RELX

A Dead Simple Way to Build Releases

★ ★ ★ ★

★ ★ ★ ★

Topics

* The basics of OTP Applications and Releases
* Why Releases are important
* How Relx builds Releases
* How different Relx options affect the built Release
* How to extend Relx

Goal

Walk out of this talk with the ability to create and use releases.

What is Relx

* Tool designed to build releases
* Designed to integrate into a unix like suite of build tools

OTP Application Refresher

Basic Building Blocks of Erlang Systems

* Well defined structure
* Well defined lifecycle (start, stop, semantics)
* Useful metadata (including an explicit dependency graph)

Well Defined Structure

```
├── <name>-<version>
    ├── ebin
    |   └── <name>.app
    ├── doc
    ├── priv
    ├── include
    └── src
```

Well Defined Lifecycle

```
-module(echo_get_app).
-behaviour(application).

%% API.
-export([start/2,
         stop/1]).

-include("echo_get.hrl").

%%%===================================================================
%%% API
%%%===================================================================
start(_Type, _Args) ->
    Dispatch = cowboy_router:compile(?DISPATCH),
    {ok, _} = cowboy:start_http(http, 100,
                                [{port, 8080}],
                                    [{env,
                                      [{dispatch, Dispatch}]}]),
    echo_get_sup:start_link().

stop(_State) ->
        ok.
```

Useful Metadata

```
{application,echo_get,
             [{description,"Cowboy GET echo example."},
              {vsn,"0.0.0+build.1.ref5b05dba"},
              {modules,[echo_get,
                        echo_get_app,
                        echo_get_handler,
                        echo_get_sup]},
             {registered,[]},
             {applications,[kernel,
                            stdlib,
                            ranch,
                            cowboy]},
             {mod,{echo_get_app,[]}},
             {env,[]}]}.
```

How We Start Applications

```
-module(echo_get).

%% API.
-export([manual_start/0]).

%%%===================================================================
%%% API
%%%===================================================================
manual_start() ->
    ok = application:start(crypto),
    ok = application:start(ranch),
    ok = application:start(cowboy),
    ok = application:start(echo_get).
```

How We Start Applications

```
#!/bin/sh
erl -noshell -pa ebin deps/*/ebin -s echo_get manual_start \
        -eval "io:format(\"Point your browser at http://localhost:8080/?echo=test~n\")."
```

What Do We Have?

* An Erlang System where dependency information is hap hazard (in at least two places, one of which is ignored)

* A system that trusts the programmer to manually startup things in the right order (without leaving anything out)

* A system where its component parts are spread around the OS filesystem (if we are lucky and they are even there)

Current State of the World

* Lots of systems are assembled poorly (ie, poorly organized, not using the right abstractions, prone to failure)

* Commonly deploy by simply manually copying the Apps and then running a script that starts it (which also starts its dependencies manually)

* Dependencies described and handled outside of the OTP Apps themselves itself

What is Needed

* Something that manages and provides clear semantics around system startup and shutdown
* Something that uses the dependencies described and manages where they occur and when they are started
* Something that does the creation and packaging and of Erlang Systems leveraging all that metadata
* Something to startup and manage systems

We Have It!

or How to Build Systems the Right way

* Something that manages and provides clear semantics around system startup and shutdown - Erlang/OTP Releases

* Something that uses the dependencies described and manages where they occur and when they are started - Erlang/OTP Release

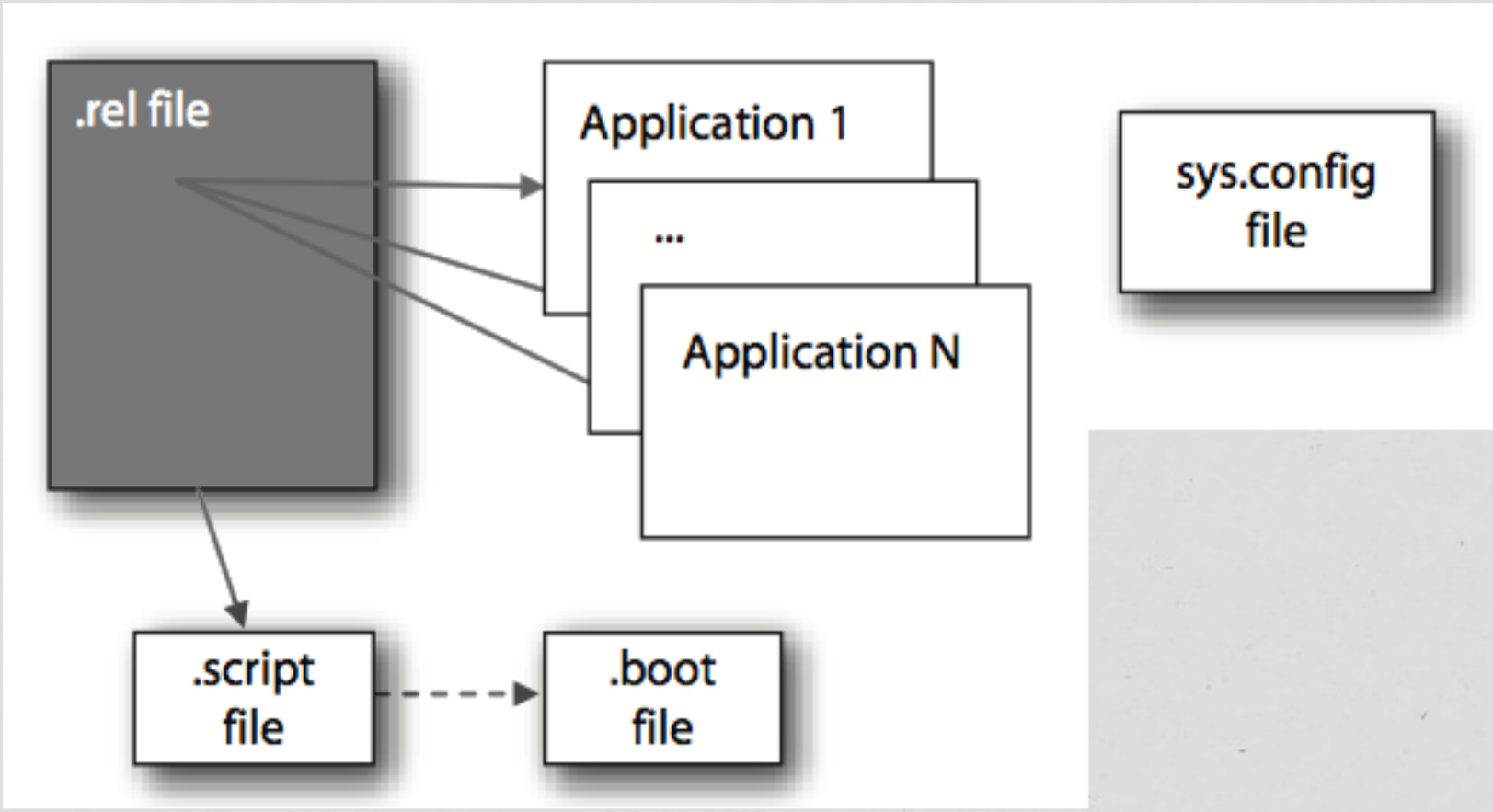
* Something that does the creation and packaging and of Erlang Systems leveraging all that metadata - Erlware's Relx

* Something to startup and manage systems - Erlang/OTP Release

What Exactly is a Release

* A Set of built, versioned OTP Applications
* Metadata that describes applications required
* An explicit configuration mechanism
* Optionally tarballs that can be managed and deployed

Release Overview



Release Metadata

```
{release,{"echo_get","0.0.1"},
         {erts,"5.10.1"},
         [{kernel,"2.16.1"},
          {stdlib,"1.19.1"},
          {ranch,"0.8.3"},
          {crypto,"2.3"},
          {cowboy,"0.8.5"},
          {echo_get,"0.0.1"}]}.
```

Release Structure

```
├── bin
│   ├── echo_get
│   └── echo_get-0.0.1
├── erts-5.10.1
├── lib
│   ├── cowboy-0.8.5
│   ├── crypto-2.3
│   ├── echo_get-0.0.0+build.1.ref5b05dba
│   ├── kernel-2.16.1
│   ├── ranch-0.8.3
│   └── stdlib-1.19.1
└── releases
    └── echo_get-0.0.1
        ├── echo_get.boot
        ├── echo_get.rel
        ├── echo_get.script
        ├── sys.config
        └── vm.args
```

Creating A Release

```
>relx

Starting relx build process ...
Resolving OTP Applications from directories:
    /Users/emerrit/workspace/EUC2013/echo_get/ebin
    /Users/emerrit/workspace/EUC2013/echo_get/deps
    /usr/local/Cellar/erlang/R16B/lib/erlang/lib

Resolving available releases from directories:
    /Users/emerrit/workspace/EUC2013/echo_get/ebin
    /Users/emerrit/workspace/EUC2013/echo_get/deps
    /usr/local/Cellar/erlang/R16B/lib/erlang/lib

Resolved echo_get-0.0.1
release successfully created!
```

What Relx Does

* Reads the configuration
* Discovers the environment (Apps and Releases available)
* Starting at the apps and constraints specified - does a constraint solve to resolve the full dependency tree
* Uses that information to assemble the release
* Creates the release, including all metadata and support functions

Configuration

```
%% -*- mode: Erlang; fill-column: 80 -*-
{release, {echo_get, "0.0.1"},
 [echo_get]}.
```

An Example (echo_get)

```
{application,echo_get,
             [{description,"Cowboy GET echo example."},
              {vsn,"0.0.0+build.1.ref5b05dba"},
              {modules,[echo_get,
                        echo_get_app,
                        echo_get_handler,
                        echo_get_sup]},
             {registered,[]},
             {applications,[kernel,
                            stdlib,
                            ranch,
                            cowboy]},
             {mod,{echo_get_app,[]}},
             {env,[]}]}.
```

echo_get Release Structure

```
├── bin
│   ├── echo_get
│   └── echo_get-0.0.1
├── erts-5.10.1
├── lib
│   ├── cowboy-0.8.5
│   ├── crypto-2.3
│   ├── echo_get-0.0.0+build.1.ref5b05dba
│   ├── kernel-2.16.1
│   ├── ranch-0.8.3
│   └── stdlib-1.19.1
└── releases
    └── echo_get-0.0.1
        ├── echo_get.boot
        ├── echo_get.rel
        ├── echo_get.script
        ├── sys.config
        └── vm.args
```

Constraining

```
%% -*- mode: Erlang; fill-column: 80 -*-
{release, {echo_get, "0.0.1"},
 [echo_get,
  sasl]}.
```

Constraining

```
%% -*- mode: Erlang; fill-column: 80 -*-
{release, {echo_get, "0.0.1"},
 [echo_get,
  {sasl, “2.3”, gte}]}.
```

App Overrides

```
{overrides, [{sexpr, "../sexpr"}]}.
```

Overlays

```
{overlay_vars, "vars.config"}.
{overlay, [{mkdir, "log/sasl"},
           {copy, "files/erl", "{{erts_vsn}}/bin/erl"},
           {copy, "files/nodetool", "{{erts_vsn}}/bin/nodetool"},
           {template, "files/app.config", "etc/app.config"},
           {template, "files/vm.args", "etc/vm.args"}]}.
```

Multiple Releases

```
%% -*- mode: Erlang; fill-column: 80 -*-
{release, {echo_get, "0.0.1"},
 [{echo_get, “0.0.1”}]}.

{release, {echo_get, "0.0.2"},
 [{echo_get, “0.0.2”]}.
```

Easily Extendable

* Simply implement the rlx_provider behaviour
* Api for accessing and manipulating releases via rlx_state, rlx_release and rlx_app_info

Implementing a Provider

```
-callback init(rlx_state:t()) ->
       {ok, rlx_state:t()} | relx:error().

-callback do(rlx_state:t()) ->
       {ok, rlx_state:t()} | relx:error().

-callback format_error(Reason::term()) ->
       iolist().
```

Adding Providers

```
{add_providers, [my_custom_functionality]}.
```

Erlware's Relx

* http://relx.org
* http://erlware.org

ErlangCamp 2013

http://erlangcamp.com

August 30th - 31st

ErlangCamp 2013

http://erlangcamp.com

October 11th - 12th

Questions??