UGA CSCI4795/6795: Cloud Computing (Spring 2019)

Programming Assignment 1 (PA#2) (out: Feb 13 2019; due: Feb 23 2019 – 11:59 p.m.)

Goals

Gain hands-on experience with Linux containers, Docker, and Elastic Beanstalk.

Introduction

In this assignment, we will study what you can do (and what you give up) with computation in the cloud that is not based on IaaS or PaaS. Specifically, in this assignment, we will investigate containers and Docker. (Note that we will be using AWS' broader PaaS support called Elastic Beanstalk to run our Docker containers. This is confusing but using Elastic Beanstalk this way is really more "Docker" than "PaaS".)

In this assignment, you will design and implement a cloud-based, scalable auto-grader system for introductory programming courses (think of how MOOC's might grade their Intro programming assignments). The particular assignment we will auto-grade is: *Write a C++ program called 'walk.cc' that prompts the user for the names of two people, and then prints to the screen a statement that says that these two people went on a walk together.*

You are required to do this assignment alone.

Part 1: Getting familiar with Docker

I know you will not review my class slide before the midterm exam 😉 Let's start by going through this excellent tutorial to get a feel for Docker: https://docker-curriculum.com/ Before starting this tutorial, set up a decent environment in AWS:

1. Start a t2 medium Ubuntu Server 14.04 LTS (**NEIGHTER** Ubuntu Server 16.04 LTS **NOR** 18.04 LTS!) (change disk size to **20GB**) [Note: you must scroll down to the very end of the "choose an Amazon Machine Image (AMI)" screen]
2. Make sure that the AWS security group has SSH open (port 22) from "anywhere"
3. Attempt to SSH onto the new VM (note that this might take a few min, while the VM boots)
    a. **Mac**: Use "Terminal", then something like "ssh ec2-54-84-24-252.compute-1.amazonaws.com –i *your_private_key_name.pem* -l ubuntu" (accept the public key, you will not be prompted for password)
    b. **Windows**:
        i. You will need to convert your PEM-format private key – get PuttyGen from here: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html ("Load" your PEM; "Save private key")
        ii. Download Putty from there and run it
            1. Host name is from EC2 console
            2. Select "Connection" and make "Seconds between keepalives" 120
            3. Expand SSH and then select Auth, hit "browse" to navigate to the private key file
            4. When Putty connects, accept the public key
            5. Log on via "ubuntu" (you will not be prompted for password)

4. Once on the VM, install Docker via: `curl -fsSL https://get.docker.com/ | sudo sh` (cut and past this)
5. Then execute: `sudo usermod -aG docker $(whoami)` Then log out and log back in.

Then start the tutorial (the first thing you would type into your Ubuntu terminal would be "docker run hello-world"). I recommend that you do the tutorial through the end of step "Docker on AWS" in Webapps with Docker. A few notes:

- Early in the tutorial, when you do "python --version", if it says something about "not installed, choose either 'minimal-python' or 'python3'" then **you're using the wrong version of Ubuntu** (kill the VM and start again with the right version). Note that if it says "Python 2.7.6", you're good. (no need to try to get 2.7.11 on your VM)
- Do what Ubuntu says to install pip (`sudo apt-get install python-pip`). Also, no need to install Java (we will not be doing this part of the tutorial)
- Re: "You can open http://localhost:32769 in your browser." You're running on the VM, and you're attempting to connect via browser running on your laptop, so [a] "localhost" won't work, and [b] 32769 is not open by default. You'll need to figure out how to address both of these concerns.
- In Section "Our First Image" in Webapps with Docker, make sure that you do not ignore the line "If you haven't already, please go ahead and clone the repository locally." (Look at the hyperlink on "Flask app" right above it)
- You many want to install Flask on your VM by "`sudo pip install Flask==0.10.1`"

Part 2: Auto-grader basic system

Now it's time to adapt what you've just learned, to create a new system: Now create a Web-server-based system that makes the student submit their 'walk.cc' program, you auto-grade it, and provide the score and feedback through the browser. An auto-grader typically first confirms the program compiles; if it does, then the student submission is executed under a controlled environment and compared against known output. In this CSCI4795 assignment PA2, you are required to autograde this particular "CS1 assignment" identified above. Below, I give you a decent "compile/execute" script (which tends to be generic) and a "compare against known output" script (which tends to be specific to the assignment); you can modify or replace these two scripts as you wish/desire.

Here is an example program that might be submitted:

```
#include <iostream>
using std::cin;
using std::cout;
#include <string>
using std::string;

int main()
{
 string name1,name2;
 std::cout << "Enter a name:";
 getline(std::cin,name1);
 std::cout << "Enter a name:";
 getline(std::cin,name2);
 std::cout << name1 + " and " + name2 + " went for a walk.\n" ;
}
```

<u>Compile/execute script</u>

```
#!/usr/bin/env python
import os
import subprocess

subprocess.call("rm -f ./a.out", shell=True)
retcode = subprocess.call("/usr/bin/g++ uploads/walk.cc", shell=True)
if retcode:
 print("failed to compile walk.cc")
 exit

subprocess.call("rm -f ./output", shell=True)
retcode = subprocess.call("./test.sh", shell=True)

print ("Score: " + str(retcode) + " out of 2 correct.")

print("*************Original submission*************")
with open('uploads/walk.cc','r') as fs:
 print(fs.read())
```

<u>walk.cc test script</u>

```
#!/bin/bash

tmpoutput=`echo -e freddy '\n' susan | ./a.out`

CORRECT=0

f1=`echo $tmpoutput | grep -q 'freddy'`
if [ $? = 0 ]; then
 let CORRECT=CORRECT+1
fi

f1=`echo $tmpoutput | grep -q 'susan'`
if [ $? = 0 ]; then
 let CORRECT=CORRECT+1
fi
exit $CORRECT
```

This code above is the “domain-specific functionality”, which must be now placed into a Web framework (in this part, which is Part 2) and a Docker framework (in the next part, which is Part 3). Are you required to create this Web/Docker code from scratch? No. Rather, as a result of performed the docker-curriculum tutorial, you have a code base that you can use as the starting point. What functionality is missing from this codebase that we have via the tutorial? For Part 2 of this assignment, essentially three things:

1. You must add code that allows the upload of “walk.cc”
2. You must add code that -- once walk.cc is uploaded -- executes the “Compile/execute script” from above
3. You must add code to show the results back to the user.

Two potentially useful URLs that could be helpful – allowing you to add these three functionalities within the Python/Flask code from the tutorial:

- https://www.tutorialspoint.com/flask/flask_file_uploading.htm
- https://docs.python.org/2/library/subprocess.html

Note that you are REQUIRED to use Python and Flask; you are NOT required to use code from these particular two URLs if you do not want to. Upon completing this section, **see the "What to submit" document (Q1) before you continue to Part 3.**

Part 3: Docker-based Auto-Grader

Once again applying what you learned from the https://docker-curriculum.com/ tutorial, turn the non-Docker system from Part 2 into a Docker-based system. In Part 3, it is expected that Docker will be run on the single VM that you're using for this assignment.

See the "What to submit (Q2)" document before you continue to Part 4.

Part 4: Docker in AWS: Elastic Beanstalk

We will now run the Docker container on AWS and will use docker hub (https://hub.docker.com/) for a Docker image repo.

1. Create your account on docker hub and do "docker login" on your VM.
2. Build your own docker image using "docker build -t *your-id-for-docker-hub*/autograder ."
3. Push it to your docker repo in the docker hub.

Now that we have the docker image in the docker hub and we're going to run it via AWS Elastic Beanstalk:

4. Select Elastic Beanstalk from the AWS Console (under "Compute"). Select "create new application" in the upper right. **Note that do not click "Get Started" in the middle of the page!**
5. Make the "Application name" be "autograde" and hit "Next".
6. On the next screen, click "Create one now", choose "Web server environment" and hit the select button.
7. On the next screen, input the "Domain" by adding your **UGA ID** to the front of "autograde-pa2" (e.g., "ik32805-autograde-pa2" for domain and hit "Check availability" If your domain is **unavailable**, add few more characters after your previous domain.
8. Go down to "Base configuration." Select "Preconfigured platform" then "Docker."
9. For application code, check "upload your code". You must then create a file on your laptop and name this file "autograder_beanstalk_config.txt" You can make this file based on Dockerrun.aws.json from the tutorial. Then upload that file and hit "Create environment"
10. Wait for 5 to 10 minutes and you will find URL for your docker application (e.g., http://ik32802-autograde-pa2.us-east-1.elasticbeanstalk.com). Sometimes, you need to refresh the page. If everything is finished well, the page will move to an overview page of your application. Now you can access your container application! **See the "What to submit (Q3)" document**

That’s it!

If your first deployment does not work (e.g., application logic error and/or application configuration error), then I suggest deleting everything in Elastic Beanstalk before trying again (In Elastic Beanstalk, after selecting your allocation, do Actions → Terminate Environment and Delete application) See the “What to submit” document before deleting all services in AWS.

<u>After finishing “What to submit” document, delete AWS beanstalk application after terminating environment.</u>