WRITING PLUGINS WITH RABBITMQ

Noah Gift and Michael Vierling
Noah’s Background

- Film
- Commercial Data Center Automation Software
- Los Angeles, Atlanta, New Zealand, San Francisco
- I like programming in ANY language that is innovative
How & Why

- Why AMQP?
- Why Erlang?
- Why OTP?
- Why RabbitMQ?
- How – Machinery
Why AMQP

- Industry Message Standard
- Cisco, Microsoft, Red Hat, VMWare, JPMorgan, Chase, Goldman Sachs
- Binary wire protocol
Why AMQP: Continued

- Key component in Distributed Architecture
- Enables Multi-OS, Multi-Language
- Reliable, Transactional, Flow-Control, Fault-tolerant
Downsides of AMQP

- AMQP 1.0 is almost final
- Support resources
- Upgrade glitches
- Monitoring issues
- Investment in the future
Why Erlang?

- Functional Language
- Immutable Data
- Scalable
- Reliable, Fault-tolerant
Why RabbitMQ

- Built with Erlang
- 10K messages per second – transient mode
- 3-5K messages per second – persistent mode
Why RabbitMQ

- Transactions
- Open Source
- Maintained by VMWare
Broker Infrastructure
Broker Infrastructure

- Repeatable Install
- Reliable storage
- Redundant hardware
Broker Infrastructure: Continued

- Cluster of nodes
- Disaster Recovery
- Logging (such as Splunk) and monitoring
- Operations & Support
Broker Infrastructure: Use Escrip

```erlang
#!/usr/local/bin/escrip

get_os () ->
    os:cmd("uname").

main(_) ->
    io:format("OS Platform: ~p~n", [get_os()]).
```

1,1
Broker Infrastructure Future?

- Integrate Rebar
- Continuous Integration Testing
- QuickCheck
RabbitMQ Clients

- Any AMQP Clients (Apache)
- Ruby
- Python
- C#
- F# (Wrap the C# dll)
```python
import pika
from pika.adapters import SelectConnection

user = 'guest'
password = 'guest'
rabbit_node = 'rabbit9.np.wc1.yellowpages.com'
rabbit_queue = 'test_queue'
# import pdb; pdb.set_trace()
# Create a global channel variable to hold our channel object in
channel = None

# Step #2
def on_connected(connection):
    
    # Called when we are fully connected to RabbitMQ
    
    # Open a channel
    connection.channel(on_channel_open)
```

# Step #3
def on_channel_open(new_channel):
    """Called when our channel has opened"""
    global channel
    channel = new_channel
    channel.queue_declare(queue=rabbit_queue, durable=False,
                           exclusive=False, auto_delete=False, callback=on_queueDeclared)

# Step #4
def on_queueDeclared(frame):
    """Called when RabbitMQ has told us our Queue has been declared, """
    """frame is the response from RabbitMQ"""

    print channel._dict_
    print handle_delivery(1,2,3,4)
    channel.basic_consume(handle_delivery, queue=rabbit_queue)
def handle_delivery(channel, method, header, body):
    print("Called when we receive a message from RabbitMQ")
    print("I was called")
    print("Body", body)

# Step #1: Connect to RabbitMQ
import pdb; pdb.set_trace()
parameters = pika.ConnectionParameters(host=rabbit_node, virtual_host='/')
connection = SelectConnection(parameters, on_connected)

try:
    # Loop so we can communicate with RabbitMQ
    connection.ioloop.start()
    print(\n    except Exception, e:
        print("I had an exception: %s" % e)
        # Gracefully close the connection
        connection.close()
        # Loop until we're fully closed, will stop on its own
        connection.ioloop.start()
Send.py Part: A

```python
# Step #5

def handle_delivery(channel, method, header, body):
    # Called when we receive a message from RabbitMQ
    print "I was called"
    print "Body", body

# Step #1: Connect to RabbitMQ
#import pdb; pdb.set_trace()
parameters = pika.ConnectionParameters(host=rabbit_node, virtual_host='/')
connection = SelectConnection(parameters, on_connected)

try:
    # Loop so we can communicate with RabbitMQ
    connection.ioloop.start()
    print connection
except Exception, e:
    print "I had an exception: %s" % e
    # Gracefully close the connection
    connection.close()
    # Loop until we're fully closed, will stop on its own
    connection.ioloop.start()
```
# Open a connection to RabbitMQ on localhost using all default parameters
connection = pika.BlockingConnection( parameters )

# Open the channel
channel = connection.channel()

# Declare the queue
channel.queue_declare(queue=rabbit_queue, durable=False,
                        exclusive=False, auto_delete=False)

channel.basic_publish(exchange='', routing_key=rabbit_queue,
                      body="Hello World!",
                      properties=pika.BasicProperties(content_type="text/plain",
                                                     delivery_mode=1))
Writing RabbitMQ Plugin: WHY

- Access internal RabbitMQ functionality
- Running in same Erlang VM as broker can increase performance
- Leverage your existing infrastructure
Writing RabbitMQ Plugin: WHY NOT

- Developers don’t know Erlang
- Poorly written plugin could crash broker
- You could lock yourself to internal API
Stable RabbitMQ Plugin

- Rabbitmq-management (replaced Alice)
- Rabbitmq-shovel
- Rabbitmq-stomp
- Rabbitmq-erlang-client
Experimental RabbitMQ Plugin Ideas

- Rabbitmq-auth-backend-ldap
- Rabbitmq-auth-mechanism-ssl
- Rabbitmq-jsonrpc-channel
- Rabbitmq-xmpp
Crazy RabbitMQ Plugin Ideas

- Distributed Data Structure
- Artificial Intelligence
- Data Mining
- Create a protocol on top
- Event Processing
Michael Background

- Apple 5 Years
- Ascend/Lucent 4 Years
- Various Startups
Supervision Tree
Plugin Supervisor
Rabbit RSS Supervisor

```erlang
-module(rabbit_rss_sup).
-behaviour(supervisor).
-export([start_link/0, init/1]).

start_link() ->
    supervisor:start_link([{local, ?MODULE}, ?MODULE, _Arg = []]).

init([]) ->
    % We allow 3 restarts within a period of 10 seconds
    ok, [{one_for_one, 3, 10},
         [{rabbit_rss_worker, start_link, []},
          permanent,
          10000,
          worker,
          [rabbit_rss_worker}]].
```
-module(rabbit_rss_worker).
-behaviour(gen_server).

-including("xmerl/include/xmerl.hrl").
-export([start/0, start/2, stop/0, stop/1, start_link/0]).
-export([init/1, handle_call/3, handle_cast/2, handle_info/2, terminate/2, code_change/3]).

-including("amqp_client/include/amqp_client.hrl").

-record(state, {channel}).

rss_url() ->
    "http://earthquake.usgs.gov/earthquakes/catalogs/eqs1day-M0.xml".

test_retry_interval() ->
    20 seconds
Rabbit RSS Worker: Slide B

```erlang
code

retry_interval() ->
  % 30 minutes
  1800000.

start() ->
  start_link(),
  ok.

start(normal, []) ->
  start_link().

stop() ->
  ok.

stop(_State) ->
  stop().
```

Rabbit RSS Worker: Slide C

```erlang
start_link() ->
  gen_server:start_link({global, ?MODULE}, ?MODULE, [], []).

%--------------------------------
% Gen Server Implementation
%--------------------------------
init([]) ->
  error_logger:info_report("worker:init"),
  application:start(inets),
  {ok, Connection} = amqp_connection:start(direct),
  {ok, Channel} = amqp_connection:open_channel(Connection),

% Declare a queue
#'{queue.declare_ok}{queue = Q} = amqp_channel:call(Channel, #'queue.declare{queue = <<"test_queue">>>}),
```
Rabbit RSS Worker: Slide D

timer:apply_after(retry_interval(), gen_server, call, [{global, ?MODULE}, fire]),
{ok, state{channel = Channel}}.

handle_call(Msg, _From, State = #state{channel = Channel}) ->
  case Msg of
    fire ->
      error_logger:info_report("rabbit_rss_worker:handle_call fire"),
      % Retrieve XML from RSS feed
      { ok, { _Status, _Headers, Body }} = httpc:request(rss_url()),
      % Parse the XML into a dedup'd list of individual items
      MList = rss:parse(Body),
% Publish each XML item into an AMQP queue
[amqp_publish( Channel, X ) || X <- MList],

% Set our timer
timer:apply_after(retry_interval(), gen_server, call, [{global, ?MODUL E}, fire]),
error_logger:info_report("rabbit_rss_worker:handle_call fire OK"),
{reply, ok, State};

- error_logger:error_report("rabbit_rss_worker:handle_call fire NOT OK")
, {reply, unknown_command, State}
end.

amqp_publish( Channel, Message ) ->

Export=xmerl:export_simple([Message],xmerl_xml),

Rabbit RSS Worker: Slide F

```erlang
BasicPublish = #"basic.publish"{exchange = <<>>, routing_key = <<"test_queue">>, Content = #amqp_msg{props = Properties, payload = XMessage}, amqp_channel:call(Channel, BasicPublish, Content).

handle_cast(_,State) ->
  {noreply,State}.

handle_info(_,State) ->
  {noreply, State}.

terminate(_,#state{channel = Channel}) ->
  amqp_channel:call(Channel, #"channel.close"{}), ok.

code_change(_,State, _Extra) ->
  {ok, State}.
```
XML Parsing
-module(rss).
-export([parse/1]).
-include_lib("xmerl/include/xmerl.hrl").
-include_lib("eunit/include/eunit.hrl").

hexstring(<<X:128/big-unsigned-integer>>>) ->
  %io:format("~32.16.0b\n", [X]).
  lists:flatten(io_lib:format("~32.16.0b", [X])).

itemExists(E) ->
  Binary = term_to_binary(E),
  Md5 = erlang:md5(Binary),
  S = hexstring(Md5),
  R = get(S),
  case R of
    undefined ->
parse(Buffer) Buffer contains XML
% returns a list of extracted XML strings
  parse(Buffer) ->
    % parses a single RSS string XML buffer
    error_logger:info_report("rss:parse"),
    {R,[]} = xmerl_scan:scan_string(Buffer),
    ItemList = lists:reverse(parseElementsList(R)),
    L = [X || X <- ItemList, not(itemExists(X))],
    L.

parseElementsList([H|T]) when H#XmlElement.name == title ->
  Export = xmerl:export_simple([H], xmerl_xml),
  XML = lists:flatten(Export),
  [XML | parseElementsList(T)];
parseElementsList([H|T]) when is_record(H, xmlElement) ->
  parseElementsList(H#xmlElement.content) ++ parseElementsList(T);
parseElementsList(X) when is_record(X, xmlElement) ->
    parseElementsList(X#xmlElement.content);
parseElementsList([]T) ->
    parseElementsList(T);
parseElementsList([]) ->
    [].

rss0_test() ->
    {ok, Buffer} = file:read_file("digg-science-rss2.xml"),
    MList = parse(binary_to_list(Buffer)),
    MList.
    %[H |_] = MList,
    %MList.

rss1_test() ->
    {ok, Buffer} = file:read_file("digg-science-rss1.xml"),
    MList = parse(binary_to_list(Buffer)),
    MList.
rss2_test() ->
    rss1_test(),
    {ok, Buffer} = file:read_file("digg-science-rss2.xml"),
    MList = parse(binary_to_list(Buffer)),
    % io:format("length = ~p~n", length(MList) ),
    % io:format("List = ~p~n", MList ),
    ?assertEqual( 40, length(MList) ) .

rss3_test() ->
    {ok, Buffer} = file:read_file("slash.xml"),
    MList = parse(binary_to_list(Buffer)),
    % io:format("length = ~p~n", length(MList) ),
    ?assertEqual( 15, length(MList) ) .
Checkout source: HG clone

```
lion% hg clone http://hg.rabbitmq.com/rabbitmq-public-umbrella
```
Make: Be Patient....
Build

```
lion% build
```
Skeleton

```
lion% vim Makefile
lion% mkdir rabbitmq-rss
lion% mkdir rabbitmq-rss/ebin
lion% mkdir rabbitmq-rss/src
lion% cd rabbitmq-rss
lion% vi ebin/rabbit_rss.app
lion% vi src/rabbitmq_rss.erl
lion% vi src/rabbitmq_rss_sup.erl
lion% vi src/rabbitmq_rss_worker.erl
lion% vi src/rss.erl
lion% 
```
Symlink Plugin

```
lion% pwd
/tmp/test/rabbitmq-rss
lion% cd ..
lion% mkdir -p rabbitmq-server/plugins
lion% cd rabbitmq-server/plugins
lion% ln -s ../../../rabbitmq-erlang-client
lion% ln -s ../../../rabbitmq-rss
```
Run Broker

$ make run
(rabbit@rabbit9)1>
(rabbit@rabbit9)1>application:which_applications().

{rabbit_rss,"Embedded Rabbit RSS Reader","0.01"},
$ mkdir test
$ vi test/rabbit_rss_tests.erl
$ vi Makefile

TEST_APPS=amqp_client rabbit_rss
TEST_COMMANDS=rabbit_rss_tests:test()
START_RABBIT_IN_TESTS=true

$ make test
Package Plugin

```
lion% make package; ls dist/* .ez
```
Plugin Supporting Machinery

- Unit Tests
- Source Code
- Logger
- Printf
- AMQP Clients
How To Build a Plugin

- RabbitMQ docs
- Source
- Examples
Use the Source!

- Documentation is hit/miss
- Google “erlang lists”
- http://www.erlang.org/doc/apps/stdlib/
- Or skip the docs and go directly to source
- ./lib/stdlib/src/lists.erl
%% last(List) returns the last element in a list.

-spec last([T,...]) -> T.
last([E|Es]) -> last(E, Es).
last(_, [E|Es]) -> last(E, Es);
last(E, []) -> E.
Logger

- Plugin executes in its own process
- No plugin stdout
- No plugin execution debugger
Logger: Continued

- Solution: Erlang’s logger
- Output to /var/log/rabbitmq/<hostname
- Nothing to configure
parse(Buffer) Buffer contains XML
%% returns a list of extracted XML strings
parse(Buffer) ->
  %% parses a single RSS string buffer
  error_logger:info_report("rss:parse"),
  {R,_,_} = xmerl_scan:string(Buffer),
  ItemList = lists:reverse(parseElementsList(R)),
  L = [X || X <- ItemList, not(itemExists(X))],
  L.
OTP Design Principles

- Supervision Tree
- Behaviors
- Applications
- Releases
- Release Handling
Machine Learning

- The next step for the RSS plugin.
- A potential use case for RabbitMQ plugins.
Fisher Classifier For RSS Feeds

- Data mining
- Classification
Fisher Classifier: Slide A

```erl
-module(intel).
-export([getwords/1]).
-import(lists,[reverse/1,member/2]).
-import(dict).

#include_lib("eunit/include/eunit.hrl").

%% mult(L) returns the all items multiplied together
%%-spec mult([number()]) -> number().

mult(L) -> mult(L, 1).
mult([H|T], Total) -> mult(T, Total * H);
mult([], Total) -> Total.
```
Fisher Classifier: Slide B

```python
fisherProb(Classifier, Doc, Cat) ->
    FList = getwords(Doc),
    FeatureProb = fun cProb/3,
    ProbCounts = [ probDocCat(Classifier, X, Cat, FeatureProb) || X <- FList ],
    P = mult(ProbCounts),
    FScore = -2 * math:log( P ),
    ICS = inverseChiSquared( FScore, length(FList) * 2 ),
    ICS.
```
Fisher Classifier: Slide C

```erlang
chiSquared( _M, I, Max, _Term, Sum ) when I >= Max ->
    Sum;
chiSquared( M, I, Max, Term, Sum ) ->
    io:format("Term: ~p ~n", [Term]),
    io:format("Sum: ~p ~n", [Sum]),
    T2 = Term * (M / I),
    S2 = Sum + T2,
    chiSquared( M, I + 1, Max, T2, S2 ).
```
inverseChiSquared(Chi, DF) ->
  io:format("Chi: ~p ~n", [Chi]),
  io:format("DF: ~p ~n", [DF]),
  M = Chi / 2.0,
  Sum = Term = math:exp( -M ),
  R = chiSquared( M, 1, (DF div 2), Term, Sum ),
  min( R, 1.0 ).
Fisher Classifier: Slide E

cProb(Classifier, Doc, Cat) ->
ProbDocCat = featureProb(Classifier, Doc, Cat),
case ProbDocCat of
  0 -> 0;
  _ ->
  CatList = catList(Classifier),
  FreqCounts = [ featureProb(Classifier, Doc, X) || X <- CatList],
  FreqSum = lists:sum(FreqCounts),
  Prob = ProbDocCat / FreqSum,
end.
Fisher Classifier: Slide E

cProb(Classifier, Doc, Cat) ->
ProbDocCat = featureProb(Classifier, Doc, Cat),
case ProbDocCat of
  0 -> 0;
  _ ->
      CatList = catList(Classifier),
      FreqCounts = [ featureProb(Classifier, Doc, X) || X <- CatList],
      FreqSum = lists:sum( FreqCounts ),
      Prob = ProbDocCat / FreqSum,
      Prob
end.
Fisher Classifier: Slide F

classify(Classifier, Doc, Threshold) ->
  CatList = catList(Classifier),
  PList = [ {bayesProb(Classifier, Doc, X), X} || X <- CatList],
  lists:reverse( lists:sort( PList ) ).

bayesProb(Classifier, Doc, Cat) ->
  CatProb = catTotal(Classifier, Cat) / docTotal(Classifier),
  FeatureProb = fun featureProb/3,
  DocProb = probDocCat(Classifier, Doc, Cat, FeatureProb),
  Prob = DocProb * CatProb,
  Prob.
Fisher Classifier: Slide G

% Find probability of Doc, given Category
probDocCat(Classifier, Doc, Cat, FeatureProb) ->
  WordList = getwords( Doc ),
  ProbCounts = [ weightedProb(Classifier, Word, Cat, FeatureProb, 1.0, 0.5) || Word <- WordList ],
  mult( ProbCounts ).

docTotal(Classifier) ->
  CatList = dict:to_list(Classifier),
  docTotal(CatList, 0).

docTotal([H|T], Sum) ->
  {_Key, {_Dict, CatCount}} = H,
  docTotal(T, Sum + CatCount);
  docTotal([], Sum) -> Sum.
Conclusion

- AMQP
- Erlang
- Python
- RSS
Questions

- Code: [https://github.com/mikev/rabbitmq_rss_plugin](https://github.com/mikev/rabbitmq_rss_plugin)
- Contact: "Noah Gift" noah.gift@gmail.com, "Michael Vierling" mvierling@gmail.com