Three Levels of Computational Mobility Using R

Cole Brokamp

Division of Biostatistics and Epidemiology
Cincinnati Children's Hospital Medical Center

November 14th, 2020

Table of Contents

What Is Computational Mobility?

- "The ability to define, create, and mantain a workflow locally while remaining confident that the workflow can be executed on different hosts"★
- Contain the entire software stack, including data, code, executables, software libraries *and* reliably move it from system to system
- R Users: versions of R packages, system dependencies, data, and R itself
- Computational mobility ♡ reproducibility

★Kurtzer, Sochat, Bauer. Singularity: Scientific containers for mobility of compute. PLoS ONE. 2017.

Examples

- moving a local R project to a high performance computer
- sending a project to a collaborator
- deploying a Shiny application
- deploying R code to a continuous integration server
- running an project that contains code written several years ago with an older version of R
- creating a research compendium to ensure the reproducibility of your project
- putting your R project into a Docker or Singularity container

Table of Contents

Level 1: Static Package Repositories

- Fixed CRAN repository snapshots
  - MRAN Time Machine
  - RStudio Package Manager
- Roll your own repository
  - /eddelbuettel/drat
  - /andrie/miniCRAN

```
options(
  repos = c(
    CRAN = "https://mran.revolutionanalytics.com/snapshot/YYYY-MM-DD"
  )
)
```

Level 1: Static Package Repositories

- ✔ package installations will always use same versions
- ✔ doesn't require extra software or R packages
- ✔ low implementation overhead

- ✖ rely on user to set repo
- ✖ interferes with other R project libraries
- ✖ only CRAN packages supported
- ✖ all package versions have to be on CRAN on the same date
- ✖ doesn't track system dependencies or R versions

Level 2: Helper Packages

- Specific file dedicated to documenting versions
- Automatic or manual dependency management
- Use private library specific to R project
- /rstudio/renv
  - `renv.lock` file
  - `renv::snapshot()` and `renv::restore()`
- /cole-brokamp/dep
  - `DESCRIPTION` file
  - `dep::ends()` and `dep::loy()`

Level 2: Helper Packages

- ✔ any package repository, any package version
- ✔ use version control for versions
- ✔ private library isolates R project
- ✔ system dependencies tracked

- ✖ requires infrastructure code setup
- ✖ still relies on online code availability

Level 3: Containers

- Software that contains entire software stack, including data, code, executables, software libraries
- Like a VM, but lighter-weight framework for scalability
- Definition files are instructions to bootstrap entire environment

- Docker, Singularity, OCI
- Several R packages for auto-generating definition files, creating containers, and managing Docker

Level 3: Containers

- ✔ complete isolation from OS
- ✔ no internet required to restore env
- ✔ long-term archives
- ✔ easier to move between systems

- ✖ requires container software on host
- ✖ high development effort
- ✖ relatively large file size
- ✖ same definition file doesn't always guarantee same container

Table of Contents

What is the best approach?

- Types of R packages needed to document
- Destination environment
- Reproducibility versus complexity
- Levels can work separately *and* together
- Stable and sustainable

Thank You

cole.brokamp@cchmc.org

https://colebrokamp.com

@cole_brokamp

@cole-brokamp