

TU PRIMER SERVIDOR EN ERLANG CON SSE

UTILIZANDO COWBOY Y SUMO DB

Fernando Benavides (@elbrujoalcon)

Inaka Labs

November 18, 2013

inaka

HELLO WORLD!

- 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*

HELLO WORLD!

- 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*

HELLO WORLD!

- 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*

HELLO WORLD!

- 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*

HELLO WORLD!

- 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*

HELLO WORLD!

- 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

CONTENIDOS

En esta charla

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

Fuera de esta charla

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

CONTENIDOS

En esta charla

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

Fuera de esta charla

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

CONTENIDOS

En esta charla

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

Fuera de esta charla

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

CONTENIDOS

En esta charla

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

Fuera de esta charla

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

CONTENIDOS

En esta charla

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

Fuera de esta charla

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

CONTENIDOS

En esta charla

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

Fuera de esta charla

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

CONTENIDOS

En esta charla

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

Fuera de esta charla

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

CONTENIDOS

En esta charla

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

Fuera de esta charla

- 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

inaka

EJEMPLOS

POST /news

```
curl -vX POST http://localhost:4004/news \  
-H"Content-Type:application/json" \  
-d'{ "source": "RubyConf",  
    "content": "@elbrujoalcon muestra su sistema para ..." }'
```

EJEMPLOS

POST /news

```
curl -vX POST http://localhost:4004/news \  
-H"Content-Type:application/json" \  
-d'{ "source": "RubyConf",  
      "content": "@elbrujoalcon 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  
<
```

inaka

EJEMPLOS

GET /news

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

EJEMPLOS

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  
< connection: keep-alive  
< server: Cowboy  
< date: Thu, 07 Nov 2013 14:31:10 GMT  
< content-type: text/event-stream  
<
```

```
event: RubyConf
```

```
data: La charla de @elbrujoalcon esta por comenzar
```

inaka

EJEMPLOS

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
< connection: keep-alive
< server: Cowboy
< date: Thu, 07 Nov 2013 14:31:10 GMT
< content-type: text/event-stream
<
```

event: RubyConf

data: La charla de @elbrujoalcon esta por comenzar

event: RubyConf

data: @elbrujoalcon muestra su sistema para ...

inaka

EJEMPLOS

GET /news

```
< HTTP/1.1 200 OK
< transfer-encoding: chunked
< connection: keep-alive
< server: Cowboy
< date: Thu, 07 Nov 2013 14:31:10 GMT
< content-type: text/event-stream
<
event: RubyConf
data: La charla de @elbrujoalcon esta por comenzar

event: RubyConf
data: @elbrujoalcon muestra su sistema para ...

event: RubyConf
data: el publico observa esta diapositiva :P
```

inaka

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:

```
init(_Transport, Req, _Opts) ->  
  case cowboy_req:method(Req) of  
    {<<"POST">>, _} ->  
      {upgrade, protocol, cowboy_rest};  
    {<<"GET">>, Req1} ->  
      handle_get(Req1)  
  end.
```

REST HANDLER

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

REST HANDLER

- 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}.
```


REST HANDLER

- 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}.
```

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.
```

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

inaka

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.
```

Lo parsea como json

inaka

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.
```

Extrae los campos title y content

inaka

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.
```

Crea y almacena la noticia

inaka

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.
```

La envía a quienes estén escuchando

inaka

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.
```

Devuelve 204 No Content

inaka

LOOP HANDLER

- 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

LOOP HANDLER

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, {}}.
```

LOOP HANDLER

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, {}}.
```

Setea el encoding y comienza a responder

inaka

LOOP HANDLER

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, {}}.
```

Obtiene las noticias de la base de datos

LOOP HANDLER

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`

inaka

LOOP HANDLER

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, {}}.
```

Se subscribe para recibir futuras noticias

inaka

LOOP HANDLER

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

LOOP HANDLER

OTRAS FUNCIONES

```
info({news_flash, NewsFlash}, Req, State) ->
    send_flash(NewsFlash, Req),
    {loop, Req, State}.

send_flash(NewsFlash, Req) ->
    Event = canillita_news:get_title(NewsFlash),
    Data  = canillita_news:get_content(NewsFlash),
    Chunk =
        <<"event: ", Event/binary, "\n",
          "data: ", Data/binary, "\n\n">>,
    cowboy_req:chunk(Chunk, Req).
```


LOOP HANDLER

OTRAS FUNCIONES

```
info({news_flash, NewsFlash}, Req, State) ->  
  send_flash(NewsFlash, Req),  
  {loop, Req, State}.  
  
send_flash(NewsFlash, Req) ->  
  Event = canillita_news:get_title(NewsFlash),  
  Data = canillita_news:get_content(NewsFlash),  
  Chunk =  
    <<"event: ", Event/binary, "\n",  
      "data: ", Data/binary, "\n\n">>,  
  cowboy_req:chunk(Chunk, Req).
```

La función `info`, envía la noticia

LOOP HANDLER

OTRAS FUNCIONES

```
info({news_flash, NewsFlash}, Req, State) ->
    send_flash(NewsFlash, Req),
    {loop, Req, State}.

send_flash(NewsFlash, Req) ->
    Event = canillita_news:get_title(NewsFlash),
    Data = canillita_news:get_content(NewsFlash),
    Chunk =
        <<"event: ", Event/binary, "\n",
          "data: ", Data/binary, "\n\n">>,
    cowboy_req:chunk(Chunk, Req).
```

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

LOOP HANDLER

OTRAS FUNCIONES

```
info({news_flash, NewsFlash}, Req, State) ->
    send_flash(NewsFlash, Req),
    {loop, Req, State}.

send_flash(NewsFlash, Req) ->
    Event = canillita_news:get_title(NewsFlash),
    Data = canillita_news:get_content(NewsFlash),
    Chunk =
        <<"event: ", Event/binary, "\n",
          "data: ", Data/binary, "\n\n">>,
    cowboy_req:chunk(Chunk, Req).
```

La función `send_flash` extrae los campos de la noticia

LOOP HANDLER

OTRAS FUNCIONES

```
info({news_flash, NewsFlash}, Req, State) ->
    send_flash(NewsFlash, Req),
    {loop, Req, State}.

send_flash(NewsFlash, Req) ->
    Event = canillita_news:get_title(NewsFlash),
    Data = canillita_news:get_content(NewsFlash),
    Chunk =
        <<"event: ", Event/binary, "\n",
        "data: ", Data/binary, "\n\n">>,
    cowboy_req:chunk(Chunk, Req).
```

Compone un bloque de texto a enviar

LOOP HANDLER

OTRAS FUNCIONES

```
info({news_flash, NewsFlash}, Req, State) ->
    send_flash(NewsFlash, Req),
    {loop, Req, State}.

send_flash(NewsFlash, Req) ->
    Event = canillita_news:get_title(NewsFlash),
    Data = canillita_news:get_content(NewsFlash),
    Chunk =
        <<"event: ", Event/binary, "\n",
          "data: ", Data/binary, "\n\n">>,
    cowboy_req:chunk(Chunk, Req).
```

Y lo envía al cliente

MATERIALES

SOBRE MÍ

- Soy **@elbrujoalcon** en Twitter
- Soy **elbrujoalcon** 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!

inaka