Project 3: Docker Cluster Setup
CSE 291 Spring 2016

Assigned: Tuesday, 29 March
Due: Tuesday, 5 April

Overview

This quarter we will be using Docker containers to enable you to simulate a distributed system on your laptop or home computer. Ubuntu Docker containers will serve as the nodes. They will be networked via the Docker bridge network. And a Docker data volume container will be used to enable the nodes to share data, simulting a distributed file system.

This lab is designed to help you to bootstrap that process. You will (i) complete the Docker tutorial, (ii) create Dockerfiles to describe a node, (ii) develop a really simple client and server, (iii) modify the Dockerfile into two files, one for a client and one for a server, (iv) demonstrate that your client andserver can interact.

Individual Project – No Partners, No Teams

We need to make sure everyone can use the infrastructure, so this is an individual assignment.

Docker Installation and Tutorial

The following page contains installation instructions: https://docs.docker.com/windows/

The following page contains the tutorial, please complete it up to, and including, the section titled, "Learning by doing: Network configuration": https://docs.docker.com/engine/userguide/containers/dockerizing/

Basic Configuration

Use a Dockerfile to define an Ubuntu image with whatever development tools you need. You will eventually create two derivatives of this, one for the client and one for the server.

Using the same Ubuntu image, but without adding the tools, create a data volume container. This data volume container will be mounted at "/data" by your other containers to simulate a distributed file system.

The volume should contain a single file name "string.txt", which will be used by each of your client and server. It should be placed into the data volume with either an ADD or a COPY.

At this point, you should be able to instantiate two of the client/server containers that mount the volume from the data containers. Connect to each image, verify that you can "cat" the data file from within each image, find each image's IP address (ifconfig), and "ping" each image from the other image. Write a "hello world" program using your choice of languages and test it.

At this point you know that you have created the basic containers, installed your language of choice, successfully mounted the data volume to simulate a DFS, successfully populated the data volume, and can use the network between them. You now have a testbed for writing distributed systems (and simple network code)!

Network client and server

You should write a simple network server, called "catserver". When it receives "LINE" followed by a new line, it should reply with the next line of the file, followed by a new line – except that the reply should be in all UPPERCASE. With each call, it should work its way down the file. If it gets to the bottom, it should start over at the top. The path to the text file should be the first command-line argument. The port number should be the second, e.g. "catserver /var/datavol/sample.txt 2000"

You should write a client called, "catclient" that queries a "catserver" every 3 seconds for 30 seconds. Upon receiving each response, it should check to see if the response is in a text file. If it is, it should echo "OK". If it isn't, it should echo "MISSING", in either case followed by a newline. The path to the text file should be the first argument. The server's port number should be the second.

Testing

You should test everything by starting a server, then starting the client, and verifying in an automated way that the output is correct. The client and server should share a common data file via a shared data container. Don't forget to daemonize the server. Similarly, remember you'll need to use "docker logs" to see the output of the client.

Deliverables

You will turn in a .zip of the directory contaning your code, the Docker files that build the client and server images from scratch, your text file, and a script that runs the demo. The script can be in whatever language is native to your operating system, e.g. VBA, shell script, etc.