2025-03-21

R package binaries for Linux - Community Edition

Dr. Patrick Schratz
devXY GmbH

Signatories

Josiah Perry (Infrastructure Steering Committee member) has acknowledged the project idea and even donated for it.

Sharing the project in the `rstats` subreddit resulted in some very positive reactions. (As my social media accounts are not having the greatest reach and Twitter/X is basically dead, the reactions there were limited.)

A blog post on devXY's website has also been shared in rweekly in December 2024.

A variety of business clients of mine have shared their enthusiams with me personally, hoping to see a positive development of the overall project idea.

Project team

Patrick Schratz, CEO devXY.

Over a decade of experience with R, combined with a PhD in applied machine learning and a background in DevOps engineering. Specialized in designing and deploying R-centric data science environments for enterprise teams. Brings strong expertise in cloud infrastructure, UNIX system administration, and optimizing R for high-performance use cases.

Contributors

None right now.

The Problem

Linux binary packages

Binary Packages are essential for efficient workflows in R. Currently, CRAN is building and publishing binary packages for Windows and macOS. Linux binaries are missing, even though most

(corporate) team environments run on Linux. In addition, the share of private R users using Linux as their host OS becomes larger every year.

Without binaries for Linux, package installations can take many minutes, sometimes even up to hours (e.g. when packages like {duckdb} or {Rfast} are involved). To my knowledge (spanning mostly from community discussions), CRAN does not seem to have plans building binary packages for Linux, at least in the forseeable future. The motivation to do so may have diminished even further, particularly after Posit launched its public Package Manager service in 2020, offering Linux binaries for various distributions on the `x86_64` architecture.

Public Posit Package Manager

The PPM has been a valuable resource for the R community since its launch. However, it has the following drawbacks:

- The build process is untransparent / not public
- The download speed is at best "acceptable"
- The usage of the binaries comes with a TOS agreement. This can be problematic for specific use cases and poses a general risk to users, as it is often overlooked and might cause (costly) architecture adjustments once it is realized
- Binaries for the arm64 architecture are missing

Alpine Linux

Alpine Linux binaries are currently not available at all. Despite this, Alpine has become the de facto standard for CI/CD builds—both containerized and non-containerized—largely due to its minimal OS footprint and lightweight system libraries.

Unlike most mainstream distributions, Alpine uses the MUSL C library instead of the more common GLIBC. This difference can cause compatibility issues for packages that interface with C code, requiring additional adjustments from package authors. However, unless specifically targeting Alpine, most developers don't make these adaptations—partly because CRAN does not run MUSL-based checks as part of its submission process.

In a recent exchange with the CRAN team (January 2025), they confirmed that introducing such checks is not currently a priority. As a result, the absence of Alpine-compatible R package binaries continues to limit R's integration in modern CI/CD pipelines, leaving it at a disadvantage compared to other languages.

Architecture: arm64

Here's a refined and subtly rephrased version of your paragraph:

Much like the situation with Alpine Linux, R package binaries for the `arm64` architecture are currently unavailable. Even Posit has yet to make progress on this front, despite indicating in a 2024 correspondence that they intend to begin building for `arm64` in 2025.

The `arm64` architecture has gained significant traction in recent years, with strong support across major cloud providers. These servers often deliver better cost-efficiency and, in many cases, superior CPU performance compared to their `x86` counterparts.

The absence of R package (and interpreter) binaries for `arm64` limits R's presence in this growing ecosystem, effectively making it a second-class citizen in this space.

R Universe

R Universe, an existing R-consortium funded project, is a first step to an alternative packaging system for the R community.

However, as of today, R universe only builds binary packages for the latest LTS release of Ubuntu (filtered on Linux in general). Packages for other architectures and distributions are missing.

In addition, the build process heavily relies on GitHub Actions. While this might seem a positive aspect for some on the first look, I'd argue that it is effectively a downside for the following reasons:

- GitHub's public runners are rather slow compared to evenly-sized cloud VMs
- GitHubs default free build minutes are quite limited and the costs for adding additonal ones are quite high, compared with the alternative of providing private runners
- GitHub did not have `arm64` support until recently, and the support for such is in an relatively early stage with limited build capacities

I argue that most of the build process could be done more efficiently, both in terms of costs and resource efficiency.

Another notable shortcoming is the opacity of the package build engine, along with its infrastructure and overall build process. While each package has a dedicated repository that runs builds publicly via GitHub Actions, the underlying logic and workflows remain somewhat abstract and not easily accessible.

This lack of transparency means users cannot readily reuse or adapt the build and publishing process for their own purposes—they are instead fully dependent on the behind-the-scenes mechanisms of R Universe.

Creating custom repositories

All of previously mentioned limitations pose significant challenges for teams looking to create and maintain their own private repository with a curated set of (internal) R packages. Nowadays, some tools exist that allow achieving this: {minicran} and drat. However, these do not provide the option to (easily) build binary packages or manage the packages in S3 buckets.

Ideally, it should be as easy as running a single function (or two) which initializes the remote storage, builds packages from a local source or remote URL and returns the final repository URL for package downloads in the end.

The motivation for developing alternative solutions has grown over the past year, particularly after Posit significantly increased the pricing of its Package Manager product. Previously, it offered a reasonably priced and convenient way to manage internal R repositories. However, the recent pricing shift has made the tool cost-prohibitive for many organizations, especially relative to its feature set; prompting numerous teams to reevaluate their tooling choices.

The proposal

Establish a R-based, **open-source** build system for R package binaries. Not only the source code of the underlying code engine should be public, but also the actual build processes (running in CI) of the publicly running service building CRAN package binaries.

The build system should be able to build packages for (common) Linux distributions and multiple architectures (for the start `x86` and `arm64`, `riscv64` in the forseeable future). As R is the best tool

for building its own packages, thanks to projects like {pak}, {cranlike}, {pkgdepends}, {cranberries} and others, the core part will be done in R itself. This also allows for possible contributions from the community itself, as the codebase should be familiar to them.

The engine should be written in a generic way, so that it can be used to build binaries from any R package source, being it a local (on disk) or remote (URL) one. It should allow publishing to custom repositories, so that the community has a "go-to" tool for building and creating their own private repositories.

Users should optionally be allowed to store the build metadata in a database (SQLite, PG, MariaDB), allowing for optional statistical analysis and other post-hoc analysis.

The main storage for the binaries itself is S3 (as the only option for now). S3 allows for flexible data storage and replication while keeping costs small. Currently, no existing build tools allow using S3 directly, i.e. binaries must be stored on hard disks. Doing so increases storage costs by many factors and becomes a problem for medium-large sized repositories.

Storing packages in S3 allows for direct distribution of such to users. Yet often enough, the raw S3 URLs are not pretty enough to use them in production environments. A simple CNAME entry can often help to resolve this and allows for a simple, custom-domain access for users. Optionally, S3 can serve as the source for a Content Delivery Network (CDN), which allows for accelerated download speed.

As of today, the current system in place provides binaries for

- Ubuntu 24.04
- Ubuntu 22.04
- RHEL8
- RHEL9
- Alpine 3.20
- Alpine 3.21

for both `x86_64` and `arm64` architectures. The download speed is 5-10x faster than when downloading binaries from the Posit Package Manager. The build process is fully transparent (https://ci.devxy.io/repos/7).

Project plan

Start-up phase

The project is already up and running with respect to the core components and infrastructure.

Technical delivery

An up-and-running global package binary download service which processes daily incoming changes of CRAN packages (additions / removals / updates).

For the core engine behind, continuous releases and improvements.

Other aspects

The project would profit from more public promotion and user testing/use.

Presentations at common R conferences are planned and are important for establishing the project within the community.

Requirements

General

- Support for continuous development of the code engine
- Cost coverage for the public running service building CRAN packages for various distributions and architectures

People

Additional people which ensure an upkeep of the public package build service and the underlying servers.

The project can be run by a single person technically, more people would be helpful to improve the project code and robustify the service maintenance.

Tools & Tech

- For the public CRAN build service, a cloud infrastructure (already existing) which provides native `x86_64` and `arm64` runners that are able to natively build the respective packages. The servers need decent CPU power to minimize build times and being able to process all existing packages, in scenarios of major OS version updates, in a reasonable time.
- A (professionally) managed database storing the build metadata (already in place).
- S3 storage in the cloud (currently using Hetzner, 1TB/m = 5 EUR).
- UI frontend showing available packages (optional). Currently, a self-built Shiny App Dashboard is in use (https://app.devxy.io/app/r-package-binaries-dashboard)

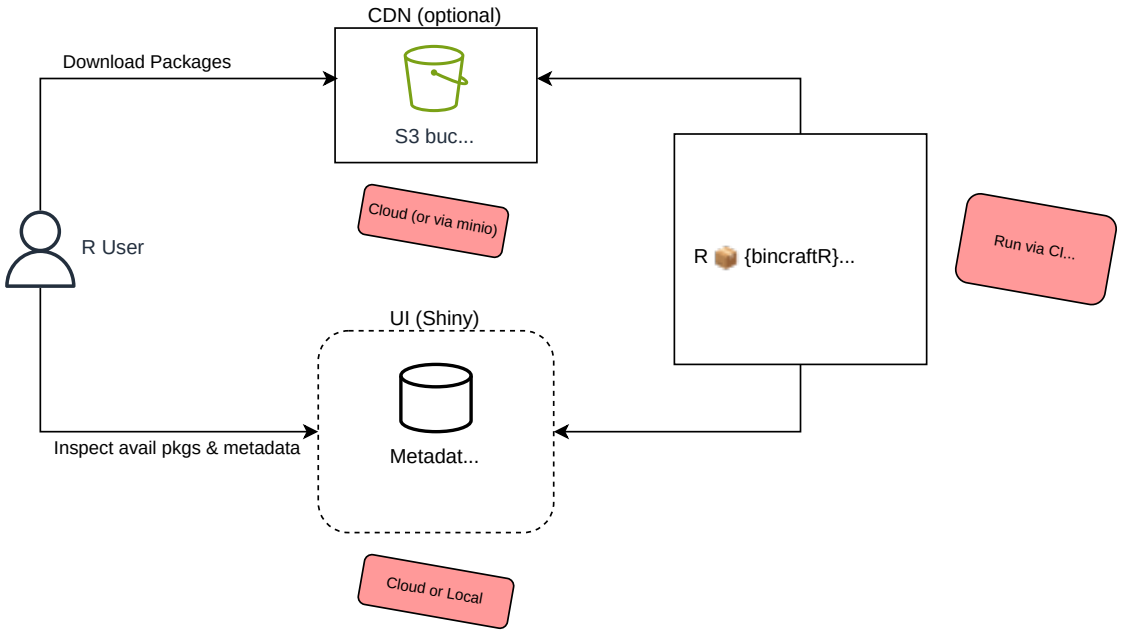

Figure 1: High level architecture diagram

Funding

There are two main types of funding:

1. General maintenance and improvement of the "engine" and its related documentation
2. Coverage of cloud costs for the running service providing binaries for CRAN packages

For (1), I do not know if there are hard-coded policies in place for human workforce costs per project/hour. It would surely be great to have financial support for this, but overall, (2) is more important in my opinion.

For (2), I'll outline a cloud cost overview which will be able to efficiently provide the technical resources to run the overall CRAN build service for the forseeable future:

Name	~ Cost/month	~ Costs / year	Notes
`x86_64` Server (8 cores, 16 threads, 64 GB RAM)	54 Euro	648 Euro	Used to build `x86_64` binaries, bare-metal root server
x2 `arm64` Server (16 cores, 32 GB RAM)	52 Euro	624 Euro	Used to build `arm64` binaries, shared Cloud Server, price for two units
Database Servers (HA)	12 Euro	144 Euro	HA Database for build metadata
S3 storage costs	15 Euro	180 Euro	Costs for 3 TB, current storage is at 2.14 TB

Name	~ Cost/month	~ Costs / year	Notes
CDN costs	?? Euro	?? Euro	CDN bandwith/transfer costs. 10$/1 TB bandwith, + local storage zones around the world

A note on the arm64 servers: while the mentioned ones are (even only one of them) is sufficient for processing the daily workloads, more power is needed for a full rebuild of new major OS versions. Having a RX170 available, would make all of this a blast. (Similar conditions apply for the rebuilds of x86 packages.) It is a question of money in the end, even though the costs are somewhat on the smaller scale, given the many resources which can be processed with it, especially in relation to other common cloud provider costs.

While the core setup costs (servers, S3, database) should stay quite static at ~ 150 / month (1800 / year) (for the minimal setup), the bandwith costs for fast downloads around the world will be dynamic. If the service is picked up by the community on a large scale, costs of multiple hundreds per month could occur, when with a cost-efficient CDN provider. One way to limit this is by setting global limits, though as a public service aiming to be the "go-to" destination for binaries, this would be hard to communicate. In the end, substantial financial support from institutions support the R ecosystem would be needed. (I'd argue personally that providing a professional and fast package repository is a key interest of each language and every penny spent on it is well invested.)

All in all: cloud cost coverage between 2-3k per year, human work costs of ~15k for this grant period to professionalize the core engine, extend documentation and establish the project. Seeing this project getting funded would also allow for more concrete planning of onboarding additional people, increasing the human bus-factor.

Summary

The project would greatly benefit from dedicated resources to support ongoing development of the core engine, as well as to cover ongoing cloud infrastructure costs.

Additionally, increasing the number of individuals with access to the core components of the system would improve resilience, reducing the risk of disruptions due to single points of failure or individual unavailability.

Success

Definition of done

1. A running, public build service providing R package binaries for Linux to the community which has caused minimal interruptions for users across a timespan covering multiple OS updates.
2. A framework/engine for building binary packages, written in R, with extensive documentation around its usage and optional extensions (e.g. metadata database), allowing users to easily host and use their own package repository.

Measuring success

- Download counts

- Tracking missing/corrupted packages in the public service
- Gathering user feedback on the overall process of creating custom user repositories

Future work

Encouraging community contributions could play a key role in sustaining the project over the long term. Ultimately, it would be fantastic to have an open, free, and professional-grade tool that not only supports the creation and maintenance of custom user repositories, but also delivers a high-performance binary package repository for CRAN packages out of the box.

Key risks

The project faces several potential risks:

- Gaining broad acceptance within the community and building trust in the service and its build process
- Competition from established players in the space (e.g., Posit) offering proprietary alternatives
- Bus factor: ensuring that multiple individuals have access to all critical components of the infrastructure to prevent reliance on a single person
- Uncertainty around long-term funding to cover operational costs—particularly bandwidth expenses, which could become significant if community adoption grows substantially