lambda DAλS

27-28 MAY 2024

KRAKÓW | POLAND

lambda DAλS
27-28 MAY 2024
KRAKÓW | POLAND

From 1 to 100k users:

Lessons learned from scaling a Haskell app

Félix Miño

Agenda

- whoami
- Project overview
- Lessons learned:
    - Observe 👀
    - What really matters? 🤔
    - Low hanging wins 🎂
    - DB performance 💾
    - A very particular case 💀
    - Scaling is not a task is a project ✅
- Summarizing

lambda DAYS 27-28 MAY 2024 KRAKÓW | POLAND

Whoami

- Software developer (mostly haskeller) and community lead at Stack Builders.

- Quito Lambda meetup coordinator:
    - https://www.meetup.com/pl-PL/Quito-Lambda-Meetup/

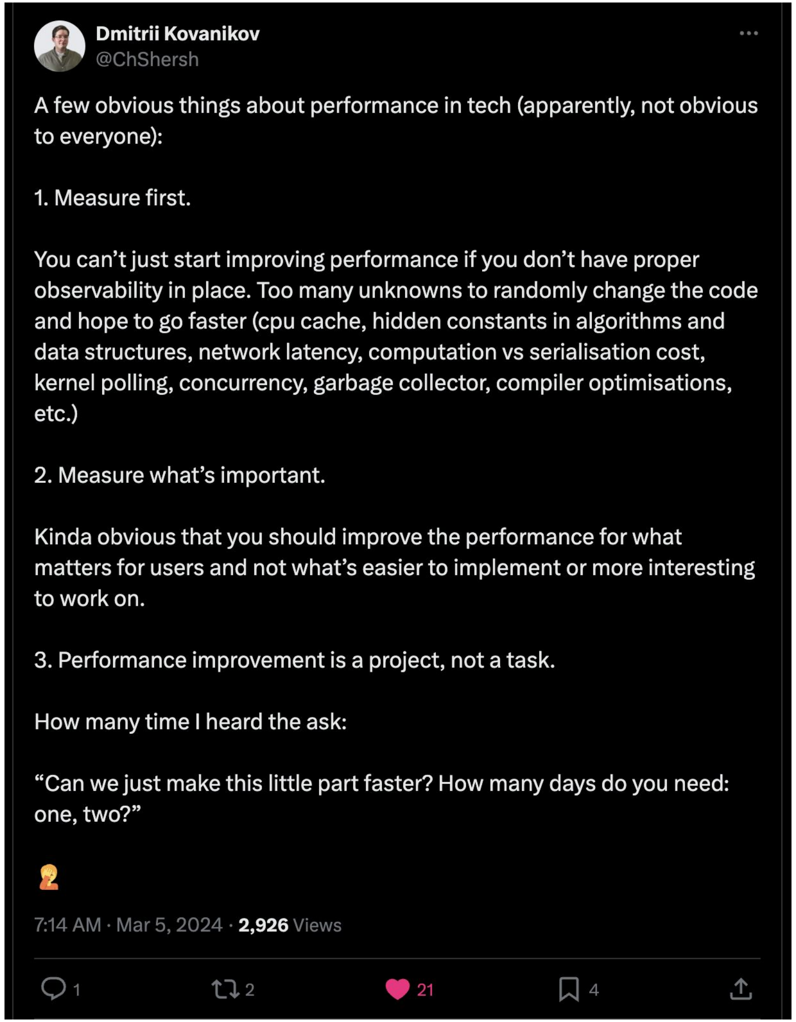
- Runner for fun.

Project overview

1. Haskell web application: servant (90%) and yesod (10%).
2. 13 microservices written in Haskell.
3. We maintain ~3M of LOC in Haskell.

Lessons learned

Reflect on our experiences, key insights identified to enhance future performance and avoid repeating past mistakes.



Dmitrii Kovanikov
@ChShersh

A few obvious things about performance in tech (apparently, not obvious to everyone):

1. Measure first.

You can't just start improving performance if you don't have proper observability in place. Too many unknowns to randomly change the code and hope to go faster (cpu cache, hidden constants in algorithms and data structures, network latency, computation vs serialisation cost, kernel polling, concurrency, garbage collector, compiler optimisations, etc.)

2. Measure what's important.

Kinda obvious that you should improve the performance for what matters for users and not what's easier to implement or more interesting to work on.

3. Performance improvement is a project, not a task.

How many time I heard the ask:

"Can we just make this little part faster? How many days do you need: one, two?"

🤦

7:14 AM · Mar 5, 2024 · **2,926** Views

1 2 21 4

lambda DAλS
27-28 MAY 2024
KRAKÓW | POLAND

Observe 👀

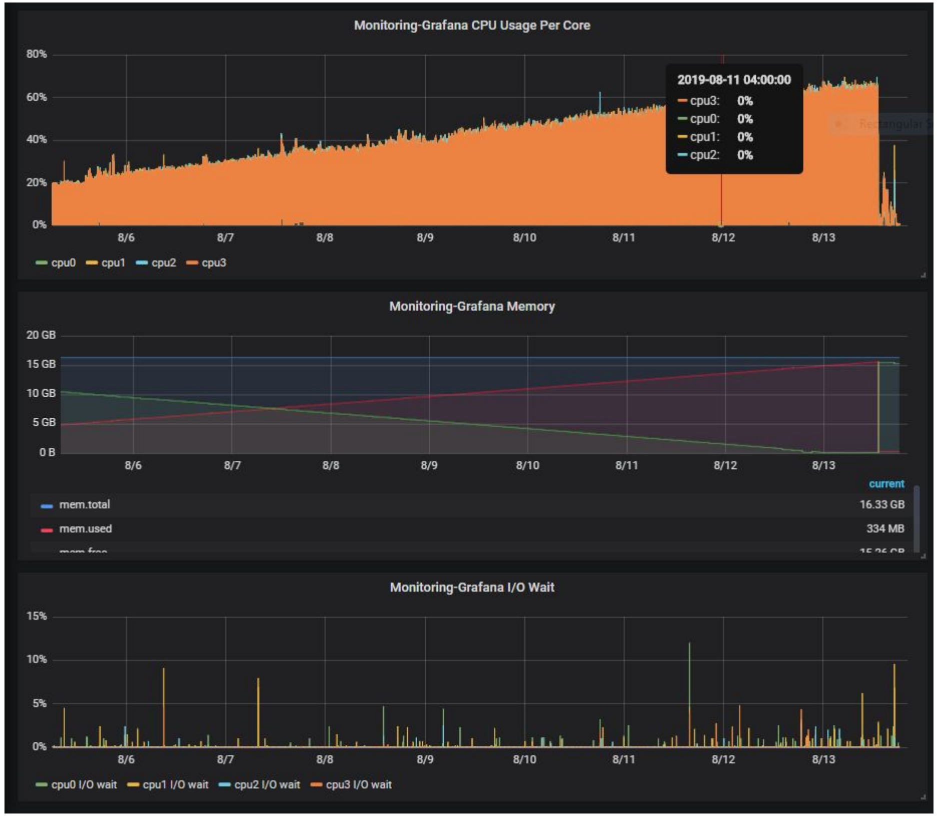
You can't change what you don't know

Observe

- Our observability was almost zero :(
- Started using servant-prometheus
- It took us ~3 months to have something in place
- Faster is better.

A step further

- Analyze your HTTP requests deeper:
  - How much time am I spending in db? And how much in actual computations?
  - We implemented an ad-hoc solution to make this distinction.



https://community.grafana.com/t/high-memory-usage-when-running-grafana-in-docker/20156

lambda DAλS
27-28 MAY 2024
KRAKÓW | POLAND

Lessons learned

- Benchmarking: Compare current percentiles with historical data to identify trends or performance improvements.
- Plan for incremental changes
- Understand your libraries:
  - Your performance can be affected
  - A note on quantiles: https://github.com/serokell/servant-prometheus?tab=readme-ov-file#a-note-on-quantiles

What really matters? 🤔

What really matters?

- What are your most critical services?
- What is the core of your business?
- What are your most used endpoints?
- What percentile do you really care? P90, p95, p99 (it usually depends of your business' need)

Lessons learned

- We spent time on things that were not as important as others.
- “I improved this endpoint by **90%**”
    - We discovered this endpoint was rarely hit when running our load tests.
- Run your load tests as often as possible.

Low hanging wins 🎂

1. N+M calls

- You can have nested N+1 calls.
- N+1 is not necessarily db calls.
- This was one of the most common performance issue we had in our application.
    - ~70% of the scalability tickets we solved involved N+1.

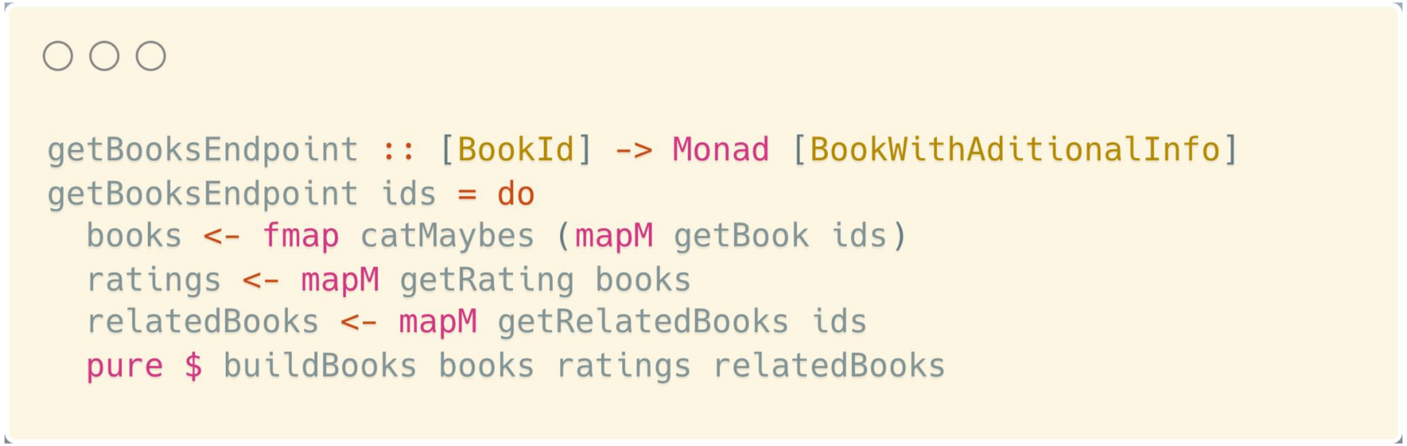
N+3 Case

- Found a particular endpoint where we had a N+1 three times.
- Two N+1 were db related and the other was a call to another internal service.

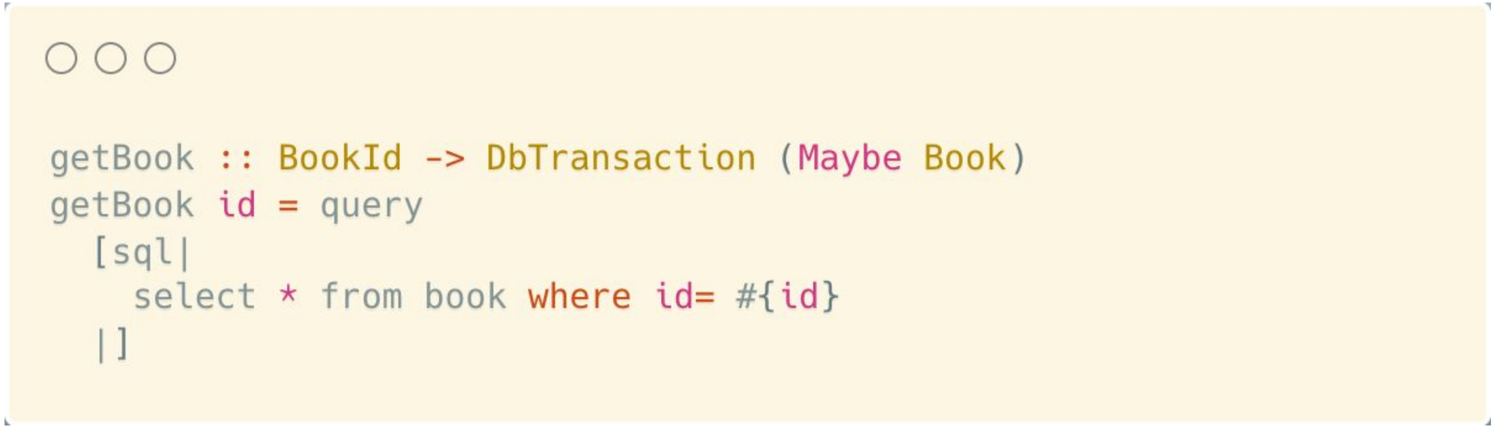
```haskell
type BookId = UUID

data Book = Book
{ bookId :: BookId,
  name :: Text,
  publishedAt :: UTCTime
}

data BookWithAditionalInfo = BookWithAditionalInfo
{ book :: Book,
  rating :: Decimal,
  relatedBooks :: [Book]
}
```

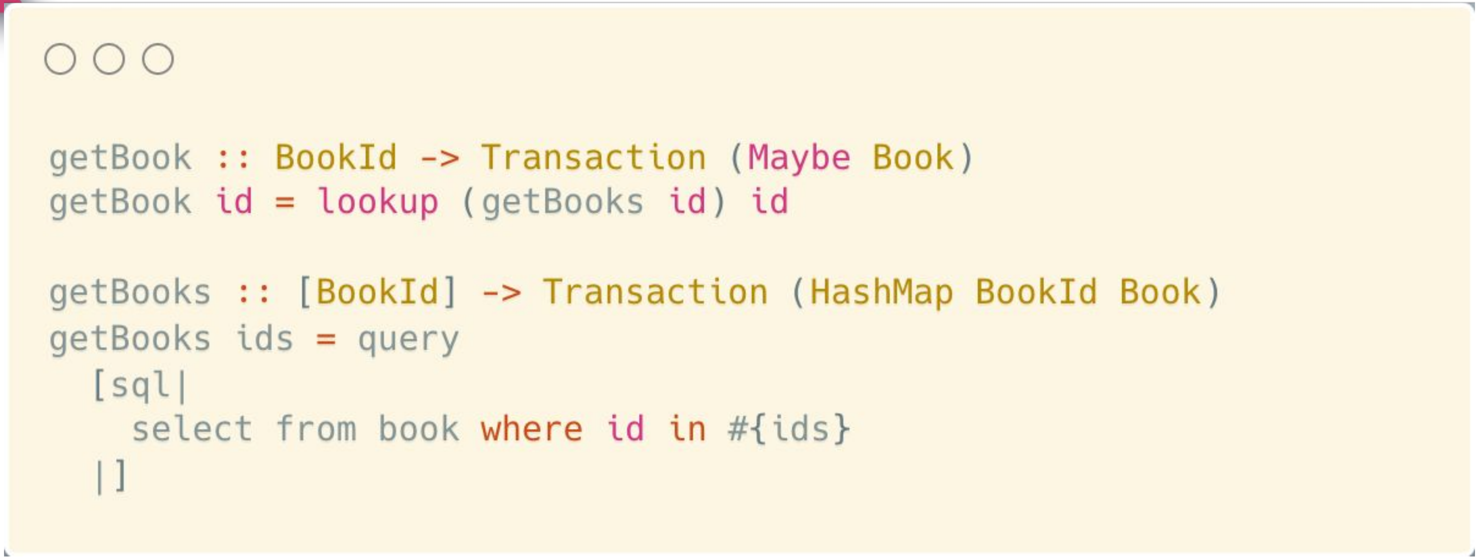


```haskell
getBooksEndpoint :: [BookId] -> Monad [BookWithAditionalInfo]
getBooksEndpoint ids = do
  books <- fmap catMaybes (mapM getBook ids)
  ratings <- mapM getRating books
  relatedBooks <- mapM getRelatedBooks ids
  pure $ buildBooks books ratings relatedBooks
```



```
getBook :: BookId -> DbTransaction (Maybe Book)
getBook id = query
  [sql|
    select * from book where id= #{id}
  |]
```

lambda DAλS
27-28 MAY 2024
KRAKÓW | POLAND

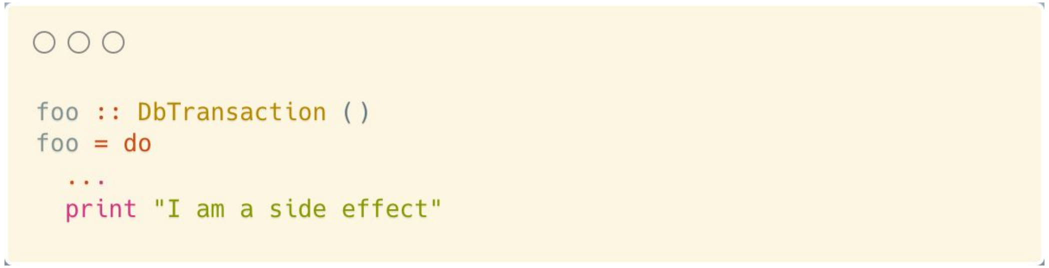


```haskell
getBook :: BookId -> Transaction (Maybe Book)
getBook id = lookup (getBooks id) id

getBooks :: [BookId] -> Transaction (HashMap BookId Book)
getBooks ids = query
  [sql|
    select from book where id in #{ids}
  |]
```

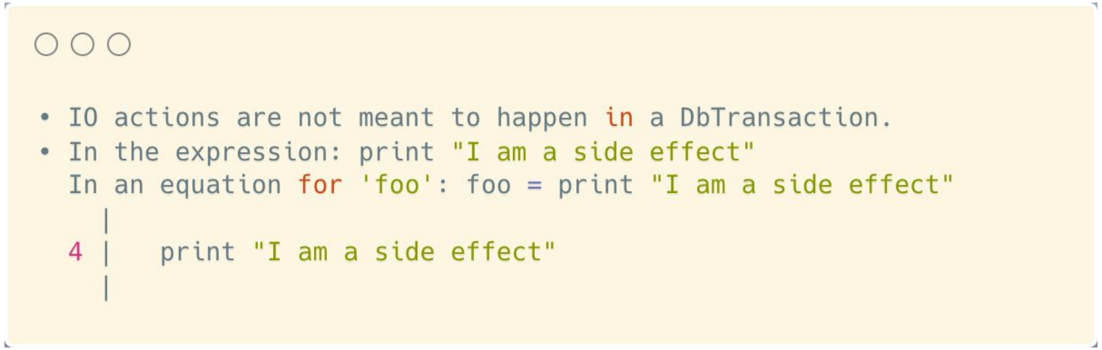
2. Using type errors and monads

- Avoid common mistakes at compile time.
    - Don’t allow IO operations inside your db transactions.
    - We want to spend the less time possible blocking our db.



```
foo :: DbTransaction ()
foo = do
  ...
  print "I am a side effect"
```

lambda DAYS
27-28 MAY 2024
KRAKÓW | POLAND



```
• IO actions are not meant to happen in a DbTransaction.
• In the expression: print "I am a side effect"
  In an equation for 'foo': foo = print "I am a side effect"
    |
  4 |   print "I am a side effect"
    |
```

lambda
DAλS
27-28 MAY 2024
KRAKÓW | POLAND

3. Haskell performance checklist

- https://github.com/haskell-perf/checklist
- **Have you setup an isolated benchmark?**
    - This is particularly important when facing convoluted code, so you can determine the root cause. Use criterion.
- **Have you looked at strictness of your function arguments?**
    - Avoid space leaks at all cost: https://chshersh.com/space-leak

lambda DAYS 27-28 MAY 2024 KRAKÓW | POLAND

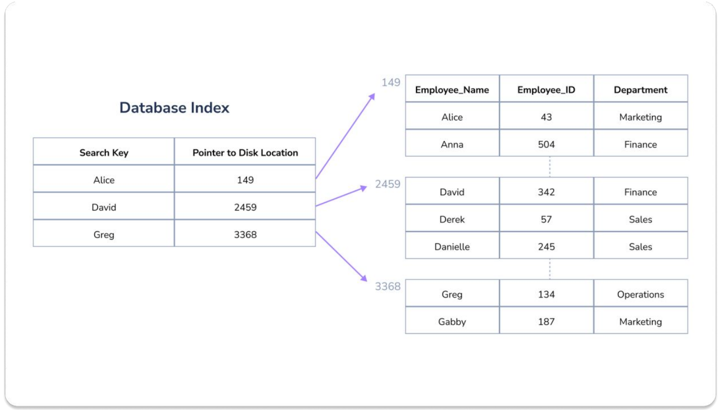
Haskell performance checklist

- **Are you using the right data structure?**
  - “Lists are almost always the wrong data structure. But sometimes they are the right one.”
- **Are your data types strict and/or unpacked?**
  - If your set of data is not that big think of using the strict version of your libraries.

DB Performance 💾

1. Database indexes

- An index is a copy of selected columns of data, from a table, that is designed to enable very efficient search.



Search Key	Pointer to Disk Location
Alice	149
David	2459
Greg	3368

Employee_Name	Employee_ID	Department
Alice	43	Marketing
Anna	504	Finance
David	342	Finance
Derek	57	Sales
Danielle	245	Sales
Greg	134	Operations
Gabby	187	Marketing

https://www.linkedin.com/pulse/database-index-selvamani-govindaraj

Database indexes (postgresql)

Index type	Characteristics
B-tree Index	Default index created by postgresql
Hash Indexes	Most suitable for equality comparisons, such as = or IN operations
GiST and SP-GiST Indexes	They are particularly useful for handling complex data structures and spatial data, use it if you want to speed up full-text search.
BRIN Indexes	It is particularly useful for data that exhibits sequential or sorted characteristics, such as time series data or data with a natural ordering

Lessons learned 1

- “It is not recommended to create an index on the fly just before running a one-off query. Creating a well-designed index requires careful planning and testing.”
- https://www.freecodecamp.org/news/postgresql-indexing-strategies/

Lesson learned 2

- Standardize your code review process.
  - Include meaningful information in your PR template.
  - Ask for db execution plan (check for sequential scans).

Lesson learned 3

- Indexes could be tricky to test locally:
  - You usually don’t have enough data to make your db use the index in your env.
  - You can force your indexes locally by using the `enable_seqscan` flag.

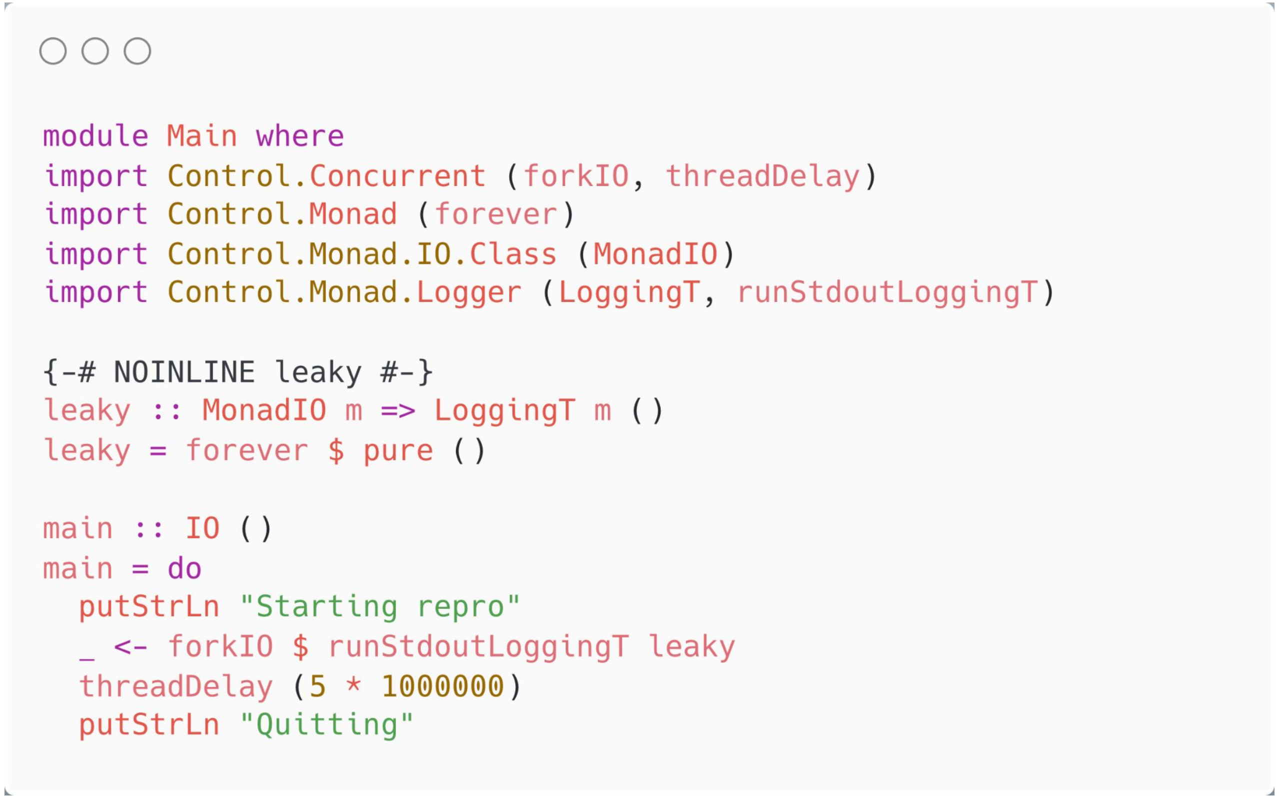
2. View when needed

- Do you have a long running query?
- Does your data not change that often?
- Have you explore using materialized views?
  - In a particular case we were able to go from a 40s query to 1s.
- You can enable slow queries at postgresql level for more insights.

A very particular case 💀

A brief story

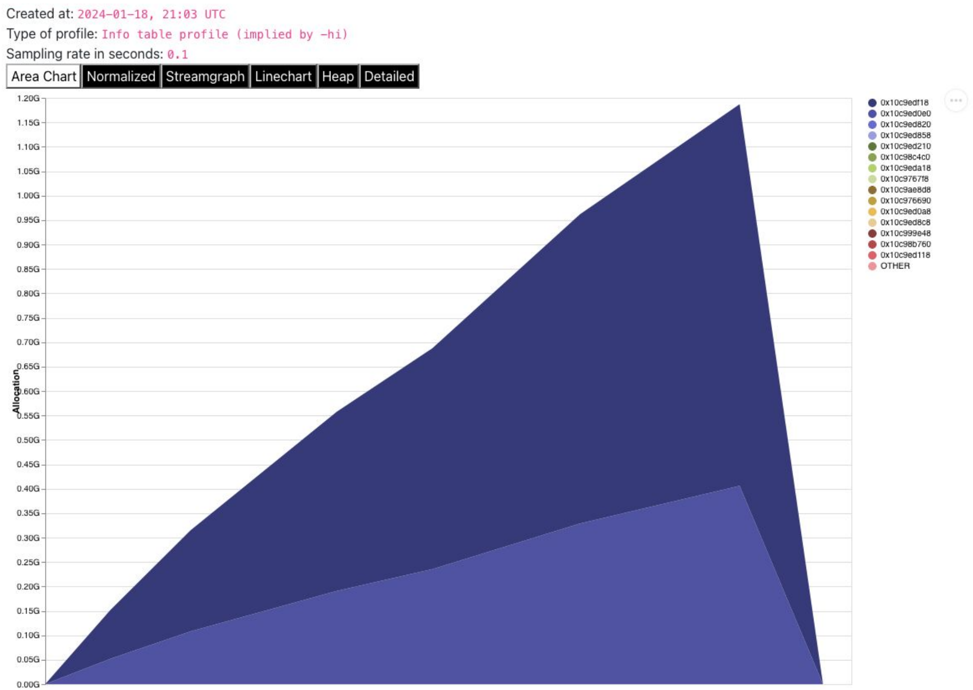
- We migrated from ghc 8 to ghc 9.
- Running our scalability tests we discovered that a service's performance was downgraded by 50%.
- We didn't pass the minimum requirements to deploy.
- We started looking at changes in that service and guess what… aside from ghc changes that were needed we didn't change it any code.



```haskell
module Main where
import Control.Concurrent (forkIO, threadDelay)
import Control.Monad (forever)
import Control.Monad.IO.Class (MonadIO)
import Control.Monad.Logger (LoggingT, runStdoutLoggingT)

{-# NOINLINE leaky #-}
leaky :: MonadIO m => LoggingT m ()
leaky = forever $ pure ()

main :: IO ()
main = do
  putStrLn "Starting repro"
  _ <- forkIO $ runStdoutLoggingT leaky
  threadDelay (5 * 1000000)
  putStrLn "Quitting"
```



Created at: `2024-01-18, 21:03 UTC`
Type of profile: `Info table profile (implied by -hi)`
Sampling rate in seconds: `0.1`

Lessons learned

- Do you have enough/right tooling?
  - Use something like eventlog2html to visualize your eventlog, thanks to this issue we started using it actively.
  - https://hackage.haskell.org/package/eventlog2html
- Isolate your issue.
  - You need a minimum reproducible example.

Scaling is not a task, is a project ✅

Our goals and efforts

- Started at 2022 Q1, our goal is 10x users by the end of 2025.
- Currently at 5x.
- This is an ongoing all company effort.
- All engineering teams were involved and accountable since the beginning.
- Currently a dedicated team of 4 people work continuously in performance improvements for our Haskell code base.

Summarizing

- Scaling an application is not an easy task, it requires commitment and continuous effort from all team members (no just engineering).
- Establish and enforce processes that enable them to effectively manage the challenges associated with scaling.
- Performance improvements can’t be threated as one time task, IT IS A PROJECT.

@felixminom

lambda DAYS 27-28 MAY 2024 KRAKÓW | POLAND