
Debugging with Livebook
An stroll into introspection
ElixirConf EU
20-21 April 2023 | Lisbon/virtual

Who am I

- Luca Dei Zotti - aka `zoten`
- 💻 Engineering Manager @ Prima/HelloPrima
  - Also, sponsor for this travel and multiple time supporter of Code BEAM and other conferences, so please show some love if you please 🙏
- 👶 Happy husband and father of one :)
- 🥋 Martial Arts and Combat Sports practictioner, agonist, teacher
- 🎷 Spare-time musician
- 📚 Slow but - hopefully - endless learner
- zoten @zoten_deschain lucadeizotti

prima

The Problem

The problem is *simple*: world is *complex*.

Software development is hard.

Modern software development is complex *and* hard. And messy. And we don’t help a lot with this.

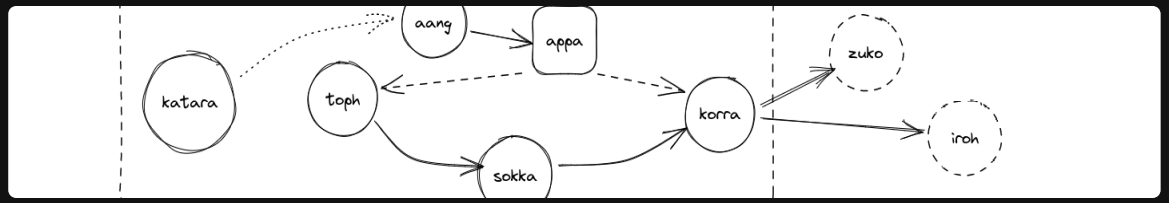
You often end up in one between some standard situations.

Deployments

On Premise Software

Simple SaaS

Microservices



They have something in common

They have something in common

(save it for later)

And then?

The main favour you can do to yourself when architecting your software is **instrumenting observability** tools.

You, yourself of the future and a lot of operation saints and on-call people will thank you for the ability to introspect the status of your system from multiple points of view.

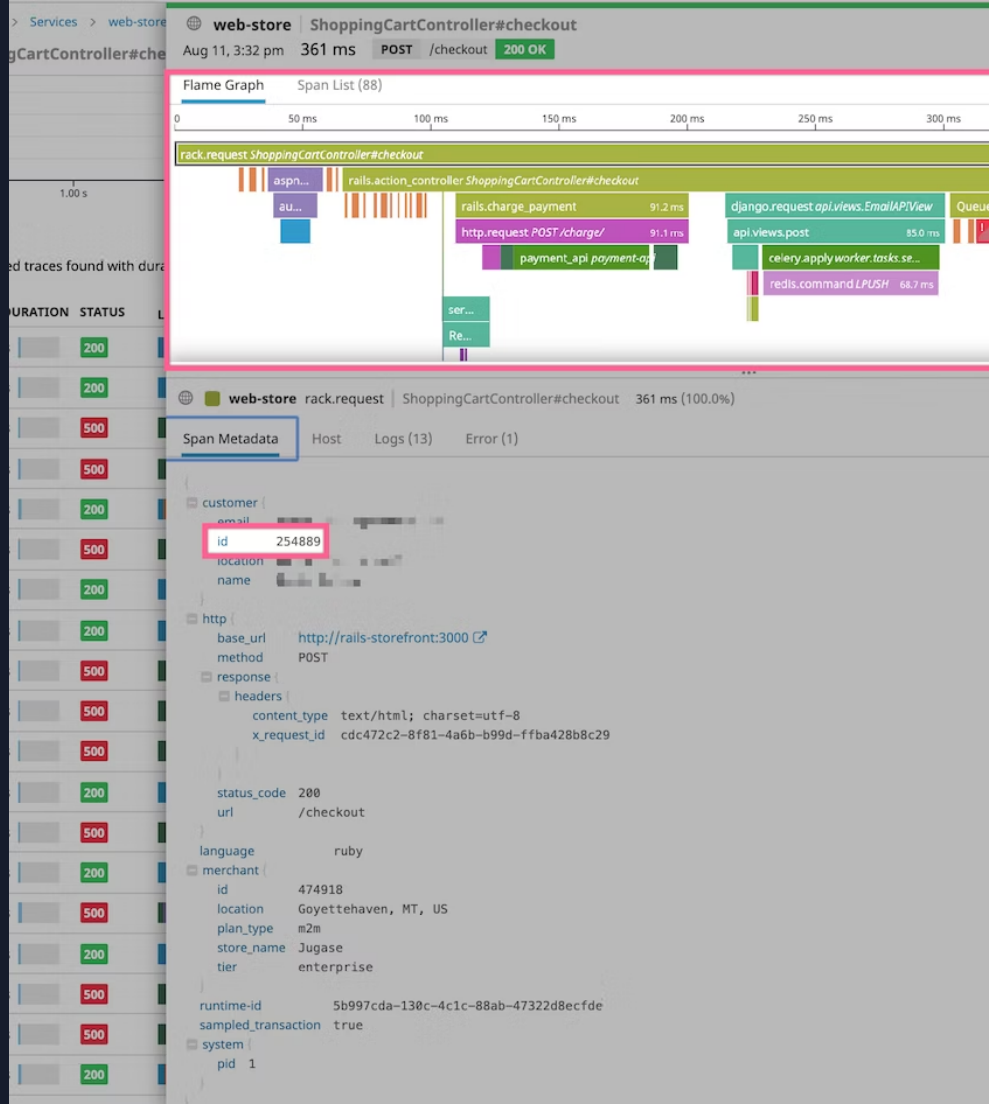
There are a lot of tools, nowadays, to integrate observability into our infrastructure.

(not related to any of them, just took a list from a 3s search)

And Then? (2)

Through observability you gain a lot of sight into what's happening [1].

Most of the libraries and tools in the ecosystem are already designed to include telemetry, and to integrate with observability standards like OpenTelemetry.



1. image courtesy of Datadog documentation

BUT

BUT

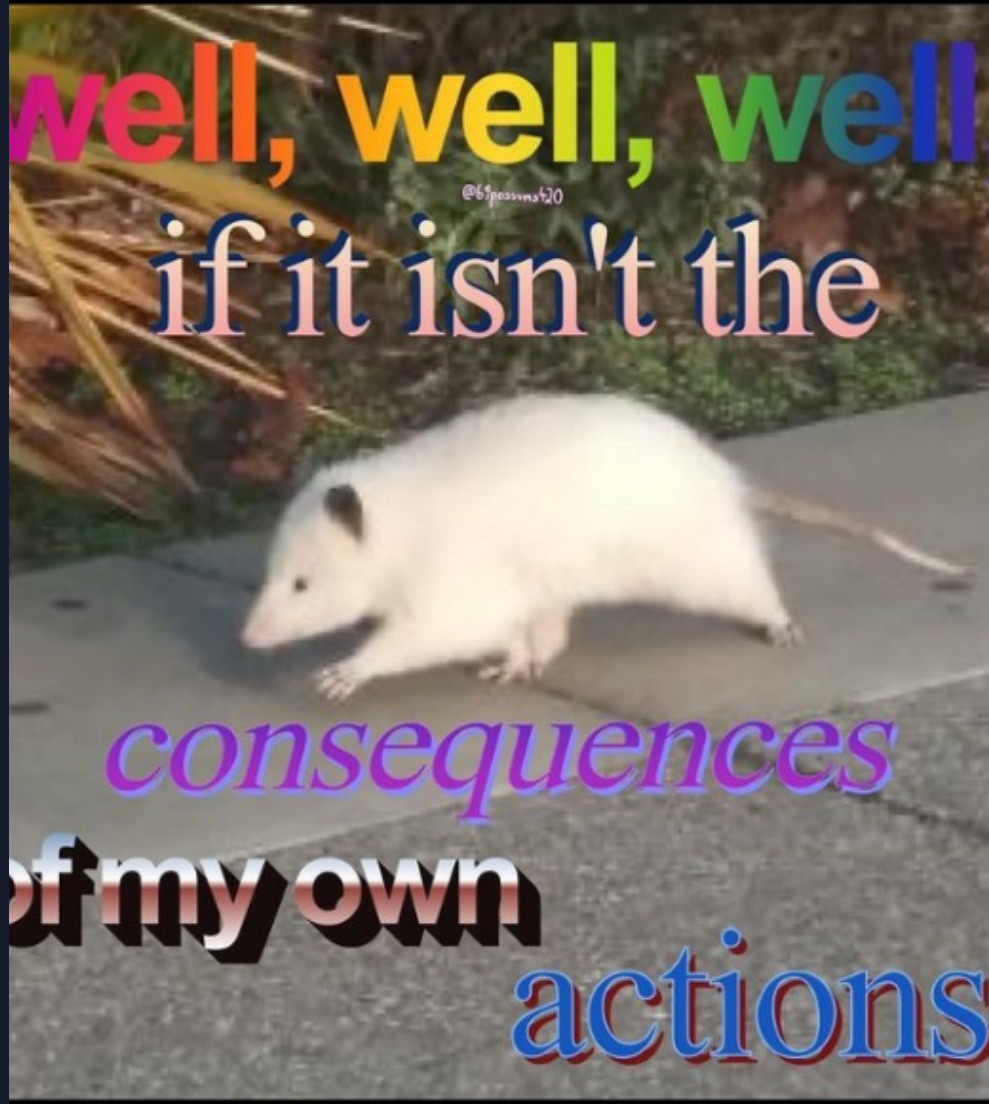
We are the ones who write the core business logic, and ...

...we make mistakes

- we forget or fail or didn't think was important to instrument observability at certain points
- we decided or have been forced **not** to use some framework, library or tool that was already instrumented
- we decided or have been forced **to** use some framework, library or tool that wasn't instrumented at all
- ...we simply *got it wrong*!

Now What?

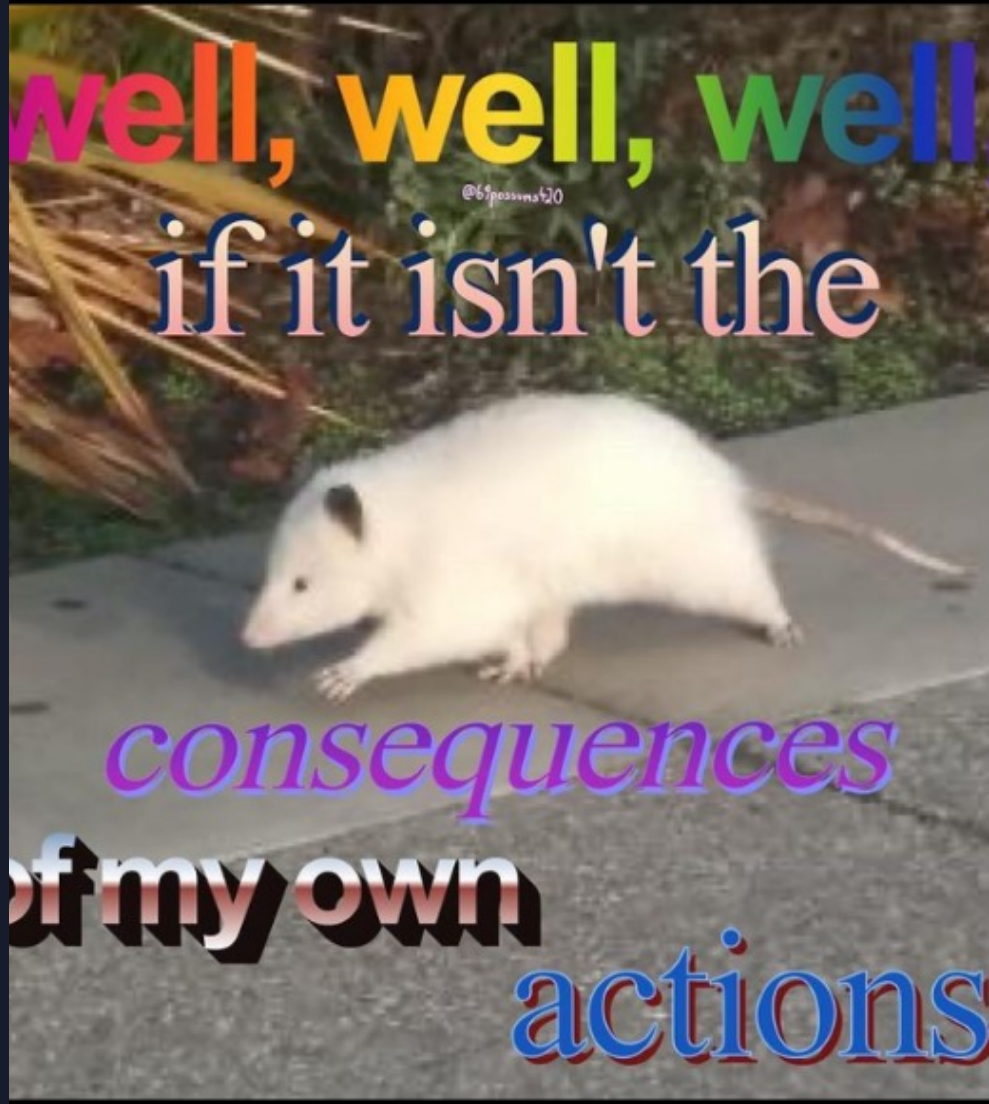
Are we left alone in the dark?



Now What?

Are we left alone in the dark?

> Remember our deployments had something in common?

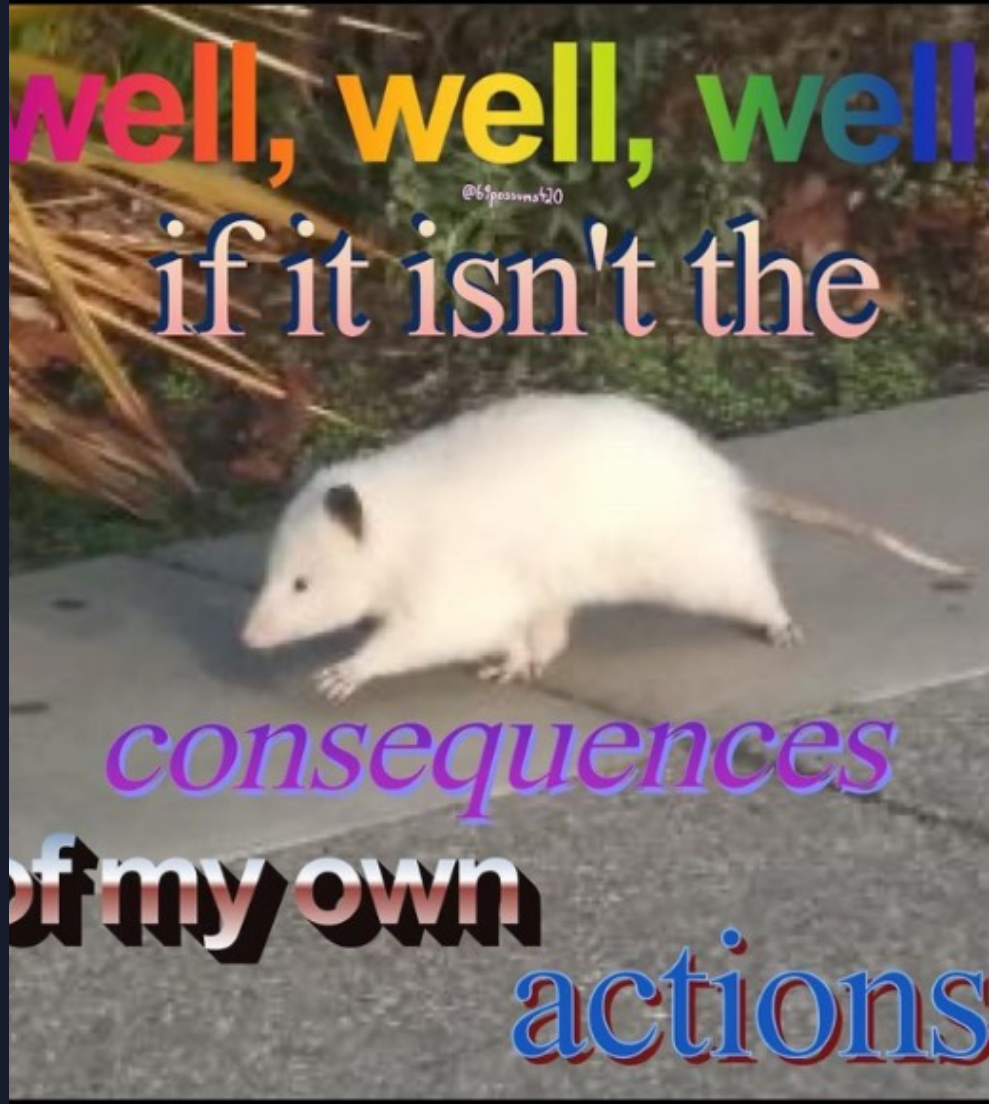


Now What?

Are we left alone in the dark?

> Remember our deployments had something in common?

They are `reachable`!



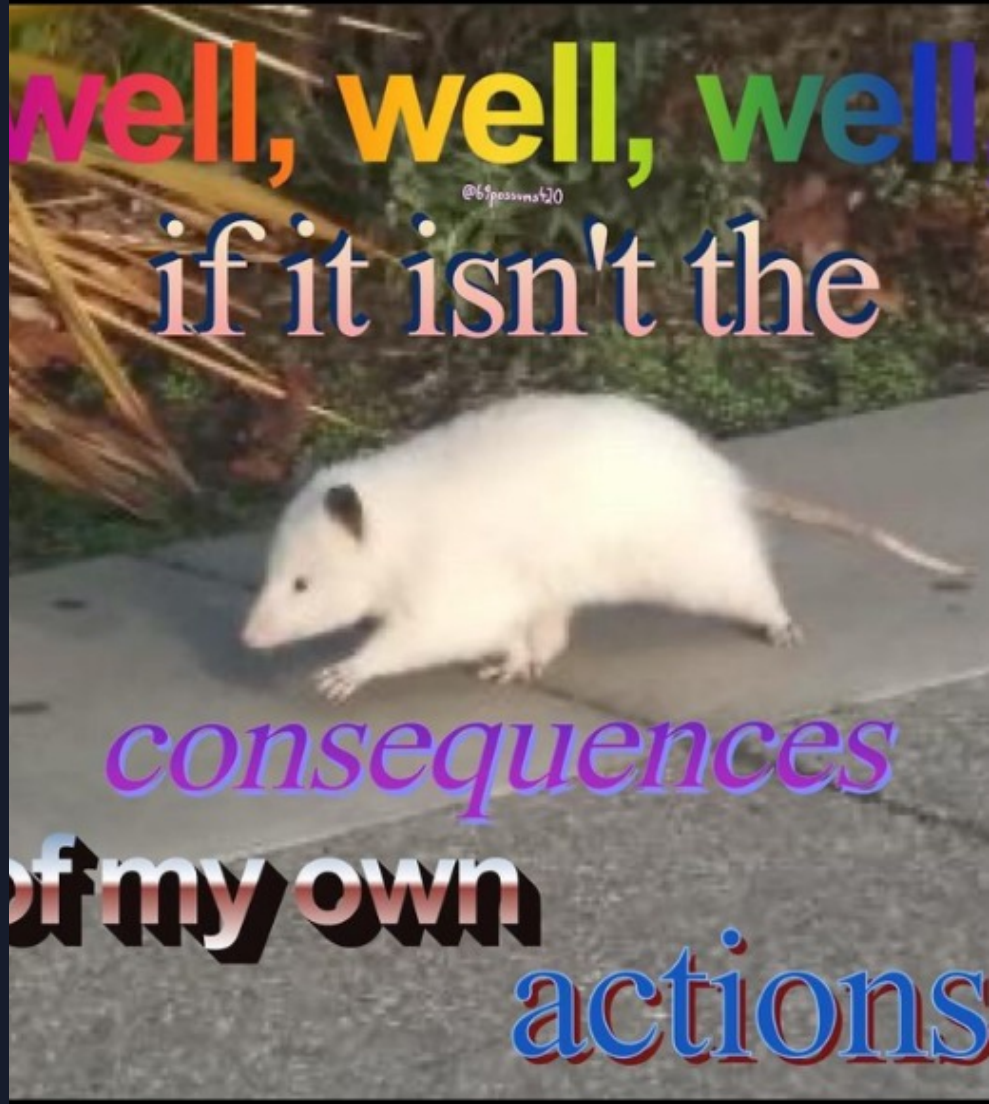
Now What?

Are we left alone in the dark?

> Remember our deployments had something in common?

They are `reachable`!

(well, if our platops approve)



Now What?

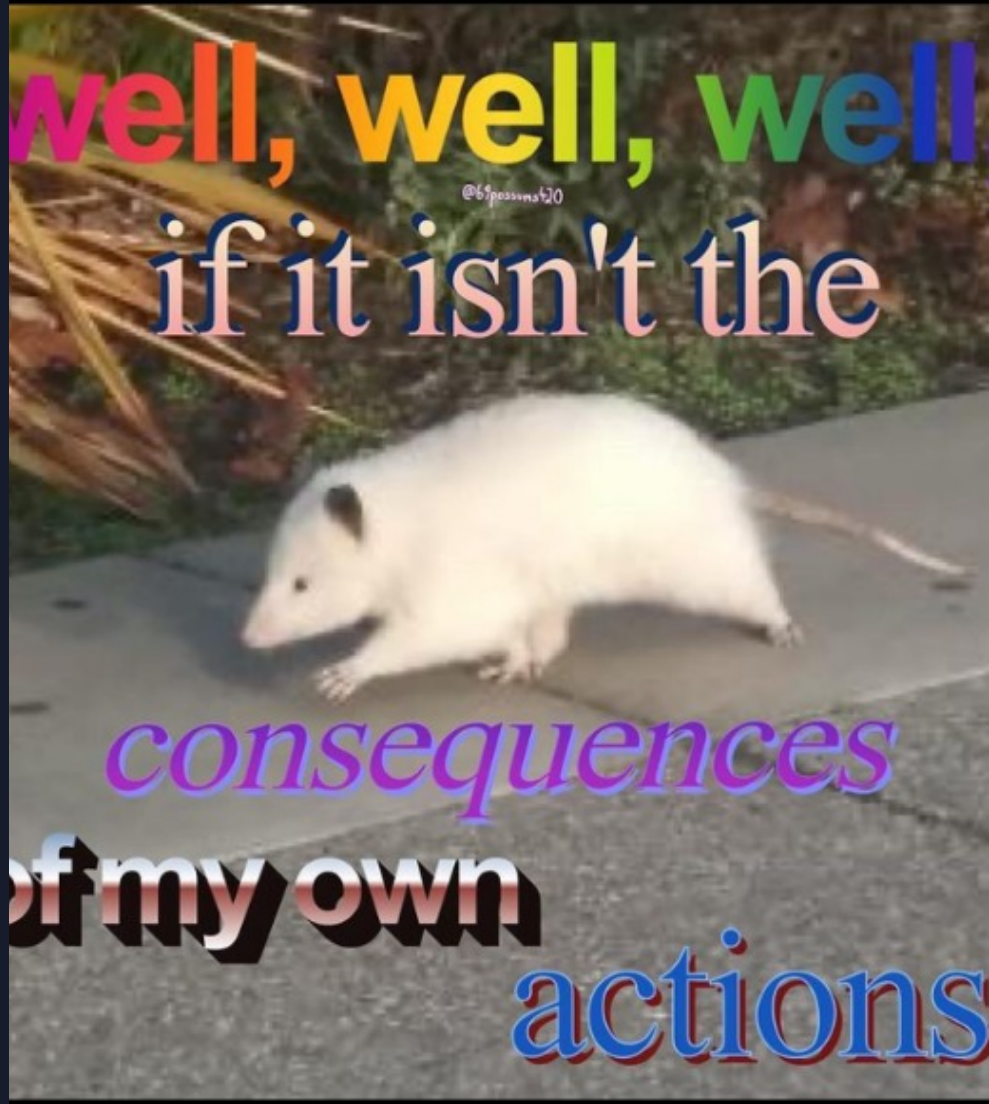
Are we left alone in the dark?

> Remember our deployments had something in common?

They are `reachable`!

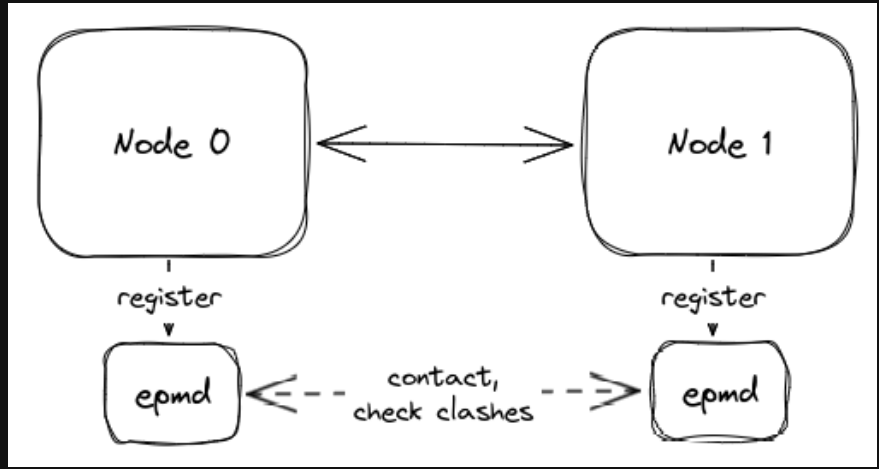
(well, if our platops approve)

...OTP to the rescue!



Nodes

- Erlang/OTP tries to give basic building blocks of distribution
  - distribution is hard, there's no *one fits all* solution
- An Erlang(/Elixir) Node is an instance of the Erlang VM (BEAM)
- A Node is assigned a name that must be unique in a *cluster*
- The most common *name server* for OTP is a process called *epmd* [1]
  - maps symbolic node names (`node@host`) to machine addresses
  - started automatically by `erl` (unless already started or differently specified)
  - multiple existing implementations
- [2]

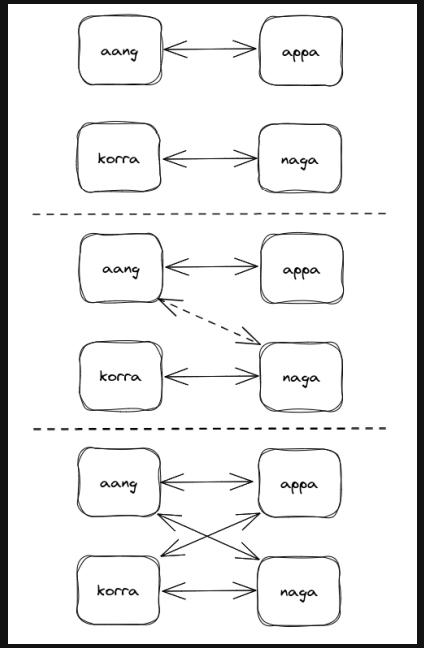


---

1. See more at Erlang Documentation - epmd
2. A lot more at Learn You Some Erlang for Greater Good - distribunomicon

Mesh

- When two nodes connect, they start monitoring each other
- When a node connects to another that is already part of a group, that node connects to the entire group
  - every node still keeps its own set of loaded modules, applications, supervision trees etc
- The good: everyone gets to talk to everyone. Super fault tolerant!
- The bad: connections and chatters grows exponentially with the number of nodes



Names

Erlang gives names to each of the nodes to be able to locate and contact them.

Names are in the form `Name@Host`, where the *host* is based on available *DNS* entries, either over the network or in your computer's host files (e.g. `/etc/hosts` in Unix-likes).

```erlang
net_kernel:connect_node(NodeName). % > true
```

```elixir
:net_kernel.connect_node(nodename) # > true
Node.connect(nodename)             # > true

```

```erlang
net_adm:ping(NodeName). % > Pong
```

```elixir
:net_adm.ping(nodename) # > :pong
```

```erlang
erlang:disconnect_node(NodeName). % > true
```

```elixir
:erlang.disconnect_node(nodename) # > true
```

```erlang
erlang:monitor_node(NodeName, true). % > true
```

```elixir
:erlang.monitor_node(nodename, true) # > true
```

name vs sname

- Long names are based on fully qualified domain names (`aaa.bbb.ccc`), and many DNS resolvers consider a domain name to be fully qualified if they have a period (`.`) inside of it.
- Short names are based on host names without a period, and are resolved going through your host file or through any available *DNS* entry.

Because of this, it is generally easier to set up a bunch of Erlang nodes together on a single computer using *short names* than *long names*. One last thing: because names need to be unique, nodes with short names cannot communicate with nodes that have long names, and the opposite is also true.

Note that you can also start nodes with only the names:

```
1   erl -sname short_name
```

```
1   erl -name long_name
```

Erlang will automatically attribute a host name based on your operating system's configuration. You also have the option of assigning a chosen IP starting a node with a name such as

```
1   erl -name name@127.0.0.1
```

Connection

Cookies are a mechanism used to divide clusters of nodes in a same machine, not an authentication mechanism.

```
erl -sname ratchet@ratchet0 -setcookie 'supersecretcookie'
iex --sname ratchet@ratchet0 --setcookie 'supersecretcookie'
```

```
erlang:set_cookie(node(), ultrasecretcookie).
```

```
:erlang.set_cookie(node(), :ultrasecretcookie).
```

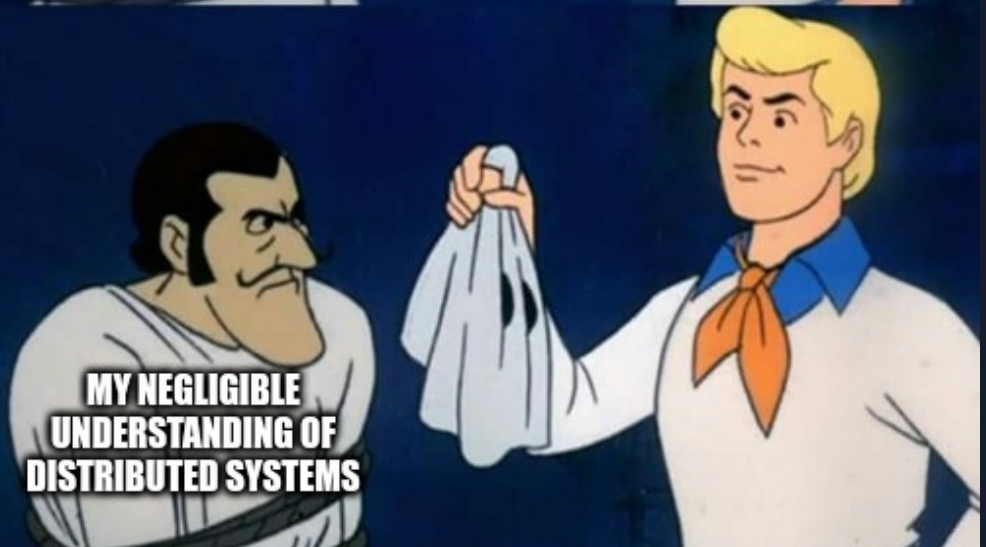
Hidden nodes

Sometimes you want to just connect to a single node in a cluster and poke around a couple of things, without alarming anyone

```
erl -sname ninja -hidden
```

```
iex --sname ninja --hidden
```

```
Node.list()           # will not show `ninja` in any node
Node.list(:hidden)    # will show `ninja` on connected node, and connected node in `ninja`
Node.list(:connected) # will show all nodes from the calling node's point of view
Node.list(:known)     # will show also past nodes
```



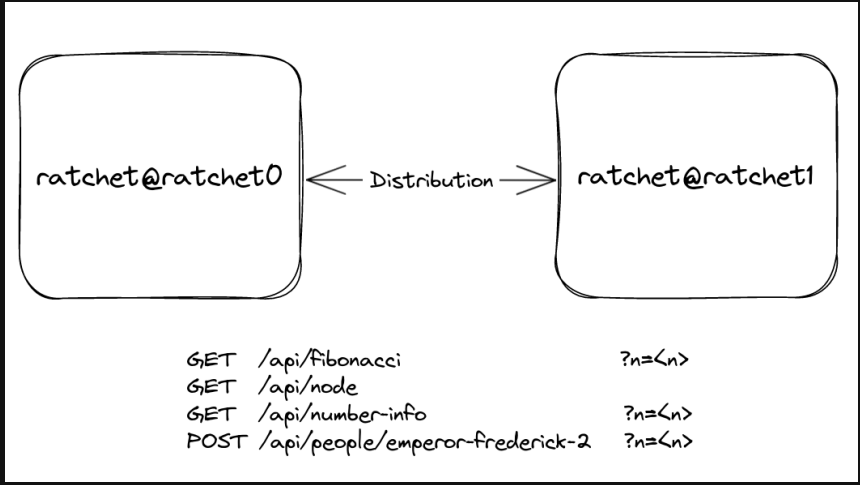
Don’t drink too much Kool-Aid (cit.)

This is not a talk about distributed systems, but remember:

- Bandwith is not infinite
- Latency exists
- Network can fail
- Network is insecure
- CAP theorem

Demo Architecture

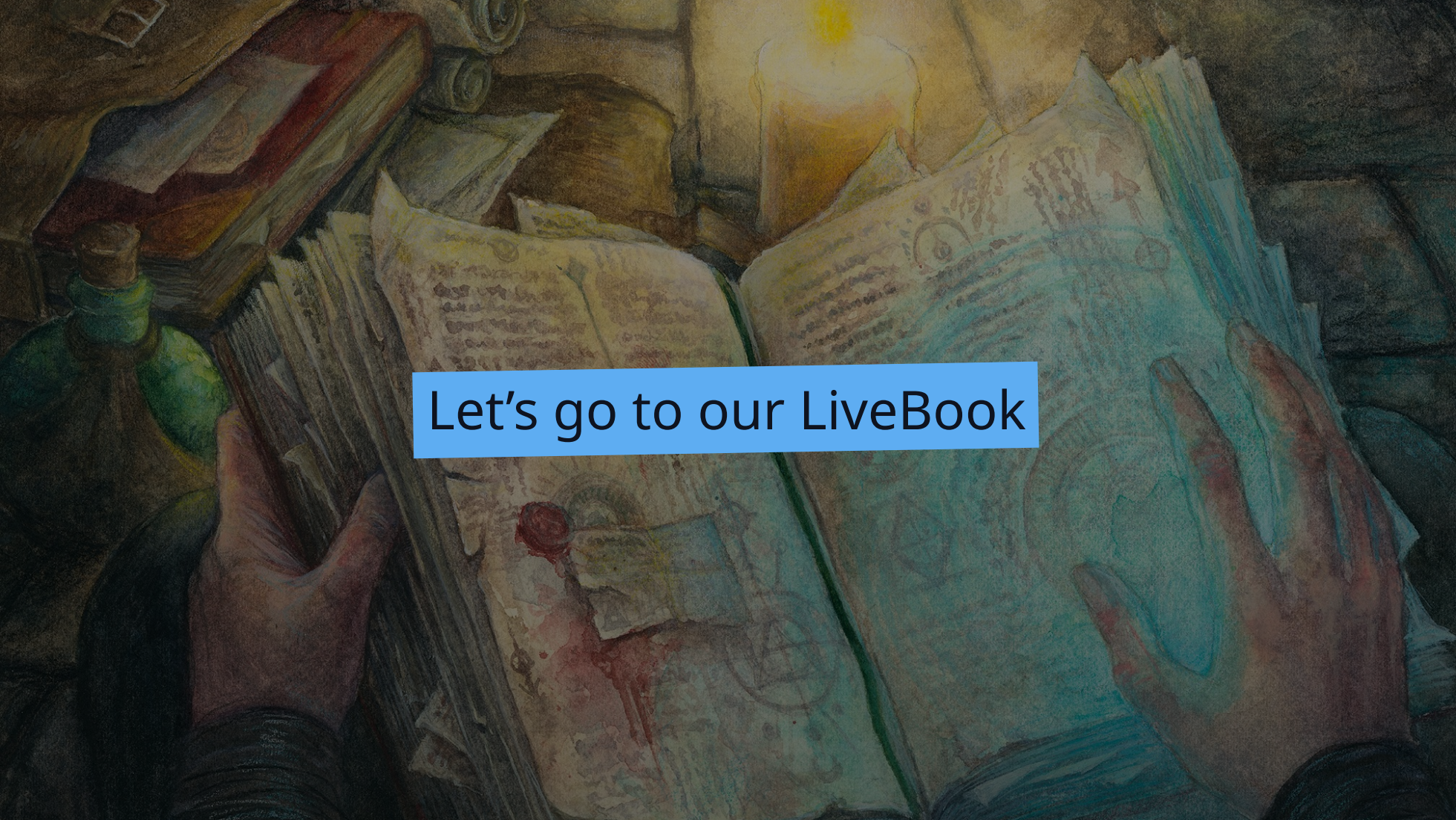
Application



A pretty standard web application, running on two nodes (via docker-compose in this example)

Ratchet

English translation of the Italian translation (Cricchetto) of the Tow Mater's original English name, because of reasons


Let’s go to our LiveBook

Takeaways

- LiveBook can be useful as a graphical tool also for debugging purposes
  - versionable
  - more handy than a series of copy-pasted shell commands
  - repeatable
  - graphical!
- Debugging live systems is fun
  - ...and *complex*
  - ...and ~~will~~*can* break things
  - ...and ~~will~~*can* throw you in a black hole of despair
  - ...and breaks the `` `immutable software rule` ``
  - ...
  - ...but can save your business. Sometimes, more than that!

Sources

Code & slides available at https://github.com/zoten/eceu2023

Gotchas

- Decided to use a modified version of eflambe to embed the SVG header instead of loading a specific asset
- Used a dependency-updated version of eflambe_live just for the sake of compatibility


Thank you!