SCALABLE AND RELIABLE VIDEO TRANSCODING IN HASKELL

Alfredo Di Napoli

Haskell Exchange 2015

ABOUT ME

Full story at: **http://goo.gl/qkKwKm**

```
launchMissile :: IO ()
```

IRIS CONNECT

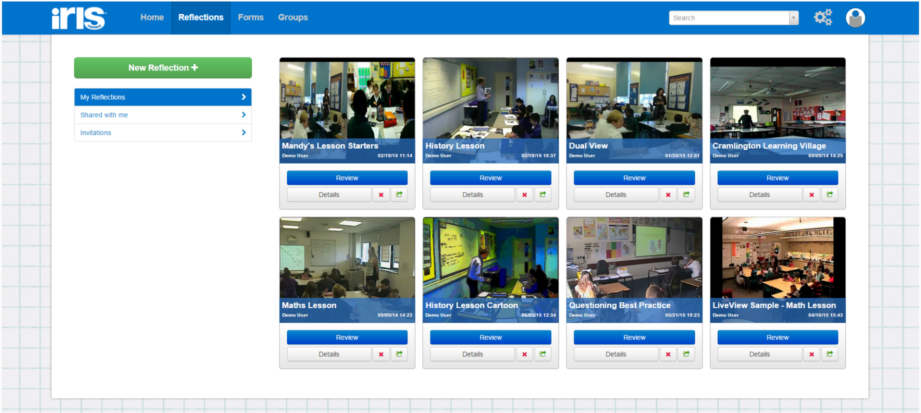
iris connect™

discover. develop. share.

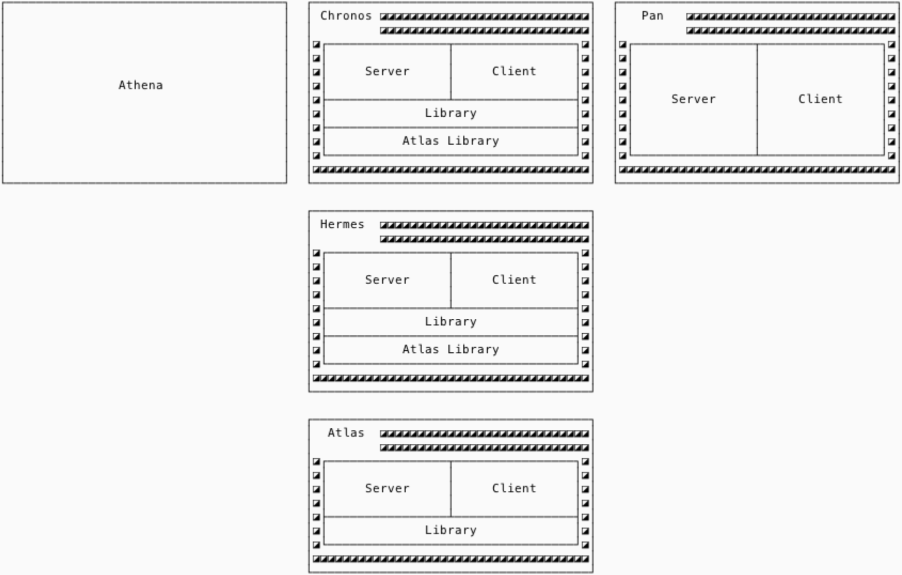
~ Present in over 1800 schools Worldwide (mostly UK, Europe, US & Australia)

~ Used by over 32000 teachers

IRIS CONNECT (CONTD.)



IRIS' GREEK ZOO



Athena

Chronos

Server

Client

Library

Atlas Library

Pan

Server

Client

Hermes

Server

Client

Library

Atlas Library

Atlas

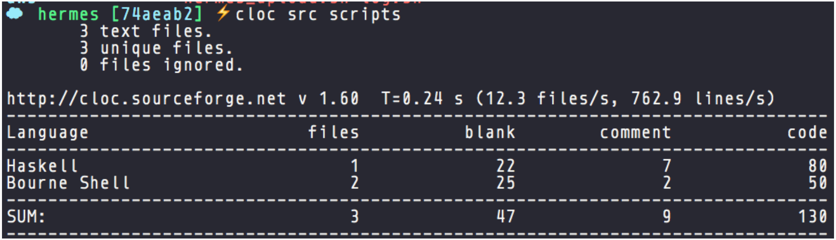
Server

Client

Library

HERMES’ EVOLUTION

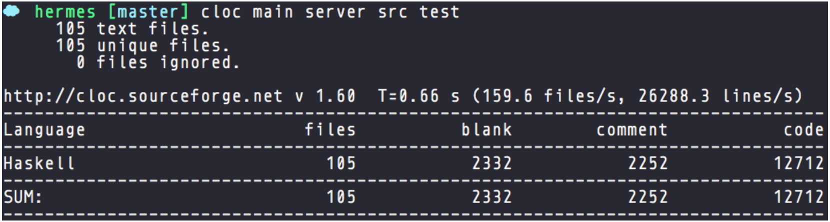
October, 2013



```
☁ hermes [74aeab2] ⚡cloc src scripts
       3 text files.
       3 unique files.
       0 files ignored.

http://cloc.sourceforge.net v 1.60  T=0.24 s (12.3 files/s, 762.9 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Haskell                          1             22              7             80
Bourne Shell                     2             25              2             50
-------------------------------------------------------------------------------
SUM:                             3             47              9            130
-------------------------------------------------------------------------------
```

October, 2015



```
☁ hermes [master] cloc main server src test
     105 text files.
     105 unique files.
       0 files ignored.

http://cloc.sourceforge.net v 1.60  T=0.66 s (159.6 files/s, 26288.3 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Haskell                        105           2332           2252          12712
-------------------------------------------------------------------------------
SUM:                           105           2332           2252          12712
-------------------------------------------------------------------------------
```

hermes' challenges

Upon taking the lead on Hermes, I was asked for a couple of requirements to be fullfilled, the most important one being that the system needed to be deployed in a cluster, capable of scaling according to demand.

hermes' challenges

More specifically, we wanted a system with these desirable properties:

- ~ Scalable
- ~ Fault tolerant
- ~ Distributed

ALL I WANT IS A CLUSTER

- ~ It's easy to see that what we want is a **cluster**, capable of scaling on demand
- ~ We need to transcode videos, which is a very stateful operation
- ~ A cluster typically implies machines talking to each other, which is also very stateful
- ~ **As good Haskell programmers**, we want to have components in our system to be **as stateless as possible**, and potentially treat videos as **persistent data structures**!

A SHARED NOTHING ARCHITECTURE (SN)

A shared nothing architecture (SN) is a distributed computing architecture in which each node is independent and self-sufficient, and there is no single point of contention across the system.

”All problems in computer science can be solved by another level of indirection.” - Butler Lampson

RABBITMQ

1. RabbitMQ was just the right tool for the job at hand:
   - ∼ Easy to setup
   - ∼ Can be configured to operate in a federation of nodes
   - ∼ Extremely reliable
   - ∼ Good Haskell bindings for it (*AMQP*)
2. A question genuinely arise: it seems extremely costly to shuffle video as binary blobs over the queues. Can we avoid that?

ABSTRACTION IS THE (MEDIA KEY)

```
root__m-stg-main-2014_10_29_13_27_26-videos-1-2333-vid-smc-oxz8dmdi1lx7fong
    ^    ^   ^        ^                 ^   ^   ^   ^   ^        ^
    |    |   |        |                 |   |   |   |   |        |
comment  |   |        |                 |   |   |   |   |        |
         |   |        |                 |   |   |   |   |        |
host ----+   |        |                 |   |   |   |   |        |
             |        |                 |   |   |   |   |        |
database --- +        |                 |   |   |   |   |        |
                      |                 |   |   |   |   |        |
dataset version ------+                 |   |   |   |   |        |
                                        |   |   |   |   |        |
resource (video or image) --------------+   |   |   |   |        |
                                            |   |   |   |        |
user ID ------------------------------------+   |   |   |        |
                                                |   |   |        |
video ID -------------------------------------- +   |   |        |
                                                    |   |        |
channel type ---------------------------------------+   |        |
                                                        |        |
video products -----------------------------------------+        |
                                                                 |
MAC (avoids submission of bogus keys) ---------------------------+
```

To be fair, the media key abstraction was already present in Atlas when I choose RabbitMQ, but it was the perfect fit for it!

WHAT ABOUT DATA STORAGE?

Fine, but RabbitMQ doesn't give you data persistence...

1. We use AWS' S3 for our storing needs

   ∼ A media key **uniquely identifies** an S3 location (it's like an **IP address for videos**!)

2. Upon upload the original file from the user is synced over S3 and we call this generation-0 file the **master file**
3. Such master file is **immutable**, and each product we transcode generates a brand new binary on S3

We are treating videos as immutable data structures!

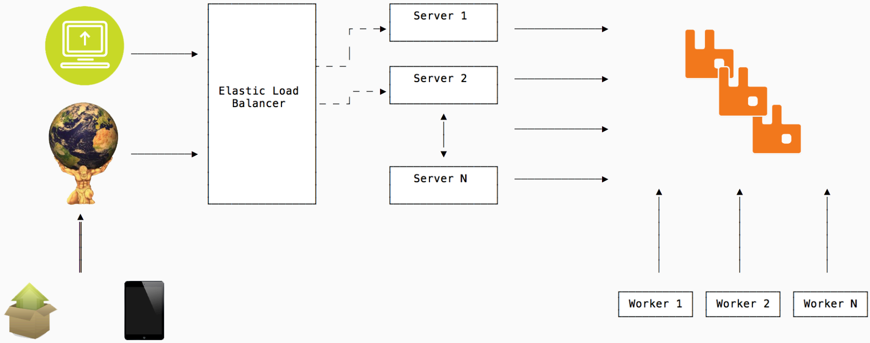
WHAT ABOUT SCALABILITY?

Fine, but RabbitMQ doesn't give you scalability...

1. We stood once again on the shoulder of giants - namely AWS' Auto Scaling Groups
2. Our very first native scaling algorith looked like:
   - ~ Scaling up: Based on CPU% over time
   - ~ Scaling down: Based on CPU% over time

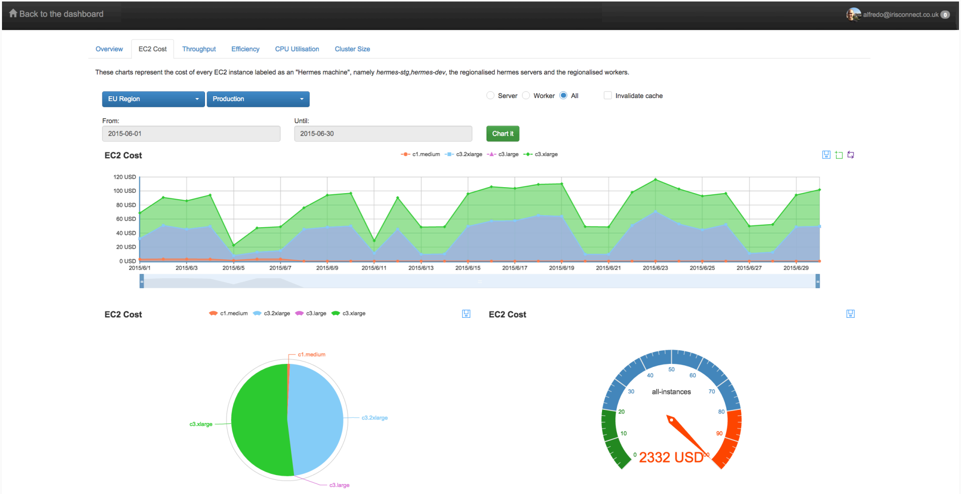
It kept us going for a while...

THE ARCHITECTURE



REVIEWING THE SCALING EXPERIENCE

1. Scaling up was too conservative and slow
   - ~ It could take up to 15 mins to spawn a new worker
2. Scaling down suffered similar problems
3. The result was unoptimal customer experience (due to the slow turnaround time) and unoptimal for us (due to the additional costs incurring from poor scaling down)



Back to the dashboard

alfredo@irisconnect.co.uk 0

Overview | EC2 Cost | Throughput | Efficiency | CPU Utilisation | Cluster Size

These charts represent the cost of every EC2 instance labeled as an "Hermes machine", namely *hermes-stg,hermes-dev*, the regionalised hermes servers and the regionalised workers.

EU Region ▾ | Production ▾

○ Server ○ Worker ● All □ Invalidate cache

From: 2015-06-01

Until: 2015-06-30

Chart it

What's the elephant in the room?

Why not use Cloud Haskell?

WHY NOT CLOUD HASKELL

1. CH encourages Erlang-style (i.e. actor based) communication, so nodes should know each other

   We do not want that!

2. Peer discovery would have been tricky in a dynamic environment where new machines born and die frequently
3. It wasn't mature enough in 2013, if not for a handful of companies using it

Thank you!

Questions?

EXTERNAL REFERENCES

My road to Haskell

http://www.alfredodinapoli.com/posts/2014-04-27-my-road-to-haskell.html

Don Stewart - Haskell in the large

http://code.haskell.org/~dons/talks/dons-google-2015-01-27.pdf