**Experiment No. 8**

**Topic :​** Load Balancer

**Aim :**​ To understand and implement Load Balancer on AWS / GCP Cloud Platform for a Web App.

**Theory:**

A ​Load Balancer​ distributes incoming application traffic across multiple EC2 instances in multiple Availability Zones. This increases the fault tolerance of your applications. Elastic Load Balancing detects unhealthy instances and routes traffic only to healthy instances.

Your load balancer serves as a single point of contact for clients. This increases the availability of your application. You can add and remove instances from your load balancer as your needs change, without disrupting the overall flow of requests to your application. Elastic Load Balancing scales your load balancer as traffic to your application changes over time. Elastic Load Balancing can scale to the vast majority of workloads automatically.

A listener checks for connection requests from clients, using the protocol and port that you configure, and forwards requests to one or more registered instances using the protocol and port number that you configure. You add one or more listeners to your load balancer.

You can configure health checks, which are used to monitor the health of the registered instances so that the load balancer only sends requests to the healthy instances.

Benefits :

● Reduced Downtime

● Scalable

● Redundancy

● Flexibility

● Efficiency

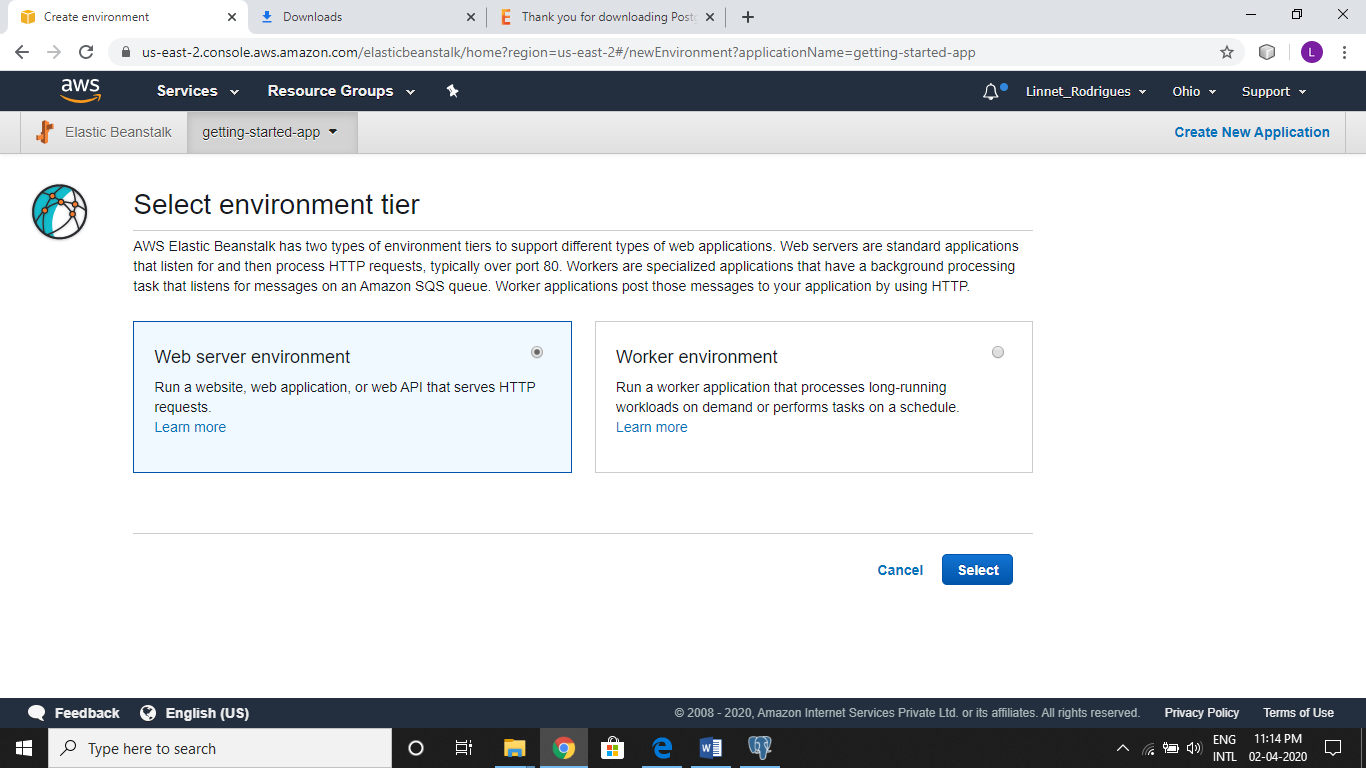
● Global Server Load Balancing

**Implementation With Output:**

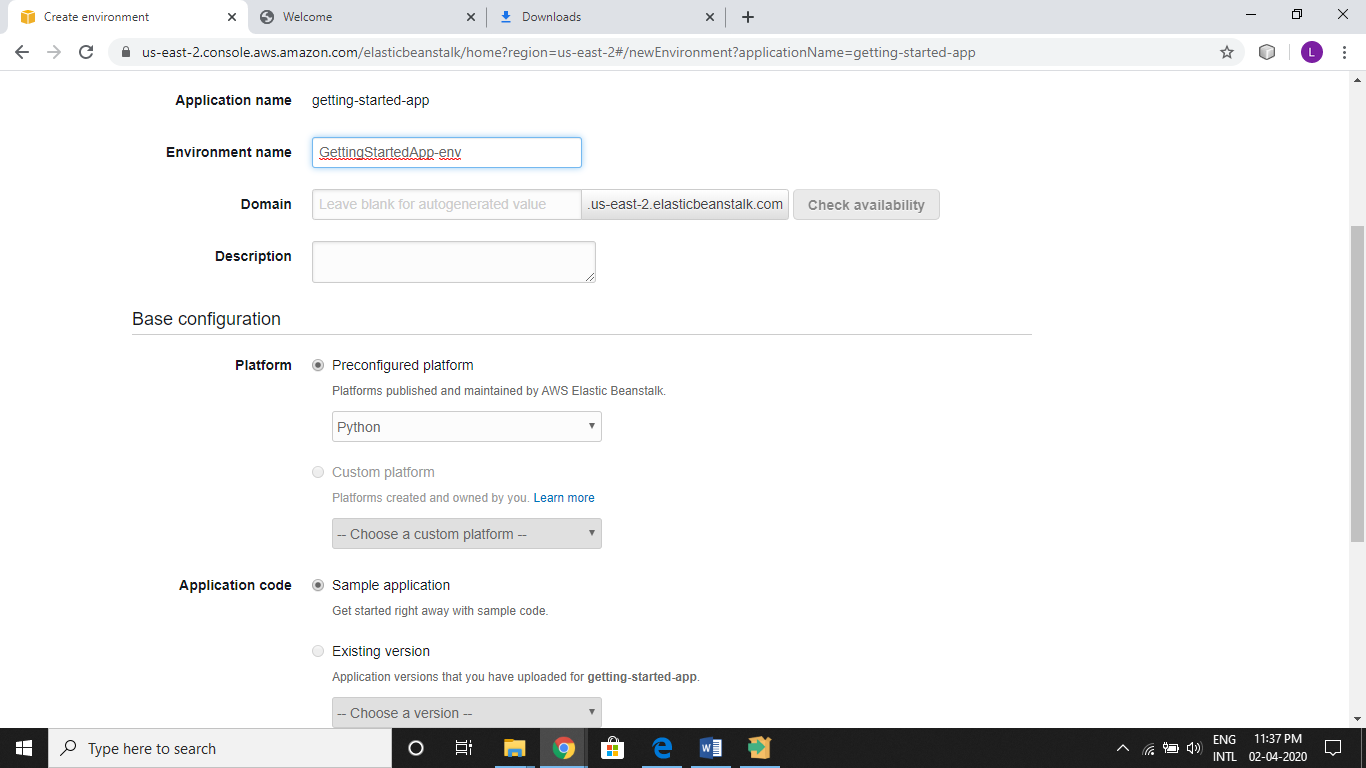
Step 1: Open Elastic Beanstalk to deploy two web apps.

Step 2: Create two applications

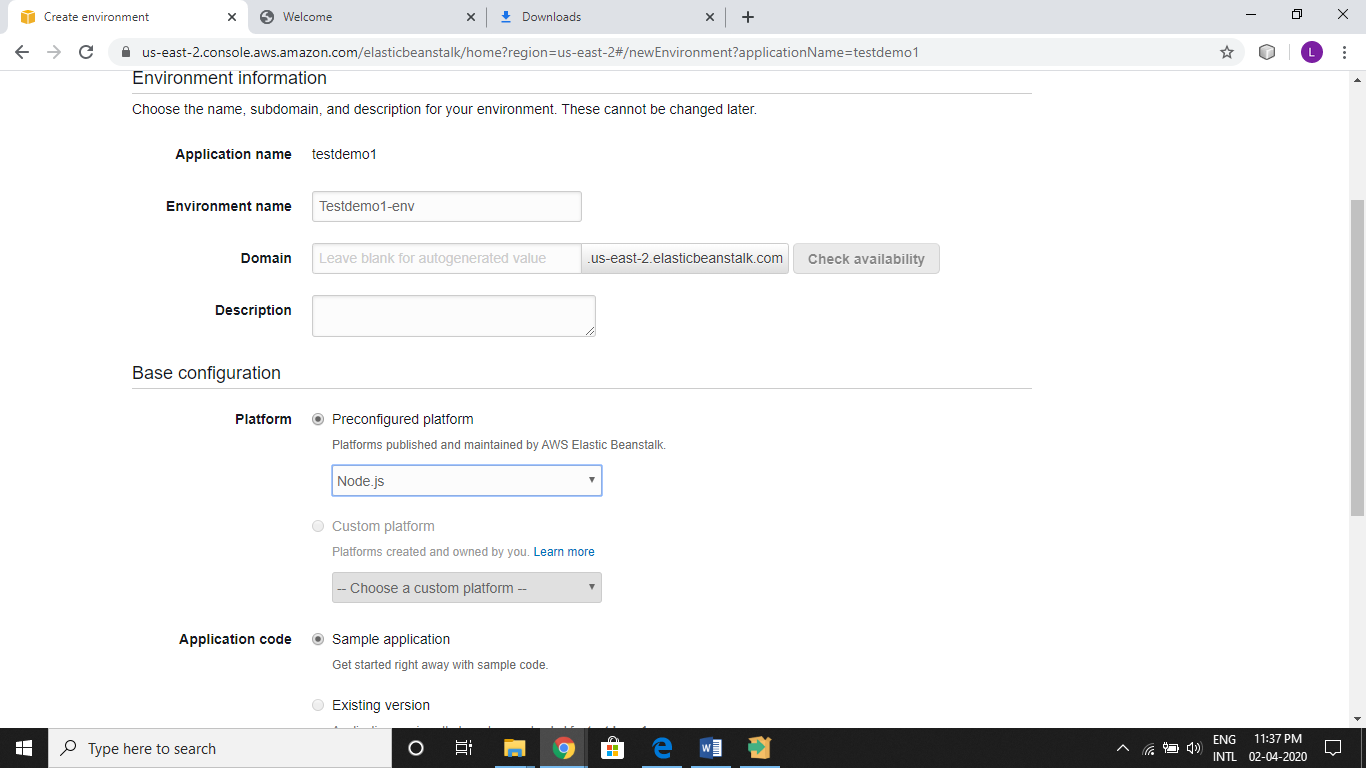
Step 3: Under Actions -> Select Create Environment



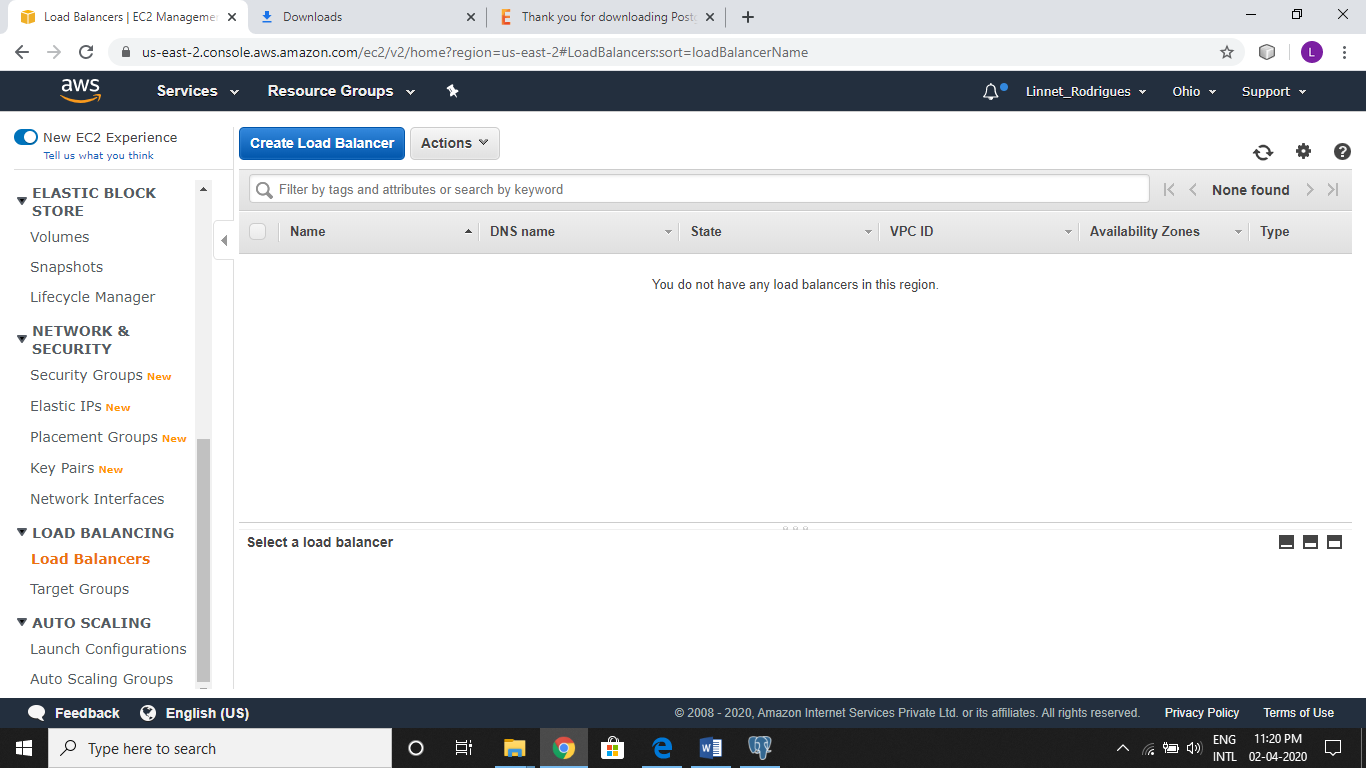
Step 4: Create one Python Web App



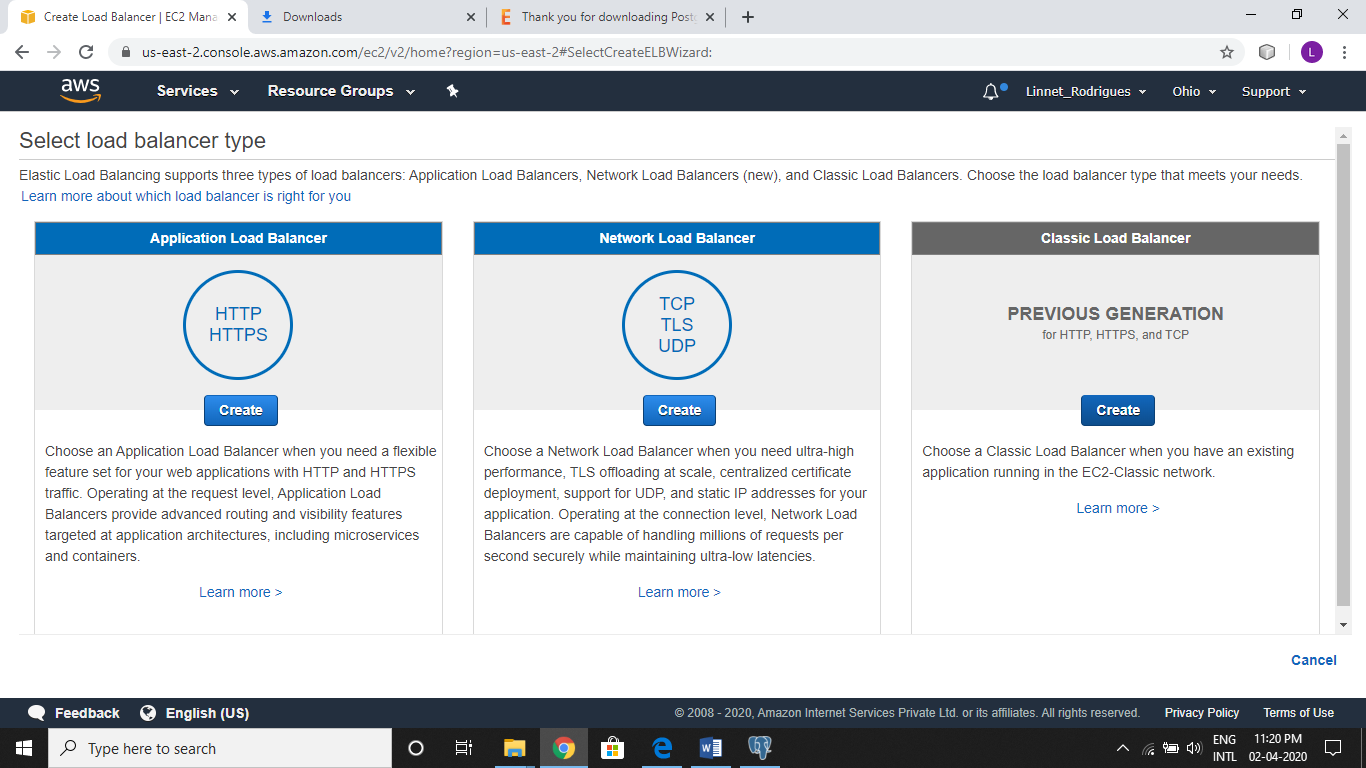
Step 5: Create another Nodejs Web App



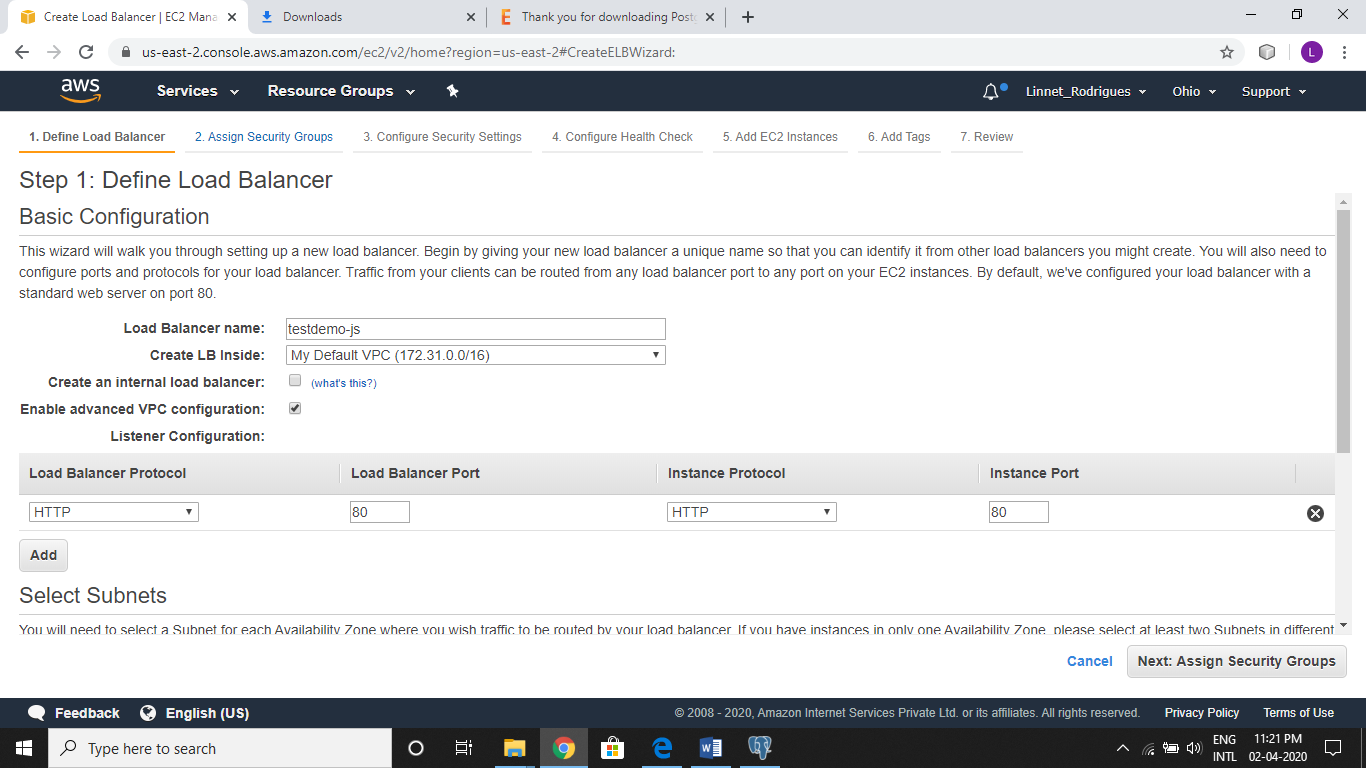
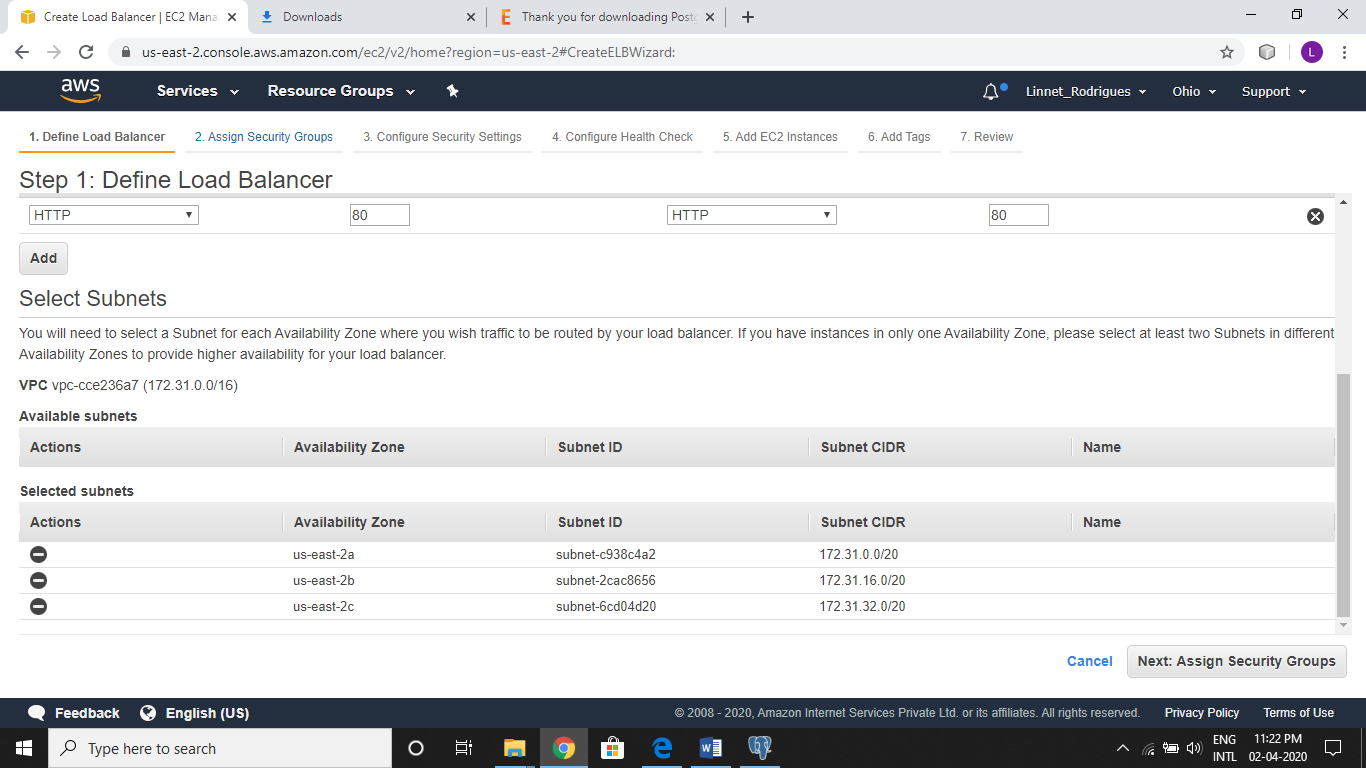
Step 6: Open EC2 Console & Select Load Balancer from Bottom Left of the screen and create Load Balancer



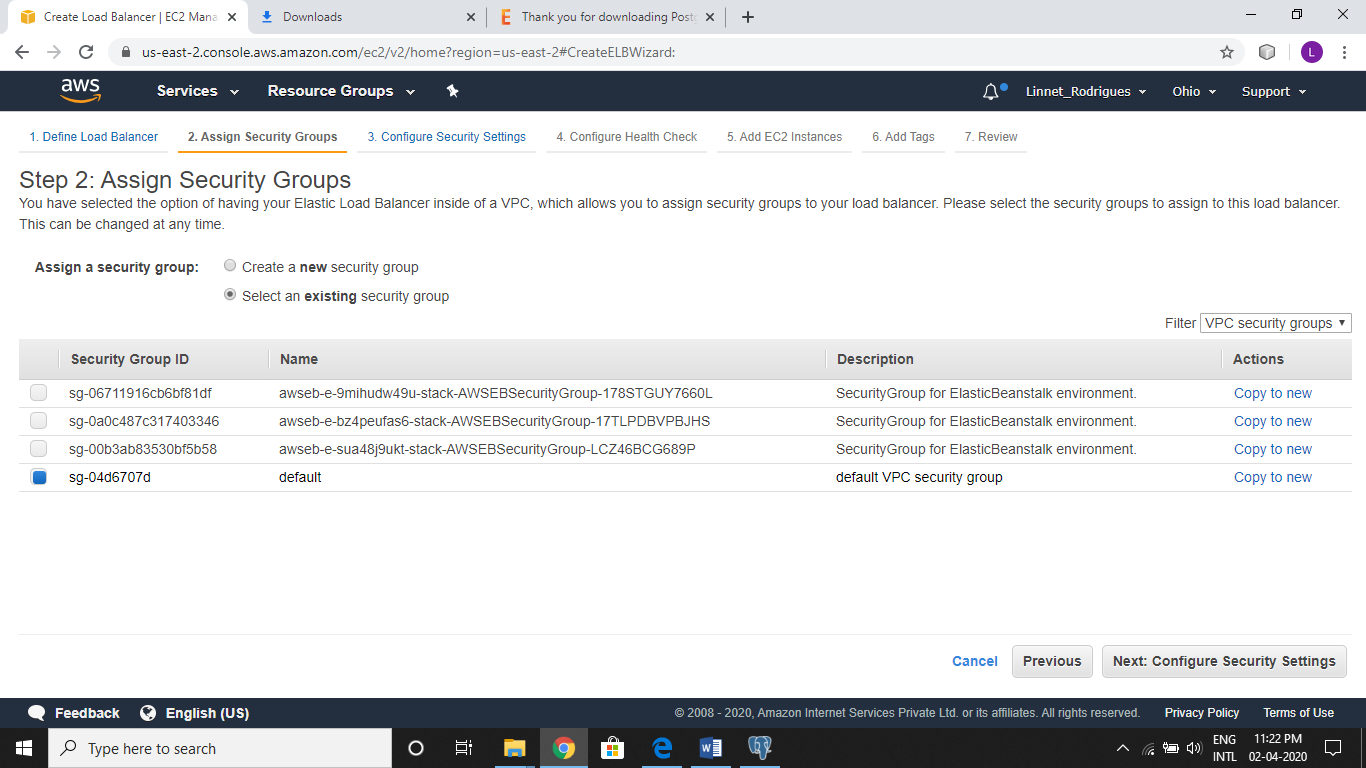
Step 7: Select Classic Load Balancer ( this is easier to configure )



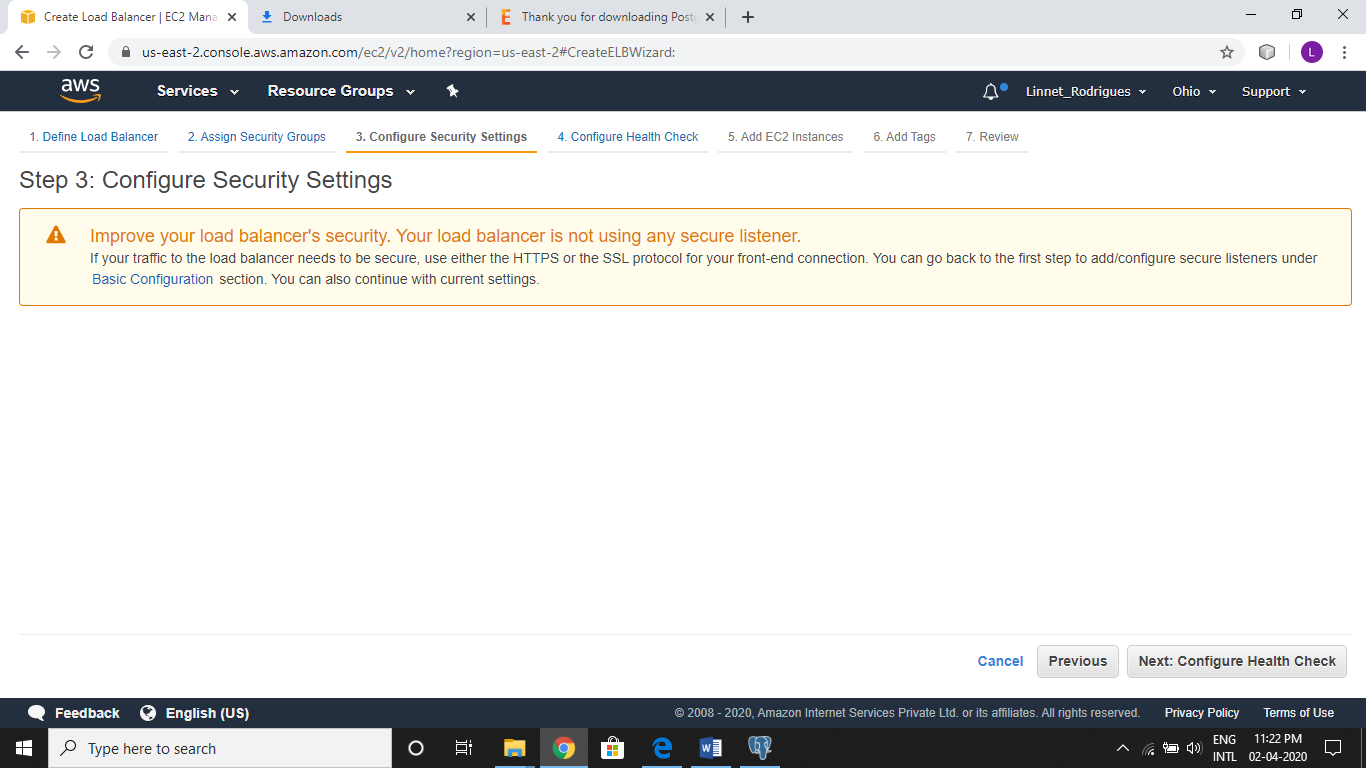
Step 8: Give name to load balancer, enable advanced VPC Configuration & Select all subnets.



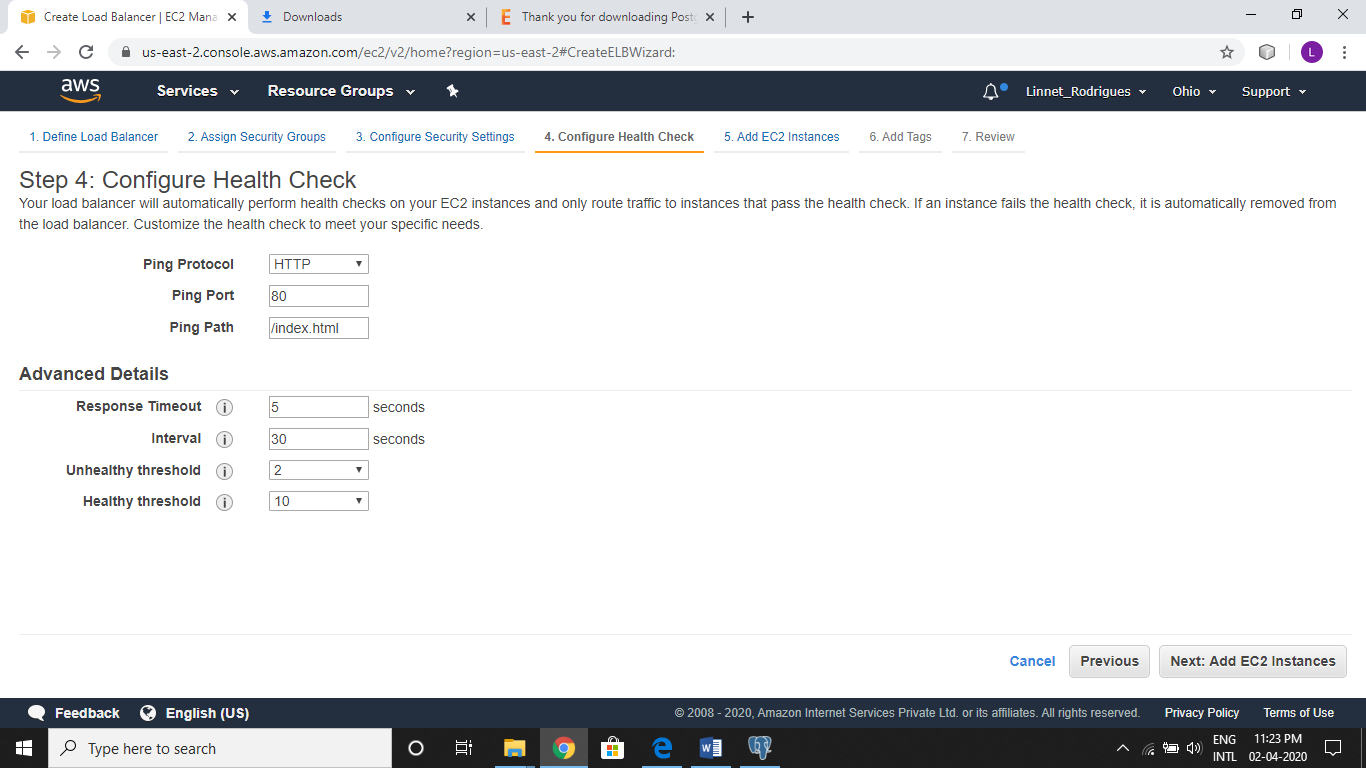
Step 9: Select Security Group



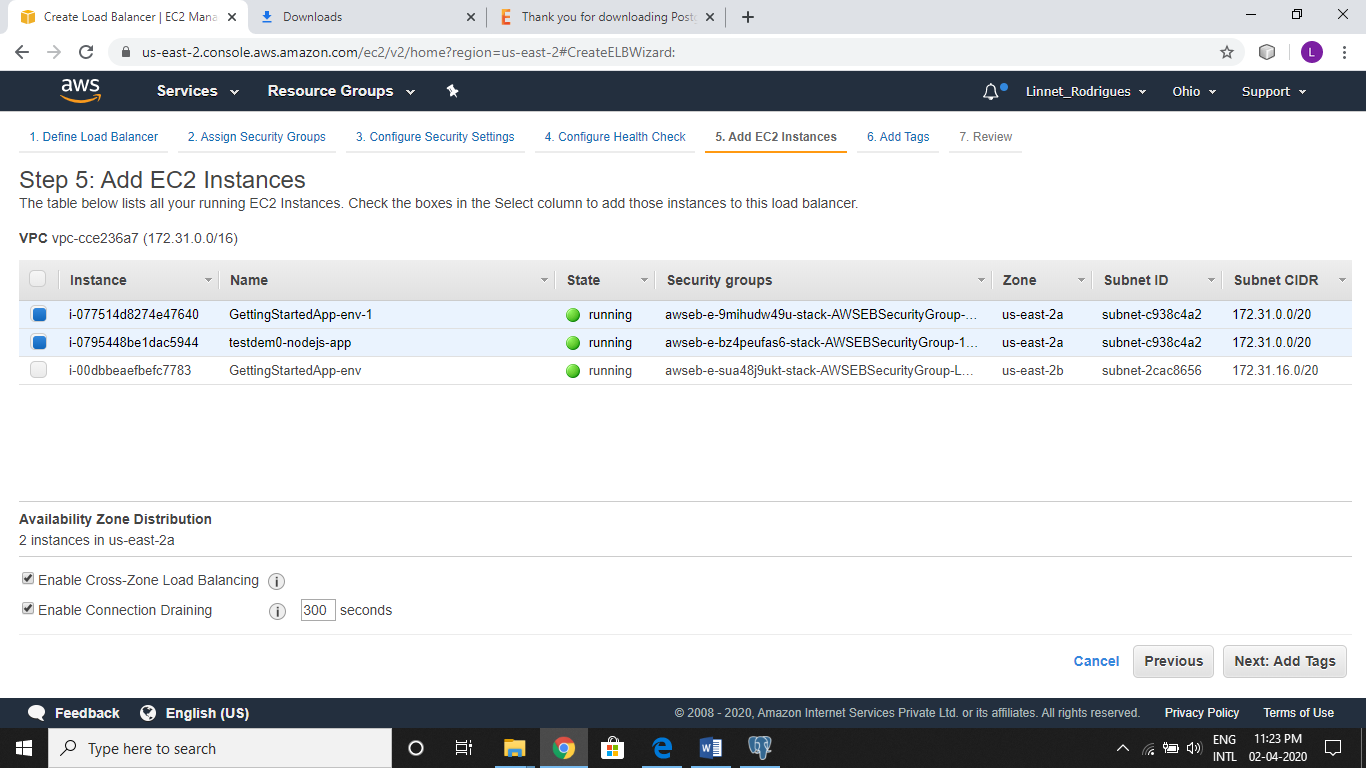
Step 10: Ignore the Security setting



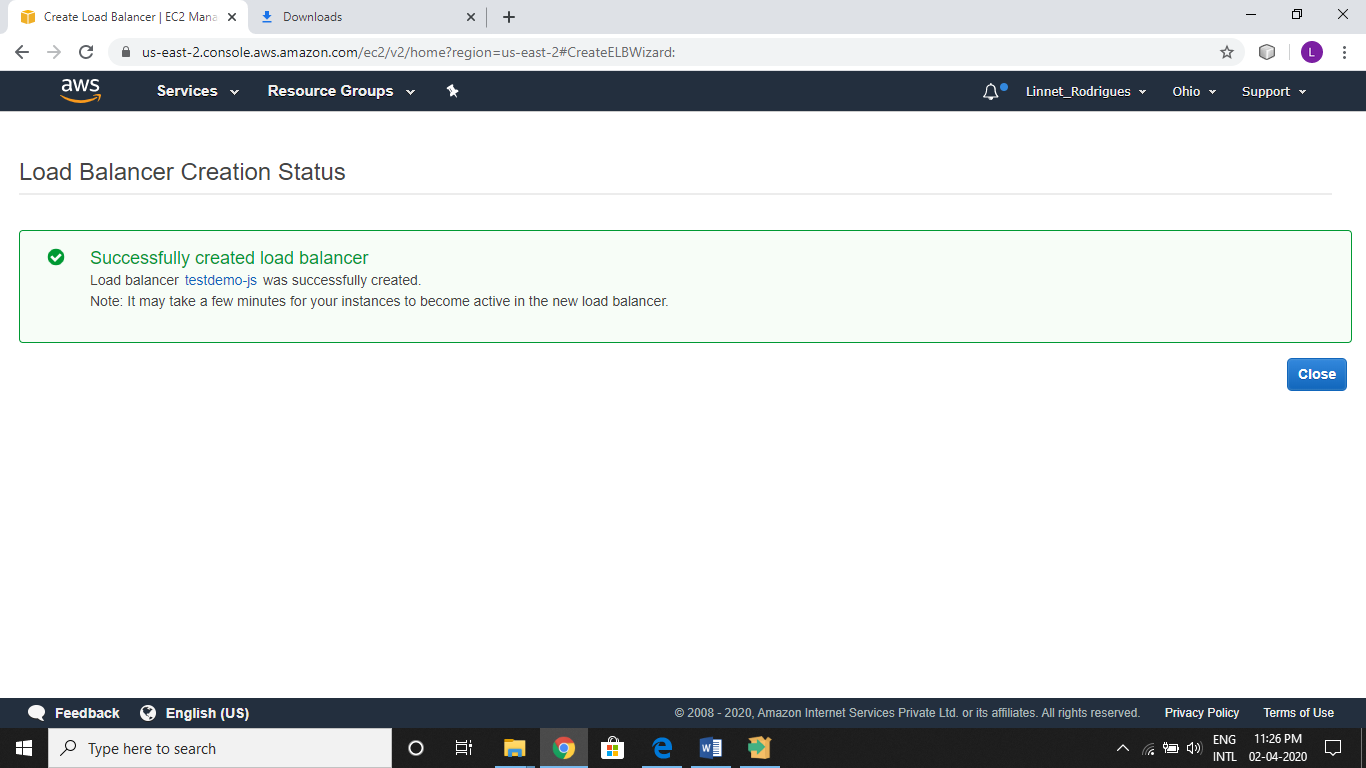
Step 11: Keep Health Check as default



Step 12: Select the EC2 instances of both apps we created using Elastic Beanstalk

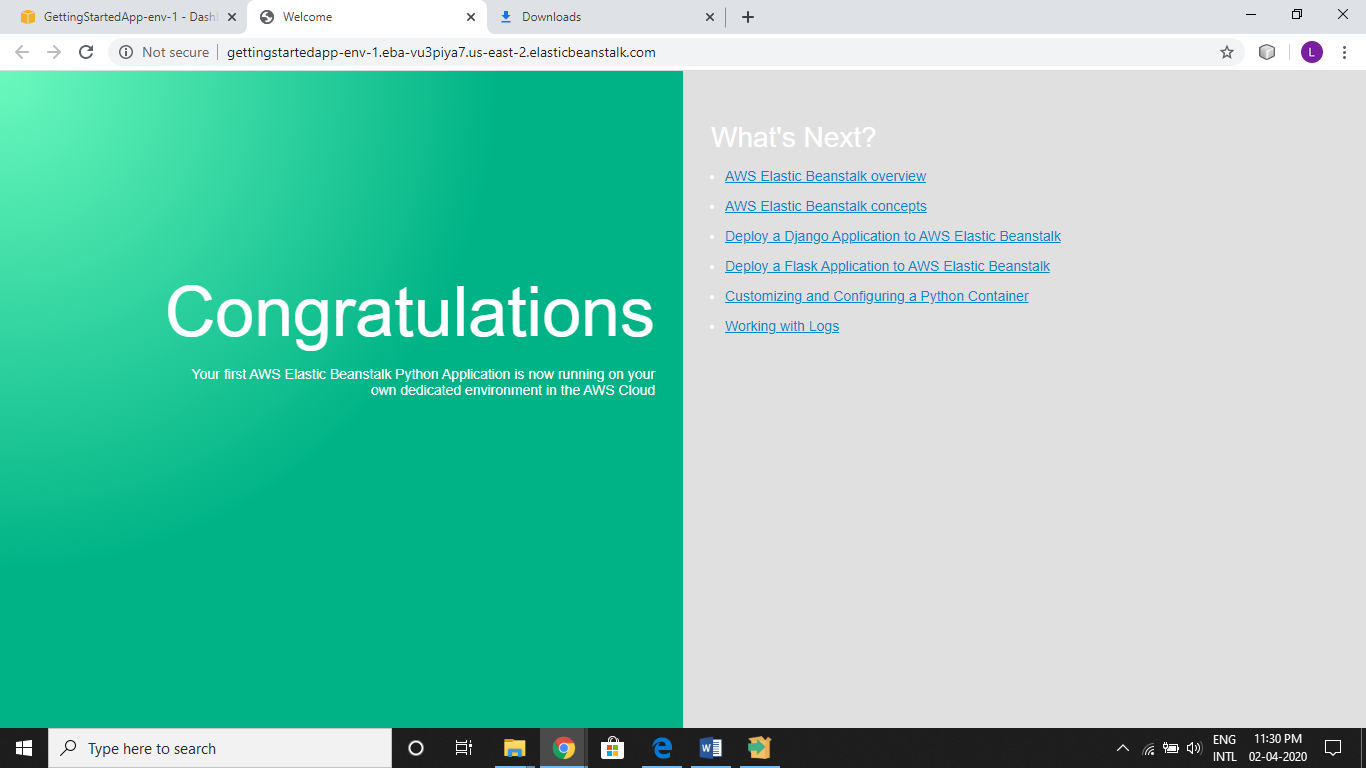


Step 13: Review and submit the configuration to create the load balancer. Wait for 5minutes for it to start working. The URL in DNS Name is the URL for your Application

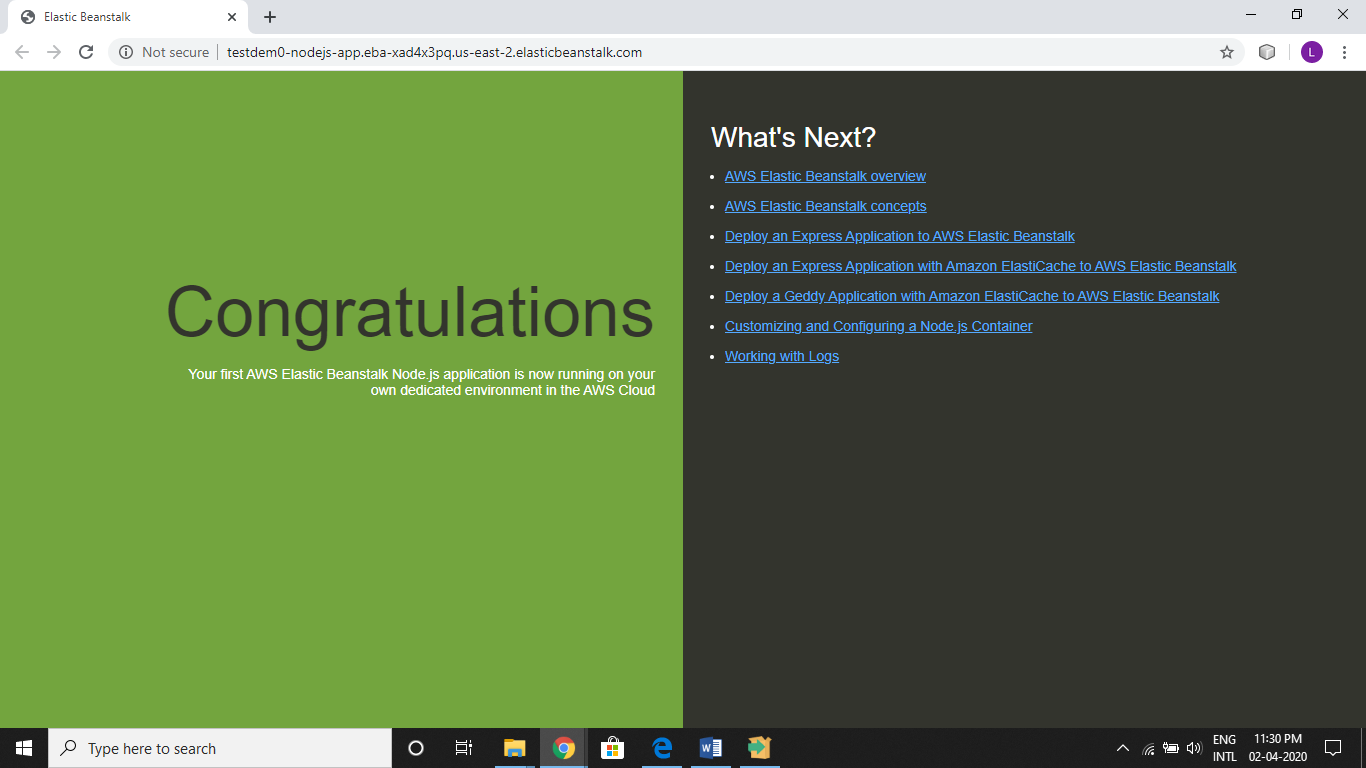


Step 14: Open the URL in different tabs, and each tab will display either of the nodejs or python app selected by LoadBalancer in Round Robin fashion.

Pyhton app



Nodejs app



**Conclusion:** Hence, we have successfully created two application using Elastic Beanstalker and by using EC2 Laod Balancer we managed the traffic across the two applications and has increased availability of the application.

**Postlabs :**

1. Suppose you have an application where you have to render images and also do some general computing. From the following services which service will best fit your needs? (Explain your choice)
   1. Classic Load Balancer
   2. Application Load Balancer
   3. Both of Them
   4. None of These

ANS:- Application Load Balancer.

Explaination:- Application Load Balancer supports path based routing, which means it can take decisions based on the URL, therefore if your task needs image rendering it will route it to a different instance, and for general computing it will route it to a different instance.

1. When should we use a Classic Load Balancer and when should we use an Application load balancer?

ANS:- A Classic Load Balancer is ideal for simple load balancing of traffic across multiple EC2 instances, while an Application Load Balancer is ideal for microservices or container-based architectures where there is a need to route traffic to multiple services or load balance across multiple ports on the same EC2 instance

1. Difference between Route53 and ELB in AWS
   1. ELB distributes traffic among Multiple Availability Zone but not to multiple Regions. Route53 can distribute traffic among multiple Regions. In short, ELBs are intended to load balance across EC2 instances in a single region whereas DNS load-balancing (Route53) is intended to help balance traffic across regions.
   2. Both Route53 and ELB perform health check and route traffic to only healthy resources. Route53 weighted routing has health checks and removes unhealthy targets from its list. However, DNS is cached so unhealthy targets will still be in the visitors cache for some time. On the other hand, ELB is not cached and will remove unhealthy targets from the target group immediately.
2. What algorithm does Amazon ELB use to balance load ?

With **Application Load Balancers**, the load balancer node that receives the request uses the following process:

1. Evaluates the listener rules in priority order to determine which rule to apply.
2. Selects a target from the target group for the rule action, using the routing algorithm configured for the target group. The default routing algorithm is round robin. Routing is performed independently for each target group, even when a target is registered with multiple target groups.

With **Network Load Balancers**, the load balancer node that receives the connection uses the following process:

* Selects a target from the target group for the default rule using a flow hash algorithm. It bases the algorithm on:

The protocol

The source IP address and source port

The destination IP address and destination port

The TCP sequence number

* Routes each individual TCP connection to a single target for the life of the connection. The TCP connections from a client have different source ports and sequence numbers, and can be routed to different targets.

With **Classic Load Balancers**, the load balancer node that receives the request selects a registered instance as follows:

1. Uses the round robin routing algorithm for TCP listeners
2. Uses the least outstanding requests routing algorithm for HTTP and HTTPS listeners