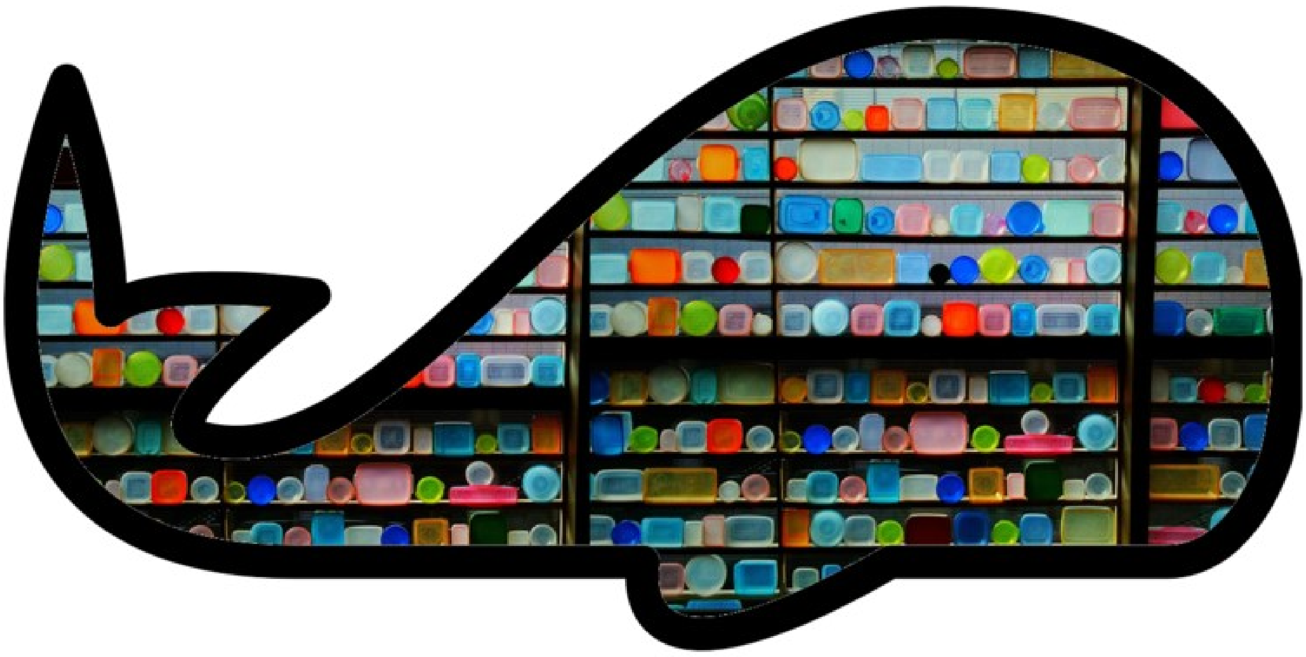
An R-docker hello world example

Elizabeth Stark Jan 4, 2018 · 6 min read



Yes, I've turned into one of those "we could containerise this!" people.

I promise I'm not just being a trend follower — container technology has amazing promise for our team who build custom analytics apps for ourselves and clients.

We use the R statistical programming language/environment for building everything from machine learning solutions to reporting dashboards. R is a great environment for analytics and rapid prototyping but it was traditionally a statistical scripting environment and it lacks some of the "production-ready" features of more traditional programming languages.

> *Docker gives us a way to manage dependencies and libraries on a per-app basis (when deploying multiple apps on a server).*
>
> *It also means that the thing that works on my machine has a good chance of working in production. And that, right there, saves many late night panics.*

About this tutorial

This article is a slightly updated version of a blog I wrote for our company blog last year. It will run you through the steps to build, run and connect a docker container that runs R. We will

- Set up a Dockerfile
- Build a container
- Run a container to serve an Rstudio environment
- Connect to that container and run a script
- Write the output back to a mounted data folder on the host
- Congratulate ourselves with a nice cup of tea (just kidding — you can probably figure that bit out for yourselves)

To play along at home, clone or download the directory from our github repository. The code has been tested on Linux and Mac — you may need minor changes on Windows (please feel free to add any extra tips in the comments).

About Docker

Docker is a management system/environment for using containers. **Containers** are built on top of **hosts**. They share the same kernel and hardware controllers but might have a different Linux flavour or set of libraries on top.

We set up container **images** that are like snapshots of the container we want — all the libraries, files etc. We then **run** the container to set up a temporary instance that contains all our working files.

> *When we are done we **stop** the container and **all data and any local changes are lost forever.***

To save the output of a container instance we must write the data back to the host or somewhere else that's permanent.

For this example I am assuming that you are working on a machine that already has docker installed.

This example

Clone or download the repository from https://github.com/SymbolixAU

/r_docker_hello.

Navigate to your new directory `R_docker_hello` using whichever command line terminal you use.

Inside this folder you will find:

- `Dockerfile`: This lives in the top directory and specifies our build options.
- Analysis folder: Holds a simple `hello_world.R` script that we will run in our container
- Data folder: The simplest input data you will ever see. We will also mount this folder when we run the container and we will write our output back to it.
- DockerConfig: Everyone likes particular libraries, so I've made a `requirements.R` that will be run whenever you build your container.

Ready? Let's do this…

Inside the Dockerfile

The Dockerfile contains the following content:

```
# Base image https://hub.docker.com/u/rocker/
FROM rocker/rstudio
```

```
## Install extra R packages using requirements.R
##   Specify requirements as R install commands e.g.
##   install.packages("<myfavouritepacakge>") or
## devtools::install("SymbolixAU/googleway")

COPY ./DockerConfig/requirements.R /tmp/requirements.R
RUN Rscript /tmp/requirements.R

## uncomment to include shiny server
# RUN export ADD=shiny && bash /etc/cont-init.d/add

# create an R user
ENV USER rstudio

## Copy your working files over
COPY ./Analysis /home/$USER/Analysis
COPY ./Data /home/$USER/Data
```

Dockerfiles are used to build up a container image.

We start **FROM** a base image. Then we **COPY** files or **RUN** extra commands or set specific **ENV** variables. The Dockerfile lives in the top of the project and should be called **Dockerfile** with a capital **D**.

In this example, we are starting from the rocker/rstudio image from docker hub. These are public (not official) but they are solid and very well supported. Rocker also have images for r-base (rocker/r-base) and a geospatial suite (rocker/rstudio-geospatial). This has all the basic spatial libraries (sp, sf) installed plus all the stuff you require outside of R to make them work (e.g. GDAL libraries).

To install extra R libraries we specify them in `requirements.R`. On build, this script is copied onto the instance and run to install the libraries.

Finally the build copies our files over - the `Analysis` folder and the `Data` folder. We put these in the home directory of our user, called `rstudio`.

Build it

Type the following command into the command line. You must be in the same directory as your Dockerfile. Depending on how you have configured your server, you may need to use `sudo` in front of the command

```
docker build --rm --force-rm -t rstudio/hello-world .
```

the `--rm --force-rm` just forces the container to delete itself once its scripts run or you log out. It just stops us filling up the server with lots of containers doing nothing.

Once this has built run

```
docker image list
```

to see your image added to the list. We've called it `rstudio/hello-world` but you can call it anything.

Run it

We want to use this image to access Rstudio so we want it running as a background service (i.e. in detached mode). We use the flag `-d` to do this. If you want to access a bash shell or other interactive mode, you need to specify `-it`.

Rstudio runs on port 8787 within the container. We need to map this to an unused port on the host machine with a `-p <host port>:<container port>` We will use 28787, but this can be any unused port.

We will call our container `hello-world`. This is the simple run command:

```
sudo docker run -d --rm -p 28787:8787 --name hello-world
rstudio/hello-world
```

Run this command and access the container through your web browser at `<yourhostip:28787>`. Username and password are both `rstudio`.

In rstudio, type

```
source("Analysis/hello.world.R")
```

You should be able to see the Analysis and Data folder but there are two problems.

1. It's all well and good to write to the local container but this **data won't be permanent**. We can write our output back to the host directory by mounting a host directory as a volume on the container with `-v /full/path/to/directory`. This is also useful in development as you can make changes in your permanent host folder which are then immediately available on the container without rebuilding it.

2. In order to **write to a file within Docker (through rstudio) you need to have the right userid**. With these rocker images you can get that by specifying `-e USERID=$UID` in the run command. Then you can write and you can make changes to files and save them within the container.

Before we fix the problem we need to stop the container that's running (it's no good for us):

```
sudo docker stop hello-world
```

Now let’s try again. If you look in `run_docker.sh` you will see a more complete version of the run command:

```
DATA_DIR=${PWD}/Data sudo docker run -d --rm -p 28787:8787 --name
hello-world2 -e USERID=$UID -e PASSWORD=SoSecret! -v
$DATA_DIR:/home/rstudio/Data rstudio/hello-world
```

Note in the above I have also set the password manually -- you can make it anything you want.

Run the commands above, log into `<yourhostip:28787>` and try sourcing the script.

It should run and write to the Data folder. `</yourhostip:28787>`

Finally, go back to the command line once more. Type `ls Data` and you should see the output file there also.

One more thing. In the rstudio window, open up `Analysis/hello.world.R` add a line to the bottom - any command you want and save it and run it.

> *Final question - if I check the contents of `Analysis/hello.world.R` on the command line (i.e. back on the host) will it have your new line?*
>
> *Why?*

One last challenge:

(Stop the old container first).

Now run it again, but set it up so you can make changes into `hello_world.R` on the command line and immediately have them show up and work in Rstudio.