A Cowboy quest for a modern web

Loïc Hoguin

Nine Nines
Dev:Extend

Erlang User Conference 2011

Why Cowboy?

- Because guns are better than arrows
- It’s all about the hat

History

- First commit early March 2011
- Mentioned on the Github blog two weeks later
- First beta early September 2011
- First talk early November 2011

Users and contributors

- The etorrent project for its acceptor pool and web interface
- The sockjs-erlang project for a websocket and HTTP server
- Smarkets, Nivertech and other companies
- The #erlounge IRC folks

Cowboy’s listeners

- Cowboy isn’t just an HTTP server
- Cowboy allows you to start your own listener
  - Essentially a transport and protocol-agnostic acceptor pool
  - An acceptor pool is a pool of processes accepting connections
- You can have as many listeners running side by side as you want
- Even listeners completely unrelated to the HTTP protocol

Transport handlers

- Tiny wrappers around transport-related code
  - listen, accept, recv, send, setopts…
- Works with any reliable transport
  - TCP, SSL
  - Theorically also the UDP-based ENet

Protocol handlers

- Contains the protocol implementation
- Usually takes the form of a state machine
- gen_fsm can of course be used
- Only a start_link/4 function is required

Acceptor loop

- Wait for a connection
- Accept the connection
  - Start a request process
  - Give the socket to that new process
  - Inform the new process everything’s ready
- Check for max # of connections
  - Wait if this max is reached
- Repeat

A pool of many acceptors

- Idea taken from Mochiweb
- Having more than one acceptor process speeds things up
- Erlang processes are cheap
- Why not use 100 processes to accept connections?

Supervision

- All processes started by a listener are supervised
- You don’t need to worry about supervision!
- We’re working on release upgrades

Connection pools

- There are different kinds of connections
- Short-lived request/response should have a small max #
- Long-lived idle connections can have a much larger max
- Pools allow separating those connection types into two groups
  - Or more
- You can add, move or delete connections from pools
- By default all connections are added to the pool named ’default’

Cowboy’s HTTP server

- Because all applications have their own HTTP interface

Initial design ideas

- gen_fsm, lists
- Normal process, lists, with [{active, once}]
- Normal process, binary, with [{active, once}]
- Normal process, binary with [{active, false}]
- Normal process, binary, calling erlang:decode_packet/3 directly
- We did want binary from the start but it required more custom code

Dispatch rules

- Idea taken from Webmachine
- Matching hostname and path to HTTP handlers
- Partial matching allows binding hostname/path parts to variables
- If we had the rule [<<"users">>, id, <<"pics">>]
  - The path /users/42/pics would match
  - {id, 42} would be added to the bindings
- Entirely optional

HTTP handlers

```erlang
-module(my_handler).
-export([init/3, handle/2, terminate/2]).

init(_TransportType, Req, _Opts) ->
    {ok, Req, undefined_state}.

handle(Req, State) ->
    {ok, Req2} = cowboy_http_req:reply(200, [], <<"Hi EUC!">>,
        Req),
    {ok, Req2, State}.

terminate(_Req, _State) ->
    ok.
```

HTTP request object

- Retrieve the method, HTTP version, peer IP and port
- Retrieve the requested hostname and path
- Retrieve headers, query string values, bindings, cookies
- Semantically parse headers
- Read the body of the request
- Send a simple or a chunked reply

HTTP handler loops

- Useful for long-polling or Event Source
- Receive messages from other processes and send back data
- Hibernate and timeouts support

HTTP handlers for long-polling

```erlang
-module(my_loop_handler).
-export([init/3, info/3, terminate/2]).

init(_TransportType, Req, _Opts) ->
    my_session_server:hello(),
    {loop, Req, undefined_state, 60000, hibernate}.

info({my_session_server, Message}, Req, State) ->
    {ok, Req2} = cowboy_http_req:reply(200, [], Message, Req),
    {ok, Req2, State};
info(_Any, Req, State) ->
    {loop, Req, State, hibernate}.

terminate(_Req, _State) -> ok = my_session_server:bye().
```

Websocket support

- Cowboy supports all Websocket protocol versions in use in web browsers
- New versions get added as soon as browsers start implementing them
- Cowboy’s websocket interface should be future-proof
- Websocket connections are on a different pool than normal HTTP

Websocket handlers

```
-module(my_ws_handler).
-export([init/3, websocket_init/3, websocket_handle/3,
    websocket_info/3, websocket_terminate/3]).

init(_TransportType, Req, _Opts) ->
    {upgrade, protocol, cowboy_http_websocket}.

websocket_init(_TransportType, Req, _Opts) ->
    Req2 = cowboy_http_req:compact(Req),
    {ok, Req2, undefined_state, 60000, hibernate}.

%% ...
```

Websocket handlers continued

```
%% ...

websocket_handle({text, Data}, Req, State) ->
    {reply, {text, Data}, Req, State, hibernate};
websocket_handle(_Frame, Req, State) ->
    {ok, Req, State, hibernate}.

websocket_info(_Info, Req, State) ->
    {ok, Req, State, hibernate}.

websocket_terminate(_Reason, _Req, _State) ->
    ok.
```

REST handlers

- Use Webmachine’s decision flow diagram
- Not a rewrite, a new implementation
- Clean, readable code not requiring the diagram to be understood

Cowboy is clean code

- Easy to understand, easy to debug
- No process dictionaries or other side effects
- No parameterized modules
- Use only documented Erlang/OTP features

Cowboy and OTP

- All Cowboy processes are supervised
- OTP upgrades work (though improvements are coming)
- HTTP handlers are inspired by gen_servers
  - gen_server: client > api call > server
  - http_handler: http client > http request > http handler

Cowboy’s performance

- 500 000 concurrent websocket connections and beyond

Does performance matter?

- It depends on your application
- It probably doesn’t matter for 99% of the applications
- In HTTP, response latency is very important
- When handling many concurrent connections, memory usage matters

1 process per connection

- Other Erlang HTTP servers use 2 processes per connection
- Cowboy uses only 1
- Saves a significant amount of memory
- Reduces latency thanks to reduced message passing

Low memory usage: binary

- Big binaries are ref counted
- Small binaries are still smaller than lists
- Sub-binary optimizations helps reduce copying

Low memory usage: cowboy_http_req:compact/1

- Removes everything unwanted from the Req object
- Lowers memory usage for long-running processes
- Works especially well with process hibernation

Requests per seconds

- Not a good indicator of performance
- But a good indicator of a design’s simplicity
- If requests/s lowers significantly between two commits, you’ve messed up

The Horse project

- Continuous performance testing of the Cowboy project
- Will measure latency, CPU usage, memory usage…
- Will produce nice graphs to quickly notice drops and fix them
- Release expected for Q1 2012

Cowboy’s related projects

- Because the core project should stay lightweight

Bullet handler

- A Socket.IO/SockJS alternative
- Sets up an always connected streaming interface between JS and Erlang
- Uses Websocket by default, other methods when not available
- A single interface both client and server-side
- More work is needed on the JS side

cowboy_static handler

- A static file handler by Magnus Klaar
- Using the sendfile code originally from Yaws
- Will be using the future REST support
- Available as a separate project

Bigwig: Spawnfest great winner

- Spawnfest is an annual Erlang programming contest
- First edition took place in June 2011
- They produced an awesome webtool replacement with many more features
- Got the IRCCloud guys interested in Cowboy

Farwest: a new kind of web development stack

- Default administration panel
  - Setup your data
  - Setup your views
  - Write simple rules to route your data to your views
  - Write simple rules to create forms and save the data
  - Edit and reload any client and server-side code live
  - Edit and reload the dispatch list live
  - Git integration
- NoSQL backend, nice development API and many plugins
- Beta expected for Q1 2012

Cowboy’s future

- Because they aren’t just figures of the past

Listener upgrades

- We want to update the dispatch list on-the-fly
- We want to update most transport or protocol options
- Also allow adding or removing acceptors

Improved dispatcher

- API will be improved
- Allow giving tokens as lists and not just binaries
- Hostname and path hierarchy will be added
- Might eventually switch to matchspecs for better performance

gen_event for request tracking and monitoring

- Ad-hoc error and access logging
- Allows writing custom event handlers
- No cost when no handlers are defined

Multipart support

- File upload support
- Convenience function
  - Save all files to temporary locations
  - Return the paths along with the additional POST parameters
- Streaming
  - Stream each parts individually
  - Stream each parts’ payload

Aiming for full HTTP/1.1 compliance

- Support for all HTTP/1.1 features
- Correct HTTP/1.0 clients handling
- Semantic parsing of all header values
- Gzip compression enabled by default for replies

Up-to-date Websocket support

- We are actively monitoring the Websocket draft changes
- Updated implementation usually supported within 7 days
- Interface shouldn’t be changing anymore

Links

- cowboy: https://github.com/extend/cowboy
- bullet: https://github.com/extend/bullet

- essen on #erlounge and #erlang on Freenode
- @lhoguin on Twitter
- Loïc Hoguin on G+
- essen@dev-extend.eu

Questions?

- I won’t shoot you for asking!