



USING R VIA ROCKER

A BRIEF INTRODUCTION TO DOCKER FOR R

Dirk Eddebuettel

Chicago R User Group Meeting

26 November 2019

http://dirk.eddelbuettel.com/papers/chirug_nov2019_rocker.pdf

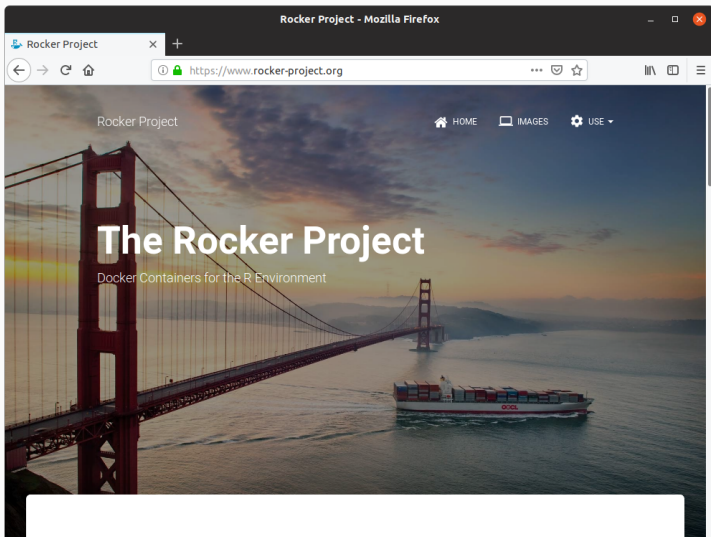
Brief Outline

- Docker Intro: Appeal of Containers, “Lego” blocks, ...
- Brief Overview of Docker and its commands
- Building and Customizing via two “blogged” examples
 - Debug with gcc-9 on macOS
 - Easy installation of “heavy” packages
- Going Further: More about Rocker

A Personal Timeline

- Docker itself started in 2013
- I started experimenting with it around spring of 2014
- Was convinced enough to say *'will change how we build and test'* in [keynote at useR! 2014 conference](#) in mid-2014
- Started the [Rocker Project](#) with Carl Boettiger fall 2014
- Gave three-hour [tutorial at useR! in 2015](#)
- Active development of multiple (widely-used) containers
- [Introductory paper](#) in R Journal, 2017

ROCKER PROJECT



Source: <https://www.rocker-project.org>

The screenshot shows a Mozilla Firefox browser window with the address bar displaying `https://journal.r-project.org/archive/2017/RJ-2017-065/index.html`. The page content includes the R logo, a navigation menu, and the article title "The R Journal: article published in 2017, volume 9:2". The article title is followed by a link to the full article: "An Introduction to Rocker: Docker Containers for R". Below this, the authors "Carl Boettiger and Dirk Eddelbuettel" and the publication details "The R Journal (2017) 9:2, pages 527-536." are listed. The abstract states: "We describe the Rocker project, which provides a widely-used suite of Docker images with customized R environments for particular tasks. We discuss how this suite is organized, and how these tools can increase portability, scaling, reproducibility, and convenience of R users and developers." The article was received on 2017-10-12 and online on 2017-11-27. It lists CRAN packages: `packrat`, `rhub`, and `tidyverse`. CRAN Task Views implied by cited CRAN packages include `ReproducibleResearch`. The article is licensed under a Creative Commons Attribution 4.0 International license. A code block at the bottom of the page shows the following R metadata:

```
@article{RJ-2017-065,
  author = {Carl Boettiger and Dirk Eddelbuettel},
  title = {{An Introduction to Rocker: Docker Containers for R}},
  year = {2017},
  journal = {{The R Journal}},
  doi = {10.32614/RJ-2017-065},
  url = {https://doi.org/10.32614/RJ-2017-065},
  pages = {527--536},
  volume = {9},
  number = {2}
}
```

Source: <https://journal.r-project.org/archive/2017/RJ-2017-065/index.html>

An Introduction to Rocker: Docker Containers for R

by Carl Boettiger, Dirk Eddelbuettel

Abstract We describe the Rocker project, which provides a widely-used suite of Docker images with customized R environments for particular tasks. We discuss how this suite is organized, and how these tools can increase portability, scaling, reproducibility, and convenience of R users and developers.

Introduction

The Rocker project was launched in October 2014 as a collaboration between the authors to provide high-quality Docker images containing the R environment (Boettiger and Eddelbuettel, 2014). Since that time, the project has seen both considerable uptake in the community and substantial development and evolution. Here we seek to document the project's objectives and uses.

What is Docker?

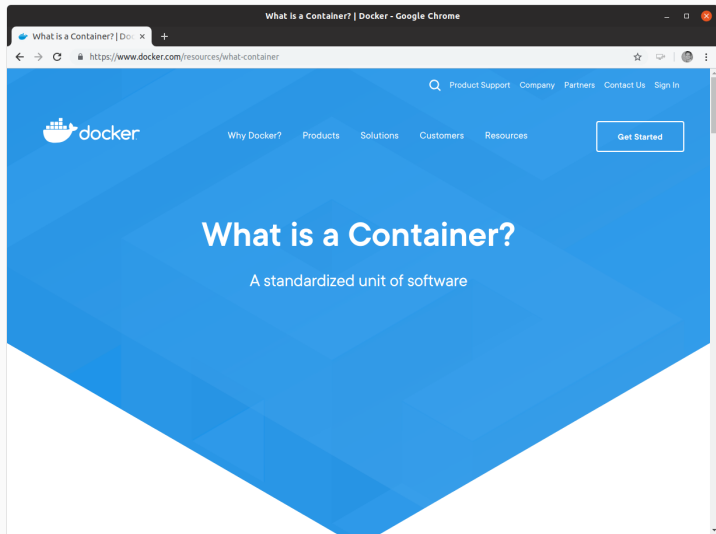
Docker is a popular open-source tool to create, distribute, deploy, and run software applications using *containers*. Containers provide a virtual environment (see Clark et al. (2014) for an overview of common virtual environments) requiring all operating-system components an application needs to run. Docker containers are lightweight as they share the operating system kernel, start instantly using a layered filesystem which minimizes disk footprint and download time, are built on open standards that run on all major platforms (Linux, Mac, Windows), and provide an added layer of security by running an application in an isolated environment (Docker, 2015). Familiarity with a few key terms is helpful in understanding this paper. The term "container" refers to an isolated software environment on a computer. R users can think of running a container as analogous to loading an R package; a container is an active instance of a static Docker image. A Docker "image" is a binary archive of that software, analogous to an R binary package: a given version is downloaded only once, and can then be "run" to create a container whenever it is needed. A "Dockerfile" is a recipe, the source-code, to create a Docker image. Pre-built Docker images are publicly available through Docker Hub, which plays a role for central distribution similar to CRAN in our analogy. Development and contributions to the Rocker project focus on the construction, organization and maintenance of these Dockerfiles.

Source: <https://journal.r-project.org/archive/2017/RJ-2017-065/index.html>

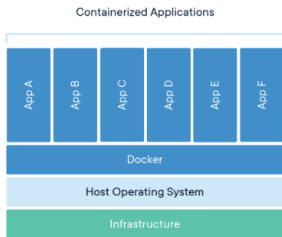
So what is it?

- Think of its 'containers' as something portable like a zipfile
- A 'container' allows you to execute code based on what is in it
- *Portable*: same container used on Linux, Window, macOS
- *However* this really shines on Linux:
 - as it requires only a *very* thin layer above the operating system
 - on macOS and Windows intermediating layer has to be provided
 - heavy usage in cloud deployments
- Still, what is phenomenal are the
 - portability
 - encapsulation
 - security
 - reproducibility

DOCKER INTRO



Source: <https://www.docker.com/resources/what-container>



Package Software into Standardized Units for Development, Shipment and Deployment

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at runtime and in the case of Docker containers - images become containers when they run on [Docker Engine](#). Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

Docker containers that run on Docker Engine:

- **Standard:** Docker created the industry standard for containers, so they could be portable anywhere
- **Lightweight:** Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs
- **Secure:** Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry

Source: <https://www.docker.com/resources/what-container>

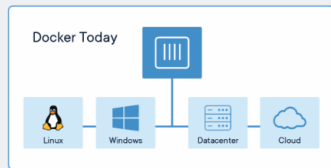
Docker Containers Are Everywhere: Linux, Windows, Data center, Cloud, Serverless, etc.

Docker container technology was launched in 2013 as an open source [Docker Engine](#).

It leveraged existing computing concepts around containers and specifically in the Linux world, primitives known as cgroups and namespaces. Docker's technology is unique because it focuses on the requirements of developers and systems operators to separate application dependencies from infrastructure.

Success in the Linux world drove a partnership with Microsoft that brought Docker containers and its functionality to Windows Server (sometimes referred to as [Docker Windows containers](#)).

Technology available from Docker and its open source project, Moby has been leveraged by all major data center vendors and cloud providers. Many of these providers are leveraging Docker for their container-native IaaS offerings. Additionally, the leading open source serverless frameworks utilize Docker container technology.

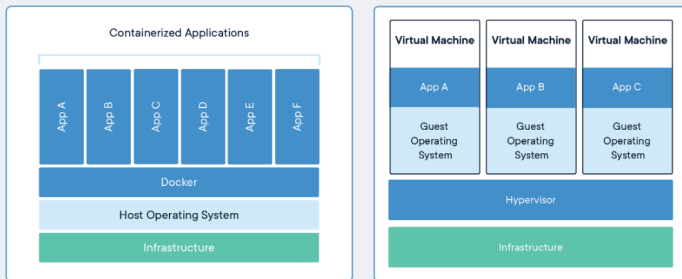


Source: <https://www.docker.com/resources/what-container>

Comparing Containers and Virtual Machines

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware.

Containers are more portable and efficient.



Source: <https://www.docker.com/resources/what-container>

Simplifying Somewhat:

- A container can run a single process
 - (as opposed to a virtual machine behaving more like a whole computer system)
- So it helps to think of Docker *encapsulating a single command*
 - (though that command may spawn more commands)
- Docker containers can be orchestrated and combined
 - each container can provide its services on a network port
 - common pattern may be one for database, one for webserver, ...)

Some Informal Definitions

- *Image* is a provided Docker run-time; can be built locally or downloaded
- *Container* is (possibly) stateful instance of a container, either running or suspended
- We will be a little sloppy and use container and image interchangeably
- On the hand, a *virtual machine*, on the hand, tends to be a heavier software layer provide a full virtual system. VMware and VirtualBox are two well-known systems.

DOCKER: BASIC COMMANDS

Basic commands

- `docker help` lists the available commands
- `docker images` lists installed images
- `docker run` runs a container (with extra args, see below)
- `docker ps` shows currently running containers
- `docker pull someuser/somecontainer:version`
imports container (`version` optional; `latest` is default)
- `docker build` to create a new container
- `docker rm container` removes a container
- `docker rmi imageid` removes an image

docker images

- list installed containers, versions, sizes
- very helpful for quick overview
- can also list sub-sets per repository and/or tag

docker run

- Bread and butter command to use Docker
- Common arguments
 - `--rm` to remove artifacts after run (“clean up”)
 - `-ti` to add *terminal* and *interactive* use
 - `-v LocalDir:MountedDir` to make local dir available
 - `-w WorkDir` to switch to workdir
 - `-p 8787:8787` to publish container port 8787 as host port 8787
 - `container/tag:version`
 - `cmdline arguments` for container application
 - *plus many more options* so see documentation
- When named container is not locally installed it is pulled

`docker pull` (and `docker commit`)

- Main command to obtain images from repository / registry
- By default uses `hub.docker.com` / `cloud.docker.com` registries
- Note that pulled containers can be altered and saved via `docker commit`

`docker pull` (and `docker commit`)

- Main command to obtain images from repository / registry
- By default uses `hub.docker.com` / `cloud.docker.com` registries
- Note that pulled containers can be altered and saved via `'docker commit'`

docker build

- Principal command to create new images
- Containers are 'layered':
 - easy to start from existing container making small change
 - creating new augmented or adapted container
- Input is a text file **Dockerfile**
- Many tutorials available to get started

USE CASE ILLUSTRATIONS

Use multiple R versions

- *E.g.* test an R package against multiple R releases
- test code against current and development versions of tools
- access to different R versions via different **r-base** containers
- just specify different *tags* for different R versions
- (Rocker also has another stack for explicitly versioned images)

```
$ docker run --rm -ti r-base:latest R --version | head -1
R version 3.6.1 (2019-07-05) -- "Action of the Toes"
$ docker run --rm -ti r-base:3.5.3 R --version | head -1
R version 3.5.3 (2019-03-11) -- "Great Truth"
$ docker run --rm -ti r-base:3.4.2 R --version | head -1
R version 3.4.2 (2017-09-28) -- "Short Summer"
$
```

Test against development versions

- Sometimes we want to test against new development versions
- These versions may still be unfinished and undergo changes
- This makes using them in a ‘sandbox’ ideal – great container use

```
edd@rob:~$ docker run --rm -ti rocker/drd:latest RD --version | head -4
R Under development (unstable) (2019-11-23 r77455) -- "Unsuffered Consequences"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

edd@rob:~$
```

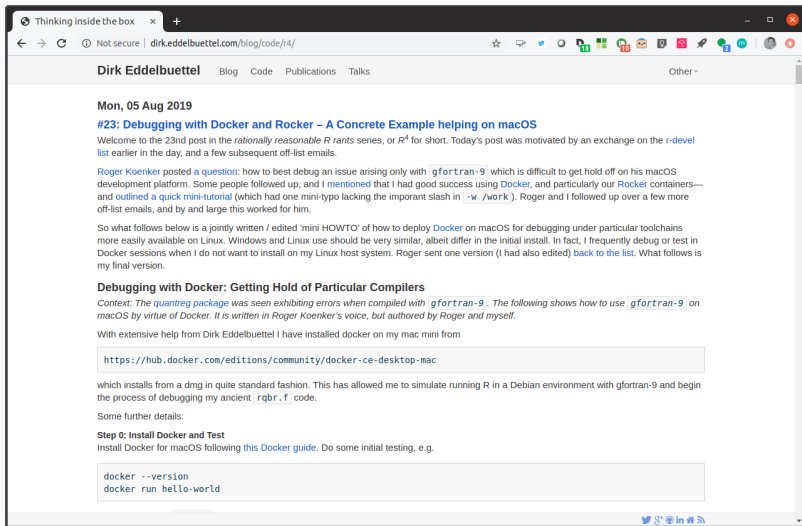
(This shows the November 23 sources of R-devel. So with very little effort we get access to recent development versions—as the container builds are triggered weekly by a **crontab** entry invoking a web trigger at hub.docker.com.)

Test with special versions

- We have R containers for ‘undefined behavior sanitizer’ (UBSAN)
- This uses R built with particular compilation options which can detect ‘undefined behavior’
- Similarly, R has another checker ‘rchk’ – and a container

MOTIVATING EXAMPLES

DEBUGGING WITH NEWER / DIFFERENT TOOLCHAINS



Thinking inside the box x +

Not secure | dirk.eddebuettel.com/blog/code/r4/

Dirk Eddebuettel Blog Code Publications Talks Other -

Mon, 05 Aug 2019

#23: Debugging with Docker and Rocker – A Concrete Example helping on macOS

Welcome to the 23rd post in the *rationally reasonable R rants* series, or R^4 for short. Today's post was motivated by an exchange on the *r-devel* list earlier in the day, and a few subsequent off-list emails.

Roger Koenker posted a [question](#): how to best debug an issue arising only with `gfortran-9` which is difficult to get hold off on his macOS development platform. Some people followed up, and I [mentioned](#) that I had good success using `Docker`, and particularly our `Rocker` containers—and [outlined a quick mini-tutorial](#) (which had one mini-typo lacking the important slash in `-w /work`). Roger and I followed up over a few more off-list emails, and by and large this worked for him.

So what follows below is a jointly written / edited 'mini HOWTO' of how to deploy `Docker` on macOS for debugging under particular toolchains more easily available on Linux. Windows and Linux use should be very similar, albeit differ in the initial install. In fact, I frequently debug or test in `Docker` sessions when I do not want to install on my Linux host system. Roger sent one version (I had also edited) [back to the list](#). What follows is my final version.

Debugging with Docker: Getting Hold of Particular Compilers

Context: The [quantreg](#) package was seen exhibiting errors when compiled with `gfortran-9`. The following shows how to use `gfortran-9` on macOS by virtue of `Docker`. It is written in Roger Koenker's voice, but authored by Roger and myself.

With extensive help from Dirk Eddebuettel I have installed docker on my mac mini from

```
https://hub.docker.com/editions/community/docker-ce-desktop-mac
```

which installs from a dmg in quite standard fashion. This has allowed me to simulate running R in a Debian environment with `gfortran-9` and begin the process of debugging my ancient `rqr.f` code.

Some further details:

Step 0: Install Docker and Test

Install Docker for macOS following [this Docker guide](#). Do some initial testing, e.g.

```
docker --version
docker run hello-world
```

Twitter GitHub LinkedIn

Context

- Roger Koenker gets CRAN email about issue with a new compiler
- He works on macOS without easy access to new `gcc` versions
- Building `gcc` from source a bit painful
- But what *if we could just run it* ?
- I email Roger, give some pointers and hints ...
- ... and he fixed the issue
- So we wrote a [blog post](#)

More detail (see the [blog post](#) for more)

- Step 0: Get Docker installed (which is easy-ish on mac/win/lin)
- Step 1: Install Rocker's **r-base** container with current R
- Step 2: Use Docker options `-vO:I -wI` to
 - map 'outer' dir O, say, `~/proj/abc` to 'inner' I, say `/work`
 - start Docker session in 'inner' dir, say `/work`
- Step 3: Update with **apt**, install `gcc-9 + gfortran-9`
- Step 4: Deal with build dependencies for the package
- Step 5: Set compiler flags in `~/R/Makevars`
- Step 6: Install R source package in question (here: [quantreg](#))
- Step 7: Debug issue at hand and solve problem

USING ROCKER WITH PPAs

Thinking inside the box x +

Not secure | dirk.eddelbuettel.com/blog/code/r4/

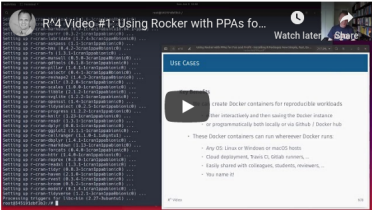
Dirk Eddelbuettel Blog Code Publications Talks Other -

Sun, 09 Jun 2019

#22: Using Rocker and PPAs for Fun and Profit

Welcome to the 22nd post in the *reasonably rational R recommendations* series, or R^4 for short.

This post premieres something new: [a matching video](#) in lightning talk style:



The topic is something we had mentioned a few times before in [this \$r^4\$ blog series](#), for example in [this post on finding deb packages](#) as well as in [this post on binary installations](#). Binaries rocks, where available, and [Michael Rutter's PPAs](#) should really be known and used more widely. Hence the video and supporting slides.

[/code/r4](#) | permanent link

Twitter GitHub LinkedIn YouTube

More detail

- This illustrates what we touched upon earlier
- Installing, say, `r-cran-rstan` from binary is
 - a **single and fast** step
 - as opposed to compiling from source
- Another famous example also shown: `r-cran-tidyverse`
- Several blog posts describe approaches
- “Seeing is believing” so I made a video and slides
- Video demonstrates installation “live” and backed by slides

DETAILED EXAMPLE: RSTAN

We fire up our `r-base` container for a working basic R installation:

```
edd@rob:~$ docker run --rm -ti r-base

R version 3.5.3 (2019-03-11) -- "Great Truth"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```


Interactively, we ask R to install `rstan`

```
> install.packages("rstan")
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)
'stringr', 'labeling', 'munsell', 'RColorBrewer', 'fanshi', 'pillar', 'pkgconfig',
'backports', 'processx', 'assertthat', 'magrittr', 'digest', 'gtable', 'lazyeval',
'plyr', 'reshape2', 'rlang', 'scales', 'tibble', 'viridisLite', 'withr',
'matrixStats', 'checkmate', 'callr', 'cli', 'crayon', 'desc', 'prettyunits', 'R6',
'rprojroot', 'ggplot2', 'StanHeaders', 'inline', 'gridExtra', 'Rcpp', 'loo',
'pkgbuild', 'RcppEigen', 'BH'

trying URL 'https://cloud.r-project.org/src/contrib/glue_1.3.1.tar.gz'
Content type 'application/x-gzip' length 122950 bytes (120 KB)
=====
downloaded 120 KB

trying URL 'https://cloud.r-project.org/src/contrib/stringi_1.4.3.tar.gz'
Content type 'application/x-gzip' length 7290890 bytes (7.0 MB)
=====
downloaded 7.0 MB

[... many more downloads omitted ...]
```

We ask R to install `rstan` (continued)

```
[... quite a bit of compilation later ...]
g++ -std=gnu++14 -shared -L/usr/lib/R/lib -Wl,-z,relro -o rstan.so chains.o init.o lang__ast_def.o
lang__grammars__bare_type_grammar_inst.o lang__grammars__expression07_grammar_inst.o
lang__grammars__expression_grammar_inst.o lang__grammars__functions_grammar_inst.o
lang__grammars__indexes_grammar_inst.o lang__grammars__program_grammar_inst.o
lang__grammars__semantic_actions.o lang__grammars__statement_2_grammar_inst.o
lang__grammars__statement_grammar_inst.o lang__grammars__term_grammar_inst.o
lang__grammars__var_deccls_grammar_inst.o lang__grammars__whitespace_grammar_inst.o misc.o
pointer-tools.o sparse_extractors.o stanc.o -L/usr/lib/R/lib -lR
```

```
installing to /usr/local/lib/R/site-library/rstan/libs
```

```
** R
** inst
** byte-compile and prepare package for lazy loading
** help
*** installing help indices
*** copying figures
** building package indices
** installing vignettes
** testing if installed package can be loaded
* DONE (rstan)
```

```
The downloaded source packages are in '/tmp/Rtmpo38sEq/downloaded_packages'
```

```
>
```

We ask R to install `rstan` (continued)

```
> library(rstan)
Loading required package: ggplot2
Use suppressPackageStartupMessages() to eliminate package startup
messages.
Loading required package: StanHeaders
rstan (Version 2.18.2, GitRev: 2e1f913d3ca3)
For execution on a local, multicore CPU with excess RAM we recommend calling
options(mc.cores = parallel::detectCores()).
To avoid recompilation of unchanged Stan programs, we recommend calling
rstan_options(auto_write = TRUE)
>
```

Now we run `rstan` in *this interactive R session*. Can we persist it?

We are in a docker container. Let's ask `docker ps`:

```
edd@rob:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS          NAMES
b236f06518b5   r-base   "R"                      29 minutes ago  Up 29 minutes             loving_neumann
edd@rob:~$
edd@rob:~$ docker commit --author "<dirk@eddelbuettel.com>" --message "rstan demo container" \
    b236f06518b5 local-rstan    ## container id here key, refers back to the running container
sha256:d72f105b396ff99400618b2d527332af2ab5fa4b45ce88ea7aaa7a5e813a9c87
edd@rob:~$
edd@rob:~$ docker images | grep stan
local-rstan    latest                d72f105b396f         19 seconds ago     1.23GB
edd@rob:~$
```

So `docker commit` can create a new container image under a new name – perfect for interactively modifying containers.

NB: Some whitespace removed, and lines reindented for display

```
edd@rob:~$ docker run --rm -ti local-rstan
```

```
R version 3.5.3 (2019-03-11) -- "Great Truth"  
Copyright (C) 2019 The R Foundation for Statistical Computing  
Platform: x86_64-pc-linux-gnu (64-bit)
```

```
[...]
```

```
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.
```

```
> library(rstan)  
Loading required package: ggplot2  
Loading required package: StanHeaders  
rstan (Version 2.18.2, GitRev: 2e1f913d3ca3)  
For execution on a local, multicore CPU with excess RAM we recommend calling  
options(mc.cores = parallel::detectCores()).  
To avoid recompilation of unchanged Stan programs, we recommend calling  
rstan_options(auto_write = TRUE)  
>
```

Run the new one

We containerized an application!

ALTERNATIVE: USE A DOCKERFILE

A 'Dockerfile' is the standard way to build a container

```
## Start from rocker's r-base or official r-base
FROM rocker/r-base:latest

## Handle for maintainer; these days using LABEL is preferred
MAINTAINER "Dirk Eddelbuettel" dirk@eddelbuettel.com

## Install rstan (downloads and builds all dependencies)
RUN Rscript -e 'install.packages("rstan")'

## Make R the default
CMD ["R"]
```

Building it

- Usually in a directory containing a Dockerfile

```
docker build --tag rocker-rstan .
```

- Lots of other options
- Once built we can push to a repository
- Excellent alternative:
 - Dockerfile at GitHub
 - Build setup at cloud.docker.com (or hub.docker.com)
 - Automatic build and provisioning by Docker

Building it from .deb binaries – “Lego” again as we reuse binaries

- A useful (if little known) alternative is to lean on the binaries
- Mentioned in my blogposts from [Dec 2017](#) and [June 2019](#)
- Simpler, faster & more failsafe as binaries and deps *pre-built*

```
## Start from Rocker container based around Rutter PPAs
FROM rocker/r-ubuntu:18.04

## Handle for maintainer; these days using LABEL is preferred
MAINTAINER "Dirk Eddelbuettel" dirk@eddelbuettel.com

## Update and install rstan -- from binary
RUN apt-get update && apt-get install -y --no-install-recommends r-cran-rstan

## Make R the default
CMD ["R"]
```

More to know

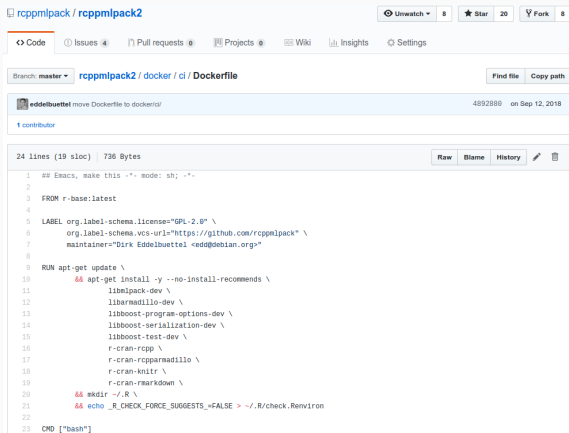
- You can include multiple **RUN** commands:
 - each produces a separate 'layer' cached during build
 - layers are applied consecutively and can be reused
- Other arguments:
 - **COPY** to transfer file from build area into container
 - **ENV** to set environment variables
 - **PORT** to provide network access to a given port
(great for 'backend' services like databases or other servers)
 - ... and much much more
- More details at [Best practices for writing Dockerfile](#)

DOCKER HUB / DOCKER CLOUD

Cloud Support for Building Docker Containers

- Branding is a little inconsistent and flips back and forth
- But in essence another *excellent and free service*:
 - Create a repo on GitHub
 - Create an account Docker Hub / Docker Cloud
 - Define an automated build linking Docker Hub to GitHub
- Then (easiest setting) each commit at GitHub triggers new build
- Very useful (though at times 'laggy' / buys queue)

DOCKER HUB / DOCKER CLOUD: GITHUB REPO



The screenshot shows the GitHub interface for the repository `rcppmlpack/rcppmlpack2`. The file `docker/Dockerfile` is selected, showing a commit by `eddelbuettel` on Sep 12, 2018. The Dockerfile content is as follows:

```
1  ## Emacs, make this -*- mode: sh; -*-
2
3  FROM r-base:latest
4
5  LABEL org.label-schema.license="GPL-2.0" \
6  org.label-schema.vcs-url="https://github.com/rcppmlpack" \
7  maintainer="Dirk Eddelbuettel <edd@debian.org>"
8
9  RUN apt-get update \
10     && apt-get install -y --no-install-recommends \
11         libmlpack-dev \
12         libarmadillo-dev \
13         libboost-program-options-dev \
14         libboost-serialization-dev \
15         libboost-test-dev \
16         r-cran-rcpp \
17         r-cran-rcpparmadillo \
18         r-cran-knitr \
19         r-cran-rmarkdown \
20     && mkdir ~/.R \
21     && echo "_R_CHECK_FORCE_SUGGESTS_FALSE" > ~/.R/check.Renv1ron
22
23  CMD ["bash"]
```

GitHub source repository (for R package requiring external MLPACK machine learning library) with definition for Dockerfile used for testing same repository.

DOCKER HUB / DOCKER CLOUD: DOCKER HUB

The screenshot shows the Docker Hub interface for a repository named 'rcppmpack'. The 'Builds' tab is active, and the 'Build configurations' section is expanded. The 'SOURCE REPOSITORY' is set to 'rcppmpack' and 'rcppmpack2'. The 'BUILD LOCATION' is 'Build on Docker Hub's infrastructure'. The 'AUTOTEST' section has 'OFF' selected. The 'REPOSITORY LINKS' section has 'OFF' selected. The 'BUILD RULES' section shows a table with columns for Source Type, Source, Docker Tag, Dockerfile location, Build Context, Autobuild, and Build Caching. The 'Autobuild' toggle is off, and 'Build Caching' is on. The 'Delete', 'Cancel', 'Save', and 'Save and Build' buttons are visible at the bottom.

Build configurations

SOURCE REPOSITORY: rcppmpack, rcppmpack2

NOTE: Changing source repository may affect existing build rules.

BUILD LOCATION: Build on Docker Hub's infrastructure

AUTOTEST: Off, Internal Pull Requests, Internal and External Pull Requests

REPOSITORY LINKS: Off, Enable for Base Image

BUILD RULES

The build rules below specify how to build your source into Docker images.

Source Type	Source	Docker Tag	Dockerfile location	Build Context	Autobuild	Build Caching
Branch	master	latest	Dockerfile	/docker/GI	<input type="checkbox"/>	<input checked="" type="checkbox"/>

View example build rules

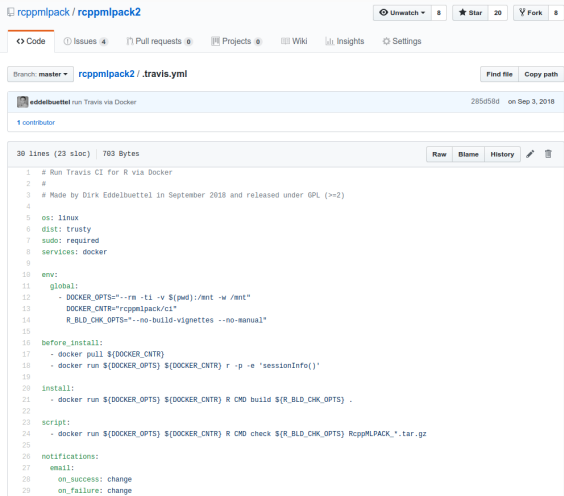
BUILD ENVIRONMENT VARIABLES

Delete Cancel Save Save and Build

Simple Docker Hub setup to tie automatic builds to a GitHub repository.

More complicated dependency-triggered builds also possible.

DOCKER HUB / DOCKER CLOUD: USE IN TRAVIS CI



```
1 # Run Travis CI for R via Docker
2 #
3 # Made by Dirk Eddelbuettel in September 2018 and released under GPL (>=2)
4
5 os: linux
6 dist: trusty
7 sudo: required
8 services: docker
9
10 env:
11   global:
12     - DOCKER_OPTS="--rm -ti -v $(pwd):/mnt -w /mnt"
13     DOCKER_CNTR="rcppmlpack/ci"
14     R_BLD_CHK_OPTS="--no-build-vignettes --no-manual"
15
16 before_install:
17   - docker pull ${DOCKER_CNTR}
18   - docker run ${DOCKER_OPTS} ${DOCKER_CNTR} r -p -e 'sessionInfo()'
19
20 install:
21   - docker run ${DOCKER_OPTS} ${DOCKER_CNTR} R CMD build $(R_BLD_CHK_OPTS) .
22
23 script:
24   - docker run ${DOCKER_OPTS} ${DOCKER_CNTR} R CMD check $(R_BLD_CHK_OPTS) RcppMLPACK_*.tar.gz
25
26 notifications:
27   email:
28     on_success: change
29     on_failure: change
```

In essence, we use the Docker container (defined in this repo but built by the Docker Hub) to run continuous integration via Travis CI.

ROCKER USE CASES

Some Examples

- ‘Difficult’ or ‘Large’ Things: Rocker Project has long maintained large ‘tidyverse’, geospatial, ... containers and more
- ‘Applications’ as for example RStudio Server or Shiny Server
- ‘Frameworks’ adding Machine Learning / Tensorflow containers

Some Examples

- One key part of Rocker are the *versioned containers* using the snapshot 'MRAN' archive provided by Microsoft
- This gives the ability to 'freeze' a container with software at a given release point
- Reproducibility: 'turn research study into container'; and [containerit](#) does that
- Using [mybinder.org](#) is another possibly using [holepunch](#)

OTHER TOPICS

Things we did not cover

- Composition: Orchestrating multiple containers has become a big topic, Kubernetes is a key application here (*c.f.* next talk)
- Docker variants and spin-offs: **containerd** is part of the Docker backend and has been spun off; there is a fair amount going on but Docker has first-mover advantage and mind-share
- Docker for science: a somewhat simpler approach called **singularity** has made inroads
- And much much more...

Some Pointers

- [A Docker 101 course](#)
- [Several usage samples](#)
- [R on Docker tutorial](#) from rOpenSci
- My (possibly dated in parts) [three hour tutorial](#) from useR! 2015
- For Rocker: [Boettiger and Eddelbuettel, 2017, RJournal](#)

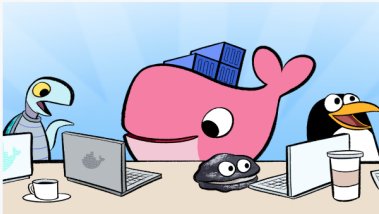
12
DEC

Thursday, December 12, 2019

Hands-on Workshop: Introduction to Docker for Developers



Hosted by
Moby Dock



Details

<https://events.docker.com/events/details/docker-chicago-presents-hands-on-workshop-introduction-to-docker-for-developers/>

Join us for the Chicago local edition of this season's global hands-on workshop series! Food and drinks will be provided! Bring your laptop and #LearnDocker. There will be swag!

Docker has (or had) some meetups in Chicago; this is AFAIK the first in a while.

Give it a go!

THANK YOU!

slides <http://dirk.eddelbuettel.com/presentations/>

web <http://dirk.eddelbuettel.com/>

mail dirk@eddelbuettel.com

github [@eddelbuettel](https://github.com/eddelbuettel)

twitter [@eddelbuettel](https://twitter.com/eddelbuettel)