r2u

CRAN as Ubuntu Binaries

Dirk Eddelbuettel

11èmes Rencontres R 2025 @ Université Mons

19 May 2025

https://dirk.eddelbuettel.com/papers/recontres_r_r2u_may2025.pdf

ILLINOIS

Agenda In a Few Key Words

Key Issues

- CRAN as our key repositories
- Binaries as fastest installation of CRAN packages
- Full CRAN *and* system dependency resolution, reliably
- Without additional tools or cognitive load
- On a widely used (cloud / ci / server / laptop) platform

Our Focus and Contribution

Key Design 'Desirables' for Package Deployment and Testing

- We want *fast* and *automated* task execution and *fast* feedback loops
- Task executions that are *easy* to setup / maintain and *just work*
- Importantly also *reliable* without 'surprises' or debugging nightmares
- Plus an additional "won't break" aspect (more on that later)
- On platforms that are pervasive (cloud, CI, servers, laptops, ...)

Key Design 'Constraint'

- We have chosen a particular Linux 'vertical' here (*i.e.*, Ubuntu)
- What we present could be implemented in other verticals

Specific Context

We focus on R here as it is 'our tool of choice'

- But nothing that we present is *specific* to R
- One could create a similar for Python, Julia, Rust, ...
- Essentially any language with a repository (but having CRAN helps)

We also focus on `.deb` binary packages

- Also nothing specific on this choice: 'my tool of choice' are `.deb` packages
- `.rpm` and others formats such as `brew` packages could be used

We equate 'Deployment' with Linux

- Common in cloud and server use
- But also on laptops and "this year, finally" on desktops

So why CRAN ?

CRAN rules R: The Good

- Excellent repository with stringent quality control
- “Everything builds at @HEAD”: Marvellous quality guaranty
- This enables scripting and automation

CRAN rules R: The Less Good

- On Linux by default compilation from source
  - we talk about alternatives such as p3m.dev a little later
- And compilation from source
  - is generally (much) slower (for non-trivial tasks)
  - is generally (much) more error prone (those dependencies ...)

Linux Distributions Offer Help

Features

- Group packages for easier installation
- Organize dependencies (e.g. by analysing dynamic libraries, and more)
- Provide ‘cohorts’ aka ‘releases’
- Exist in a range
  - ‘vertical’ with one platform as e.g. Linux distributions
  - ‘horizontal’ as e.g. Conda across platforms
- We will focus on the vertical case

Linux Distributions Offer Help

Why Focus Here?

- The example of Ubuntu is quite intriguing
- It is generally “pretty good, pretty complete, pretty current”
- As well as polished and maintained well-enough (including first-rate security)
- Hence default at eg GitHub Actions, Google Colab and other instances
- Broad support for ‘other’ software in Ubuntu .deb form
  - Intel supports it for ‘oneAPI’ (i.e. MKL, TBB, ...)
  - Nvidia supports it to support its GPUs
  - and many more tools (Code, Positron, ...)

This is not to start an argument. Similar efforts can be done and are being done for Fedora and OpenSUSE.

In a Distribution

- Generally runs as system user, can install system dependencies
- Can cover everything that is offered within the distribution
- Can cover version dependencies and possibly version pinning
- But cannot cover what is not packaged: often the relevant applications

In an Application

- Application-level package managers are common and excellent
- They cover a lot of ground getting other packages *for this app*
- They can cover package dependencies and possibly pinning
- They generally cannot install system dependencies

So Promise In Approach To "Join" Both

- Integrate the application packages into the distribution repositories
- Now both united and dealt with jointly the system package manager
- Application level requirements satisfied by system level manager

Several Past Attempts

- Several incomplete attempts in the past, c2d4u max'ed out at 6k out of 21k
- Some details in our 'binary R packages' arXiv paper (and earlier r2u talks)
- But now r2u is the first that got it working
  - As the paper notes they are similar attempts for RHEL/FC (Iñaki), OpenSUSE (Detlef)

apt install r-cran-whatever

- Every CRAN package built, name prefixed with r-cran- and lower-cased
- The distribution build process wraps around the R installation
- So it is guaranteed the package can actually be loaded
- Thus every dependency formally declared, resolvable – and tested
- Supported (currently) for the three Ubuntu LTS releases on x86_64 aka amd64
    - And now also for arm64 for Ubuntu 24.04
- More details at https://eddelbuettel.github.io/r2u

Alternatives

p3m.dev

- Compelling system offering breath across packages and OS choices
- Generally provides binaries
  - But sometimes only source, unclear ex ante if one gets source or binary
  - This can differ for the same package across LTS releases
  - Also covers ‘time machine’ aspect continuing from MRAN
- Not integrated with the package manager: system deps are ‘harder’
  - Mechanism to obtain required commands, but not automated / integrated
- But useful and e.g. it does provide our inputs here
- Overall ‘not bad at all’ given the huge task of deploying across OSs
- But *we can do better* by integrating with the system package manager

This would be a fair place to compare to PyPI, Conda,... but that is out of scope here – and I do not really use either of those. Sorry.

Advantages of System Integration

Missing Libraries are Installed

- Installing, say, a PostgreSQL-using package leads to installation of Postgres library

Used Libraries are Never Removed

- Package manager *knows* that the client package uses libpqN
  - So when the system updates the library from releases N to N+1
  - Package is no longer left broken by removing the dependency
- This feature is lacking in without system integration
  - p3m.dev and r-universe binaries do not tie this back to system
  - of course it also lacks when we install directly from source

In the narro sense you can argue that *e.g.* a CI check run is ephemeral and system updates never happen. Fine. It is still damn convenient on all other systems.

Advantages of System Integration: Breaks Without

```
FROM rocker/r-ubuntu:20.04

RUN    apt update -qq \
    && apt install -y r-cran-rcppgsl libgsl-dev \
    && Rscript -e 'install.packages("RcppZiggurat")' \
    && Rscript -e 'library(RcppZiggurat); cat("All good\n")'

## Upgrade from focal to jammy, which means GSL 2.3.* to 2.7.*
RUN    sed -i -e 's/focal/jammy/g' /etc/apt/sources.list \
    && sed -i -e 's/focal/jammy/g' \
       /etc/apt/sources.list.d/c2d4u_team-ubuntu-c2d4u4_0_-focal.list

## 326 packages if we upgrade, so skip now, run if you prefer
#RUN    apt update -qq \
#    && apt upgrade -y

## Now (for expedience just) upgrade RcppGSL, brings
## upgraded libgsl28, removes libgsl27
RUN    apt update -qq \
    && apt install -y r-cran-rcppgsl

## And RcppZiggurat is borked -- this fails because libgsl23 is gone
## so we comment it out not run break the docker build, but see 'manually'
#RUN Rscript -e 'library(RcppZiggurat)'
```

Simple Demo “Proof”: Breaks under Ubuntu

We install eg RcppGSL (available as binary) and the GSL, then build RcppZiggurat

We upgrade from ‘focal’ to ‘jammy’, this gets us a new libgsl2* version.

And that breaks RcppZiggurat.

Similar for other versioned shared libraries: libicu*, libpq*, ...

Advantages of System Integration: No Break With r2u

```
FROM rocker/r2u:20.04

## under r2u this installs RcppZiggurat binary and its dependencies
RUN    apt update -qq \
    && Rscript -e 'install.packages("RcppZiggurat")' \
    && Rscript -e 'library(RcppZiggurat); cat("All good\n")'

## Upgrade from noble to oracular, which means GSL 2.3.* to 2.7.*
RUN    sed -i -e 's/focal/jammy/g' /etc/apt/sources.list \
    && sed -i -e 's/focal/jammy/g' /etc/apt/sources.list.d/r2u.list

## Now (for expedience just) upgrade RcppGSL,
## brings upgraded libgsl27, removes libgsl23
RUN    apt update -qq \
    && apt install -y r-cran-rcppgsl

## So RcppZiggurat is not broken as it got upgraded too
## Because the package manager knows it was affected
Rscript -e 'library(RcppZiggurat); cat("All good\n")'
```

Simple Demo “Proof”: Works under r2u

Doing equivalent steps under r2u but with packaged RcppZiggurat

But now upgrading RcppGSL … also gets updated RcppZiggurat: No breakage.

To replicate, Dockerfiles from previous and this slide are at GitHub.

Advantages of System Integration: Discussion

General Issue

- Whenever you have a versioned shared library:
  - package manager may roll to the next version
  - but unless package manager knows of a 'client package' ...
  - ... a current or previous version may get uninstalled
  - leaving a *non-package manager known* build like a standard R package stranding
- This happened for example a lot with `libicu*` for Unicode
  - my (non-r2u) desktop has libicu57, libicu70, libicu72 and libicu74 installed
  - my current `stringi` builds uses libicu74 so all good
- Other examples are 'versioned' database, graphics libraries, or the spatial stack!
- r2u guarantees you will not have this breakage: the package manager knows!

BSPM: Another Key Component

Bridge To Package Manager (by Iñaki Ucar)

- Cleverly 'intercepts' `install.packages()` calls made by R
- So `install.packages(c("xgboost", "mlpack"))` does what you expect
- Translates these into corresponding `apt` calls
  - Now *R users* do not need to know about `apt`
  - Also works with `dnf` and other package managers
- We can just take a package and say 'install dependencies' (eg via `remotes`)
- Ideal use case is for example continuous integrations
  - Drop-in setup, no system admin needs, no debugging
  - Use for example by my r-ci uses it

Actual CI Example

Only whitespace removed to fit display

Can be used as drop-in file `ci.yaml`

“Easy. Fast. Reliable.” for CI

More documentation at r-ci

```yaml
# Run CI for R using https://eddelbuettel.github.io/r-ci/
name: ci
on:
  push:
  pull_request:
env:
  _R_CHECK_FORCE_SUGGESTS_: "false"
jobs:
  ci:
    strategy:
      matrix:
        include:
          - {os: macOS-latest}
          - {os: ubuntu-latest}
    runs-on: ${{ matrix.os }}
    steps:
      - name: Checkout
        uses: actions/checkout@v4
      - name: Setup
        uses: eddelbuettel/github-actions/r-ci@master
      - name: Dependencies
        run: ./run.sh install_all
      - name: Test
        run: ./run.sh run_tests
```

Quick Background: What is a Binary Package?

`R CMD INSTALL --build` creates compressed archive

Corresponds to tree of installed package directory

But "naked" binary: no system dependency meta info

NB: This example uses a simple zero-dependency package.

```
~/git/zigg(master)$ R CMD INSTALL --build .
* installing to library '/usr/local/lib/R/site-library'
* installing *source* package 'zigg' ...
** this is package 'zigg' version '0.0.2'
** using staged installation
** libs
using C compiler: 'gcc (Ubuntu 14.2.0-4ubuntu2) 14.2.0'
using C++ compiler: 'g++-14 (Ubuntu 14.2.0-4ubuntu2) 14.2.0'
gcc -std=gnu2x -I"/usr/share/R/include" -DNDEBUG -I../inst/include      -fpic  -O3 -Wall -pipe -pedantic -c
g++-14 -I"/usr/share/R/include" -DNDEBUG -I../inst/include      -fpic  -O3 -Wall -pipe -pedantic -Wno-ignore
g++-14 -Wl,-S -shared -L/usr/lib/R/lib -Wl,-Bsymbolic-functions -flto=auto -ffat-lto-objects -Wl,-z,relro -o
installing to /usr/local/lib/R/site-library/00LOCK-zigg/00new/zigg/libs
** R
** inst
** byte-compile and prepare package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded from temporary location
** checking absolute paths in shared objects and dynamic libraries
** testing if installed package can be loaded from final location
** testing if installed package keeps a record of temporary installation path
* creating tarball
packaged installation of 'zigg' as 'zigg_0.0.2_R_x86_64-pc-linux-gnu.tar.gz'
* DONE (zigg)
~/git/zigg(master)$
```

Quick Background: What is a Binary Package?

A .deb package contains a similar (embedded) tarball

But also contains meta information (and possibly helper scripts)

The package build step runs both the inner `R CMD INSTALL` as well as additional steps to determine the meta data

```
$ apt-cache show r-cran-sf_1.0-20-1.ca2404.2_amd64.deb
Package: r-cran-sf
Version: 1.0-20-1.ca2404.2
Architecture: amd64
Maintainer: Dirk Eddelbuettel <edd@debian.org>
Installed-Size: 7968
Depends: libc6 (>= 2.38), libgcc-s1 (>= 3.0), libgdal34t64 (>= 3.8.0), \
         libgeos-c1t64 (>= 3.11.0), libproj25 (>= 7.1.0), libstdc++6 (>= 13.1), \
         r-base-core (>= 4.4.0), r-api-4.0, r-cran-classint, \
         r-cran-dbi, r-cran-magrittr, r-cran-s2, r-cran-units, r-cran-rcpp
Suggests: r-cran-blob, r-cran-nanoarrow, r-cran-covr, r-cran-dplyr, r-cran-ggplot2, [....]
Section: gnu-r
Priority: optional
Homepage: https://cran.r-project.org/package=sf
Description: CRAN Package 'sf' (Simple Features for R)
 Support for simple feature access, a standardized way to encode and analyze
 spatial vector data. Binds to 'GDAL' <doi:10.5281/zenodo.5884351> for
 reading and writing data, to 'GEOS' <doi:10.5281/zenodo.11396894> for
 geometrical operations, and to 'PROJ' <doi:10.5281/zenodo.5884394> for
 projection conversions and datum transformations. Uses by default the 's2'
 package for geometry operations on geodetic (long/lat degree) coordinates.
$
```

NB We shortened the Suggests: list here.

How Does It Work?

- We run `dpkg-buildpackage` for each package inside a Docker container
  - This gets us proper library dependencies as if Ubuntu built it
  - Proper steps of a genuine distribution package
- We accelerate the builds where we can by using p3m.dev R binaries
  - Which we unpackage inside the directory tree of the build
  - So in most (but not all) cases we can skip the `R CMD INSTALL` step of package build
- Small amount of meta data for extra dependencies we need to declare
  - Or a few builds we blacklist for various reasons
- We use standard tools to create a repository for, server it locally
- Internet2 mirror thanks to Tech Support in Liberal Arts & Sciences at Illinois

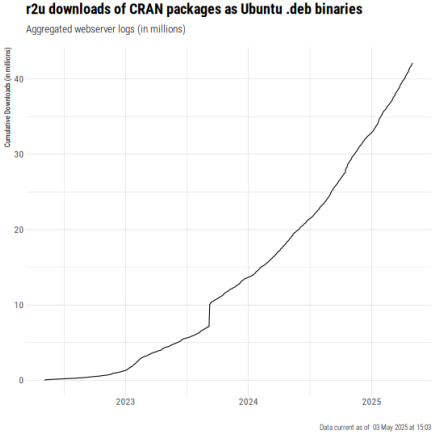
How Does One Use It?

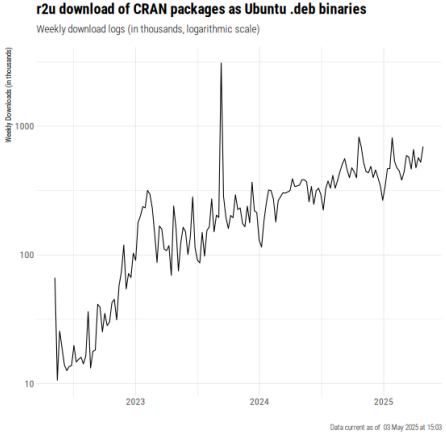
- Documentation site has script with four (or five with `bspm`) steps for Ubuntu
  - Used thousands of times in continuous integration at GitHub
  - There is also a dedicated GitHub Action to have this done in one step

How To Get Started?

- Dedicated Docker containers for deployment `rocker/r2u` for three flavours
- Or ‘manually’ apply script steps to a standard Ubuntu system, or run script
- Or ‘drop in’ the CI script from the previous page

Usage Steadily Growing: Now over 42 Million Packages Shipped





Discussion

- r2u shows we can integrate curated 'application repositories' into a Linux distro
- Doing so creates 'total sum greater than sum of parts' effects
  - We get all the benefits of our preferred compute environment (here: Ubuntu)
  - We get all the packages of our preferred application language (here: CRAN for R)
  - The integrations is *fast, easy, reliable* as fully featured binaries are used
- First Extension: adding arm64/aarch64 for the approx 25% binary packages
- This can serve as model for other languages and/or environments
- Nothing *fundamentally* limiting this to either Ubuntu or R + CRAN

r2u Useful For Non-Apt Binaries

One can also install Ubuntu 24.04 binaries from r-universe and r-multiverse, the

```
rp <- c("https://community.r-multiverse.org/bin/linux/noble/4.5",
        getOption("repos"))
install.packages(c("glaredb", "polars"), repos=rp)
```

On an r2u container, installs two Rust-based R packages not-on-CRAN as *r-multiverse* binary Ubuntu packages along with their dependency `nanoarrow` (here installed from r2u via `bspm`). In 23 seconds.

And Some 'Movies'

Some More Screencapture "Movies" Of r2u In Action

- tidyverse and rstan and brms in 33s (gif)
- tidyverse in 20s (gif)
- tidyverse and sf and lme4 (mp4)
- Single-Cell Genomics: 389 packages in 75 seconds (gif)
- ML packages xgboost, lightgbm, mlpack, torch in 10 seconds (gif)

These and more are at `https://github.com/eddelbuettel/images`

That's It!

Thank You! And Thanks To

- R (package) authors for creating something wonderful in the commons
- The CRAN team for all they do making it *reliably* accessible
- All past 'cran2deb' members: Albrecht, David, Stefan, Charles, Don, Michael, ...
- posit for p3m.dev which provides a very usable base layer
- Iñaki for `bspm` making interfacing the repo so smooth, and many discussions
- Rami Dass and Liberal Arts & Sciences IT at UIUC for hosting r2u
- My GitHub sponsors for all the coffee money

And see https://eddelbuettel.github.io/r2u/ for **r2u**