Project 1: Warm-Up and Basic Setup

Overview

Throughout this quarter, you will be gradually developing a blogging Web site, which can be considered as a simplified version of WordPress. The entire Web site will be built on the "MEAN stack" (MongoDB, Express, Angular, and Node.js) eventually, but the MEAN stack is hard to learn and appreciate unless you have some experience with a more "traditional" stack. In the first two projects, therefore, you will start development with a more "traditional" stack, based on MySQL and Apache Tomcat. Hopefully, this will give you enough experience to use the MEAN stack in later projects.

In Project 1, you are asked to perform the following tasks:

1. You will have to set up our project development environment on your machine, which is based on Docker.
2. You will have to brush up with Unix Command-Line Interface (CLI).
3. You will have to brush up with your MySQL knowledge.
4. You will have to brush up with Java programming.

Part A: Setup Your Development Environment

Docker is a popular software development and deployment tool that implements *container technology*. It allows running multiple isolated OS environments on a single machine. As the first task of Project 1, you **must** go over the following tutorial and follow its step-by-step instruction:

- Docker Setup and Basic Use

The page will guide you setting up the Docker App on your machine and teach the most important concepts and commands for using Docker. For the first two projects, we will use the `tomcat` container that we prepared for this class. As described in the Docker tutorial, download the container image and run it on your machine through the following command (if not done yet) replacing `{your_shared_dir}` with your directory of choice, say, `/Users/cho/cs144/`):

```
$ docker run -it -v {your_shared_dir}:/home/cs144/shared -p 8888:8888 --name tomca
```

The container image has MariaDB (a "freer" version of MySQL), JDK, Apache Tomcat, and Python installed. Its underlying operating system is Linux, a variant of Unix. For the most part, therefore, you will be using a Unix shell through a terminal app. If you are not familiar with Unix shell commands, read the Unix Tutorial for Beginners and learn the basic Unix commands. Your username inside the container is "cs144" with password "password".

Part B: Unix Command-Line Interface (CLI)

All of your project development should be done inside a Docker container that we provide, which is based on Linux operating system. For the most part, therefore, you will be using a Unix shell through a terminal app. If you are not familiar with Unix shell commands, read the Unix Tutorial for Beginners and learn the basic Unix commands.

In playing with basic Unix commands, you can use the "tomcat" container created from the "junghoo/tomcat" image in Part A. As we explained in the Docker Setup and Basic Use page, you can start the "tomcat" container using the `docker start` command:

```
$ docker start -i tomcat
```

Your username inside the container is "cs144" with password "password". There are a few important *environment variables* that have been set within the "junghoo/tomcat" container:

- **JAVA_HOME**: The `JAVA_HOME` environment variable points to the JDK installation directory, which is `/usr/lib/jvm/default-java` in our container.
- **CATALINA_HOME**: This variable specifies the location of the Tomcat installation, which is `/usr/share/tomcat9`.
- **CATALINA_BASE**: This variable specifies the base directory of a Tomcat instance, which is `/var/lib/tomcat9`.
- **HOME**: This variable specifies the location of your home directory, which is `/home/cs144`.

You can refer to the values of these variables in your shell, like `$JAVA_HOME`, `$CATALINA_HOME`, etc. For example, if you issue the following `echo` command inside the container

```
$ echo $JAVA_HOME
/usr/lib/jvm/default-java
```

you will see the value of `JAVA_HOME`. To exit from the "tomcat" container, you just need to type `exit`:

```
$ exit
```

Part C: MariaDB Warm-Up

In this part, you get yourself familiar with the basic MySQL (more precisely, MariaDB) commands by creating and loading a table and issuing a few queries.

First, download the actors.csv file to the current directory of the container. To download any file, you can use the `wget` command in the container like the following:

```
$ wget http://oak.cs.ucla.edu/classes/cs144/project1/actors.csv
```

Then, read our basic MySQL tutorial to learn how you can interact with MySQL and issue SQL commands. In particular, the tutorial provides a brief explanation on `CREATE TABLE`, `LOAD DATA` and `SELECT` commands, which will be important to finish this part of the project. If you need to brush up on SQL commands other than the ones explained in the tutorial, you may find the *How Does This RDBMS Thing Work?* section of SQL for Web Nerds helpful.

As we explained in the MySQL tutorial, we have created the database "CS144" (note a database name is case sensitive) in the MySQL of our container. For your project work, use the MySQL user "cs144" (no password), which has full access to the "CS144" database. The MySQL user "root" with password "password" has full unrestricted access to everything and should be used only for special administrative operations, like creating new users and databases, etc.

Now take the following steps to create, load, and query a table in MySQL:

1. Create a table called "Actors" in the database "CS144". The "Actors" table should have the following schema:

   ```
   Actors(name:VARCHAR(40), movie:VARCHAR(80), year:INTEGER, role:VARCHAR(40))
   ```

   Please note that database, table, and attribute names are *case sensitive* in MySQL, so **use the above schema EXACTLY as it is, including the case.**

2. Load the downloaded `./actors.csv` file into the "Actors" table. Make sure that the double quotes enclosing some of the attributes in the data file are removed when they are loaded.

3. Retrieve some loaded data from the "Actors" table. In particular, write a query that returns the answer to this question: "Give me the names of all the actors in the movie 'Die Another Day'."

Create a MySQL batch script file named `actors.sql` that shows every one of the above 3 steps. Again, the MySQL tutorial had a brief explanation on how to create and run MySQL batch script file. Make sure that your script is executable and has no error, meaning that we should be able to run your script by issuing the following command:

```
$ mysql CS144 < actors.sql
```

You **MUST** use "`./actors.csv`" as the location of the data file inside your script. **Do NOT use an absolute path.** You may assume that there is no "Actors" table in the "CS144" database when we execute your script. If needed, use the `--` tag to make comments within your SQL script.

Notes on CR/LF issue: If your host OS is Windows, you need to pay particular attention to how each line of a text file (including your script file) ends. By convention, Windows uses a pair of CR (carriage return) and LF (line feed) characters to terminate lines. On the other hand, Unix (including Linux and Mac OS X) uses only a LF character. Therefore, problems arise when you are feeding a text file generated from a Windows program to a Unix tool (such as `mysql`). Since the end of the line of the input file is different from what the tools expect, you may encounter unexpected behavior from these tools. If you encounter any wired error when you run your script, you may want to run the `dos2unix` command in the container on your Windows-generated text file. This command converts CR and LF at the end of each line in the input file to just LF. Type `dos2unix --help` to learn how to use this command.

Part D: Java Warm-Up

If you are new to Java or if it has been a while since your last Java programming, first read A Crash Course from C++ to Java. This excellent tutorial explains the basics of Java, including how you can name, compile, and run your Java program. (It is okay to skip the parts on BlueJ in the tutorial, since we will not be using it.) If you are quite familiar with Java, but you just want to brush up on minor details quickly, you may want to read slides on Java instead. All basic tools needed for Java programming (e.g., `javac` and `java`) are available on our container.

Now implement a Java program that computes the SHA256 hash over the content of an input file. SHA256 is a cryptographic one-way hash function that computes a 256 bit value from a sequence of bytes. Your Java program should satisfy the following requirements:

1. Your program should be implemented as a single Java class `ComputeSHA`. Your program should take the input filename as the first command line parameter. For example, a user should be able to execute your program like

   ```
   $ java ComputeSHA filename.txt
   ```

   where `filename.txt` is the name of the input file.

2. Given the input file, your program should compute the SHA256 hash value over the entire content of the file and print the computed hash value on screen as follows:

   ```
   addb41d9c1ea67ec0d6cc29dbb2f0248a3f077b7aedf6e5d3405fb462e4b955b
   ```

   Your program should print the above hash value if you provide the sample-input.txt file as its input. Please ensure that the output is formatted exactly as above including its case. Print a newline at the end of the hash value. To test the accuracy of your implementation on other input files, you may compare your output against that of the `sha256sum` command. Please make sure that your program works for both text file and binary file ***of any size***.

In implementing your Java program, you may find the Java class `java.security.MessageDigest` useful, which provides a number of cryptographic hash functions including SHA256 and MD5. If you decide to use `MessageDigest` class, make sure you import `java.security.*` packages in your source code. If what we just said sounds Greek to you, again, read A Crash Course from C++ to Java. To learn how you may use MessageDigest class in Java, try a query like "Java MessageDigest example" on Google and look at the top few results. They are likely to contain good example codes that show you how you can use the class. If you want to learn about basic file I/O in Java, see Java File I/O tutorial.

Notes on editors for Java development: You can choose whatever editors you like for Java development. Options include:

- Unix text editors in the container: You may use any text editor in the container (`vi` and `nano` are available) to edit text files directly.

- Your favorite text editor on the host: You may use your favorite text editor from your host OS (e.g., Sublime Text or Visual Studio Code) and transfer the edited file to the container through the shared directory. Remember that the directory that you specified as the shared directory from the host (e.g., `/Users/cho/cs144` from Mac) is available at `$HOME/shared` in the container. Again, be careful with the CR/LF issue if you use this option.
- If you use Visual Studio Code as your editor, it supports "remote development in a container", which makes it easy to edit and run codes in the container all within Visual Studio Code. Read VS Code document on remote development inside a container to learn how to set it up.

Your Final Submission

Your project must be submitted electronically before the deadline through GradeScope.

What to Submit

The zip file that you submit must be named `project1.zip`, created using a zip compression utility (like using `zip` command in the container). You should submit this single file **project1.zip** that has the following packaging structure.

```
project1.zip
 +- actors.sql
 +- ComputeSHA.java
 +- README.txt (Optionally)
```

Each file or directory is as following:

1. **actors.sql:** A copy of your MySQL batch script file from Part C of this project.
    - You **must** use `./actors.csv` as the location of the data file inside your script.
    - The "Actors" table **must** the provided schema exactly as it is including its case.
2. **ComputeSHA.java:** The source code of you Java program for SHA256 hash computation from Part D.
3. (Optional) **README.txt:** A (plain text) README file containing anything else you find worth mentioning.

Please ensure that your submission is packaged correctly with all required files. Make sure that each file is correctly named (including its case) and `project1.zip` contains all files directly, not within a subdirectory. In other words, unzipping project1.zip should produce the files in the same directory as `project1.zip`. **You may get as small as zero points for your work if the grader encounters an error due to incorrect packaging, missing files, and failure to follow our exact spec.**

To help you package your submission zip file, we prepared "a packaging script" p1_package. After downloading the script in the directory that has your Project 1 file, set its executable permission and run it as follows:

```
$ wget http://oak.cs.ucla.edu/classes/cs144/project1/p1_package
$ chmod +x ./p1_package
$ ./p1_package
zip project1.zip create.sql load.sql
  adding: create.sql (deflated 63%)
  adding: load.sql (deflated 81%)
[SUCCESS] Created '/home/cs144/p1/project1.zip'
```

When executed, our packaging script will collect all necessary (and optional) files located in the current directory and create the `project1.zip` file according to our specification that can be submitted to GradeScope.