Zero Support In Production

An Erlang Success(ish) Story (by @robashton)
Id3as

The company I work for
Bespoke Media Workflows

(That’s what we build)
System Schematic
The kind of thing we do

- On demand encode jobs + CDN Delivery
- Live web-based video editing of events
- Remote commentary via WebRTC (sportsballs)
- External Audio contributions to live video events
- Various combinations of all of these (every customer is a special snowflake)
- 100s of 1000s of events every year...
The Programming Team
The Support Team
Build New Stuff

Without supporting old stuff
Our Approach to...
Our Approach To... CODE
Code Quality

All UIs written in Elm = No UI Errors Ever
-record(container, { value }).

add_these(#container { value = X }, #container { value = Y }) -> #container { value = X + Y }.
-record(container, {  
    value  
  }).

add_these(#container {  value = X  },  
    #container {  value = Y}) ->  
    #container {  value = X + Y  }.

-export([  
    two_negative_numbers_output_negative_numbers/0,  
    two_positive_numbers_output_positive_numbers/0,  
    etc_what_a_boring_example/0  
]).
-record(container, {  
    value :: integer()  
}).

-type container :: #container{}.

%%% @doc Adds two containers together and returns a new container with  
%%% the new values  
-spec add_these(container(), container()) -> container().
add_these(#container { value = X },
    #container { value = Y}) ->
    #container { value = X + Y }.

-export([  
    two_negative_numbers_output_negative_numbers/0,  
    two_positive_numbers_output_positive_numbers/0,  
    etc_what_a_boring_example/0  
]).
Nearly all code gets thrown away

... So there is little point worrying about quality the first time around
Our Approach To Code

- Write code to be thrown away
- Re-use everything
- Don’t use third party code
- Except when third party code is better
- OWN third party code
- Don’t re-use EVERYTHING
Our Approach To... SYSTEM DESIGN
Identify Your Absolute Truths
Absolute truth

- Usually external data that the client owns
- Can come via calls to their systems
- Often practical to just come as a spreadsheet...
- Can come via some sort of mgmt UI
- Is essentially an API

We don’t own this data and we don’t modify this data
From this truth..

We can re-build all of our state from scratch
Examples..

Channels (input addresses, output addresses, output encodes)

Events (Start time, end time, source to use, outputs..)
Identify Jobs

We should be able to generate all the config from the Absolute Truth
Jobs...

- Self contained
- Single purpose
- Single units of failure
- Blind + Autonomous
Essentially..
1) Controller Receives Absolute Truth
2) Controller Assesses Current Situation
3) Controller Generates Jobs + Config
4) Controller PUSHES Config
5) Controller PULLS Status
6) Goto #2
Truth changes

Controller

Job

???!

Job

Job

Job

Job
This is just a pattern
And is everywhere
One way data flow

...\n
Looks like something else..
Elm Architecture
import Html exposing (..)

-- MODEL

type alias Model = { ... }

-- UPDATE

type Msg = Reset | ...

update : Msg -> Model -> Model
update msg model =
  case msg of
    Reset -> ...
    ...

-- VIEW

view : Model -> Html Msg
view model =
  ...
This works so long as infra works..

But infra never works
Our Approach to... FAILURE
In reality..

- Netsplits
- DNS failure
- Bugs in jobs (yes, things crash!)
- Disk failure
- Human error
Inside An Erlang application this is automatic..
-module(prf_encoder_deferred_sup).
-include("./include/api.hrl").
-export([start_link/0]).
-export([init/1]).

define(CHILD(I, Type), {I, {I, start_link, []}, permanent, 5000, Type, [I]}).

start_link() ->
    supervisor:start_link({local, ?MODULE}, ?MODULE, []).
init([], []) ->
    #prf_commentary_setup {
        contribution = Contribution
    } = prf_encoder_dynamic_config:commentary_session(),

    {ok, [one_for_all, 10, 10], [
        ?CHILD(prf_encoder_workflow_transcode, worker),
        ?CHILD(prf_encoder_commentary_session, worker),
        case Contribution of
            #mta_contribution{} ->
                ?CHILD(prf_encoder_udp_commentary_workflow, worker);
            #wab_contribution{} ->
                ?CHILD(prf_encoder_web_commentary_workflow, worker)
        end
    ]}.}.
Between Erlang Applications.

Controller

HTTP GET

erlang:monitor(Pid)

Job

{ pid: "<0.232.1>" }

/status

Job

NET_TICKTIME=1
Avoiding zombies
A common scenario
Keep-alives on EVERYTHING

... 

Trust that nothing will work because everything will break
handle_info(keep_alive, State) ->
{ noreply, reset_timer(State) };

handle_info(keep_alive_expired, State = #state { id = Id }) ->
?WARNING("Cripes, this all went horribly wrong didn't it ~p", [ Id ]),
ok = job_store:delete(Id),
{ stop, normal, State }.
Controllers Should Prune Obsolete Config

```erlang
handle_info({keep_alive, Id}, State = #state { configs = Configs }) ->
    NewConfigs = touch(Id, Configs),
    {noreply, State#state { configs = NewConfigs }};

handle_info(tick, State = #state { configs = Configs }) ->
    NewConfigs = prune_stale_configs(Configs),
    {noreply, State#config { configs = NewConfigs } };
```
Keep hitting the world with a hammer

```erlang
handle_info(tick, State = #state { config = Config }) ->
  hammer_that_config_downstream(Config),
  {noreply, State}.
```
Let it crash?

- Not always
  - (Unexpected message? Sure thing)
  - Can’t find a file you expected? Perhaps handle this - business context dependent
Our Approach To... DEVELOPMENT

... Working on all of this locally..
Emulate Everything
An example
Mirror your production envs locally

- Fewer surprises when you deploy (even to a dev env)
- Faster iteration
- Easier for other team members to just “jump in”
Our Approach To... TESTING
Unit testing

... 

On established core technology only
For example - Auctioning
Integration testing

...meh
Chaos Monkey Abuse

Much more interesting
stress_worker(Channels, MaxEvents, MaxCommentary) ->
mutate_system(Channels, MaxEvents, MaxCommentary),

    receive
        stop -> ok
    after 30000 ->
        stress_worker(Channels, MaxEvents, MaxCommentary)
    end.

mutate_system(Channels, MaxEvents, MaxCommentary) ->
    ensure_channels_running(Channels),
    maybe_start_events(Channels, MaxEvents, MaxCommentary),
    maybe_adjust_mta(Channels, MaxEvents, MaxCommentary),
    maybe_terminate_channels(),
    maybe_terminate_sessions(),
    maybe_terminate_events(),
    maybe_terminate_encoders(),
    maybe_reset_nodes().
We release to prod if...

- The chaos monkey runs all weekend AND
- No zombies
- No crashes
- No failed jobs
- Quitting the chaos monkey means the system springs back to life fully

It’s all about TRUST
TLDR:

- Detailed testing of core assets is a must
- Full end to end testing of entire system is a must
- Most things in-between don’t tend to add value
- New code is likely wrong and going to be deleted anyway
Our Approach To... Monitoring
These are big systems

- 100s of nodes
- 1000s of moving parts
- Lot of things that can go wrong
- How do we surface them?
The darling children

- InfluxDB
- Riemann
- Grafana
Shiny Graphs
Nobody looks at graphs

- Superb sales tool
- Customers love them in demos
- We absolutely do all this work and show it to everybody who cares
- They’re just vanity metrics though...
Flags

- Every node can post to [http://localhost/flags](http://localhost/flags)

```
{
    "job_id": "packager-1337",
    "msg": "Discontinuity on source { 5001, { 231, 11, 41, 1 } }"
}
```
Visible in Controller UI

Channel: 239.11.221.1 (CH01)

Actions

DELETE  RESTART  DISABLE  CONFIGURE

Assignation
Encoder: 2017-11-21T12:02:23 (resource-daeb33f0-99bf-4dd3-a4f1-03a231b6e841-1)

Input
239.11.221.1:10000

Status
No traffic on source capture
2017-11-21 12:12:33 (10288 times since 7h ago)

Internal Outputs
239.11.222.1:5001 (190x108@112kbs / 48kbs)
239.11.222.1:5002 (384x216@288kbs / none)
239.11.222.1:5003 (512x288@400kbs / none)
239.11.222.1:5004 (704x396@675kbs / none)
239.11.222.1:5005 (704x396@1380kbs / 64kbs)
239.11.222.1:5006 (940x540@2500kbs / 128kbs)
239.11.222.1:5007 (1280x720@3000kbs / 128kbs)
239.11.222.1:5008 (1920x1080@5000kbs / 128kbs)
Media Workflow Diagrams
Highlight the problems

... Everything else is noise
Our Approach To... Disaster
Things still go wrong

- Unallocated jobs (WHHYYY?)
- Jobs with bad state after a crash
- State state state
The “Fix it yourself” pattern

CommentarySession: MTA: 239.11.221.1 / clean

Assignation
Node: 10_23_203_106 (resource-5ce1e29f-d002-491a-b8db-feb7cc2de1cb-1)

Actions
TERMINATE   RESTART   VIEW LOGS

Assignation
Node: 10_23_203_100 (resource-f17387f7-72c8-4988-b40f-9e0870f72439b-1)

Actions
TERMINATE   VIEW LOGS

Assigned Channels
Encoder: Encoder_fermare_v0.10.101-dev
239.11.221.2 (running)   VIEW

Channel: 239.11.221.1 (CH01)

Assignation

Packager:
prf_packager_v0.10.101-dev deployed to 10_23_203_108:2   RESTART

Actions
DELETE   RESTART   DISABLE   CONFIGURE   CONFIGURE MTA   VIEW LOGS

Node: 10_23_203_100 (localhost)

Actions
RESTART   VIEW LOGS

Settings
Allow Allocations   SAVE

Allocations
Encoder (2018-05-03T10:46:26)   VIEW
- 239.11.221.2
We can do this because

- Absolute source of truth isn’t broken (99.99% of the time)
- Our local state is what is broken most likely
- Delete all our local state, re-create it
- Profit
Plans?

Start writing everything in purescript - Thanks Nick! --->

https://github.com/purerl/purescript
-spec start_link( channel(), update_fn() ) -> { atom(), state() }.

start_link(Channel, Update) ->
gen_server:start_link([{local, ?MODULE}], ?MODULE, [ Channel, Update ], []).

init([ Channel, Update ]) ->
lager:debug("Started Channel ~p", [ Channel ]),
?SUBSCRIBE_BUS_MESSAGES([ #encoder_job_allocated_bus_msg{},
                         #encoder_job_lost_bus_msg{}
                     ]).
State = #state { channel = Channel, update = update },
{ ok, schedule_tick(1000, State) }

handle_info(tick, State = #state { channel = Channel, update = Update }) ->
  NewState = case do_updates(Channel) of
    undefined -> State;
    NewChannel -> update(NewChannel), State#{channel = NewChannel }
  end,
  {noreply, schedule_tick(?TICK_INTERVAL, NewState )};

handle_info(#encoder_job_allocated_bus_msg { id = EncoderJobId }, State = #state { channel = #channel { id = Id } }
  where Id == EncoderJobId -> {noreply, schedule_tick(0, State )};

handle_info(#encoder_job_lost_bus_msg { id = EncoderJobId }, State = #state { channel = #channel { id = Id } }
  where Id == EncoderJobId -> {noreply, schedule_tick(0, State )};

handle_info(_IgnoredMessage, State) -> {noreply, State }
let state = { channel
, tickRef : Nothing
, update }
newState <- scheduleTick 1000 state
pure $ newState

handleTick :: State -> Effect State
handleTick state@( channel, update ) = do
    channelToMaybeSave <- doUpdates channel
    _ <- maybe (pure unit) update channelToMaybeSave
    let newState = maybe state ((\c -> (state { channel = c })) channelToMaybeSave)
scheduleTick tickInterval newState

handleEncoderJobEvent :: EncoderJobEvent -> State -> Effect State
handleEncoderJobEvent (EncoderJobReady jobId) state@(channel)
    | (encoderJobId channel) == jobId = scheduleTick 0 state
    | otherwise = pure state

handleEncoderJobEvent (EncoderJobLost jobId) state@(channel)
    | (encoderJobId channel) == jobId = scheduleTick 0 state
    | otherwise = pure state
I can talk about our stuff till the cows come home but I suspect by this point I’ve rather ran out of time...
What about...

- Kubernetes
- Nomad
- Amazon Fargate
- Terraform
- Ansible
- Etc

These are very generic solutions

This is Core Business for us

In a few years, the “winner” may well be the answer

For now, they tend to add complexity outside of our control, so thanks but no thanks