Optimizing cost and performance with arm64

Liz Fong-Jones
Principal Developer Advocate, honeycomb.io
@lizthegrey

honeycomb.io

WTF is architecture? Why multiarch?

Instruction set architecture

Reference implementations & microarchitectures

Software support for all the above

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

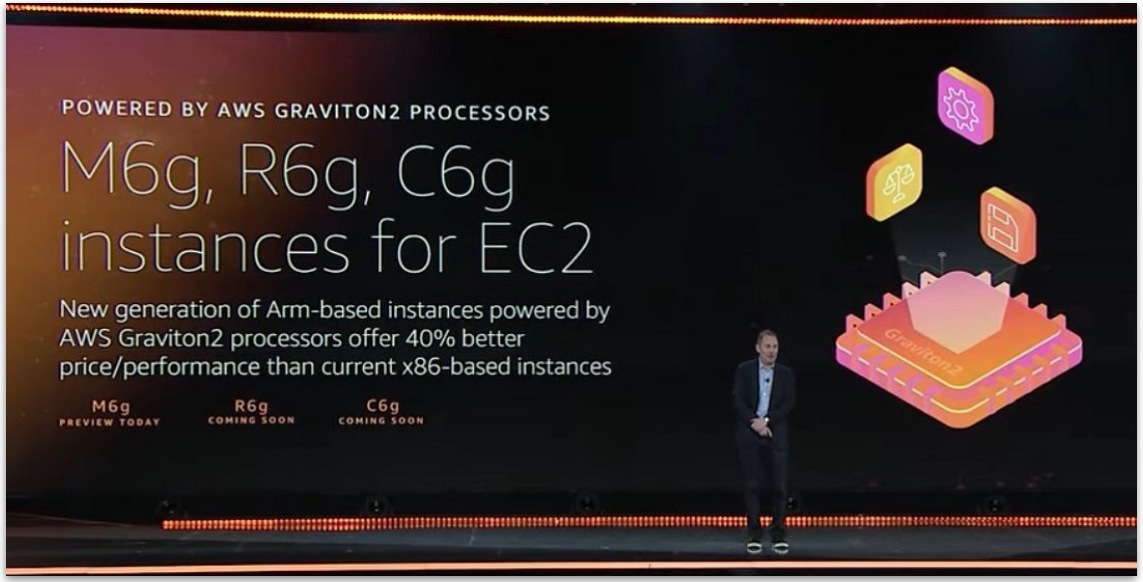
Mobile was an edge case… or was it?

Open source and scale-out workloads changed the game

Graviton2 announced Dec 2019

Promised improvements

- cost
- performance
- environmental impact



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

ARM is more efficient.

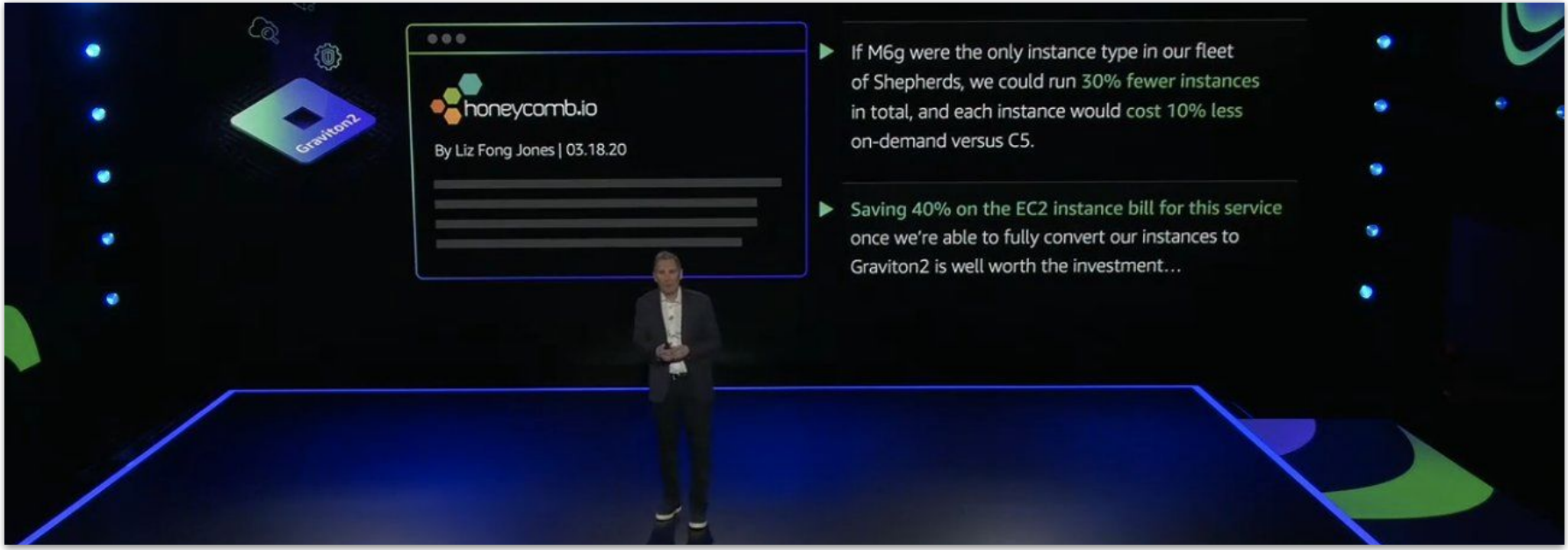
Why it's cheaper/power-efficient

- 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

One year later



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

Liz Fong-Jones

Principal Developer Advocate at Honeycomb.io

@lizthegrey

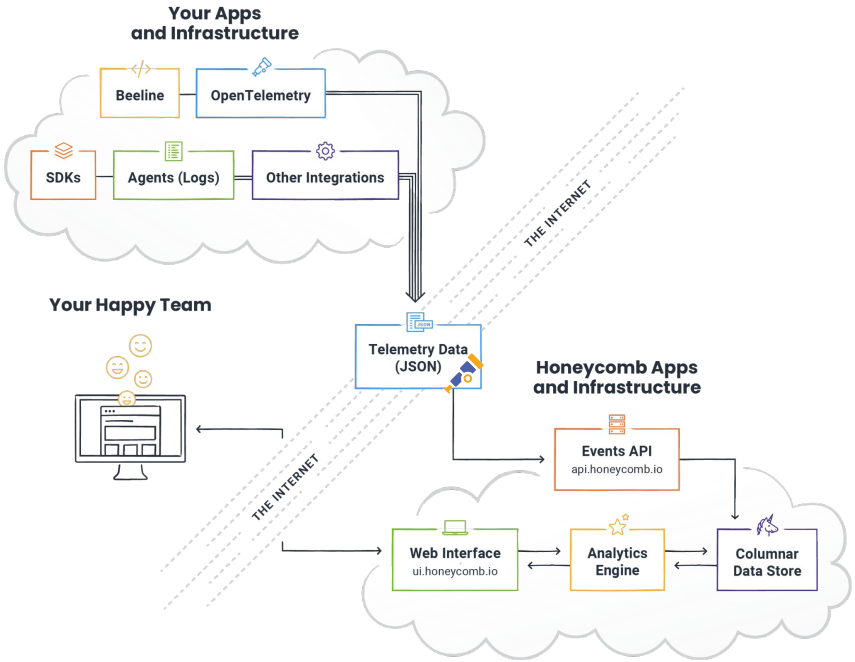
Was it worth the RISC?

What’s important to Honeycomb?

Data storage engine and analytics tool

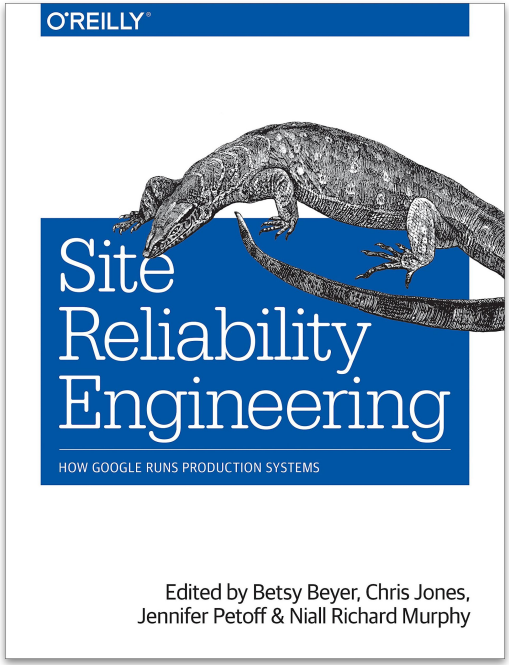
What Honeycomb does

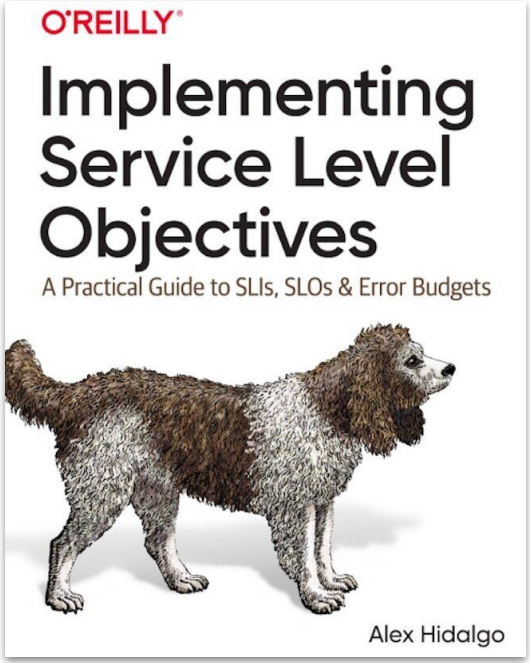
- Ingests customer's telemetry
- Indexes on every column
- Enables near-real-time querying on newly ingested data



Service Level Objectives (SLOs)

Common language between engineers and business stakeholders

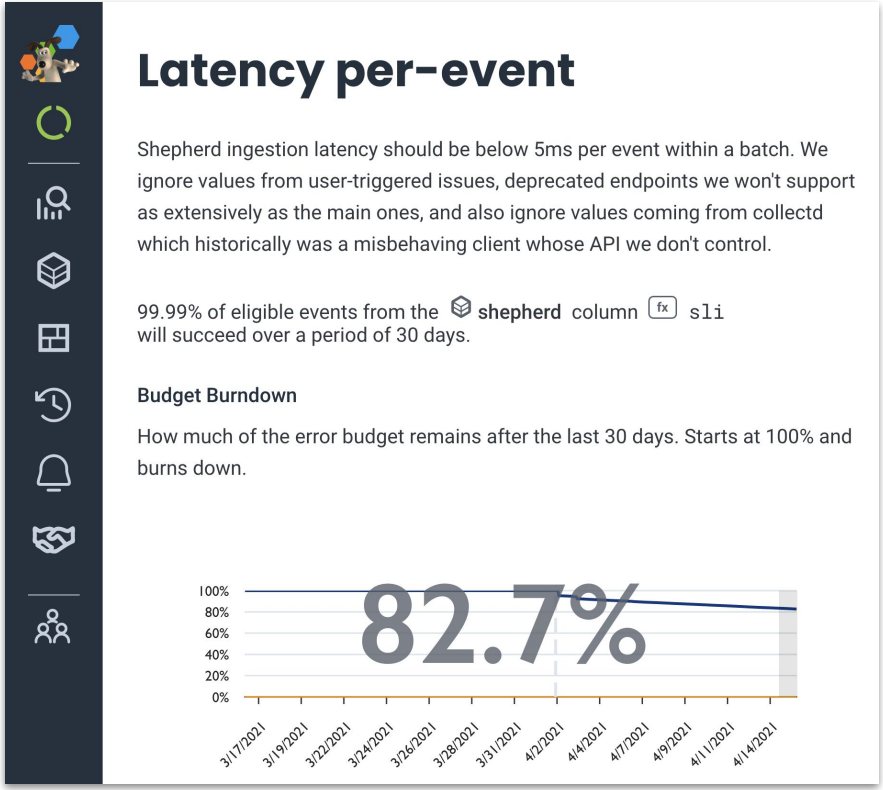




SLOs are user flows

Honeycomb's SLOs

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

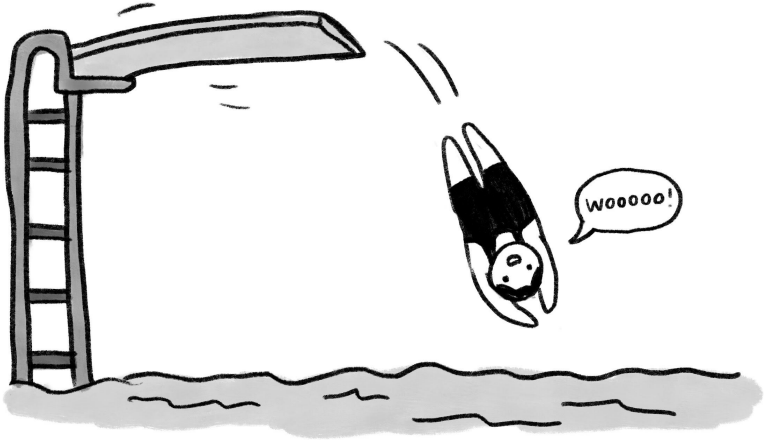


Same reliability, lower costs with ARM64

Infra is our #2 expense after feeding our honeybees

Infra cost scales with traffic

"Cost of Goods Sold" and other business acronyms



Complexity stayed manageable

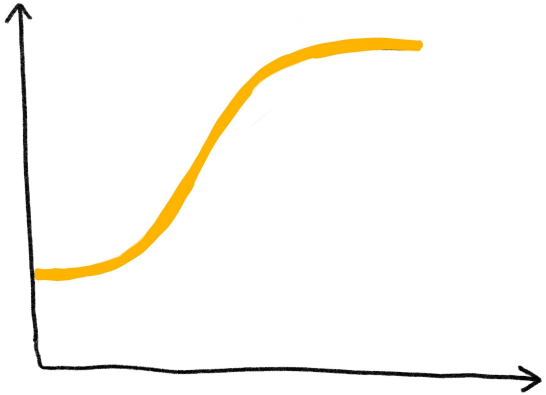
Bootstrap/automation code shared between amd64 and arm64

No special snowflake behavior

Convergence to one architecture over time

Choosing where to start

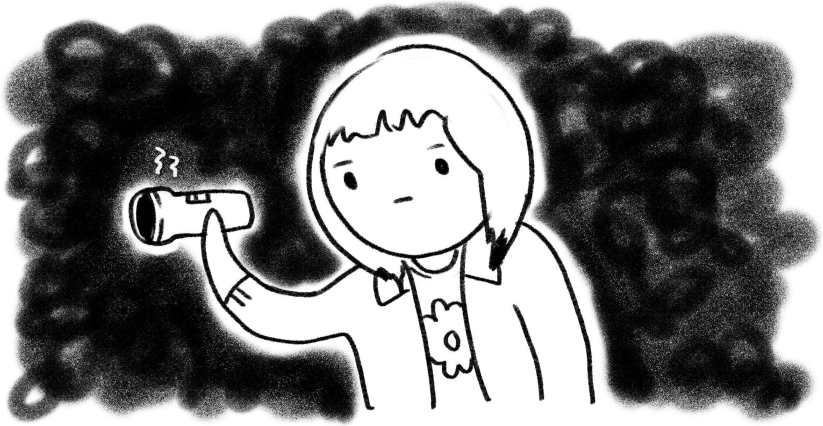
Prod: customers observe data



Dogfood observes prod

Production telemetry → Dogfood ingest

Same code as production



ui-dogfood.honeycomb.io/prod/datasets/poodle/result/4mcC7bzEXTn?hideMarkers

SLO Usage Tracking Query changed. Update Board

SLO Adoption: Page Views

Add to Board Share Last 1 month 4 days 7 hours 57 minutes 42 seconds

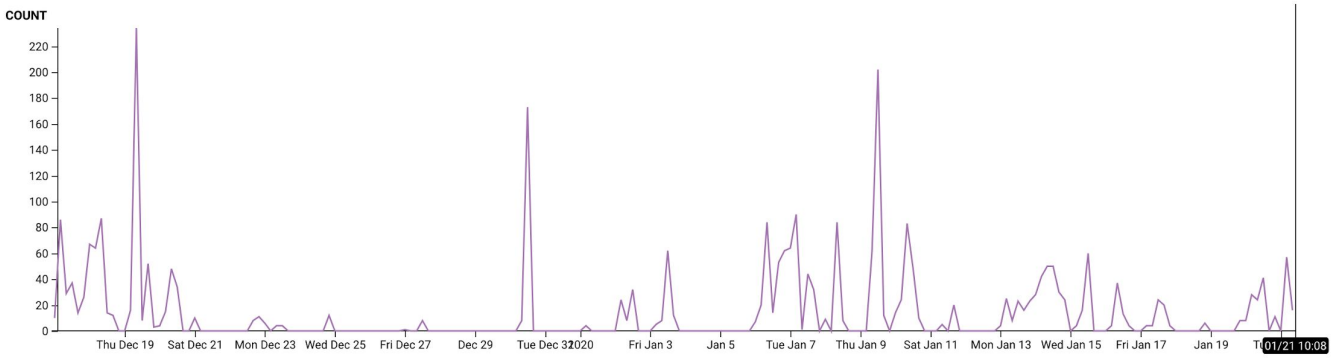
Datasets / poodle

Add a description for this query

VISUALIZE	WHERE AND	GROUP BY	ORDER BY	LIMIT	
COUNT	request.url exists request.url contains /slo/	None; combine all rows	COUNT desc	None	Run Query Run 2 minutes ago

Dec 17 2019, 2:10 AM — Jan 21 2020, 10:08 AM Granularity: 4 hr

Results BubbleUp Traces Raw Data Graph Settings



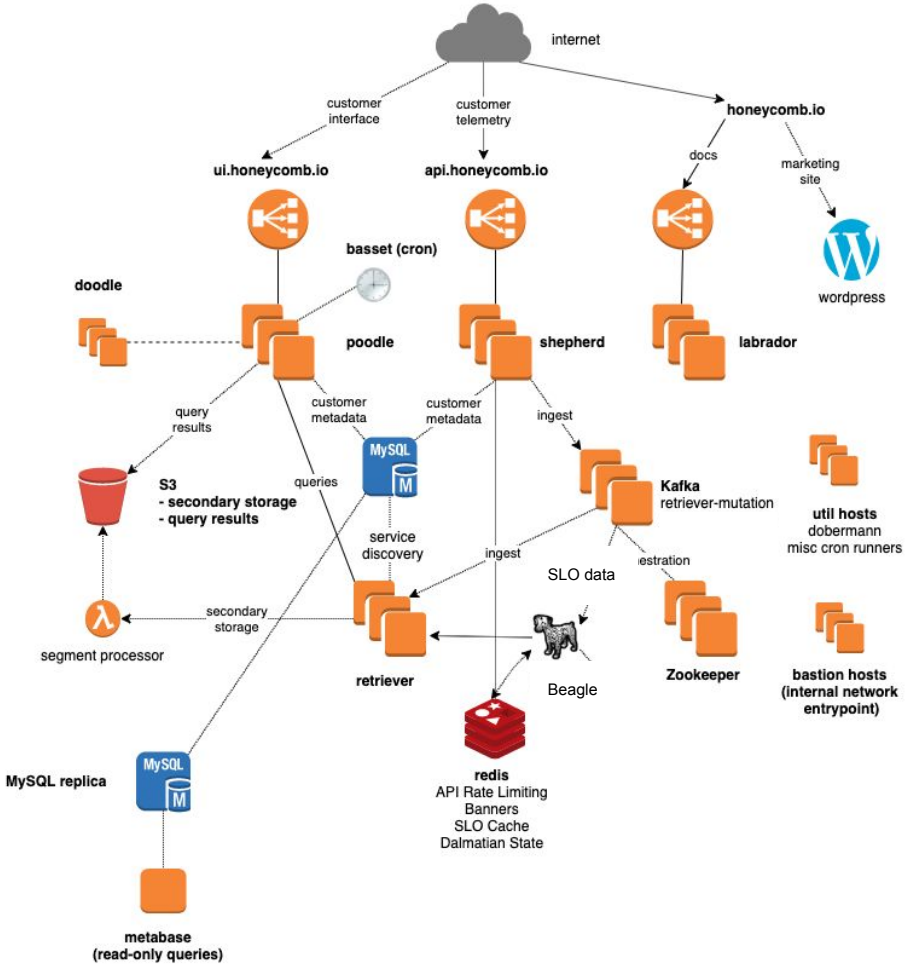
COUNT
3,029

Kibble observes dogfood

Service Architecture

Honeycomb's services

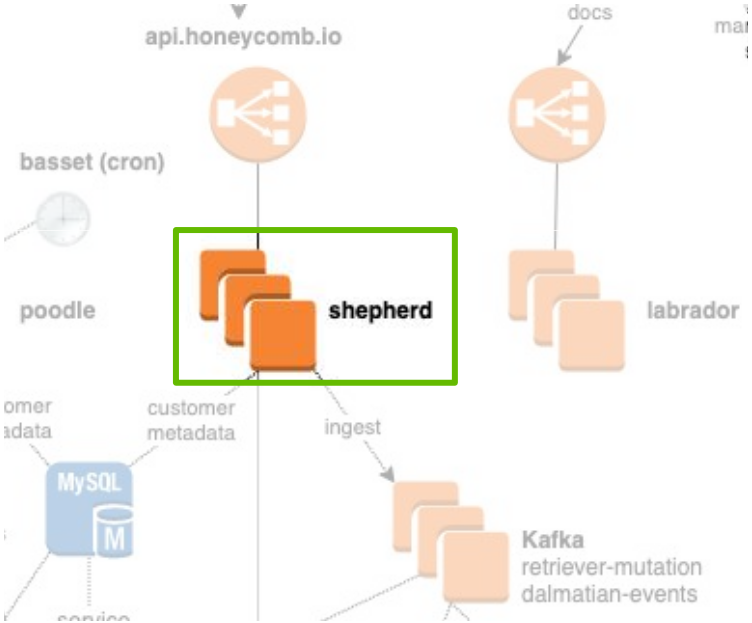
- shepherd (ingest API)
- kafka (ingest event streaming)
- retriever (indexing and querying)
- beagle (streaming SLO evaluation)
- poodle (frontend web app)
- refinery (sampling)
- doodle (images)
- labrador (docs, bins, nginx redirects)
- basset (alerting, lives on poodle)
- basenji (encryption)



Shepherd: ingest API service

Why Shepherd?

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

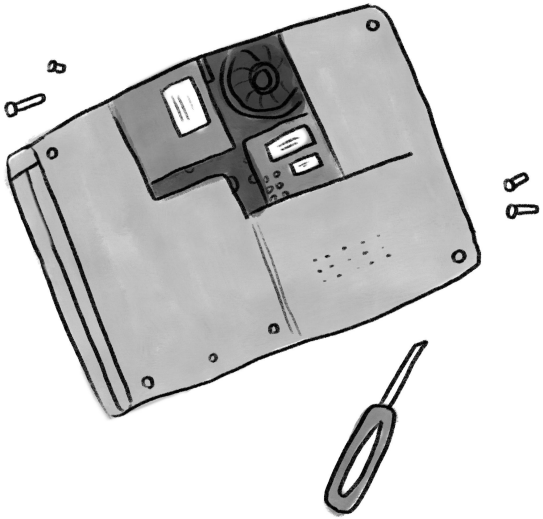


Preparing to test out the change

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

```
+        - go_build:
+            name: go_build_arm64
+            goarch: arm64
+            requires:
+              - 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.

[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

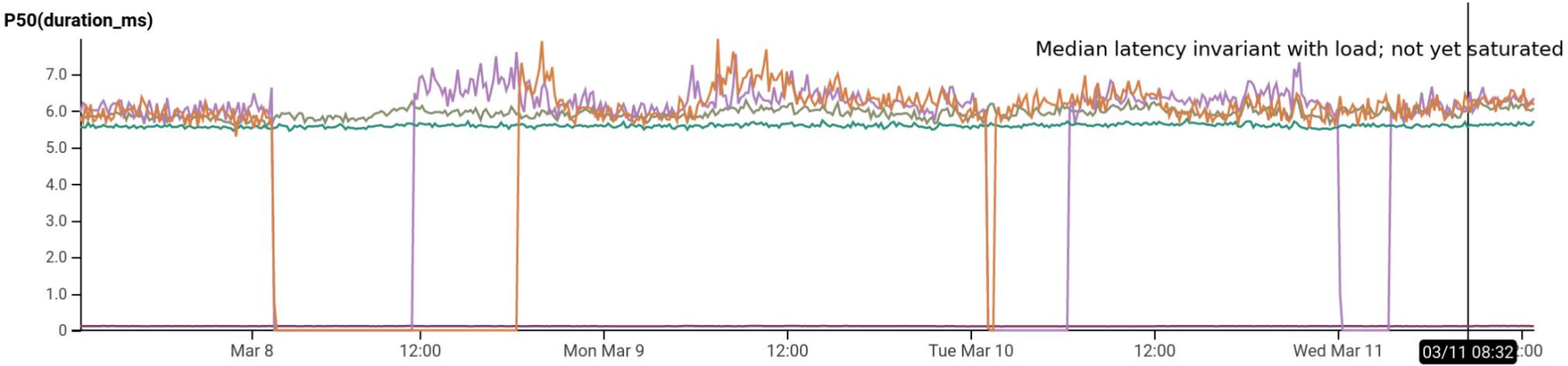
- started with one instance
- bumped to 20% to observe

```
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 = {
   -     dogfood    = 1
50 +     dogfood    = 3
51       production = 0
52       kibble     = 0
```

	global.instance_type	HEATMAP(log_duration_ms)
	c5.xlarge	
	m6g.xlarge	Shifted left, narrower peak, less tail
	c5d.xlarge	
	c5n.xlarge	

Distribution of request latency on different instance types



	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

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

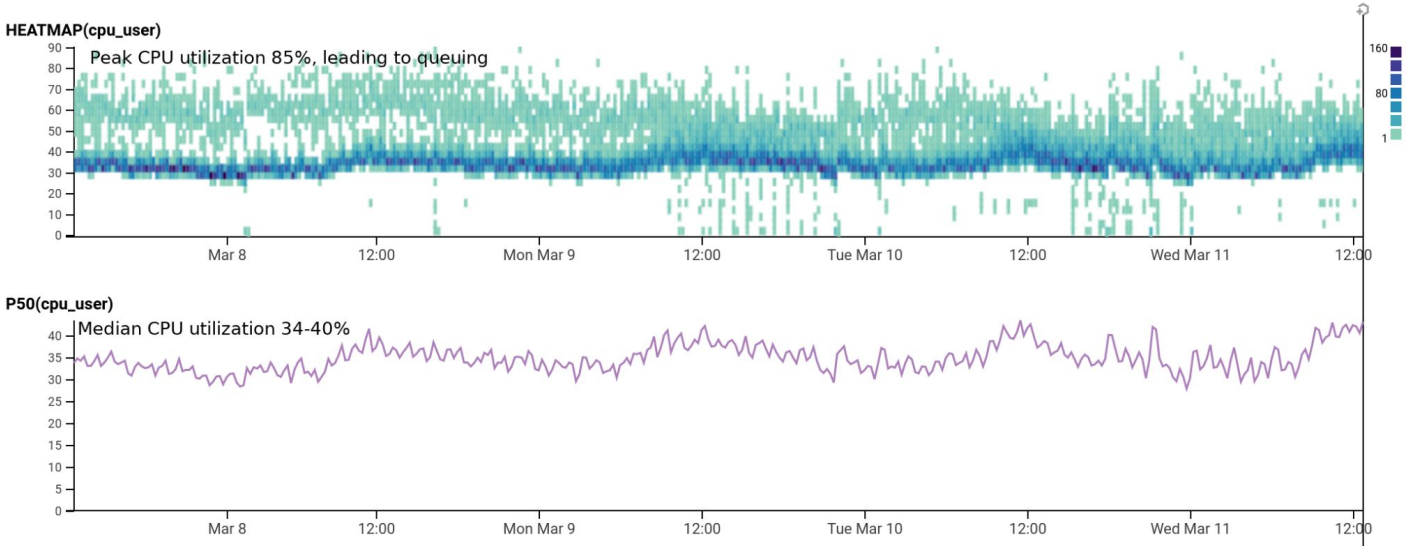
Run Query

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 old instances

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

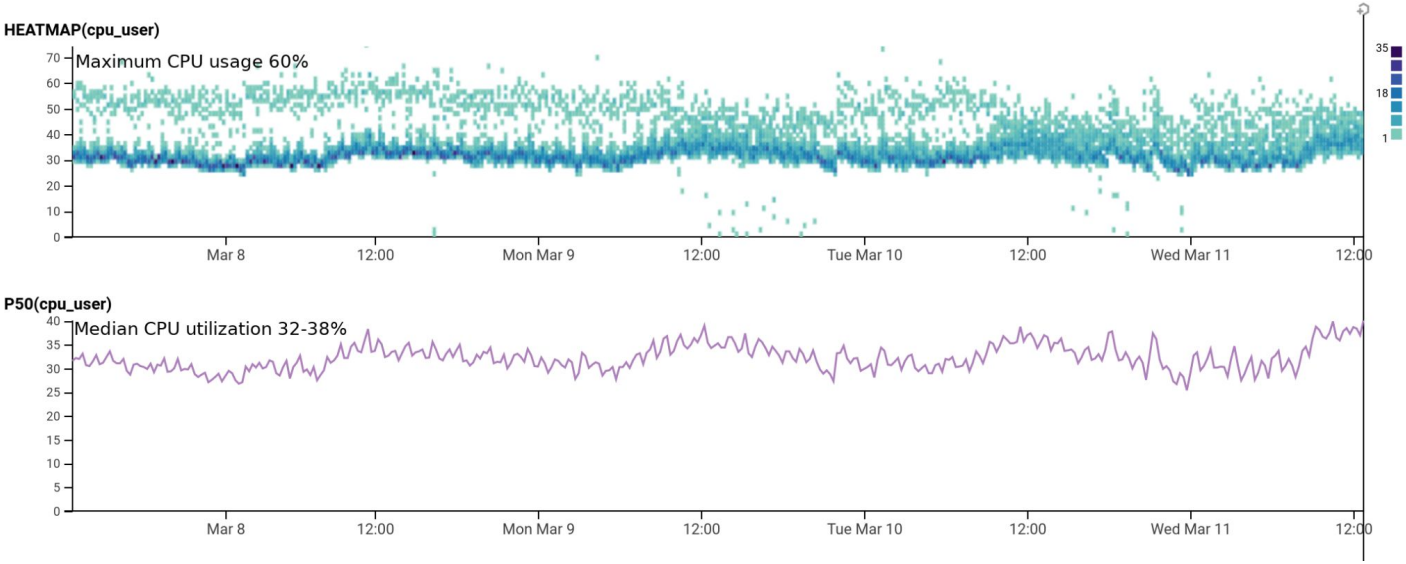
Run Query

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 new architecture

Dataset / system_stats

1 board | Add to Board | Share | Last 1 year 2 months 13 days 14 seconds

vCPU and number of hosts, x86 vs. Arm64

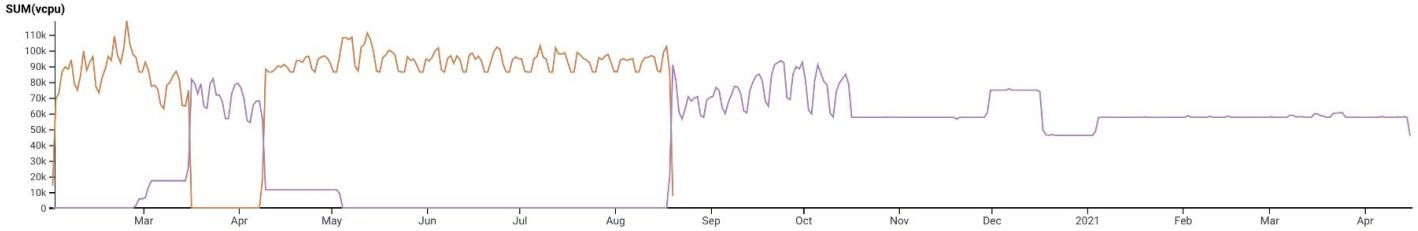
VISUALIZE	WHERE (AND)	GROUP BY	HAVING	ORDER BY
SUM(vcpu) COUNT_DISTINCT(hostname)	host_group = shepherd environment = dogfood	machine_type	No constraints	SUM(vcpu) desc

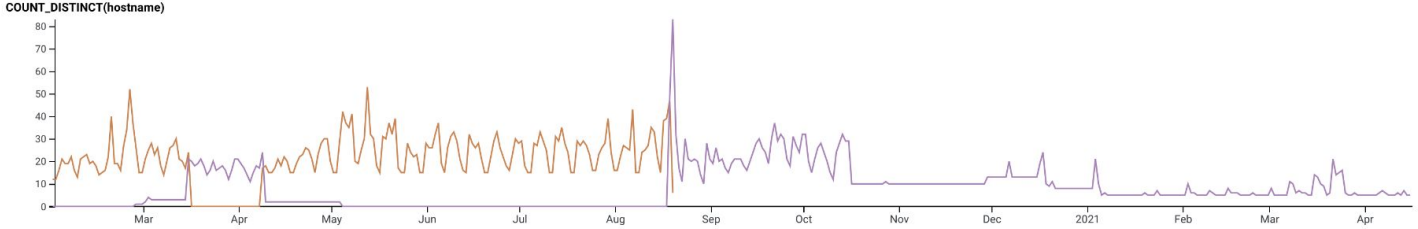
Run Query

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



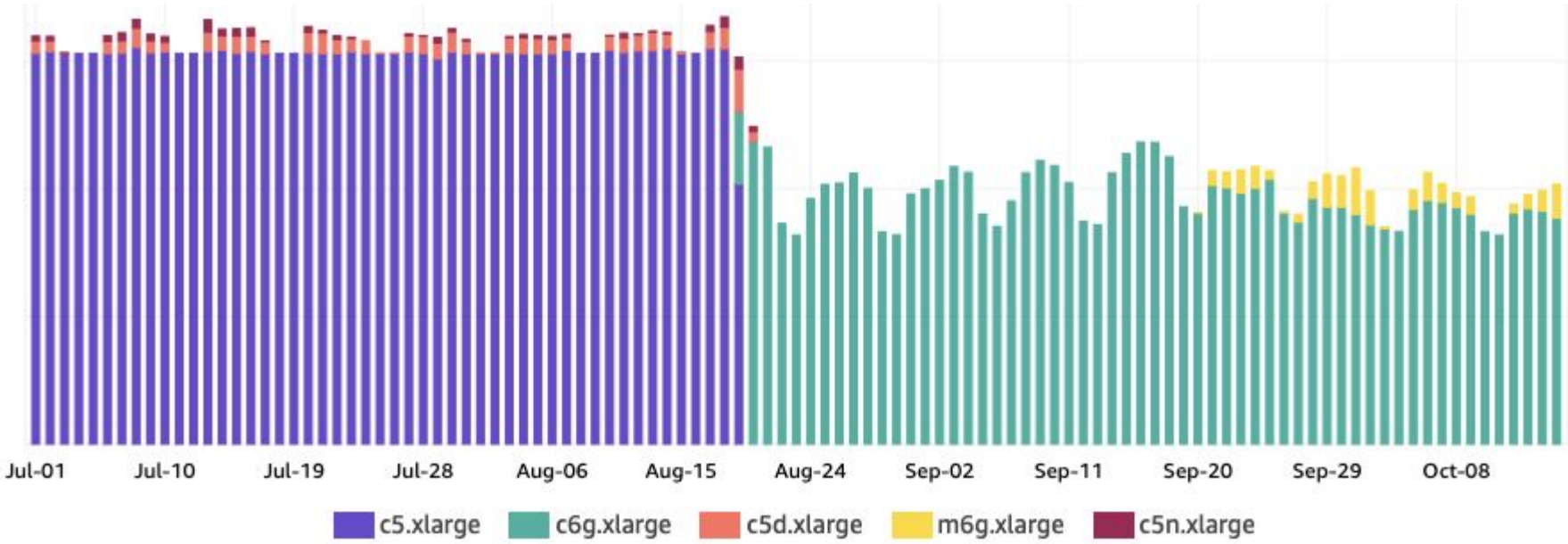


	machine_type	SUM(vcpu)	COUNT_DISTINCT(hostname)
		17,178,568	987
	x86_64	16,148,100	1,383

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

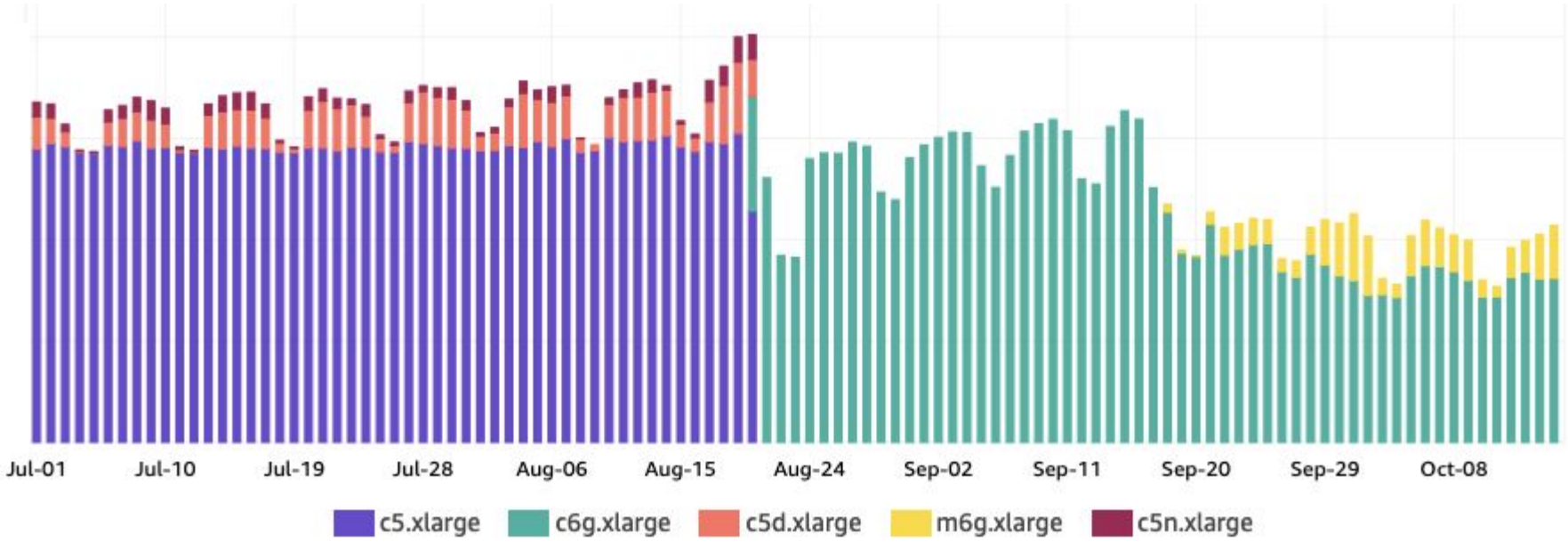
Dogfood Shepherd cost reduction



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!

Headroom was critical for demand growth

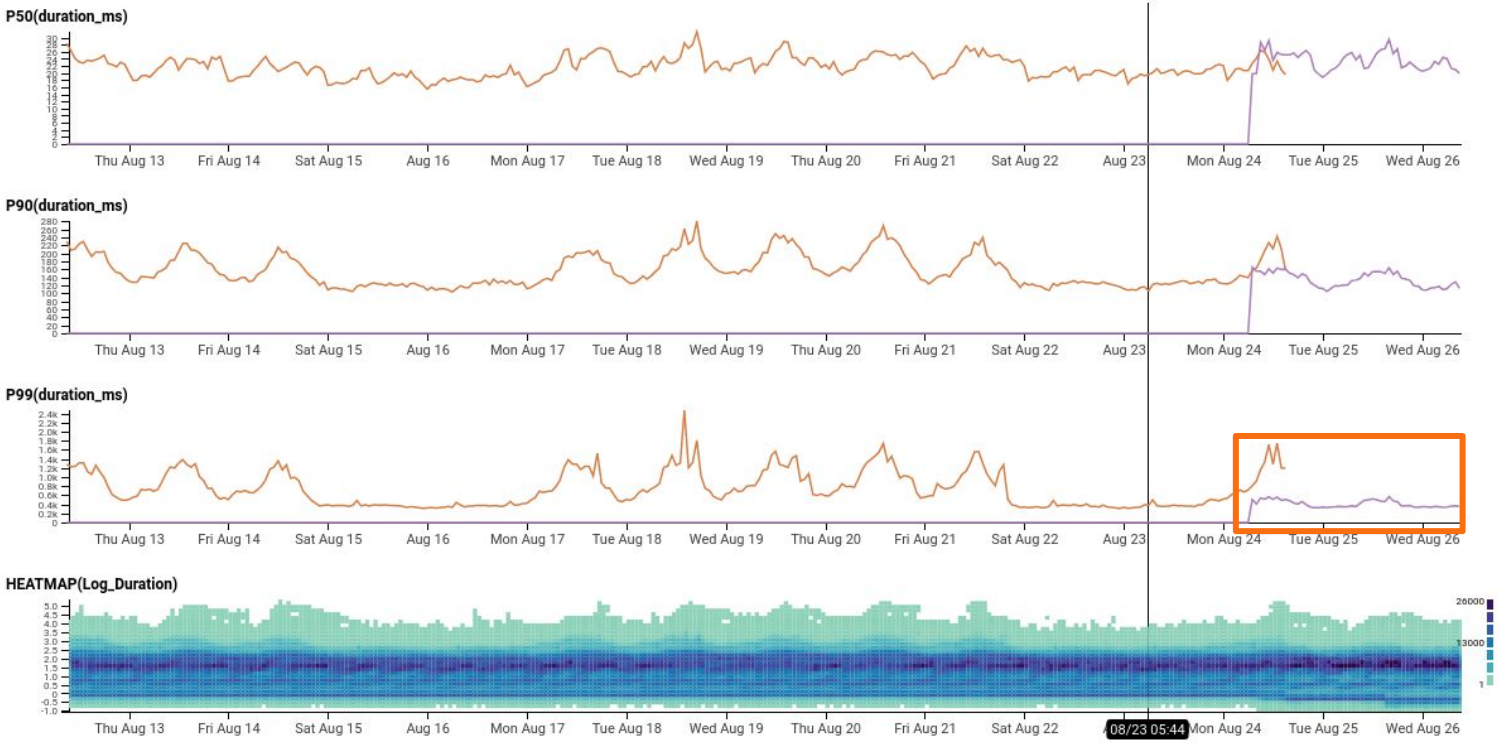
Triggers Successful Timely Runs

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	808.1141	

Production Retriever migration

Retriever traffic volume and Graviton2 migration

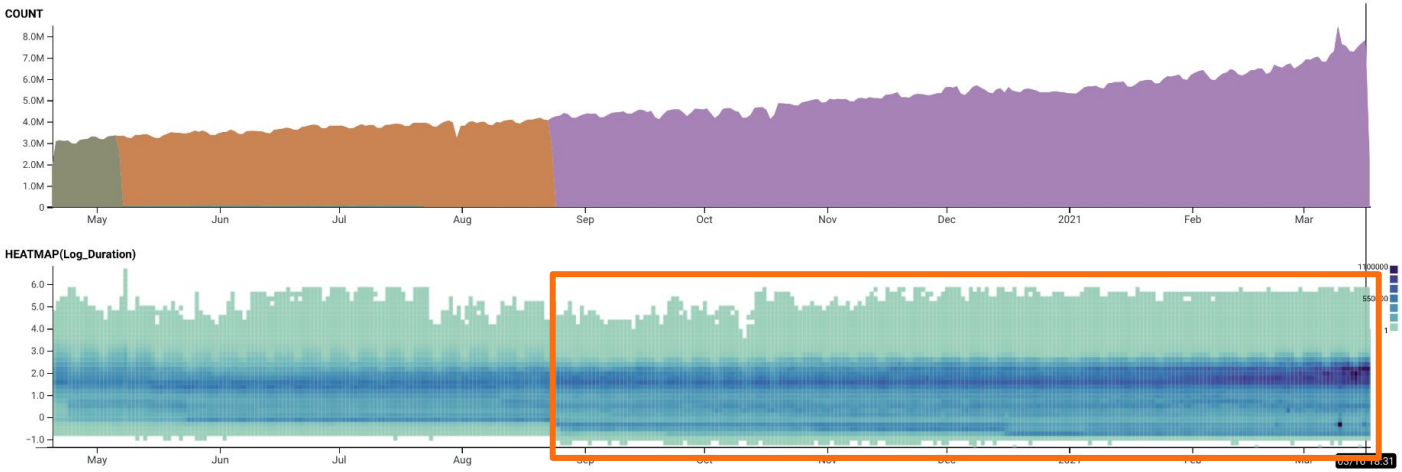
VISUALIZE	WHERE AND	GROUP BY	HAVING	ORDER BY
COUNT HEATMAP(Log_Duration)	name = ReadRows service_name = retriever	global.instance_type	No constraints	COUNT desc

Run Query

Run 7 minutes ago

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

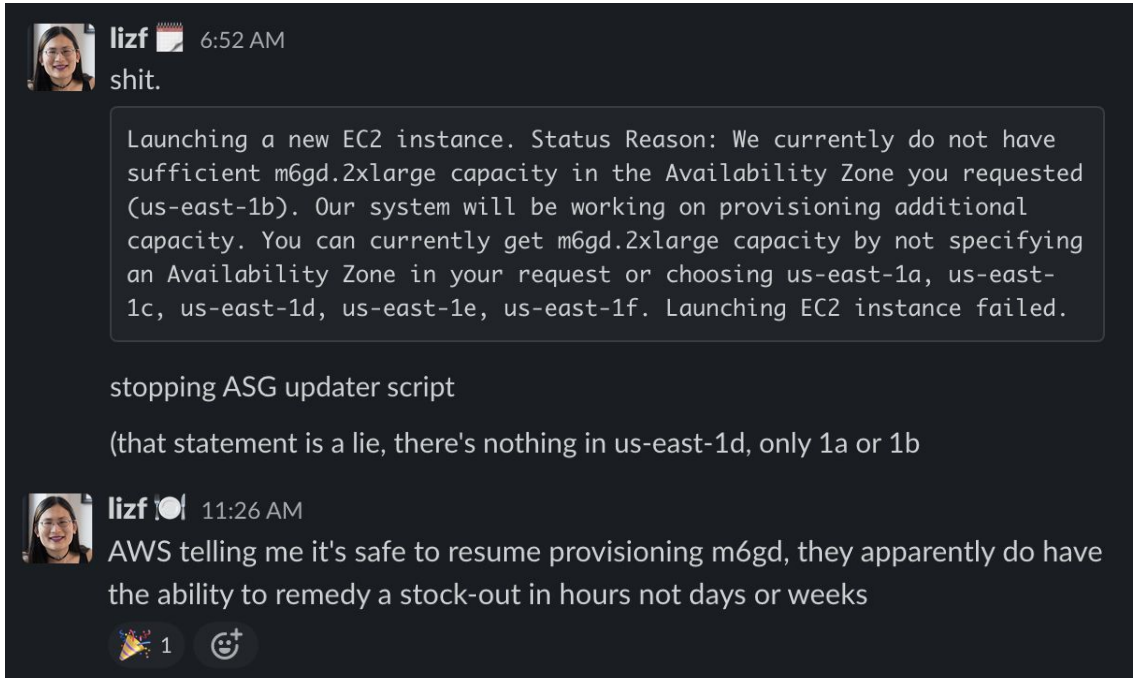
Results | BubbleUp | Traces | Raw Data | Graph Settings



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

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

AWS ran out of m6gd spot instances



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

Kafka + the long tail

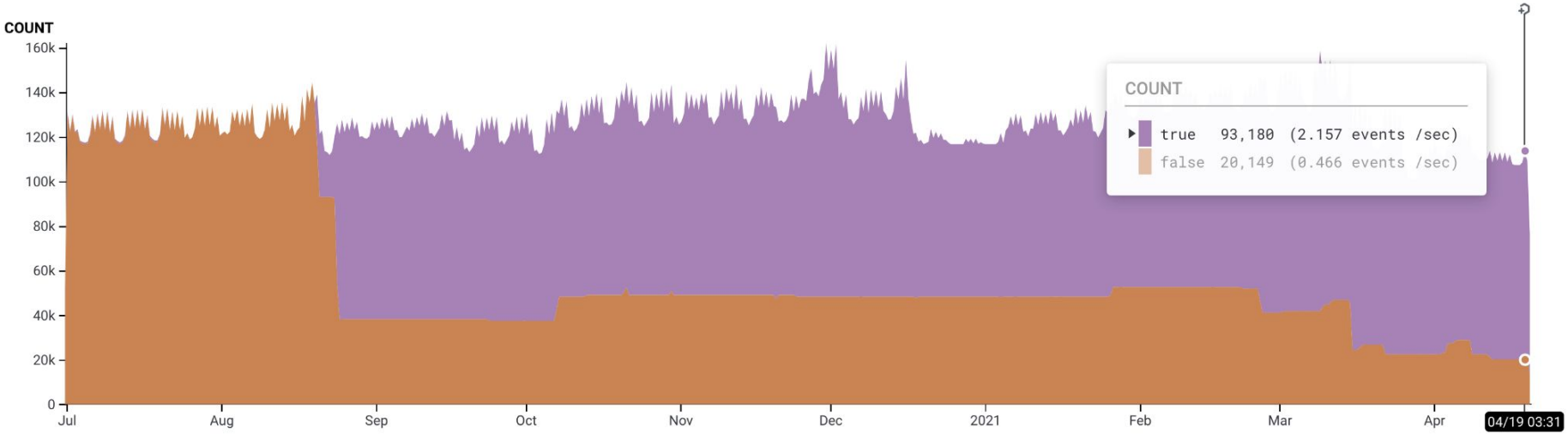
```
15  - kafka_instance_type          = "c6gn.2xlarge"
    15 + kafka_instance_type       = "i3en.2xlarge"
```

Services fully on Graviton2:

- shepherd
- retriever
- poodle
- beagle
- 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

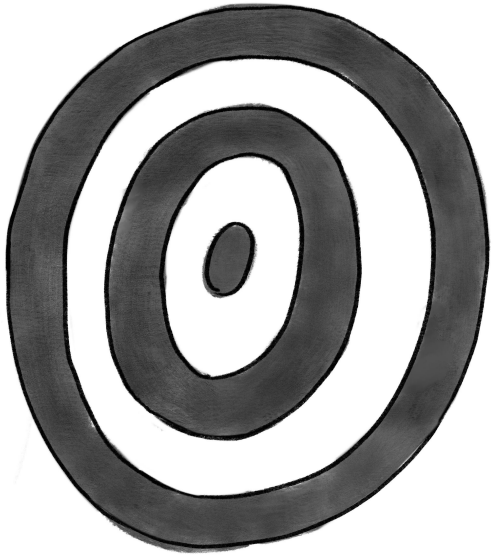
Need to be able to compare to baseline!

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.

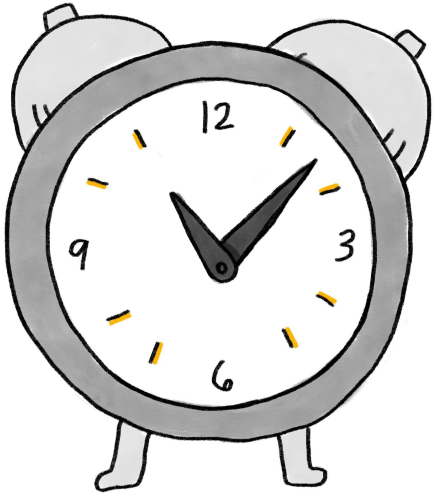
Optimize for safety

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](go.hny.co/arm64)

April 2021: go.hny.co/graviton2-retro

Ways to try out arm64

aws

EQUINIX

METAL

ORACLE Cloud

SolidRun
Embedded Edge Computing

PINE64

Reach out for help!

honeycomb.io/liz

@lizthegrey

www.honeycomb.io