CS 346 (Fall 21): Introduction to Computer Programming II

Project #1

due at 5pm, **Fri** 10 Sep 2021

Overview

Soon, we'll be writing true Projects, where you will devise code to solve a problem. But right now, we're very early in the class, and so this "Project" will (mostly) be a series of exercises.

For each Exercise, I will ask you to investigate something or run some commands; I will then have an itemized list of questions. Turn in a single file, which contains all of the answers to the numbered questions. Name your file `proj01_[yourNetID].pdf` .

I will also ask you to make a script or two. Turn those in as well, using the filenames that I've provided.

Exercise 1

Exercise 1 was deleted. **Do not turn in any answers for Exercise 1.** However, if you're curious what's going on, you can investigate the **Layers** section of `docker inspect`. It's true that Docker tracks changes to your filesystem as a series of 'diff's, but this tracking happens at the level of layers, not images.

Instead, an image is **made up of layers** (plus some other things). Each time that you create a new image, you are creating a new layer - so the image that you create is defined as all of the layers of your old image, plus one new layer.

If you're interested, try it out: build a chain of images like we've done in class, and then look at the Layers section of `docker inspect` for each one. You will see that each additional image has one more layer than the last:

```
"RootFS": {
    "Type": "layers",
    "Layers": [
        "sha256:7555a8182c42c7737a384cfe03a3c7329f646a3bf389c4bcd75379fc85e6c144",
        "sha256:0f481154a61fa835e94d32457a625f0b2bc8adac14d31d86ee06773684177e23",
        "sha256:f22211d869cfc3248f8dfce31b14f3eb0691cb940756d5a8e5a8ce9fe1454ab5"
    ]
},
```

So what was my problem? It turns out, when you upload an image to Docker Hub, it uploads the image with all of the layers exactly as you had them - but it forgets any parent/child relationships for images. This is true even if you upload **both** the parent and child image.

Docker is still very smart about efficiency. If you upload both a parent and child image, they will share a bunch of layers, and each layer will be downloaded only once. And each layer is only stored on your local computer once. But even so, if you upload a parent/child pair, and somebody else downloads them, they will share layers - but they will **not** show up as parent and child.

sigh.

(exercises continue on the next page)

Exercise 2 - `ssh`, `scp`, and Shell Scripting

From inside an `ubuntu` container, `ssh` to Lectura. I have created a directory inside my home directory which you can access; go into `/home/russelll/cs346_exercises` . Use `ls` to figure out what's in there; you'll find a script that I've written. It's quite harmless; feel free to try it out, or to open it up and read the code. Then answer the questions below.

(a) Run the script on Lectura, and report what it prints there.

(b) Copy the file into your container (with `scp`, not by hand!) and run it there. Report what the script prints there.

(c) Give the exact `scp` command that you used to copy the file from Lectura to your container.

(d) Open the script with your favorite text editor. What command does Linux use to execute this script? (That is, what do you find on the shebang line?)

(e) Shell scripts are actually quite simple; they are a list of commands which should be executed on the shell. Investigate this script, and get a rough idea how it works. While you won't understand it all, you should be able to make educated guesses about what most of it does.

There is a trivial trick that you can do which will fool the **second** test in this script, to make it think that it's not inside a container anymore. What is this simple change you can make to the container itself?

(In case you're wondering, I don't know how to fool the first check.)

(exercises continue on the next page)

Exercise 3 - Repeatable Config

Now that you know a tiny bit about shell scripting, you will write a script which automatically configures an `ubuntu` with various packages (and creates a few simple files as well).

Name your file `proj01_[yourNetID]_config_ubuntu` .

Design your shell script to run inside the container; its shebang line must specify that it runs under `bash`. It should perform all of the steps necessary to take a brand-new `ubuntu` container, and install a bunch of software packages. (Install at least six.)

In addition, this script should create some "config" files in the root directory of the container. These files are **meaningless**, but use `echo` to write a simple message into each. Name them `config1.txt, config2.txt, config3.txt`

Test out your script, and confirm that it works properly. To test, it, do the following:

- Create a fresh container based off of `ubuntu`
- Use `docker cp` to copy the script into the container
- Run the command inside the container

If your script is working correctly, your container should be fully configured when the script runs.

GRADING NOTE:
While your TA will probably not run everybody's script (because it would take up too much time and network bandwidth), they **will** be looking at it, so make sure to test it!

COOL INFO:
Docker has a mechanism for automating the creation of new Docker images. You define something called a "Dockerfile," which has a list of commands to execute. When these commands are done, Docker automatically commits the new image.

While a Dockerfile is not the same thing as the script you're writing for this Exercise, it has some important similarities. Hopefully, we'll have time to discuss them later in the semester.

(exercises continue on the next page)

Exercise 4 - `alpine`

Linux has many "distributions," which are different groups' attempt to package all of Linux (its kernel, plus its **many** different tools) into easily-used groups. We've been playing with the `ubuntu` distribution in Docker, but there are others: `fedora` is produced by RedHat, and `alpine` is a minimal Linux distro, perfect for containers.

In this Exercise, you will investigate how to set up a container based on `alpine`. But I'm not going to tell you how to do it - I want you to Google for the answers.

Academic Integrity
Worried whether it's OK to Google for answers to such things? Remember, the rule in this class is as follows:

- Never search for an algorithm, or a solution to a problem that requires creativity or engineering ...
- But it's always OK (except on Exams) to use the Internet to **educate yourself about something you don't know.**

In this case, you're going to need to learn how to install packages on an `alpine` container. You're also going to have to do some research about which package names you would use to install each package.

(a) Investigate each of the following commands. Which ones exist in your `alpine` container?

- `cat`
- `curl`
- `grep`
- `less`
- `nano`
- `python3`
- `ssh`
- `telnet`
- `vi`

(Turn in your observations with the rest of the Exercises.)

(b) Next, remember the script from two exercises ago, which checked if you were inside a container? Copy this into an `alpine` container and try it out; you will see that it completely fails to run.

Consider the shebang line in the script, and explain why it doesn't run.

(Turn in your observations with the rest of the Exercises.)

(c) Now, write a config script - just like you did in the previous Exercise. Name this one `proj01_[yourNetID]_config_alpine` .

In this one, you aren't required to create any dummy text files, but you must install packages for **all** of the commands, from the above list, that are missing.

Do not install any packages which are unnecessary.

(Turn in this file.)

WARNING: Think carefully about the shebang line for this script. What command can you use to execute this script, if you start with a perfectly stock `alpine` container?