

# Adopting Graviton2

---

How Honeycomb Reduced Infra Spend by 40%  
On Its Highest-Volume Service





# Shelby Spees

Developer Advocate at Honeycomb.io

 @shelbyspees



# Why Graviton2?

## Promised improvements

- cost
- performance
- environmental impact

POWERED BY AWS GRAVITON2 PROCESSORS

# M6g, R6g, C6g instances for EC2

New generation of Arm-based instances powered by AWS Graviton2 processors offer 40% better price/performance than current x86-based instances

Instance Type	Status
M6g	PREVIEW TODAY
R6g	COMING SOON
C6g	COMING SOON

The slide features a large, stylized illustration of a Graviton2 processor chip in the center-right, with three floating icons above it: a gear (settings), a dollar sign (cost), and a document (performance). A person is standing on the stage to the right of the slide.

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



# More efficient processor architecture

---

## Why it's cheaper/power-efficient

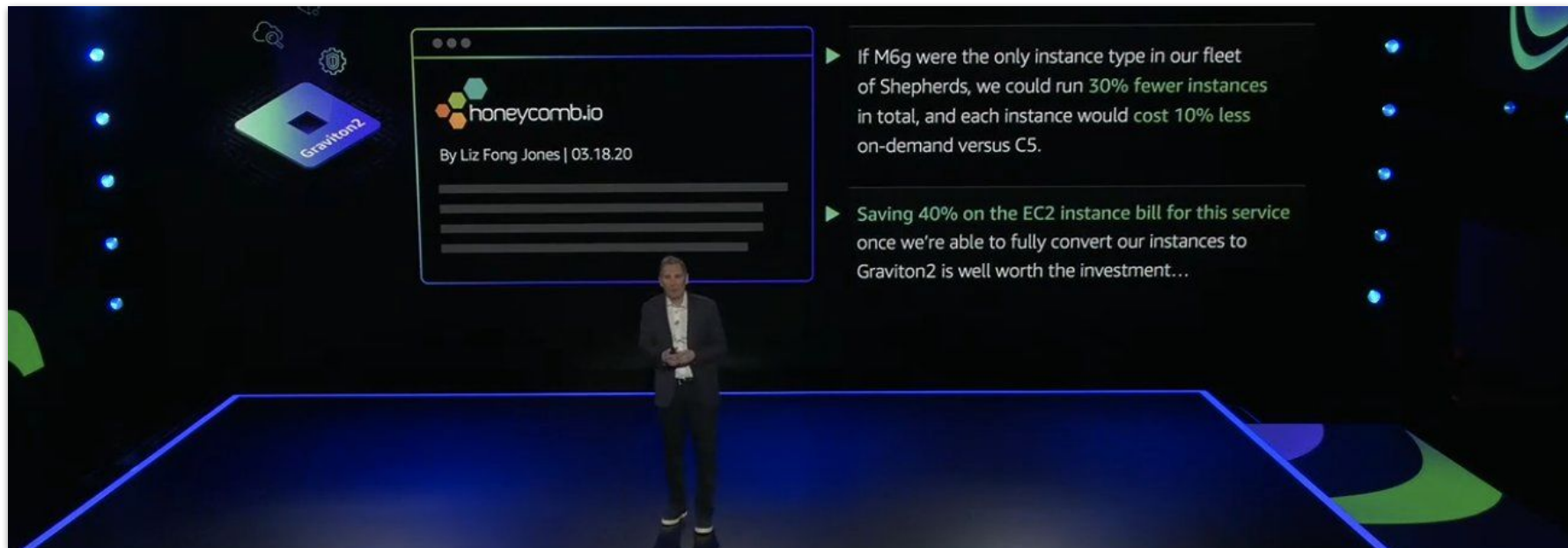
- x86 is CISC
- Arm is RISC
- More of Arm CPU die dedicated towards just doing compute
- 7nm process node = less power consumption

## 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



# One year later



Andy Jassy talks about Honeycomb during keynote at AWS re:Invent 2020



# Is it worth the RISC?

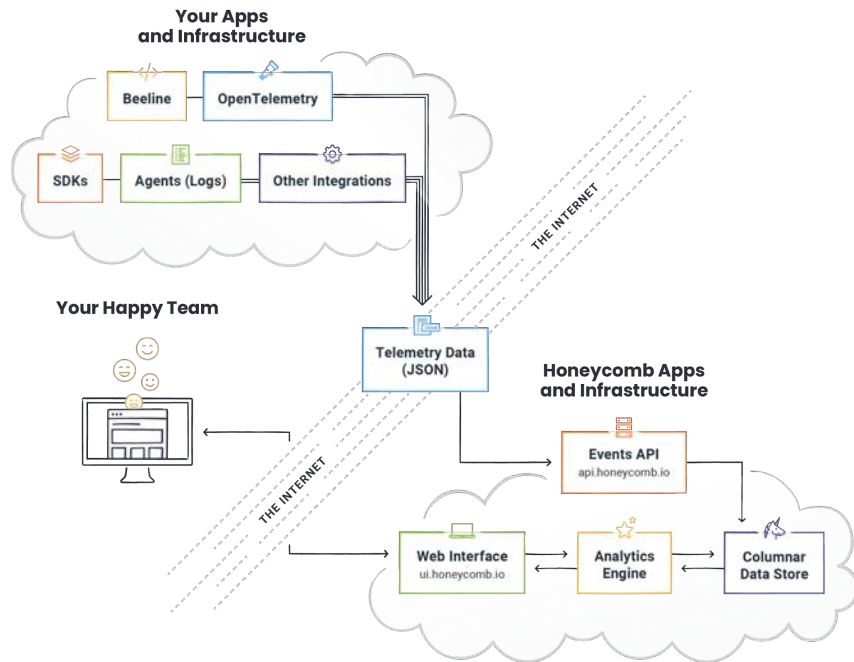
---

What's important to Honeycomb?

# Data storage engine and analytics tool

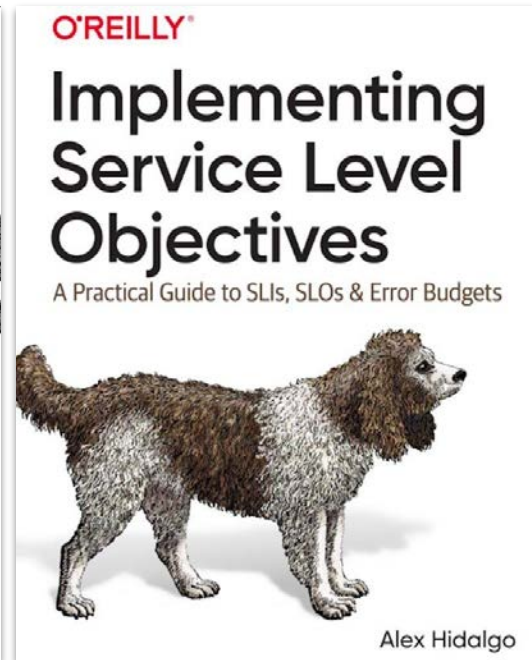
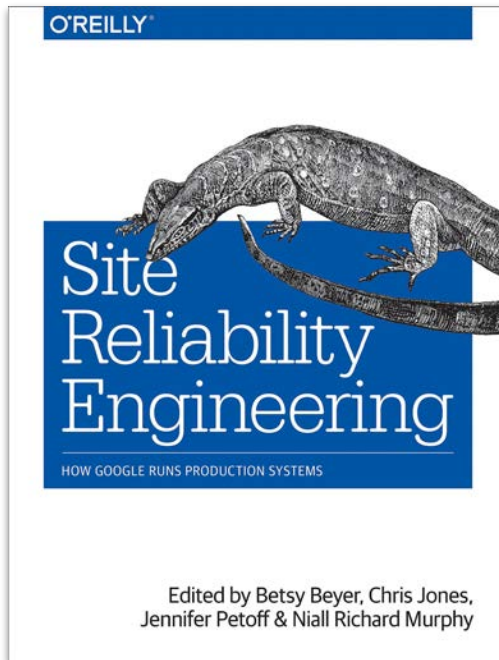
## What Honeycomb does

- Ingests customer's telemetry
- Indexes on every column
- Enables near-real-time querying on newly ingested data



# Service Level Objectives (SLOs)

Common language between engineers  
and business stakeholders





# SLOs are user flows

## Honeycomb's SLOs

- home page loads quickly
- user-run queries are fast
- customer data gets ingested fast



## Latency per-event

Shepherd ingestion latency should be below 5ms per event within a batch. We ignore values from user-triggered issues, deprecated endpoints we won't support as extensively as the main ones, and also ignore values coming from collectd which historically was a misbehaving client whose API we don't control.

99.99% of eligible events from the `shepherd` column `sli` 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.

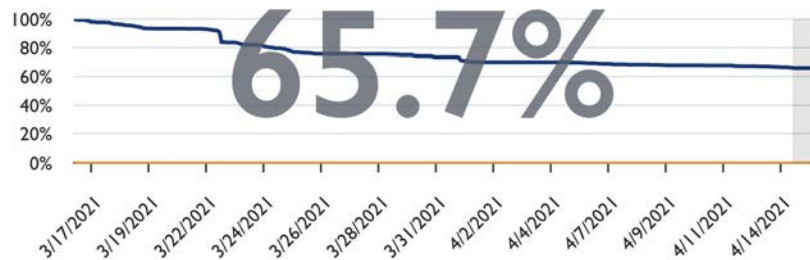


# Error budget: allowed unavailability

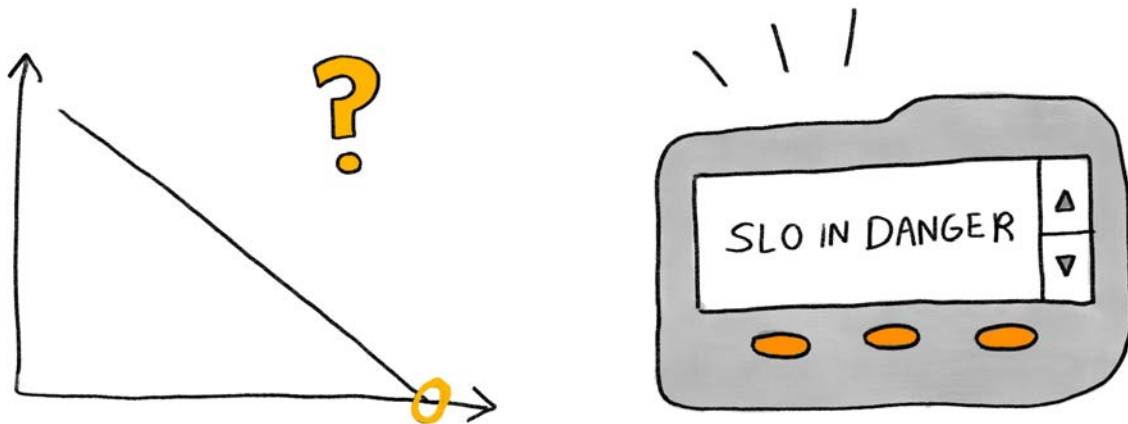
1 - X

## Budget Burndown

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



# Alert proactively based on budget burn rate



# Period of reliability = time to cut costs

---

Infra is our #2 expense after taking care of our honeybees

Infra cost scales with traffic

"Cost of Goods Sold" and other business acronyms

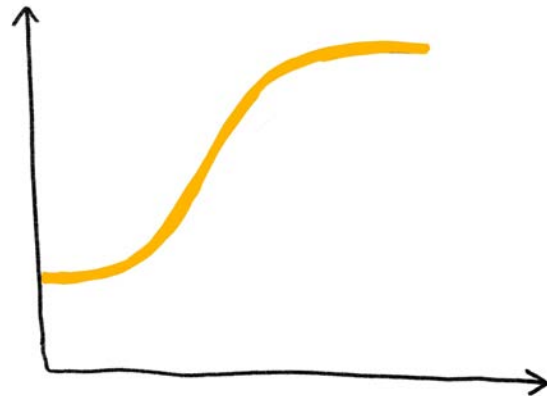


# **Choosing where to start**

---

# Prod: customers observe data

---



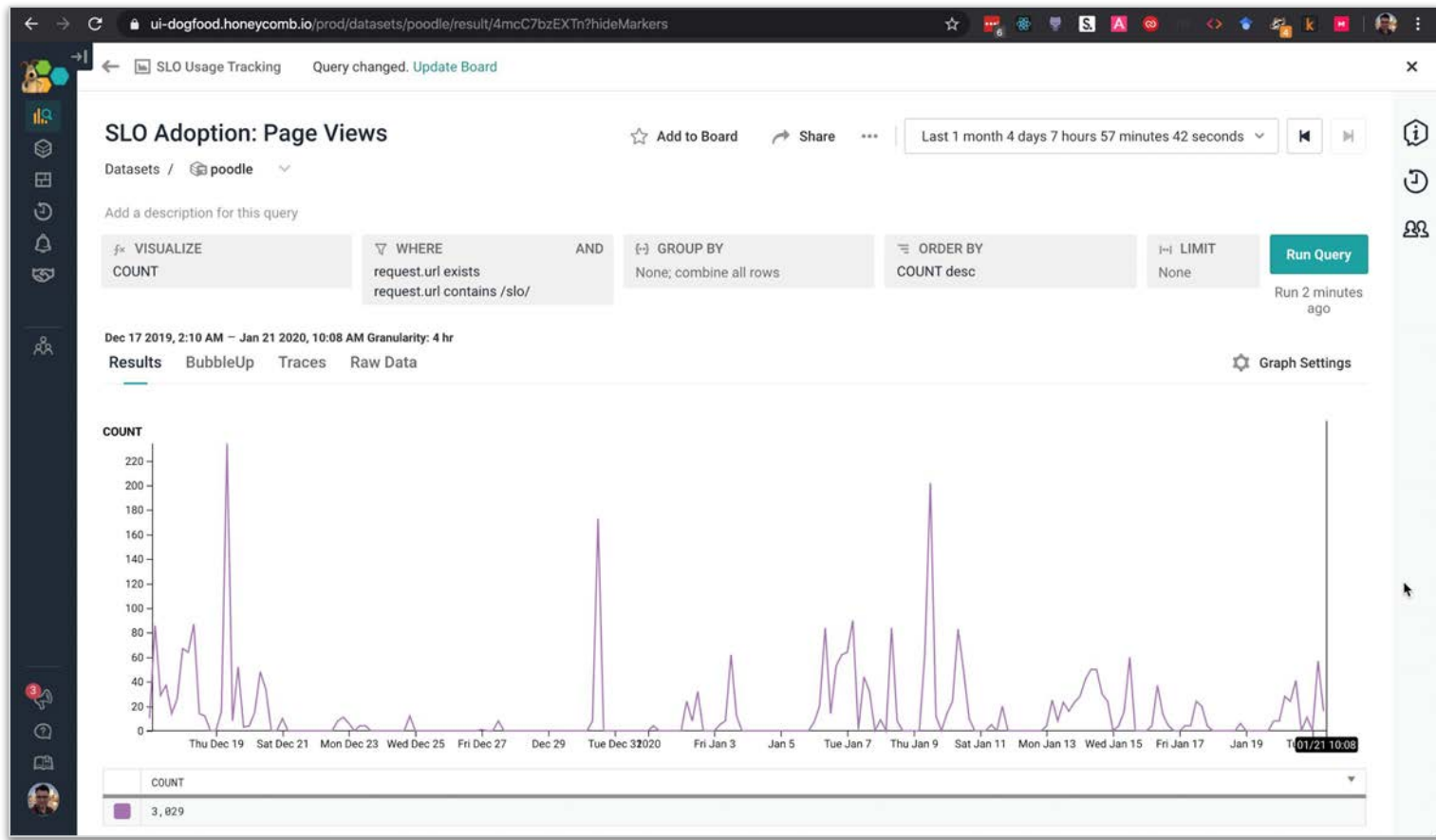
# Dogfood observes prod

---

Production telemetry → Dogfood ingest

Same code as production







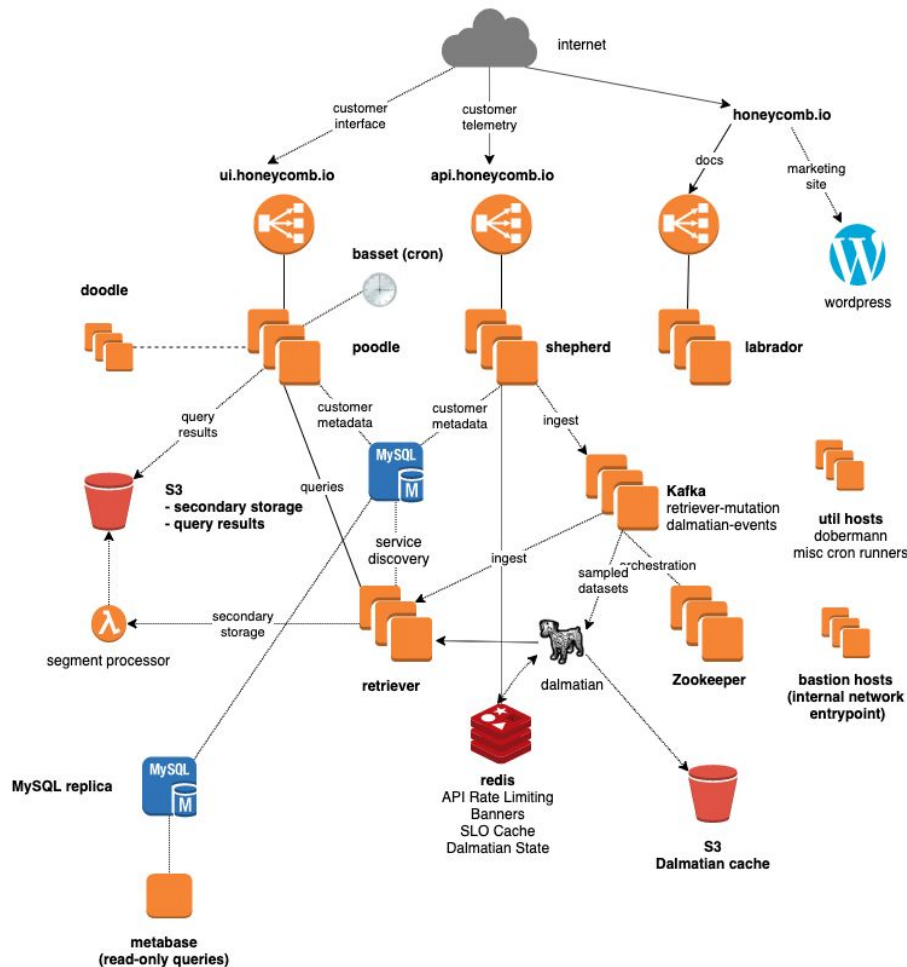
# Kibble observes dogfood

---



## Honeycomb's services

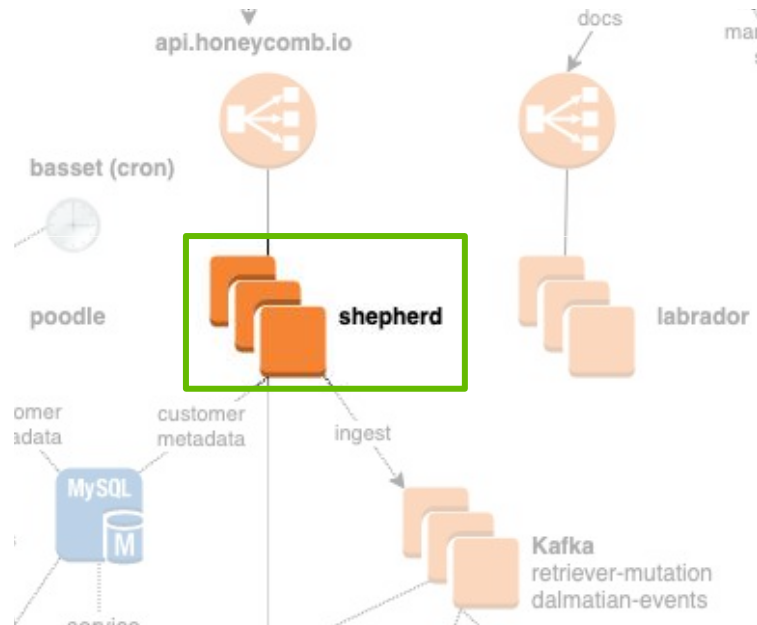
- shepherd (ingest API)
- kafka (ingest event streaming)
- retriever (indexing and querying)
- poodle (frontend web app)
- refinery (sampling)
- doodle (images)
- labrador (docs, bins, nginx redirects)
- basset (alerting, lives on poodle)
- basenji (encryption)



# Shepherd: ingest API service

## Why Shepherd?

- highest-traffic service
- stateless, most straightforward
- only scales on CPU utilization
- cares about throughput first, latency close second



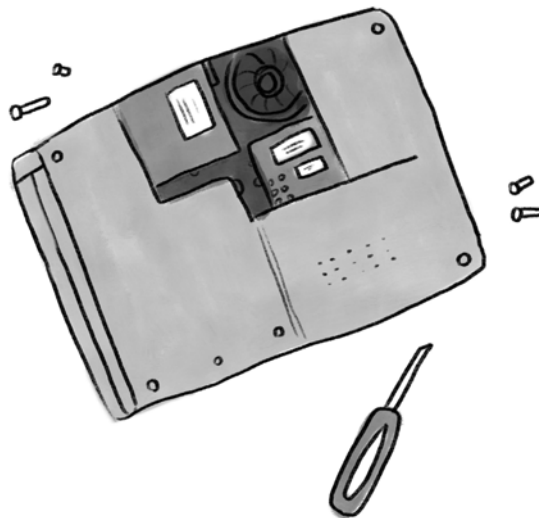
# **Preparing to test out the change**

# 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-0fde967703997cc55)

### Feature Flag in Use (if applicable):

n/a

### Docs Plan and/or Rollout Plan:

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

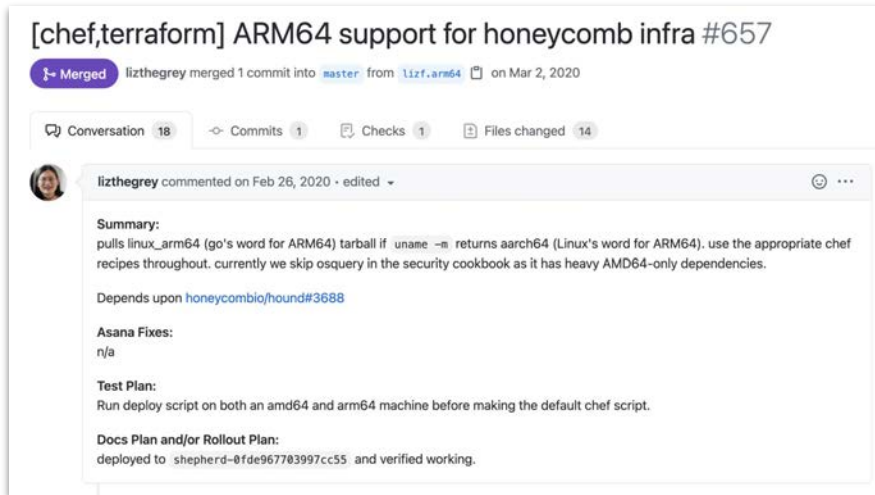


# Initial findings

## m6g is superior to c5 for our workloads

- lower cost on-demand
- more RAM
- lower median latency
- significantly lower tail latency

Cost of this experiment?  
A few spare afternoons.



# 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

```



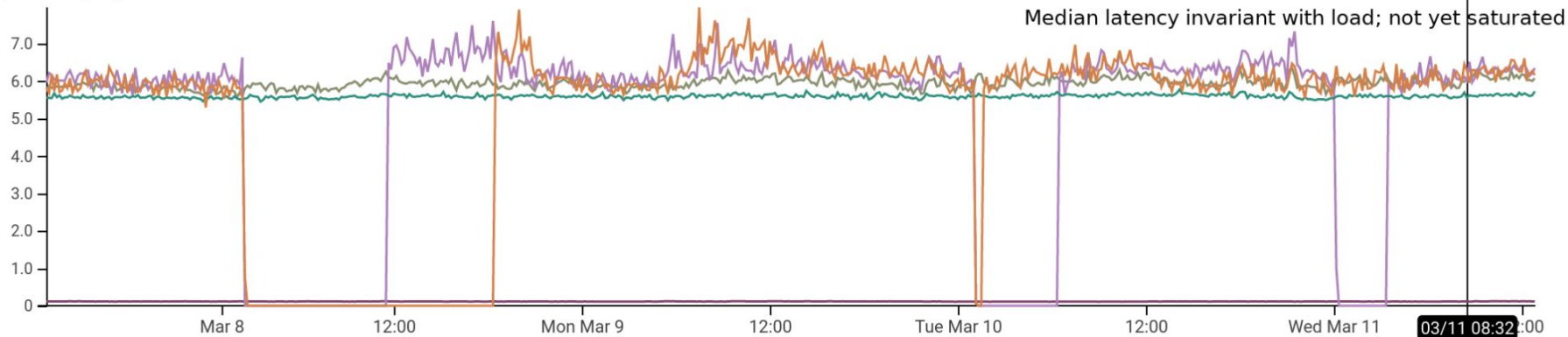




Distribution of request latency on different instance types



P50(duration\_ms)



	global.instance_type	HEATMAP(duration_ms)	P50(duration_ms)	
	c5d.xlarge		6.26814	
	c5n.xlarge		6.15874	
	c5.xlarge		5.93416	
	m6g.xlarge		5.59069	10% faster median latency



f× VISUALIZE

HEATMAP(cpu\_user)  
P50(cpu\_user)  
COUNT\_DISTINCT(hostnam

WHERE

host\_group = shepherd  
environment = dogfood  
instance\_type =  
m6g.xlarge

AND

GROUP BY

None; combine all rows

ORDER BY

P50(cpu\_user) desc

LIMIT

None

Run Query

Run a few  
seconds ago

Mar 7 2020, 12:45 PM – Mar 11 2020, 12:45 PM Granularity: 15 min

Results

BubbleUp

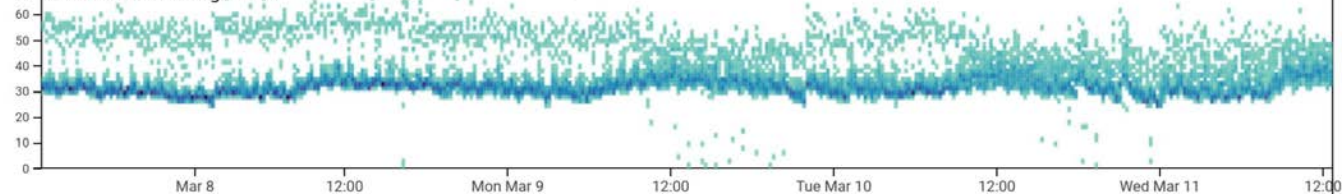
Traces

Raw Data

Graph Settings

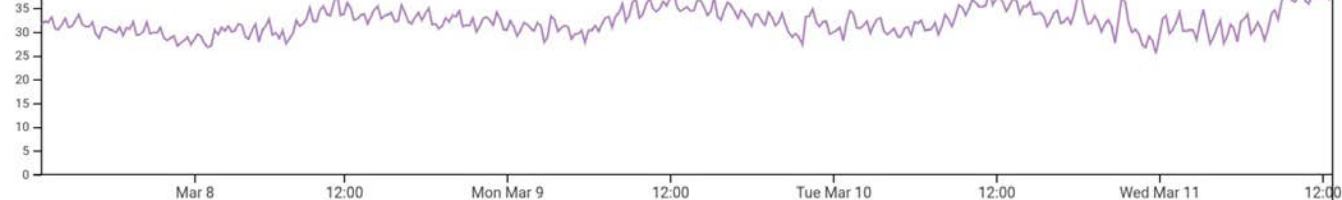
HEATMAP(cpu\_user)

Maximum CPU usage 60%



P50(cpu\_user)

Median CPU utilization 32-38%



CPU utilization on old architecture



fx VISUALIZE  
HEATMAP(cpu\_user)  
P50(cpu\_user)  
COUNT\_DISTINCT(hostnam

WHERE AND  
host\_group = shepherd  
environment = dogfood  
instance\_type !=  
m6g.xlarge

GROUP BY  
None; combine all rows

ORDER BY  
P50(cpu\_user) desc

LIMIT  
None

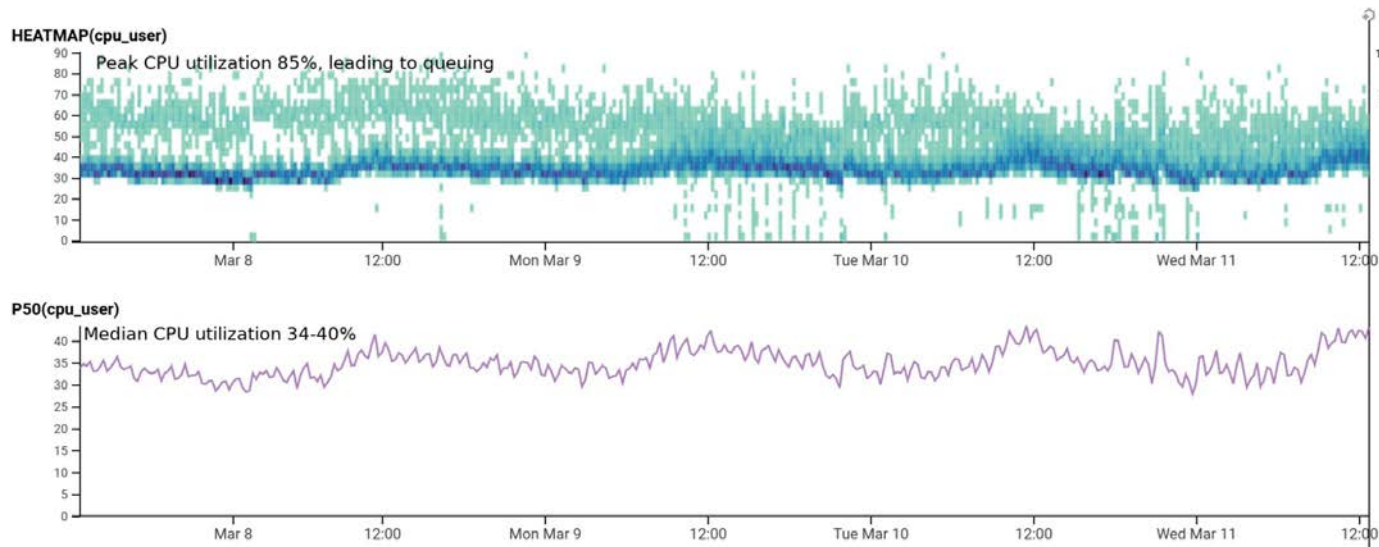
Run Query

Run a few  
seconds ago

Mar 7 2020, 12:45 PM – Mar 11 2020, 12:45 PM Granularity: 15 min

Results BubbleUp Traces Raw Data

Graph Settings



CPU utilization on Graviton2 instances



## vCPU and number of hosts, x86 vs. Arm64

f- VISUALIZE

SUM(vcpu)

COUNT\_DISTINCT(hostname)

WHERE

host\_group = shepherd

environment = dogfood

AND

GROUP BY

machine\_type

HAVING

No constraints

ORDER BY

SUM(vcpu) desc

Run Query

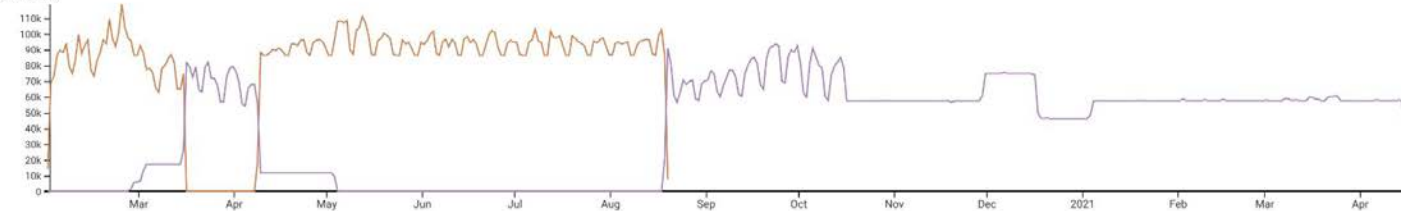
Run 3 days ago

Feb 1 2020, 11:05 AM – Apr 16 2021, 12:06 PM Granularity: 1 day

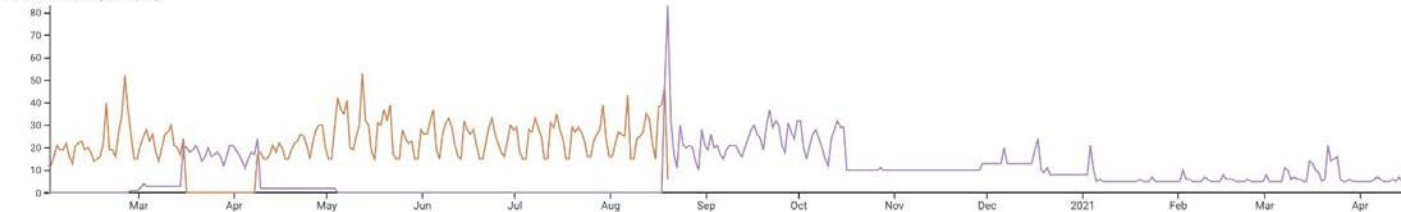
Results BubbleUp Traces Raw Data

Graph Settings

SUM(vcpu)



COUNT\_DISTINCT(hostname)



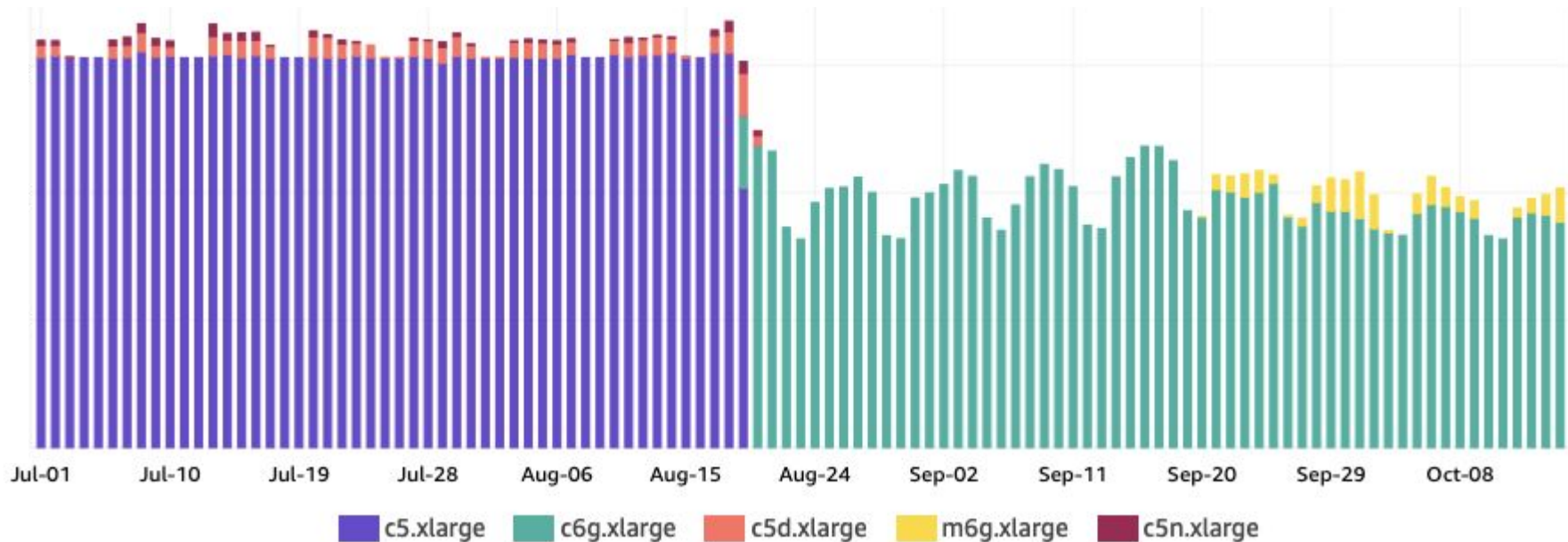
machine_type	SUM(vcpu)	COUNT_DISTINCT(hostname)
 x86_64	17,178,568	987
 x86_64	16,148,100	1,383

elapsed query time: 1.017064248s rows examined: 174,744,591 nodes reporting: 100%

Migration to Graviton2 instances in dogfood Shepherd, February 2020 to April 2021



# Dogfood Shepherd cost reduction



Dogfood Shepherd EC2 cost, grouped by instance type

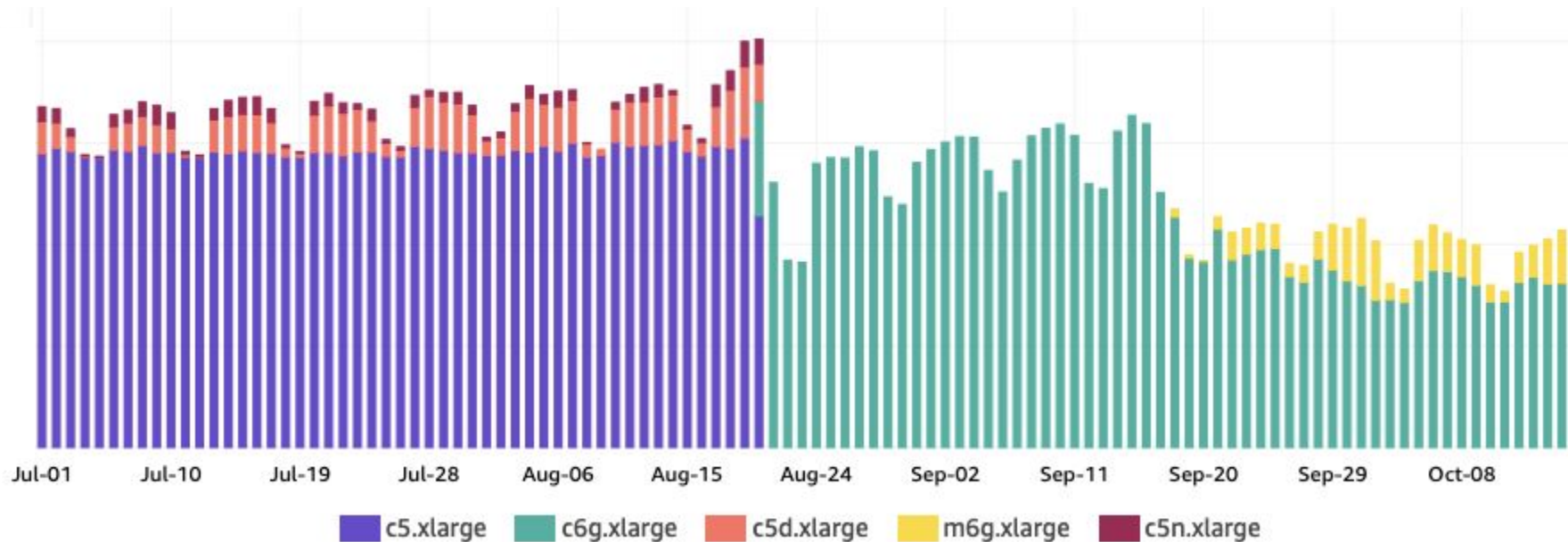




# **What happened next?**

---

# 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!

### 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.

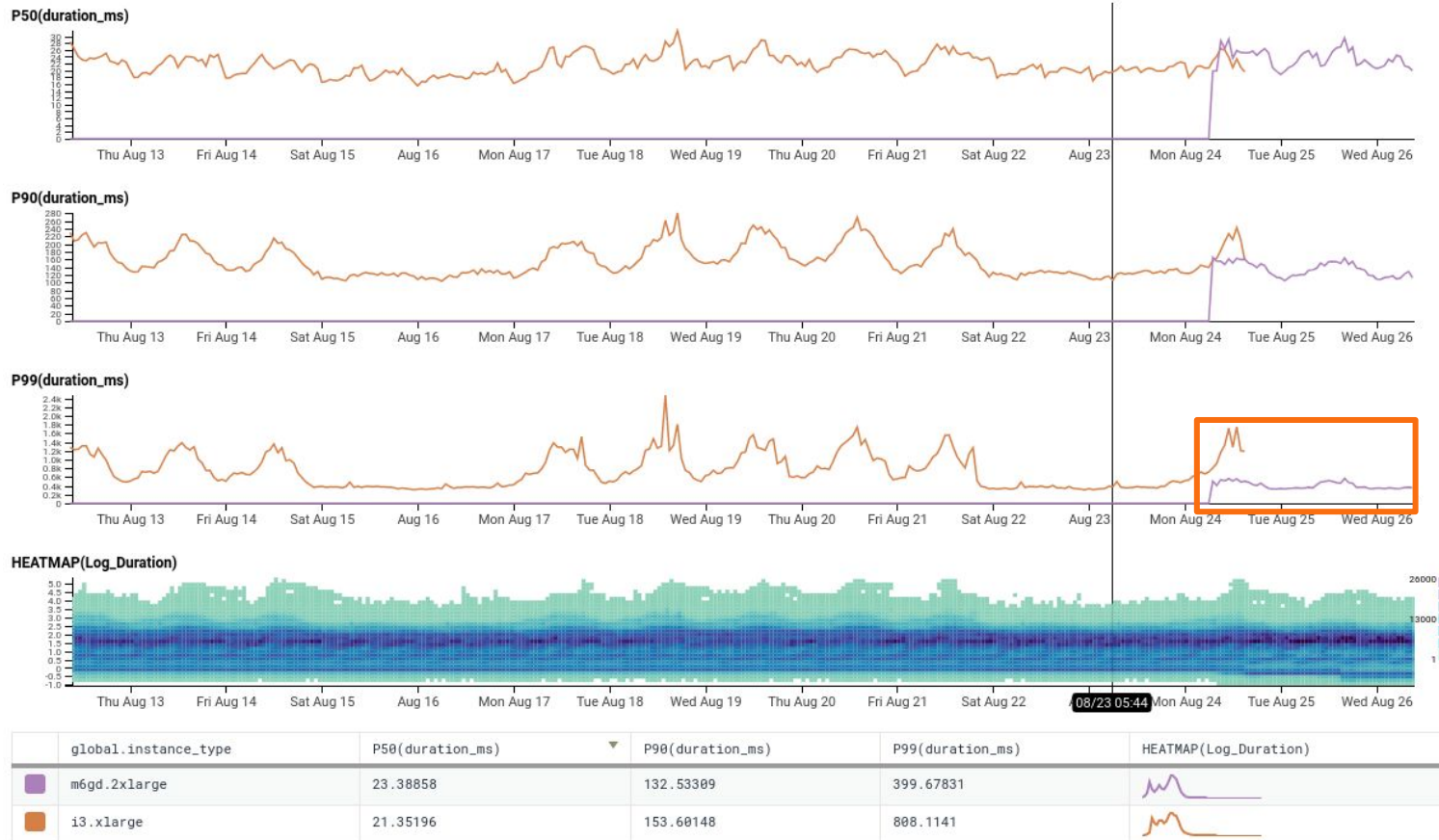
99.95% of eligible events from the  **basset** column

 `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.





Production Retriever migration



## Retriever traffic volume and Graviton2 migration

- 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

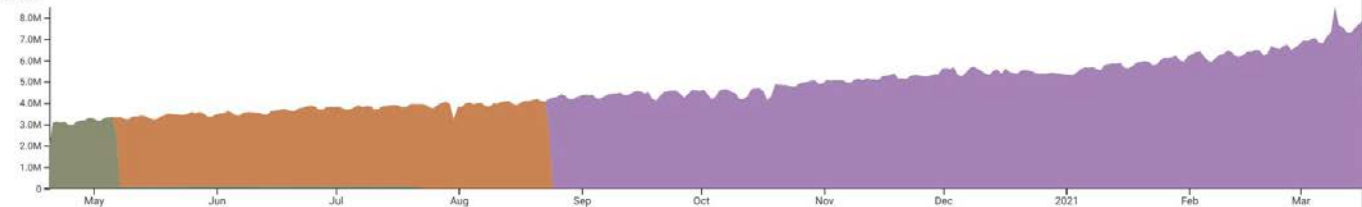
BubbleUp

Traces

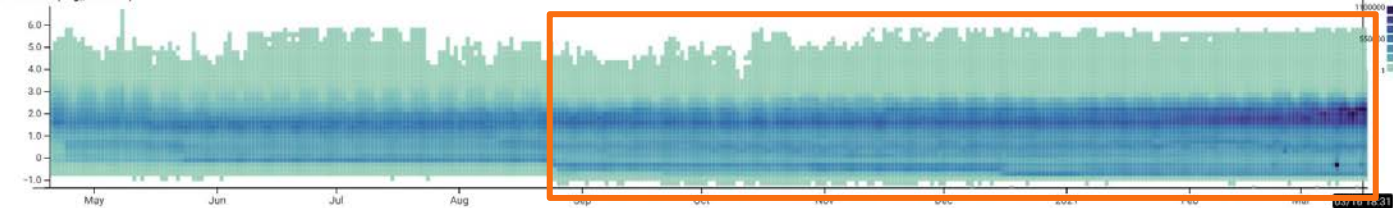
Raw Data

Graph Settings

COUNT



HEATMAP(Log\_Duration)



	global.instance_type	COUNT	HEATMAP(Log_Duration)
	m6gd.2xlarge	1,116,583,460	
	i3.xlarge	400,737,554	
	m5.xlarge	55,313,544	
	m6g.2xlarge	4,592,316	

elapsed query time: 2m38.837311559s rows examined: 459,833,822,980 nodes reporting: 100%



# AWS ran out of m6gd spot instances



**lizf** 6:52 AM

shit.

Launching a new EC2 instance. Status Reason: We currently do not have sufficient m6gd.2xlarge capacity in the Availability Zone you requested (us-east-1b). Our system will be working on provisioning additional capacity. You can currently get m6gd.2xlarge capacity by not specifying an Availability Zone in your request or choosing us-east-1a, us-east-1c, us-east-1d, us-east-1e, us-east-1f. Launching EC2 instance failed.

stopping ASG updater script

(that statement is a lie, there's nothing in us-east-1d, only 1a or 1b



**lizf** 11:26 AM

AWS telling me it's safe to resume provisioning m6gd, they apparently do have the ability to remedy a stock-out in hours not days or weeks



1



# Kafka

---

Longtime Confluent Kafka users

First to use Kafka on Graviton2 at scale

**Changed multiple variables at once**

- move to tiered storage
- i3en → c6gn
- AWS Nitro



Read more: [go.hny.co/kafka-lessons](https://go.hny.co/kafka-lessons)



# Kafka + the long tail

15	- kafka_instance_type	= "c6gn.2xlarge"
15	+ kafka_instance_type	= "i3en.2xlarge"

## Services fully on Graviton2:

- shepherd
- retriever
- poodle
- refinery

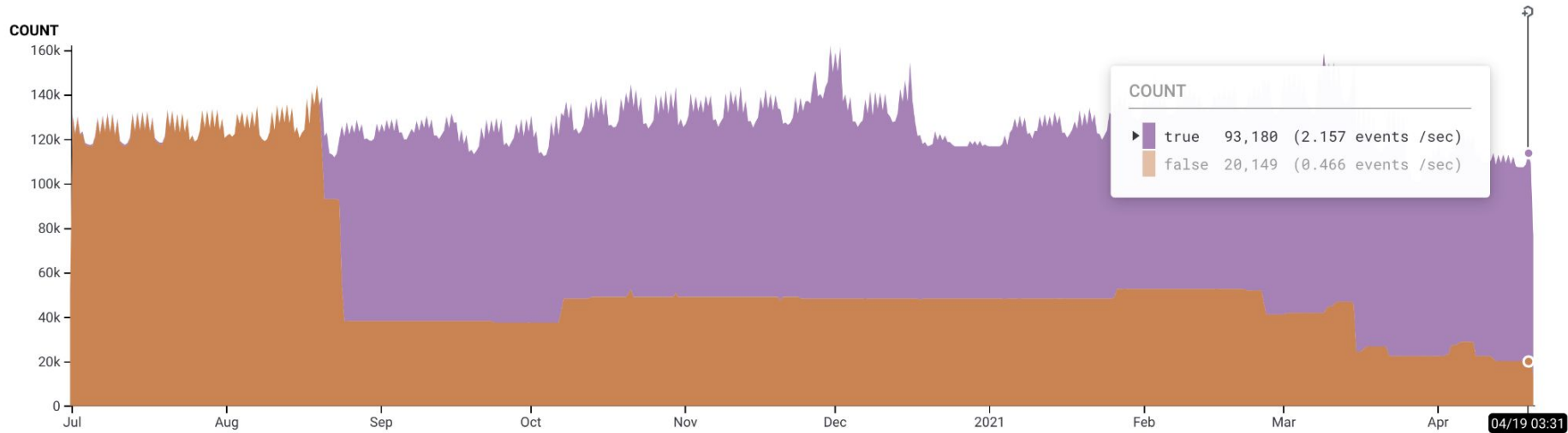
```

poodle_instance_type      = "c6g.large"
doodle_instance_type      = "t2.small"
labrador_instance_type    = "m4.large"
zk_instance_type          = "m5.large"
shepherd_instance_types   = ["c6g.4xlarge", "m6g.4xlarge"]
wfproxy_instance_type     = "c5.large"
samproxy_instance_type    = "m6g.xlarge"
kafka_instance_type       = "i3en.2xlarge"
retriever_instance_type   = "m6gd.2xlarge"
loadtest_instance_type    = "m4.large"
util_instance_type        = "m4.large"
dalmatian_instance_types  = ["r5.large", "r5d.large", "r5n.large", "r5dn.large"]
dalmatian_catchup_instance_count = 0
mysql_instance_type       = "db.m5.2xlarge"
util_mysql_instance_type  = "db.m4.large"
slos_mysql_instance_type  = "db.m5.large"
ratelimits_redis_instance_type = "cache.m6g.xlarge"
banners_redis_instance_type = "cache.t2.small"
dalmatian_redis_instance_type = "cache.m5.large"
slos_redis_instance_type  = "cache.t2.small"
query_cache_redis_instance_type = "cache.t2.small"
samproxy_redis_instance_type = "cache.t2.small"

```



# Graviton2 going strong



Amount of traffic running on Graviton2 instances



# **Takeaways**



# Have a measurable goal in mind

---

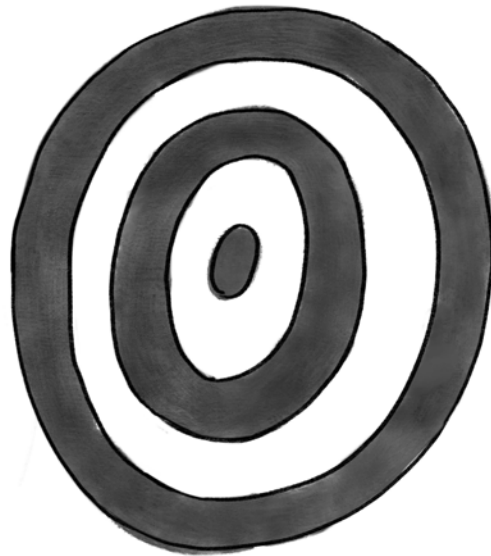
Need to be able to compare to baseline!

## Ask yourself:

- What are you currently measuring?
- Do your existing dashboards reflect customer impact?

## Most importantly:

- Start by measuring *something*
- Then learn and iterate



# Acknowledge hidden risks

## Examples of hidden risks

- Operational complexity
- Existing tech debt
- Cost of learning new tech and practices
- Vendor code and architecture
- Upstream dependencies



# Take care of your people

---

## Existing incident response practices

- Escalate when you need a break / hand-off
- Remind (or enforce) time off work to make up for off-hours incident response

## Newly official Honeycomb policy

- Incident responders are encouraged to expense meals for themselves and family during an incident



### **We hire adults.**

Pay attention to your mind and body so you can give and get help. All of us wobble, and being transparent about that means we can support each other. Participate fully in collaboration, coaching and management. If any group of us were together in a car on a long road trip, there would be no need for a dividing line in the back seat to keep people from hitting each other.



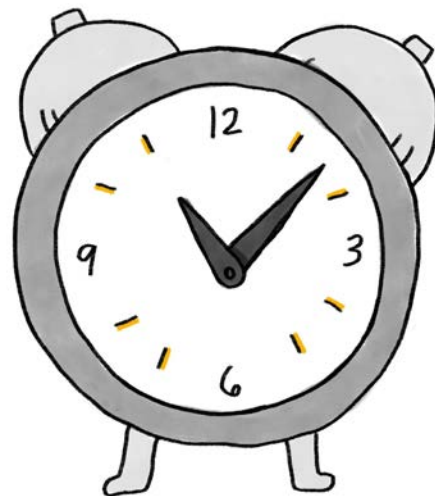
# Optimize for safety

Ensure people don't feel rushed.

## Complexity multiplies

- if a software program change takes  $t$  hours,
- software *system* change takes  $3t$  hours
- software *product* change also takes  $3t$  hours
- software *system product* change =  $9t$  hours

Maintain tight feedback loops, but not everything has an immediate impact.

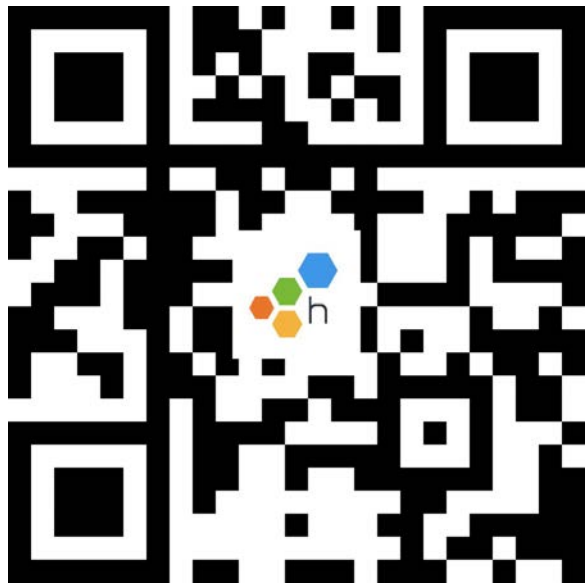


Source: *Code Complete, 2nd Ed.*



# Graviton2 blog posts

---



March 2020: [go.hny.co/arm64](https://go.hny.co/arm64)

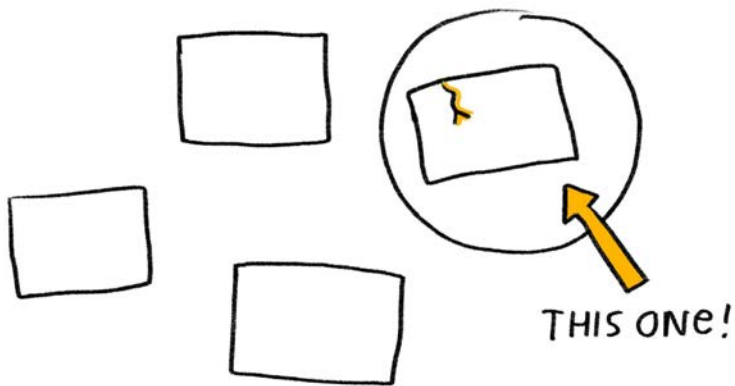


April 2021: [go.hny.co/graviton2-retro](https://go.hny.co/graviton2-retro)



# Learn about observability

Make production a more welcoming place.



Read more: [go.hny.co/o11y101](https://go.hny.co/o11y101)





# Reach out!

---

[honeycomb.io/shelby](https://honeycomb.io/shelby)

@shelbyspees



[www.honeycomb.io](http://www.honeycomb.io)