Erlang for Authoritative DNS

Anthony Eden
Founder of DNSimple

What is Authoritative DNS?

http://www.flickr.com/photos/31797858@N00/6337153123

Where We Came From

http://www.flickr.com/photos/8929612@N04/6021468948

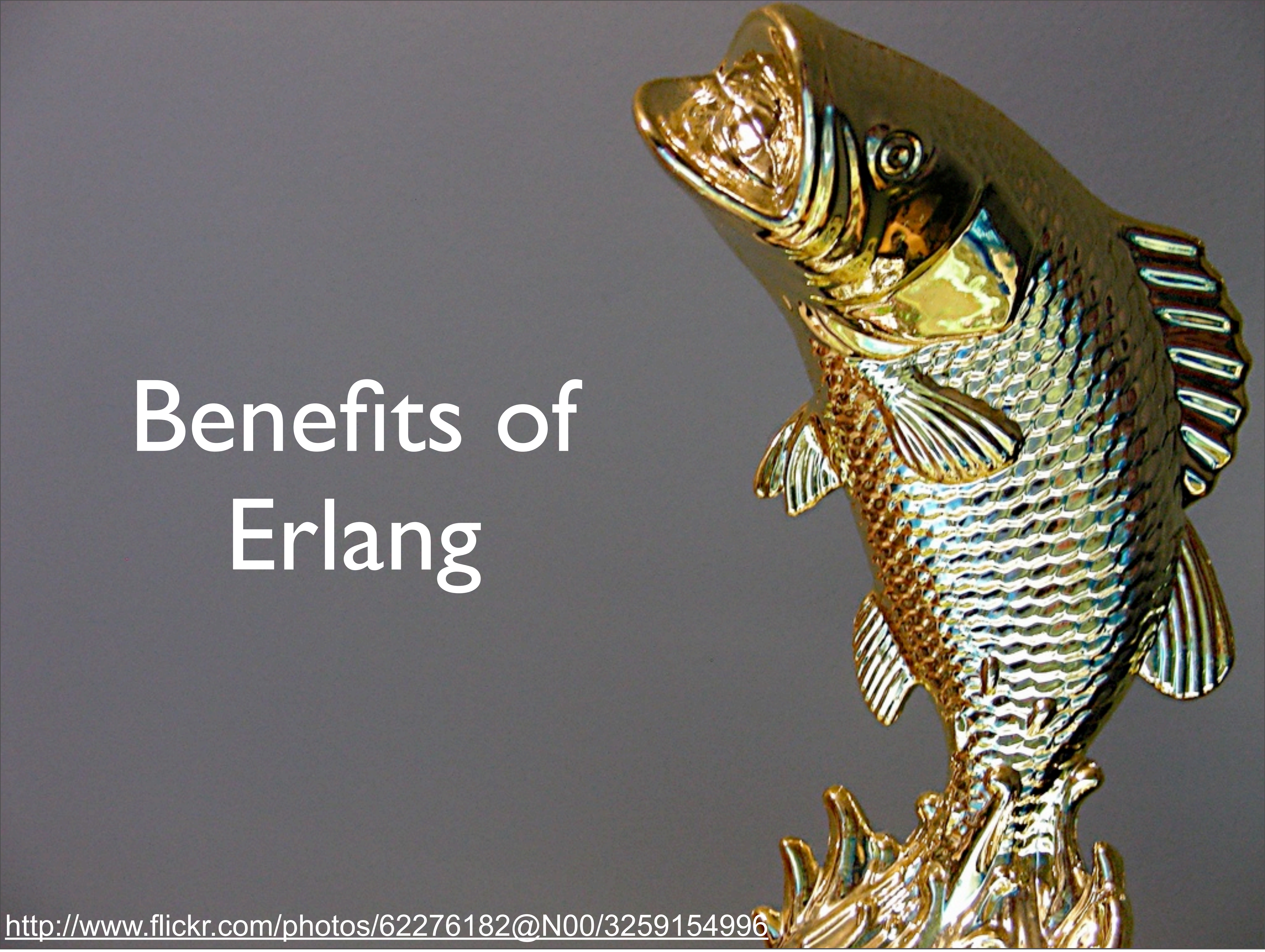
PowerDNS + Ruby backend + MySQL

Sunday, September 22, 13

Ruby backend communicating with Clojure service talking to MySQL
Lua extension for PowerDNS + MySQL

Sunday, September 22, 13

Known for parallel processing
Started as a way of learning Erlang



3 benefits with examples.

Binary Bit Syntax

```
decode_message(<<Id:16, QR:1, OC:4, AA:1, TC:1, RD:1, RA:
1, 0:1, AD:1, CD:1, RC:4, QC:16, ANC:16, AUC:16, ADC:16,
Rest/binary>> = MsgBin) ->
```

```erlang
decode_message(<<Id:16, QR:1, OC:4, AA:1, TC:1, RD:1, RA:
1, 0:1, AD:1, CD:1, RC:4, QC:16, ANC:16, AUC:16, ADC:16,
Rest/binary>> = MsgBin) ->
```

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```


Maps well to the RFC 1035 message diagram.

ID is 16 bits (2 octets)
QR = Query (0) or Response (1)
Opcode = Four bits defining type of query (0 = standard, 1 = inverse query, 2 = server status)
AA = Authoritative Answer bit
TC = Truncation bit
RD = Recursion Desired bit
RA = Recursion Available bit
Z = Reserved
RCODE = Response code
QC = Question Count
ANC = Answer Count
AUC = Authority Count
ADC = Additional Count

Pattern Matching

```
json_record_to_erlang([Name, <<"A">>, Ttl, Data]) ->
  Ip = proplists:get_value(<<"ip">>, Data),
  case inet_parse:address(binary_to_list(Ip)) of
    {ok, Address} -> #dns_rr{
      name = Name,
      type = ?DNS_TYPE_A,
      data = #dns_rrdata_a{ip = Address},
      ttl = Ttl
    };
    {error, Reason} -> {}
  end;

json_record_to_erlang([Name, <<"CNAME">>, Ttl, Data]) ->
  #dns_rr{
    name = Name,
    type = ?DNS_TYPE_CNAME,
    data = #dns_rrdata_cname{dname =
proplists:get_value(<<"dname">>, Data)},
    ttl = Ttl
  };
```

Processes and OTP

Spawn → Init → Receive → Exit

OTP process lifecycle

Packet Cache

Sunday, September 22, 13

Maintains state.

```erlang
-module(erldns_packet_cache).

-behavior(gen_server).

% API
-export([start_link/0, get/1, put/2, sweep/0, clear/0]).

% Gen server hooks
-export([init/1,
	 handle_call/3,
	  handle_cast/2,
	  handle_info/2,
	  terminate/2,
	  code_change/3
	       ]).

-define(SERVER, ?MODULE).
-define(SWEEP_INTERVAL, 1000 * 60 * 10). % Every 10 minutes

-record(state, {ttl, tref}).
```

```erlang
% API
-export([start_link/0, get/1, put/2, sweep/0, clear/0]).
```


Messages for get, put, sweep and clear.

```erlang
%% Public API
start_link() ->
  gen_server:start_link({local, ?SERVER}, ?MODULE, [], []).

get(Question) ->
  gen_server:call(?SERVER, {get_packet, Question}).

put(Question, Response) ->
  gen_server:call(?SERVER, {set_packet, [Question, Response]}).

sweep() ->
  gen_server:cast(?SERVER, {sweep, []}).

clear() ->
  gen_server:cast(?SERVER, {clear}).
```

```
init([]) ->
  init([20]);

init([TTL]) ->
  ets:new(packet_cache, [set, named_table]),
  {ok, Tref} = timer:apply_interval(?SWEEP_INTERVAL, ?MODULE,
sweep, []),
  {ok, #state{ttl = TTL, tref = Tref}}.
```


Initialize the packet cache. If a TTL is not specified, use a default.
Use of pattern matching again.
State includes the TTL and a reference to the timer.

```erlang
handle_call({get_packet, Question}, _From, State) ->
  case ets:lookup(packet_cache, Question) of
    [{Question, {Response, ExpiresAt}}] ->
      {_,T,_} = erlang:now(),
      case T > ExpiresAt of
        true ->
          lager:debug("Cache hit but expired"),
          {reply, {error, cache_expired}, State};
        false ->
          lager:debug("Time is ~p. Packet hit expires at ~p.", [T, ExpiresAt]),
          {reply, {ok, Response}, State}
      end;
    _ -> {reply, {error, cache_miss}, State}
  end;
```

```
handle_call({set_packet, [Question, Response]}, _From, State) ->
  {_,T,_} = erlang:now(),
  ets:insert(packet_cache, {Question, {Response, T + State#state.ttl}}),
  {reply, ok, State}.
```

```erlang
handle_cast({sweep, []}, State) ->
  lager:debug("Sweeping packet cache"),
  {_, T, _} = erlang:now(),
  Keys = ets:select(packet_cache, [
    {{'$1', {'_', '$2'}}, [{'<', '$2', T - 10}], ['$1']}
  ]),
  lager:debug("Found keys: ~p", [Keys]),
  lists:foreach(fun(K) -> ets:delete(packet_cache, K) end, Keys),
  {noreply, State};


handle_cast({clear}, State) ->
  ets:delete_all_objects(packet_cache),
  {noreply, State}.
```

Challenges

http://www.flickr.com/photos/36698822@N00/1935755228

Thinking in Erlang

Sunday, September 22, 13

Early challenge

Thinking in functions and processes.
Transforming data vs. updating data.

Operations

Sunday, September 22, 13

Later challenge

How do you deploy, test and operate Erlang systems?

Working around what Erlang isn't good at

Sunday, September 22, 13

Based on our needs:

Erlang is not suited for data manipulation on large amounts of data.
Erlang is not suited for producing large amounts of structured data.

Your mileage may vary.

Identify Bottlenecks

Sunday, September 22, 13

Measure from the outside.
Measure from the inside.

Optimize sequential paths

Sunday, September 22, 13

Amdahl's Law: The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program.

Minimize the amount of time spent in single-threaded processes.
As soon as possible hand off to an independent, parallel process.
Be aware that the time to hand off to the process can be significant.

Reduce garbage collection

Sunday, September 22, 13

Copying data can cause excessive garbage collection.
Reuse processes wherever possible.

Where we ended up

http://www.flickr.com/photos/24387390@N08/3585729893

Packet read loop

Sunday, September 22, 13

Single process
Measured within using timer module.
Measured without using a timed test suite.
Optimized the tight loop.
Initially processed all requests in the loop, 30 to 250ms per request
then moved to poolboy, 4 ms per request
then moved to queue module, <1ms per request

```erlang
handle_request(Socket, Host, Port, Bin, State) ->
  case queue:out(State#state.workers) of
    {{value, Worker}, Queue} ->
      gen_server:cast(Worker, {udp_query, Socket, Host, Port, Bin}),
      {noreply, State#state{workers = queue:in(Worker, Queue)}};
    {empty, _Queue} ->
      lager:info("Queue is empty, dropping packet"),
      {noreply, State}
  end.
```

```erlang
handle_info({udp, Socket, Host, Port, Bin}, State) ->
  Response = folsom_metrics:histogram_timed_update(
    udp_handoff_histogram, ?MODULE, handle_request,
    [Socket, Host, Port, Bin, State]
  ),
  inet:setopts(State#state.socket, [{active, once}]),
  Response;
```

Sunday, September 22, 13

99th percentile ranges from 35 microseconds to 60 microseconds.
17,000 – 25000 questions per second
680,000 – 1,000,000 qps across 40 nodes

Process pool

# of workers is configured externally.
When the UDP handler is initialized all workers are created.
queue:out to retrieve a worker + cast + queue:in
Assumption is the worker will finish processing before the queue is exhausted, drop packet otherwise

In-memory cache of zone data

Originally we pulled from a database.
MySQL, then Postgres
Ultimately the best performance was to load the zones in memory
This presents its own challenges
Naming and caching - hard problems.

Remote zone loading from a special purpose server

Sunday, September 22, 13

Sacrifice start up time.
Minimize the transformation of zone data.
Let Golang handle production of zone data.

Current Performance Metrics

Sunday, September 22, 13

Folsom all the things.


Where we’re headed
http://www.flickr.com/photos/33118864@N00/1391641134

SO_REUSEPORT

Multiple processes read from the same UDP port.

Metrics

Sunday, September 22, 13

Identify other high-latency paths and optimize those.
Example: start up bottleneck, which can be parallel.

Sunday, September 22, 13

Embrace the Erlang tools, but don't give up on other tooling.
Erlang tools include the ability to connect to and operate on running nodes.
Another option is to embed Cowboy to provide an HTTP API for command and control.

http://www.flickr.com/photos/kmacke/3300751503

Erlang’s sweet spot

Questions?

http://www.flickr.com/photos/60368684@N00/8578067721

Erlang for Authoritative DNS

Anthony Eden
Founder of DNSimple

http://github.com/aetrion/erl-dns