Packaging Erlang Services for Production

Anthony Molinaro

Looking Back

My First Experience Packaging Software

By WillMcC at the English language Wikipedia, CC BY-SA 3.0,

My Subsequent Experiences Packaging Services

- Late 1990s - Idealab (wigwam)
  - Mostly Perl/PHP Web Services
- Early 2000s - Overture/Yahoo (skeletor)
  - C/Java Web Services
  - C++ Queuing Services
  - C UDP Services
- 2008-2017 - N54/OpenX (framewerk)
  - C/Java/Erlang Services
- 2018- - OpenX (rebar3_build_rpm)
  - Erlang Services

What is a Linux Service?

A process or set of processes which run as a daemon on a system and are integrated with init.

Services

- can be started/stopped and reloaded via scripts
- adhere to filesystem standards
- log to known locations or to syslog facilities.
- are inspectible
- are debuggable

What Needs to be in an Erlang Service?

- Release
  - Erlang Runtime
  - Beam files
  - Script to start/stop the Erlang node
- Integration with System Service Infrastructure
  - SysV/Supervisord/Systemd/Other
  - Container
- Possible Integration with System Logging Service
  - syslog
- Inspectible/Debuggable
  - remsh/recon/eval

Why Package your Service?

- Consistency
- Reliability
- Repeatability
- Low barrier of entry
- Enterprise ready

Who’s Already Packaging Services?

- **Riak** - released for many different operating systems via node_package
- **VerneMQ** - releases debs and rpms via node_package
- **RabbitMQ** - releases debs, rpms, and many others via Makefiles
- **Flussonic/Erlyvideo** - assumed rpm/deb based on epm
- **Sonic Pi** - releases debs, Mac OS dmg, Windows Installer
- **Wings3D** - self extracting shell scripts, Mac OS dmg
- **CouchDB** - releases debs, rpms, Windows .exe, MacOS via Makefiles
- **OpenX** - release rpms internally for about 12 services using framewerk and fw-template-erlang-rebar

How Are They Packaging?

How Are They Packaging?

- Distribution style (e.g. Fedora Core, Ubuntu)
  - One package for everything, use .spec files and system build tools

fedora PACKAGES

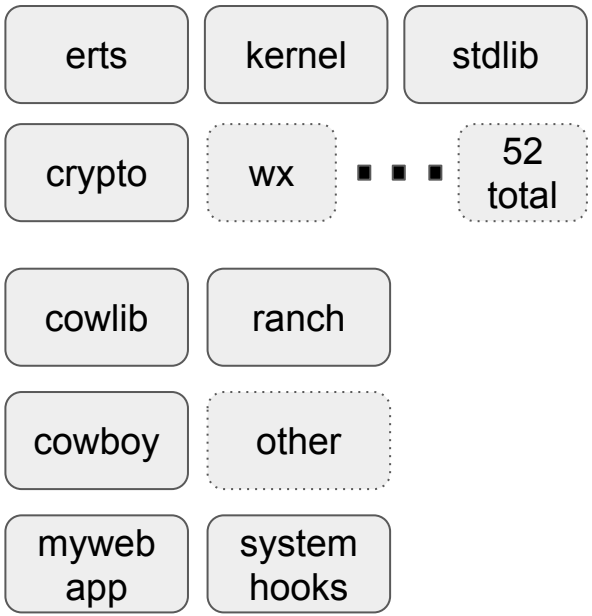
erlang

Search

700 results

erlang	General-purpose programming language and runtime environment
erlang-asn1	Provides support for Abstract Syntax Notation One
erlang-common_test	A portable framework for automatic testing
erlang-compiler	A byte code compiler for Erlang which produces highly compact code
erlang-cosEvent	Orber OMG Event Service
erlang-cosEventDomain	Orber OMG Event Domain Service

Distribution Style



Features

- One Package for Erlang Runtime System
- One Package for each Erlang App/Library
- Dependencies managed by System Package Manager

Pros

- Only packages you need are installed
- You can upgrade Erlang separately from your Applications/Libs
- You may be able to upgrades parts of applications

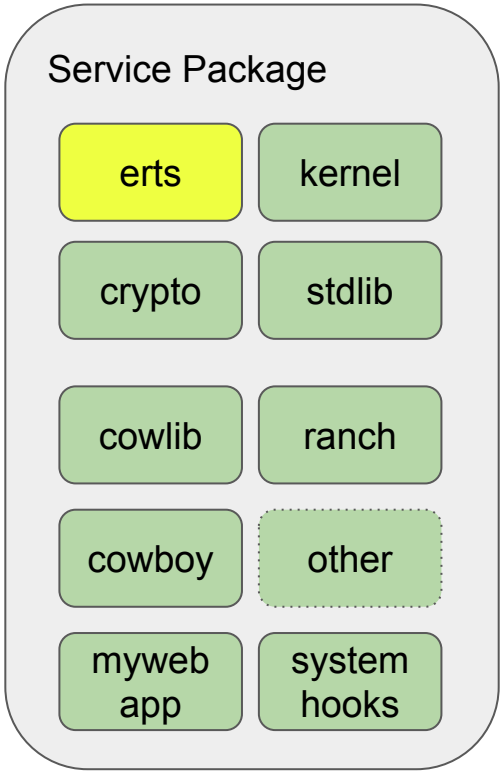
Cons

- Lots and Lots of Packages to manage
- Dependencies determined at install time
- Usually you need .spec files for Package construction

How Are they Packaging?

- Distribution style (e.g. Fedora Core, Ubuntu)
  - One package for everything, use .spec files and system build tools
- Node_Package / Release style
  - Single Package with just the applications you depend on
  - Optionally include the Erlang Runtime System

Node_Package + Release Style



Features

- One Package (or 2 if erts is left out)
- Dependencies managed by rebar 2/3

Pros

- Only prereqs are Erlang and rebar 2/3
- Single package with minimal external dependencies
- Integrates with several operating systems (node_package)

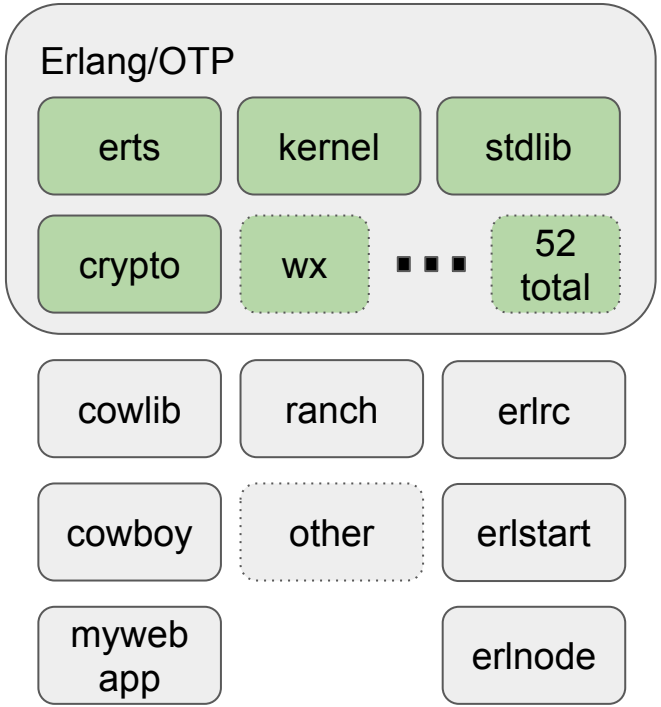
Cons

- Lots of complicated makefiles, templates, etc.
- Poor documentation, examples
- (node_package) Only supports reltool and rebar 2 (some rebar3 support in a fork)

How Are they Packaging?

- Distribution style (e.g. Fedora Core, Ubuntu)
  - One package for everything, use .spec files and system build tools
- Node_Package / Release style
  - Single Package with just the applications you depend on
  - Optionally include the Erlang Runtime System
- Framewerk style
  - One package for Erlang Runtime plus all OTP applications, compiler, etc
  - One package per Erlang Library/App
  - Erlrc to integrate Hot Code Loading with System Package Manager upgrades, and boot the application
  - Erlstart/Erlnode to manage the starting/stopping of the erlang node and integrate with System Services Infrastructure

Framewerk Style



Features

- One Package for Erlang/OTP
- One Package for each other Erlang Appl/Library
- Dependencies managed by System Package Manager

Pros

- Only packages you need are installed
- You can upgrade Erlang separately from your Applications/Libs
- You can upgrade parts of the overall service without rebuilding others.

Cons

- Lots of Packages
- Dependency resolution happens at install time
- Different hosts might have slightly different sets of packages

Framewerk + Erlrc + Erlstart + Erlnode

It’s more of a system to run many services

- Framewerk (plus templates)
  - Normalize building of first party and thirdparty software into system packages
  - Built as a set of autotools macros as well as shell scripts and tools
  - Highly pluggable (languages, revision control systems, packaging systems, etc)
- Erlrc
  - Controls boot order
  - Integrates system package transactions with Erlang hot code loading
- Erlstart
  - Start/Stop scripts for running an Erlang Node on a system
- Erlnode
  - Integration with RHEL/Centos SysV based init system

What Does this Look like in Practice?

Toy problem: create a cowboy hello world app using Framewerk

Steps:

1. Build prereqs
    a. Build and install erlang and rebar
    b. Build and install framewerk and fw-template-erlang-rebar
2. Build erlnode
    a. Build and install erlrc, erlstart, goldrush, lager, syslog, lager_syslog, cecho, entop and recon
    b. Build and install erlnode
3. Build prereqs for app
    a. Build and install cowlib, ranch and cowboy
4. Create and build the app

Framewerk Demo (Highlights)

```
# get prerequisite packages for demos
% git clone 'git@github.com:djnym/beamsf2018-demo.git'
% cd beamsf2018-demo
% make prereqs
# wait about 10 minutes
% rpm -qa | grep TEST
erlang-18.3.4.7-TEST1.x86_64
rebar3-3.5.0-TEST1.x86_64
rebar-2.6.4-TEST1.x86_64
```

Existing Framewerk Projects

```
% git clone git@github.com:dukesoferl/fw.git
% cd fw && git checkout 0.9.8
% ./bootstrap && ./configure && make package
% sudo rpm -i fw-pkgout/framewerk-0.9.8-TEST1.noarch.rpm

% git clone git@github.com:dukesoferl/fw-template-erlang-rebar.git
% cd fw-template-erlang-rebar && git checkout 0.5.0
% ./bootstrap && ./configure && make package
% sudo rpm -i \
fw-pkgout/fw-template-erlang-rebar-0.5.0-TEST1.noarch.rpm
```

Existing Framewerk Projects

```
% git clone git@github.com:dukesoferl/fw.git
% cd fw && git checkout 0.9.8
% ./bootstrap && ./configure && make package
% sudo rpm -i fw-pkgout/framewerk-0.9.8-TEST1.noarch.rpm

% git clone git@github.com:dukesoferl/fw-template-erlang-rebar.git
% cd fw-template-erlang-rebar && git checkout 0.5.0
% ./bootstrap && ./configure && make package
% sudo rpm -i \
fw-pkgout/fw-template-erlang-rebar-0.5.0-TEST1.noarch.rpm
```

Example of Framewerk Wrapped Package

```
% fw-init --name goldrush \
          --revision none \
          --template erlang-rebar \
          --with_build_prefix 1 \
          --wrap_git_path git@github.com:DeadZen/goldrush.git \
          --wrap_git_tag 0.1.6
% cd goldrush
% ./bootstrap && ./configure && make package
% sudo rpm -i fw-pkgout/erlang-18-goldrush-0.1.6-TEST1.x86_64.rpm
```

Example package contents

```
% rpm -ql erlang-18-goldrush | head
/etc/erlrc.d/applications/goldrush
/usr/lib64/erlang/lib/goldrush-0.1.6
/usr/lib64/erlang/lib/goldrush-0.1.6/ebin
/usr/lib64/erlang/lib/goldrush-0.1.6/ebin/glc.beam
/usr/lib64/erlang/lib/goldrush-0.1.6/ebin/glc_code.beam
/usr/lib64/erlang/lib/goldrush-0.1.6/ebin/glc_lib.beam
/usr/lib64/erlang/lib/goldrush-0.1.6/ebin/glc_ops.beam
/usr/lib64/erlang/lib/goldrush-0.1.6/ebin/goldrush.app
/usr/lib64/erlang/lib/goldrush-0.1.6/ebin/gr_app.beam
/usr/lib64/erlang/lib/goldrush-0.1.6/ebin/gr_context.beam
```

Example of Framewerk Wrapped with Deps

```
% fw-init --name lager \
          --revision none \
          --template erlang-rebar \
          --with_build_prefix 1 \
          --wrap_git_path git@github.com:erlang-lager/lager.git \
          --wrap_git_tag 2.1.1 \
          --with_deps goldrush
% cd lager
% ./bootstrap && ./configure && make package
% sudo rpm -i fw-pkgout/erlang-18-lager-2.1.1-TEST1.x86_64.rpm
```

Dependencie are Part of Package

```
% rpm -q --requires erlang-18-lager
/bin/sh
/sbin/ldconfig
erlang-18-goldrush >= 0.1.6-TEST1
rpmlib(CompressedFileNames) <= 3.0.4-1
rpmlib(FileDigests) <= 4.6.0-1
rpmlib(PayloadFilesHavePrefix) <= 4.0-1
rpmlib(PayloadIsXz) <= 5.2-1
```

Skipping a Few Steps

Imagine minor variations on the previous few slides for the following

- erlstart/erlrc
- syslog
- lager_syslog (depends on lager and syslog)
- cecho
- entop (depends on cecho)
- recon
- erlnode (depends on erlrc, erlstart, recon, entop and lager_syslog)
- cowlib
- ranch
- cowboy (depends on ranch and cowlib)

Packages so Far

```
% rpm -qa | grep TEST | sort
erlang-18.3.4.7-TEST1                  erlang-18-lager-2.1.1-TEST1
erlang-18-cecho-0.4.1-TEST1            erlang-18-lager_syslog-2.1.2-TEST1
erlang-18-cowboy-1.1.2-TEST1           erlang-18-ranch-1.3.2-TEST1
erlang-18-cowlib-1.0.2-TEST1           erlang-18-recon-2.3.1-TEST1
erlang-18-entop-0.2.0-TEST1            erlang-18-syslog-1.0.3-TEST1
erlang-18-erlnode-0.7.1-TEST1          framewerk-0.9.8-TEST1
erlang-18-erlrc-1.1.2-TEST1            fw-template-erlang-rebar-0.5.0-TEST1
erlang-18-erlstart-0.0.7-TEST1         rebar-2.6.4-TEST1
erlang-18-goldrush-0.1.6-TEST1         rebar3-3.5.0-TEST1
```

Generating/Developing/Building/Releasing

```
% fw-init --name mywebapp --revision none --template erlang-rebar \
    --with_build_prefix 1 --with_deps cowboy
% cd mywebapp
% rm -f include/myapp.hrl src/myapp.erl
% cp ../../packages/cowboy/cowboy-origin/examples/hello_world/src/* src
% mv src/hello_world.app.src src/mywebapp.app.src
% mv src/hello_world_app.erl src/mywebapp_app.erl
% mv src/hello_world_sup.erl src/mywebapp_sup.erl
% sed -i 's/hello_world/mywebapp/g' src/*
% ./bootstrap && ./configure && make package
% sudo rpm -ivh fw-pkgout/erlang-18-mywebapp-0.0.0-TEST1.x86_64.rpm
```

Starting It Up and Testing It

```
# start it up
% sudo service erlnode start
Starting Erlang... done.

# test it
% curl 'http://localhost:8080/'
Hello World!
```

Everything Running as Expected

```
% erlstart-eval \
 'lists:sort([ A || {A,_,V} <- application:which_applications()])'
[asn1, compiler, cowboy, cowlib, crypto, erlrc,
 goldrush, kernel, lager, lager_syslog,
 mywebapp, public_key, ranch, recon, ssl,
 stdlib, syntax_tools, syslog ]
```

Other Commands

- erlstart-remsh
  - get a remsh attached to the node
- erlstart-etop
  - run etop against the node
- erlstart-entop
  - run entop against the node

That’s Complicated!

But manageable once it’s setup which is why it’s lasted for 8 years or so, but it’s especially painful for a few reasons

- There are lots of packages and the system package resolvers don’t necessarily always do what you want
- It’s easy to upgrade the ERTS to a newer version, but harder to get all the dependent packages to build
- It’s not based on any standard Erlang release tools
- It lacks sufficient documentation

Can I make things better?

What do I want to achieve?

For me

- I want to not be the bottleneck for thirdparty packages
- I want to have fewer overall packages

For our developers

- Let them use more standard tools (e.g. rebar3 and relx)
- Give them more flexibility to compile and release with different versions of the ERTS or different versions of packages

So What was Missing From the Standard Tools?

A way to make a package, I had found 3 different ways:

- Makefiles and .spec files?
  - With framewerk those were all hidden, so introducing them now requires learning even more tools
- node_package
  - Used older release tools and older rebar, seemed overly complicated and not heavily adopted
- epm
  - Interesting attempt to replicate fpm in pure erlang, interesting because it didn’t rely on any external tools

rebar3_build_rpm plugin

What:

- Take a relx release and package as an rpm using epm

Pre-Reqs:

- Erlang Runtime and rebar3
- rebar3_service_rpm_template and rebar3_build_rpm plugin

Outputs:

- Single RPM (or deb)
- System service integration scripts

What Does this Look like in Practice?

Toy problem: create a cowboy hello world app using rebar3_build_rpm

Steps:

1. Build prereqs
    a. Build and install erlang and rebar3 (same as before)
2. Create the structure with rebar3_service_rpm_template
3. Add code, dependencies, and update release section
4. Create the rpm with rebar3_build_rpm

Get the Template

```
% mkdir templates
% cd templates
% git clone git@github.com:openx/rebar3_service_rpm_template.git
% echo "{template_dir, \"`pwd`\"}." >> $HOME/.config/rebar3/rebar.config
% rebar3 new
app (built-in): Complete OTP Application structure.
cmake (built-in): Standalone Makefile for building C/C++ in c_src
escript (built-in): Complete escriptized application structure
lib (built-in): Complete OTP Library application (no processes) structure
plugin (built-in): Rebar3 plugin project structure
release (built-in): OTP Release structure for executable programs
service_rpm (custom): Rebar3 template for an erlang service
umbrella (built-in): OTP structure for executable programs (alias of 'release'
template)
```

Generating the Basic Structure

```
% rebar3 new service_rpm mywebapp
===> Writing mywebapp/.gitignore
===> Writing mywebapp/README.md
===> Writing mywebapp/rebar.config
===> Writing mywebapp/bootstrap
===> Writing mywebapp/Makefile
===> Writing mywebapp/config/sys.config
===> Writing mywebapp/config/vm.args
===> Writing mywebapp/vars.config
```

Generating the Basic Structure cont.

```
===> Writing mywebapp/priv/init.script
===> Writing mywebapp/priv/admin.script
===> Writing mywebapp/priv/supervisord.ini
===> Writing mywebapp/priv/systemd.service
===> Writing mywebapp/priv/pkg.config
===> Writing mywebapp/priv/pre.sh
===> Writing mywebapp/priv/post.sh
===> Writing mywebapp/priv/preun.sh
===> Writing mywebapp/priv/postun.sh
===> Writing mywebapp/priv/sysconfig.example
```

Create our App

```
% cd mywebapp
% mkdir src
% cp \
../../framewerk/packages/cowboy/cowboy-origin/examples/hello_world/src/* \
src
% mv src/hello_world.app.src src/mywebapp.app.src ; \
  mv src/hello_world_app.erl src/mywebapp_app.erl ; \
  mv src/hello_world_sup.erl src/mywebapp_sup.erl ; \
  sed -i 's/hello_world/mywebapp/g' src/*
% sed -i 's/8080/8081/' src/mywebapp_app.erl
```

Default Deps Definition

```
{ deps,
  [{ lager, {git, "git@github.com:basho/lager.git", {tag, "2.0.1"}}},
   { syslog, {git, "git@github.com:Vagabond/erlang-syslog.git", {tag,
"1.0.5"}} },
   { lager_syslog, {git, "git@github.com:basho/lager_syslog.git", {tag,
"2.0.1"}} },
   { recon, {git, "git@github.com:ferd/recon.git", {tag, "2.3.1"}}},
   { cecho, {git, "git@github.com:djnym/cecho.git",{tag, "0.5.2"}}},
   { entop, {git, "git@github.com:mazenharake/entop.git", {tag, "0.3.0"}}}


  ]
}.
```

Add cowboy to the Deps

```
{ deps,
  [{ lager, {git, "git@github.com:basho/lager.git", {tag, "2.0.1"}}},
   { syslog, {git, "git@github.com:Vagabond/erlang-syslog.git", {tag,
"1.0.5"}} },
   { lager_syslog, {git, "git@github.com:basho/lager_syslog.git", {tag,
"2.0.1"}} },
   { recon, {git, "git@github.com:ferd/recon.git", {tag, "2.3.1"}}},
   { cecho, {git, "git@github.com:djnym/cecho.git",{tag, "0.5.2"}}},
   { entop, {git, "git@github.com:mazenharake/entop.git", {tag, "0.3.0"}}},
   { cowboy, {git, "git@github.com:ninenines/cowboy.git", {tag, "1.1.2"}}}
  ]
}.
```

Default release Section

```
{relx, [ { release, { 'mywebapp', "0.1.0" },
           [

             sasl,
             syslog,
             lager_syslog,
             lager,
             recon,
             {cecho, none}, % prevent cecho and entop from being loaded
             {entop, none}
           ]
         },
```

Default release Section

```
{relx, [ { release, { 'mywebapp', "0.1.0" },
           [
             mywebapp,      % ← app added here
             sasl,
             syslog,
             lager_syslog,
             lager,
             recon,
             {cecho, none}, % prevent cecho and entop from being loaded
             {entop, none}
           ]
         },
```

Build/Install/Start/Test

```
# build it
% make package
# install it
% sudo rpm -ivh _build/prod/rpm/mywebapp-0.1.0-TEST1.x86_64.rpm
# start it
% sudo service mywebapp start
Starting mywebapp:                                         [OK]
# test it
% curl 'http://localhost:8081/'
Hello world!
```

What’s the Inside of that Package Look Like?

```
% rpm -ql mywebapp | head
/etc/init.d/mywebapp
/etc/mywebapp
/etc/mywebapp/sys.config
/etc/mywebapp/vm.args
/etc/supervisord.d/mywebapp.ini
/etc/systemd/system/mywebapp.service
/usr/bin/mywebapp-admin
/usr/lib64/mywebapp
/usr/lib64/mywebapp/bin
```

What's the Inside of that Package Look Like?

```
/usr/lib64/mywebapp/lib/lager-2.1.1/src/lager_trunc_io.erl
/usr/lib64/mywebapp/lib/lager-2.1.1/src/lager_util.erl
/usr/lib64/mywebapp/lib/lager_syslog-2.1.2
/usr/lib64/mywebapp/lib/lager_syslog-2.1.2/ebin
/usr/lib64/mywebapp/lib/lager_syslog-2.1.2/ebin/lager_syslog.app
/usr/lib64/mywebapp/lib/lager_syslog-2.1.2/ebin/lager_syslog_backe
nd.beam
/usr/lib64/mywebapp/lib/mywebapp-1
/usr/lib64/mywebapp/lib/mywebapp-1/ebin
/usr/lib64/mywebapp/lib/mywebapp-1/ebin/mywebapp.app
/usr/lib64/mywebapp/lib/mywebapp-1/ebin/mywebapp_app.beam
```

Check the Running Apps

```
% mywebapp-admin eval \
'lists:sort([ A || {A,_,V} <- application:which_applications()])'
[ asn1, compiler, cowboy, cowlib, crypto,
  goldrush, kernel, lager, lager_syslog,
  mywebapp, public_key, ranch, recon, sasl, ssl,
  stdlib, syntax_tools, syslog ]
```

Other Commands

- mywebapp-admin remsh - get a remsh attached to the node
- mywebapp-admin entop - run entop against the node

Am I happier?

Comparison of Styles

	Framewerk	**rebar3_build_rpm**
PROS	● Normalizes build across many languages ● Integrates with revision control and provides checks against common mistakes ● Delegates dependency management to system	● Written in erlang ● Few external tools required ● Many fewer packages (hopefully 12 when all done)
CONS	● Written with esoteric tools ● Results in many, many packages (about 149 at last count)	● Light integration with revision control ● Dependency management is stricter (e.g. no range support)

What’s the Future Look Like?

Questions?

Where you can reach me

- {github,twitter,keybase}/djnym

Where to find the templates and plugin

- github.com/openx/rebar3_service_rpm_template
- github.com/openx/rebar3_build_rpm

Reproduce the demo output

- github.com/djnym/beamsf2018-demo