



# RELIABLE REPRODUCIBLE RESEARCH VIA CONTAINERS FROM THE ROCKER PROJECT

Dirk Eddebuettel

Data Science, Statistics & Visualization 2020

29 July 2020

# OUTLINE

---

- Brief Bio
- What is Docker? How can I use it?
- What does Rocker do? What does it offer?
- Some general thoughts on Reproducibility

## Now

- At [TileDB](#) creating *Universal Data Engine* (esp. the R parts)
- Adj. Clinical Professor at Illinois: [Data Science Progr. Methods](#)

## Previously

- 20+ years in quantitative research, development, trading

## But Also

- Rather involved with Open Source for 25+ years
- *i.e.* as a Debian Developer building a Linux distribution
- And R packages, Rocker, R Foundation Board, JSS, ...

## Key Points of Debian Experience

- Centered around package management that *simply works*
- No dependency hell whatsoever
  - components can be added
  - or removed fluidly
- Constructing “state”: reliable, repeatable, reproducible
- Interesting other sub-parts: reproducible builds
- This has spread to many other distributions

## My Journey

- The Debian experience of (then nearly 20 years) left a mark
- We had something that was technically excellent
- ... yet not widely used (on desktops/laptops)
- Virtual machines were an alternative, but clunky
- Enters Docker (in late 2013 / early 2014)

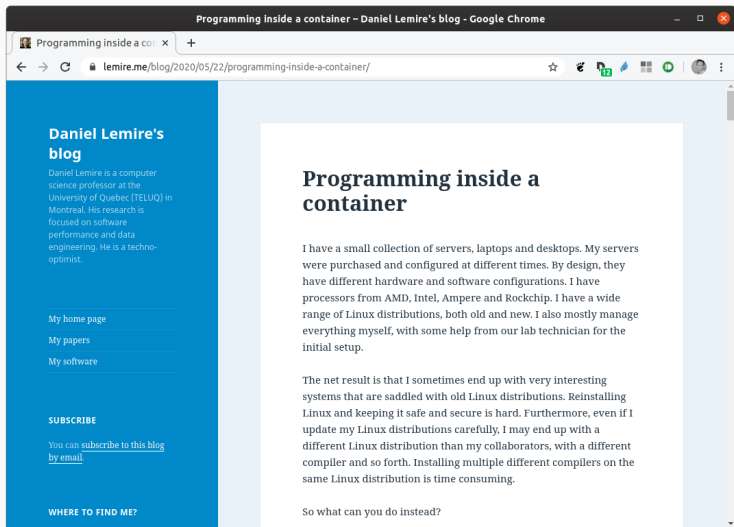
## My useR! 2014 keynote

- Carved out 10 at the end of a talk (mostly on Rcpp)
- Predicted "will change how we build / test / deploy"
- At that prediction held up well as these decays
  - **Continuous Integration / Cloud Deployments** pervasive
  - Just about every service is (primarily) **Linux based**
- So we
  - still "lost" the desktop
  - but "won" compute infrastructure ("cloud")
  - and this has reproducibility implications

## SO WHAT ARE DOCKER AND ROCKER?

---





The screenshot shows a Google Chrome browser window with the address bar displaying `lemire.me/blog/2020/05/22/programming-inside-a-container/`. The page has a blue sidebar on the left and a main content area on the right. The sidebar contains the author's name, a bio, navigation links, a subscribe button, and a 'WHERE TO FIND ME?' section. The main content area features the article title, a paragraph of text, and a sub-heading.

**Programming inside a container – Daniel Lemire's blog - Google Chrome**

Programming inside a container | +

← → ↻ 🔒 lemire.me/blog/2020/05/22/programming-inside-a-container/ ☆ 🗄️ 📄 📧 📺 📱 ⋮

**Daniel Lemire's blog**

Daniel Lemire is a computer science professor at the University of Quebec (TELUQ) in Montreal. His research is focused on software performance and data engineering. He is a techno-optimist.

My home page

My papers

My software

**SUBSCRIBE**

You can [subscribe to this blog by email](#).

**WHERE TO FIND ME?**

## Programming inside a container

I have a small collection of servers, laptops and desktops. My servers were purchased and configured at different times. By design, they have different hardware and software configurations. I have processors from AMD, Intel, Ampere and Rockchip. I have a wide range of Linux distributions, both old and new. I also mostly manage everything myself, with some help from our lab technician for the initial setup.

The net result is that I sometimes end up with very interesting systems that are saddled with old Linux distributions. Reinstalling Linux and keeping it safe and secure is hard. Furthermore, even if I update my Linux distributions carefully, I may end up with a different Linux distribution than my collaborators, with a different compiler and so forth. Installing multiple different compilers on the same Linux distribution is time consuming.

So what can you do instead?

Source: <https://lemire.me/blog/2020/05/22/programming-inside-a-container/>

## Excellent discussion by Daniel Lemire

*The idea of a container approach is to **always start from a pristine state**. So you define the configuration that your database server needs to have, and you launch it, **in this precise state each time**. This makes your infrastructure predictable.*

We can of course substitute “predictable” with “reproducible” ...

## ON IN SIMPLER TERMS:

## ON IN SIMPLER TERMS:



Source: [https://commons.wikimedia.org/wiki/File:3\\_D-Box.jpg](https://commons.wikimedia.org/wiki/File:3_D-Box.jpg)

### Docker is “a box”

- Standardized shape and form, identical for everyone
- “Take it anywhere, useable by everyone”
- Genius of “if someone can **run a box** they can **run your box**”
- Examples of running
  - RStudio Server on OS that RStudio does not provide it for (!!)
  - Shiny Server on an OS that RStudio does not provide it for (!!)
  - *standardized setups* for colleagues, PIs, students, ...

# DOCKER

---

# DOCKER: SO WHAT IS IT?

# DOCKER: 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



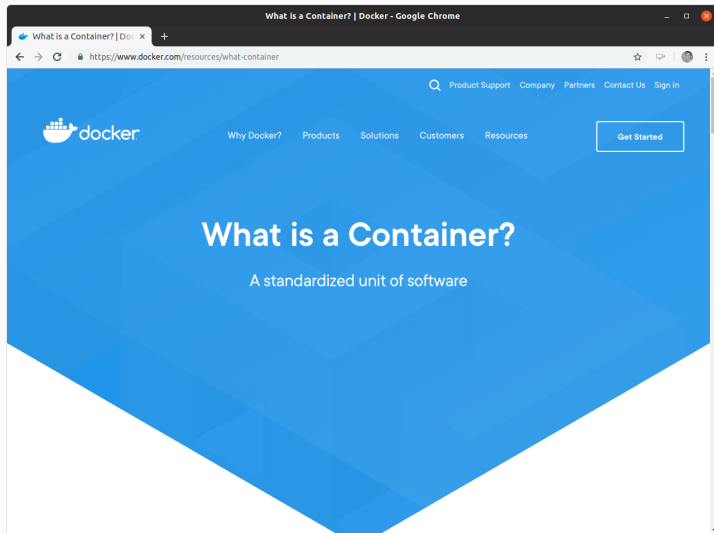
# DOCKER: SO WHAT IS IT?

- 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**:
  - it requires only a *very thin* layer above the operating system
  - macOS & Windows need intermediating layer via VM
  - hence very heavy Linux usage in cloud deployments

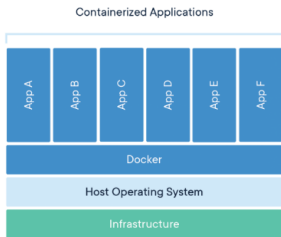
# DOCKER: SO WHAT IS IT?

- 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**:
  - it requires only a *very thin* layer above the operating system
  - macOS & Windows need intermediating layer via VM
  - hence very heavy Linux usage in cloud deployments
- 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>

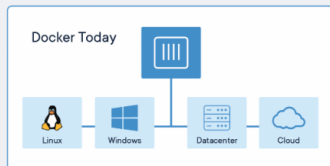
## Docker Containers Are Everywhere: Linux, Windows, Data center, Cloud, Serverless, etc.

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.

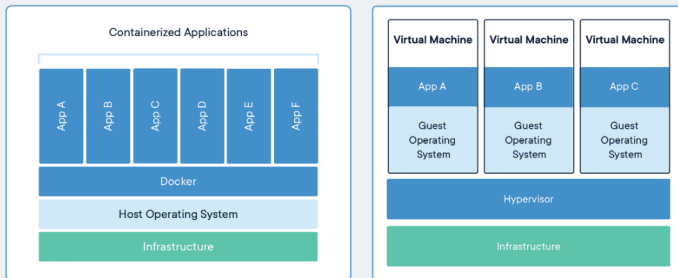


Source: <https://www.docker.com/resources/what-container>

## 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**
  - not a virtual machine which is more like a whole computer
- So it helps to **think of Docker encapsulating a single command**
  - though that first 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 each for database, 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 sloppy and use container and image interchangeably
- On the other hand, a *virtual machine* tends to be
  - a heavier software layer providing a *full* virtual system
  - VMware and VirtualBox are two well-known systems.



## 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` provides container port 8787 as host port 8787
  - `container/tag:version`
  - `cmdline arguments` for container application
  - *plus many more options* so see documentation
  - often use a shell alias `dkrr` to ‘fix’ some of these
- 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 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 CASES AND 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
- more advanced use use of different R builds is also possible

Use multiple R versions (and an alias `dkrr`)

```
$ dkrr r-base:latest R --version | head -1
R version 4.0.2 (2020-06-22) -- "Taking Off Again"
$ dkrr r-base:3.6.3 R --version | head -1
R version 3.6.3 (2020-02-29) -- "Holding the Windsock"
$ dkrr r-base:3.5.3 R --version | head -1
R version 3.5.3 (2019-03-11) -- "Great Truth"
$ dkrr r-base:3.4.2 R --version | head -1
R version 3.4.2 (2017-09-28) -- "Short Summer"
$
```

which generalizes to the the triplet:

`dockerCommand dockerContainer args`



## Test against development versions

- Sometimes we want to test against new development versions
- These versions may still be unfinished and undergo changes
- Containers provide ideal use via a ‘sandbox’

```
edd@rob:~$ docker run --rm -ti rocker/drd:latest RD --version | head -4
R Under development (unstable) (2020-07-18 r78872) -- "Unsuferred Consequences"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

edd@rob:~$
```

(This shows the July 18 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](https://hub.docker.com).)

## A worked example

- Installing a complex package, say, `rstan` can be challenging
- Proving it in a container is a good to offer it
- We show several ways and illustrate Docker use along the way

# DETAILED EXAMPLE: RSTAN INTERACTIVELY

---

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

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

```
R version 4.0.2 (2020-06-22) -- "Taking Off Again"  
Copyright (C) 2020 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")
install.packages("rstan")
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)
also installing the dependencies 'rstudioapi', 'evaluate', 'pkgload', 'praise', 'colorspace',
'utf8', 'ps', 'testthat', 'farver', 'labeling', 'lifecycle', 'munsell', 'RColorBrewer',
'viridisLite', 'ellipsis', 'fansi', 'magrittr', 'pillar', 'pkgconfig', 'vctrs', 'backports',
'processx', 'assertthat', 'digest', 'glue', 'gtable', 'isoband', 'rlang', 'scales', 'tibble',
'checkmate', 'matrixStats', 'callr', 'cli', 'crayon', 'desc', 'prettyunits', 'R6', 'rprojroot',
'jsonlite', 'curl', 'StanHeaders', 'ggplot2', 'inline', 'gridExtra', 'Rcpp', 'RcppParallel',
'loo', 'pkgbuild', 'withr', 'V8', 'RcppEigen', 'BH'

trying URL 'https://cloud.r-project.org/src/contrib/rstudioapi_0.11.tar.gz'
Content type 'application/x-gzip' length 98082 bytes (95 KB)
=====
downloaded 95 KB

trying URL 'https://cloud.r-project.org/src/contrib/evaluate_0.14.tar.gz'
Content type 'application/x-gzip' length 24206 bytes (23 KB)
=====
downloaded 23 KB

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

## We ask R to install `rstan` (continued)

```
[... quite a bit of compilation, and build help to curl and v8, later ...]
```

```
ar -rs ../inst/lib//libStanServices.a stan_fit.o stan_fit_base.o
ar: creating ../inst/lib//libStanServices.a
installing to /usr/local/lib/R/site-library/00LOCK-rstan/00new/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 from temporary location
** checking absolute paths in shared objects and dynamic libraries
** testing if installed package can be loaded from final location
** testing if installed package keeps a record of temporary installation path
* DONE (rstan)
```

```
The downloaded source packages are in
  '/tmp/Rtmp1NGFGf/downloaded_packages'
>
```

We ask R to install `rstan` (continued)

```
> library(rstan)
library(rstan)
Loading required package: StanHeaders
Loading required package: ggplot2
rstan (Version 2.21.2, 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"                      19 minutes ago  Up 19 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  1.58GB
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 4.0.2 (2020-06-22) -- "Taking Off Again"  
Copyright (C) 2020 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: StanHeaders  
Loading required package: ggplot2  
rstan (Version 2.21.2, 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](https://cloud.docker.com) (or [hub.docker.com](https://hub.docker.com))
  - Automatic build and provisioning by Docker

**ALTERNATIVE:**

**USE A DOCKERFILE WITH BINARIES**

---

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

- A useful (if little known) alternative is to lean on the binaries
- C.f. my blog (and videos) [Dec 2017](#), [June 2019](#) and [June 2020](#)
- 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

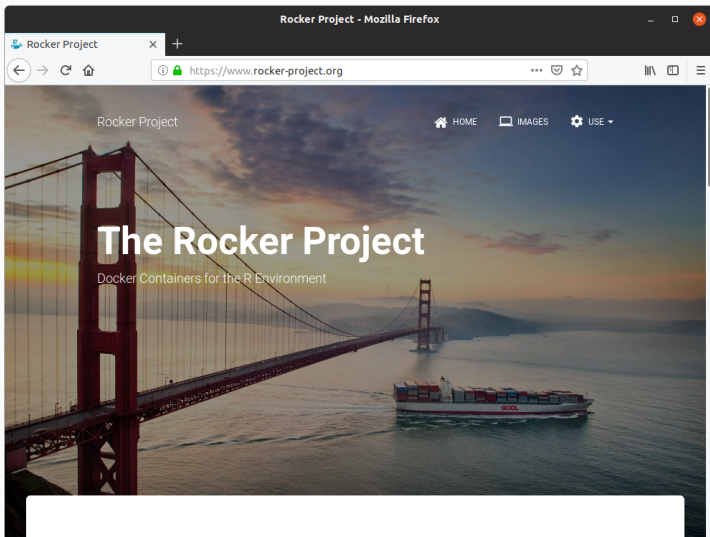
- You can include multiple **RUN** commands:
  - each produces a separate 'layer' cached during build
  - layers are applied consecutively and can be reused
- Other arguments:
  - **COPY** to transfer file from build area into container
  - **ENV** to set environment variables
  - **PORT** to provide 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](#)

# ROCKER

---



# ROCKER PROJECT



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

# EARLIER ROCKER FOR REPRODUCIBILITY PAPER

Makes (early, Jan 2015)  
case for Docker

The screenshot shows a web browser displaying the ACM Digital Library page for the article "An introduction to Docker for reproducible research". The browser's address bar shows the URL: [dl.acm.org.proxy2.library.jillinois.edu/doi/abs/10.1145/2723872.2723882](https://dl.acm.org.proxy2.library.jillinois.edu/doi/abs/10.1145/2723872.2723882). The page header includes the ACM Digital Library logo and navigation links for Journals, Magazines, Proceedings, Books, SIGs, Conferences, and People. The article title is "An introduction to Docker for reproducible research" by Carl Reintjes. The publication information is "ACM SIGOPS Operating Systems Review • January 2015". The article has 273 citations and 3,904 views. The abstract text is visible, starting with "As computational work becomes more and more integral to many aspects of scientific research, computational reproducibility has become an issue of increasing importance to computer systems researchers and domain scientists alike. Though computational reproducibility seems more straight forward than replicating physical experiments, the complex and rapidly changing nature of computer environments makes being able to reproduce and extend such work a serious challenge. In this paper, I explore common reasons that code developed for one research project cannot be successfully executed or extended by subsequent researchers. I review current approaches to these issues, including virtual machines and workflow systems, and their limitations. I then examine how the popular emerging technology Docker combines several areas from systems research - such as operating system virtualization, cross-platform portability, modular re-usable elements, versioning, and a 'DevOps' philosophy, to address these challenges. I illustrate this with several examples of Docker use with a focus on the R statistical environment."

Source: <https://dl.acm.org/doi/10.1145/2723872.2723882>

## An introduction to Docker for reproducible research

Carl Boettiger

Center for Stock Assessment Research,  
110 Shaffer Rd, Santa Cruz, CA 95050, USA  
cboettig(at)gmail.com

### ABSTRACT

As computational work becomes more and more integral to many aspects of scientific research, computational reproducibility has become an issue of increasing importance to computer systems researchers and domain scientists alike. Though computational reproducibility seems more straightforward than replicating physical experiments, the complex and rapidly changing nature of computer environments makes being able to reproduce and extend such work a serious challenge. In this paper, I explore common reasons that code developed for one research project cannot be successfully executed or extended by subsequent researchers. I review current approaches to these issues, including virtual machines and workflow systems, and their limitations. I then examine how the popular emerging technology Docker combines several areas from systems research - such as operating system virtualization, cross-platform portability, modular re-usable elements, versioning, and a 'DevOps' philosophy, to address these challenges. I illustrate this with several examples of Docker use with a focus on the R statistical environment.

### *Systems research & reproducibility*


Systems research has long concerned itself with the issues of computational reproducibility and the technologies that can facilitate those objectives [6]. Docker is a new but already very popular open source tool that combines many of these approaches in a user friendly implementation, including: (1) performing Linux container (LXC) based operating system (OS) level virtualization, (2) portable deployment of containers across platforms, (3) component reuse, (4) sharing, (5) archiving, and (6) versioning of container images. While Docker's market success has largely focused on the needs of businesses in deploying web applications and the potential for a lightweight alternative to full virtualization, these features have potentially important implications for systems research in the area of scientific reproducibility.

In this paper, I seek to address two audiences. First, that of the domain scientist, those conducting research in ecology, bioinformatics, economics, psychology and so many other disciplines in which computation plays an ever-increasing role. I seek to help this audience become more aware of the concerns

Source: <https://dl.acm.org/doi/10.1145/2723872.2723882>

An Introduction to Rocker... The R Journal - Mozilla Firefox


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




**Navigation**

- Current Issue
- Accepted articles
- Archive
- R News
- News and Notes
- Submissions
- Reviews and Proofreading
- Editorial Board

**Subscribe**

- RSS Feed 
- ISSN: 2073-4859


## The R Journal: article published in 2017, volume 9:2

An Introduction to Rocker: Docker Containers for R 

Carl Boettiger and Dirk Eddelbuettel, *The R Journal* (2017) 9:2, pages 527-536.

**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.

Received: 2017-10-12; online 2017-11-27  
 CRAN packages: [packrat](#), [rhub](#), [tidyverse](#)  
 CRAN Task Views Implied by cited CRAN packages: [ReproducibleResearch](#)

 This article is licensed under a [Creative Commons Attribution 4.0 International license](#).

```
@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>

# UPCOMING ROCKERVERSE PAPER

The screenshot shows a web browser displaying the arXiv page for the paper "The Rockerverse: Packages and Applications for Containerization with R". The page header includes the Cornell University logo and a search bar. The main content area features the title, authors (Daniel Nüst, Dirk Eddelbuettel, Dom Bennett, Robrecht Cannoodt, Dav Clark, Gergely Daroczi, Mark Edmondson, Colin Fay, Ellis Hughes, Lars Kjeldgaard, Sean Lopp, Ben Marwick, Heather Nolis, Jacqueline Nolis, Hong Ooi, Karthik Ram, Noam Ross, Lori Shepherd, Péter Sólymos, Tyson Lee Swetnam, Nitesh Turaga, Charlotte Van Pelegem, Jason Williams, Craig Willis, Nan Xiao), and a short abstract. The abstract text reads: "The Rocker Project provides widely used Docker images for R across different application scenarios. This article surveys downstream projects that build upon the Rocker Project images and presents the current state of R packages for managing Docker images and controlling containers. These use cases cover diverse topics such as package development, reproducible research, collaborative work, cloud-based data processing, and production deployment of services. The variety of applications demonstrates the power of the Rocker Project specifically and containerisation in general. Across the diverse ways to use containers, we identified common themes: reproducible environments, scalability and efficiency, and portability across clouds. We conclude that the current growth and diversification of use cases is likely to continue its positive impact, but see the need for consolidating the Rockerverse ecosystem of packages, developing common practices for applications, and exploring alternative containerisation software." The page also includes a "Download:" section with links for PDF and other formats, a "References & Citations" section with links to NASA ADS, Google Scholar, and Semantic Scholar, and a "Bookmark" section. The footer of the page contains the source URL: <https://arxiv.org/abs/2001.10641>.

Source: <https://arxiv.org/abs/2001.10641>

Very recent and wide-ranging survey of Rocker container use

Several aspect of reproducibility with Docker and Rocker covered

Forthcoming in the *R Journal*

## The Rockerverse: Packages and Applications for Containerisation with R

*by Daniel Nüst, Dirk Eddelbuettel, Dom Bennett, Robrecht Cannoodt, Dav Clark, Gergely Daróczi, Mark Edmondson, Colin Fay, Ellis Hughes, Lars Kjeldgaard, Sean Lopp, Ben Marwick, Heather Nolis, Jacqueline Nolis, Hong Ooi, Karthik Ram, Noam Ross, Lori Shepherd, Péter Sólymos, Tyson Lee Swetnam, Nitesh Turaga, Charlotte Van Petegem, Jason Williams, Craig Willis, Nan Xiao*

**Abstract** The Rocker Project provides widely used Docker images for R across different application scenarios. This article surveys downstream projects that build upon the Rocker Project images and presents the current state of R packages for managing Docker images and controlling containers. These use cases cover diverse topics such as package development, reproducible research, collaborative work, cloud-based data processing, and production deployment of services. The variety of applications demonstrates the power of the Rocker Project specifically and containerisation in general. Across the diverse ways to use containers, we identified common themes: reproducible environments, scalability and efficiency, and portability across clouds. We conclude that the current growth and diversification of use cases is likely to continue its positive impact, but see the need for consolidating the Rockerverse ecosystem of packages, developing common practices for applications, and exploring alternative containerisation software.

Source: <https://arxiv.org/abs/2001.10641>

## 👥 Team



The Rocker project was created by **Carl Boettiger** and **Dirk Eddelbuettel**, and is now maintained by Carl, Dirk, and **Noam Ross**, with significant contributions from a broad community of users and developers. Get in touch on **GitHub issues** with bug reports, feature requests, or other feedback.

We gratefully acknowledge funding from CZI to allow continued development of the Rocker Project.

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



## Two Key Sets of Containers

- The Base Containers
  - Key **base layer**: our `rocker/r-base` is the official `r-base`
  - Containers `r-devel`, `r-devel-san`, `r-rspm`, ... built off these
- Versioned Stack
  - **Difficult / large** containers: `tidyverse`, `geospatial`, ...
  - **Applications** as for example RStudio Server or Shiny Server
  - **Frameworks** for Machine Learning / Tensorflow are added
  - A lot of this is in the (new, rewritten) 'versioned2' stack

## Some Examples

- Rocker *versioned containers* used the snapshot 'MRAN' archive provided by Microsoft, now similar via RSPM
- Can 'freeze' container with software at given release point

## Alternative / Derivations

- Reproducibility: 'turn research study into container'
- For example [containerit](#) does just that
- Using [mybinder.org](#) is another possibly using [holepunch](#)

# REPRODUCIBILITY

---

## Where we are today

- The world has changed somewhat
- When I was a grad student *data repositories* were starting
- Later Journals began to experiment with code repositories
- ... but that was “here, have a Fortran, or Stata, or ...” file
- Now many formal reproducibility efforts underway
- That is undeniably good progress

## How did we get here

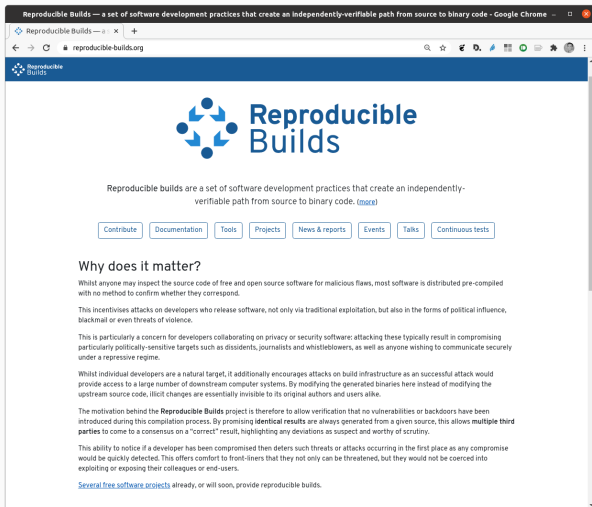
- Open Source “winning” helped
- We have high-quality research software in multiple languages
- (But still too many ‘red vs blue hammer’ discussions)
- *Fundamentally* programming languages choice does irrelevant
- In practice it does of course matter
  - as we sink human capital into knowledge
  - and fields and disciplines focus on particular “stacks”
  - leading to different resource for different “stacks”

## What may be a description of the status quo

- “Freezing” a local installation of a software stack common
- Python `virtualenv`, R `renv` (and `packrat`), Node/JS too ...
- There are likely many others I don’t know about
- Docker makes it easy to operationalize this
- Plus Docker use offers somewhat “more”
  - as it gets closer to behaving like a whole machine
  - without requiring a whole machine

## Personal Views

- Freezing a software stack in a directory tree is a band aid
- It may stop the bleeding for a bit
- It is not a fundamental solution
- *Deep down* this is an engineering problem
  - That *could* get fixed with better practice
  - I just don't know if we can get there
- Other aspects (hardware, kernel, ...) affect reproducibility
- Still ... while we made good progress there is lots more to do



Originally started  
with Debian

Now several key  
distributions  
involved

Extends the set of  
reproducibly  
made  
components



## MORE INFO

---

## Some Pointers

- A [Docker 101 course](#)
- Several [usage samples](#)
- [R on Docker tutorial](#) from rOpenSci
- My (maybe dated in parts) [three hour tutorial](#) from useR! 2015
- For Rocker: [Boettiger and Eddelbuettel, 2017, RJournal](#)
- Rockerverse: [Nuest, Eddelbuettel et al, 2020, arXiv, accepted RJournal](#)

## New zine: How Containers Work!

On Friday I published a new zine: "How Containers Work!". I also launched a fun redesign of [wizardzines.com](https://wizardzines.com).

You can get it for \$12 at <https://wizardzines.com/zines/containers>. If you buy it, you'll get a PDF that you can either print out or read on your computer. Or you can get a pack of all 8 zines so far.

Here's the cover and table of contents:



### table of contents

why containers?.....	4	cgroups.....	13
the big idea: include		namespaces.....	14
EVERY dependency.....	5	how to make a namespace.....	15
containers aren't magic.....	6	PID namespaces.....	16
containers = processes.....	7	user namespaces.....	17
container kernel features.....	8	network namespaces.....	18
pivot_root.....	9	container IP addresses.....	19
layers.....	10	capabilities.....	20
overlay filesystems.....	11	seccomp/BPF.....	21
container registries.....	12	configuration options.....	22

Zine by Julia Evans

\$12 likely well-spent

have not seen zine

but have enjoyed several posts

See <https://jvns.ca/blog/2020/04/27/new-zine-how-containers-work/>

## 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...

# THANK YOU!

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

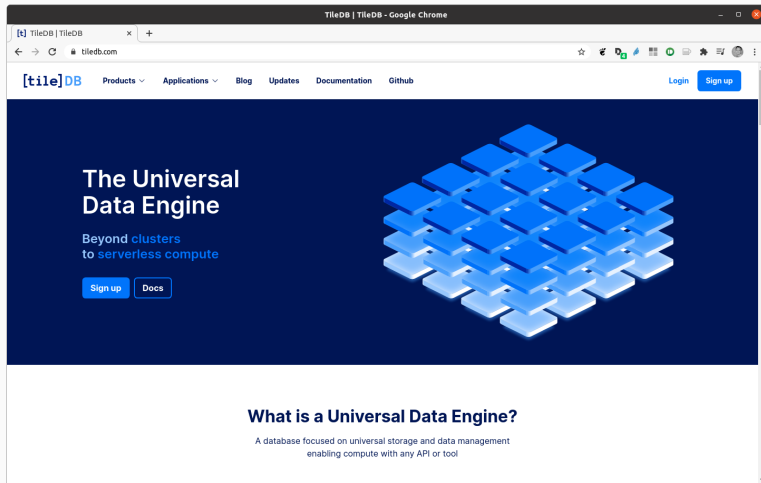
web <http://dirk.eddelbuettel.com/>

mail [dirk@eddelbuettel.com](mailto:dirk@eddelbuettel.com)

github [@eddelbuettel](#)

twitter [@eddelbuettel](#)

# TILEDB IS HIRING: APPLY AT TILEDB.WORKABLE.COM !



The screenshot shows the TileDB website homepage. The browser window title is "TileDB | TileDB - Google Chrome". The address bar shows "tiledb.com". The navigation menu includes "Products", "Applications", "Blog", "Updates", "Documentation", and "Github". There are "Login" and "Sign up" buttons in the top right. The main content area features the headline "The Universal Data Engine" with the subtext "Beyond clusters to serverless compute". Below this are "Sign up" and "Docs" buttons. To the right is a 3D isometric graphic of blue rectangular blocks arranged in a grid. Below the main content is a section titled "What is a Universal Data Engine?" with a brief description: "A database focused on universal storage and data management enabling compute with any API or tool".

TileDB | TileDB - Google Chrome

tiledb.com

[tile]DB Products Applications Blog Updates Documentation Github Login Sign up

## The Universal Data Engine

Beyond clusters to serverless compute

Sign up Docs

### What is a Universal Data Engine?

A database focused on universal storage and data management enabling compute with any API or tool