# TU PRIMER SERVIDOR EN ERLANG CON SSE UTILIZANDO COWBOY Y SUMO DB

Fernando Benavides (@elbrujohalcon)

Inaka Labs

November 18, 2013





- Soy programador desde que tenía 10 años
- Hago programación funcional desde hace 5 años, en Erlang
- Soy Director of Engineering en Inaka
- Me dedico a diseñar y, a veces, construir servidores
- No soy un programador Ruby
- Qué hago acá? o\_C





- Soy programador desde que tenía 10 años
- Hago programación funcional desde hace 5 años, en Erlang
- Soy Director of Engineering en Inaka
- Me dedico a diseñar y, a veces, construir servidores
- No soy un programador Ruby
- Qué hago acá? o\_O





- Soy programador desde que tenía 10 años
- Hago programación funcional desde hace 5 años, er Erlang
- Soy Director of Engineering en Inaka
- Me dedico a diseñar y, a veces, construir servidores
- No soy un programador Ruby
- Qué hago acá? o\_O





- Soy programador desde que tenía 10 años
- Hago programación funcional desde hace 5 años, en Erlang
- Soy Director of Engineering en Inaka
- Me dedico a diseñar y, a veces, construir servidores
- No soy un programador Ruby
- Qué hago acá? o\_C





- Soy programador desde que tenía 10 años
- Hago programación funcional desde hace 5 años, en Erlang
- Soy Director of Engineering en Inaka
- Me dedico a diseñar y, a veces, construir servidores
- No soy un programador Ruby
- Qué hago acá? o\_O





- Soy programador desde que tenía 10 años
- Hago programación funcional desde hace 5 años, en Erlang
- Soy Director of Engineering en Inaka
- Me dedico a diseñar y, a veces, construir servidores
- No soy un programador Ruby
- Qué hago acá? o₋O





# RESUMEN

#### ESCENARIO

- Servidor con API tipo REST
- Clientes necesitan actualizaciones en Real-Time

### Solución

- Se puede resolver con Ruby? Sí
- Existen otras soluciones? Sí

# ERLANG

- Es un paradigma distinto, requiere aprendizaje
- Es ideal para este tipo de escenarios





# RESUMEN

### ESCENARIO

- Servidor con API tipo REST
- Clientes necesitan actualizaciones en Real-Time

# Solución

- Se puede resolver con Ruby? Sí
- Existen otras soluciones? Sí

### ERLANG

- Es un paradigma distinto, requiere aprendizaje
- Es ideal para este tipo de escenarios





# RESUMEN

### ESCENARIO

- Servidor con API tipo REST
- Clientes necesitan actualizaciones en Real-Time

# Solución

- Se puede resolver con Ruby? Sí
- Existen otras soluciones? Sí

### **ERLANG**

- Es un paradigma distinto, requiere aprendizaje
- Es ideal para este tipo de escenarios





### En esta charla

- SSE
- Erlang / OTP básico
- Sumo DB básico
- Cowboy básico

- REST avanzado
- Erlang / OTP avanzado
- Sumo DB avanzado
- Elixir





### En esta charla

- SSE
- Erlang / OTP básico
- Sumo DB básico
- Cowboy básico

- REST avanzado
- Erlang / OTP avanzado
- Sumo DB avanzado
- Elixir





### En esta charla

- SSE
- Erlang / OTP básico
- Sumo DB básico
- Cowboy básico

- REST avanzado
- Erlang / OTP avanzado
- Sumo DB avanzado
- Elixir





### En esta charla

- SSE
- Erlang / OTP básico
- Sumo DB básico
- Cowboy básico

- REST avanzado
- Erlang / OTP avanzado
- Sumo DB avanzado
- Elixir





### En esta charla

- SSE
- Erlang / OTP básico
- Sumo DB básico
- Cowboy básico

- REST avanzado
- Erlang / OTP avanzado
- Sumo DB avanzado
- Elixir





### En esta charla

- SSE
- Erlang / OTP básico
- Sumo DB básico
- Cowboy básico

- REST avanzado
- Erlang / OTP avanzado
- Sumo DB avanzado
- Elixir





### En esta charla

- SSE
- Erlang / OTP básico
- Sumo DB básico
- Cowboy básico

- REST avanzado
- Erlang / OTP avanzado
- Sumo DB avanzado
- Elixir





### En esta charla

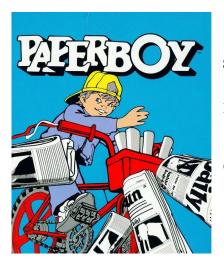
- SSE
- Erlang / OTP básico
- Sumo DB básico
- Cowboy básico

- REST avanzado
- Erlang / OTP avanzado
- Sumo DB avanzado
- Elixir





# La Aplicación



# Canillita

Simple API con dos endpoints:

POST /NEWS

para publicar noticias

GET /NEWS

para recibir noticias



### POST /news

<ロ > ← 日 → ← 日 → ← 日 → 上 目 = ・ の Q (へ)

### POST /news

```
curl -vX POST http://localhost:4004/news \
-H"Content-Type:application/json" \
-d'{ "source": "RubyConf",
     "content": "@elbrujohalcon muestra su sistema para ..." }'
> POST /news HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:4004
> Accept: */*
> Content-Type:application/json
> Content-Length: 50
>
< HTTP/1.1 204 No Content
< connection: keep-alive
< server: Cowboy
< date: Fri. 08 Nov 2013 20:06:01 GMT
< content-length: 0
<
```

```
GET /news
```

```
curl -vX GET http://localhost:4004/news
```

```
GET /news
curl -vX GET http://localhost:4004/news
> GET /news HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:4004
> Accept: */*
>
< HTTP/1.1 200 OK
< transfer-encoding: chunked</pre>
< connection: keep-alive
< server: Cowboy
< date: Thu, 07 Nov 2013 14:31:10 GMT
< content-type: text/event-stream</pre>
<
event: RubyConf
data: La charla de @elbrujohalcon esta por comenzar
```

### GET /news

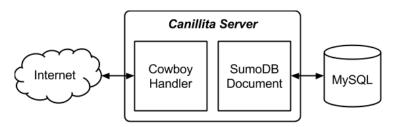
```
> GET /news HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:4004
> Accept: */*
>
< HTTP/1.1 200 OK
< transfer-encoding: chunked</pre>
< connection: keep-alive
< server: Cowboy
< date: Thu, 07 Nov 2013 14:31:10 GMT
< content-type: text/event-stream</pre>
<
event: RubyConf
data: La charla de @elbrujohalcon esta por comenzar
event: RubyConf
data: @elbrujohalcon muestra su sistema para ...
```

### GET /news

```
< HTTP/1.1 200 OK
< transfer-encoding: chunked</pre>
< connection: keep-alive
< server: Cowboy
< date: Thu, 07 Nov 2013 14:31:10 GMT
< content-type: text/event-stream</pre>
<
event: RubyConf
data: La charla de @elbrujohalcon esta por comenzar
event: RubyConf
data: @elbrujohalcon muestra su sistema para ...
event: RubyConf
data: el publico observa esta diapositiva :P
```



# ESQUEMA GENERAL

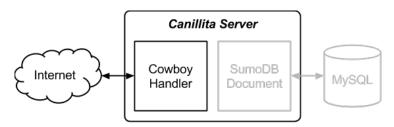


- canillita es una applicación Erlang típica
- Usa cowboy como web framework
- Y sumo\_db como motor de persistencia





# ESQUEMA GENERAL

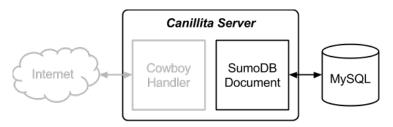


- canillita es una applicación Erlang típica
- Usa cowboy como web framework
- Y sumo\_db como motor de persistencia





# ESQUEMA GENERAL



- canillita es una applicación Erlang típica
- Usa cowboy como web framework
- Y sumo\_db como motor de persistencia





# COWBOY HANDLER

#### canillita\_news\_handler

- Procesa requests HTTP
- Responde a POST /news utilizando un REST handler
- Responde a GET /news utilizando un Loop handler





# COWBOY HANDLER

#### canillita\_news\_handler

- Procesa requests HTTP
- Responde a POST /news utilizando un REST handler
- Responde a GET /news utilizando un Loop handler

# En la función init se determina qué handler utilizar:

- cowboy\_rest define múltiples funciones a implementar para procesar un request
- sólo se implementan las que se necesitan
- en nuestro caso:





- cowboy\_rest define múltiples funciones a implementar para procesar un request
- sólo se implementan las que se necesitan
- en nuestro caso:

```
allowed_methods(Req, State) ->
   {[<<"POST">>], Req, State}.

content_types_accepted(Req, State) ->
   {[{<<"application/json">>, handle_post}], Req, State}.

resource_exists(Req, State) ->
   {false, Req, State}.
```





- cowboy\_rest define múltiples funciones a implementar para procesar un request
- sólo se implementan las que se necesitan
- en nuestro caso:

```
allowed_methods(Req, State) ->
    {[<<"POST">>], Req, State}.

content_types_accepted(Req, State) ->
    {[{<<"application/json">>, (handle_post)}], Req, State}.

resource_exists(Req, State) ->
    {false, Req, State}.
```





LA FUNCIÓN HANDLE\_POST

```
handle_post(Req, State) ->
  {ok, Body, Req1} = cowboy_req:body(Req),
  case json_decode(Body) of
    {Params} ->
      Title =
        proplists:get_value(<<"title">>, Params, <<"News">>)
      Content =
        proplists:get_value(<<"content">>>, Params, <<"">>>).
      NewsFlash = canillita_news:new(Title, Content),
      notify (NewsFlash),
      {true, Req1, State};
    {bad_json, Reason} ->
      \{ok, Req2\} =
        cowboy_req:reply(400,[],jiffy:encode(Reason),Req1),
      {halt, Req2, State}
  end.
```

# Rest Handler

LA FUNCIÓN HANDLE\_POST

```
handle_post(Req, State) ->
  {ok, Body, Req1} = cowboy_req:body(Req),
  case json_decode(Body) of
    {Params} ->
      Title =
        proplists:get_value(<<"title">>, Params, <<"News">>)
      Content =
        proplists:get_value(<<"content">>>, Params, <<"">>>).
      NewsFlash = canillita_news:new(Title, Content),
      notify (NewsFlash),
      {true, Req1, State};
    {bad_json, Reason} ->
      \{ok, Req2\} =
        cowboy_req:reply(400,[],jiffy:encode(Reason),Req1),
      {halt, Req2, State}
  end.
```

Obtiene el body del request



LA FUNCIÓN HANDLE\_POST

```
handle_post(Req, State) ->
 {ok, Body, Req1} = cowboy_req:body(Req),
 case json_decode(Body) of
   {Params} ->
      Title =
        proplists:get_value(<<"title">>, Params, <<"News">>)
      Content =
        proplists:get_value(<<"content">>>, Params, <<"">>>).
      NewsFlash = canillita_news:new(Title, Content),
      notify (NewsFlash),
      {true, Req1, State};
    {bad_json, Reason} ->
      \{ok, Req2\} =
        cowboy_req:reply(400,[],jiffy:encode(Reason),Req1),
      {halt, Req2, State}
 end
```

Lo parsea como json

LA FUNCIÓN HANDLE\_POST

```
handle_post(Req, State) ->
 {ok, Body, Req1} = cowboy_req:body(Req),
  case json_decode(Body) of
    {Params} ->
      Title =
        proplists:get_value(<<"title">>, Params, <<"News">>)
      Content =
        proplists:get_value(<<"content">>, Params, <<"">>>),
      NewsFlash = canillita_news:new(Title, Content),
      notify (NewsFlash),
      {true, Req1, State};
    {bad_json, Reason} ->
      \{ok, Req2\} =
        cowboy_req:reply(400,[],jiffy:encode(Reason),Req1),
      {halt, Req2, State}
 end.
```

Extrae los campos title y content

inaka

LA FUNCIÓN HANDLE\_POST

```
handle_post(Req, State) ->
  {ok, Body, Req1} = cowboy_req:body(Req),
  case json_decode(Body) of
    {Params} ->
      Title =
        proplists:get_value(<<"title">>, Params, <<"News">>)
      Content =
        proplists:get_value(<<"content">>>, Params, <<"">>>),
      NewsFlash = canillita_news:new(Title, Content),
      notify (NewsFlash),
      {true, Req1, State};
    {bad_json, Reason} ->
      \{ok, Req2\} =
        cowboy_req:reply(400,[],jiffy:encode(Reason),Req1),
      {halt, Req2, State}
  end.
```

Crea y almacena la noticia

inaka

LA FUNCIÓN HANDLE\_POST

```
handle_post(Req, State) ->
  {ok, Body, Req1} = cowboy_req:body(Req),
  case json_decode(Body) of
    {Params} ->
      Title =
        proplists:get_value(<<"title">>, Params, <<"News">>)
      Content =
        proplists:get_value(<<"content">>>, Params, <<"">>>).
      NewsFlash = canillita_news:new(Title, Content),
      notify(NewsFlash),
      {true, Req1, State};
    {bad_json, Reason} ->
      \{ok, Req2\} =
        cowboy_req:reply(400,[],jiffy:encode(Reason),Req1),
      {halt, Req2, State}
  end.
```

La envía a quienes estén escuchando

inaka

LA FUNCIÓN HANDLE\_POST

```
handle_post(Req, State) ->
  {ok, Body, Req1} = cowboy_req:body(Req),
  case json_decode(Body) of
    {Params} ->
      Title =
        proplists:get_value(<<"title">>, Params, <<"News">>)
      Content =
        proplists:get_value(<<"content">>>, Params, <<"">>>).
      NewsFlash = canillita_news:new(Title, Content),
      notify (NewsFlash),
      {true, Req1, State};
    {bad_json, Reason} ->
      \{ok, Req2\} =
        cowboy_req:reply(400,[],jiffy:encode(Reason),Req1),
      {halt, Req2, State}
  end.
```

Devuelve 204 No Content

inaka

- en nuestro handler, desde init estamos llamando a handle\_get
- más allá de init y handle\_get, cowboy\_req define una única función para implementar un loop handler: info
- es llamada cada vez que el proceso recibe un mensaje





LA FUNCIÓN HANDLE\_GET

```
handle_get(Req) ->
  \{ok, Req1\} =
    cowboy_req:chunked_reply(
      200, [{"content-type", <<"text/event-stream">>}], Req),
  LatestNews = canillita news: latest news().
  lists:foreach(
    fun(NewsFlash) ->
      send_flash(NewsFlash, Req1)
    end. LatestNews).
  pg2:join(canillita_listeners, self()),
  {loop, Req1, {}}.
```

inaka



LA FUNCIÓN HANDLE\_GET

```
handle_get(Req) ->
 \{ok, Req1\} =
    cowboy_req:chunked_reply(
      200, [{"content-type", <<"text/event-stream">>}], Req)
 LatestNews = canillita news: latest news().
 lists:foreach(
    fun(NewsFlash) ->
      send_flash(NewsFlash, Reg1)
    end. LatestNews).
 pg2:join(canillita_listeners, self()),
 {loop, Req1, {}}.
```

Setea el encoding y comienza a responder





LA FUNCIÓN HANDLE\_GET

```
handle_get(Req) ->
 \{ok, Req1\} =
    cowboy_req:chunked_reply(
      200, [{"content-type", <<"text/event-stream">>}], Req),
 LatestNews = canillita_news:latest_news(),
 lists:foreach(
   fun(NewsFlash) ->
      send_flash(NewsFlash, Reg1)
    end. LatestNews).
 pg2:join(canillita_listeners, self()),
 {loop, Req1, {}}.
```

Obtiene las noticias de la base de datos





LA FUNCIÓN HANDLE\_GET

```
handle_get(Req) ->
  \{ok, Req1\} =
    cowboy_req:chunked_reply(
      200, [{"content-type", <<"text/event-stream">>}], Req),
  LatestNews = canillita news: latest news().
  lists:foreach(
    fun(NewsFlash) ->
      send_flash(NewsFlash, Req1)
    end , LatestNews) ,
  pg2:join(canillita_listeners, self()),
  {loop, Req1, {}}.
```

Envía cada una de ellas usando la función send flash





LA FUNCIÓN HANDLE\_GET

```
handle_get(Req) ->
  \{ok, Req1\} =
    cowboy_req:chunked_reply(
      200, [{"content-type", <<"text/event-stream">>}], Req),
  LatestNews = canillita news: latest news().
  lists:foreach(
    fun(NewsFlash) ->
      send_flash(NewsFlash, Reg1)
    end. LatestNews).
 pg2: join(canillita_listeners, self()),
  {loop, Req1, {}}.
```

Se subscribe para recibir futuras noticias





LA FUNCIÓN HANDLE\_GET

```
handle_get(Req) ->
  \{ok, Req1\} =
    cowboy_req:chunked_reply(
      200, [{"content-type", <<"text/event-stream">>}], Req),
  LatestNews = canillita news: latest news().
  lists:foreach(
    fun(NewsFlash) ->
      send_flash(NewsFlash, Req1)
    end. LatestNews).
  pg2:join(canillita_listeners, self()),
  {loop, Req1, {}}.
```

y comienza a girar :)

inaka



OTRAS FUNCIONES





OTRAS FUNCIONES

La función info, envía la noticia





OTRAS FUNCIONES

La función info, envía la noticia y sigue girando :)





OTRAS FUNCIONES

La función send\_flash extrae los campos de la noticia





OTRAS FUNCIONES

Compone un bloque de texto a enviar





OTRAS FUNCIONES

Y lo envía al cliente





#### MATERIALES

#### Sobre mí

- Soy @elbrujohalcon en Twitter
- Soy elbrujohalcon en GitHub

#### SOBRE INAKA

- Pueden ver nuestro sitio web: http://inaka.net
- Y nuestro Blog: http://inaka.net/blog

#### SOBRE CANILLITA

- El código está en GitHub: inaka/canillita
- Las slides también: inaka/talks





# Muchas Gracias!



