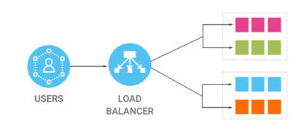
ELASTIC LOAD BALANCER:

Elastic Load Balancing is shortly called as ELB that allows the income of traffic to get easily distributed across the plenty of healthy EC2 instances, IP Address, and containers in an automatic way. It is considered as the single point to make contact with the client and helps to increase the availability of applications which allows adding or else removal of different EC2 instances across single or else more availability zones. It mostly achieves the fault tolerance for any of the applications which mainly ensures performance, scalability, and security.

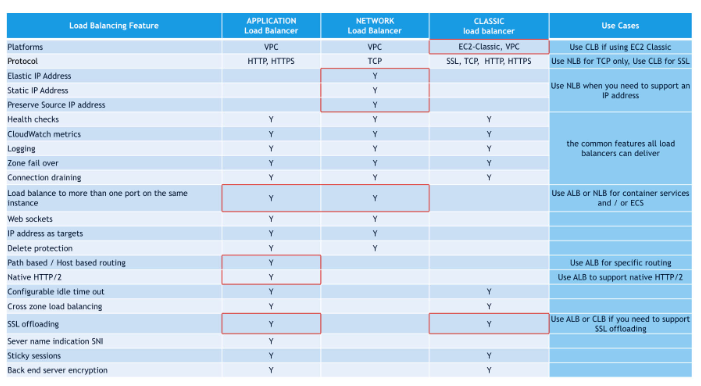


There are 3 types of Elastic Load Balancer: CLB (Classic Load Balancer), ALB (Application Load Balancer) and NLB (Network Load Balancer). AWS releases CLB first, then ALB, that’s why CLB sometimes is referred as ELB-V1, and ALB is referred as ELB-V2. Then NLB comes as the latest release.

From my point of view, the reason to have 3 types of ELB is that AWS was initially trying to do both TCP (L4) and HTTP (L7) load balancing in CLB/ELB-V1. But for some technical limits, it could not do it well – both L4 and L7 comes with limited functions. Then AWS decided to split it into two types: http/https only and TCP only (as Jobs said less is more!). That’s why it ends up having 3 types ELB.

While each load balancing use case will be unique, here are the simple rules of thumb that I use when considering which load balancer to choose:

* If you need to support a static or elastic IP address: Use Network Load Balancer
* if you need control over your SSL cipher: Use Classic Load Balancer
* If using container services and/or ECS: Use Application Load Balancer or Network Load Balancer
* If you need to support SSL offloading: Use Application Load Balancer or Classic Load Balancer



**DIFFERENCE BETWEEN APPLICATION LOAD BALANCER AND NETWORK LOAD BALANCER**

**Network load balancing** is the distribution of traffic based on network variables, such as IP address and destination ports. It is layer 4 (TCP) and below and is not designed to take into consideration anything at the application layer such as content type, cookie data, custom headers, user location, or the application behaviour. It is context-less, caring only about the network-layer information contained within the packets it is directing this way and that.

**Application load balancing** is the distribution of requests based on multiple variables, from the network layer to the application layer. It is context-aware and can direct requests based on any single variable as easily as it can a combination of variables. Applications are load balanced based on their peculiar behaviour and not solely on server (operating system or virtualization layer) information.

The difference between the two is important because network load balancing cannot assure availability of the application. This is because it bases its decisions solely on network and TCP-layer variables and has no awareness of the application at all. Generally a network load balancer will determine “availability” based on the ability of a server to respond to ICMP ping, or to correctly complete the three-way TCP handshake. An application load balancer goes much deeper, and is capable of determining availability based on not only a successful HTTP GET of a particular page but also the verification that the content is as was expected based on the input parameters.

This is also important to note when considering the deployment of multiple applications on the same host sharing IP addresses (virtual hosts in old skool speak). A network load balancer will not differentiate between Application A and Application B when checking availability (indeed it cannot unless ports are different) but an application load balancer will differentiate between the two applications by examining the application layer data available to it. This difference means that a network load balancer may end up sending requests to an application that has crashed or is offline, but an application load balancer will never make that same mistake.

**Note: CERTIFICATES**:  If we are using HTTPS, is whether the load balancer has your its certificates or not. As I understand it, Network Load Balancer doesn't need to see it, it just passes the traffic through without understanding what the traffic is. Application Load Balancer, on the other hand, needs to have your certs to decrypt the data so that it can intelligently decide where to send the data.

**Setting up an ELB through Ansible playbook:**

---

- name: A simple Load Balanced Web Server

hosts: localhost

connection: local

gather\_facts: False

vars\_files:

- awscreds

tasks:

- name: spin up the instance

ec2:

aws\_access\_key: "{{aws\_id}}"

aws\_secret\_key: "{{aws\_key}}"

region: "{{aws\_region}}"

key\_name: ubuntu\_key

group\_id: "{{group}}"

instance\_type: t2.micro

image: "{{ami}}"

wait: yes

instance\_tags:

group: webserver

exact\_count: 1

count\_tag: webserver

register: ec2\_instances

tags: ec2

- debug: msg= "{{ ec2\_instances }}"

tags: ec2\_instance

- name: Wait for servers to come online

wait\_for:

host: "{{ item.public\_ip }}"

port: 22

timeout: 120

with\_items: "{{ ec2\_instances.tagged\_instances }}"

- name: sleep for 20 sec

pause:

minutes: 1

- name: Add ec2 instances as known hosts

known\_hosts:

name: "{{ item.public\_ip }}"

key: "{{ lookup('pipe', 'ssh-keyscan -t rsa ' + item.public\_ip) }}"

with\_items: "{{ ec2\_instances.tagged\_instances }}"

- name: add server ip addresses to hosts group

add\_host:

hostname: "{{ item.public\_ip }}"

ansible\_ssh\_user: ec2-user

ansible\_ssh\_private\_key\_file: ubuntu\_key

groupname: launched

with\_items: "{{ ec2\_instances.tagged\_instances }}"

- name: configure the webservers

hosts: launched

remote\_user: ec2-user

become: True

become\_method: sudo

tasks:

- name: installing httpd package

yum: name=httpd state=installed

- name: copy index.html

copy: src=index.html dest=/var/www/html/index.html

- name: start apache2 service

service: name=httpd state=started

- name: spin up the load balancer and add the servers to it

hosts: 127.0.0.1

connection: local

become: False

vars\_files:

- awscreds

gather\_facts: False

tasks:

- name: setup a simple load balancer

ec2\_elb\_lb:

aws\_access\_key: "{{aws\_id}}"

aws\_secret\_key: "{{aws\_key}}"

name: aws-elb

state: present

region: "{{aws\_region}}"

zones:

- us-east-1a

listeners:

- protocol: http

load\_balancer\_port: 80

instance\_port: 80

register: aws-elb

- hosts: localhost

connection: local

become: False

vars\_files:

- awscreds

tasks:

- name: Add each EC2 instance to the ELB

ec2\_elb:

aws\_access\_key: "{{aws\_id}}"

aws\_secret\_key: "{{aws\_key}}"

state: present

ec2\_elbs: aws-elb

region: us-east-1

instance\_id: "{{ item.id }}"

with\_items: "{{ ec2\_instances.tagged\_instances }}"

**ASSIGNING A AUTOSCALING FOR ELB THROUGH ANSIBLE PLAYBOOK:**

---

- hosts: localhost

connection: local

remote\_user: root

become: yes

gather\_facts: no

vars\_files:

- awscreds

vars\_prompt:

- name: architecture

prompt: "enter the desired name for your infrastructure"

private: no

vars:

keypair: "ubuntu\_key"

subnetid: subnet-738b8138

loadb: "{{architecture}}-lb"

auto\_sc: "{{architecture}}\_sg"

lc: "{{architecture}}\_lc"

zone: us-east-1a

tasks:

- name: creating load balancer

ec2\_elb\_lb:

aws\_access\_key: "{{aws\_id}}"

aws\_secret\_key: "{{aws\_key}}"

region: "{{aws\_region}}"

name: "{{loadb}}"

zones:

- us-east-1a

- us-east-1b

- us-east-1c

- us-east-1d

- us-east-1e

state: present

listeners:

- protocol: http

load\_balancer\_port: 80

instance\_port: 80

- name: creating launch configuration

ec2\_lc:

aws\_access\_key: "{{aws\_id}}"

aws\_secret\_key: "{{aws\_key}}"

region: "{{aws\_region}}"

name: "{{lc}}"

image\_id: "{{ami}}"

key\_name: "{{keypair}}"

security\_groups: "{{group}}"

instance\_type: t2.micro

volumes:

- device\_name: /dev/sda1

volume\_size: 10

device\_type: io1

iops: 300

delete\_on\_termination: true

- name: creating auto scaling group

ec2\_asg:

aws\_access\_key: "{{aws\_id}}"

aws\_secret\_key: "{{aws\_key}}"

name: "{{auto\_sc}}"

region: "{{aws\_region}}"

load\_balancers: "{{loadb}}"

availability\_zones: "{{zone}}"

launch\_config\_name: "{{lc}}"

min\_size: 1

max\_size: 2

desired\_capacity: 1

vpc\_zone\_identifier: "{{subnetid}}"

# wait\_for\_instance: true

- name: creating auto scaling policy

ec2\_scaling\_policy:

aws\_access\_key: "{{aws\_id}}"

aws\_secret\_key: "{{aws\_key}}"

region: "{{aws\_region}}"

state: present

name: "{{item.pol\_name}}"

# name: "Scaledown\_policy"

adjustment\_type: "ChangeInCapacity"

asg\_name: "{{auto\_sc}}"

scaling\_adjustment: "{{item.changes}}"

# scaling\_adjustment: -1

min\_adjustment\_step: 1

cooldown: 300

# name: "Scaleup\_policy"

# adjustment\_type: "ChangeInCapacity"

# asg\_name: "{{auto\_sc}}"

# scaling\_adjustment: "{{item.changes}}"

# scaling\_adjustment: +1

# min\_adjustment\_step: 1

# cooldown: 300

register: policies

with\_items:

- pol\_name: "cpuUP\_{{auto\_sc}}\_policy"

changes: +1

- pol\_name: "cpuDown\_{{auto\_sc}}\_policy"

changes: -1

- name: creating cloud watch alarm for CPU Utilization

ec2\_metric\_alarm:

aws\_access\_key: "{{aws\_id}}"

aws\_secret\_key: "{{aws\_key}}"

region: "{{aws\_region}}"

state: present

name: "{{item.names}}"

metric: "CPUUtilization"

namespace: "AWS/EC2"

statistic: Average

comparison: "{{item.compare}}"

threshold: "{{item.limits}}"

period: 60

evaluation\_periods: 1

unit: "Percent"

description: "{{item.desc}}"

dimensions: {'AutoScalingGroupName':'{{auto\_sc}}'}

alarm\_actions: "{{item.pol}}"

with\_items:

- names: "cpuUP\_{{auto\_sc}}"

compare: ">="

limits: "20.0"

desc: "This will alarm when the average cpu usage of the ASG is greater than 20% for 1 minute"

pol: "{{policies.results[0]['arn']}}"

- names: "cpuDown\_{{auto\_sc}}"

compare: "<="

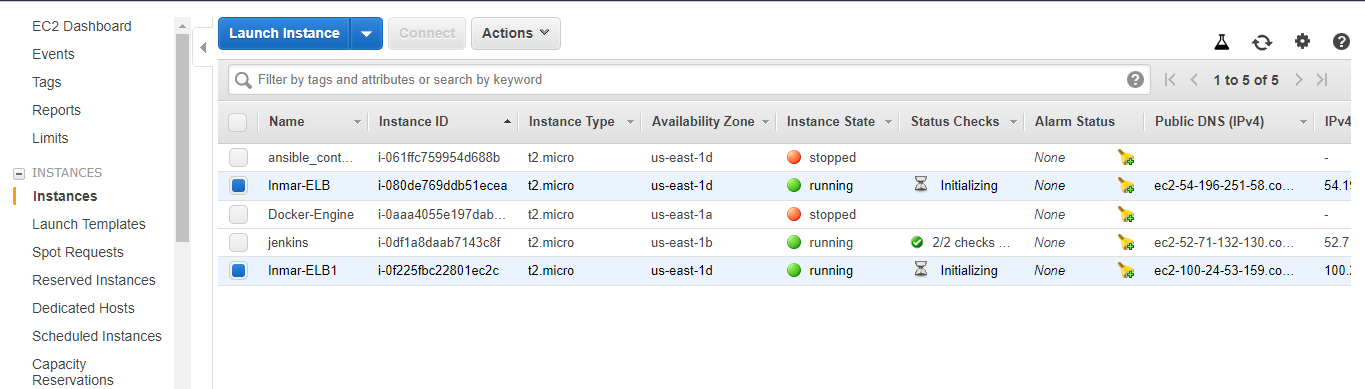
limits: "10.0"

desc: "This will alarm when the average cpu usage of the ASG is less than 10% for 1 minute"

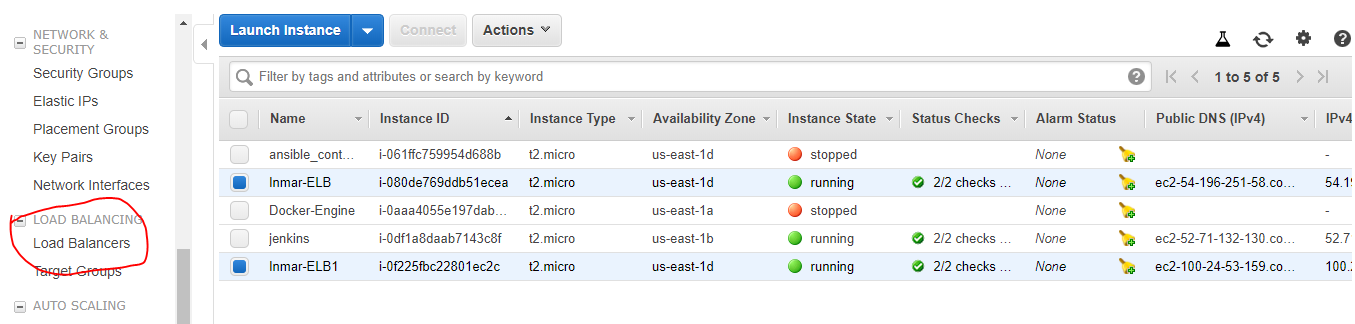
pol: "{{policies.results[1]['arn']}}"

AWS-CONSOLE:

1. Create 2 EC2 instances, configure virtual private cloud (VPC) with at least one private subnet in each of these Availability Zones. These priavte subnets are used to configure the load balancer. You can launch your EC2 instances in other subnets of these Availability Zones instead.



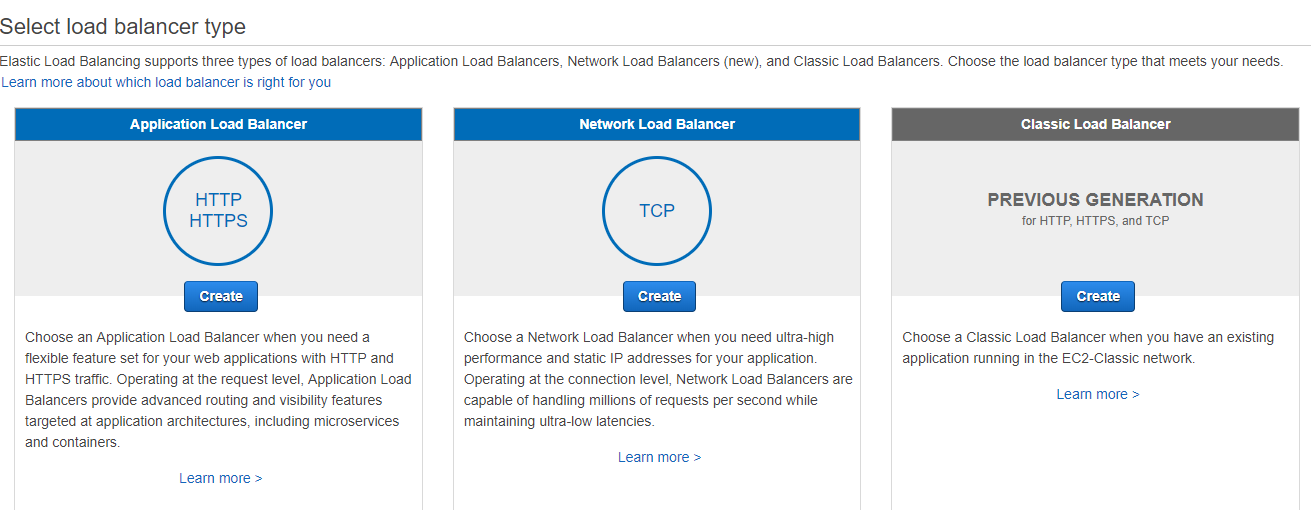
1. Now on the navigation pane, under **LOAD BALANCING**, choose **Load Balancers**.



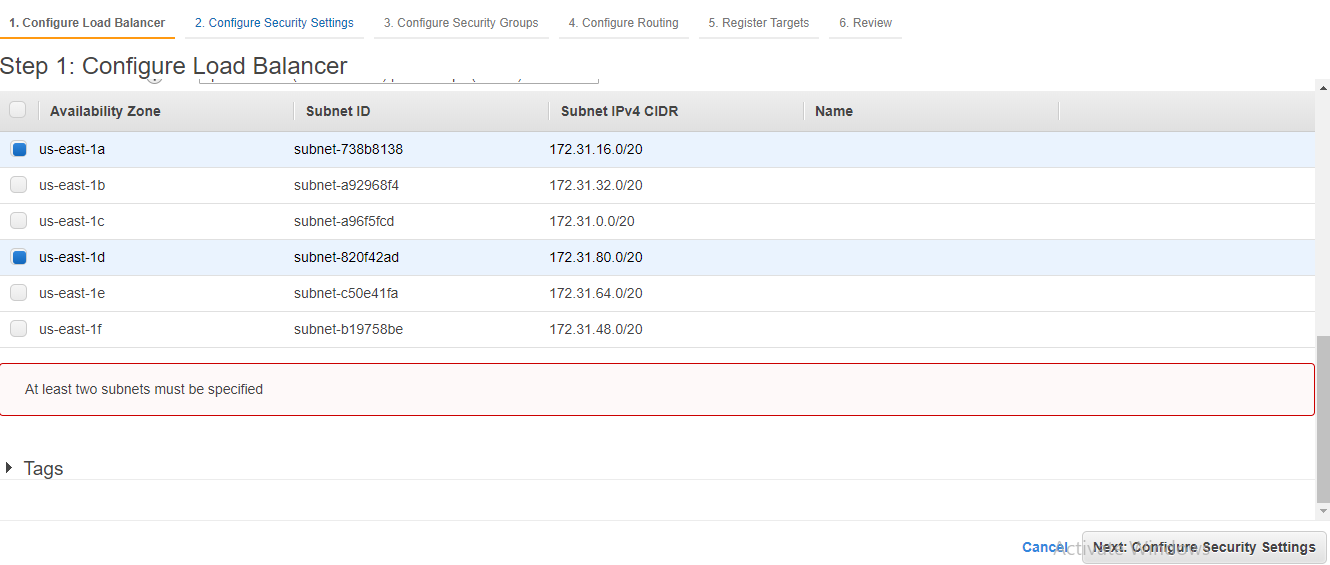
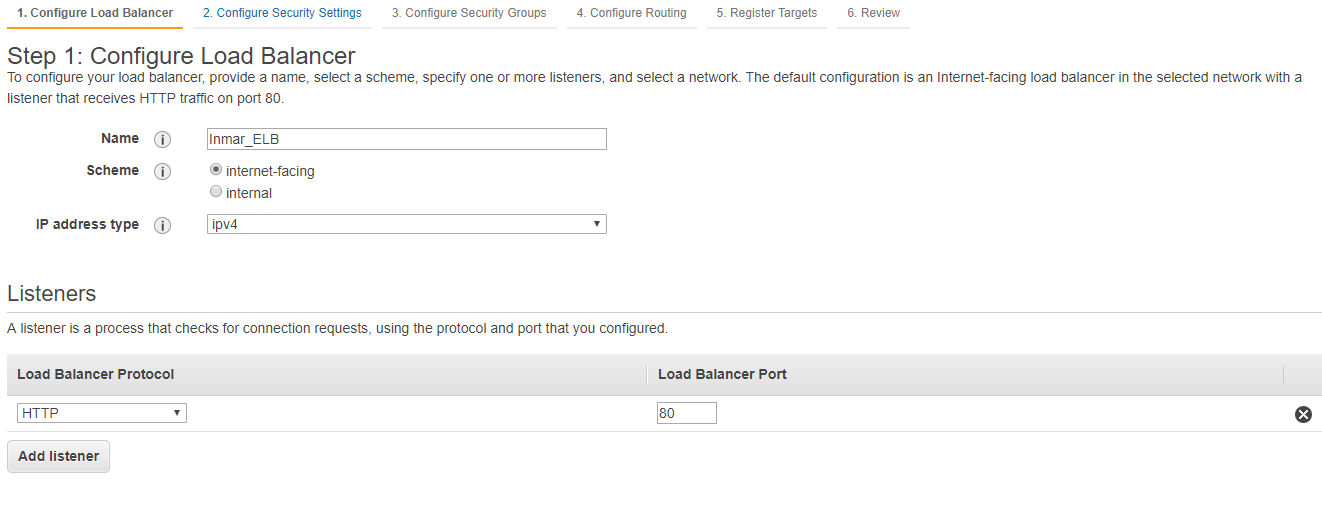
1. Choose **Create Load Balancer**.



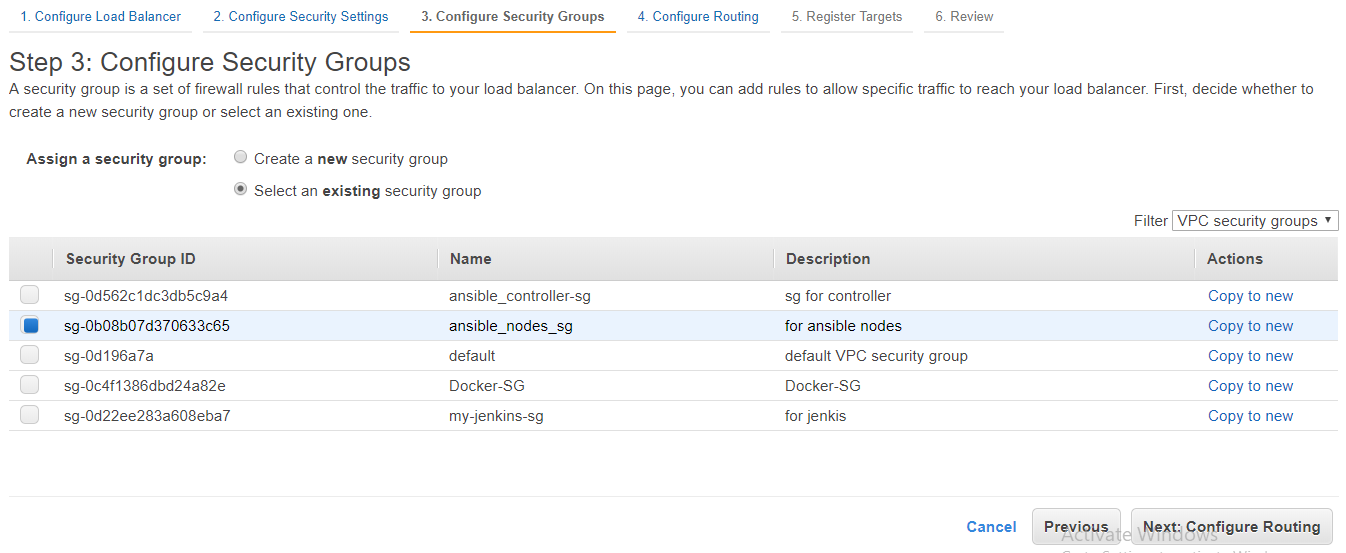
1. Choose **Application Load Balancer**, choose **Create**.



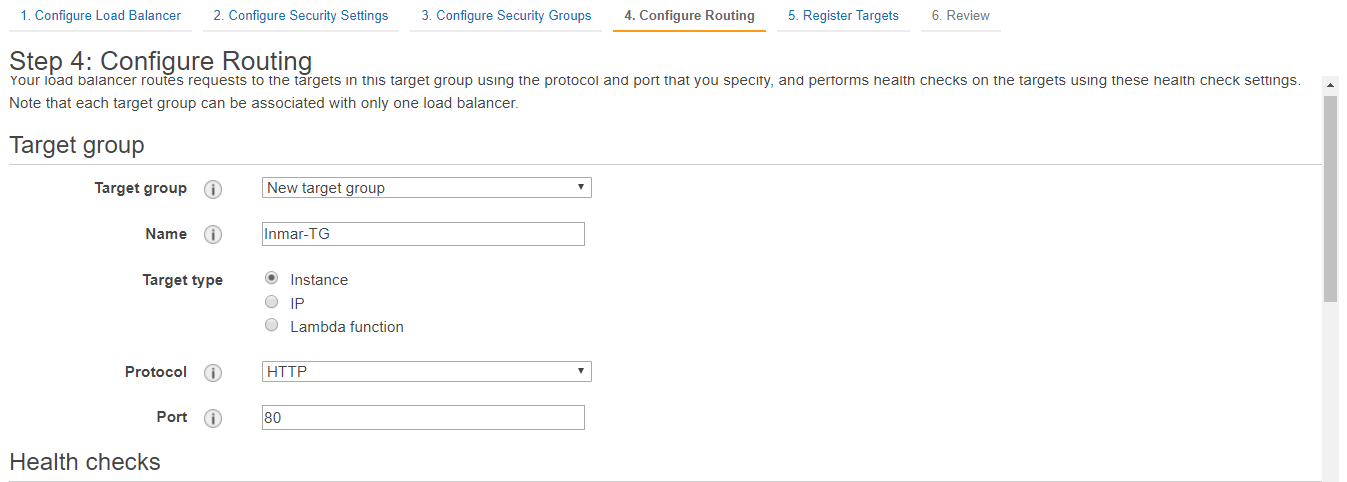
1. Configure the Load Balancer :

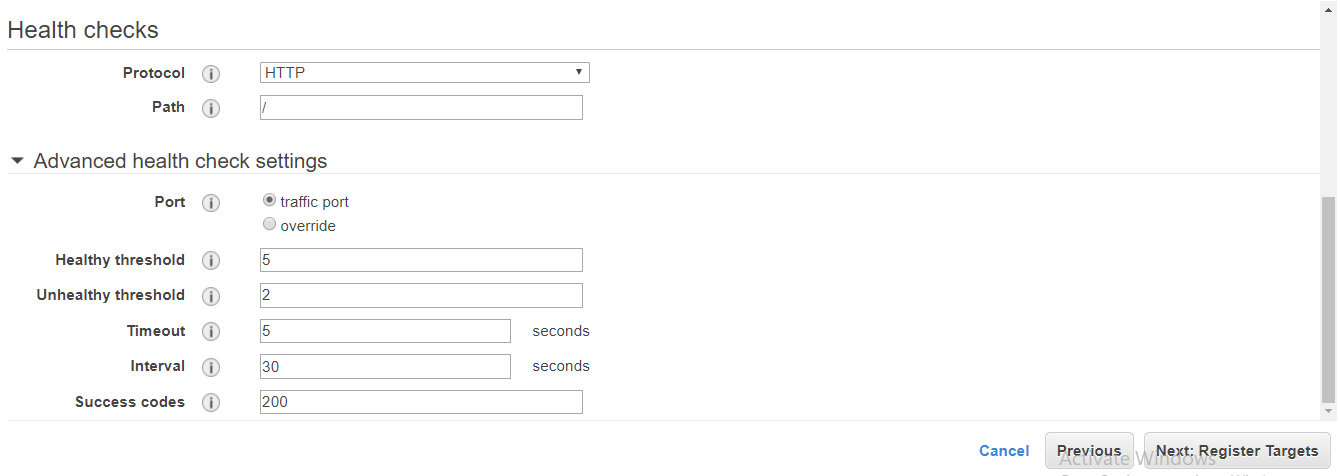


1. Select the VPC:

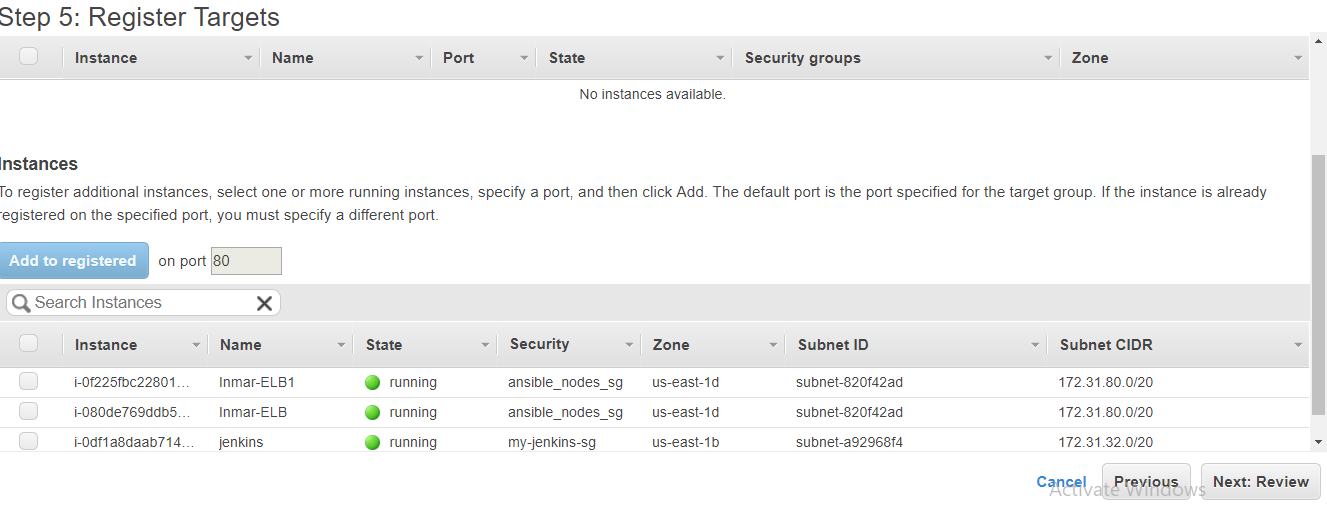


1. Configure Routing:

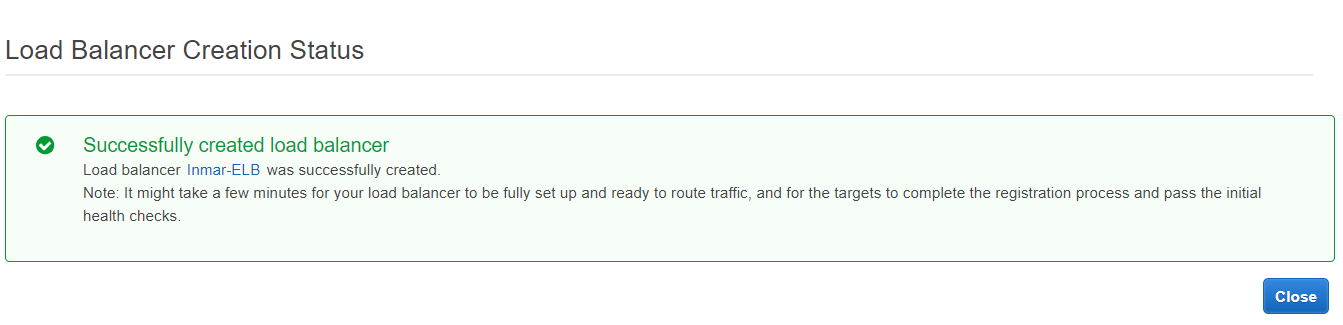




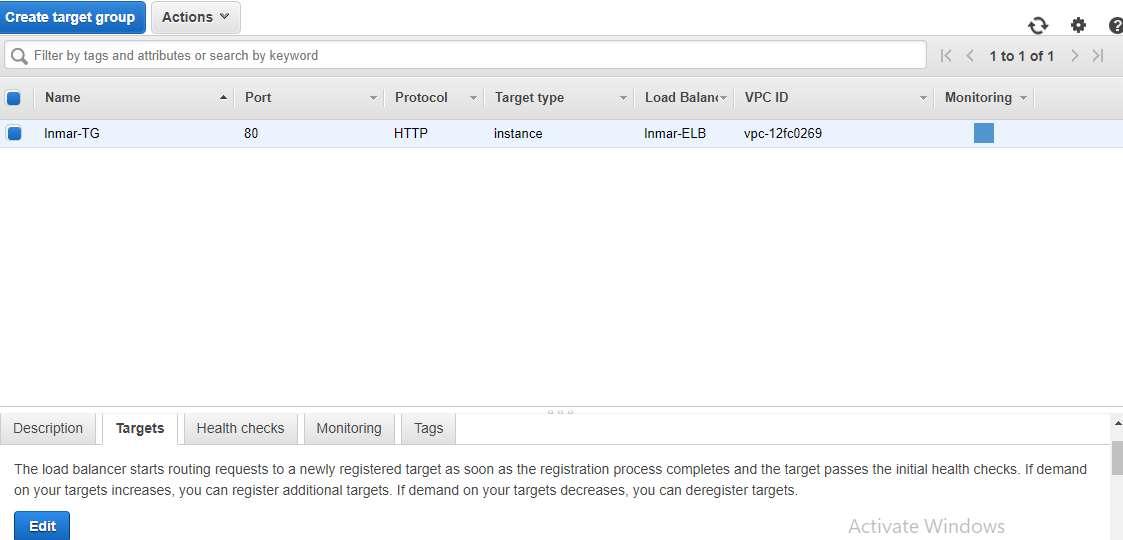
1. Target Register:



1. Load Balancer created:



1. 
2. Now add the EC2 instances, go to the Targets tab below and click on Edit button:



1. Now choose the desired EC2 instances, and click on “Add to registered” button and click on save:

