Containers for reproducibility

Karl Broman

Biostatistics & Medical Informatics, UW–Madison

kbroman.org
github.com/kbroman
@kwbroman
Course web: kbroman.org/AdvData

In this lecture, we'll look at the how to keep track of dependencies to further enhance reproducible research. In particular, we will focus on Docker containers.

Reproducible research

organize the data and code in a way
that you can hand them to someone else
and they can re-run the code
and get the same results
(the same figures and tables)

A central theme of this course has been reproducible research: organizational strategies and tools so that your computational work can be reproduced.

Dependency Hell

- What software does your project depend on?
  - operating system
  - system libraries
  - R or python
  - packages or modules
  - other tools (e.g. pandoc and LaTeX)
- Can you install all necessary dependencies?
- Have dependencies changed? Do you need particular versions?
- How much time does it take to set things up?

We have not much touched on how to keep track of dependencies. This can be a major challenge. In some research areas, things are evolving rapidly. And the more tools you use, the greater chance that some will change over time and that things will be broken.

Even if you don't rely on much other software, it can still be a huge pain for users to collect and install all of the necessary tools. So much so that they end up giving up.

Capturing dependencies

- R: `renv`

```
renv::init()
renv::snapshot()
renv::restore()
```

  Also see MRAN

- Python: `conda`

```
conda create
conda install
conda activate
conda env list --explicit
```

  Also the built-in `venv`

These best ways to keep track of dependencies: for R, use the renv package from RStudio. For python, use conda.

Also for R, see MRAN: Microsoft is taking daily snapshots of CRAN.

And for Python: I mention conda here, but that's really a general, language-agnostic solution. Also consider the built-in venv solution for "virtual environments."

Or create package/module

- R package
  - dependencies in `DESCRIPTION` file
  - data in `inst/ext_data`
  - analyses as vignettes
- Python package
  - multiple modules, plus `__init__.py` and `setup.py`
  - define dependencies with `setuptools.setup`

Another approach would be to make your analysis project an R package, identifying all of the package dependencies, including the data, and with the analyses included as vignettes.

With python, it seems like you'd want to make a package, which is like a set of modules plus a bunch of structured information (`__init__.py, setup.py`). In the setup, use `setuptools.setup` to define dependencies.

Docker containers

- Light-weight virtual machine
  - Uses the host machine's linux kernel
  - On Mac/Windows, containers run within a separate virtual machine
- Capture all dependencies, down to the OS
- Binary image with everything pre-installed, including data
- Text-based recipes for creating the image
- Can build recipe starting from some previous one

Docker containers are the recommended approach for fully capturing all aspects of one's environments, in a way that is fully portable.

Reproducibility without the difficulties of downloading and installing a bunch of packages and libraries.

You can share your full environment with collaborators, or with an interested reader. It potentially lowers the barrier on being able to play around with your code and results.

Getting started with Docker

- Download and install docker, from `docker.com`
- Get an account at `hub.docker.com`

First, install docker.

Second, get an account at `hub.docker.com`, for which you can download container images, or upload your own.

Docker stuff

- Container
  A running docker thing
- Image
  A binary file with a snapshot of a container
- `Dockerfile`
  Text file with recipe to create a new container

A Docker container is the thing you ultimately run.

A Docker image is a saved snapshot of a container.

A Dockerfile is a plain text file with the recipe for creating a new container.

Rocker images

- Docker containers for R
- Can run locally, and have RStudio in the web browser
- Poke around:
  - hub.docker.com/u/rocker
  - rocker-project.org
  - github.com/rocker-org

```
docker pull rocker/rstudio

docker run -e PASSWORD=[blah] -p 8787:8787 rocker/rstudio

  -v $(pwd):/home/rstudio
```


The Rocker project is an effort to create a set of Docker containers suitable for R users.

Once you've got the rocker/rstudio image running, go to browser and go to `localhost:8787` and use login `rstudio` and password whatever you set in that line.

Note: you need to use 8787, because that port is hard-coded in the docker image.

`-v` to connect the current working directory (or some other directory) to the home directory in the container. `$(pwd)` will fill in the local directory using the unix command `pwd` (for "print working directory")

Changes in that directory in the container will be reflected in the local directory.

Jupyter images

- Docker containers set up for Jupyter notebooks
- Look at hub.docker.com/u/jupyter

```
docker pull jupyter/minimal-notebook

docker run -v $(pwd):/home/jovyan -p 8888:8888 jupyter/minimal-notebook
```


You can also have a docker-based Jupyter notebook accessible in a browser.

To use local directory for notebooks, the docker directory seems to need to be `/home/jovyan`

Note you need to use port 8888 because that is hard-coded in the docker image.

Creating a docker image

- Start from some previous image
- Use a `Dockerfile`
  - explicit
  - human-readabb
  - an often-small script
- Create a container interactively and then write it to an image
  - `docker cp` to copy stuff into the container
  - `docker commit` to save a container to an image file

If you want to make a container for your work, you should first start with an image that has the basic stuff you want.

From that, you can either create a `Dockerfile` with the explicit commands to generate the container you want.

Alternatively, fire up the container, modify it interactively, and then write it to an image file.

Creating a new docker image

```
docker run -d -e PASSWORD=rqtl --name rqtl -p 8787:8787 rocker/rstudio

install.packages("qtl")
download.file("https://rqtl.org/sug.csv", "sug.csv")

docker commit rqtl rstudio_rqtl

docker tag e3ae59d1443f kbroman/rstudio_rqtl:firsttry
docker login
docker push kbroman/rstudio_rqtl
```


Suppose I wanted to create a Docker image with R/qtl installed plus with a dataset included.

I could start a Docker container running and then install R/qtl and download a datafile, and the container is then in the state that I would want it.

I then use `docker commit` to save it to an image file.

Finally, `docker tag` to tag the current state, and `docker login` to log into `hub.docker.com`, and `docker push` to push the image to `hub.docker.com`.

You'll need to then go to `hub.docker.com` in a browser, log in, find your new repository, and add a description.

Example Dockerfile

```
FROM java
MAINTAINER daroczig@rapporter.net

## Prepare folder for the Minecraft stuff
RUN mkdir -p /minecraft

## Download Spigot build tools
RUN wget https://hub.spigotmc.org/jenkins/job/BuildTools/[clip]/target/BuildTools.jar -P /minecraft/

## Build the Spigot server
RUN cd /minecraft && java -jar BuildTools.jar

## Symlink for the built Spigot server
RUN ln -s /minecraft/spigot*.jar /minecraft/spigot.jar

## Accept EULA
RUN echo "eula=true" > /minecraft/eula.txt

## Download and install the RaspberryJuice plugin for API access
RUN mkdir -p /minecraft/plugins \
    && wget https://github.com/zhuowei/RaspberryJuice/raw/master/jars/raspberryjuice-1.11.jar \
    && mv raspberryjuice-1.11.jar /minecraft/plugins/

## Open up API port
EXPOSE 4711
## Open up Game port
EXPOSE 25565

## Start the server
CMD cd /minecraft; java -Xms512M -Xmx1G -XX:MaxPermSize=128M -XX:+UseConcMarkSweepGC -jar spigot.jar
```


This is the Dockerfile for a minecraft server, from the miner package.

`https://github.com/kbroman/miner/blob/master/inst/Dockerfile`

Use `docker build -t minecraft` to build it to an image named `minecraft`.

`FROM` which takes the java image as the base for this new one

`RUN` to run a command

`EXPOSE` to open a port

`CMD` is run when the container starts

Another example

`github.com/rocker-org/rocker-versioned`

`/rstudio/latest.Dockerfile`

Poke through the docker files for the Rocker images. You can see why you wouldn't want to start from scratch.

Managing Docker stuff

```
docker images
docker image ls

docker ps -a
docker container ls -a

docker container stop adoring_hamilton
docker container start adoring_hamilton

docker rm adoring_hamilton
docker image rm alpine
docker rm $(docker ps -a -q)
```


First two commands to list images.

Next two commands to list containers.

`stop`/`start` to pause a container and set it running again.

Last few commands to remove containers and images. The last command removes all stopped containers.

binder

- mybinder.org
- add two files to a github repo → docker container in the cloud
  - `runtime.txt` telling date of R
  - `install.R` with `install.packages()` calls
  - special url with `?urlpath=rstudio`
- examples:
  - kbroman.org/blog/2019/02/18/omg_binder
  - github.com/kbroman/Teaching_CTC2019

Binder is a magical tool for turning a github repository into a docker container running in the cloud.

RStudio or a Jupyter notebook running in browser, with all your stuff ready for play.

Summary

- Want to capture the full environment for a project
  - code + data
  - dependent packages, libraries
- Want to lower the barrier to the set-up of this stuff
- Docker containers
  - portable
  - shareable
  - extendable
  - `Dockerfile` script to define
- `mybinder.org`
  - github $\rightarrow$ docker in the cloud
  - magical set-up

I always like a summary.