Welcome to The Carpentries Etherpad!

This pad is synchronized as you type, so that everyone viewing this page sees the same text. This allows you to collaborate seamlessly on documents.

Use of this service is restricted to members of The Carpentries community; this is not for general purpose use (for that, try https://etherpad.wikimedia.org).

Users are expected to follow our code of conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

All content is publicly available under the Creative Commons Attribution License: https://creativecommons.org/licenses/by/4.0/

Welcome to the Intro to Docker Workshop!

Links:

Workshop Webpage:https://uw-madison-datascience.github.io/2022-09-07-uwmadison-mini/
Curriculum: https://carpentries-incubator.github.io/docker-introduction/
Intro Slides: https://docs.google.com/presentation/d/1fWV1WcfkL8Jlrp8spzO6kBsXjHkGGU5kEJCWncr2Pck/edit?usp=sharing
Wrap-up slides: https://docs.google.com/presentation/d/1H3LquxdRlA5GiBAkPtrZyQhG9IGGj7NaEtjJ_2E2nAc/edit?usp=sharing
Feedback: https://forms.gle/FfeiEAe473C2e3Bz5

Sign-in Day 1:

Name, Department, and What motivated you to take this workshop? What is one thing you'd like to learn about Docker or use it for?

- Sarah Stevens, Data Science Hub, I'm motivated to teach this workshop because containers can help make scientific software more portable and improve produciblity of scientific analyses. - *Instructor*
- Chris Endemann, Data Science Hub, I'm motivated to help others learn how to reproduce analysis workflows
- Beth Lett, ERP/SMPH. I am motivated to take this workshop because I've only used one docker before and know little about them and how to run them.
- Trisha Adamus, Ebling Library, I think Docker is great and I would like to incorporate it into my work more often than I do.
- Jingrui Wei, Department of Materials science engineering, would like to learn more about docker so that make it easier to run job on remote servers.

- Erwin Lares. RCI. I know a little bit about Docker and I want to become more familiar with it
- Marine (Pin Chun).
- Meltem Uyanik

Sign-in Day 2:

Name, Department

- Sarah Stevens, Data Science Hub - *Instructor*
- Chris Endemann, Data Science Hub - *Helper*
- Mariah A. Knowles, Data Science Hub - *Helper*
- Beth Lett, ERP/SMPH.
- Marine (Pin Chun).
- Trisha Adamus, Ebling Library - Helper
- Erwin Lares. RCI. I want to learn to use Docker to submit jobs to CHTC
- Meltem Uyanik

Notes (scroll down for day 2 notes, line 213):

```
# Open terminal (mac) or PowerShell (windows)
docker --version # check docker version

docker # print manual for docker

docker container --help # get manual specifically for container management command

# How to use someone else's images and containers
docker image ls # list images currently on your local machine

# Let's get an image to use. We'll use the hello world image first
docker image pull hello-world # pull down the hello-world image from Docker Hub

docker container run hello-world # run the hello-world container in its default configuration

docker run hello-world # same as above, but explictly saying container helps reinforce that a container is being run rather than an image

docker container run hello-world > test-hello-world.txt # direct output of container's run to a text file

# 3 things happen when we run the container
# 1) start container
# 2) perform default action
# 3) shut down container

# Note: the container still exists after it shuts down, and you can run it again at any point.
```

Try downloading the alpine container image and using it to run a container. You can do it in two steps, or one. What are the steps?
docker container run alpine # pulls alpine from Docker Hub (if not already on your local machine) and runs it

What is an image? An image specifies the instructions to start a container, and the contents within the container. One analogy is to think about making copies of a piece of paper. You can think of an image as the original piece of paper, and the container as one of the copies. Another analogy is to think of the image as a cookie cutter template, and each cookie cut out as an individual container.

Let's see what containers we have on our local mahcine
docker container ls # only shows running images and so nothing shows up (nothing running at the moment)
docker container ls --all # show all instances of previous container runs

docker container run alpine cat /etc/os-release # run alpine image and create new container doing so, run cat /etc/os-release inside the container, and then stop the container

Can you run a copy of the alpine container and make it print a "hello world" message?
docker container run alpine echo "hello world"
docker container ls -a # same as -all. We can see we have a new container from running alpine with "hello-world" printed out rather than running the default command

Run interactive container. Have to specify sh for shell. All actions you type after running the below command (e.g., pwd, ls) will take place inside the container.
docker container run -it alpine sh
exit # exit container
pwd # back in your local machine

Exercise
Can you find out the version of Ubuntu installed on the ubuntu container image? (Hint: You can use the same command as used to find the version of alpine.)
Can you also find the apt-get program? What does it do? (Hint: try passing --help to almost any command will give you more information.)

# interactive version
$ docker container run -it ubuntu sh

# OR use bash shell alternatively
$ docker container run -it ubuntu bash

THEN
/# cat /etc/os-release
/# apt-get --help

/# exit

non-interactive solution

$ docker container run ubuntu cat /etc/os-release
$ docker container run ubuntu apt-get --help

Note: adding --rm to a run command will automatically delete the container after it's done running. We'll explore this later.

Difference between sh and bash? They're both shells (command lines), but slightly different versions. Bash (Born Again Shell) has slightly different commands than shell.

docker image ls
docker container ls -a

docker image rm hello-world # error; can't remove image that has existing containers (even when those containers have stopped running)

to remove a container, reference its container ID (shown in docker container ls -a) or reference its name (also shown in the ls -a command)
docker container rm NAME_OF_CONTAINER # you'll have a different container name for hello-world

docker container prune # remove all stopped containers. You'll see a warning message. Press y to continue with the prune operation
docker container ls -a # no containers

docker image rm hello-world # no error. hello-world image removed successfully.

docker image ls

Go to hub.docker.com. If you don't have an account, make one during the next break please! We'll use those accounts more tomorrow.

You can search docker hub for 'python' to see images related to python
Verified content (most trusted) and community images (not as tested as verified content). Typically, you want to stick with verified images.

Exploring tags from the python image (different versions of an image)
- usually one called "latest" and others with specific version tags

pull specific tagged version of python image
docker image pull python:3.8 # version 3.8 of python; represented by the 3.8 tag
docker image pull python:3.6 # version 3.6 of python; represented by the 3.6 tag

docker image ls # check images on local machine

Note: you may also see images pulled that reference a specific user, e.g. docker image pull sstevens2/python-test:latest to pull Sarah's latest python image (don't actually run this since python-test doesn't actually exist)

Exercise: How would I download the Docker container image produced by the rocker group that has version 3.6.1 of R and the tidyverse installed?

Note: the container image described in this exercise is large and won’t be used later in this lesson, so you don’t actually need to pull the container image – constructing the correct docker pull command is sufficient.
docker image pull rocker/tidyverse:3.6.1

alternatively, you can skip the image specification
docker pull rocker/tidyverse:3.6.1

What is an image? An image specifies the instructions to start a container, and the contents within the container. One analogy is to think about making copies of a piece of paper. You can think of an image as the original piece of paper, and the container as one of the copies. Another analogy is to think of the image as a cookie cutter template, and each cookie cut out as an individual container.

Creating our own image. Start with a template image that you want to base your image off of. We'll use the alpine image as our starting point to create our own image.

interactively test out installation of alpine
docker container run -it alpine sh
pwd
python3 # python3: not found. We'll need to install python within the alpine container
apk add --update python3 py3-pip # apk is how you can install python in alpine
python3 --version # python 3.0 successfully installed

test if pip installed correctly. pip is the package manager that python uses
pip install cython

exit # exit container

docker container ls -a

When creating an image, best practice is to have a folder dedicated to the image build
cd Desktop/docker-intro/
ls # basic folder for first build we'll do. sum folder for build we'll do tomorrow
cd basic/
ls # Dockerfile is found here. When building an image, you'll need a Dockerfile
nano Dockerfile # you'll see a template for what Dockerfiles typically look like

editing Dockerfile
FROM alpine # our image will use the alpine image as its starting point
RUN apk add --update python3 py3-pip python3-dev
RUN pip install cython
CMD ["python3", " --version"] # exec format, spaces replaced by commas
to exit nano....
Ctrl+x
y
Enter
cat Dockerfile # check file contents

Once we've written our Dockerfile, we can then do the docker build step

docker image build -t alpine-python . # -t flag to give our image a specific name (alpine-python in this case). Build from within current folder by specifing . or ./

docker image ls # new image appears! the default tag is latest because we didn't specify a tag

adjust the name of the image (add username leading up to it)
docker image tag alpine-python YOUR_DOCKER_USERNAME/alpine-python

docker login # login to your docker hub account by entering in your Username and password

push our new local docker image to docker hub
docker image push YOUR_DOCKER_USERNAME/alpine-python # need your docker username before the image name

Day 2 Notes

open terminal/powershell
cd Desktop/docker-intro/sum
ls

docker image ls
docker container run alpine-python python3 sum.py # attempt to run the sum.py script from the alpine-python container (No such file or directory because our sum.py file isn't inside this container)

Let's "mount" our script (i.e., share files/folders between host system and container) and then run the sum.py script
docker container run --mount type=bind,source=${PWD},target=/temp alpine-python python3 /temp/sum.py # ${PWD} is a shortcut to represent the current folder; place source folder in a root folder called temp

What happens if you use the docker container run command above and put numbers after the script name?
docker container run --mount type=bind,source=${PWD},target=/temp alpine-python python3 /temp/sum.py 14 28 37 291 29292 # sum = 29662

Our Docker command has gotten much longer! Can you go through each piece of the Docker command above and explain what it does? How would you characterize the key components of a Docker command? Here’s a breakdown of each piece of the command above

- docker container run: use Docker to run a container
- --mount type=bind,source=${PWD},target=/temp: connect my current working directory (${PWD}) as a folder inside the container called /temp
- alice/alpine-python: name of the container image to use to run the container
- python3 /temp/sum.py: what commands to run in the container

More generally, every Docker command will have the form: docker [action] [docker options] [docker container image] [command to run inside]

Try using the directory mount option but run the container interactively. Can you find the folder that’s connected to your host computer? What’s inside?

```
docker container run --mount type=bind,source=${PWD},target=/temp -it alice/alpine-python sh
pwd
ls
cd temp
ls
touch test.txt # make blank test.txt file
ls
exit

pwd
ls # we still have the test.txt file in our host machine

nano Dockerfile # remember we'll use this for our image build
Ctrl+x
y
enter

cp ../basic/Dockerfile .
nano Dockerfile # we see the file we made yesterday

# copy existing file into our image when we build it
FROM alpine
RUN apk add --update python3 py3-pip python3-dev
RUN pip install cython
COPY sum.py /home # place file in new folder called home
CMD ["python3", "--version"]

Ctrl+x # exit nano
y # save changes
enter

cat Dockerfile
docker image build -t YOUR_DOCKERHUB_USERNAME/alpine-sum . # the . represents the current folder (where we want to build the docker image from; where the Dockerfile is and all files included in the image)
docker image ls # new image appears

# Can you remember how to run a container interactively? Try that with this one. Once inside, try running the Python script.
docker container run -it YOUR_DOCKERHUB_USERNAME/alpine-sum sh
python3 /home/sum.py 10 15 20 # sum=45

# tar (compress) the docker image
docker image save YOUR_DOCKERHUB_USERNAME/alpine-python -o alpine-python.tar
ls
```