CS 455-555: Distributed Systems

Homework 4 (50 points)

Due Date –On Class Website–

DevOps is the practice of operations and development engineers participating together in the entire service life cycle, from design through the development process to production support.

DevOps is also characterized by operations staff making use of many of the same techniques as developers for their systems work.

`http://theagileadmin.com/what-is-devops/`

Objectives

- Learn to use Docker, a light-weight container that is useful for distributed applications.
- Use the time server and client code provided in your git repo `hw4` folder to demonstrate the use of Docker. The server code is the same as in `examples/sockets/tcp/multithreaded/` in the CS455-resources git repo.
- Experience a few more morsels of DevOps! Or is this Brain Candy for DevOps?

Setup

Do a `git pull -rebase` in your individual backpack git repo to get the starter files for the homework.

Recipe

Step 0: Start up your Linux system (use a VM or create one if needed) Use you own Linux (on bare metal or in a VM) system for this homework as you will need root access. Note that if you are running a custom Linux kernel, docker will probably not work. Just reboot into a standard kernel! If you don't have a VM ready, then install one using VMware virtualization software that you should have used in earlier classes.

Step 1: Install Docker in your Linux VM Follow the *Getting Started* instructions on the Docker website to install Fedora (or Ubuntu) Linux Community CE docker and work through the tutorial.

Step 2 After completing the hello-world tutorial, run the official Fedora (or Ubuntu) Linux docker image as follows.

```
sudo docker run -it fedora bash
```

That will download the image for you and run it in an interactive mode so you can play with it. The image provided is intentionally minimal. However, you can install packages in it as a separate system from the host system. For example (in the docker instance), try:

`dnf install net-tools`

Replace `dnf` with `apt-get` for installing on Ubuntu.

You can now run commands, like `ifconfig` to see what network address it was assigned. By the way, you can also do that outside of the container using the command:

`sudo docker network inspect bridge`

Step 3: Build a custom image for TimeServer (10 points) Create your custom image (from the official CentOS Linux image) by using a `Dockerfile`. The documentation for creating `Dockerfile` is here:

`https://docs.docker.com/develop/develop-images/dockerfile_best-practices/`

Set up the image to run the TimeServer on startup. For example, you may want to pre-install `java`, `java-devel`, `net-tools` and `git`. You can then clone the `CS455-resources` examples repo into the image, compile the server code and have it ready to go in the image. Then you simply will run the TimeServer when the docker image runs. (*Hint: Lookup the CMD option to Dockerfile*).

Once you have the `Dockerfile` ready, you can create the image with the following command in the same folder as your `Dockerfile`:

`sudo docker build -t <YourDockerImageName> .`

You can list your images with the following command:

`sudo docker image ls`

- It is possible to copy files as well as to mount host file systems into a docker instance but the assignment asks you to stage the image as above to make a point of how images can be "cooked" from scratch and then deployed anywhere.

Step 4a: Test the docker TimeServer with a TimeClient (5 points) Run the docker Timeserver with the following command:

`sudo docker run <YourDockerImageName>`

Run the TimeClient on your localhost and connect to the TimeServer running in Docker using its IP address and port number. See above on how to get the address. The TimeServer program has been updated to print the network interface address for your convenience as well.

Step 4b: Test a second copy of TimeServer running at the same time Run a second copy of the TimeServer on another Docker instance. Test to see if the client can connect to this instance as well. Take a screenshot showing the client connecting to the two servers.

Step 5: Modify TimeClient to fail over to second server when first one fails (10 points) **Required**: Modify the TimeClient code so that it if fails to connect to the first server, it automatically tries the second server. You would modify the code so that it accepts a list of IP addresses for multiple servers and keeps a list internally to try. Use a Java collection such as *LinkedList* to track the hosts.

```
[amit@fedora hw4]$ java TimeClient
Usage: java TimeClient <port> <serverhost> [<serverhost>...]
```

Test this functionality. For example, you can stop the first docker server, so the client is forced to go to the second one. Note that socket timeouts are very long, so you use the `setSoTimeout` method to set it something shorter. You can stop a running instance with the following command:

```
sudo docker stop <instance name>
```

Or you can kill the docker instance with the command:

```
sudo docker kill <instance name>
```

You can get a list of the running instances with the command:

```
sudo docker ps
```

Take a screenshot showing how you stopped the first server and then ran the client with the two server IPs as argument and it was able to find the second server.

Required Files

- Please place all your files in the `hw4` folder in your git repo.
- Please use the class name TimeServer for the server class for your program and TimeClient for the client class. Please submit the source code for the client class. Note that the TimeServer class will be pulled from the git repo and be part of your docker image so there is no need to submit it separately.
- Please include your `Dockerfile` as well as screenshot(s) showing your two instances of TimeServer servers running and the client connecting to one and then to another server (when the first server gets stopped/killed).
- Include a brief `README.md` that contains your name, class number and section, homework #, and your observations on the homework.

Submitting the Homework

Homework is individual so we will use your individual github classroom repository. To obtain credit, submit your homework through classroom github as described below.

Change to the directory called hw4 in your classroom individual repo and place all required files in that folder. Then do the following steps (on your master branch):

- git add [The appropriate files]
- git commit -m "Homework hw4 complete"
- git push origin master