Using package in R is easy. You install from CRAN using install.packages("packagename"), it resolves dependencies and you’re good to go. What R natively doesn’t handle so well is installing a particular package version without jumping through hoops. Technically you need the source file of the package version you want to install AND all source files of the dependencies (in the correct version, of course). This has been made almost seamless with packages packrat and recently, renv.

This comes handy when you are constructing a Docker file to run in production. Usually you want to run this defensively and do not want things to change from one image build to another. To get there, you can save all your package names and version into a file (renv.lock) and use that to reconstruct the now defined package structure with predictable versions.

Below you’ll find modifications I found necessary to get my packages installed.

My workflow is designed in such a way that I have a Dockerfile located in a folder above my R project I want to deploy. I install all the necessary tools and copy the application into the Docker image. One caveat I noticed when doing this, hidden files were copied too, and .Rprofile (more info i.e. [here](http://www.onthelambda.com/2014/09/17/fun-with-rprofile-and-customizing-r-startup/)) was causing me a lot of grief. For example, it contained commands that were looking for renv/activate.R script, which was, naturally, non existent because that’s not something you commit to your git repository. This was solved by overwriting it (or deleting it).

Here is an example that is working for me. See if you can find anything useful in it.

FROM rocker/r-ver:3.6.2

MAINTAINER Yours Truly "[yours.truly@checksnbalances.com](mailto:yours.truly@checksnbalances.com)"

# Here is where I install all the necessary system libraries needed

# by R packages. Don't worry, R will, after compiling for 30

# minutes and file, tell you what packages you would need.

RUN apt-get update && apt-get install -y \

zlib1g-dev \

libcurl4-openssl-dev \

libssl-dev

# This is something I use to deploy apps to shinyproxy. It is

# probably something that could be avoided by specifying host and

# port in `runApp()`. Note that the location may be OS dependent.

RUN echo 'local({options(shiny.port = 3838, shiny.host = "0.0.0.0")})' >> /usr/local/lib/R/etc/Rprofile.site

# Your code should not be run by root, so creating and switching to

# a new user. Feel free to come up with your own fun ID.

RUN useradd -m -u 2000 poldeta

USER poldeta

# Recursively make an R library folder. This is where installed R

# packages will be stored.

RUN mkdir /home/poldeta/R/library -p && mkdir /home/poldeta/shinyapp

# Moving to the app folder is probably not necessary at this

# particular point, but you know, whatever.

WORKDIR /home/poldeta/shinyapp

# Create .Rprofile site that will include your favorite (writable)

# location for installed R packages

RUN echo ".libPaths('/home/poldeta/R')" >> .Rprofile && R -e "install.packages('renv')"

# Switch to a superuser and copy your application into your Docker

# image.

USER root

COPY shinyapp /home/poldeta/shinyapp

# Make sure folder and file permissions are set to your new username.

RUN chown -R poldeta:poldeta /home/poldeta/shinyapp

# This is where the magic happens. When copying the app into the

# docker image, `renv.lock` file was also transferred. Because is

# being called from the working directory where `renv.lock` is

# located, calling `restore()` with defaults makes everything work

# as intended.

USER poldeta

RUN R -e "renv::restore()"

# Do any other necessary things to your image.

EXPOSE 3838

# Finally, run the app to be served.

CMD ["R", "-e", "shiny::runApp('/home/poldeta/shinyapp')"]