Fast, Reproducible Builds

With Docker Bake

Liz Fong-Jones

Field CTO, honeycomb.io

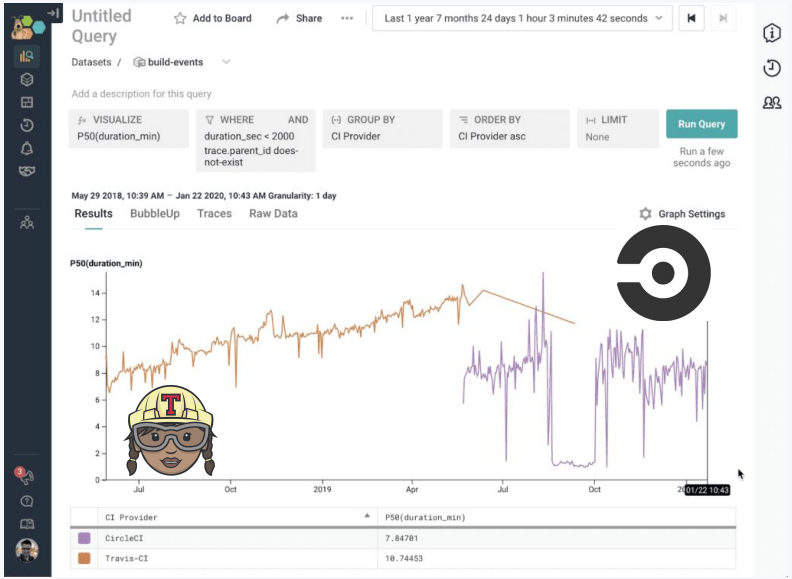
The build time is too damn high!

- Crashes that repro only in prod 😡😡😡
- Build, test for crash, build, test for crash…

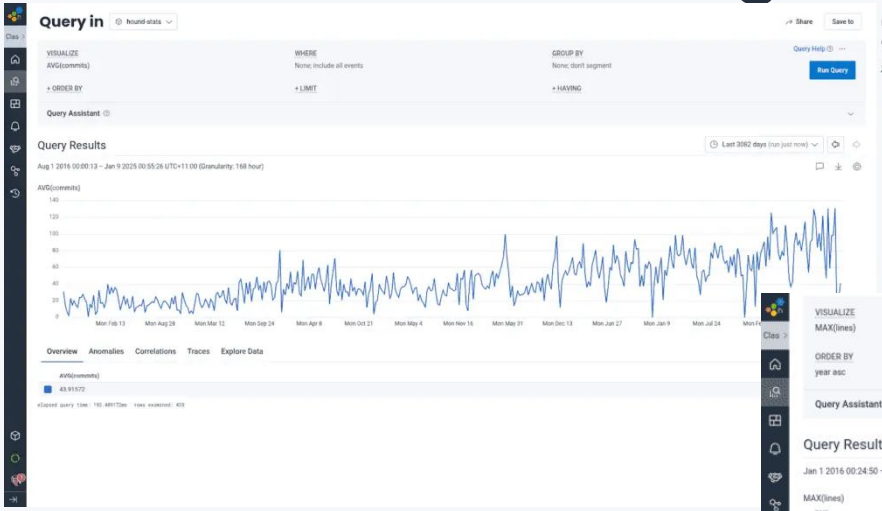
- But build time was 12-15 minutes. 🙀🙀🙀

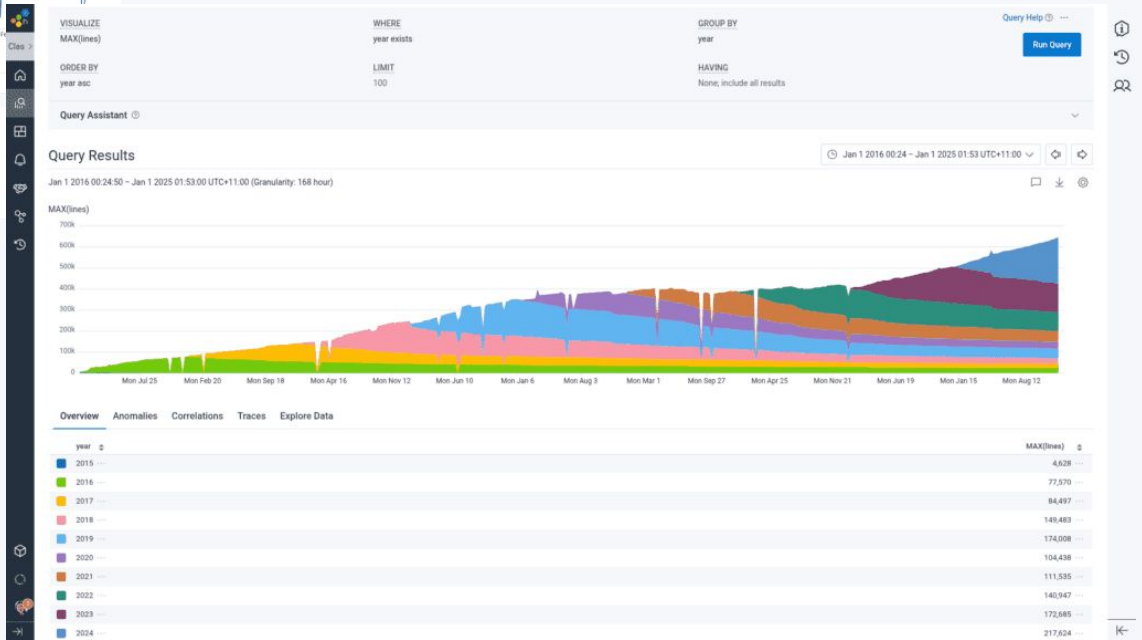
Has it always been this bad?

Hm. It used to be 8 minutes in 2020. But it's consistently 12 now.



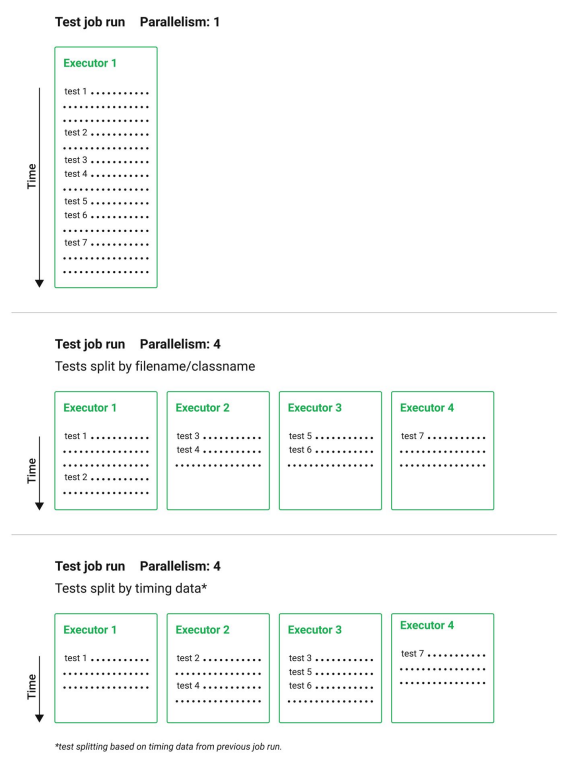
And how did we get to here?





We thought we had optimised…

After all, CircleCI got us to 8 minutes with parallelism & test splitting.

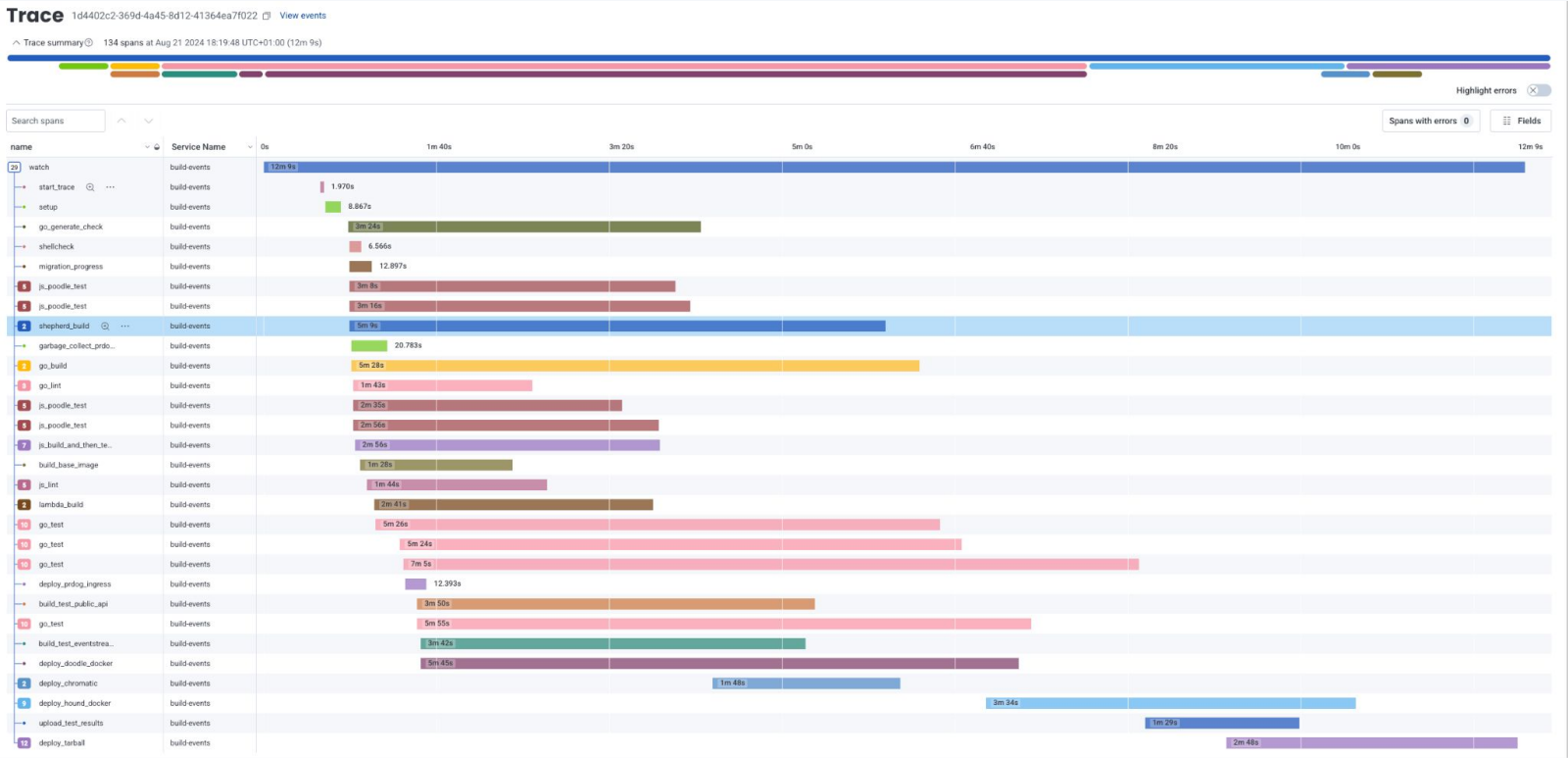


main hound / .circleci / config.yml

Code Blame 1536 lines (1455 loc) · 66.7 KB

```
46  #
47  # == Optimizations ==
48  #
49  # You are probably reading this section because "the build is slow" and you're
50  # trying to fix it. Sorry about that. Here are some pointers and things that
51  # have been tried in the past to hopefully save you some trouble.
52  #
53  # Running jobs in parallel requires two bits of overhead that need to be
54  # accounted for: the job spin up time and transferring files between jobs. Let's
55  # take these one at a time.
56  #
57  # Job spin up: Every job that Circle runs is a new container. We use `cimg`
58  # containers, which are Circle's recommendation to hopefully have the image
59  # cached on the host by virtue of a lot of people all using the image.
60  # Unfortunately, it's just luck whether the host has a cached image or not, and
61  # the more jobs we run the more we run the risk of hitting a host without the
62  # correct image. Having to fetch the image fresh can take the "Spin up
63  # environment" stage from ~10s to >2min.
64  #
65  # File transfer: Any files that need to persist between jobs need to be shared
66  # either via cache or workspace. Either of these solutions work the same way:
67  # they archive the contents of the directories and store them into S3 for
68  # retrieval in the next job. Large files or a large number of files can cause
69  # this to be very slow (minutes), so avoiding this overhead where possible can
70  # speed up the total build time.
71  #
72  # Additionally, workspaces are always applied for any downstream job. Putting a
73  # large directory into the workspace to share it among some downstream parallel
74  # jobs can take >1min to upload, then >1min to apply to each downstream job /and
75  # any jobs downstream of those/. Eg, if a job graph is generate_node_modules ->
76  # parallel [ lint | build | test ] -> deploy, the deploy job will also need to
77  # download the node_modules workspace layer even if it only needs the result of
78  # the build job.
79  #
```

So why is it so bad?

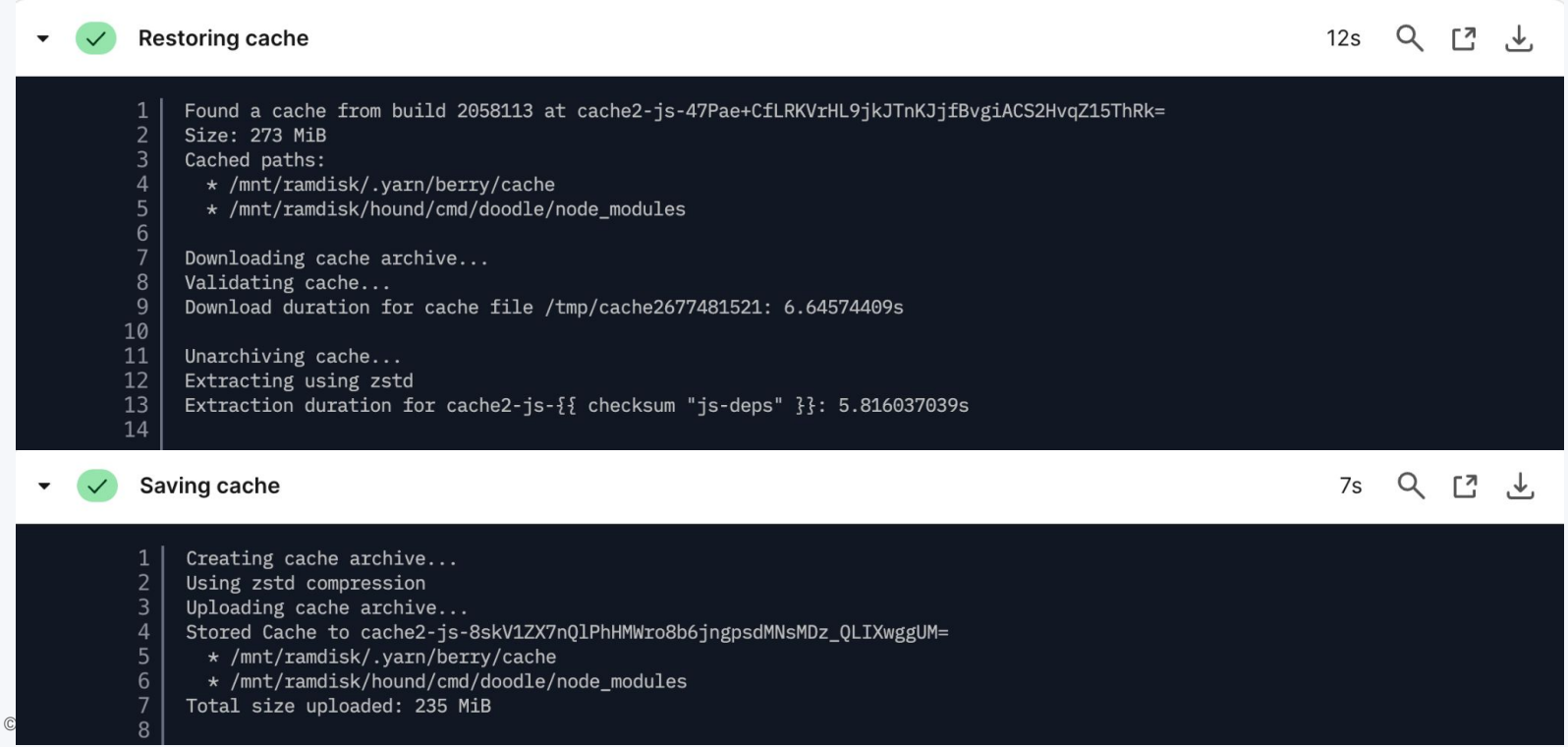


Two answers for slowness

- Amdahl's Law

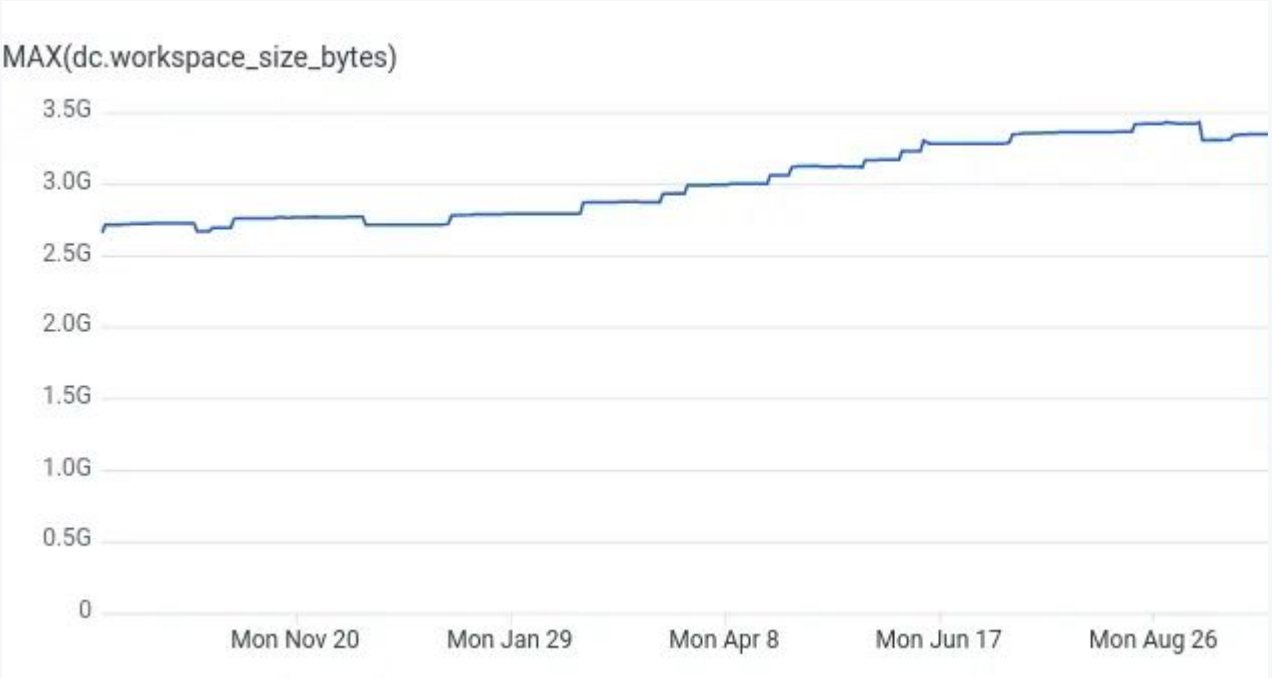
Two answers for slowness

- Amdahl's Law
- Data locality



But the caches weren't the problem. This was.

300MB in cache is okay. 3GB in workspace is… not. (esp w/ gzip)



Do we need to copy all that data?

No. **Hell no.**

- Everything deploys as containers now (we designed for raw VMs)
- Do we even need those intermediate artifacts?
- What if we built everything in one place?

Serial build & passing binaries between steps: terrible.

Introducing parallel build with **Docker Bake**.

We all know Dockerfiles.

But each Docker build is independent.

Spinning up builders takes… a while.

Copying the cache takes even longer.

```
FROM golang:1.21 AS builder
COPY go.mod go.sum ./
RUN go mod download
COPY . .
RUN go build -o myapp
```

Set up remote Docker and Docker layer caching

```
- setup_remote_docker:
    docker_layer_caching: true
```

This step helps you avoid the Docker-in-Docker problem. In fact, you're setting up an environment that is isolated from the CI or primary container environment, then using the remote host's Docker Engine.

The `setup_remote_docker` key configures your remote Docker environment. By setting up a remote Docker environment, you can run Docker commands, such as building and pushing Docker images, within your job.

The `docker_layer_caching` option for your remote environment is set to `true`, which enables Docker layer caching. With Docker layer caching enabled, CircleCI can reuse layers from previous builds if they haven't changed. This significantly reduces build times, as unchanged layers do not need to be rebuilt.

Docker Bake allows composition.

Common tags, repo settings, and more.

```
group "example" {
  targets = ["api", "worker", "frontend"]
}

target "api" {
  dockerfile = "build/api.Dockerfile"
  tags = ["myapp/api:latest"]
}

target "worker" {
  inherits = ["api"]
  dockerfile = "build/worker.Dockerfile"
}

target "frontend" {
  dockerfile = "build/frontend.Dockerfile"
  cache-from = ["type=registry,ref=myapp/cache"]
}
```

Docker Bake allows chaining.

Output of one as input to another.

```
target "poodle-js" {
  contexts = {
    src = "cmd/poodle"
  }
  args = {
    NODE_VERSION = NODE_VERSION
    BUILD_ID = BUILD_ID
  }
  dockerfile = "dockerfiles/poodle.Dockerfile"
}
```

Docker Bake allows chaining.

Output of one as input to another.

```
target "doodle" {
  contexts = {
    src = "cmd/doodle"
    yarn = "cmd/poodle/.yarn"
    poodle-js = "target:poodle-js"
    config = "config"
  }
  args = {
    NODE_VERSION = NODE_VERSION
    BUILD_ID = BUILD_ID
  }
  dockerfile = "dockerfiles/doodle.Dockerfile"
  output = ["type=registry,oci-mediatypes=true,compression=zstd"]
  tags = ["${ECR_HOST}/doodle:${TAG}", "${ECR_HOST}/doodle:${SHA}"]
}
```

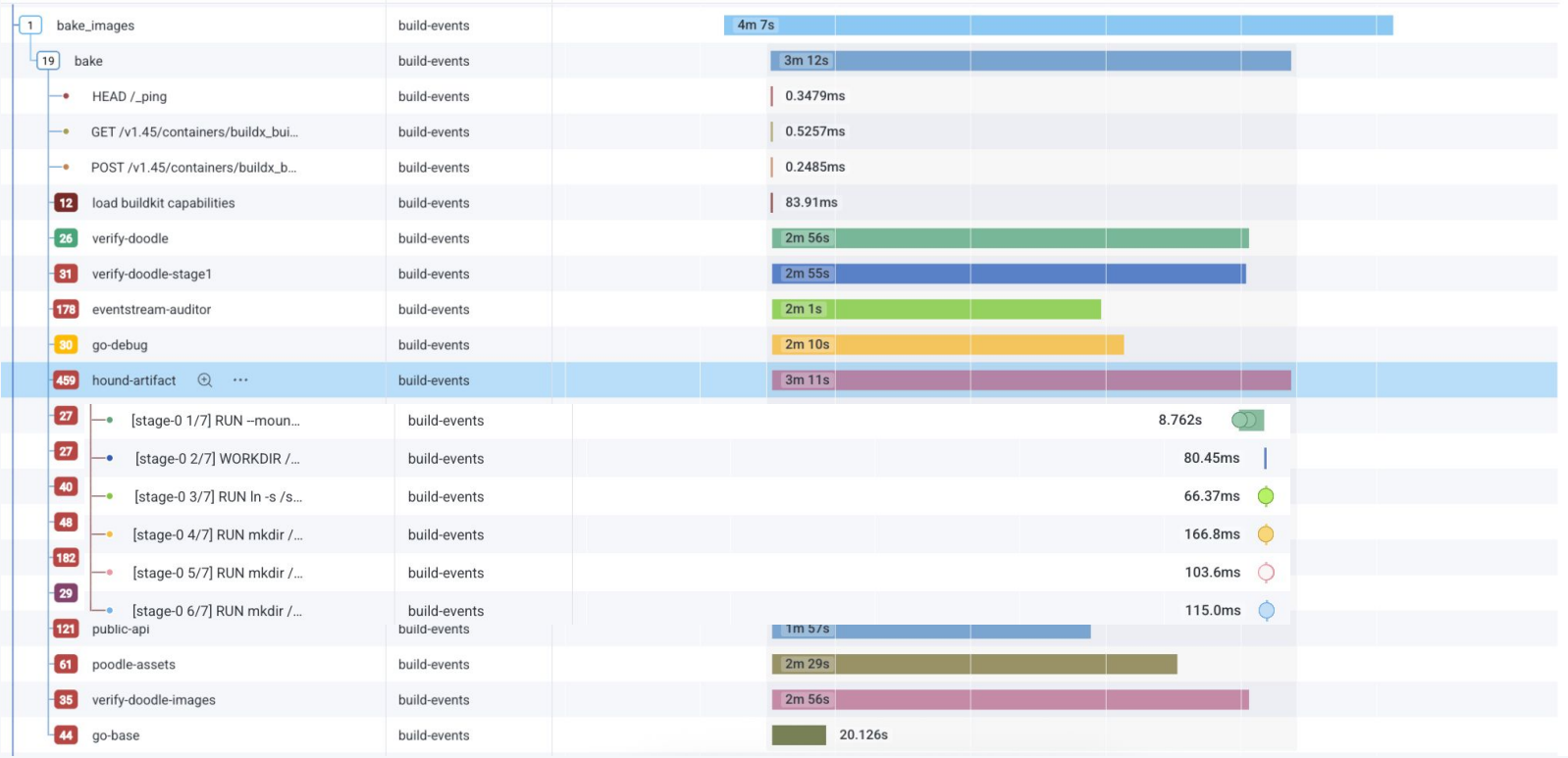
Docker Bake automatically pushes SBOMs.

Attestation & build logging is included for free / pushed to ECR.


```
{
  "builder": {
    "id": ""
  },
  "buildType": "https://mobyproject.org/buildkit@v1",
  "materials": [
    {
      "uri": "pkg:docker/node@22-trixie-slim?
digest=sha256:535ba2ed7dcf0dec29b0af4cac2b87ccdd935880212d4b9537e767b078ce1ca3\u0026platform=linux%2Farm64",
      "digest": {
        "sha256": "535ba2ed7dcf0dec29b0af4cac2b87ccdd935880212d4b9537e767b078ce1ca3"
      }
    }
  ],
  "invocation": {
    "configSource": {},
    "parameters": {
      "frontend": "dockerfile.v0",
      "args": {
        "context:poodle-js": "input:1-poodle-js",
        "dockerfilekey": "dockerfile",
        "frontend.caps": "moby.buildkit.frontend.contexts+forward",
        "input-metadata:1-poodle-js": "{\"containerimage.config\":\"[...]\"}"
      },
      "locals": [
        {
          "name": "context"
        },
        {
          "name": "dockerfile"
        },
        {
          "name": "src"
        }
      ]
    },
    "environment": {
      "platform": "linux/arm64"
    }
  },
  "buildConfig": {
    "llbDefinition": [
      {
        "id": "step0",
        "op": {
          "Op": {
            "source": {
              "identifier": "docker-image://docker.io/library/node:22-trixie-
slim@sha256:535ba2ed7dcf0dec29b0af4cac2b87ccdd935880212d4b9537e767b078ce1ca3"
            }
          },
          "platform": {
            "Architecture": "arm64",
            "OS": "linux"
```

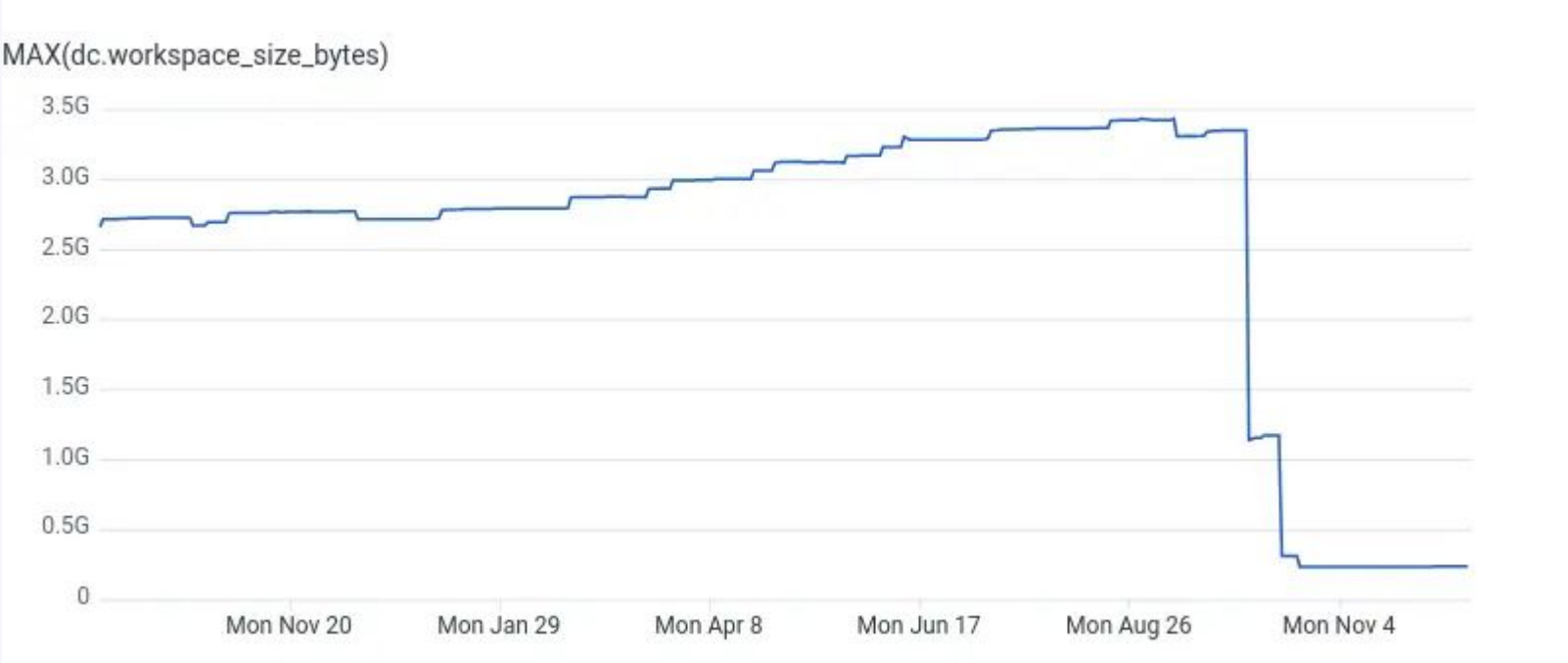
```
        },
        "infos": [
          {
            "filename": "poodle.Dockerfile",
            "language": "Dockerfile",
            "data":
"QVJHIE5PREVfVkVSU0lPTgpGUk9NIG5vZGU6JHtOT0RFX1ZFUlNJT059LXRyaXhpZS1zbGltIGFzIGJhc2UKClNIRUxMIFsiL2Jpbi9iYXNoIiwgIi1ldW8iLCAicGlwZWZhaWwiLCA
iLWMiXQojIFlvdSBjYW4gdW5jb21tZW50IHBhY2thZ2VzIHdpdGggZm9yY2VkIHZlcnNpb25zIGJlbG93IGFzCiMgdGVtcG9yYXJ5IG92ZXJyaWRlcyB0byBnZXQgc2VjdXJpdHkgcGF
0Y2hlcyBub3QgaW4gdGhlIGJhc2UgaW1hZ2UuCiMgSWYgeW91IGJ1bXAgdGhlIGJhc2UgaW1hZ2UsIHNlZSBpZiB0aGVzZSBwYWNrYWdlcyBzdGlsbCBuZWVkIHRvIGJlCiMgcGlubmV
kIGF0IHRoZXNlIHZlcnNpb25zLgojUlVOIC0tbW91bnQ9dHlwZT1jYWNoZSx0YXJnZXQ9L3Zhci9jYWNoZS9hcHQsc2hhcmluZz1sb2NrZWQgYXB0IHVwZGF0ZSBcCiMgICYmIGFwdCB
pbnN0YWxsIC15IC0tbm8taW5zdGFsbC1yZWNvbW1lbmRzIC10IHRyaXhpZS1zZWN1cml0eSBcCiMgIG9wZW5zc2gtY2xpZW50IFwKIyAgY3VybCBcCiMgICYmIGFwdCBpbnN0YWxsIC1
5IC0tbm8taW5zdGFsbC1yZWNvbW1lbmRzIFwKIyAgbm9kZS1xcz02LjkuNCtkcy0xK2RlYjExdTEKCkVOViBDSSB0cnVlCldPUktESVIgL2hvdW5kL2NtZC9wb29kbGUKQ09QWSAtLWZ
yb209c3JjIC0tbGluayAuIC9ob3VuZC9jbWQvcG9vZGxlClJVTiAtLW1vdW50PXR5cGU9Y2FjaGUsdGFyZ2V0PSIvcm9vdC8ueWFybi9iZXJyeS9jYWNoZSIsc2hhcmluZz1wcml2YXR
lIHlhcm4gaW5zdGFsbCAtLWltbXV0YWJsZQoKIyBtdXN0IGJlIGRlY2xhcmVkIGFmdGVyIHRoZSBGUk9NIGJ1dCBhZnRlciB5YXJuIGluc3RhbGwsIHRvIGF2b2lkIGludmFsaWRhdGl
vbiEKQVJHIEJVSUxEX0lECgpSVU4gLS1tb3VudD10eXBlPWNhY2hlLHRhcmdldD0iL2hvdW5kL2NtZC9wb29kbGUvbm9kZV9tb2R1bGVzLy5jYWNoZSIsc2hhcmluZz1wcml2YXRlIHl
hcm4gYnVpbGQgLS1lbnYgQlVJTERfSUQ9JHtCVUlMRF9JRH0gLS1lbnYgQ0k9dHJ1ZQo=",
            "llbDefinition": [
              {
                "id": "step0",
                "op": {
                  "Op": {
                    "source": {
                      "identifier": "local://dockerfile",
                      "attrs": {
                        "local.differ": "none",
                        "local.followpaths": "[\"poodle.Dockerfile\",\"poodle.Dockerfile.dockerignore\"]",
                        "local.sharedkeyhint": "dockerfile"
                      }
                    }
                  },
                  "constraints": {}
                }
              },
              {
                "id": "step1",
                "op": {
                  "Op": {}
                },
                "inputs": [
                  "step0:0"
                ]
              }
            ],
            "digestMapping": {
              "sha256:edf5825ef934bed059ed0b7ffbee57552f7b4efe8101d01ceddf7c91a18eba13": "step0",
              "sha256:fea1856d4a4d316aadf16d801920f38e21d5fbf1e2d790d32713678799743889": "step1"
            }
          },
          {
            "filename": "Dockerfile",
            "language": "Dockerfile",
            "data":
"RlJPTSBzY3JhdGNoCkNPUFkgLS1mcm9tPXBvb2RsZS1qcyAtLWxpbmsgL2hvdW5kL2NtZC9wb29kbGUvc3RhdGljL2Rpc3QgY21kL3Bvb2RsZS9zdGF0aWMvZGlzdAo=",
            "llbDefinition": [
```

Also, it natively supports OTel!



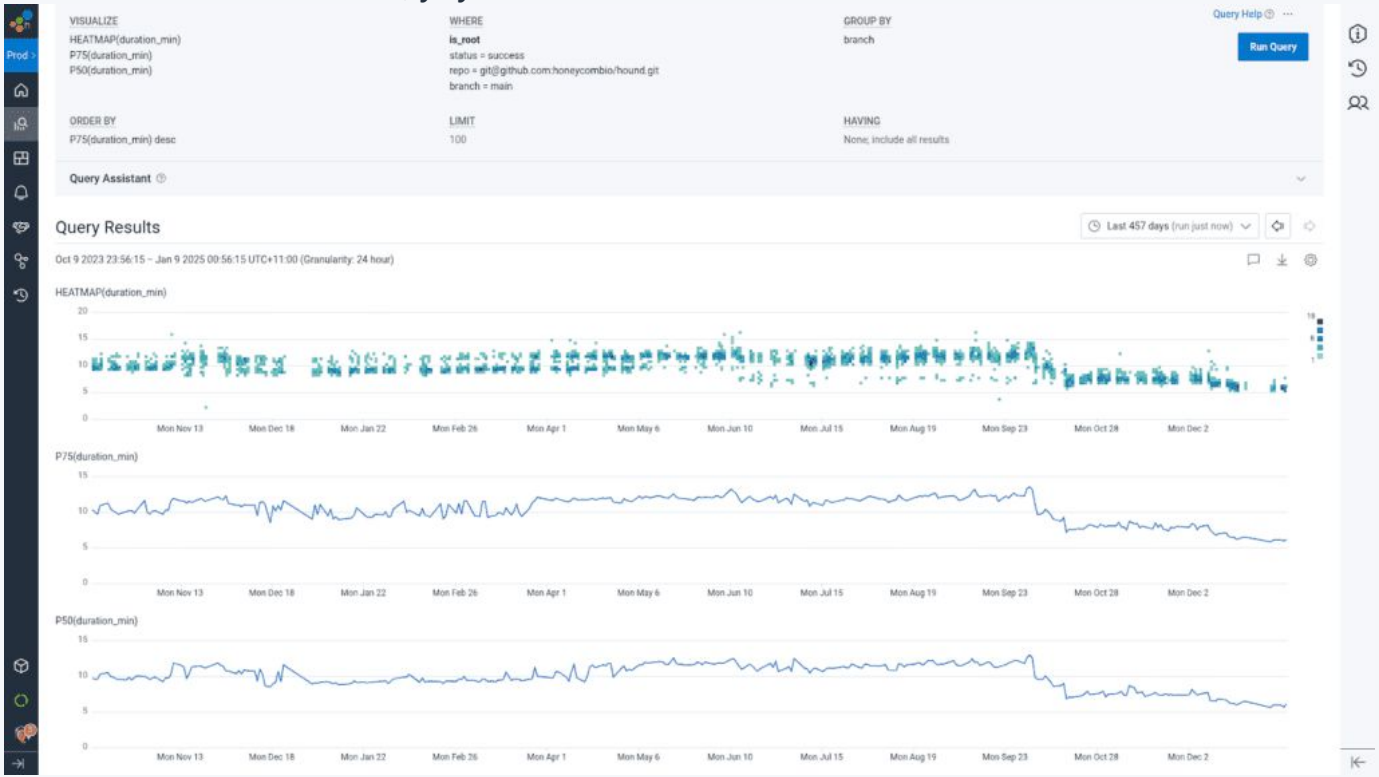
So how'd it work out for us?

Much less data transferred!



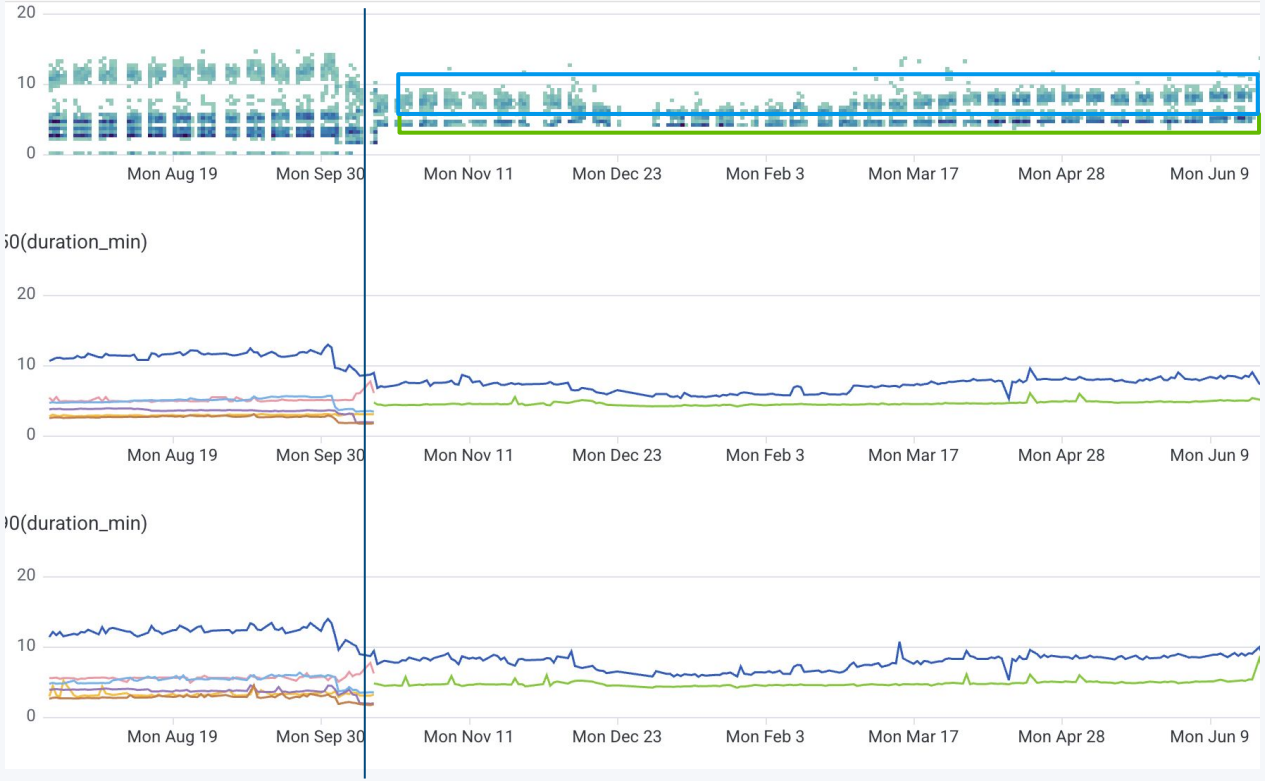
So how'd it work out for us?

Back down to 7 minutes, yay!



But the honeymoon didn't last…

Bake time remained static (5 min), other dependencies regressed.



Making things faster

First, fix the test parallelism / dependencies. Then:

Remember when we turned OFF Docker Layer Caching to save transfer latency & costs? What if we could have our cake and eat it too?

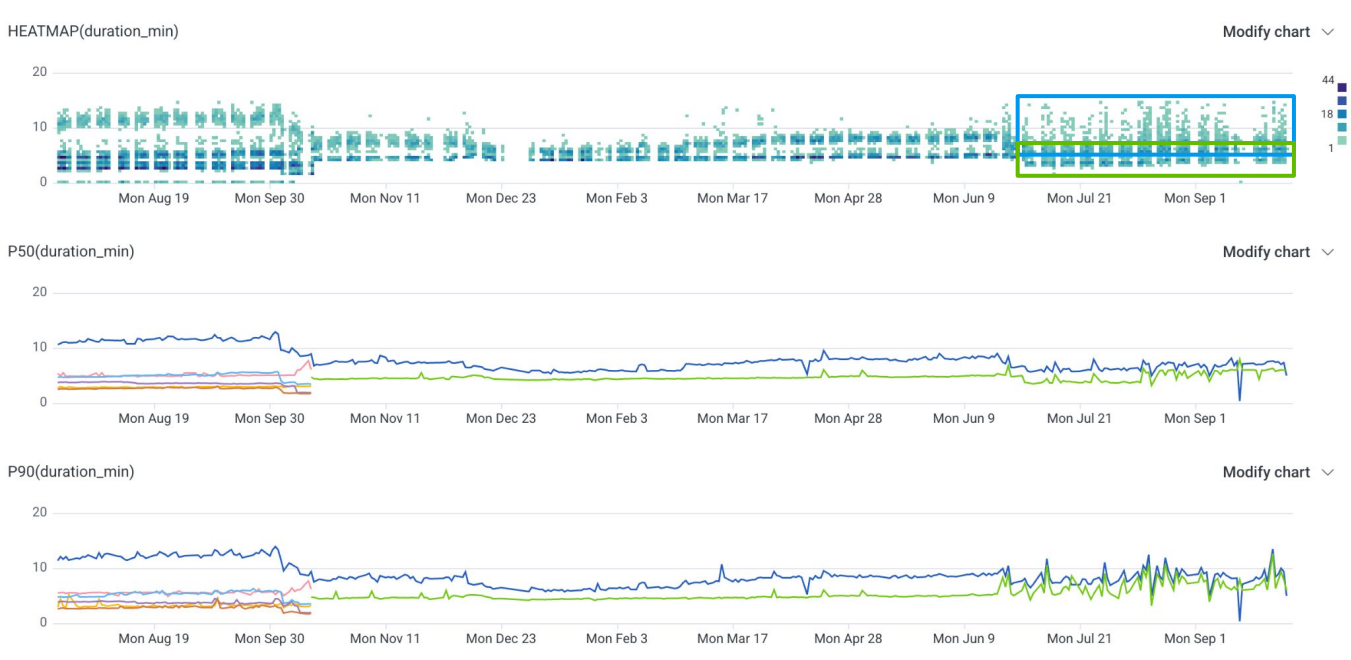
namespace


```
--mount=type=cache,target="/hound/cmd/poodle/node_modules/.cache",sharing=private
--mount=type=cache,target="/root/.yarn/berry/cache",sharing=private
--mount=type=cache,target="/root/.cache/go-build"
--mount=type=cache,target="/go/pkg"
--mount=type=cache,target=/var/cache/apt,sharing=locked
```

Technical tradeoffs

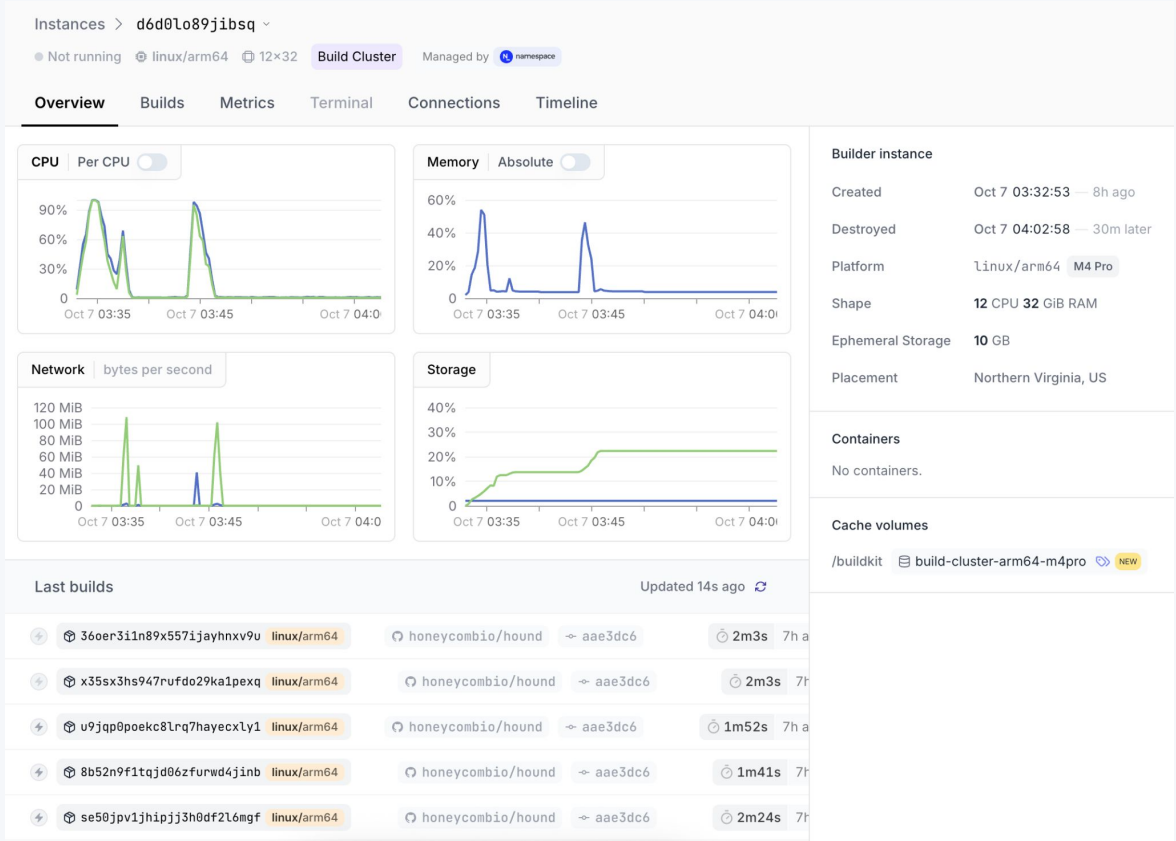
CircleCI has a massive pool (essentially all of AWS)

Namespace is bare metal & provisions on-demand. p50 vs p90 latency



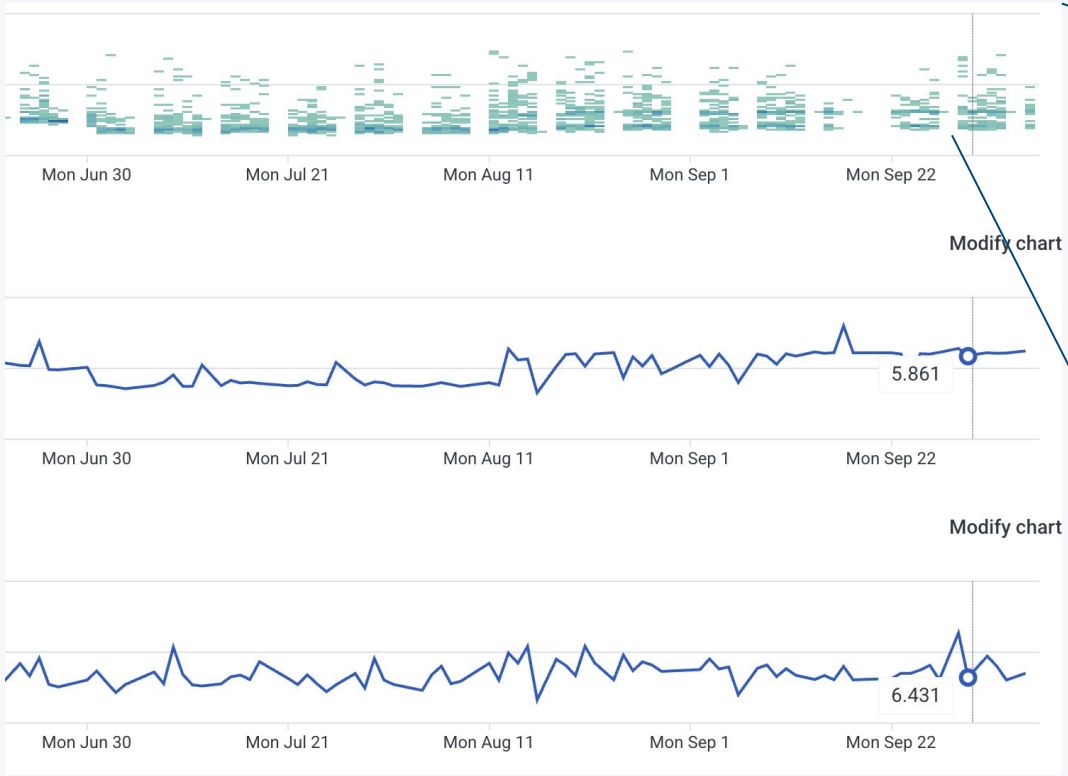
Build & machine level observability

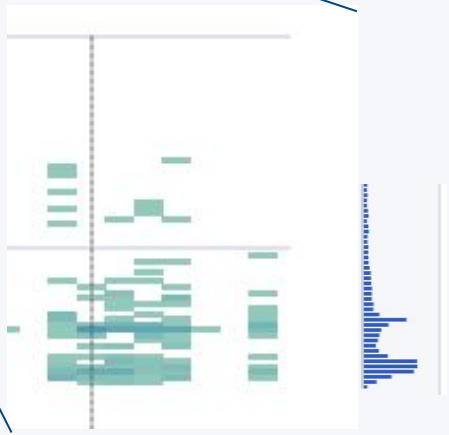
Build & machine level observability



(this explains the trimodal behaviour!)

Cold cache vs warm cache vs overloaded machine

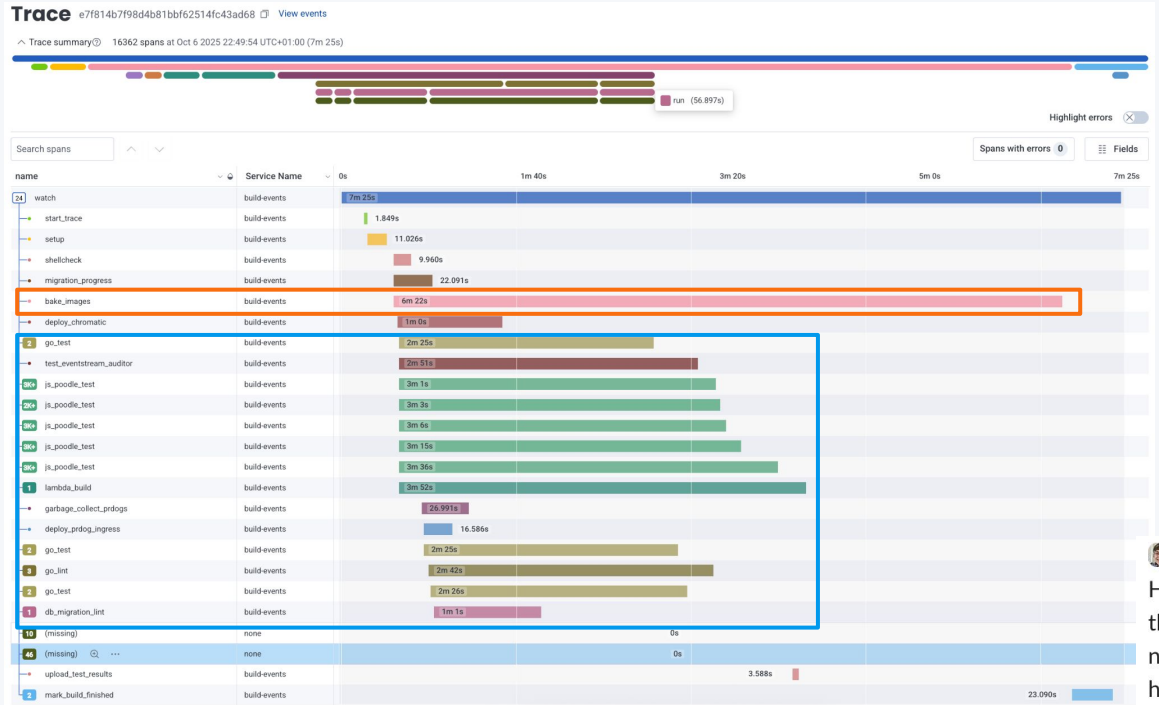




Optimising for devex

Fast common case > consistent mediocrity

Even the 2025 "bad" builds are better than old 2024 "good" builds



dean (on sabbatical until 11/3) 10:01 PM

strawpoll: `make test` works and all tests pass

✅ yes

❌ no

😎 I do not ever run `make test` locally

✅ 1 ❌ 3 😎 14

+2 21 replies Last reply 2 months ago

talor Sep 26th at 7:16 PM

builds at my last job lasted 20-30min. 5 min build is life changing. shoutout @lizf_dub_srecon & everyone else working to keep it that way

11 2 3 2

1 reply

dean (on sabbatical until 11/3) Sep 26th at 7:52 PM

I think we're closer to 7 or 8 right now but I am hell bent on getting it <5

4 THANK YOU 2

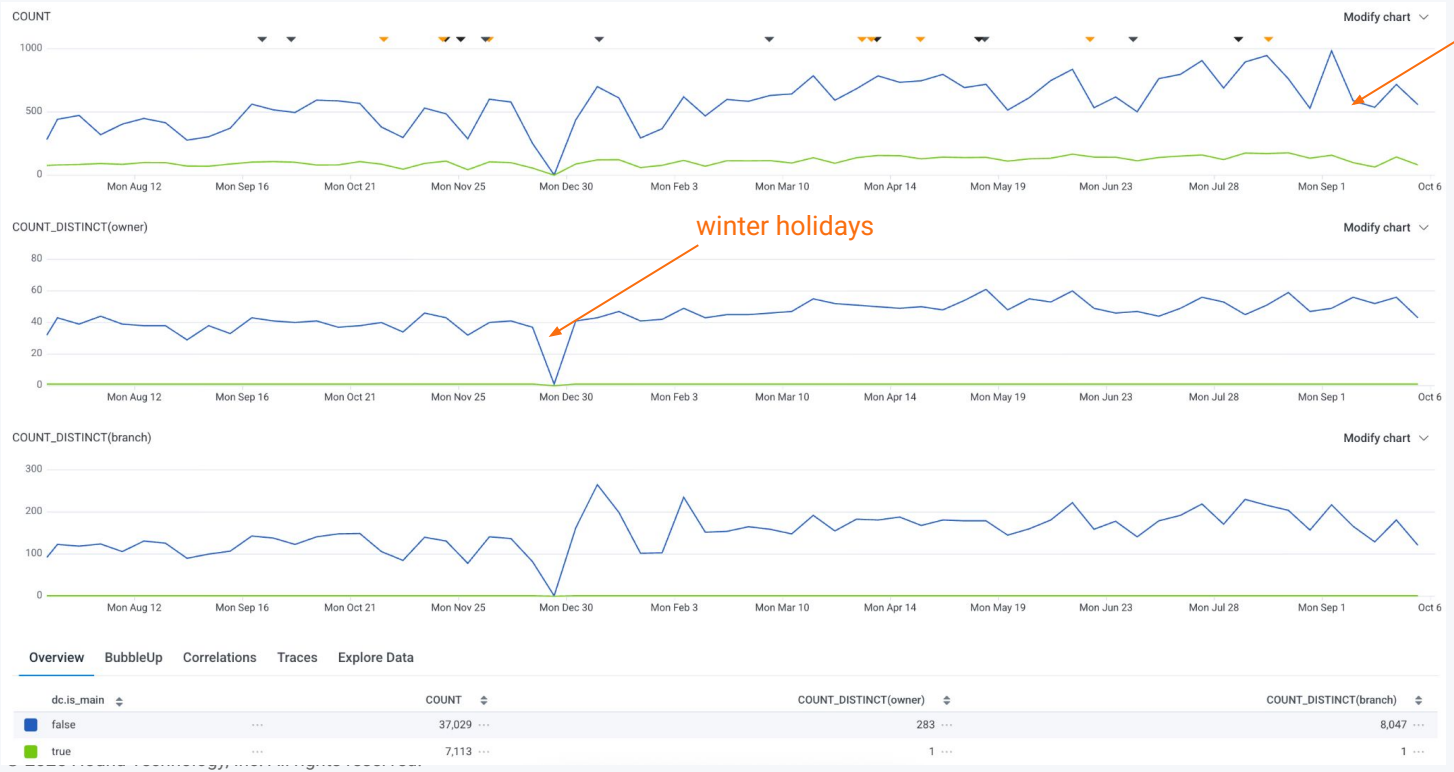
Ruthie Irvin

Huge love to @lizf_dub_srecon because my build just failed in the last minute due to css module linting errors only, and the next build will take so so so many fewer minutes than if this had happened to me just a few months ago. (edited)

Posted in # love | Dec 17th, 2024 | View message

Oh, and AI made this problem more critical…

Builds per week doubled… (but our CI spend didn't!)



Eyebrow Subtitle

Optimisation results

2x builds

20 merges per day to 40 merges per day

–50% time

15 minutes to 7 minutes at p90

Key takeaways

1. Data transfer between jobs can dominate build time
2. Docker Bake enables parallelism without data movement
3. Buildkit's native OTel support enables deep observability
4. Cache locality matters more than scale-up runner pool
5. Measure P50/P90/P99, not just vibes
6. Tradeoffs are explicit choices, not accidents
7. Your architecture should match your deployment model

Should you do this at home?

Consider if:

- You're building multiple related containers
- You're moving >1GB between build steps
- You deploy exclusively to containers
- You have complex build dependencies

Skip if:

- You build one simple container (or not containerised yet)
- Your builds are already under 3 minutes

Start here:

- Instrument your current builds (buildevents)
- Find & fix your serialisation bottlenecks
- Identify build input/output reuses.

honeycomb.io

Liz Fong-Jones

Field CTO | Honeycomb

/in/efong

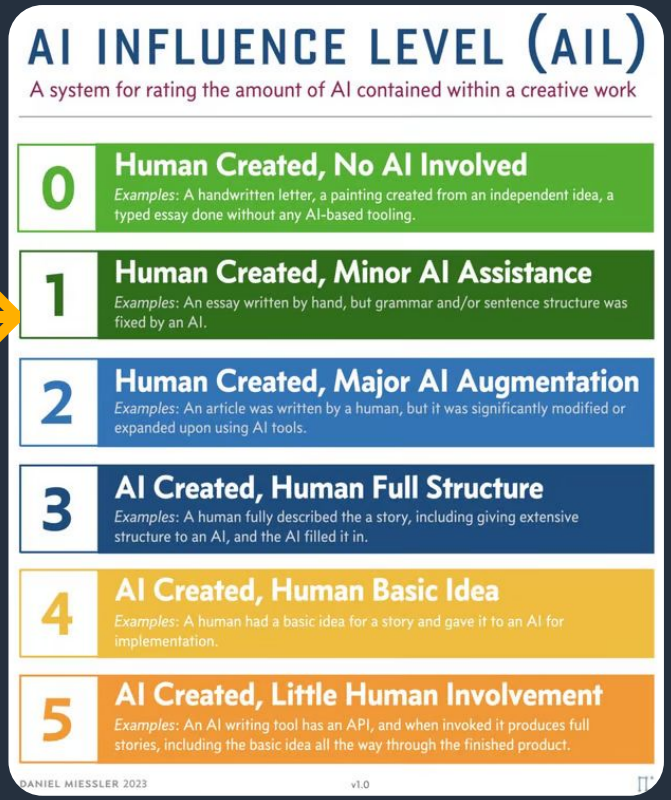
lizf@honeycomb.io

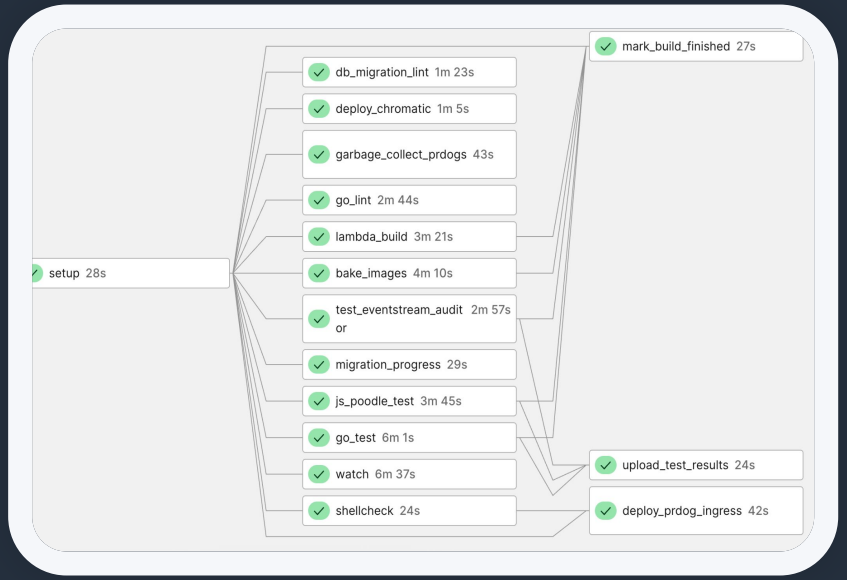
lizthegrey.com

Resources

- hny.co/blog/most-important-developer-productivity-metric-build-times
- circleci.com/developer/orbs/orb/honeycombio/buildevents
- docs.docker.com/build/bake
- namespace.so/docs/solutions/docker-builders

Thank you.





This talk is **AIL-1.5** | All images are **AIL-0**
(either taken by me, or licenced stock images)

Art should be made by artists, not machines.

danielmiessler.com/blog/ai-influence-level-ail

See everything. Solve anything.

honeycomb.io