



USING R VIA ROCKER

A BRIEF INTRODUCTION WITH EXAMPLES

Dirk Eddelbuettel

Köln R User Group

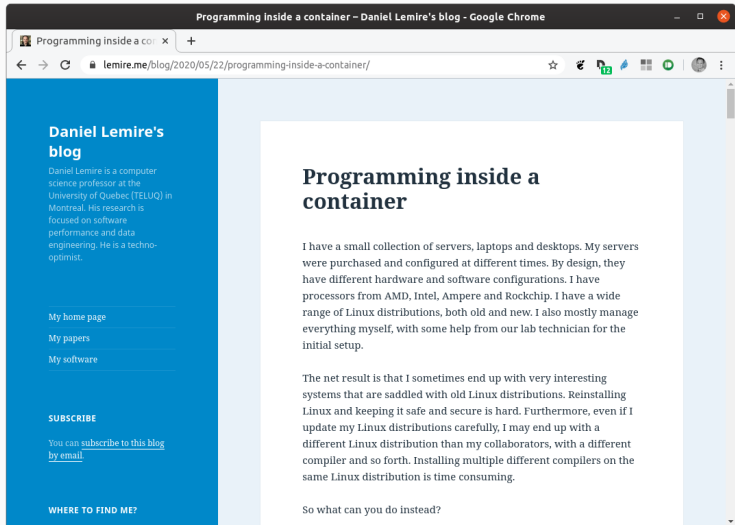
19 Nov 2020

Brief Outline

- Docker Intro: Appeal, Motivation, “Lego” blocks, ...
- Brief Overview of Docker and its commands
- 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
- [Rockerverse paper](#) (lead by Daniel Nuest) in R Journal, 2020



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

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

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 or running
 - RStudio Server on OS that RStudio does not provide it for (!!)
 - Shiny Server on an OS that RStudio does not provide it for (!!)
 - other *standardized setups* your colleagues, PIs, students, ... cannot setup or maintain

NOW WHY THE FIDDLE DO I CARE?

NOW WHY THE FIDDLE DO I CARE?



Grad School Social Distancer @darinself · 4h



So before I update to R 4.0 how has this worked for everyone?

[#RStats](#)



NOW WHY THE FIDDLE DO I CARE?



Perfect for a pristine box:

- get R 4.0.0 “in a box”
- put your code in
- check!

LIVE DEMO!

The BSPM (“Bridge to System Package Manager”) Package

- `bspm` is a pretty new R package by Iñaki Ucar
- There are a few posts at [my blog](#)
- Today really ‘high level’:
 - Assume a Linux system as
 - *e.g.* cloud server, continuous integration server, ...
 - Assume you know little Unix system admin details
- **Claim: Using ‘bspm’ on a suitable system makes life peachy**

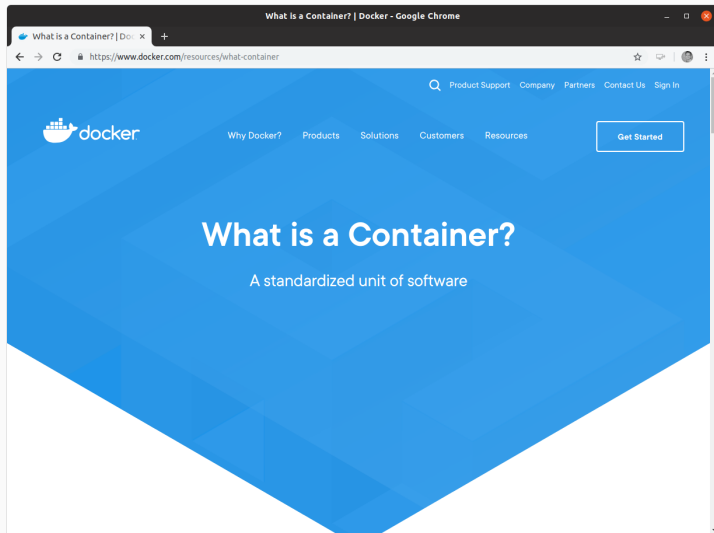
Here ‘suitable system’ means one that has binaries with system dependencies such as the one Iñaki maintains for Fedora with ~ 15k CRAN binaries, or the ‘Rutter PPA’ on Launchpad. You can mostly ignore the details encoded in containers `rocker/r-bspm` with tags for Ubuntu 18.04, 20.04 and Debian testing. We would need another talk for fuller details. Just enjoy the demo.

DOCKER

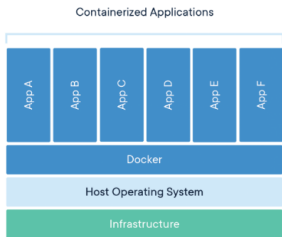
DOCKER: SO WHAT IS IT?

- 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
 - on macOS + Windows intermediating layer has to be provided
 - 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>

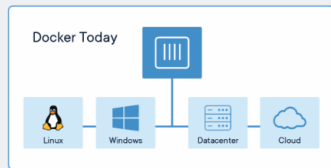
Docker Containers Are Everywhere: Linux, Windows, Data center, Cloud, Serverless, etc.

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.

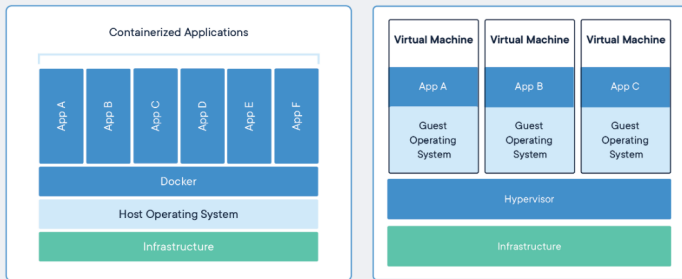


Source: <https://www.docker.com/resources/what-container>

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 (*i.e.* not a whole computer system)
- 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:
 - one container for database
 - one container for webserver
 - ...
 - all 'orchestrated' (but that is beyond our file)

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` sets container port 8787 as host port 8787
 - `container/tag:version` selects container
 - `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 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
- more advanced use use of different R builds is also possible

Use multiple R versions

```
$ docker run --rm -ti r-base:latest R --version | head -1
R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
$ docker run --rm -ti r-base:3.6.3 R --version | head -1
R version 3.6.3 (2020-02-29) -- "Holding the Windsock"
$ 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"
$
```

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

```
$ dkrr r-base:latest R --version | head -1
R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
$ 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) (2020-11-14 r79432) -- "Unsufered Consequences"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

edd@rob:~$
```

(This shows the Nov 14 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.)

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 with and without GPU support

Some Examples

- Key part of Roker are *versioned containers*
 - use dated snapshots of CRAN archives
 - gives ability to ‘freeze’ container at a given release point

Allows for add-ons

- Reproducibility: ‘turn research study into container’
 - e.g. [containerit](#) does that
 - or [mybinder.org](#) is another possibility using [holepunch](#)

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...

BACK TO OUR DEMO

Recipe

- Start from system with R and suitable setup (*i.e.* PPA)
- This is *fully* transparent in [corresponding Dockerfile sources](#)
- We run two system commands merely to refresh the system:
`apt update -qq && apt upgrade -y`
- Then *one R command*: `install.packages("tidyverse")`

Recipe (cont.)

- It installed 100 binaries fully resolving all dependencies
- Best of all we could then ‘save’ the modified Docker container:
 - use second shell, run `docker ps` to see the ‘CONTAINER ID’
 - use `docker commit CONTAINER_ID local-name:tag`
 - and subsequently run `docker run ... local-name:tag`
- Can now start modified container with our package set
- We built ourselves **a new box!**

Dockerfile Alternative

```
# Minimal but valid Dockerfile for talk

# Container we start from, as in interactive use
FROM rocker/r-bspm:20.04

# Optional LABEL for author etc, skipped here

# Run our three commands (most efficiently as one)
RUN apt update -qq && \
    apt upgrade -y && \
    Rscript -e 'install.packages("tidyverse")'

# Default to starting R
CMD ["R"]
```

and use it via `docker build -t local-name:tag .`

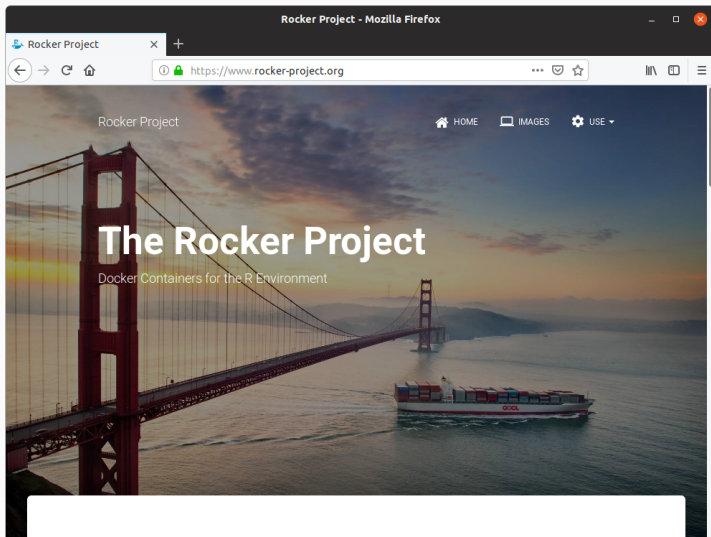
So ...

THAT BOX ANALOGY



Source: https://commons.wikimedia.org/wiki/File:Pralinenteller_-_box_of_chocolates.jpg

ROCKER PROJECT



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

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.

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 R Project provides widely used Docker images for R across different application scenarios. This article surveys downstream projects that build upon the R 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 R 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.

Introduction

The R community continues to grow. This can be seen in the number of new packages on CRAN, which is still on growing exponentially (Hornik et al., 2019), but also in the numbers of conferences, open educational resources, meetups, unconferences, and companies that are adopting R, as exemplified by the useR! conference series¹, the global growth of the R and R-Ladies user groups², or the foundation and impact of the R Consortium³. These trends cement the role of R as the *lingua franca* of statistics, data

Source: <https://journal.r-project.org/archive/2020/RJ-2020-007/>

👥 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.

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

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](#) and 'Rockerverse': [Nuest, Eddelbuettel, et al, 2020, 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.

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/>

THANK YOU!

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

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

mail dirk@eddelbuettel.com

github [@eddelbuettel](#)

twitter [@eddelbuettel](#)