**Homework 7 Solution Template: Load Balancers and Autoscaling (Total Points: 100)**

Due: Sunday October 28 11:59PM

1. **Build an Application Load Balancer (Points: 60)**

**Build the AWS Load Balancer corresponding to the diagram. Follow the steps below to do that:**

* 1. Create two EC2 instances that contain a web server and index.html in /var/www/html/
  2. Create two EC2 instances that contain a web server and index.html in var/www/html/alt/

Paste the screenshot of the list of your 4 running instances:

|  |
| --- |
|  |

* 1. Set up two Target Groups
     1. Target Group 1 holds your first two web servers. All traffic is routed here by default.

Paste the screenshot of the Target Group 1 with Targets tab visible:

|  |
| --- |
|  |

* + 1. Target Group 2 holds your second two web servers. All traffic with “/alt” in its url will be routed here.

Paste the screenshot of the Target Group 2 with Targets tab visible:

|  |
| --- |
|  |

* 1. Set up your load balancer named “YourLastNameLB” to distribute work to the two Target Groups that you created. Http traffic should be routed to Group 1 by default, but all requests with “/alt/” in URL should be routed to Group 2.

Paste the screenshot of the Load Balancer with Description tab visible:

|  |
| --- |
|  |

Go to Listeners Tab and click on View/Edit rules - paste the screenshot showing your rules:

|  |
| --- |
|  |

* 1. Check that traffic is routed correctly based on your URLs. Traffic sent to /alt should reach your /alt servers, and all other traffic should reach your other servers.

Paste the LB URL pointing to index.html in Group 1:

For example: http://your-load-balancer-dns/index.html

<http://okonechnikovlb-1708000264.us-east-1.elb.amazonaws.com/index.html>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Paste the LB URL pointing to index.html in Group 2:

For example: <http://your-load-balancer-dns/alt/index.html>

<http://okonechnikovlb-1708000264.us-east-1.elb.amazonaws.com/alt/index.html>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

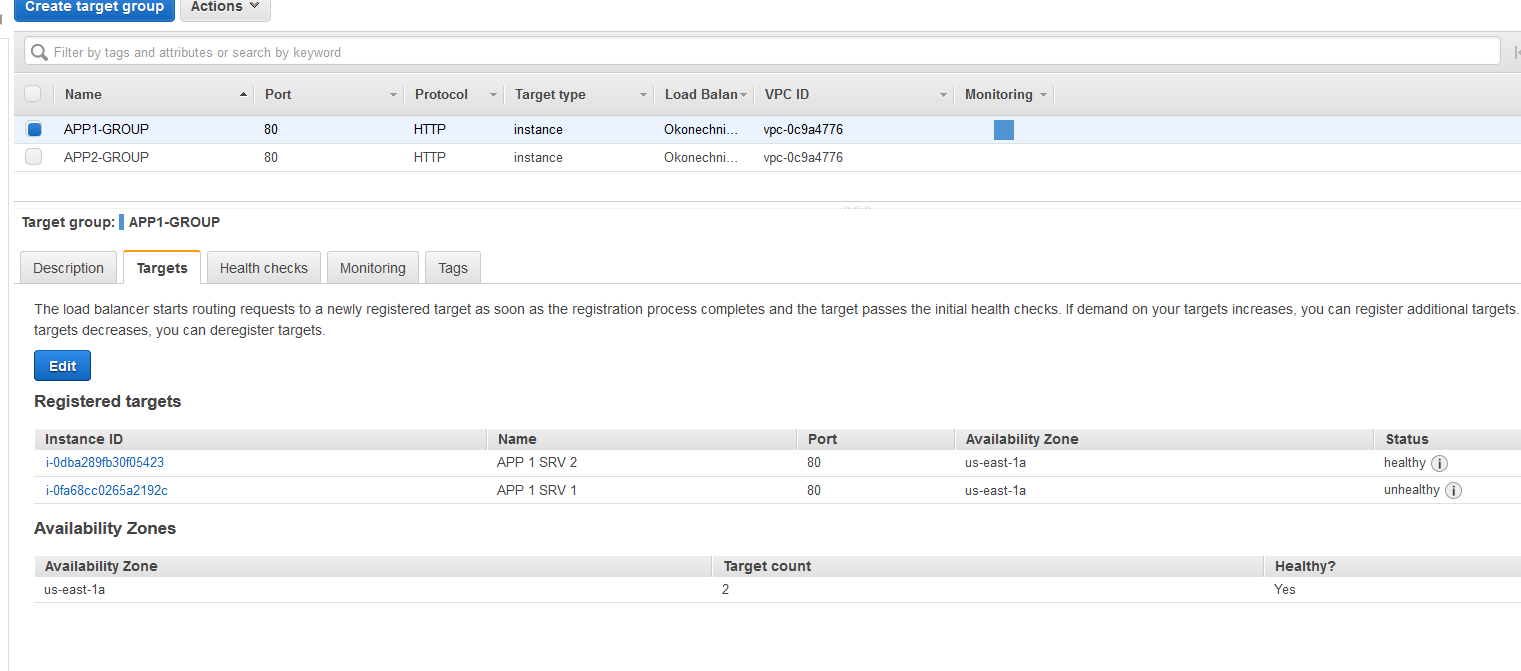
When you reload the above URLs multiple times what content do you displayed to you? Briefly describe what you see and explain why you see that.

In both the alt and non alt paths, we alternate the instances hit. This is the load balancer distributing requests between the instances, so each receives roughly 50% of the traffic. In a normal setup, the content served by each instance would be the same, and the user wouldn’t see a difference.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. Remove the healthcheck.html from one server. Demonstrate that when you try to hit that pair of servers, the server that is missing its healthcheck is not longer receiving any traffic.

Briefly describe what you see reloading URLs after having removed the healthcheck.html file from one of the servers:

After removing the health check file from instance 1 and waiting the requisite time for it to register as unhealthy, the load balancer will stop forwarding to it. Instead, the load will be fully placed on the only remaining healthy instance.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Build an Autoscaling Group (Points: 40)**
   1. Create Auto-Scaling Group with the following specifications:
      1. Create Launch Configuration with Enable CloudWatch detailed monitoring activated, using the following public AMI:

**Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type** - ami-0ff8a91507f77f867

Paste the screenshot of the Launch Configuration with Details tab visible:

|  |
| --- |
|  |

* + 1. Create Auto-Scaling group with Launch Configuration created in the previous step.

Paste the screenshot of the Scaling Group with **Details** tab visible:

|  |
| --- |
|  |

Paste the screenshot of the Scaling Group with **Scaling Policies** tab visible:

|  |
| --- |
|  |

* 1. Once the autoscaling group is created, proceed to “Instances” tab of the group.
     1. Wait for the instance to be created.

Paste the screenshot of the Scaling Group with **Instances** tab visible:

|  |
| --- |
|  |

* + 1. Demonstrate that your auto-scaling group works. To do that go to the Scaling Policies tab of your Scaling group and execute your Increase Group Size policy with alert value greater that your threshold, let’s say 80%. Within a minute you should see that another EC2 instance is being automatically started in the scaling group.

Again, paste the screenshot of the Scaling Group with **Instances** tab visible (it should have at least 2 instances in the list by now):

|  |
| --- |
|  |

1. **Bonus. Simulate High CPU load to make Auto Scaling work by itself.   
   (Bonus Points: 15)**
   1. Download CPULoad.jar to your PC and then copy it from your PC to the instance in scaling group using scp command:  
        
      scp -i your\_key.pem CPULoad.jar ec2-user@your\_instance\_dns:~/.
   2. Connect to the instance using ssh. Verify that CPUload.jar file is in your current home directory on the EC2 instance
   3. Run the CPULoad.jar using the following command:  
        
      java -jar CPULoad.jar  
        
      You will see message:  
      *Running CPU load simulation. Terminate with Ctrl-C when done testing…*
   4. Go to your AWS Console and observe the behavior of the instances in the created Auto-scaling group. It will take a few minutes for the changes to start happening. Keep refreshing the screen. Describe what you observe. Do you see that the number of instances is increasing? Why? What would you need to do to make the number of instances decrease back to 1?
   5. **Alternatively!** You can use your own way to incur high CPU usage on your initial EC2 instance instead of using the provided CPULoad.jar utility.

Paste the screenshot of the Scaling Group with **Instances** tab visible (it should have at least 2 instances in the list by now):

|  |
| --- |
|  |

In AWS Console open the page describing the EC2 instance where your high CPU load was tested and show the screenshot of the Monitoring tab of that instance - it should show the CPU Utilization chart with the increased CPU usage, above the threshold.

|  |
| --- |
| The jar maxes out the CPU, and the alarm to create a new instance is for over an average of 30% across instances in the auto scaler. This increased the nubmer of instances to 4, resulting in an average of 25% cpu usage assuming little to no usage from the newly spun up instances. As a result, a 5th instance is not created, since the alarm is no longer triggered.  To decrease down to a single instance either the scaling policies can be adjusted to ignore the high cpu usage, or the ar execution can be terminated. This will trigger the ‘decrease group size’ policy, and terminate all but 1 instance. |