# Tu primer servidor en Erlang con SSE

## utilizando Cowboy y Sumo DB

Fernando Benavides (*@elbrujohalcon*)

Inaka Labs

November 22, 2013

inaka

# Hello World!

- Soy programador desde que tenía 10 años
- Hago programación *funcional* 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*

inaka

# Hello World!

- Soy programador desde que tenía 10 años
- Hago programación *funcional* 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*

inaka

# Hello World!

- Soy programador desde que tenía 10 años
- Hago programación *funcional* 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*

inaka

# Hello World!

- Soy programador desde que tenía 10 años
- Hago programación *funcional* 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*

inaka

# Hello World!

- Soy programador desde que tenía 10 años
- Hago programación *funcional* 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*

inaka

# Hello World!

- Soy programador desde que tenía 10 años
- Hago programación *funcional* 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*

inaka

# 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 **ideal** para este tipo de escenarios
- Trae *de fábrica*:
  - Concurrencia con procesos livianos
  - Comunicación por envío de mensajes
  - Supervisión de procesos
  - Escalabilidad horizontal *automática*

inaka

# 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 **ideal** para este tipo de escenarios
- Trae *de fábrica*:
  - Concurrencia con procesos livianos
  - Comunicación por envío de mensajes
  - Supervisión de procesos
  - Escalabilidad horizontal *automática*

inaka

# 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 **ideal** para este tipo de escenarios
- Trae *de fábrica*:
  - Concurrencia con procesos livianos
  - Comunicación por envío de mensajes
  - Supervisión de procesos
  - Escalabilidad horizontal *automática*

inaka

# Contenidos

## En esta charla

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

## Fuera de esta charla

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

inaka

# Contenidos

## En esta charla

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

## Fuera de esta charla

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

inaka

# Contenidos

## En esta charla

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

## Fuera de esta charla

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

inaka

# Contenidos

## En esta charla

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

## Fuera de esta charla

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

inaka

# Contenidos

## En esta charla

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

## Fuera de esta charla

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

inaka

# Contenidos

## En esta charla

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

## Fuera de esta charla

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

inaka

# Contenidos

## En esta charla

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

## Fuera de esta charla

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

inaka

# Contenidos

## En esta charla

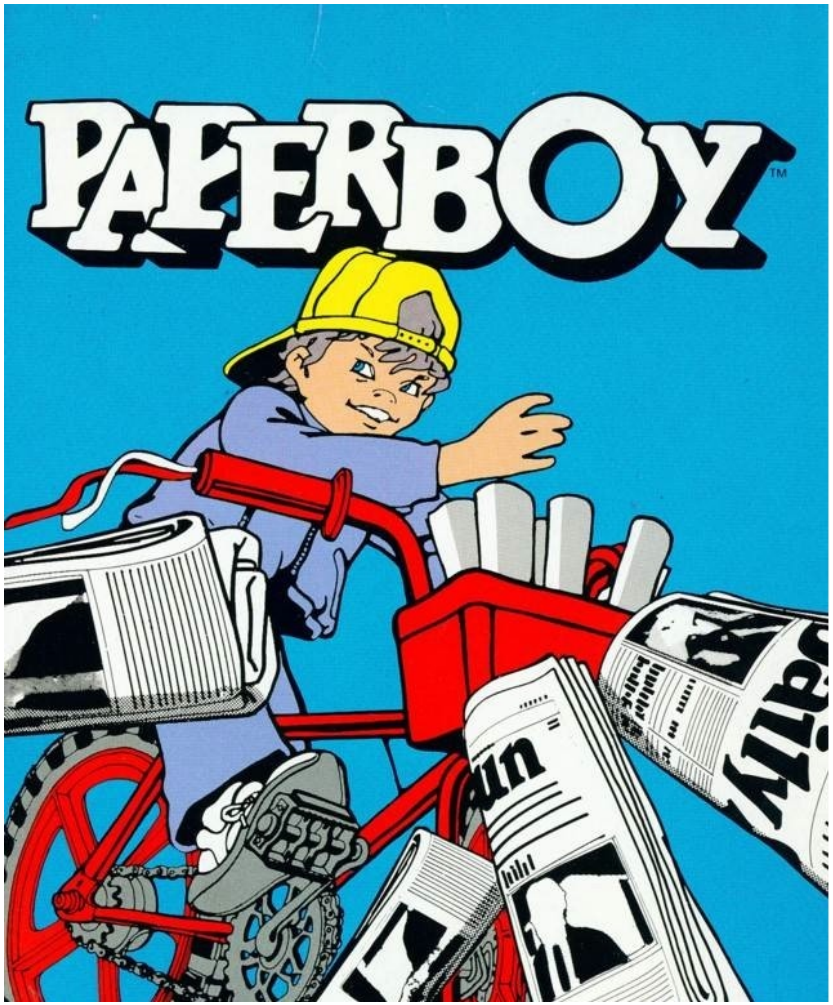
- SSE
- Erlang *básico*
- Sumo DB *básico*
- Cowboy *básico*

## Fuera de esta charla

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

inaka

# La Aplicación




## *Canillita*

Provee una API sencilla con dos endpoints:

POST /news
: para publicar noticias

GET /news
: para recibir noticias

# Ejemplos

## POST /news

```
curl -vX POST http://localhost:4004/news \
-H"Content-Type:application/json" \
-d'{ "title": "Tu Primer Servidor Erlang con SSE",
     "content": "@elbrujohalcon muestra su sistema para ..." }'
```

inaka

# Ejemplos

## POST /news

```
curl -vX POST http://localhost:4004/news \
-H"Content-Type:application/json" \
-d'{ "title": "Tu Primer Servidor Erlang con SSE",
     "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
<
```

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: old_news_flash
data: Nueva charla en la RubyConfAR!
data: La charla de @elbrujohalcon esta por comenzar
```

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: old_news_flash
data: Nueva charla en la RubyConfAR!
data: La charla de @elbrujohalcon esta por comenzar

event: news_flash
data: Tu Primer Servidor Erlang con SSE
data: @elbrujohalcon muestra su sistema para ...
```

inaka

# Ejemplos

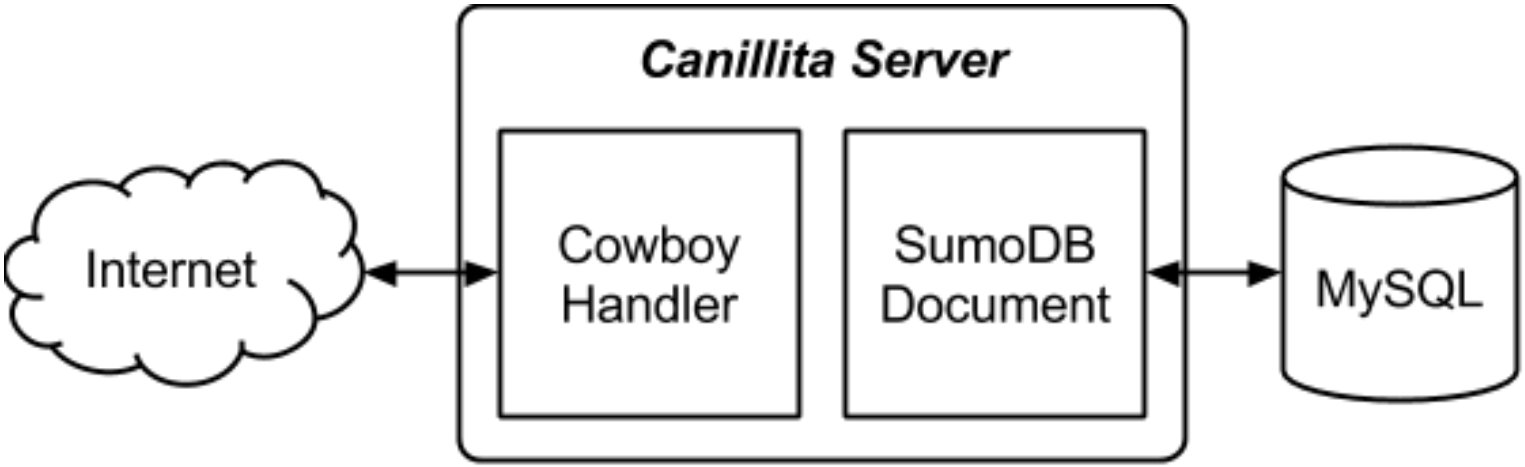
```
GET /news
< HTTP/1.1 200 OK
< transfer-encoding: chunked
< connection: keep-alive
< server: Cowboy
< date: Thu, 07 Nov 2013 14:31:10 GMT
< content-type: text/event-stream
<
event: old_news_flash
data: Nueva charla en la RubyConfAR!
data: La charla de @elbrujohalcon esta por comenzar

event: news_flash
data: Tu Primer Servidor Erlang con SSE
data: @elbrujohalcon muestra su sistema para ...

event: news_flash
data: Gran Concurrencia!
data: el publico observa esta diapositiva :P
```

inaka

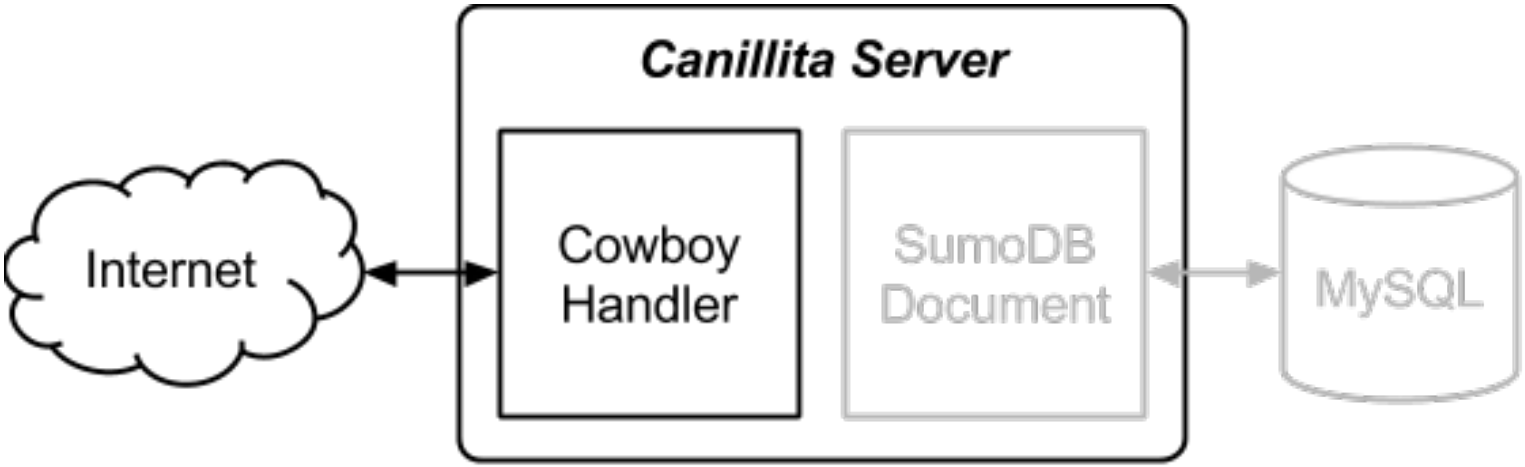
# Esquema General




- `canillita` es una application Erlang típica
- Usa `cowboy` como web framework
- Y `sumo_db` como motor de persistencia

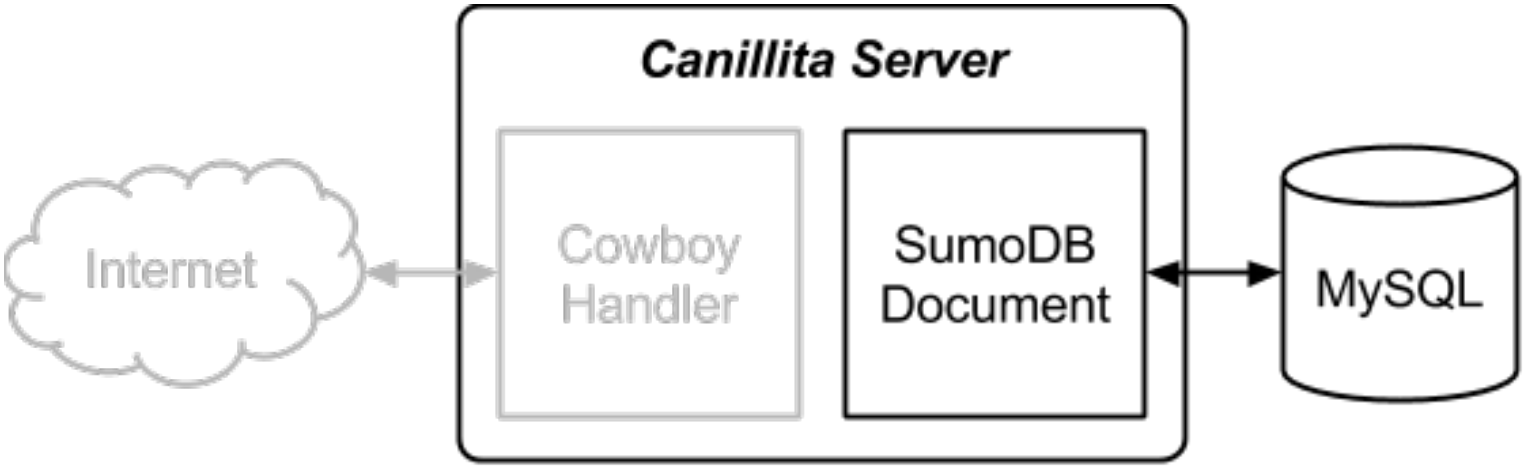
inaka

# Esquema General




- `canillita` es una application Erlang típica
- Usa `cowboy` como web framework
- Y `sumo_db` como motor de persistencia

# Esquema General




- `canillita` es una application Erlang típica
- Usa `cowboy` como web framework
- Y `sumo_db` como motor de persistencia

inaka

# Sintaxis

Antes de continuar, un poco de sintaxis Erlang...

```
send_if_found(Key, Hash, Pid) ->
        case proplists:get_value(Key, Hash) of
                undefined -> do_nothing;
                Value -> Pid ! {value, Value}
        end.
```

# Sintaxis

```
send_if_found(Key, Hash, Pid) ->
        case proplists:get_value(Key, Hash) of
                undefined -> do_nothing;
                Value -> Pid ! {value, Value}
        end.
```

Ésta es la función `send_if_found`

# Sintaxis

```
send_if_found(Key, Hash, Pid) ->
        case proplists:get_value(Key, Hash) of
                undefined -> do_nothing;
                Value -> Pid ! {value, Value}
        end.
```

Tiene 3 argumentos, cada uno de ellos se asocia a una variable

# Sintaxis

```
send_if_found(Key, Hash, Pid) ->
        case proplists:get_value(Key, Hash) of
                undefined -> do_nothing;
                Value -> Pid ! {value, Value}
        end.
```

Ésta es una estructura de `case`. Cada cláusula comienza con $->$ y culmina con ;

# Sintaxis

```
send_if_found(Key, Hash, Pid) ->
        case proplists:get_value(Key, Hash) of
                undefined -> do_nothing;
                Value -> Pid ! {value, Value}
        end.
```

En esta expresión se evalúa la función `get_value` del módulo `proplists`

# Sintaxis

```
send_if_found(Key, Hash, Pid) ->
        case proplists:get_value(Key, Hash) of
                undefined -> do_nothing;
                Value -> Pid ! {value, Value}
        end.
```

En esta cláusula se construye una *tupla* con el átomo `value` y el valor encontrado en el hash

# Sintaxis

```
send_if_found(Key, Hash, Pid) ->
        case proplists:get_value(Key, Hash) of
                undefined -> do_nothing;
                Value -> Pid ! {value, Value}
        end.
```

Y se envía esa tupla al proceso `Pid`

# Sintaxis

```
send_if_found(Key, Hash, Pid) ->
        case proplists:get_value(Key, Hash) of
                undefined -> do_nothing;
                Value -> Pid ! {value, Value}
        end.
```

Todas las clásuelas deben devolver algo, por eso ésta devuelve el átomo `do_nothing`

# Cowboy Handler

`canillita_news_handler`

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

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**

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.
```

inaka

# 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}.
```

inaka

# 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}.
```

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.
```

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.
```

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`, `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(<<"old_news_flash">>, NewsFlash, Req1)
    end, LatestNews),

  pg2:join(canillita_listeners, self()),

  {loop, Req1, {}}.
```

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(<<"old_news_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(<<"old_news_flash">>, NewsFlash, Req1)
    end, LatestNews),

  pg2:join(canillita_listeners, self()),

  {loop, Req1, {}}.
```

Obtiene las noticias de la base de datos

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(<<"old_news_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(<<"old_news_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(<<"old_news_flash">>, NewsFlash, Req1)
    end, LatestNews),

  pg2:join(canillita_listeners, self()),

  {loop, Req1, {}}.
```

y se queda esperando mensajes, que llegarán a `info`

inaka

# Loop Handler

## Otras funciones

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

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

notify(NewsFlash) ->
  lists:foreach(fun(Listener) ->
      Listener ! {news_flash, NewsFlash}
    end, pg2:get_members(canillita_listeners)).
```

inaka

# Loop Handler

## Otras funciones

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

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

notify(NewsFlash) ->
  lists:foreach(fun(Listener) ->
      Listener ! {news_flash, NewsFlash}
    end, pg2:get_members(canillita_listeners)).
```

La función `info`, envía la noticia

inaka

# Loop Handler

## Otras funciones

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

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

notify(NewsFlash) ->
  lists:foreach(fun(Listener) ->
      Listener ! {news_flash, NewsFlash}
    end, pg2:get_members(canillita_listeners)).
```

y vuelve a esperar mensajes

inaka

# Loop Handler

## Otras funciones

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

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

notify(NewsFlash) ->
  lists:foreach(fun(Listener) ->
      Listener ! {news_flash, NewsFlash}
    end, pg2:get_members(canillita_listeners)).
```

La función `send_flash` extrae los campos de la noticia

inaka

# Loop Handler

## Otras funciones

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

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

notify(NewsFlash) ->
  lists:foreach(fun(Listener) ->
      Listener ! {news_flash, NewsFlash}
    end, pg2:get_members(canillita_listeners)).
```

compone un bloque de texto a enviar

inaka

# Loop Handler

## Otras funciones

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

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

notify(NewsFlash) ->
  lists:foreach(fun(Listener) ->
      Listener ! {news_flash, NewsFlash}
    end, pg2:get_members(canillita_listeners)).
```

y lo envía al cliente

inaka

# Loop Handler

## Otras funciones

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

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

notify(NewsFlash) ->
  lists:foreach(fun(Listener) ->
      Listener ! {news_flash, NewsFlash}
    end, pg2:get_members(canillita_listeners)).
```

La función `notify` recorre la lista de subscriptos

inaka

# Loop Handler

## Otras funciones

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

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

notify(NewsFlash) ->
  lists:foreach(fun(Listener) ->
      Listener ! {news_flash, NewsFlash}
    end, pg2:get_members(canillita_listeners)).
```

y le envía un mensaje a cada uno

inaka

# SumoDB Document

`canillita_news`

- Implementa el behaviour `sumo_doc`
- Encapsula estado y comportamiento del modelo **News**
- Administra su persistencia a través de *SumoDB*

El behaviour `sumo_doc` define tres funciones a implementar:

- `sumo_schema`: definición del modelo
- `sumo_sleep`: traducción al formato de *SumoDB*
- `sumo_wakeup`: traducción a nuestro formato

inaka

# SumoDB Document

`canillita_news`

- Implementa el behaviour `sumo_doc`
- Encapsula estado y comportamiento del modelo **News**
- Administra su persistencia a través de *SumoDB*

El behaviour `sumo_doc` define tres funciones a implementar:

- `sumo_schema`: definición del modelo
- `sumo_sleep`: traducción al formato de *SumoDB*
- `sumo_wakeup`: traducción a nuestro formato

inaka

# Canillita News
## Behaviour Callbacks

```
sumo_schema() ->
  sumo:new_schema(?MODULE,
    [ sumo:new_field(id,              integer,
                          [id, not_null, auto_increment])
    , sumo:new_field(title,           text,      [not_null])
    , sumo:new_field(content,         text,      [not_null])
    , sumo:new_field(created_at,      datetime,  [not_null])
    , sumo:new_field(updated_at,      datetime,  [not_null])
    ]).

sumo_sleep(NewsFlash) -> NewsFlash.

sumo_wakeup(NewsFlash) -> NewsFlash.
```

inaka

# Canillita News

## Otras funciones

```
new(Title, Content) ->
  Now = {datetime, calendar:universal_time()},
  NewsFlash = [ {title,      Title}
              , {content,    Content}
              , {created_at, Now}
              , {updated_at, Now}],
  sumo:persist(canillita_news, NewsFlash).

get_title(NewsFlash) ->
  proplists:get_value(title, NewsFlash).
get_content(NewsFlash) ->
  proplists:get_value(content, NewsFlash).

latest_news() -> sumo:find_all(canillita_news).
```

inaka

# Canillita News

## Otras funciones

```
new(Title, Content) ->
  Now = {datetime, calendar:universal_time()},
  NewsFlash = [ {title,      Title}
              , {content,    Content}
              , {created_at, Now}
              , {updated_at, Now}],
  sumo:persist(canillita_news, NewsFlash).

get_title(NewsFlash) ->
  proplists:get_value(title, NewsFlash).
get_content(NewsFlash) ->
  proplists:get_value(content, NewsFlash).

latest_news() -> sumo:find_all(canillita_news).
```

La función `new`, crea una entidad y la persiste

inaka

# Canillita News

## Otras funciones

```
new(Title, Content) ->
  Now = {datetime, calendar:universal_time()},
  NewsFlash = [ {title,      Title}
              , {content,    Content}
              , {created_at, Now}
              , {updated_at, Now}],
  sumo:persist(canillita_news, NewsFlash).

get_title(NewsFlash) ->
  proplists:get_value(title, NewsFlash).
get_content(NewsFlash) ->
  proplists:get_value(content, NewsFlash).

latest_news() -> sumo:find_all(canillita_news).
```

Las funciones `get_*` son simples proyectores

inaka

# Canillita News

## Otras funciones

```
new(Title, Content) ->
  Now = {datetime, calendar:universal_time()},
  NewsFlash = [ {title,      Title}
              , {content,    Content}
              , {created_at, Now}
              , {updated_at, Now}],
  sumo:persist(canillita_news, NewsFlash).

get_title(NewsFlash) ->
  proplists:get_value(title, NewsFlash).
get_content(NewsFlash) ->
  proplists:get_value(content, NewsFlash).

latest_news() -> sumo:find_all(canillita_news).
```

La función `latest_news` retorna todos las entidades

inaka

# ¿Y qué tal funciona?

# Tsung


- http://tsung.erlang-projects.org/
- Herramienta de medición de carga multi-protocolo distribuida
- Hecha en Erlang
- Puede utilizarse también para testear

inaka

# Tsung


- http://tsung.erlang-projects.org/
- Herramienta de medición de carga multi-protocolo distribuida
- Hecha en Erlang
- Puede utilizarse también para testear

inaka

# Tsung


- http://tsung.erlang-projects.org/
- Herramienta de medición de carga multi-protocolo distribuida
- Hecha en Erlang
- Puede utilizarse también para testear

inaka

# Escenario

**Test**

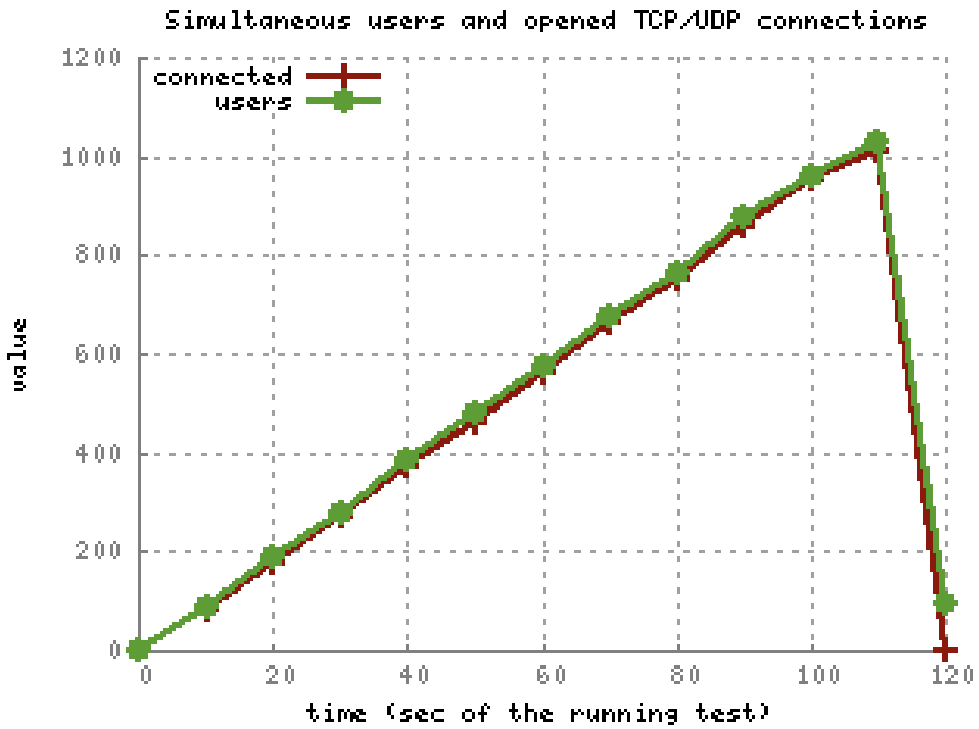
- Duración: 500 segundos
- Requests a `POST /news`: 1 por segundo
- Requests a `GET /news`: 50 por segundo

**Hardware**

- MacBook PRO con OSX 10.9
- Procesador: 2.4 GHz Intel Core i5
- Memoria: 8 GB 1600 MHz DDR3

inaka

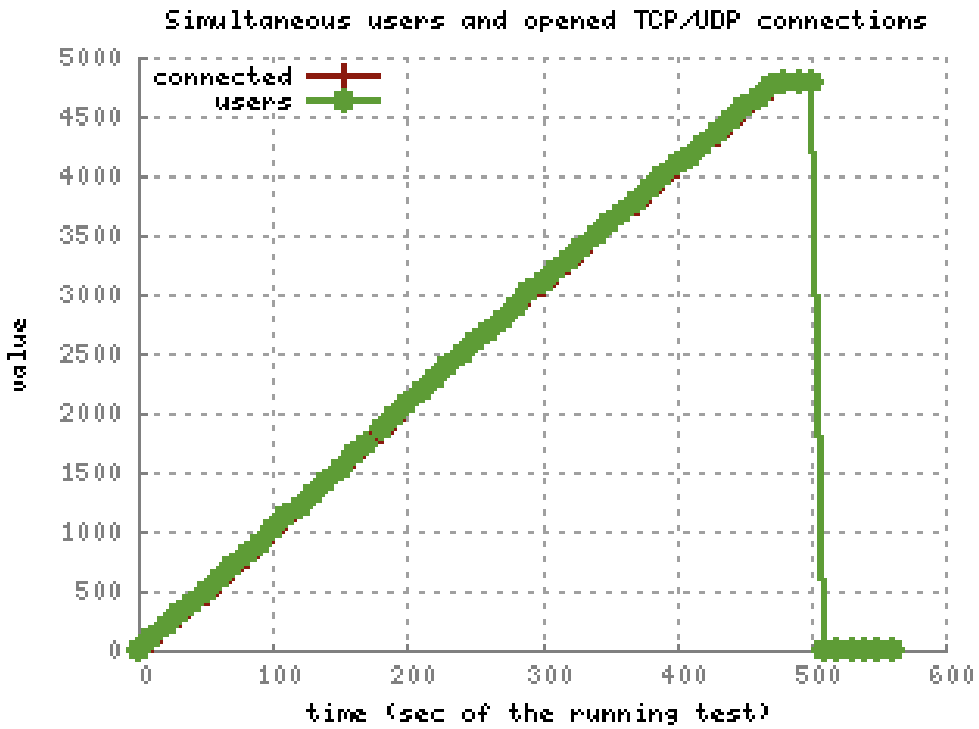
# Usuarios conectados




Un servidor *básico* en *Ruby* soportó cerca de **1000** conexiones simultáneas y luego la virtual murió

inaka

# Usuarios conectados




Un servidor *básico* en *Erlang* soportó más de **4500** conexiones simultáneas y luego comenzó a no responder a nuevas conexiones

inaka

# Entonces...

¿**Erlang** es mejor que **Ruby** en este caso?

*Yo qué sé*
*(Warren Sanchez)*

Lo que sé es que **mi** servidor **Erlang** es mejor que **mi** servidor **Ruby** en este caso

Y eso no es poco!! :)

inaka

# Entonces...

¿**Erlang** es mejor que **Ruby** en este caso?

*Yo qué sé*

*(Warren Sanchez)*

Lo que sé es que **mi** servidor **Erlang** es mejor que **mi** servidor **Ruby** en este caso

Y eso no es poco!! :)

inaka

# Entonces. . .

¿**Erlang** es mejor que **Ruby** en este caso?

*Yo qué sé*
*(Warren Sanchez)*

Lo que sé es que **mi** servidor **Erlang** es mejor que **mi** servidor **Ruby** en este caso

Y eso no es poco!! :)

inaka

Entonces. . .

¿**Erlang** es mejor que **Ruby** en este caso?

*Yo qué sé*
*(Warren Sanchez)*

Lo que sé es que **mi** servidor **Erlang** es mejor que **mi** servidor **Ruby** en este caso

Y eso no es poco!! :)

inaka

# Moraleja

Construir tu primer sistema en *Erlang* puede ser algo costoso y hasta abrumador. Sin embargo, para ciertos escenarios habituales no resulta difícil construir las aplicaciones necesarias e integrarlas con otros sistemas, aún sin dominar su arquitectura a fondo.
Y esas aplicaciones traen *gratis* todas las ventajas de Erlang, como *escalabilidad horizontal*, *supervisión de procesos*, *manejo de múltiples nodos*, etc.
Además, con ese punto de partida, se puede continuar aprendiendo el lenguaje paso a paso.

inaka

# Moraleja

Construir tu primer sistema en *Erlang* puede ser algo costoso y hasta abrumador. Sin embargo, para ciertos escenarios habituales no resulta difícil construir las aplicaciones necesarias e integrarlas con otros sistemas, aún sin dominar su arquitectura a fondo.

Y esas aplicaciones traen *gratis* todas las ventajas de Erlang, como *escalabilidad horizontal*, *supervisión de procesos*, *manejo de múltiples nodos*, etc.

Además, con ese punto de partida, se puede continuar aprendiendo el lenguaje paso a paso.

inaka

# Moraleja

Construir tu primer sistema en *Erlang* puede ser algo costoso y hasta abrumador. Sin embargo, para ciertos escenarios habituales no resulta difícil construir las aplicaciones necesarias e integrarlas con otros sistemas, aún sin dominar su arquitectura a fondo.
Y esas aplicaciones traen *gratis* todas las ventajas de Erlang, como *escalabilidad horizontal*, *supervisión de procesos*, *manejo de múltiples nodos*, etc.
Además, con ese punto de partida, se puede continuar aprendiendo el lenguaje paso a paso.

inaka

# Moraleja

Construir tu primer sistema en *Erlang* puede ser algo costoso y hasta abrumador. Sin embargo, para ciertos escenarios habituales no resulta difícil construir las aplicaciones necesarias e integrarlas con otros sistemas, aún sin dominar su arquitectura a fondo.
Y esas aplicaciones traen *gratis* todas las ventajas de Erlang, como *escalabilidad horizontal*, *supervisión de procesos*, *manejo de múltiples nodos*, etc.
Además, con ese punto de partida, se puede continuar aprendiendo el lenguaje paso a paso.

inaka

# 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**

inaka

# Muchas Gracias!

inaka