CS 524 Lab Assignment #2

In this assignment, you will learn to develop and load-balance your own infrastructure (a server farm) while applying your knowledge of DNS and other application entities and protocols.

You will also learn to use Cloud Storage for back-up and recovery of your data.

In addition, you will perform rudimentary analysis of the IP traffic.

You can use *Nginx* (a popular *http* server software available at http://nginx.org/), or any other available server software, to host a simple website on four Amazon EC2 instances; you will also configure an *Nginx* server on another instance, which will act as a load balancer. You will learn how to distribute networking workload across multiple servers.

A total of **100** points will be given to you if you successfully implement the outlined steps. In addition, at the discretion of course assistant, you can be given extra points (up to the maximum of 5**0**) for devising and programming extra features.

As before, remember to double check the Amazon SLAs and ensure that you take all the necessary steps not to exceed the resource.

Please take the following steps:

1) Create the Amazon EC2 instances

You need to create *five* EC2 instances of the same type you have created in the previous Lab. One of these instances will act as a load balancer; the other four will act as web servers named Server1, Server2, Server3, Server4, as demonstrated below:

Incoming traffic

|

```
                    Load Balancer

                          |

          ________________________________

         |          |          |          |

      Server1    Server2    Server3    Server4
```

2) Install *Nginx* (or whatever other server that you use) on each instance

After launching the instances, use *amazon-linux-extras* (the Amazon Linux native software manager) to install Nginx on every instance and start the Nginx service.

To verify the *Nginx* is working, visit the hosted default webpage through instance's public DNS from an internet browser. You will see the **Welcome message** (if *Nginx* http server works properly).

You need to edit the default *index.html* file on four servers. This file can be found in the directory */usr/share/nginx/html*.

Use a text editor (such as *vi*) to Add the header with server ID in the index.html file like following:

```
<h1>[SERVER_ID]</h1>
```

Of course, you need to change SERVER_ID to the particular server name (i.e., Server1, Server2, Server3, or Server4).

3) Configure the load balancer

To configure the load balancer, you need to edit the load balancer’s configuration file */etc/nginx/nginx.conf.*

Use a text editor (such as *vi*) to replace the existing text with the following:

```
events {

 worker_connections 768;

}

http {

 upstream myapp {

 #ip_hash;
 server [SERVER_PUBLIC_DNS_NAME] weight=1;

 server [SERVER_PUBLIC_DNS_NAME] weight=1;

 server [SERVER_PUBLIC_DNS_NAME] weight=1;

 server [SERVER_PUBLIC_DNS_NAME] weight=1;

 }

 server {

 listen 80;

 server_name myapp.com;

 location / {

 proxy_pass http://myapp;

         }

     }
```

Again, you need to find the public DNS name of your servers and replace each occurrence of the *SERVER_PUBLIC_DNS_NAME* with the respective string.

The assignment *server[<name>] weight=<number>* establishes the weight of the server in the upstream cluster to be equal to the specified *<number>.* The present default is 1. To experiment with assigning some servers a greater proportion of the traffic, set weights correspondingly.

Restart the nginx service

You also need to replace the SERVER_PUBLIC_DNS part with the public DNS of your instances.

Now you can use the *curl* command in the shell to visit the balancer, which will distribute traffic among the servers.

```
$ curl [LOAD_BALANCER_DNS_NAME]
```

4) Collect the information on visits to your site

You can use the *visit server* tool, provided in the Appendix, or write your own tool, to track the distribution of the load. The tool visits the cluster 2000 times and returns the visit count on each server.

The following example illustrates the use of this tool:

```
$ visit_server.rb -d LOAD_BALANCER_DNS_NAME
```

The output would look like this:

```
Starting to visit load balancing server

-------------------------

  Summary

-------------------------

Server1 visit counts : 500

Server2 visit counts : 500

Server3 visit counts : 500

Server4 visit counts : 500

Total visit counts : 2000
```

The following table shows three different scenarios of weight combination of the four servers. Configure the nginx.conf file according to these scenarios. Then use visit_server for each scenario and record the outputs.

Weight / Server / Scenario	Server 1	Server 2	Server 3	Server 4
1	1	1	1	1
2	1	2	3	4
3	1	2	1	2

Submission

Your submission should include a report that includes

1) the description of the steps that you have done in this assignment.

2) the results of the execution of the three scenarios

3) all additional observations.

Appendix: The *Visit Server* program

```
#!/usr/bin/env ruby

#

# This program is used for collecting web server visit information.

#

# Author: A. Genius

#

require 'optparse'

def print_usage

 puts "USAGE: visit_server -d DNS_NAME"

 exit

end

# add option switch and handler

options = {}

option_parser = OptionParser.new do |opts|


 # DNS_NAME argument

 options[:dns_name] = nil

 opts.on('-d', '--dns-name DNS_NAME', 'Specify a DNS NAME') { |dns_name|

options[:dns_name] = dns_name }

# HELP argument

 options[:help] = nil

 opts.on('-h', '--help', 'Display usage') { |help| options[:help] = help }

end
```

```
option_parser.parse!

# verify arguments

if options[:dns_name] then

 dns_name = options[:dns_name]

else

 puts "Please set a balancer's DNS."

 print_usage

 exit

end

if options[:help] then

 print_usage

 exit

end

# Keep STDOUT

#orig_stdout = $stdout

# redirect stdout to /dev/null

#$stdout = File.new('/dev/null', 'w')

server1_visit_count = 0

server2_visit_count = 0

server3_visit_count = 0

server4_visit_count = 0

# starting to visit load balancing server

puts "Starting to visit load balancing server"

2000.times do
```

```
# visit load balancer

#o = `curl #{dns_name}`

o = `curl -s #{dns_name}`

if o =~ /server\s*1/i

server1_visit_count += 1

elsif o =~ /server\s*2/i

server2_visit_count += 1

elsif o =~ /server\s*3/i

server3_visit_count += 1

elsif o =~ /server\s*4/i

server4_visit_count += 1

end

print "."

end

# redirect output to stdout

#$stdout = orig_stdout

# print visit information

puts

puts '-------------------------'

puts ' Summary'

puts '-------------------------'

puts "Server1 visit counts : " + server1_visit_count.to_s

puts "Server2 visit counts : " + server2_visit_count.to_s

puts "Server3 visit counts : " + server3_visit_count.to_s
```

puts "Server4 visit counts : " + server4_visit_count.to_s

puts "Total visit counts : " + (server1_visit_count + server2_visit_count + server3_visit_count +

server4_visit_count).to_s

Additional steps are outlined below.

Create EC2 by using command line

1. Use the `script` command to record all the commands you use to create the instance
2. Explain every step you have used to achieve your goal

After having deployed the balancer

1. Use the `tcpdump` command to collect all the packets that had been exchanged.
2. Analyze the packets and report your observations.

As an additional step, perform the EC2 backup and restore:

1. Register an AMI with your Load Balancer instance image and launch a new EC2 instance with that.
2. Verify that the new instance has all the files created in the load balancer instance. List and explain all the steps in achieving this goal.

Please provide screenshots to document your report.