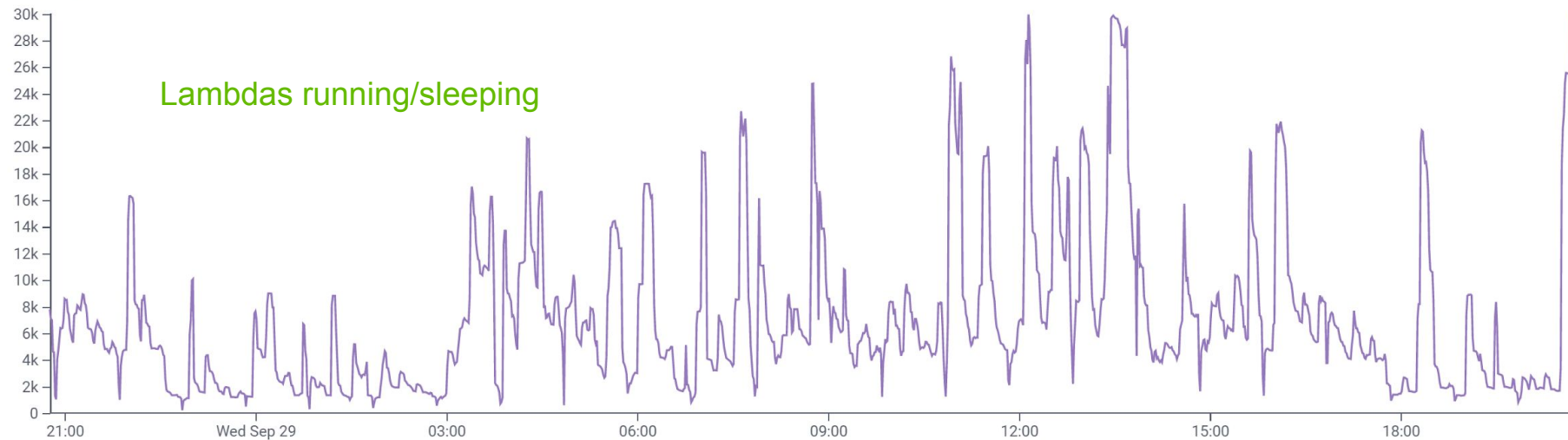
name	service_name	0s	200s	400s	600s	800s	1,000s	1,186s
215 main	lambda	13.24ms						
run	lambda	1.863s						
sleep	lambda	3.945s						
run	lambda	353.7ms						
sleep	lambda	33.204s						
run	lambda	614.8ms						
sleep	lambda	40.089s						
run	lambda	1.225s						
sleep	lambda	15.48ms						
run	lambda	974.4ms						
sleep	lambda	9.063s						
run	lambda	1.333s						
sleep	lambda	8.284s						
run	lambda	1.297s						
sleep	lambda	12.119s						
run	lambda	1.356s						
sleep	lambda	5.723s						
run	lambda	3.608s						
sleep	lambda	22.405s						
run	lambda	1.738s						
sleep	lambda	13.15ms						



CONCURRENCY



CONCURRENCY



Lambda scales up our compute

50ms

median startup
time

90%

of ours return
within 2.5s

3-4x

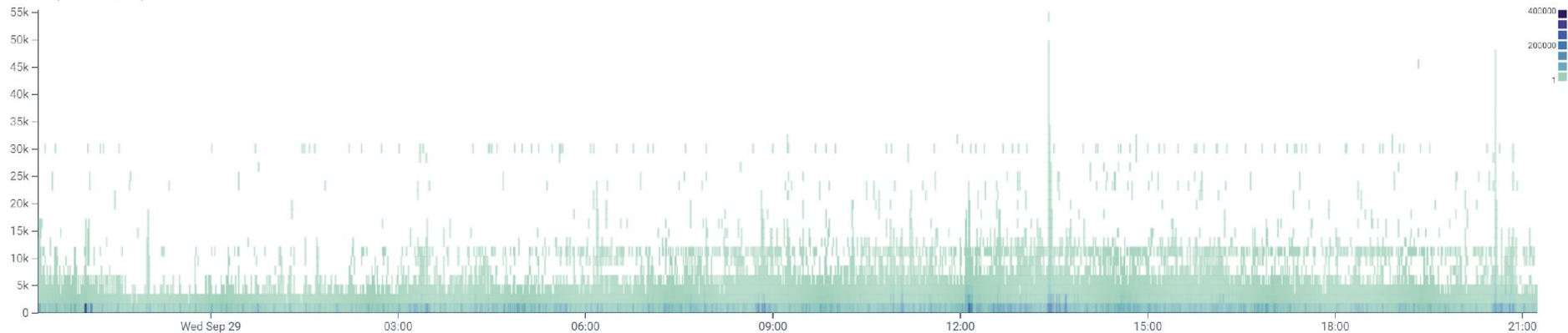
as expensive as
EC2



Functions return... usually

Sep 28 2021, 9:14 PM – Sep 29 2021, 9:14 PM (Granularity: 1 min)

HEATMAP(duration_ms)



Functions accept... JSON

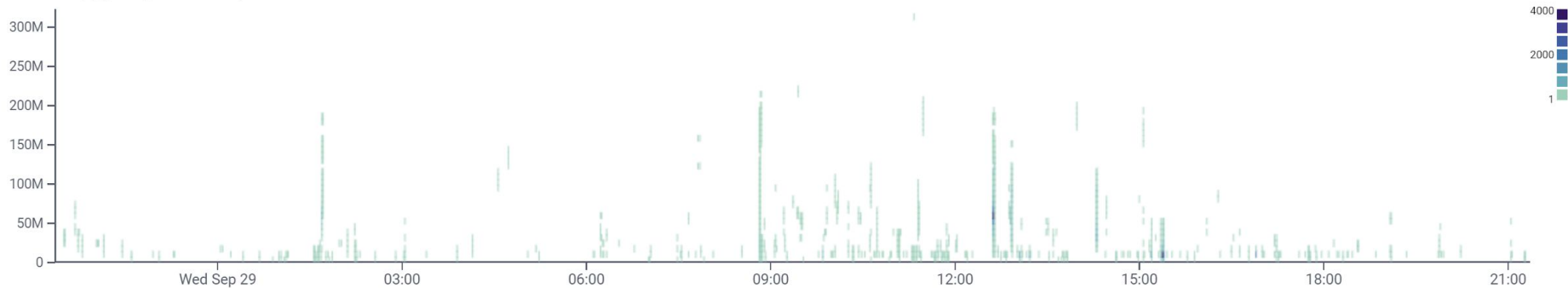
Put the data in S3 and send a link.



Functions return... up to 6Mb

Sep 28 2021, 9:22 PM – Sep 29 2021, 9:22 PM (Granularity: 1 min)

HEATMAP(app.response_size)



Put the data in S3 and send a link.



Lambda scales up our compute

50ms

median startup
time

90%

of ours return
within 2.5s

3-4x

as expensive as
EC2



Functions cost... something

Query run every 1440 minutes Define the calculation to perform and any relevant filters

VISUALIZE

× SUM(lambda_cost)

WHERE

× dataset_id exists

AND ∨

GROUP BY

× dataset_id

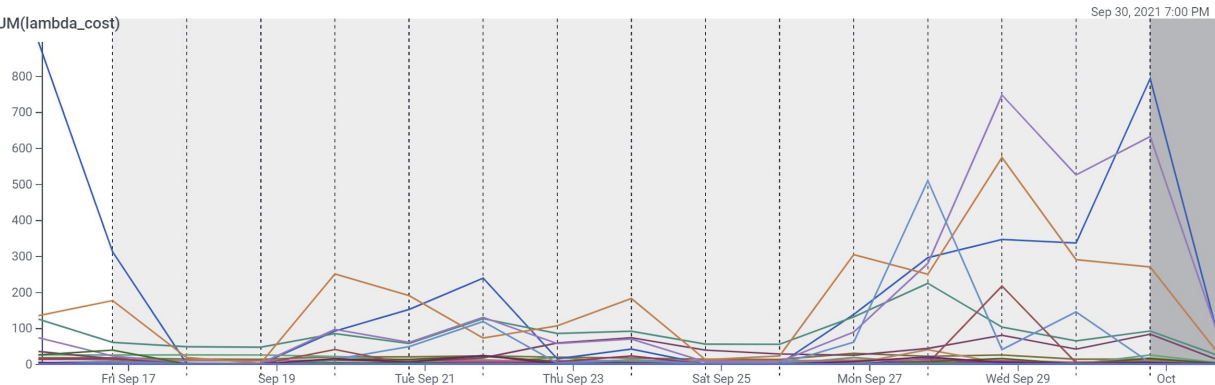
Triggering Queries are constrained to one calculation, and as many filters as you'd like.

Below, we show the SUM(lambda_cost) trends for each dataset_id (where dataset_id exists) for the last 16 1440-minute intervals.

The markers indicate the last 16 points at which the trigger would have run.

Sep 15 2021, 8:15:43 PM – Oct 1 2021, 8:15:43 PM (Granularity: 1 day)

SUM(lambda_cost)



Threshold

Trigger notification if returned SUM(lambda_cost) for any dataset_id is

>= ∨

300



Functions cost... let's make it less?

[AWS News Blog](#)

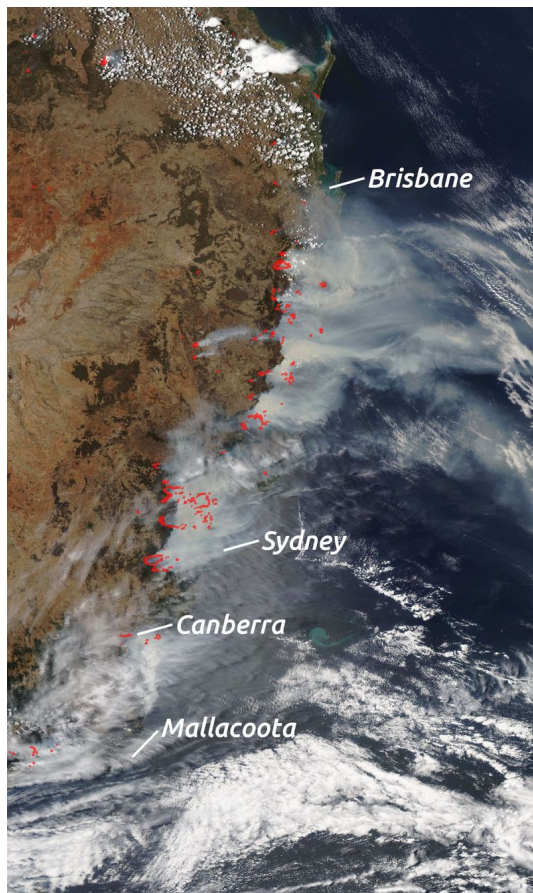
AWS Lambda Functions Powered by AWS Graviton2 Processor – Run Your Functions on Arm and Get Up to 34% Better Price Performance

by [Danilo Poccia](#) | on 29 SEP 2021 | in [AWS Lambda](#), [Compute](#), [Graviton](#), [Serverless](#) | [Permalink](#) | [💬 Comments](#) | [↪ Share](#)

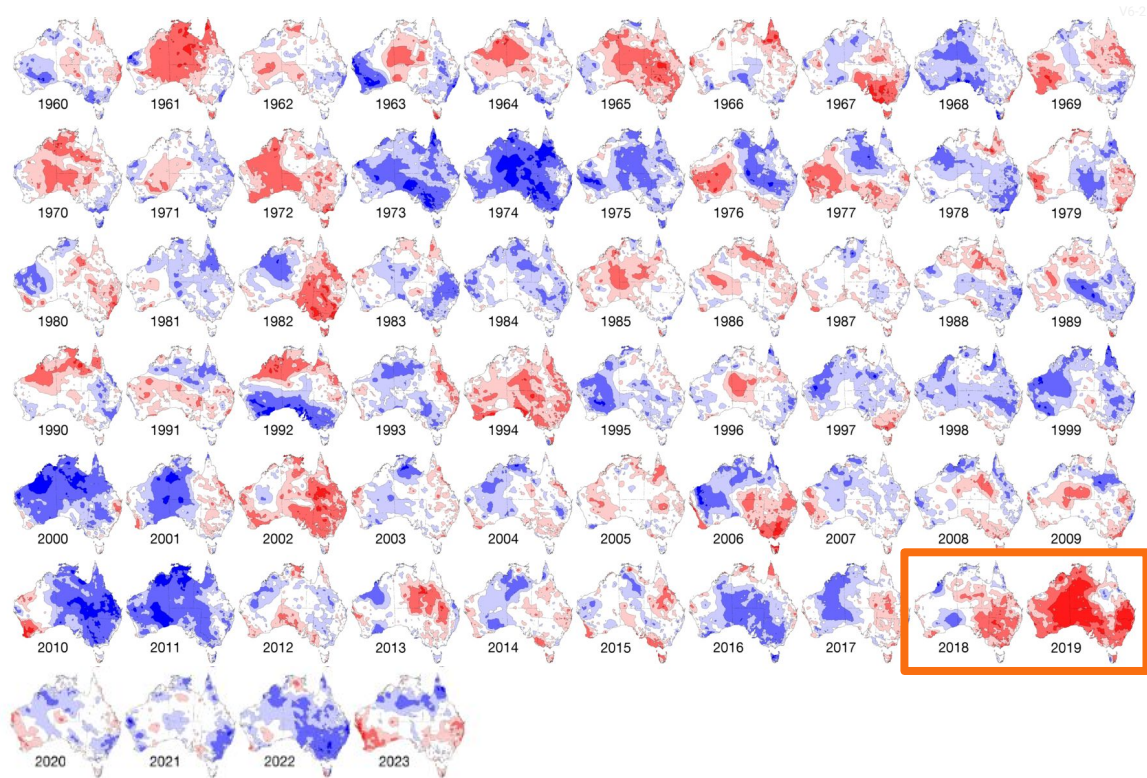




but it's not all about the money.



Sources: NASA (Public Domain), Australia BOM (CC-BY)

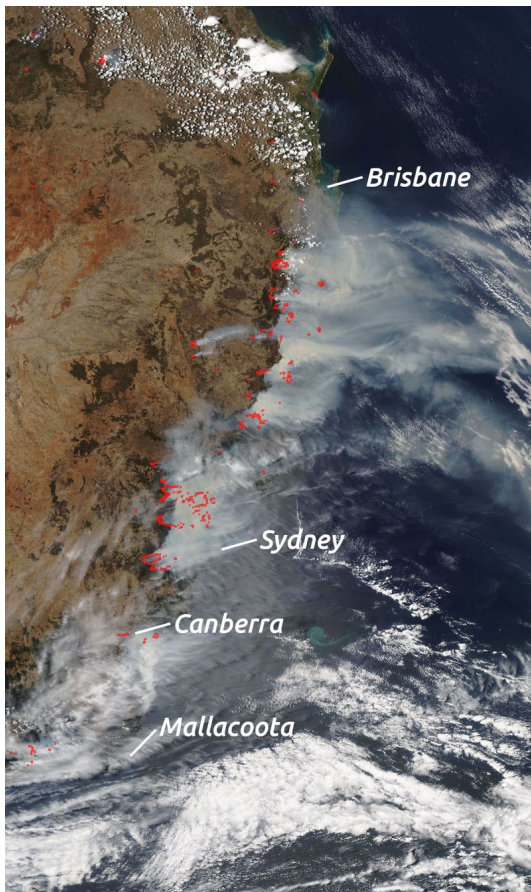


Unless otherwise noted, all maps, graphs and diagrams in this page are licensed under the [Creative Commons Attribution 4.0 International Licence](https://creativecommons.org/licenses/by/4.0/)

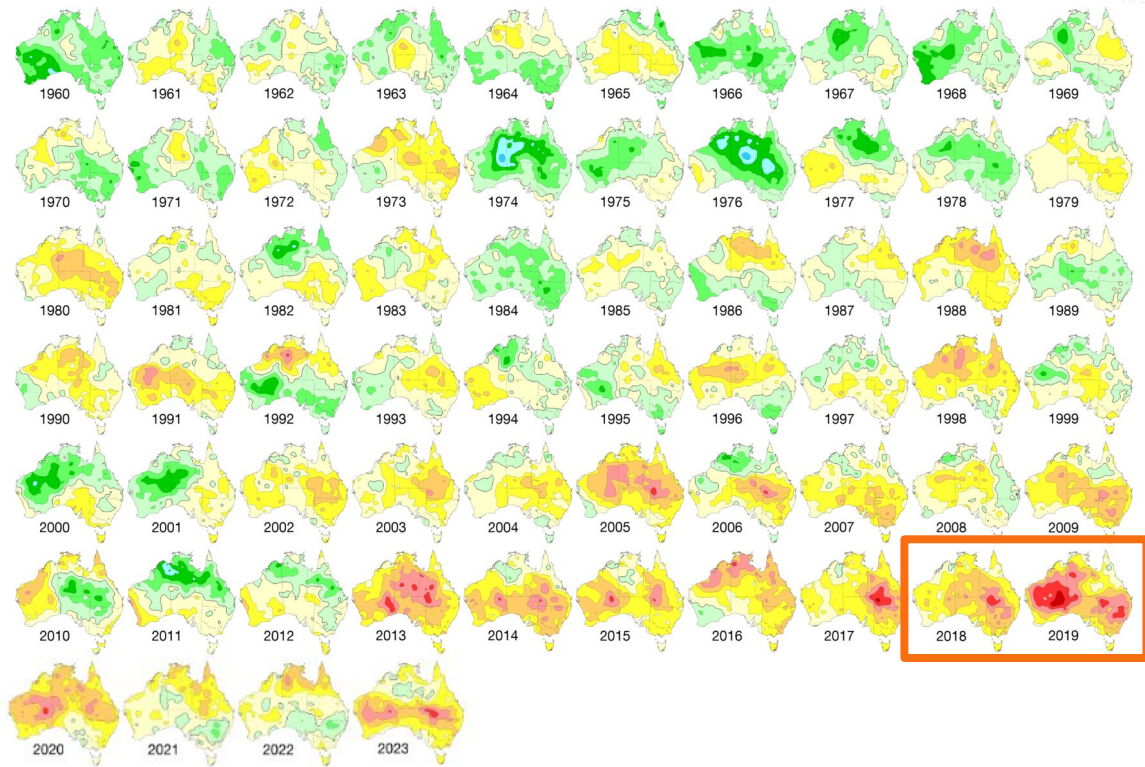


© 2023 Hound Technology, Inc. All Rights Reserved.

@lizthegrey at #GotoORD



Sources: NASA (Public Domain), Australia BOM (CC-BY)



Product code: IDCKPT5AA0

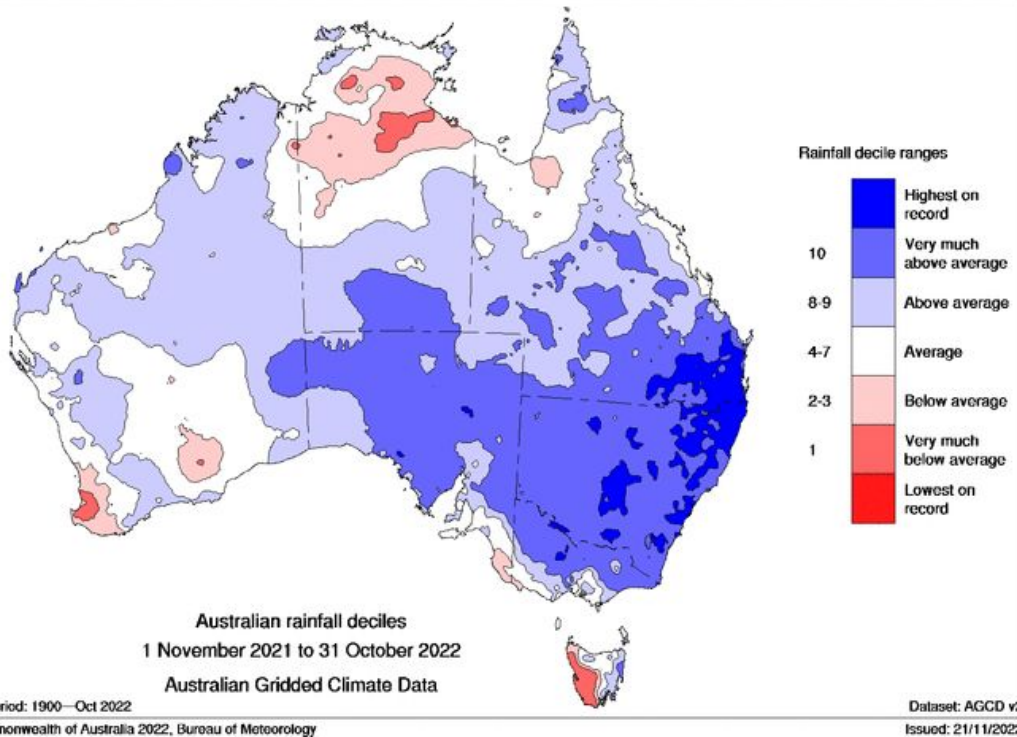


Unless otherwise noted, all maps, graphs and diagrams in this page are licensed under the [Creative Commons Attribution 4.0 International Licence](https://creativecommons.org/licenses/by/4.0/)

NSW floods now Australia's most expensive natural disaster as insurance claims skyrocket

Mayor of flood-hit Forbes says some residents have been told their policies won't be renewed, as parts of the state remain isolated

- Follow our Australia news live blog for the latest updates
- Get our morning and afternoon news emails, free app or daily news podcast



Sources: Australia BOM (CC-BY), The Guardian (fair use)



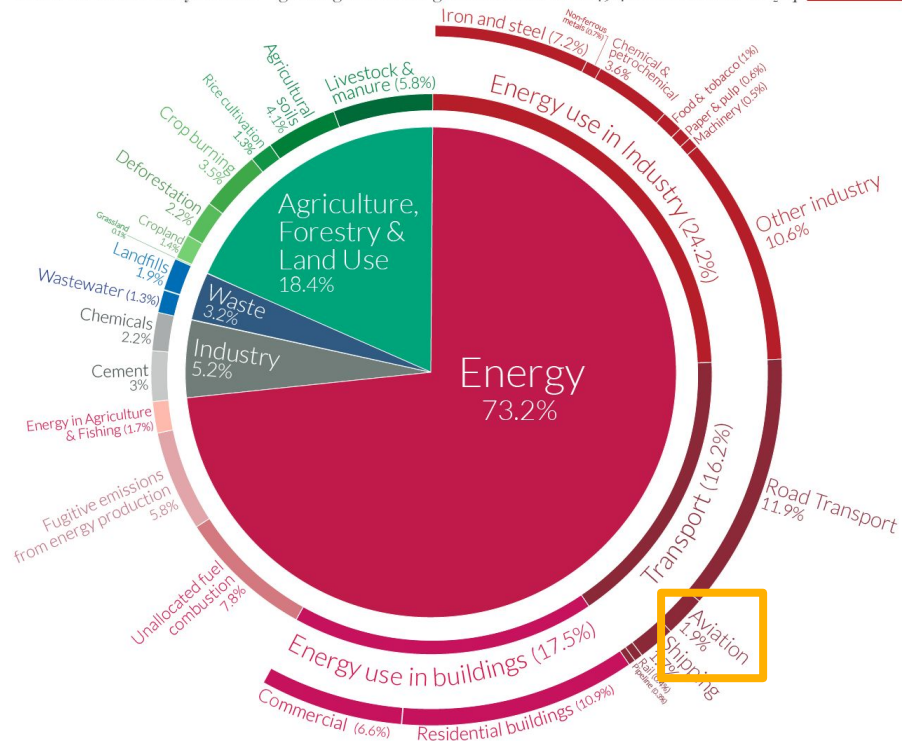
Software contributes significantly.

Companies have "carbon footprints"; let's not blame individuals

Global greenhouse gas emissions by sector

This is shown for the year 2016 – global greenhouse gas emissions were 49.4 billion tonnes CO₂eq.

Our World
in Data



OurWorldinData.org – Research and data to make progress against the world's largest problems.

Source: Climate Watch, the World Resources Institute (2020).

Licensed under CC-BY by the author Hannah Ritchie (2020).

Sources: Our World In Data - Hannah Ritchie (CC-BY); Patterns (CC-BY)

Review

The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations

Charlotte Freitag¹, Mike Berners-Lee¹, Kelly Widdicks²  , Bran Knowles², Gordon S. Blair², Adrian Friday²

Show more 

+ Add to Mendeley  Share  Cite

<https://doi.org/10.1016/j.patter.2021.100340>

Get rights and content

Under a Creative Commons license

 Open access

Referred to by Charlotte Freitag, Mike Berners-Lee, Kelly Widdicks, Bran Knowles, Gordon S. Blair, Adrian Friday

[The real climate and transformative impact of ICT: A critique of estimates, trends, and regulatio...](#)

Patterns, Volume 3, Issue 8, 12 August 2022, Pages 100576

 Download PDF

The bigger picture

To avoid catastrophic consequences from climate change, all sectors of the global economy, including *Information Communication Technology* (ICT), must keep their greenhouse gas (GHG) emissions in line with the Paris Agreement. We examine peer-reviewed estimates of ICT's GHG emissions, which put ICT's share of global GHG emissions at 1.8%–2.8%. We find pronounced differences and much debate concerning the underlying assumptions behind the peer-reviewed studies, which could suggest that global emissions from ICT are as high as 2.1%–3.9%. All study analysts agree that ICT emissions *will not reduce* without major concerted political





Offsetting isn't the solution.

Support the Guardian

Available for everyone, funded by readers

Support us →

The Guardian

[News](#)
[Opinion](#)
[Sport](#)
[Culture](#)
[Lifestyle](#)
[More ▾](#)

[Australia](#)
[Coronavirus](#)
[World](#)
[AU politics](#)
[Environment](#)
[Football](#)
[Indigenous Australia](#)
[Immigration](#)
[Media](#)
[Business](#)
[Science](#)
[Tech](#)

Greenhouse gas emissions

● This article is more than **8 months old**

Australia's carbon credit scheme 'largely a sham', says whistleblower who tried to rein it in

Prof Andrew Macintosh says the system, which gives credits for projects such as regrowing native forests after clearing, is 'a fraud' on the environment, taxpayers and consumers

● **Get our free news app; get our morning email briefing**

Adam Morton
Climate and environment editor

Wed 23 Mar 2022 19:35 AEDT



Source: The Guardian (fair use)



© 2023 Hound Technology, Inc. All Rights Reserved.

@lizthegrey at #GotoORD



Wasting less power is.

How to use less power:

- **Don't run the workload at all.** (do you really need very large language models or that fancy AI art?)
- **Optimize the workload to run faster/with fewer total resources.** Are you sure you're not wasting CPU somewhere?
- **Use more efficient technology.** Don't use a power hog.

We've seen amazing results with Graviton.

How did we scale economically?

AWS EC2 Spot, AWS Lambda, and AWS Graviton.



WTF is architecture? Why multiarch?

Instruction set architecture

Reference implementations & microarchitectures

Software support for all the above

(see Matt Godbolt's talk for the gorey details of ISAs, low-level perf, etc)



History: 80s, 90s, 00s, 10s, and beyond

In the beginning, there were mainframes (VAX, Z80, S370).

Then came desktop computing (Intel x86 and 68k).

Mini-frames/workstations (Alpha, POWER, MIPS, and SPARC).

The world goes 64-bit (Itanium, x86_64/amd64)

The mobile revolution (ARM32)

arm64, RISC-V, and the future



If it ain't broke...

x86_64/amd64 ecosystem is incumbent and healthy

Mobile was an edge case... or was it?

Open source and scale-out workloads changed the game



ARM is more efficient.

Why it's cheaper/power-efficient

- x86 is CISC
- Arm is RISC
- More of Arm CPU die dedicated towards just doing compute
- 7nm process node = less power consumption

Why it's faster

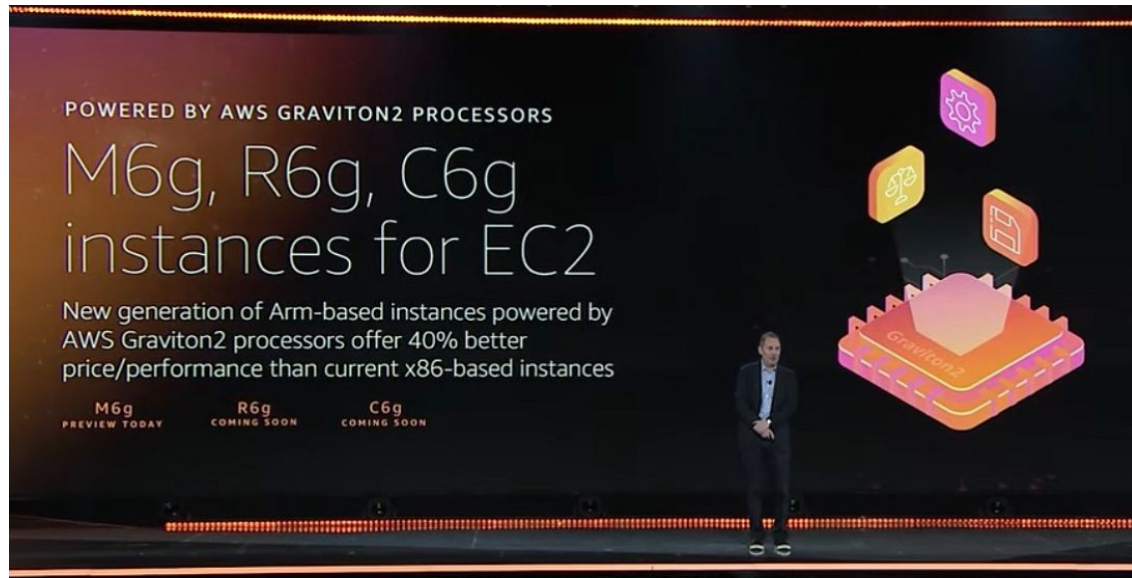
- x86 SMT: 2 vCPU = 1 execution unit
- Arm: 1 vCPU = 1 execution unit
- Arm execution units not shared between threads running on different vCPUs
- Less tail latency, performance variability



Graviton2 was announced Dec 2019

Promised improvements

- cost
- performance
- environmental impact



Andy Jassy announces Graviton2 instance types during keynote at AWS re:Invent 2019



“

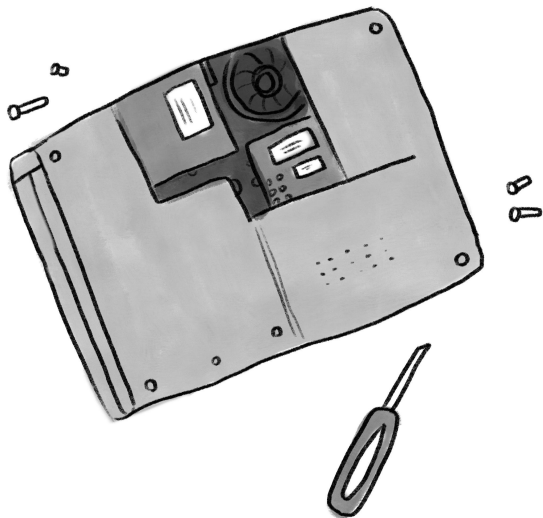
M6g instances are superior to C5 in every aspect—they cost less, have more RAM, exhibit lower median and significantly narrower tail latency, and run cooler with the same proportional workload per host. Converting our entire ingest worker fleet has allowed us to run 30% fewer instances, and each instance costs 10% less.

Yours Truly

Is it feasible to migrate?

What's needed?

- Base images & tooling (Docker or AMI)
- Audit application code for arch-specific code (e.g. inline assembly)
- CI tooling (producing build artifacts)



Producing artifacts for Arm64

Honeycomb uses Go

- Don't need an Arm box to cross-compile
- Need an Arm box to build Arm Docker images efficiently

Other languages

- Java, Python use arch-independent binaries, no changes needed
- C++ with hand-assembly would need updates

```

416 +         - go_build:
417 +             name: go_build_arm64
418 +             goarch: arm64
419 +             requires:
420 +                 - setup

```

[infra] cross-compile & package Honeycomb binaries for ARM #3688

 Merged lizthegrey merged 5 commits into master from lizf.arm on Feb 28, 2020

 Conversation 11  Commits 5  Checks 1  Files changed 1



lizthegrey commented on Dec 3, 2019 · edited

Summary:

cross-compiles all binaries for arm64

Asana Fixes:

n/a

Test Plan:

Run a Shepherd on an AWS R6g instance! (done! shepherd-0fde967783997cc55)

Feature Flag in Use (if applicable):

n/a

Docs Plan and/or Rollout Plan:

Dogfood first, so we can watch it with kibble. (done)



Initial findings


It built without errors! Now what?

Validating takes considerably more time & effort.

[chef,terraform] ARM64 support for honeycomb infra #657

Merged lizthegrey merged 1 commit into `master` from `lizf.arm64` on Mar 2, 2020

Conversation 18 · Commits 1 · Checks 1 · Files changed 14

 lizthegrey commented on Feb 26, 2020 · edited

Summary:
pulls linux_arm64 (go's word for ARM64) tarball if `uname -m` returns aarch64 (Linux's word for ARM64). use the appropriate chef recipes throughout. currently we skip osquery in the security cookbook as it has heavy AMD64-only dependencies.

Depends upon [honeycombio/hound#3688](#)

Asana Fixes:
n/a

Test Plan:
Run deploy script on both an amd64 and arm64 machine before making the default chef script.

Docs Plan and/or Rollout Plan:
deployed to `shepherd-0fde967703997cc55` and verified working.



A/B testing

Limited variables

- same build ID (different compilation targets)
- single service

Slow rollout

- started with one instance
- bumped to 20% to observe

```

46 + variable "shepherd_instance_count_arm" {
47 +   type      = map(number)
48 +   description = "Number of experimental ARM shepherd instances to maintain"
49 +   default = {
50 +     dogfood    = 1
51 +     production = 0
52 +     kibble     = 0
53 +   }
54 + }
55 +

```

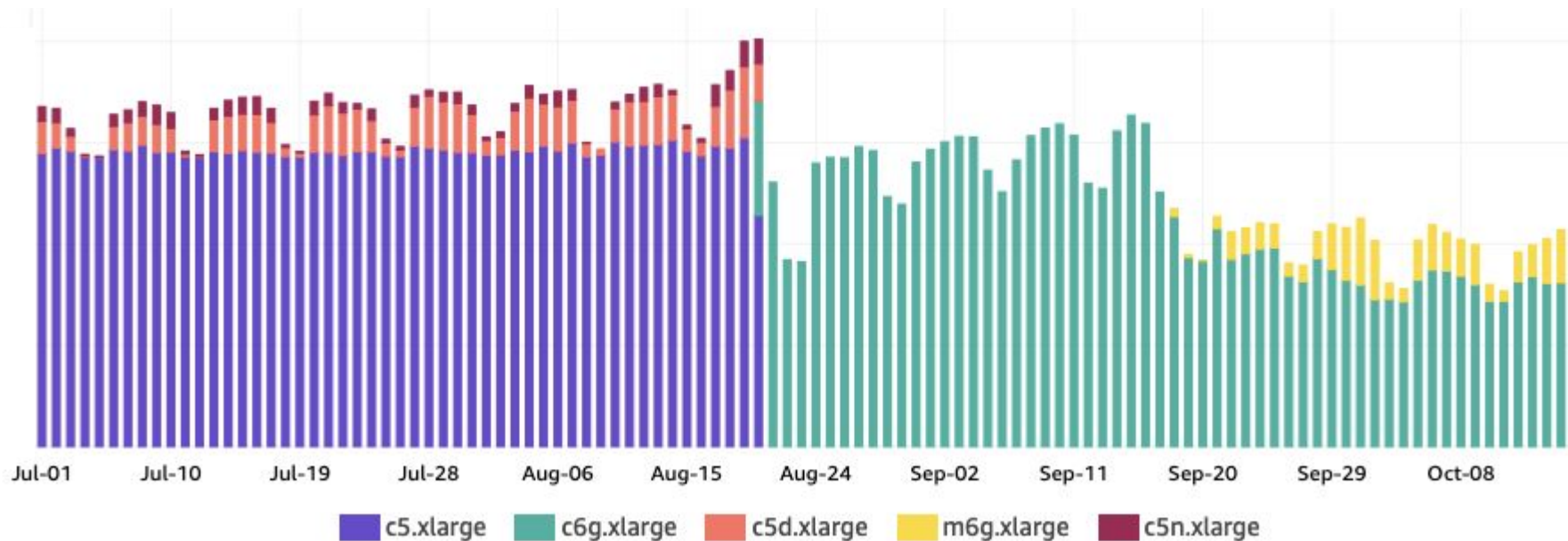
```

46 variable "shepherd_instance_count_arm" {
47   type      = map(number)
48   description = "Number of experimental ARM shepherd instances to maintain"
49   default = {
50 -     dogfood    = 1
50 +     dogfood    = 3
51   production = 0
52   kibble     = 0

```



Migrated prod Shepherd



Production Shepherd EC2 cost, grouped by instance type



Migrated prod Retriever

Retriever is our query engine


- Cost savings wasn't a goal
- Instead, we tuned performance

For a 10% increase in cost, we could get a 3x performance improvement!

Headroom was critical for demand growth

Triggers Successful Timely Runs

Triggers are run at a configured interval, and don't repeat query ranges over the same dataset. Missing a trigger run means a given item won't be seen, and if their runs start overlapping, we end up querying the same time range multiple time while missing others. We want to ensure that we're able to run triggers in a timely and sustainable manner, so that people can properly be alerted for misbehaving systems or notified of rare events. Additionally, no errors in execution are found, aside from user-triggered errors (bad webhook config, error in types usage) Since triggers are all or nothing, we're being more demanding in terms of their reliability (at 99.95%) and can readjust over time.

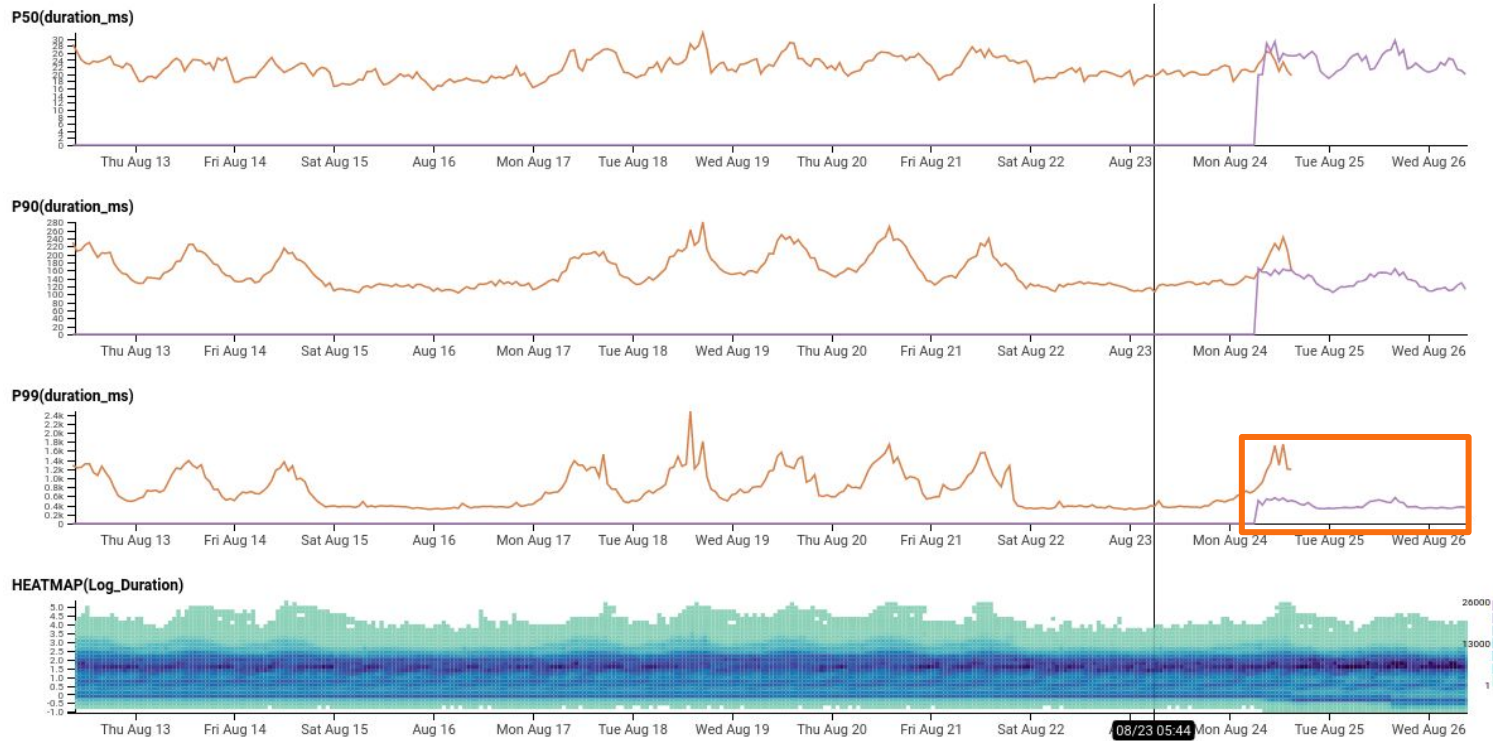
99.95% of eligible events from the  **basset** column

 `trigger_delay_to_frequency_ratio_acceptable_no_error` will succeed over a period of 30 days.

Budget Burndown

How much of the error budget remains after the last 30 days. Starts at 100% and burns down.





	global.instance_type	P50(duration_ms)	P90(duration_ms)	P99(duration_ms)	HEATMAP(Log_Duration)
	m6gd.2xlarge	23.38858	132.53309	399.67831	
	i3.xlarge	21.35196	153.60148	888.1141	

Production Retriever migration



Retriever traffic volume and Graviton2 migration

f= VISUALIZE

COUNT

HEATMAP(Log_Duration)

WHERE

name = ReadRows

service_name = retriever

AND

GROUP BY

global.instance_type

HAVING

No constraints

ORDER BY

COUNT desc

Run Query

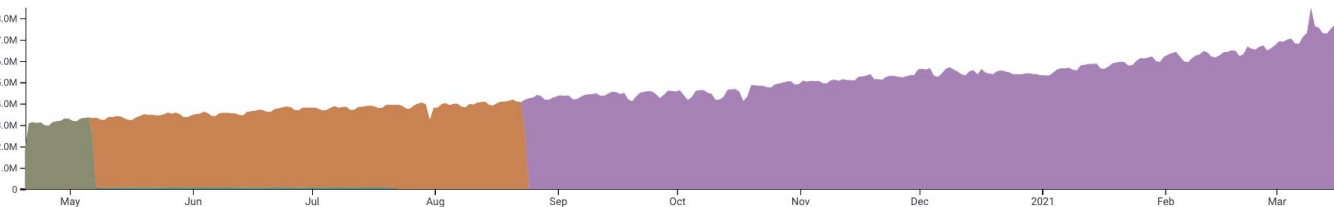
Run 7 minutes
ago

Apr 20 2020, 12:00 AM – Mar 18 2021, 12:00 AM Granularity: 1 day

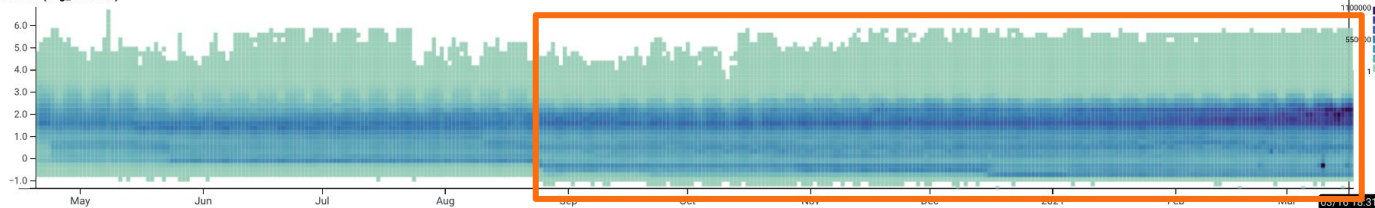
Results BubbleUp Traces Raw Data

Graph Settings

COUNT



HEATMAP(Log_Duration)



	global.instance_type	COUNT	HEATMAP(Log_Duration)
	m6gd.2xlarge	1,116,503,460	
	i3.xlarge	400,737,554	
	m6g.xlarge	55,313,544	
	m6g.2xlarge	4,592,316	

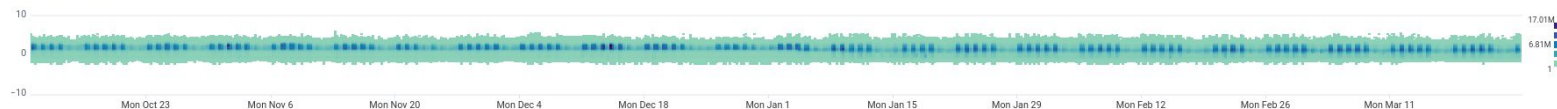
elapsed query time: 2m38.037311359s rows examined: 459,833,822,980 nodes reporting: 100%



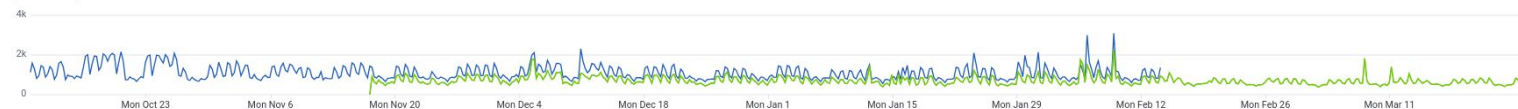
Query Results

Oct 10 2023 00:21 – Mar 25 2024 21:32 UTC-07:00 (Granularity: 6 hour)

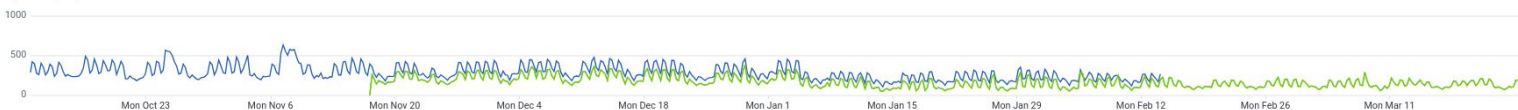
HEATMAP(Log_Duration)



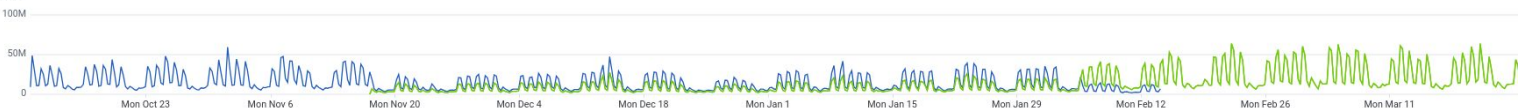
P99(duration_ms)



P90(duration_ms)



COUNT



COUNT_DISTINCT(meta.local_hostname)



Overview BubbleUp Correlations Traces Explore Data

global.instance_type	HEATMAP(Log_Duration)	P99(duration_ms)	P90(duration_ms)	COUNT	COUNT_DISTINCT(meta.local_hostname)
m6gd.4xlarge		1,322.81726	356.94113	7,392,357,361	240
m7gd.4xlarge		777.66382	198.69076	7,274,711,722	240

elapsed query time: 17.122966846s



30% improved throughput with AWS Graviton3



30% improved throughput with AWS Graviton3



30% improved throughput with AWS Graviton3



30% improved throughput with AWS Graviton3

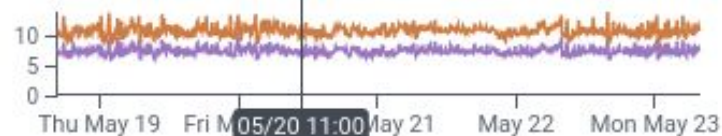
SUM(amazonaws.com/AWS/ApplicationELB/RequestCount.count)



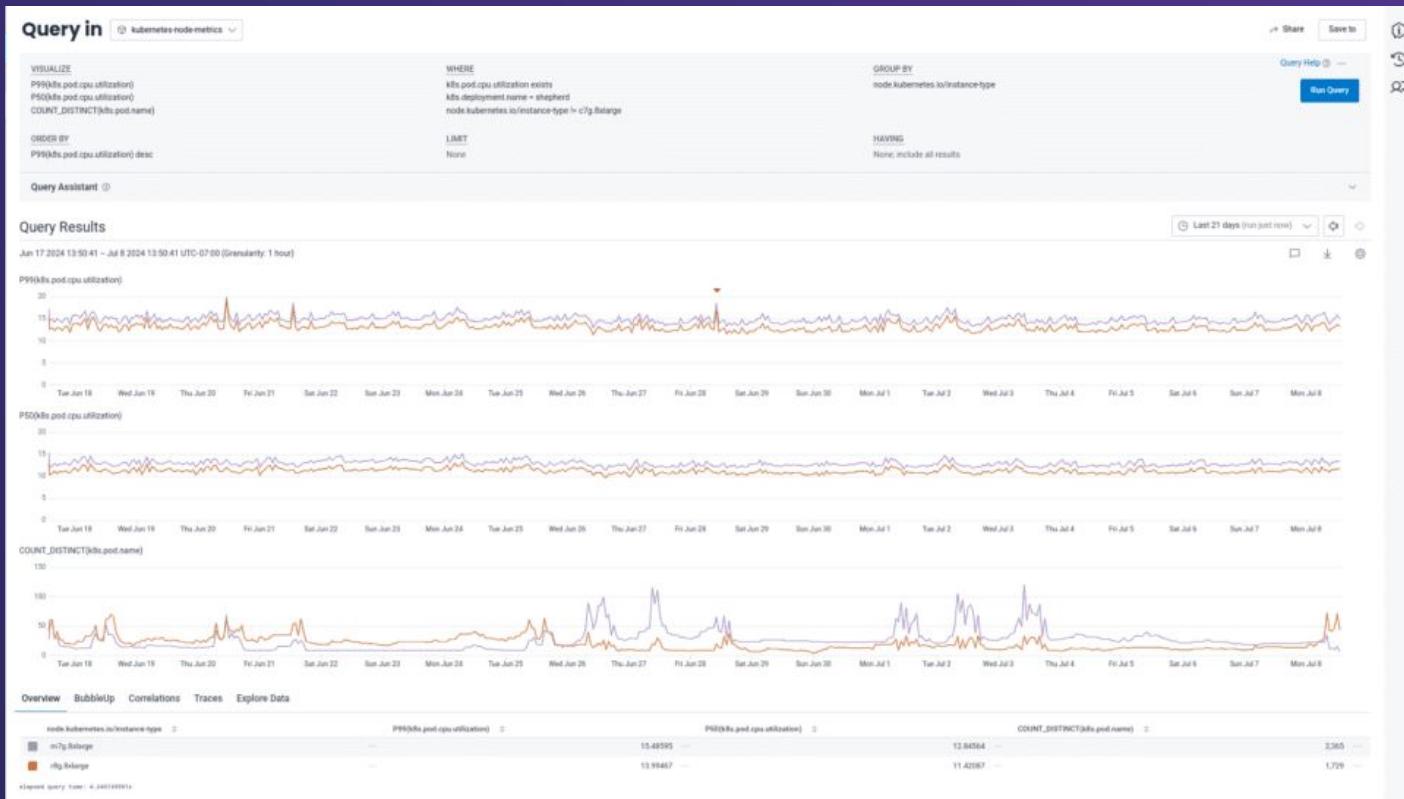
COUNT_DISTINCT(k8s.pod.name)



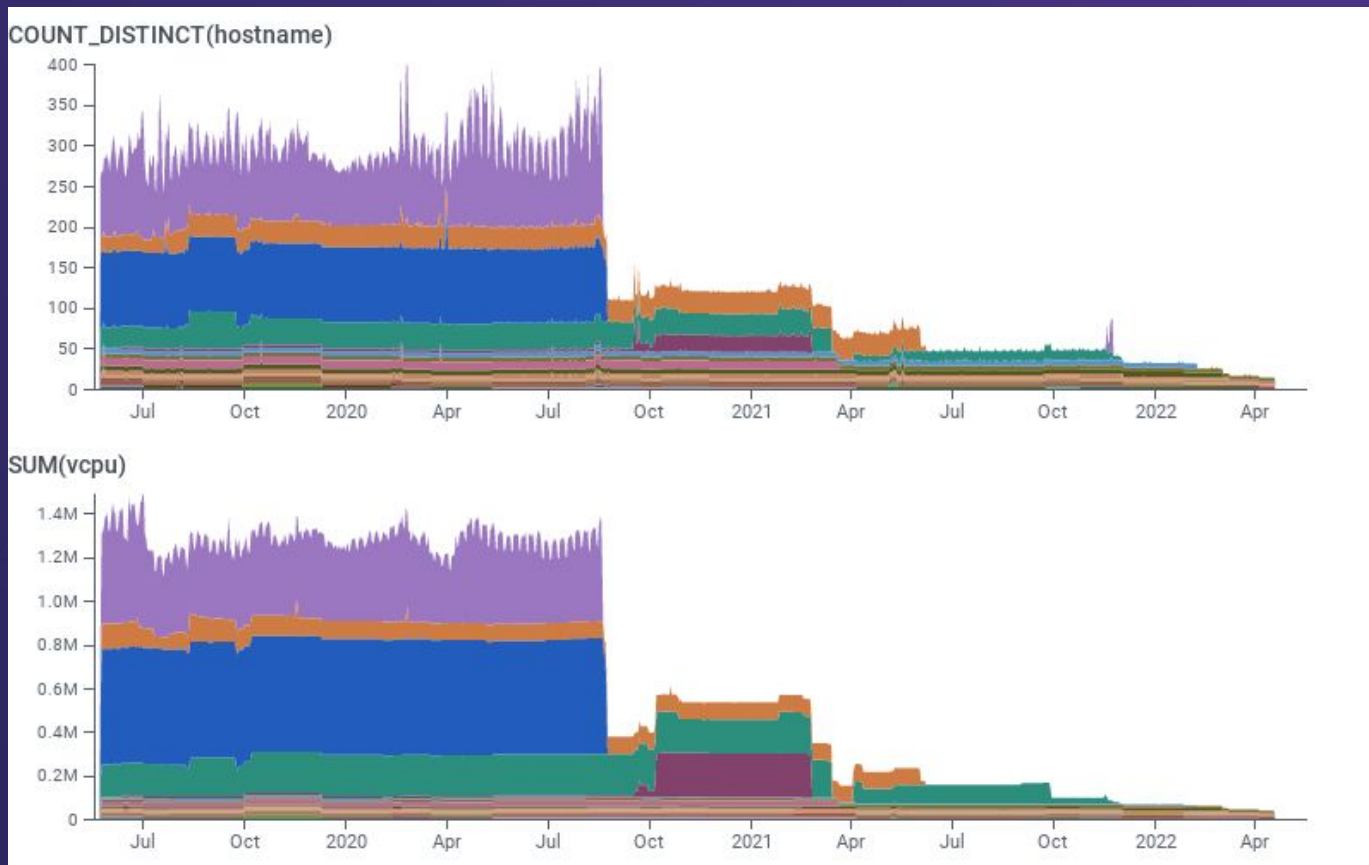
P99(metrics.cpu.usage)



AWS Graviton4 is 30% better than AWS Graviton3.



We've been 100% ARM servers for 2.5 years.

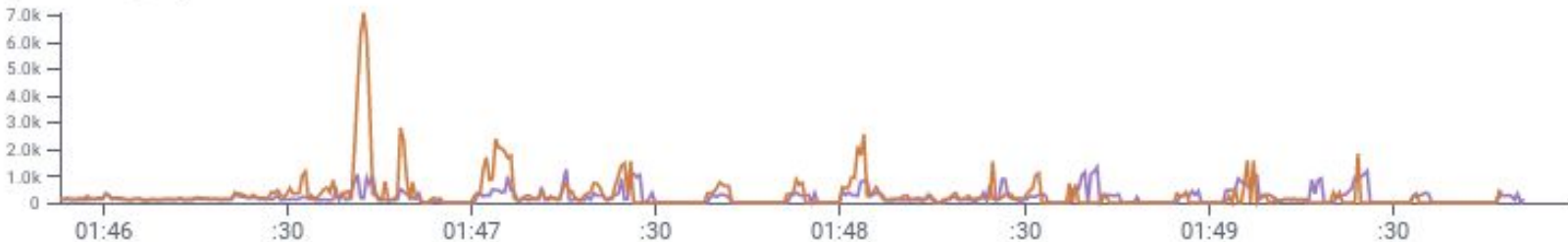


Observability helps!

P99(duration_ms)



P50(duration_ms)



arch	COUNT	HEATMAP(Log_Duration)	P99(duration_ms)	P50(duration_ms)
amd64	262,988		1,168.09377	139.24663
arm64	161,394		2,677.62006	175.50275



LaunchDarkly APP 6:48 PM

Liz Fong-Jones updated the flag **Retriever Lambda ARM Percentage**

- Added the variation **1% ARM**

Liz Fong-Jones updated the flag **Retriever Lambda ARM Percentage** in **Production**

- Changed the default variation from **50% ARM** to **1% ARM**



lizf 🌙 6:49 PM

reverting ARM experiment, just keeping a trickle on 1% for validation of non-breakage/dogfooding of the lambda layer on both archs. it was 20% slower at p50 and 100% slower at p99, so we need to roll back.



1



1



1



1 reply 17 days ago





COUNT

arch exists

arch

Run a few
seconds agoORDER BY

COUNT desc

LIMIT

None

HAVING

None; include all results



Results BubbleUp Metrics Traces Raw Data

☐ Compare to

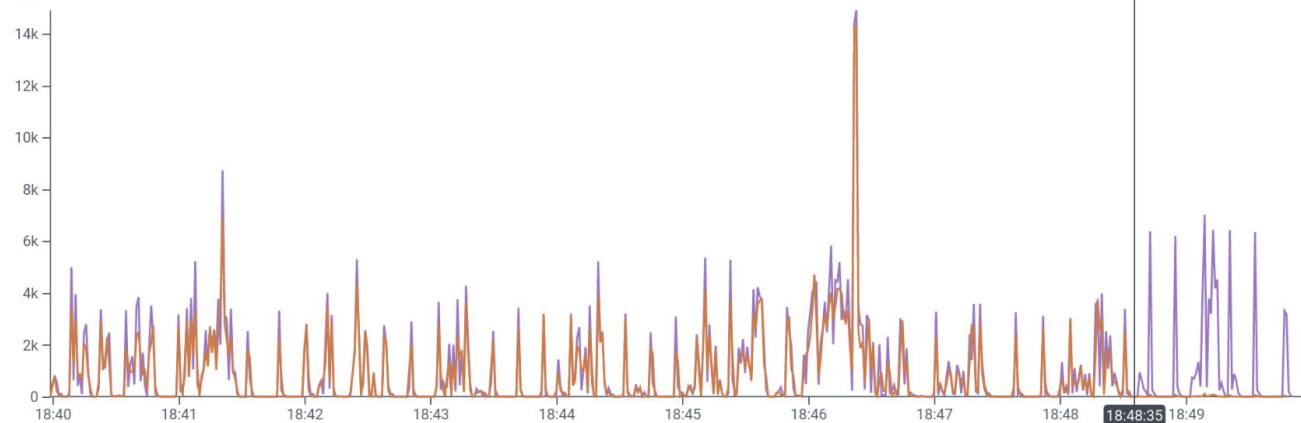
10 minutes prior



Graph Settings

Oct 1 2021, 6:39:59 PM – Oct 1 2021, 6:49:59 PM (Granularity: 1 sec)

COUNT



arch

COUNT



amd64

583,172

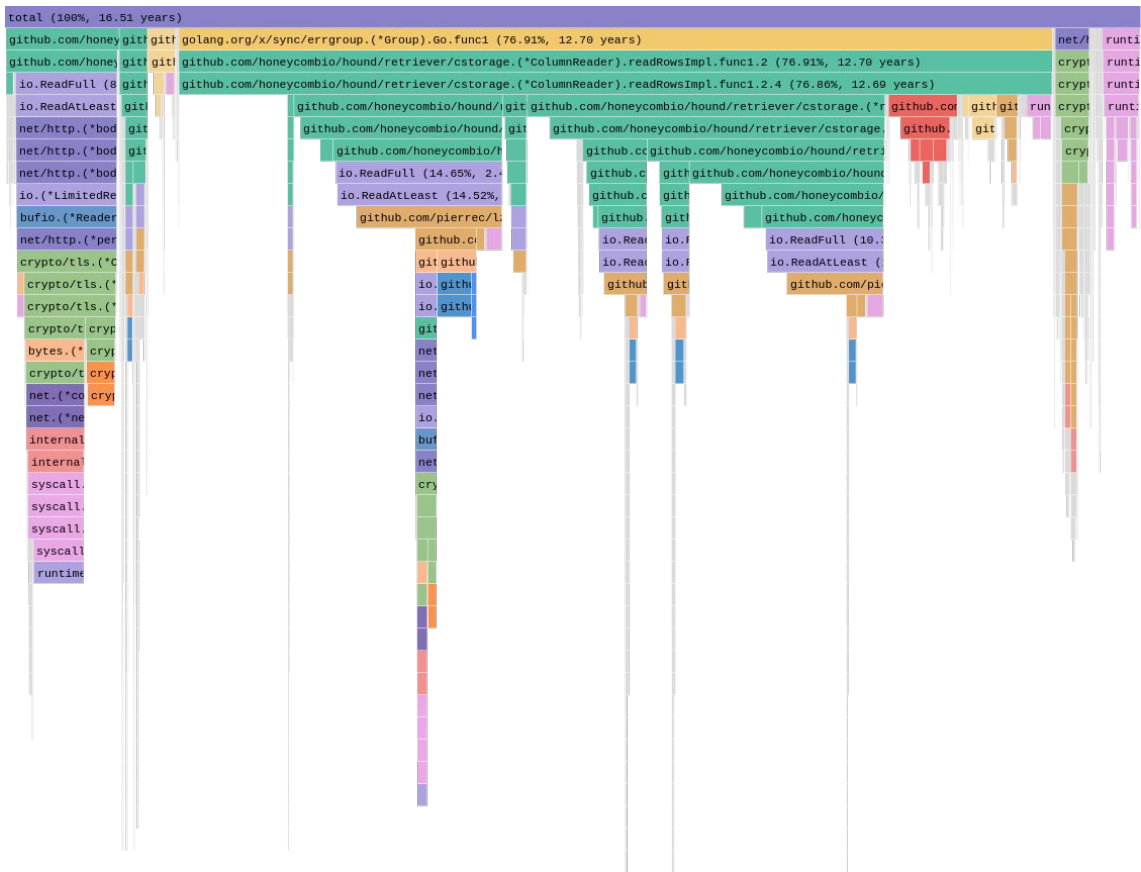


arm64

455,704



- AWS capacity constraints
- Go register calling convention
- lz4 library asm optimization
- (and TLS cert validation haha)



Making progress carefully



LaunchDarkly APP 11:06 AM

Liz Fong-Jones turned on the flag **Profile Lambda Percent** in **Production**

Liz Fong-Jones scheduled changes for the flag **Profile Lambda Percent** in **Production**

- Changes will occur on **Sat, 16 Oct 2021 18:15:00 UTC**
- Turn off the flag

Liz Fong-Jones scheduled changes for the flag **Retriever Lambda ARM Percentage** in **Production**

- Changes will occur on **Sat, 16 Oct 2021 18:20:00 UTC**
- Update default variation to **serve 1% ARM**



LaunchDarkly APP 11:15 AM

Completed scheduled changes to the flag **Profile Lambda Percent** in **Production** (via API)

- Turned the flag off



**VISUALIZE**

SUM(Fraction)

WHERE

None; include all events

GROUP BY

Resource

...

Run QueryRun a few
seconds ago**ORDER BY**

SUM(Fraction) desc

LIMIT

None

HAVING

None; include all results

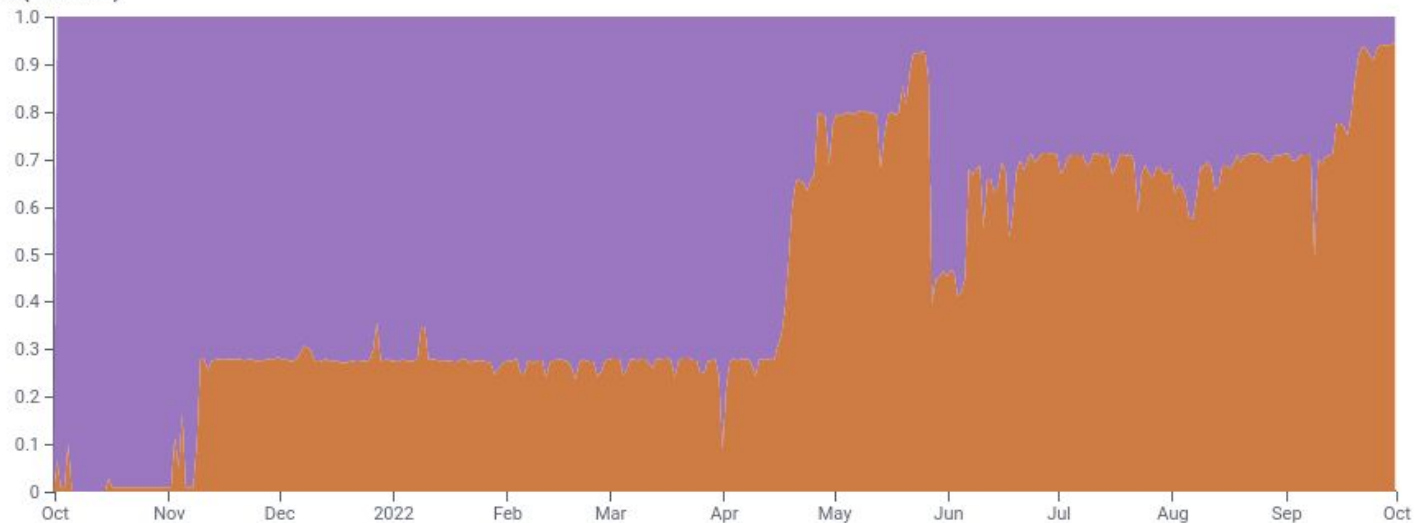
Results BubbleUp Traces Raw Data☐ Compare to

Previous time range ▾

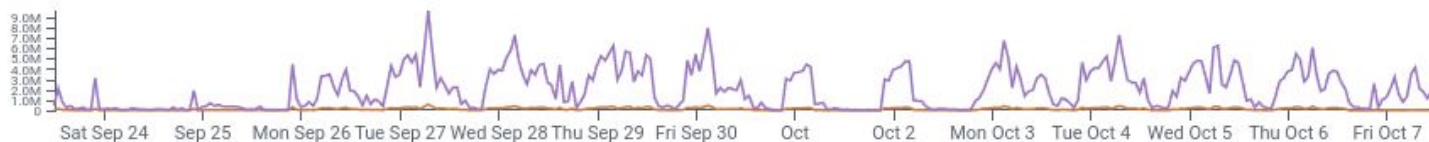
Graph Settings

Oct 1 2021, 12:00 AM – Oct 1 2022, 12:00 AM (Granularity: 1 day)

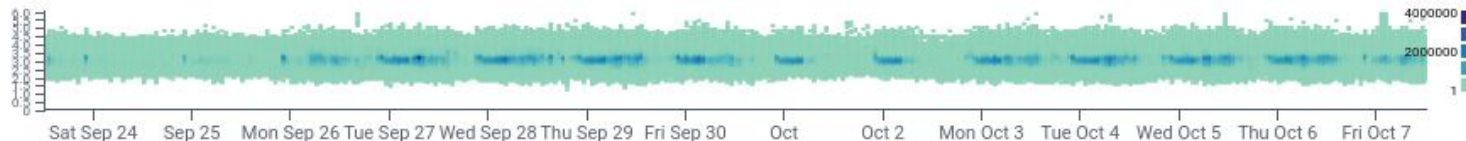
SUM(Fraction)



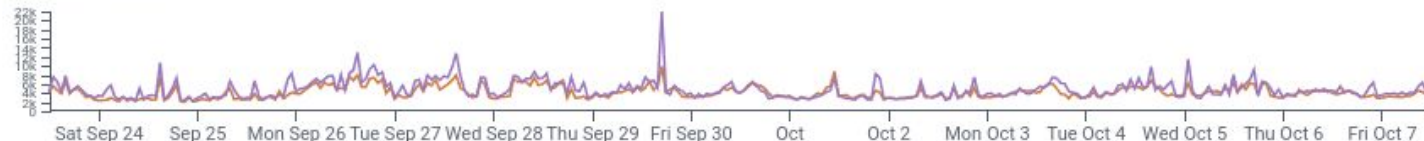
COUNT



HEATMAP(Log_Duration)



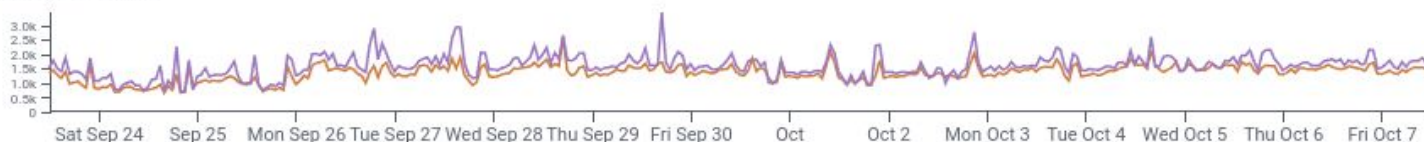
P99(duration_ms)



P50(duration_ms)



AVG(duration_ms)



~ 4 to change 1 to destroy

Filter resources by address... Filter by action Show data sources Tenaforn 1.7.5 Download raw log

▼ **aws** module.lambda.aws_lambda_function.retriever-segment-processor

```

- architectures :      [
  -   "x86_64"
  ] → null
- arn :                "arn:aws:lambda:us-east-1:702835727665:function:retriever-segment-processor-production"
- code_sha256 :       "hqvp2ia1JumVLxQVQFdmUNRjzEIK2TVgq0InvTQ/avV="
- code_signing_config_arn : ""
- description :       ""
- filename :          "../modules/lambda/lambda_placeholder.zip"
- function_name :     "retriever-segment-processor-production"
- handler :           "retriever-lambda"
- id :                "retriever-segment-processor-production"
- image_uri :         ""
- invoke_arn :        "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:702835727665:function:retriever-segment-processor-production/invocations"
- kms_key_arn :       ""
- last_modified :     "2024-10-21T14:51:06.000+0000"
- layers :            [
  -   "arn:aws:lambda:us-east-1:702835727665:layer:honeycomb-lambda-extension-x86_64-v11-1-2:1"
  ] → null
- memory_size :       5307
- package_type :      "Zip"
- publish :           false
- qualified_arn :      "arn:aws:lambda:us-east-1:702835727665:function:retriever-segment-processor-production:7794"
- qualified_invoke_arn : "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:702835727665:function:retriever-segment-processor-production:7794/invocations"
- reserved_concurrent_executions : -1
- role :              "arn:aws:iam::702835727665:role/retriever-lambda"
- runtime :           "provided.al2023"
- signing_job_arn :   ""
- signing_profile_version_arn : ""
- skip_destroy :      false
- source_code_hash :  "h+4y/9fOE3IdcGSnctJjmoFwrvWVBwH1HFZ80krlhsI="
- source_code_size :  33644216
- tags :              { }
- tags_all :          {
  -   Environment :      "production"
  } → null
- timeout :           900
- version :           "7794"
- environment {
  -   variables :        {
    -   ENV :            "production"
    -   LIBHONEY_API_HOST : "https://api-dogfood.honeycomb.io"
    -   LIBHONEY_API_KEY : Sensitive value
    -   LIBHONEY_DATASET : "retriever-traces"
  }
  }

```

Yes*, do this at home!

Most realtime bulk workloads benefit

- **Move** state from local machines onto object storage
- **Shard** list of objects into work units
- **Parallelize** object processing
- **Reduce** results outside Lambda afterwards



Just beware the dragons

- **Avoid latency-insensitive** batch workloads (cost)
- **Avoid tiny** workloads (set-up latency)
- Check **cloud provider limits**, state your intentions (capacity planning)
- **Test cross-compilation** including profiling to avoid performance issues with switching archs.



Do this before scaling out

- Ensure it's **tuned properly** (items/invoke, CPU/RAM ratio)
- Ensure your code is **optimized properly** (esp if multi-arch)
- Ensure you use **observability layers** (e.g. OTel layer)
- Measure **metrics** carefully (esp cost)



Arm64 is a major cost & planet saver

- **Configure** your CI pipelines to make artifacts
- **Prepare** multi-arch clusters
- **Deploy** to production
- **Measure** the results
- **Iterate** to maximise cost savings



“

**Remember: nothing matters
unless users (developers) are happy**



Observability Engineering

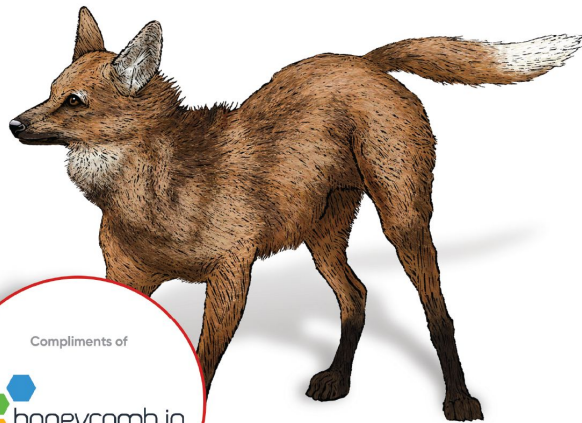
Chapter 16: Efficient Data Storage



O'REILLY®

Observability Engineering

Achieving Production Excellence



Charity Majors,
Liz Fong-Jones
& George Miranda

Benefits of AWS Graviton Processors

2.2x

Price-perf,
Amazon EC2
8th gen vs
5th gen.

20x

Scaling over past
5 years.
SLOs and budgets
met, no sweat!

1.6x

Cost efficiency per
ingested trace
(while adding new
features)





Go forth ARMed with this info!



www.honeycomb.io

