Docker Quick-start Guide

Gregory J. Hunt, Johann A. Gagnon-Bartsch

The following is a quick-start guide to building and running a docker container. Here, we show how to build, run, and interact with a minimum working example.

In addition to this guide, readers may find the following resources useful:

- docker installation instructions
- official Docker documentation
- starting guide

1. Running a Docker container

To run our minimum working example one can use the following command typed into either the linux shell or windows command prompt

This will download a container from Dockerhub and start it. Depending on the operating system settings one may need to run this with elevated privledges (e.g. using sudo in linux).

```
docker run -it --name ex_container --rm -p 127.0.0.1:80:80 gjhunt/mwe:2
```

In our case, once the container is started a welcome message is displayed:

```
To get started, please enter the following URL into your web browswer:

          http://localhost/

Note: If you are unable to copy and paste the link with your mouse,
you may need to use command-c or control-c on the keyboard to copy,
and command-v or control-v to paste.  Alternatively, you can enter
the link manually.

To quit, type exit and press enter.
```

One can interact with the running container either through the shell/command prompt or follow the `localhost` link displayed. We recommend the latter.

We will now explain the various parts of this `docker run` command. (Full documentation of `docker run` may be found on the official docker docs here.)

- `docker run`

The main command to run docker images is `docker run`. This will either run a local image (if one exists) otherwise it will pull the image from Dockerhub and run it.

- `-it`

The `-it` flag specifies that the container should be run in interactive mode. For exploring analyses interactively this is generally how an image will be run.

- `--name ex_container`

The `--name` flag names the container. In our case we name it `ex_container`. While this is optional, it is good practice to give meaningful names to containers if one is running multiple containers at once.

- `--rm`

The `--rm` flag tells `docker` to remove the image after we exit. If this flag is not specified then the container will continue to run in the background after we have exited. We recommend this unless running in the background is specifically desired.

- `-p 127.0.0.1:80:80`

This flag forwards the ports from the container to the localhost at port 80 so that the container can be accessed through `localhost`. Here we use the full IP address 127.0.0.1 so as to ensure that the docker container is not exposed to the broader network.

- `gjhunt/mwe:2`

The final part of the command is the image name. In this case our image name is `gjhunt/mwe` and the additional tag `:2` is a tag that specifies the version. We will see when building our image that one can specify arbitrary text as a tag at the end of the Docker image name to create different versions of the image. If unspecified `docker` implicitly uses the tag `:latest`. In our case, we have specified version `:2` of our minimum working example.

Using `docker run` is typically all that is needed to be done in order to run the container supplied by a third party. Nonetheless, one can write their own docker images. This is covered in our next section.

2. Building a Docker container

In order to build a docker image one writes a `Dockerfile`. A `Dockerfile` file specifies how the container should be set up. One can view the `Dockerfile` we use in this example on github. One can download all of the files we use for this project here.

In full, our `Dockerfile` has 6 lines:

```
FROM johanngb/rep-int:latest
WORKDIR /home/rep/
COPY data.csv data.csv
COPY analysis.ipynb analysis.ipynb
RUN jupytext --set-formats ipynb,rmarkdown,R analysis.ipynb
RUN jupyter nbconvert --to html analysis.ipynb
```

We explain the commands in this file in order:

- `FROM`

The `FROM` command specifies the base image off of which to build the docker image. Full documentation for this command can be found in the Docker documentation here. In our case, we build off a Docker image we have created to give interactive exploration tools for statistical analyses. Our image is called `johanngb/rep-int`. The tag at the end of this image name `:latest` specifies that we should be building off of the newest released version of this image.

- `WORKDIR`

The `WORKDIR` command specifies the location within the container where the user will begin. It also serves as a base directory from which further commands like `COPY` are referenced later in the `Dockerfile`. In our case we create set the starting directory to `/home/rep`. The official docker docs for `WORKDIR` are available here.

- `COPY`

The `COPY` command copies files from the computer into the image. Here we copy a data file data.csv and an analysis jupyter notebook analysis.ipynb. Since we set `WORKDIR` to `/home/rep` these files will be copied into the `/home/rep/` directory within the container. Sometimes it is helpful to add the `--chown=1000` command to correctly set the permissions of the copied files (depending on the base image). We have written our base image so this is not necessary. The official docker docs for `COPY` are available here.

- `RUN`

The `RUN` command runs commands when building the Docker image. In our case we first run `jupytext` to mirror the `.ipynb` notebook into `rmarkdown` and and `.R` script. This creates

the appropriate files we will see when we run the container. Additionally, we use the RUN command to convert our notebook for a `.html` file for display using the `jupyter nbconvert` command. Other common idioms are to install software add-ons to the image. In our image, R and `python` are already installed. We could use the RUN command to install the `ggplot2` package by appending the following line to our `Dockerfile`

```
RUN R -e "install.packages('ggplot2',repos='http://cran.us.r-project.org')"
```

The typical workflow for writing a `Dockerfile` involves one FROM at the beginning of the file to set the base image, one `WORKDIR` to set the working directory, and then a series of invocations of `COPY` to copy over analysis files, and RUN to run commands, install packages, etc.

Once the `Dockerfile` has been written, one simply runs the `docker build` command from within the directory where the `Dockerfile` resides. To do this we use a command akin to the following:

```
docker build -t username/mwe:2 .
```

This command `docker build` builds the image from the Dockerfile naming it the argument to the flag `-t`. In our case we specify `-t username/mwe:2` which names the image `username/mwe` and specifies the tag `:2` indicating that it is version `:2`. The official docs for `docker build` can be found at this link.

For local use any naming convention can be adopted, for example, one could instead use the command

```
docker build -t myimage .
```

which would build the same image but name it `myimage`. Notice in both instances we end the command with a period `.` which specifies the build context i.e. that the `Dockerfile` and associated files to `COPY` are in the current directory.

If one is going to upload their image to `Dockerhub` then they need to follow the convention `<username>/<imagename>` for naming their images.

After building the image one can test it by running it by using the `docker run` command as above. In this case, if we have named the image `myimage` then one can run it with the following command:

```
docker run -it --name ex_container --rm -p 127.0.0.1:80:80 myimage
```

3. Saving and Loading Docker images

If desired, one can upload their image to `Dockerhub` using the `docker push` command (official docs here). For example this could be done using

```
docker push <username>/<imagename>:<tag>
```

substituting the `<username>`, `<imagename>`, and `<tag>`, for appropriate values.

Alternatively, one can save a local image using `docker save` (official docs here) This saves the image as a file on the computer that can be shared like any other file. For example, we recommend uploading images to Zenodo.

In our case one could use the following command to save their image named `myimage`

```
docker save -o image.tar myimage
```

This saves the images to a `.tar` compressed file named `image.tar` that can be shared. Here

the flag `-o` specifies the file name `image.tar` we want to give our image on disk. We also specify the image name `myimage` at the end.

One can then share this with `.tar` file with third-parties. They can load the image using the command `docker load` (official docs here). For example:

```
docker load -i image.tar
```

Here, the tag `-i` specifies the `.tar` file to load. After this file has been loaded it can be run just like any other `docker` image. For example:

```
docker run -it --name ex_container --rm -p 127.0.0.1:80:80 myimage
```

4. Other Useful Commands

In addition to `docker run`, `docker build`, `docker save` and `docker load` the following are some useful commands for handling containers on a system:

- `docker image ls -a` to see images (Official Docs)
- `docker image rm` <name> to remove image <name> (Official Docs)
- `docker container ls -a` to see containers (including stopped ones) (Official Docs)
- `docker container rm` <name> to remove container <name> (Official Docs)
- `docker system prune` to clean up no-longer-needed files (Official Docs)
- `docker cp` <name>`:`<`src`> <`dest`> to copy files from a running container <name> in the folder <`src`> to the local machine at destination <`dest`> (Official Docs)