

Rails -> Phoenix

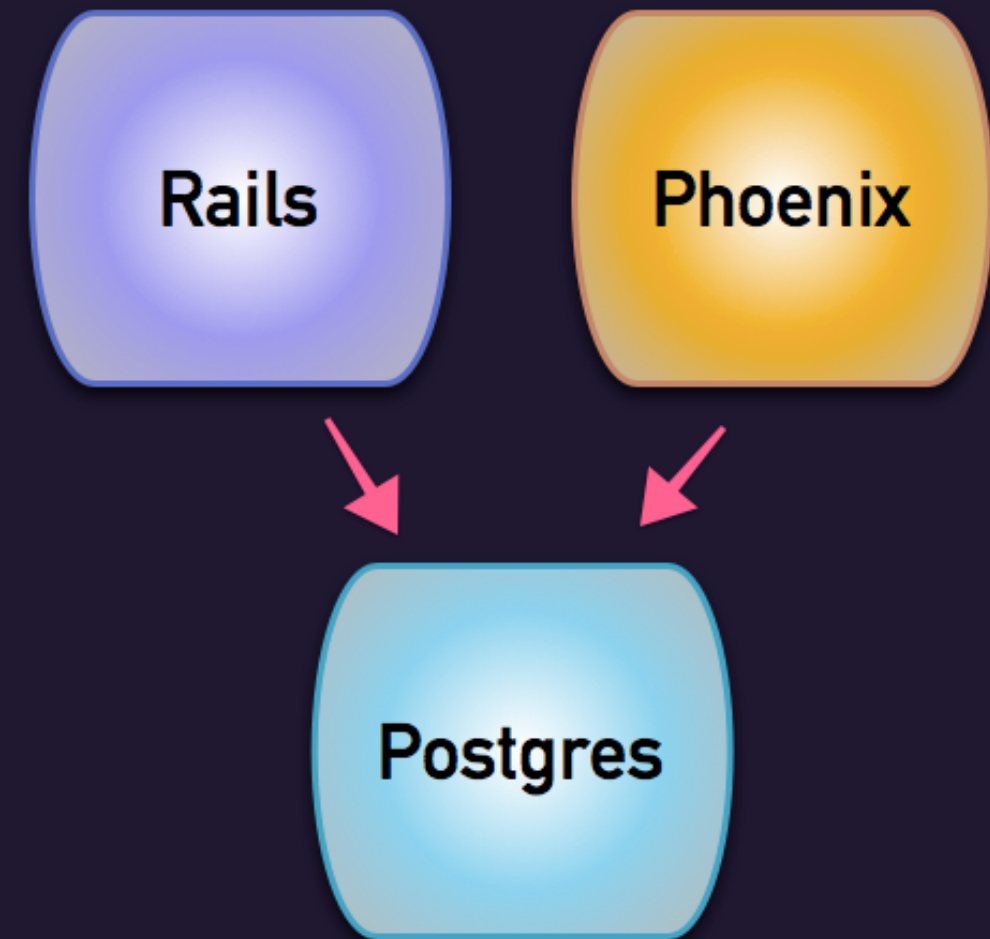
Retour d'expérience sur une
rewrite (en cours) d'application SaaS.



Pourquoi une rewrite?

- Elixir flaggé "stratégique" sur mon radar tech 😎
- Produits SaaS et applications internes (dev solo)
- Consulting (scalabilité, ETL, ...)
- Robotique, IoT, ETL, Streaming.
- Rewrite = apprentissage & comparaison
- Comparaison des écosystèmes

Rewrite "side-by-side"



Comment débiter

→ "Programming Elixir" de A à Z

→ Koans (<http://github.com/thbar/elixir-playground>)

```
test "function declaration and invocation" do
  sum = fn (a, b) -> a + b end
  assert sum.(2, 10) == 12

  tuple_sum = fn { a, b, c, d } -> a + b + c + d end
  assert tuple_sum.({ 10, 100, 1000, 10000 }) == 11110
end
```


Frictions

- Vendor lock-in Ruby:
 - Productivité très forte.
 - Ecosystème très mûr et large.
 - Zone de confort énorme.
- Accepter d'être lent au début.

Motivations

- Faire plus avec moins de personnes.
- Scaler plus fort avec moins de machines.
- Aller vers l'internet des choses, le streaming, le flux continu de données, le très scalable, le redondant.
- Même "bon feeling" qu'avec Rails en 2005.

T.B.D.D.D.D.D

Tests & base de données driven-development

Exporter une base de dév depuis Rails

```
$ rake db:seed
```

```
$ pg_dump -0 -x wisecash_dev -f legacy-dumps/rails-db-seed.sql
```

Importer la base Rails vers Ecto

```
$ ecto.load --dump-path legacy-dumps/rails-db-seed.sql
```

Importer la base Rails vers Ecto, automatiquement, avant les tests

```
defp aliases do
  [
    "test": [
      "ecto.create --quiet",
      "ecto.load --dump-path legacy-dumps/rails-db-seed.sql",
      "test"
    ]
  ]
end
```

Continuous integration ♥

(quelques heures pour la mise en place initiale sur CircleCI)

Se mettre à l'aise pour expérimenter

- Feedback-loop rapide avec `mix_test_watch`
- Tests "exploratoires"
- Colorisation de l'output avec `apex`

```
defp deps do
  [
    {:mix_test_watch, "~> 0.2", only: :dev},
    {:apex, "~> 0.4.0", only: [:dev, :test]}
  ]
end
```

Zoomer sur un problème à la fois

```
ExUnit.configure(  
  exclude: :test,  
  include: :focus,  
  ...  
)
```

```
@tag :focus  
def test_this do  
  # snip  
end
```


Débugger une dépendance facilement

```
$ atom deps/openmaize_jwt/lib
```

```
config :mix_test_watch,  
  [  
    "deps.compile openmaize_jwt",  
    "test"  
  ]
```

Adaptations Ecto <-> ActiveRecord

```
schema "entries" do
  timestamps inserted_at: :created_at
end
```

Evènements récurrents

Plusieurs tentatives:

1. Postgres `generate_series` (ne colle pas au besoin) 😞
2. Fonction PL/pgSQL (des contournements à faire) 😞
3. Fonction Elixir (terminée en quelques heures) 😎

Merci `Timex.shift`

Queries SQL sur des fonctions (avec Ecto 2)

```
Logger.configure(level: :debug)
```

```
Ecto.Adapters.SQL.query!(WisecashEx.Repo, query, [from, to])
```

```
def postgres_setup!(repo) do
  source = File.read!(query_file("generate_recurring_events.sql"))
  Ecto.Adapters.SQL.query!(repo, source, [])
end
```

→ Ecto = plus limité que Sequel en Ruby

Authentication

Choix large (comme avec Rails en 2005):

- Guardian (pas de gestion de BDD, JWT)
- Openmaize (BDD, code généré, JWT) 👍
- Addict
- ...

Mais quelle solution sera réellement pérenne ???

Adaptation du schéma de Rails & Devise vers Openmaize

```
schema "users" do
  field :email, :string
  field :encrypted_password, :string
  field :password, :string, virtual: true
  field :role, :string, virtual: true, default: "user"
  timestamps inserted_at: :created_at
end
```

```
config :openmaize, hash_name: :encrypted_password
```


Openmaize dans le contrôleur

```
plug Openmaize.Login, [  
  db_module: WisecashEx.OpenmaizeEcto,  
  unique_id: :email  
] when action in [:login_user]
```

Reprise en main du code généré par Openmaize.

Tests d'acceptance (hound + phantomjs ♥)

```
navigate_to("/")  
"/" = current_path
```

```
find_element(:link_text, "Login")  
|> click  
"/login" = current_path
```

```
find_element(:id, "user_email")  
|> fill_field("john@example.com")  
`
```

```
find_element(:id, "user_password")  
|> fill_field("testtest")
```

Tests d'acceptance (hound + phantomjs ♥)

```
find_element(:tag, "form")  
|> submit_element
```

```
assert visible_page_text =~ ~r/Logged as john@example.com/i
```

```
find_element(:link_text, "Logout")  
|> click
```

```
assert visible_page_text =~ ~r/You have been logged out/i
```

Déploiement

- Heroku: build-pack fonctionnel rapidement
- EngineYard: custom cookbooks, pas horrible
- Pas de hot-reloading pour le moment

Mes conclusions pour le moment

- Données immutables = fondation solide
- Effets de 2nd ordre (levier, composabilité)
- TDD = aide beaucoup pour prendre en main
- Librairies jeunes (perte de temps - maintenance)
- Confort très similaire à Ruby en régime de croisière
- Moins de "compromis duct-tape" qu'avec Ruby