Build and Packaging Update

Markus Kitsinger
github.com/SwooshyCueb
Software Developer, iRODS Consortium

July 5-8, 2022
iRODS User Group Meeting 2022
Leuven, Belgium

Areas of Interest

- External dependencies
- External dependency packaging
- iRODS Buildsystem
    - Compiler and C++ Standard Library
    - Dependency management
- iRODS packaging

Brief Overview

- Current state of affairs, rationale, and caveats
  - Externals
  - `libc++`
  - Packaging
- The new approach
- All the friends we'll meet along the way

The Current State of Affairs - Overview

- External dependencies packaged with `fpm`
- iRODS built with CMake and packaged with CPack
- Everything is built with `clang` and `libc++` that we provide
- Two flavors of packages supported: `dpkg` (`deb`) and `rpm`

The Current State of Affairs - Externals

What

- A set of separately packaged dependencies
- Not our code
- Live in `/opt/irods-externals` (by default)
- https://github.com/irods/externals

Why

- Distributions do not have all our dependencies in their package repositories
- Distributions tend to have older versions of our dependencies

The Current State of Affairs - Externals - Caveats

- Externals are not well-integrated into system
- Currently not set up to provide different sets of externals for different distros
- Current iRODS buildsystem relies pretty heavily on how our externals are packaged

More on this later...

The Current State of Affairs - `libc++`

What

- iRODS and most of our externals are built with `clang`
- All C++ built against `libc++`
- Using `clang` and `libc++` from our externals

Why

- Newer `clang` and newer `libc++` than is in distribution repositories
- Much of our code is not `gcc`-friendly
- At one point, `clang`/`libc++` adopted new features faster than `gcc`/`libstdc++`

The Current State of Affairs - `libc++` - Caveats

- Mixing binaries built against `libc++` and `libstdc++` is problematic
- Distro-provided packages generally use `libstdc++`
- Increases the number of externals we must provide
- Makes building against iRODS more complicated

The Current State of Affairs - Packaging

What

- iRODS built using CMake and packaged with CPack
  - Buildsystem does a lot of platform-specific heavy lifting
  - Most packaging defined in CMake
  - File/directory ownership handled programmatically with postinst scripts
  - Libraries in `/usr/lib`, regardless of what the distro expects
- Externals packaged with `fpm`

Why

- CPack and `fpm` are one-size-fits-all solutions, easier to wield than `dpkg-buildpackage` and `rpmbuild`
- The approach at the time was lazy-but-sufficient

The Current State of Affairs - Packaging - Caveats

- Cannot provide debian or rpm source packages
- Service account shenanigans
  - More on this later
- Using system-provided dependencies in lieu of externals we provide is tricky
  - May require buildsystem changes
- No package linting
- No "start from zero" package builds (no `pbuilder`)

More on next slide...

"Lazy but Sufficient" is Neither

- We need to provide debian and rpm source packages
- Service account hot-potato means no `systemd` unit files
- CMake has to know a lot about the target distros to produce usable packages
  - All dependencies must be specified manually (no `dpkg-shlibdeps`)
- Adding support for another distro requires more work and a new release
  - Likewise for a new version of an already supported distro
- We want to reduce the number of externals packages we provide
- No automated symbol tracking

The Future of iRODS Build and Packaging: "Normal and Boring"

- We will shift to using the standard tools (`dpkg-buildpackage` and `rpmbuild`) for packaging
  - `git-buildpackage` will be used to maintain debian packages, Salsa-style
    - Possibly rpm packages as well, still investigating
  - We will not provide an externals package if the distribution already provides a usable package
  - Debian and rpm source packages will be provided in our repositories
  - We will follow established patterns for setting up service accounts
  - We will install our libraries in the normal locations
  - We will provide default systemd unit(s)
- We will build against `libstdc++`
- We will decouple the iRODS buildsystem from externals packaging implementation details

Clearing Hurdles and Shaving Yaks

- I am still familiarizing myself with `rpmbuild` and friends. Most of my packaging experience is with `dpkg`, `PKGBUILD`, and Wix.
- New workflow and instrumentation for building packages.
  - Separate workflows for "from zero" builds and routine development builds.
- Service account hot-potato is actually part of a larger issue that must be solved *with care* as part of this transition.
- Distros without a new-enough `libstdc++` will need a `libstdc++` externals package.
- We will have to write CMake find modules for non-CMake dependencies that do not already have them.
  - We may have to also write CMake find module wrappers to work around bugs and oversights in the CMake-provided find modules, such as `FindODBC.cmake`.
- This transition cannot be easily broken up into stages.

Timeline (or lack thereof)

We don't know.

- Yak shaving
- Known unknowns

We may have a better idea of the time table at the next UGM.

Questions?