As a package maintainer you might be observing an increasing number of questions raised by people that have recently migrated to R 4.0.0 and are now trying to get your package to work. Yet, rhub::check\_with\_rrelease() currently still uses R 3.6.3 as test base. While migrating to a new R version is always tempting maybe you don’t feel like disrupting your development environment just now as you have even more fun things to do. Fear not! A quick docker container can help you to test your package and its functionality without touching your main environment.

If you are familiar with R and docker containers, your first guess might be to simply spin up an instance of rocker/r-devel:latest from the command line, but:

docker run -ti rocker/r-devel:latest

...

R version 3.6.3 (2020-02-29) -- "Holding the Windsock"

Copyright (C) 2020 The R Foundation for Statistical Computing

Platform: x86\_64-pc-linux-gnu (64-bit)

...

It appears as if the rocker images are still on R 3.6.3. So, we need to build our own docker container. Luckily, [Dirk Eddelbuettel has provided all the pointers that we need for that in a recent post](http://dirk.eddelbuettel.com/blog/2020/04/11/#024_rocker_r_4_0_0_testing).

First, you need a working environment with docker installed. I won’t go into detail here as I think that the [documentation provided on the docker homepage](https://www.docker.com/) is already fairly extensive. Next, create a directory with a fancy name (e.g., docker\_r4test) and copy the following into a text file with the name Dockerfile.

FROM rocker/r-ubuntu:18.04

MAINTAINER Joachim Gassen "[gassen@wiwi.hu-berlin.de](mailto:gassen@wiwi.hu-berlin.de)"

# Update packages lists

RUN apt-get update

# Latex for pdflatex (needed for R CMD check)

RUN apt-get install -y --no-install-recommends\

texlive-latex-base texlive-fonts-recommended texlive-fonts-extra \

texlive-latex-extra texinfo

# Extra stuff that your package or its dependencies need

RUN apt-get install -y --no-install-recommends \

libxml2-dev libcurl4-openssl-dev libssl-dev

# Update R to 4.0 - from

# <http://dirk.eddelbuettel.com/papers/r4_r400_binaries.pdf>

# Update to current repo state

RUN apt-get dist-upgrade -y

# then add the edd/r-4.0 PPA

RUN add-apt-repository -y ppa:edd/r-4.0

# upgrade again

RUN apt-get dist-upgrade -y

# Add the R remotes package to eventually install your package if it resides

# on Github

RUN R -e "install.packages('remotes')"

# Install dependencies of your package (yes.. {tidycovid19} \_is\_

# dependency heavy ...)

RUN R -e "install.packages(c('xml2', 'rvest', 'readxl', \

'magrittr', 'dplyr', 'tidyr', 'countrycode', \

'ggplot2', 'wbstats', 'gghighlight', 'gtrendsR', \

'ggrepel', 'readr', 'rlang', 'scales', 'lubridate', \

'shiny', 'shinyWidgets', 'shinyjs', 'rclipboard', \

'stringr', 'zoo', 'maps', 'gganimate'))"

# Once you know that your package works you can

# also install it here.

# RUN R -e "remotes::library('tidycovid19')"

But is should be  
relatively straightforward to adjust it to your needs. Just change the  
maintainer and the packages that your package depends on. If you are using  
{knitr} to build your vignettes, make sure that you install this as well.

Then, go to the command line, cd in your new directory and run

docker build -t r4test .

Do not forget the dot indicating the path to where your Dockerfile resides. Obviously, you can change the tag to another name if you prefer.

Building the docker container will take a while and might fail. If it fails, it is most likely because your package or one of its dependencies need additional OS components, e.g., special libraries. In most cases the failing package install process gives you a hint on which Ubuntu package to add. Obey and simply add them to the RUN apt-get install line indicated in the Dockerfile.

Eventually, you will have successfully build your very own R 4.0.0 docker container. Congratulations! At the command line, spin up your new container using the directory containing the project folder of your package, *not* your package directory itself, as your exported working volume. This is the part of the -v option prior to the colon. I name this directory /projects in the docker container but you can give it any name you like. The -w switch causes docker to use it as your start up path before calling the command that is given in the last parameter (a shell in our case). **NOTE**: By starting the below, you provide docker access to your current directory. Every change that you make to this directory from within docker are just as permanent as changes that you do directly. Tread lightly.

docker run -ti -v ~/github:/projects -w /projects --name my\_r4test\_instance r4test bash

If you provided the same tag that you used above (r4test in the example), you should be greeted with a prompt:

root@026a999fe336:/projects#

Super! You are now “inside” your docker container. Try it by issuing a first command, e.g. ls to see the content of your work directory. It should look familiar.

What to do next? Well, a first step could be to fire up R (issue R command) to verify that you actually have R 4.0.0 installed in your container.

root@026a999fe336:/projects# R

R version 4.0.0 (2020-04-24) -- "Arbor Day"

Copyright (C) 2020 The R Foundation for Statistical Computing

Platform: x86\_64-pc-linux-gnu (64-bit)

...

This looks good. Exit R quit() and don’t save your workspace image. If you like to develop CRAN type clean packages a next good step would be to build and check your package. As R CMD check also tests your examples provided in the documentation, you can be reasonably sure that, if R CMD check passes, your package “basically works” under R 4.0.0.

The first step is to build your package. The parameter after the build command is the local path to your package directory.

root@026a999fe336:/projects# R CMD build tidycovid19

If this fails this is most likely because you have not included all dependencies in the Dockerfile. Add the missing ones. As docker caches the steps when building the Dockerfile, you can speed up the rebuild by adding the missing packages as a separate RUN R -e "install.packages()" step at the end of the Dockerfile.

After you have successfully built your package, you will have a gzipped tar file containing your package in your project directory (now it is a good time to verify that you are seeing this file also outside of docker to understand that your docker container can make changes to your files). Next, run R CMD check to the gzipped file to see how your package and R 4.0.0 get along. Good luck!

root@026a999fe336:/projects# R CMD check tidycovid19\_0.0.0.9000.tar.gz

As the feedback from R CMD check varies slightly across platforms, you might receive some notes, warnings and maybe even errors that are unrelated to R 4.0.0 if you are not using Ubuntu 18.04 as your main development environment. In most cases, these are informative as well and will motivate you to improve and streamline your package code. For this, it is helpful that you can always access the results of the check from outside the docker container by exploring the contents of the [yourpackage].Rcheck directory.

Eventually, you might be greeted by:

root@026a999fe336:/projects# R CMD check tidycovid19\_0.0.0.9000.tar.gz

... [many lines of checks] ...

Status: OK

Congratulations! Your package checks without errors notes and warnings on Ubuntu 18.04/R4.0.0!

The advantage of using your own docker container is that you do not have to stop here. Instead, you can now install your package in the docker container, e.g. by uncommenting and editing the last line of the Dockerfile above or by running install.packages("your\_package\_path", type = "source", repos = NULL) from within R and then test some code using your package to see whether it behaves as you expect. The container will also be helpful in case users identify issues in certain parts of your package while running R 4.0.0. You could even think about extending the Dockerfile to also include an RStudio environment so that you can use it as your R 4.0.0 test development environment.

When you are done with R 4.0.0 and your docker container, run exit from the docker shell to return your computer’s command line. From there, at any time, you can restart the container with docker start -i my\_r4test\_instance The container will start at the state where you left it.

If you want to permanently delete your container, run docker rm my\_r4test\_instance from your command line. This will delete the container but not the image from which the container was created (tagged with r4test in our example). If you want to remove the image as well, e.g. to free up disk space, run docker rmi r4test. This will remove the tagged image and all of its un-tagged parent images, meaning in our case for example the rocker/r-ubuntu:18.04 image.

This is it. Have fun exploring R 4.0.0 with docker!