Reborn the web development with Phoenix

cabify

self()

Miguel Garcia a.k.a Rock Neurotiko

Antiguo alumno de la FI UPM

Vieja guardia de ACM

Senior Backend Developer @ Cabify

I ❤ Programming ❤ Cars ❤ Cats ❤

cabify

About Cabify

Change the cities transport models

Provide alternatives for the private vehicle

We build the future in Go and Elixir

We’re hiring :)

cabify

Erlang & Elixir

cabify

Key concepts

Concurrencia + IVA

Distributed

Soft real-time

Extreme reliability

Hot swapping

cabify

Usages

Telecom, GPRS/3G/5G
Message broker
Distributed databases
Webservers
Data pipelines
Finance and blockchain
Real time apps
Embedded
...

Concurrency model

Lightweight concurrency

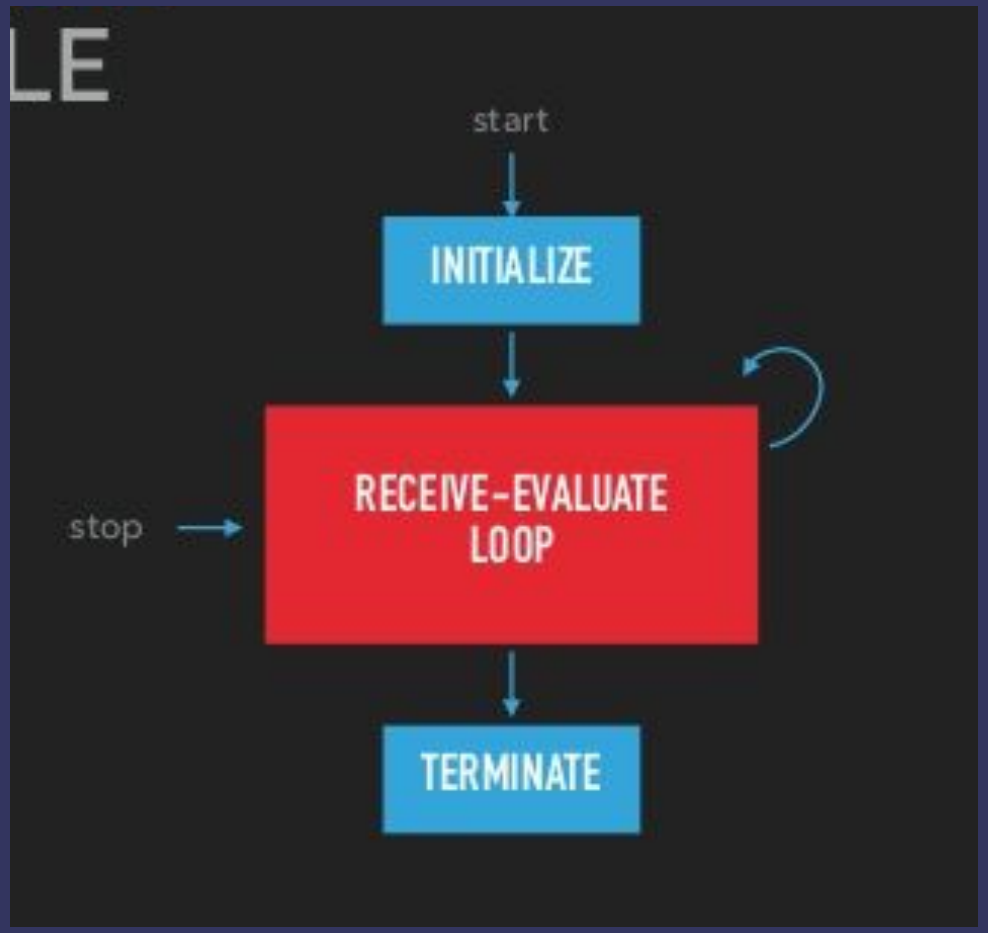
Asynchronous communication

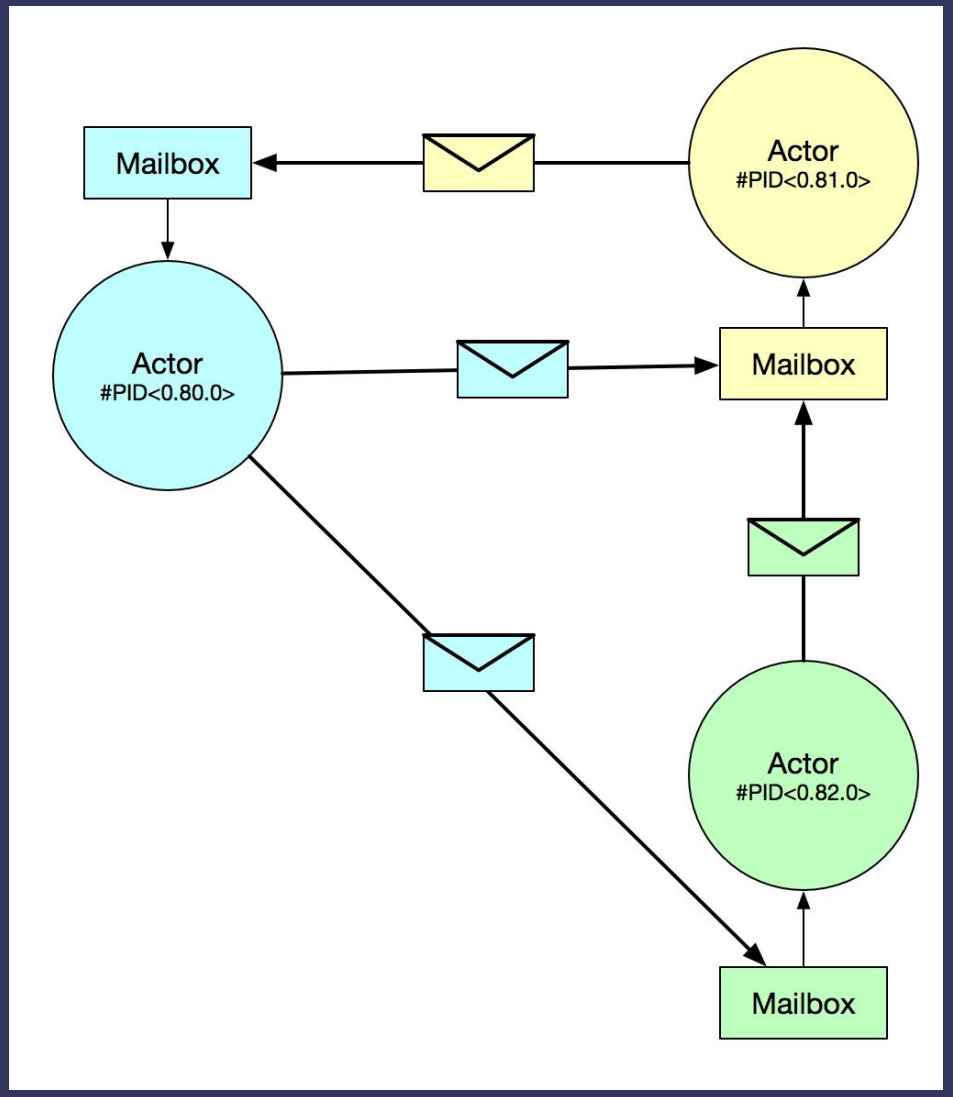
Error handling

Isolated processes

Message passing

Distributed Actor Model





cabify

Example in raw Elixir

```elixir
defmodule MyStateActor do
  def loop, do: loop(%{})

  def loop(state) do
    receive do
      {from, {:set, key, value}} ->
        send(from, {self(), :ok})
        loop(Map.put(state, key, value))
      {from, {:get, key}} ->
        send(from, {self(), Map.get(state, key)})
        loop(state)
    end
  end
end

```


cabify

Example in raw Elixir Usage

```
iex()> pid = spawn(&MyStateActor.loop/0)
#PID<0.123.0>
iex()> send(pid, {self(), {:get, :key1}})
{#PID<0.104.0>, {:get, :key1}}
iex()> flush()
{#PID<0.123.0>, nil}
:ok
iex()> send(pid, {self(), {:set, :key1, "Random value"}})
{#PID<0.104.0>, {:set, :key1, "Random value"}}
iex()> flush()
{#PID<0.123.0>, :ok}
:ok
iex()> send(pid, {self(), {:get, :key1}})
{#PID<0.104.0>, {:get, :key1}}
iex()> flush()
{#PID<0.123.0>, "Random value"}
:ok
```


cabify

OTP since 1998

Basic building block
    Design, distribute, configure and reuse code

Supervisor/DynamicSupervisor

Workers: GenServer, Agent, Task, GenStage, ...

cabify

Example with OTP

```elixir
defmodule MyStateActor do
  use GenServer

  def init(_), do: {:ok, %{}}

  def handle_call({:set, key, value}, _from, state) do
    {:reply, :ok, Map.put(state, key, value)}
  end

  def handle_call({:get, key}, _from, state) do
    {:reply, Map.get(state, key), state}
  end
end


```


cabify

Example with OTP Usage

```
1 iex(11)> {:ok, pid} = GenServer.start_link(MyStateActor, :ok)
2 {:ok, #PID<0.149.0>}
3 iex(12)> GenServer.call(pid, {:get, :key1})
4 nil
5 iex(13)> GenServer.call(pid, {:set, :key1, "Random value 2"})
6 :ok
7 iex(14)> GenServer.call(pid, {:get, :key1})
8 "Random value 2"
```

Why Elixir

Runs on Erlang/OTP

FP Language

Extensible design with macros, protocols, behaviours

Easy to learn, explicit

First class documentation

Why Elixir

Excellent tooling (mix, dialyzer, hex, credo, ...)

Great testing support (ex_unit, mox, stream_data, ...)

Phoenix

Great community

More and more companies using it

Phoenix Framework

cabify

Framework web

Standardize how to build a web

Database

Templates

URL mapping

Security

Good abstractions

Phoenix flow

```
1 connection
2 |> endpoint
3 |> router
4 |> pipelines
5 |> controller
6 |> model
7 |> view
8
```


cabify

Setup

```
mix archive.install hex phx_new 1.4.0

mix phx.new todo --no-ecto

cd todo

mix deps.get

mix compile

iex -S mix phx.server
```

Setup

github.com/rockneurotiko/tryit19-phoenix

tiny.cc/tryit19-rock

cabify

Domain- TODO

```elixir
defmodule Todo.Todo do
  alias Todo.Reference

  @type t :: %__MODULE__{
          id: String.t(),
          title: String.t(),
          finished: boolean,
          board: String.t()
        }

  defstruct [:id, :title, :board, finished: false]

  def new(data) when is_map(data) do
    id = Reference.gen_reference()
    data = Map.put(data, :id, id)
    struct(__MODULE__, data)
  end
end
```


cabify

Domain - Store - Part 1

```elixir
defmodule Todo.Todo.Store do
  use GenServer

  @impl true
  def init(_) do
    {:ok, %{todos: %{}}}
  end
end
```


cabify

Domain - Store - Part 2

```elixir
def handle_call({:save, todo}, _, state) do
  todos = Map.put(state.todos, todo.id, todo)
  {:reply, :ok, %{state | todos: todos}}
end

def handle_call({:get_todo, board_id, todo_id}, _, state) do
  result = fetch_todo(state, board_id, todo_id)
  {:reply, result, state}
end

def handle_call({:get_board, board_id}, _, state) do
  todos =
    state.todos
    |> Map.values()
    |> Stream.filter(&(&1.board == board_id))
  |> Enum.sort_by(&todo_sort/1)

  {:reply, todos, state}
end
```


cabify

Domain - Store - Part 3

```elixir
defp todo_sort(todo) do
  d = todo.created_at
  {d.year, d.month, d.day, d.hour, d.minute, d.second}
end

defp fetch_todo(state, board_id, todo_id) do
  with {:ok, todo} <- Map.fetch(state.todos, todo_id),
       true <- todo.board == board_id do
    {:ok, todo}
  else
    _ -> {:error, :not_found}
  end
end
```


cabify

Domain - Store - Part 4

```elixir
alias Todo.Todo

@name __MODULE__

def start_link(opts) do
  opts = Keyword.put_new(opts, :name, @name)
  GenServer.start_link(__MODULE__, :ok, opts)
end

# init
```


cabify

Domain - Store - Part 5

```elixir
alias Todo.Todo

@name __MODULE__

def start_link(opts) do
  opts = Keyword.put_new(opts, :name, @name)
  GenServer.start_link(__MODULE__, :ok, opts)
end

# init
```


cabify

Domain - Store - Part 6

```elixir
def save(%Todo{} = todo, pid \\ @name) do
  GenServer.call(pid, {:save, todo})
end

def get(board_id, todo_id, pid \\ @name) do
  GenServer.call(pid, {:get_todo, board_id, todo_id})
end

def board(board_id, pid \\ @name) do
  GenServer.call(pid, {:get_board, board_id})
end

# handle_call
```


cabify

Domain - Store - Part 7

```
1 children = [
2   TodoWeb.Endpoint,
3   Todo.Todo.Store
4 ]
5
6 opts = [strategy: :one_for_one, name: Todo.Supervisor]
7 Supervisor.start_link(children, opts)
8
```


cabify

Domain - Usage

```
iex()> todo = Todo.Todo.new(%{title: "My TODO", board: "123"})
%Todo.Todo{board: "123", finished: false, id: "4ORQDT", title: "My TODO"}

iex()> Todo.Todo.Store.save(todo)
:ok
iex()> Todo.Todo.Store.board("123")
[%Todo.Todo{board: "123", finished: false, id: "4ORQDT", title: "My TODO"}]

iex()> todo = %{todo | finished: true}
%Todo.Todo{board: "123", finished: true, id: "4ORQDT", title: "My TODO"}

iex()> Todo.Todo.Store.save(todo)
:ok
iex()> Todo.Todo.Store.board("123")
[%Todo.Todo{board: "123", finished: true, id: "4ORQDT", title: "My TODO"}]
```


cabify

Plug

Core building block of Phoenix

Endpoints, Routers and Controllers are Plugs

Plug unify Request & Response

Plug basically is:

(Plug.Conn, params) -> Plug.Conn

Plug - Method

```elixir
plug :accepts, ["html"]
plug :put_headers, %{content_encoding: "gzip"}

defp put_headers(conn, headers) do
  Enum.reduce(headers, conn, fn {key, value} ->
    Plug.Conn.put_req_header(conn, key, value)
  end)
end
```


cabify

Plug - Module

```elixir
plug PutHeaders, %{content_encoding: "gzip"}

defmodule PutHeaders do
  def init(headers), do: headers

  def call(conn, headers) do
    Enum.reduce(headers, conn, fn {key, value} ->
      Plug.Conn.put_req_header(conn, key, value)
    end)
  end
end
```


cabify

Pipelines

Group plugs to apply many in sequence and selectively

```elixir
pipeline :browser do
  plug :accepts, ["html"]
  plug :fetch_session
end

pipeline :api do
  plug :accepts, ["json"]
end

scope "/api" do
  pipe_through [:api]
end

scope "/" do
  pipe_through [:browser]
end
```


cabify

Endpoint

The entry for every connection

Wrapper for start and stop endpoint's server

Initial plug pipelines (static files, request id, ...)

Web configuration for your app (port, ssl, ...)

Router

DSL for creating your custom routes

Apply pipelines and dispatch to controllers

cabify

Router - Pipelines

```elixir
pipeline :browser do
  plug :accepts, ["html"]
  plug :fetch_session
  plug :fetch_flash
  plug :protect_from_forgery
  plug :put_secure_browser_headers
end

pipeline :api do
  plug :accepts, ["json"]
end
```


cabify

Router - Routes

```elixir
scope "/", TodoWeb do
  pipe_through :browser

  get "/", TodoController, :empty
  get "/:board_id", TodoController, :board
end

scope "/api", TodoWeb do
  pipe_through :api
  get "/board/:board_id", TodoApiController, :get_board
  post "/board/:board_id/task", TodoApiController, :create_task
  post "/board/:board_id/task/:task_id/toggle", TodoApiController, :toggle_task
end
```


cabify

Controller

Defines how the specific route has to be handled

On the methods you receive the Plug.Conn connection and the parameters received

It has to return a connection with answer (redirect, render, low level answer, ...)

Controller - Redirect

```elixir
defmodule TodoWeb.TodoController do
  use TodoWeb, :controller

  def empty(conn, _params) do
    bid = Todo.Reference.gen_reference()
    path = Routes.todo_path(conn, :board, bid)
    conn |> redirect(to: path)
  end
end
```


cabify

Controller - Render

```elixir
def board(conn, %{"board_id" => bid}) do
  todos = Todo.Todo.Store.board(bid)
  conn |> render("board.html", todos: todos, board: bid)
end
```


cabify

Controller - Direct Answer

```elixir
def create_task(conn, %{"board_id" => board, "title" => title}) do
  params = %{title: title, board: board}
  todo = Todo.Todo.new(params)

  case Store.save(todo) do
    :ok -> render(conn, "created.json", todo: todo)
    _ -> conn |> put_status(500) |> json(%{done: false})
  end
end
```


cabify

Views / Templates

Two main jobs:

- Render templates & layouts
- Transform data to be consumed

cabify

Views - HTML View

```elixir
defmodule ExampleWeb.LayoutView do
  use ExampleWeb, :view

  def title(), do: "My Web Title"
end
```


cabify

Views - JSON View

```elixir
defmodule TodoWeb.TodoApiView do
  use TodoWeb, :view

  def render("created.json", %{todo: todo}) do
    todo_json(todo)
  end

  @todo_params [:id, :title, :finished]
  defp todo_json(todo) do
    Map.take(todo, @todo_params)
  end
end
```


cabify

Templates

- Files which we pass data to create the final response (Usually HTML)
- Templating vith EEx (Similar to Ruby's ERB)
- There are Layouts & Templates, the entrypoint are layouts, and this ones render templates

- It has a hierarchy, if you call `render(conn, "index.html")` in the `UsersController`, the template used will be: `templates/users/index.html.eex`
- Assigns with `@assign`
- Calls to views with `<%= title() %>`

Templates

```
<div class="todos">
  <table class="table">
    <thead>
      <tr>
        <th>Title</th>
        <th>Finished?</th>
      </tr>
    </thead>
    <tbody id="todo-table-body">
      <%= for todo <- @todos do %>
        <%= render "_todo.html", todo: todo %>
      <% end %>
    </tbody>
  </table>
  <input id="create-todo-title" type="text" placeholder="Create TODO">
  <button onclick="createTodo()" type="button">Create!</button>
</div>

```

Channels

Soft real time

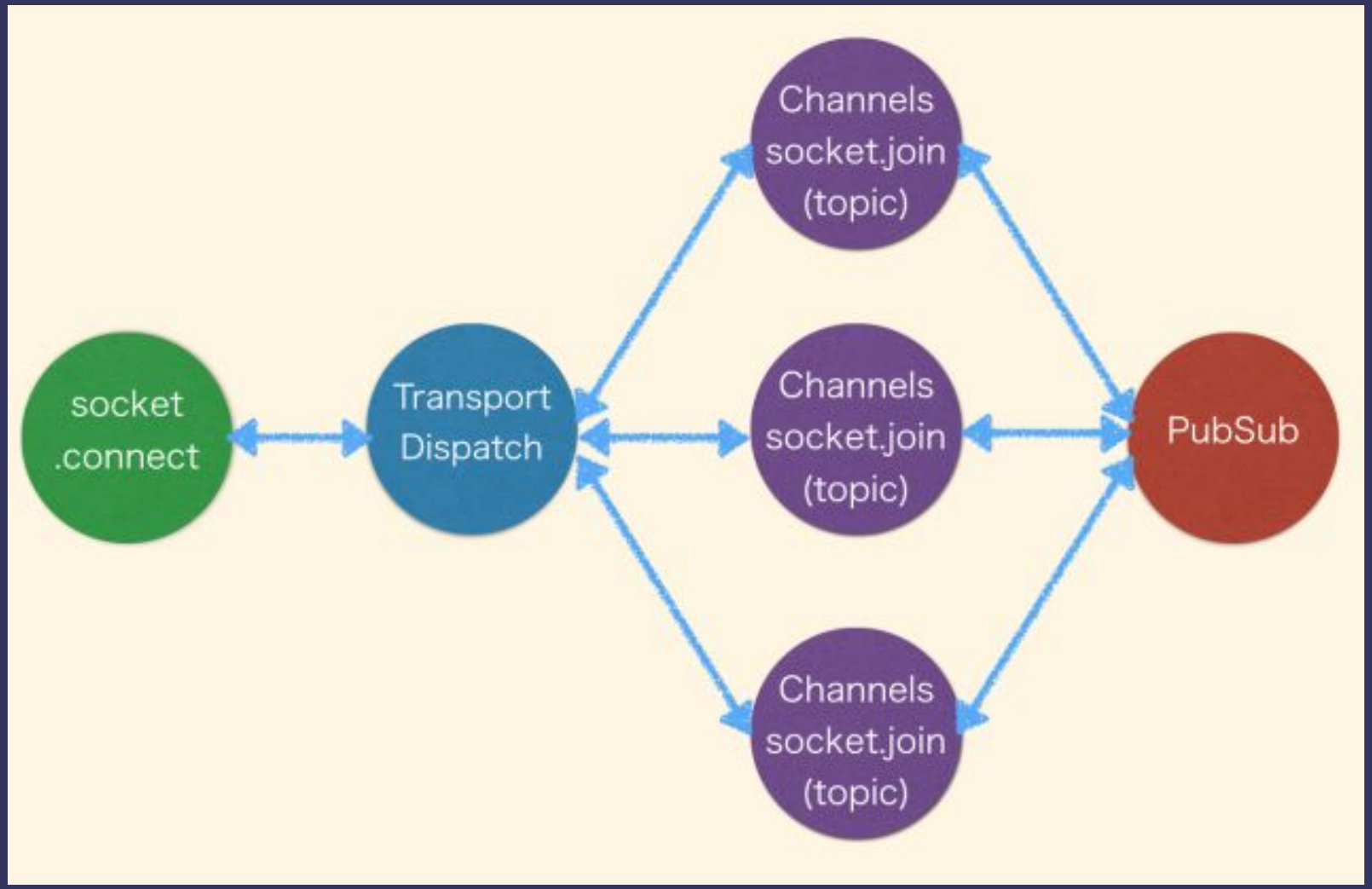
- chat rooms, updates, tracking, multiplayer events, monitoring, ...

Clients connect to topics (public_chat, board:123)

Transport agnostic

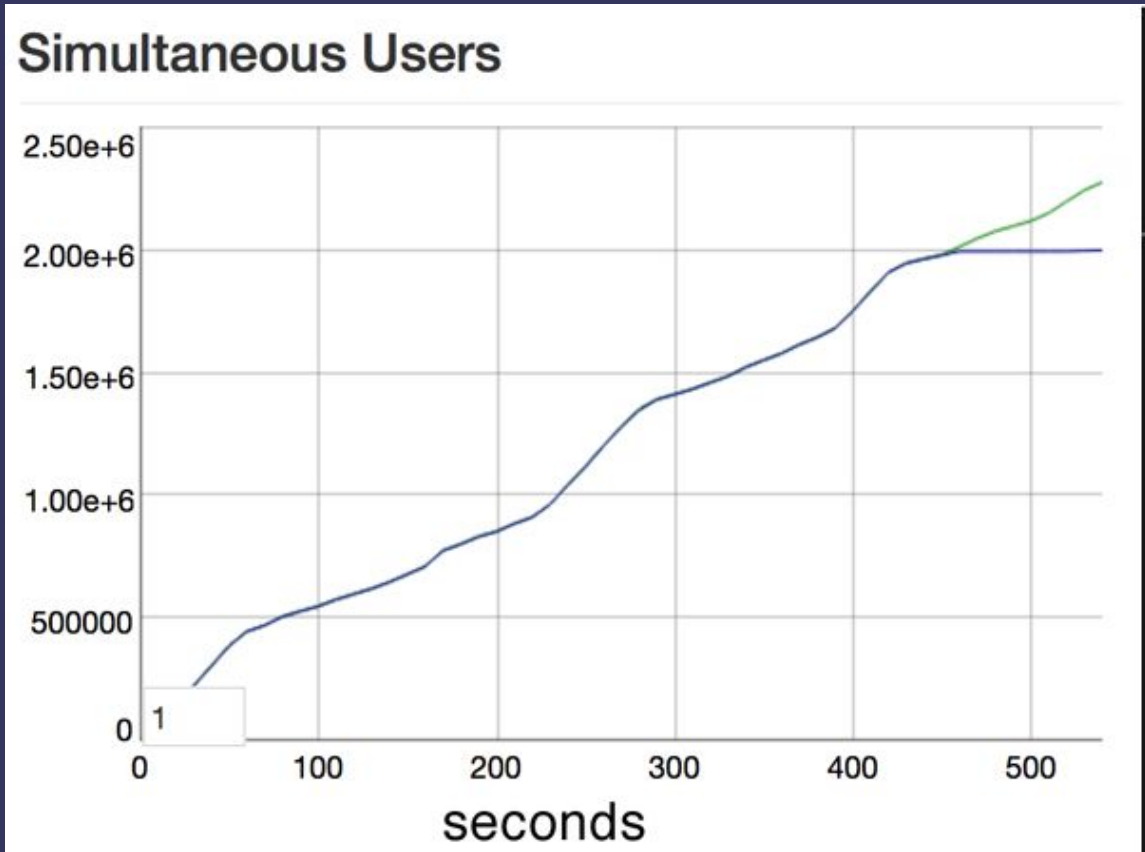
Bidirectional and broadcast communication

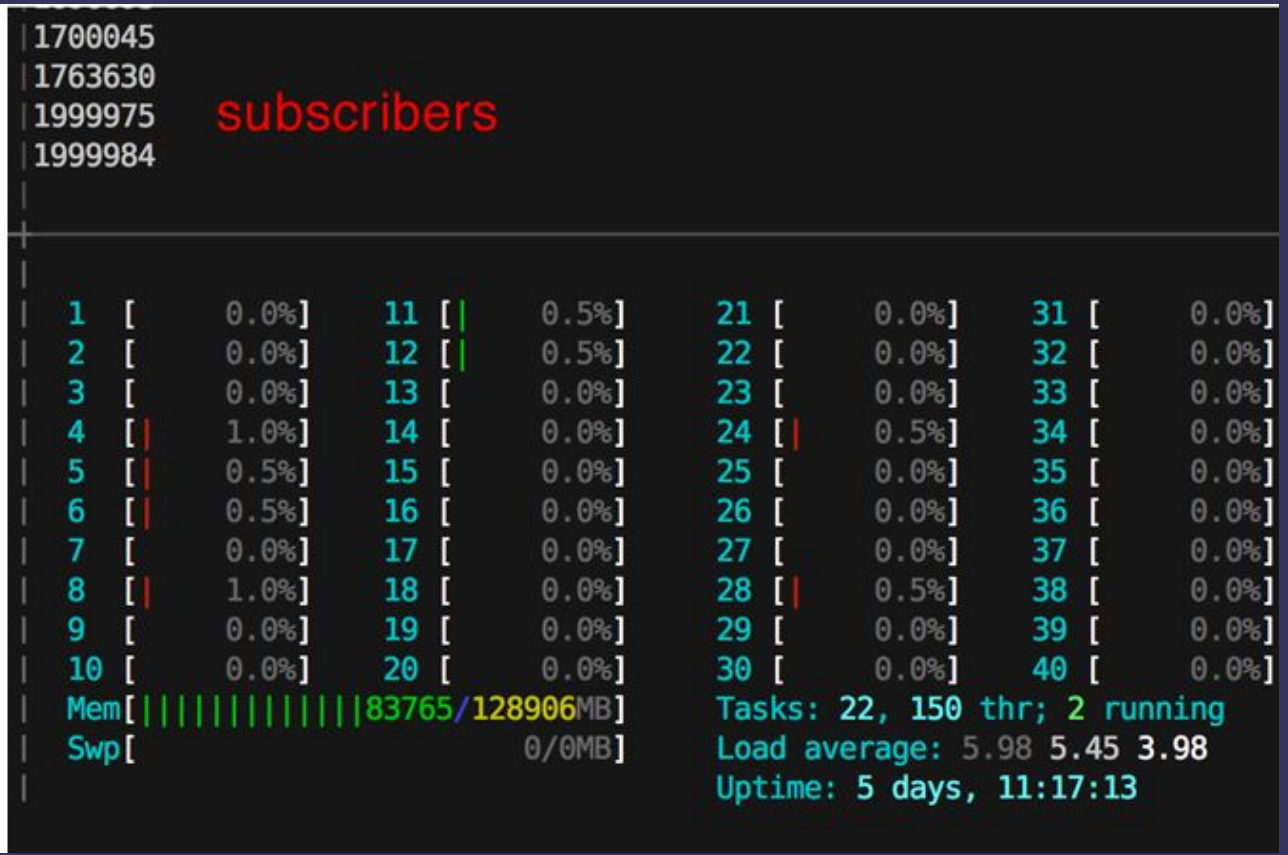
Channels



cabify

Channels





cabify

Channels - JS

```
1  import socket from "./socket";
2
3  let channel = socket.channel(`board:${board}`, {});
4
5  channel.on("create", ({body: todo}) => addTodo(todo));
6  channel.on("toggle", ({body: todo}) => {
7      setTodoFinished(todo.id, todo.finished);
8  });
9
10 channel.join()
11     .receive("ok", resp => { console.log(`Board ${board} joined successfully`, resp) })
12     .receive("error", resp => { console.log(`Board ${board} unable to join`, resp) });
13
```


cabify

Channels - Socket

```elixir
defmodule TodoWeb.UserSocket do
  use Phoenix.Socket

  channel "board:*", TodoWeb.BoardChannel

  def connect(_params, socket, _connect_info) do
    {:ok, socket}
  end

  def id(_socket), do: nil
end
```


cabify

Channels - Channel

```elixir
defmodule TodoWeb.BoardChannel do
  use Phoenix.Channel

  def join("board:" <> board_id, _params, socket) do
    {:ok, socket |> assign(:board, board_id)}
  end

  def notify({event, board, body}) do
    topic = "board:" <> board
    event = to_string(event)

    TodoWeb.Endpoint.broadcast(topic, event, %{body: body})
  end
end
```


cabify

Channels - Notify

```elixir
def create_task(conn, %{"board_id" => board, "title" => title}) do
  # ...
  case Store.save(todo) do
    :ok ->
      notify({:create, board, todo})
      render(conn, "created.json", todo: todo)
    # ...
  end
end

def toggle_task(conn, %{"board_id" => board, "task_id" => task}) do
  with #...
       :ok <- Store.save(todo) do
      notify({:toggle, board, todo})
      render(conn, "created.json", todo: todo)
    # ...
  end
end

defp notify(event) do
  TodoWeb.BoardChannel.notify(event)
end
```


cabify

Channels - Demo

tiny.cc/tryit19

bit.do/tryit19

cabify

Live View

JavaScript? 🔫

Server-Side rendered HTML
+
Channels

cabify

Live View

https://dockyard.com/blog/2018/12/12/phoenix-liveview-interactive-real-time-apps-no-need-to-write-javascript

https://github.com/phoenixframework/phoenix_live_view

https://elixirschool.com/blog/phoenix-live-view/

¡Thanks!
Questions?
(We're *hiring*)

Miguel Garcia
Senior Backend Developer @ Cabify
@rockneurotiko

tiny.cc/tryit19-rock

cabify