Homework 1 – A scalable environment

Lab Information

Due Date:

Homework 1 Dropbox Deadline

Coordination:

This homework is to be completed alone

Objectives/Goal:

To understand popular incarnations of interception technologies that deal with web applications.

You have learned about various web technologies this week. Your goal is to set up an environment that is designed to scale with a large amount of load. As you design this, consider how each of these components interacts with HTTP Requests. Your goal will be to set up an environment with 3 different types of components: A Load Balancer, A caching server, and at least two Web Servers.

Deliverables:

• A zipped file that contains a Dockerfile/compose.yaml (and associated files) [1 for each activity]

• A breakdown in support of your choices.

Table of Contents:

Lab Information 1

Activity 1: Webserver Setup: 2

Activity 2: Caching Server Setup: 2

Activity 3: Load Balancer Setup 3

Activity 4: Proxy Scanning 4

Writeup 4

Deliverables 6

Activity 1: Webserver Setup:

Your first objective, if you choose to accept it, is to set up a web server. While Apache has the highest market share there are a number of possible, mature, alternatives. Webservers are simply servers that will respond to HTTP requests, although as we have gone over in class there are other supported protocols in many cases.

Step 1: Choose a webserver

Research available webservers and choose one which you think best suits the need of the remainder of the lab. Generally, you are given the choice of the following:

• Apache

• Nginx

• IIS

You may make, or choose a different technology with professor approval, however, the aforementioned technologies may be used without additional approval.

Step 2: setup a webserver

Now that you’ve made a decision on what platform you will use, it is time for you to install that webserver and configure it to serve a basic page. The configuration will vary based on which platform and OS you choose. Once you have the web server installed setup a basic text-based page that simply displays “Hello World” as part of the body tag. Your page’s source must be verified as HTML 5 compliant using the W3C validator (https://validator.w3.org). This will be checked in your submitted Docker file.

**This webserver should be submitted as a Dockerfile.** Docker is supported on Windows, MacOS, and Linux. If you are unfamiliar with Docker, you might check out this tutorial: https://www.digitalocean.com/community/tutorials/docker-explained-using-dockerfiles-to-automate-building-of-images

Activity 2: Caching Server Setup:

Caching servers are important staples of computing security. At each layer where there are speed changes, caches can be used to mitigate the change if the right algorithm is used to take advantage of locality. Your second activity is to use such a technology to speed up your environment.

Step 1: Choose a caching server

Research available caching servers and choose one which you think best suits the need of the remainder of the lab. Generally, you are given the choice of the following:

• Varnish

• Nginx

• Squid

You may make, or choose a different technology with professor approval, however, the aforementioned technologies may be used without additional approval.

Step 2: setup a caching server

Now that you’ve made a decision on what platform you will use, you should install it. You should come up with a simple method to verify that the caching server is, in fact, working as this will be tested in your submitted work.

Hint: this may involve changing the page and demonstrating that the cache still displays on the web browser.

**You should create a new Docker image when creating your Caching Server. This Caching Server Docker image you’ve made should work in conjunction with the Web Server Docker image you made in the previous step.** To do this you should use a Docker Compose. Docker Compose is a tool for setting up multi-container environments. on Windows, MacOS, and Linux. If you are unfamiliar with Docker Compose, you might check out this tutorial: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-14-04

Activity 3: Load Balancer Setup

Now we are going to complicate things. Your task is going to be to do some research and determine the best location for a load balancer within the environment you already created. This will, in general, require you to add another webserver.

Step 1: Choose a load balancer

Research available caching servers and choose one which you think best suits the need of the remainder of the lab. Generally, you are given the choice of the following:

• Nginx

• HAProxy

• Varnish

• Apache

• Zen Load Balancer

You may make, or choose a different technology with professor approval, however, the aforementioned technologies may be used without additional approval.

Step 2: setup a load balancer

Now that you’ve chosen a load balancer, determine where you’ll put it. Once you’ve figured out where it goes in your topology (web server and caching server so far), you might find you need another webserver – deploy that as needed. Finally, deploy your load balancer and demonstrate it is functional by having it serve [slightly] different versions of the same page. Instead of saying ‘Hello World’, have the body of the other server say ‘Hello World2’.

**You should create a new Docker image when creating your Load Balancer. This Load Balancer Docker image you’ve made should work in conjunction with the Web Server Docker image you made in the first step. You’ll need at least two copies of the webserver.**

Activity 4: Proxy Scanning

As we discussed in class HTTP has built-in proxy capabilities. Often these are capabilities are disabled, but if you can find a webserver that supports HTTP proxying, it can be very useful for surfing the web anonymously. In this exercise, you will be using an existing library to try and find these open proxies.

Step 1: Using the requests library (or equivalent).

If you’ve never used Python requests now would be a great time to become familiar with it. The library makes it very easy to make standard HTTP requests to a site without having to know much about the underlying protocol. In the remainder of the course, we will be creating a similar library. Use the requests library to request csec.rit.edu.

**Submit this step as a ZIP that contains a Python Script named act4ste1.py and a requirements.txt that outlines which libraries your script requires.**

Step 2: Write a scanner that will try to find an anonymous HTTP proxy

Now that you have a basic HTTP request script running, all you’re going to need to do is slightly expand it. You should write this script such that it can take in a range of IP’s and scan each one looking for an open HTTP proxy.

We talked briefly about how these requests look but you will only need to know how to use Python Requests (or similar) in order to generate it (hint: no need for a raw HTTP packet). Once you have it correctly generating scan ranges of the internet until you find an open proxy server. Your script should return the IP addresses (one per line) of the Proxies discovered (Note: a proxy server is distinct from a web server).

Hint: You may wish to leverage threading to speed this up.

**Submit this step as a ZIP that contains a Python Script named act4ste2.py and a requirements.txt that outlines which libraries your script requires. Act4ste2.py should take two command line arguments a start IP and an end IP.**

Writeup

As you go through the remainder of your career you will often be faced with many choices. In this case, you were given a set of options, all of which were free – but there are also many situations where the cost will factor in. Given the above options, how did you select which technologies you would use? Do some of these work better in conjunction? Are some easier to use or deploy?

Also, attach your topology and discuss why you chose this topology and what the alternatives might have been. Lastly, include a discussion about why these technologies might be needed. At the same time, you may wish to discuss why the single text web page example we just created might not need all the complexity we developed.

Deliverables

Activity 1 – Docker/Compose files for a functional webserver (20%)

Activity 2 – Docker/compose files for a Caching Server as part of Docker Compose environment (20%)

Activity 3 – Docker/compose files for a Load Balancer as part of Docker Compose environment (20%)

Activity 4 – Two different Python scripts – one from Step 1 and one from Step 2 (part 1 – 10%, part 2 – 20%)

Writeup (10%)