IST769 Lab B

Docker, git and the command line.

In this lab we will learn about docker, git, and the command line, also known as the CLI or Command Line Interface. We will also learn how to take screenshots.

Learning Outcomes

At the end of this lab, you should be able to:

- use the command line interface on your computer,
- use simple git commands from the command line,
- identify if you are in the correct git folder,
- start and stop docker lab containers,
- determine which docker containers are running,
- discover open ports of your running docker containers
- understand how to take acceptable screenshots for problem set submission

Pre-Requisites

Before you begin make sure you've connected to the Azure Lab environment.

Part 1: Command line interface (CLI) basics

In this section we will learn just enough Command Line Interface to be useful in this course. Everyone should learn the command line. It is by far the simplest method to work with a computer. Yes it's not a pretty as the Operating System's GUI (Graphical User Interface), but is very effective for most tasks, especially when you need to sequence commands together into a script!

1. Open the command line. Look for the Windows terminal icon. In the Windows Start Menu. It looks like this:

   Click on the icon to launch windows terminal

2. In your window you should see a **command prompt** the command prompt is where you enter commands. ;-) It will let you know the folder on your computer for which you are running the commands. The default windows terminal prompt is the PowerShell prompt. It starts with **PS** and ends with **>**

   NOTE: The prompt can be changed, but this is the default prompt.

3. In most documentation, the symbol used for the command prompt is **$**. That way authors do not need to express the command prompt for each Operating System or shell within that operating system.

NOTE: You will see many different command prompts in this course. We will use PowerShell on Windows, Bash on Linux inside the Docker containers, SQL command prompts as well as special prompts for each of the databases and their command-line utilities. For clarification, I will display these prompts differently so you have a better indication of where you should be when you execute each command. To start:

- On Windows, the PowerShell prompt will be represented like this: **`PS:>`**
- On Linux, the prompt will be represented like this: **`$`**

4. The command line just waits for us to enter commands.

   - Click your mouse inside the **PowerShell** window to activate it.
   - Press the ENTER (RETURN on a Mac) key a couple of times, and you should see the command prompt repeating.
   - This is because you are entering a command, it is finishing and then prompting you for the next command.
   - EVERY command typed at the command line must end with ENTER/RETURN to activate the command.

5. Let's try our first command. From the command prompt, enter:
   ```
   PS:> pwd
   ```
   to print the current working directory. This will display your location on the computer's disk.
   NOTE: I used the general syntax to indicate the Windows PowerShell - do not type **`PS:>`** !!!

6. Bad commands! When a command works, the command line interface won't say much about it. When the command doesn't work, you will get an earful. This is by design as the more experienced you get with the command line the less assurances you will required from it.

   Let's type a bad command, Type:
   ```
   PS:> fudge
   ```

   The PowerShell prompt will output:
   ```
   the term 'fudge' is not recognized…
   ```

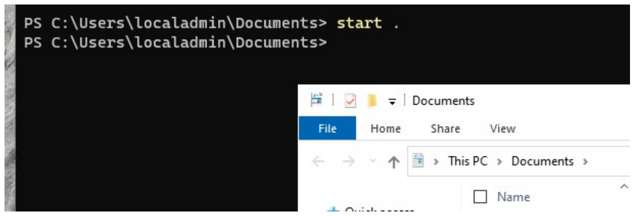
   Not very helpful, but at least we know something was wrong!

7. Let's open the same disk location within the Operating System GUI (Graphical User Interface) as within the command line.

   At Windows PowerShell, type:
   ```
   PS:> start .
   ```
   ←← Don't forget the period! This refers to the current folder

This will launch Windows Explorer in the current working directory.

8. In the GUI window, titled you see a window titled “Documents” which matches the name of the CLI folder.



```
PS C:\Users\localadmin\Documents> start .
PS C:\Users\localadmin\Documents>
```

In the GUI, you see the contents of the folder, **advanced-databases**.

To see this at Windows PowerShell, type:

```
PS:> ls
```

Both windows now display the **advanced-databases** folder.

9. The **advanced-databases** folder contains the code instructions for our lab environment. You will need to change the current working directory (folder) to this folder before starting any lab activities.

To change the current directory, at the Windows PowerShell, type:

```
PS:> cd advanced-databases
```

The command prompt should now read:

```
advanced-databases>
```

Indicating the current working directory is now a different folder.

NOTE: This only works because the **advanced-databases** folder is within the current folder, the **Documents** folder.

Summary

Before you can start your lab you will need to:

- Open Windows Terminal

- Change the current working directory to **advanced-databases**:
  ```
  PS:> cd advanced-databases
  ```
- You need to do this because the code to run the labs is contained within this folder.

Part 2: Git Source Code Management

As we learned in the previous section, the **advanced-databases** folder contains the code to run the lab environment. Every student in the course has a copy of this code. If your professor needs to make a change to the code, how can they easily share that change with you? This is the purpose of source code management, and what the **git** tool provides.

The **advanced-databases** folder is a special kind of folder called a **code repository.** Basically, the changes in this folder can be tracked and versioned. In addition, the folder as a mirror in the cloud, called the **remote**.

If I need to update code for the entire class, I can send my changes to the remote, and then you can execute a command to receive them.

Let's take time to learn some basic Git commands.

10. How do I know this is a Git repository? Valid question! Type:
    ```
    PS:> git status
    ```

    You will see this output:
    ```
    On branch main
    Your branch is up to date with origin/main
    ```

    The **git** output tells us 3 things:

    1. We are working in the **main** branch. Branching allows us to work on separate features independently. In this class we will stick to the **main** branch unless instructed otherwise.

    2. The **main** branch is up to date with the remote copy on **origin**. **Origin** is the name of the remote in this case. It's a strange name but git uses it by default.

    3. A list of any code that might need to be committed (none appears in this example)

11. You might be wonder where is this origin? You can find that out by typing:
    ```
    PS:> git remote -v
    ```

    Your output will look something like this:
    ```
    origin  https://github.com/mafudge/advanced-databases.git (fetch)
    origin  https://github.com/mafudge/advanced-databases.git (push)
    ```

The origin will likely be a public Git hosting service like Github.com or Gitlab.com. Notice how there are two urls one for **fetch** and another for **push**. This provides flexibility in our git flow, we can fetch code from one URL and push the changes we make to it to a completely different one.

12. There might come a time when you need up **update your local code to match the remote**. Common use-case might be fixing an issue with the lab or distributing updated examples. To get an update locally, type:

    ```
    PS:> git pull
    ```

    NOTE: You will be instructed to perform pulls. There is no need to do this routinely.

Part 3: Docker and the lab images

In this part we will explore how to run the pre-built docker lab images.

At this point you might be wondering what is docker and how does it work? Luckily, I've made a video to explain it.

https://youtu.be/fQORO9QEJN4

Please Watch the Video Before Proceeding

In this course we will use **docker-compose** almost exclusively. This command only works when you are in the same folder as the **docker-compose.yaml** file.

13. What are the available services? To list the available services, type:

    ```
    PS:> docker-compose config --services
    ```

14. What services are running? Type:

    ```
    PS:> docker-compose ps
    ```

    The **State** column that says **Up** indicates the service is running.
    The **Ports** column displays the tcp or udp ports that the service uses to communicate. These ports are open and exposed to the host computer.

15. Starting services is simple. The only caveat is knowing the name of the services. This example starts the **minio** and **Jupyter** services. Run this command now:

    ```
    PS:> docker-compose start minio jupyter
    ```

    Perform a `docker-compose ps` to verify the two services are running.

16. Sometimes you need to view the logs of a service to troubleshoot an issue or capture output. This example views the Jupyter logs.

```
PS:> docker-compose logs Jupyter
```

17. Stopping services you are no longer using is important to for conserving resources on the lab computer. To stop all services, type:

```
PS:> docker-compose stop
```

NOTE: You should plan on doing this at the end of each lab.

Part 4: Screenshots Guidelines

You'll have to take screenshots as part of turning in your homework problem sets. Here are the guidelines for screenshots:

- Only screenshot the window or portion of the window relevant to answering the question.
- The screenshot should include code AND output of the code.
- Screenshots should be appropriately sized and easily legible.
- Your name or netid must be included within EVERY screenshot, as evidence it is YOUR screenshot.
    - For code, this can be as simple as including your name in the comments.
    - For screenshots with output, you can include a notes window with your name/netid in it.

Taking Screenshots

- The Windows 10 snipping tool is an excellent choice for taking screenshots: https://support.microsoft.com/en-us/windows/open-snipping-tool-and-take-a-screenshot-a35ac9ff-4a58-24c9-3253-f12bac9f9d44
- You can also try the open source tool called Green shot: https://getgreenshot.org/

Lab Problem Set

Answer each of the following using the Problem Set Submission Form, included with this lab.

1. Take a screenshot of an open Windows Terminal with the current working directory set to the **advanced-databases** folder. Make sure to follow the screenshot guidelines and include your name/netid in the screenshot.
2. Explain the difference between a Docker image and a container. No screen shot necessary.
3. What is the purpose of a docker volume? No Screen shot necessary.
4. Start the redis and retwis services with docker-compose. Provide a screenshot of the command you typed, and another screenshot showing the two services are running.
5. What does the last line in the redis logs say? Provide a screenshot of this message. What command did you have to type to see the redis logs?
6. Which ports are being used by the currently running docker services? How do know they are running, and which ports are being used?
7. What are the volumes created by all the services in the docker-compose file? What command did you type to get this answer and provide a screenshot of the output.

IMPORTANT: When you are finished with the lab, execute:

```
PS:> docker-compose stop
```

To turn off all running services, then shut down your Azure Lab instance.