Hexes, Charms and Spells

MARK ALLEN – ALERT LOGIC

MRALLEN1@YAHOO.COM - @BYTEMEORG

What is Hex?

- https://hex.pm
- Internet facing package service/repository for Elixir and Erlang software.
- Written in Elixir
- Uses the Fastly CDN points of presence for worldwide distribution. (Thanks Fastly!)

Why do this?

- We have proprietary code we don't want to publish to the internet.
- We want to manage our vendored forks of open-source code.
- We have a common stack of applications we need to carefully manage.
- Speed and/or politeness (caching frequently fetched packages from upstream)

Why don't you host your own hex?

- Requires a database
- Has notions of "user accounts" and "access control"
  - **IMPORTANT:** I am not saying that authentication and authorization are stupid for **all** use cases; just that they require a layer of sophistication that isn't needed *for this application*.
- Requires a server with disk space and elixir and an operating system
- In short, it is a lot of infrastructure I'd rather not manage.

Doesn’t hex already support this?

- Hex has a design specification for “private packages” but some or all of that functionality is not implemented.
- Seems like it would require putting build artifacts out into the public Internet. (Even if encrypted and/or strictly access controlled.)
- As a practical matter, The Management will table flip if you glibly mention that you’ve uploaded all your secret sauce apps to an Internet package service.
- “What could possibly go wrong?”

Why can't you use github?

You can.

You should realize what you are signing up for in that model.

What are the design goals?

- Simple to install – remember no infrastructure to manage
- Simple to integrate with cloud services (for example, storing packages in S3, running code as “serverless”)
- No (okay, minimal) changes to rebar3
- Proof of concept server is written in *gasp* Python. Why?

For realz? Python?!

- Yes. Really.
- But why?
  - Great client libraries for cloud services
  - Really easy to write a simple RESTful web server (e.g., bottle)
  - erl_terms is a python library which can parse Erlang file:consult/1 format files and render them into Python data structures.

rebar3

- Recently gave a talk about why you should use rebar3 and some of its advanced features.
- It's on YouTube. You should watch it.
- Plugin system

Rebar3 plugins

- Rebar3 itself is implemented as a large set of “out of the box” plugins which call functions in some common library code.
- There are a couple of “built in” plugins that deal with hex downloads.
  - Indices (rebar_prv_update)
  - Packages (rebar_pkg_resource)

How rebar3 uses hex package indices

- Hex publishes a number of indices.
- The index resources are fully documented by the hex maintenance team.
- Rebar3 uses the “v1” package index

ETS Registry

NOTE: This registry format will be deprecated in favor of the new version

The registry is an ETS table serialized with `ets:tab2file/1`. Clients consuming the registry entries should always match on only the front of a list, as new elements may be added to the tail in the future.

Below is the layout of the table.

- `{'$$version$$', Version}` - the registry version
  - Version: integer, incremented on breaking changes
- `{Package, [Versions]}` - all releases of a package
  - Package: binary string
  - Versions: list of semver versions as binary strings
- `{{Package, Version}, [Deps, Checksum, BuildTools]}` - a package release's dependencies
  - Package: binary string
  - Version: binary string semver version
  - Deps: list of dependencies [Dep1, Dep2, ..., DepN]
    - Dep: [Name, Requirement, Optional, App]
      - Name: binary package name
      - Requirement: binary Elixir version requirement
      - Optional: boolean, true if it's an optional dependency
      - App: binary, OTP application name
  - Checksum: binary hex encoded sha256 checksum of package, see Package Tarball
  - BuildTools: list of build tool names as binary strings

Slight digression about v1/v2 registry

That last side read,

“This is deprecated.”

Why is rebar3 still using it?

The v2 hex registry specification

- RSA signed protobuffer encoded data files
  - A “names” resource
  - A “versions” resource
  - A per-package “package” resource
- Such confuse, much sad. Wow.

Slight digression about v1/v2 registry

That last sip: read,

"This is deprecated."

Why is rebar3 still using it?

AWKS

Rebar3 hex index cached on disk

- $HOME/.cache/rebar3/hex/default
  - packages.idx – just the packages rebar3 cares about
  - registry – comes from upstream
- Loads packages.idx when rebar3 runs
- You can explore this yourself using `ets:file2tab/1`

“I love it when a plan comes together.”

John “Hannibal” Smith, A-Team

Subverting the rebar3 package index

- We now know the way that rebar3 looks up projects **from** hex.
- Modify rebar3 to expect **either** a serialized ETS table from the endpoint, or a JSON document with the same data.
  - Rebar3 (already) folds over the upstream data to construct its own package index.
  - Rebar3 saves its index to the cache location.
  - The cache contents drives decisions about what packages exist.

The hex package format

- A hex package artifact is a tarball consisting of the following things:
  - VERSION (an ASCII digit)
  - metadata.config (in `file:consult/1` format)
  - contents.tar.gz (the compressed files themselves)
  - CHECKSUM (concatenate the three previous files, calculate SHA256, emit hex values in uppercase.)
- Implementation is in the rebar3_hex plugin.

Endpoints

- GET /registry.ets.gz (from upstream/can vendor/control updates)
- GET /registry.json (**not an official endpoint**)
- GET /tarballs/PACKAGE-VERSION.tar
- POST /packages/{name}/releases
  - The body is the hex tarball

Putting the pieces together

1. Lightly modified rebar3
2. HEX_CDN environment variable
3. A simple python server implementing the endpoints
4. ???
5. Profit

So what's next?

- Upstream rebar3 is strongly -1 on this hack. (I am sympathetic to this position ☺)
- Just because you **can** do something doesn't mean you **should** do something.

Solving this issue “the right way”

- Implement hex v2 registry resources in rebar3
- When? Probably in summer 2018.
- Why then? Because we get maps in rebar3.
- How:
  - Use the same basic ideas here to collect a set of registry files from different sources (some private, some public perhaps)
  - Overlay the collected resources into a materialized view

Resources

- https://github.com/mrallen1/charm (fair warning, this repo is a hot mess)
- https://github.com/hexpm/specifications
- https://gist.github.com/mrallen1/b29507badc5c8ad3bd61b7d2205c42c7 (how to decode hex v2 resources in Erlang – a work in progress)
- https://github.com/tsloughter/rebar3_hex

A final plea

- Please consider putting your libraries on hex.
- Relying on the Lazy Web for your software recommendations leads to sadness and frustration.
- It makes your project more visible to people who might have the same problem to solve.
- It's easy.
- It's free.

Thank you!
Questions?

Mark Allen – Alert Logic

mrallen1@yahoo.com - @bytemeorg