Table of Contents

1. AWS ELB

What is AWS ELB?

AWS ELB (Amazon Elastic Load Balance) helps to distribute the application traffic to various different targets such as EC2 instances. The vacant targets which are ready to collect the traffic are monitored by Amazon ELB whether they are healthy or not and the traffic is sent to the healthy one.
AWS ELB comes in three versions which perform different tasks.

- The Version 1 provides detailed instructions for using *Classic Load Balancers*.
- 2nd version provides detailed instructions for using *Application Load Balancers*.
- 3rd provides detailed instructions for using *Network Load Balancers*.

Types of Load Balancer in AWS

These are the unique Versions of AWS ELB, let's discuss them one by one:

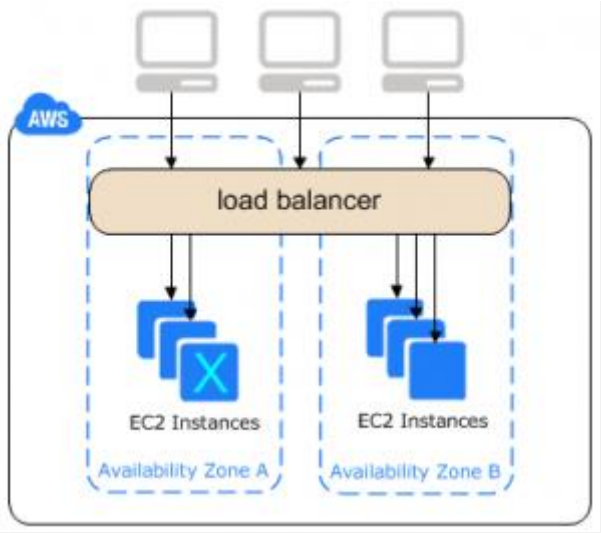
a. Classic Load Balancers

Classic Load Balancers distribute upcoming traffic to different EC2 instances in multiple Availability Zones. During this process, there is a chance of the fault tolerance of your application. These Load Balancers detect healthy and unhealthy instance and direct the traffic towards only healthy ones.

It also helps in a way such that without disrupting the flow of requests to your application you can add or remove instances from your load balancers as your need changes.

AWS ELB can calculate the majority of workloads automatically. Protocol and port which a person configures are used to detect the connection requests from clients it also forwards requests to one or more registered instances.

The number of instances can be modified. Health checks can be monitored so that the load balancer only sends requests to the healthy instances.



Learn AWS ELB – Classic Load Balancers

Benefits of Load Balancers-

- Provides Support to TCP and SSL listeners.
- Support to Sticky Session.
- Support to EC2- Classic.

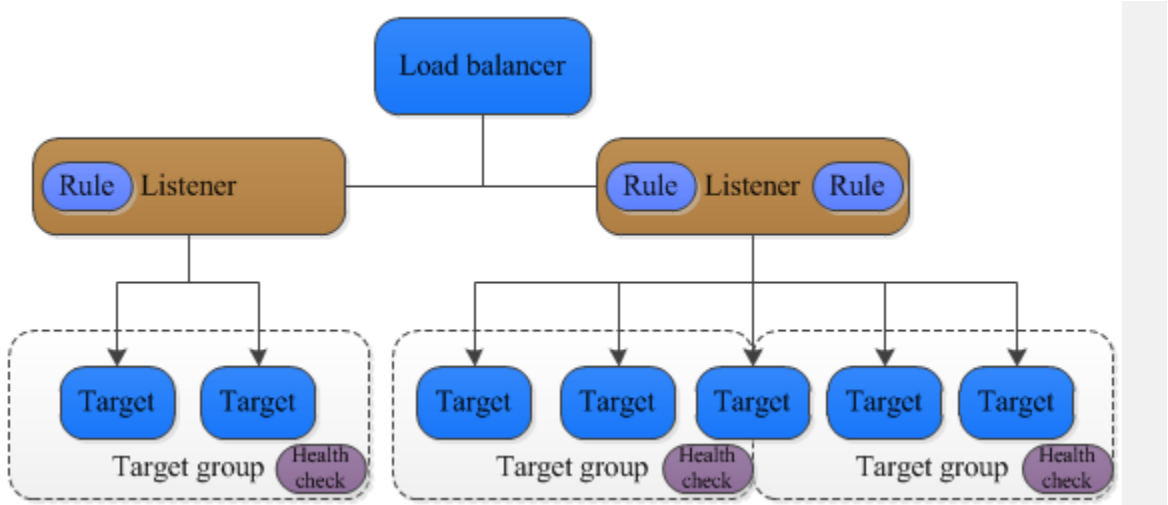
b. Application Load Balancers

After receiving the request Application Load Balancer analyzes the rules provide by the listener in priority order and determines the rule which has to apply. After that, it selects a target from the target group for the rule action.

An Application Load Balancer functions at the Open Systems Interconnection (OSI) model which is the seventh layer of the OSI model.

The User can analyze the rules of the listener and can modify it by sending it to different target groups based on the content of the application traffic even when the target is associated with multiple target groups.

Addition and removal of tags can do from the load balancers as per your needs. This can done without breaking the flow of your requests of the application.



AWS ELB – Application Load Balancers

Benefits of Application Load balancers-

- Load Balancer's performance improve in Application Load Balancer.
- Access logs containing information compress such that they may not require the additional space.
- Provides benefit for registering targets by IP address, including targets outside the VPC for the load balancer.

c. Network Load Balancers

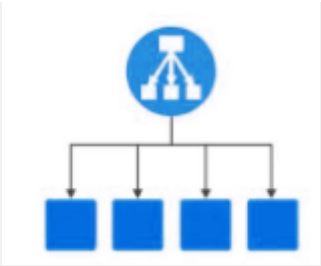
It is the fourth layer of the Open System Interconnection Model. After the load balancer receives a connection request, it selects a target from the group which targets for the default rule.

After enabling the availability zone Elastic Load Balancer creates the load balancer node in the availability zone. Each load balancer node automatically distributes traffic across the registered targets in its Availability Zone only.

Cross-zone Load Balancing enables to distribute traffic across the registered targets in all enabled Availability Zones.

Enabling Multiple Availability Zone can cause harm by increasing the fault tolerance of the applications and it will happen if each target group has at least one target in each enabled Availability Zone.

The problem can overcome in such a way that if one or more target groups do not have a healthy target in an Availability Zone, the IP address for the corresponding subnet from DNS is removed. If a person attempts again the request fails.



AWS ELB – NLB

Benefits of Network Load Balancers-

- NLB Provides the Support for static IP addresses for the load balancer.
- Provides support for registering targets by IP address which includes target outside the VPC for the Load Balancer.
- Provides support for monitoring the health of each service independently.

So, this was all about AWS ELB Tutorial. Hope you like our explanation.

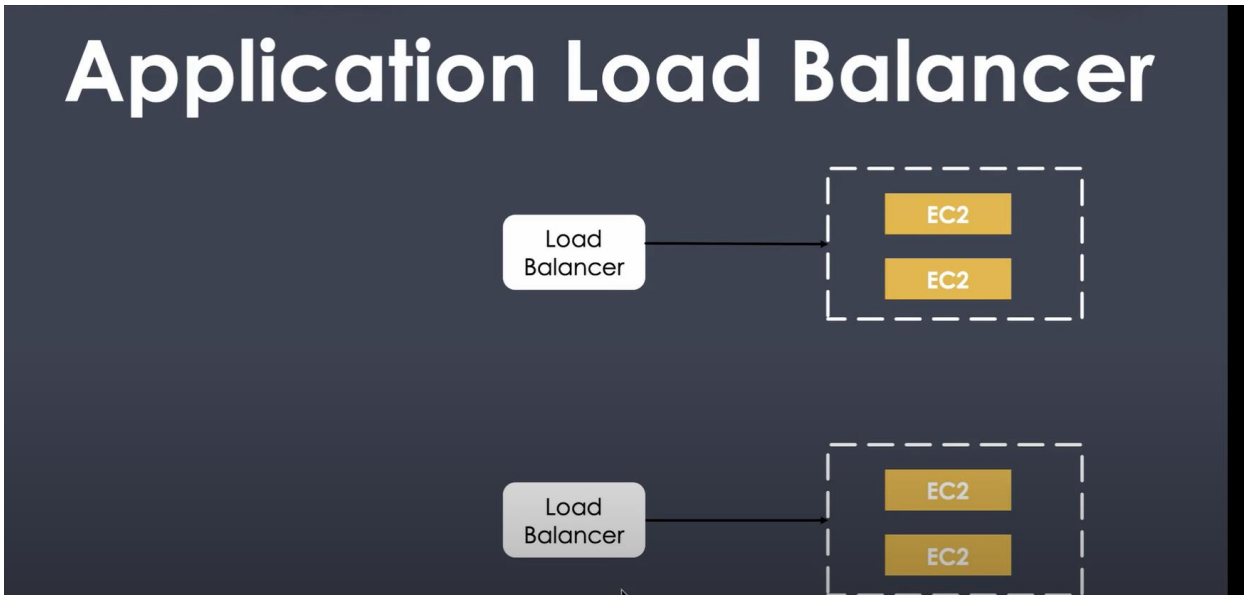
Conclusion

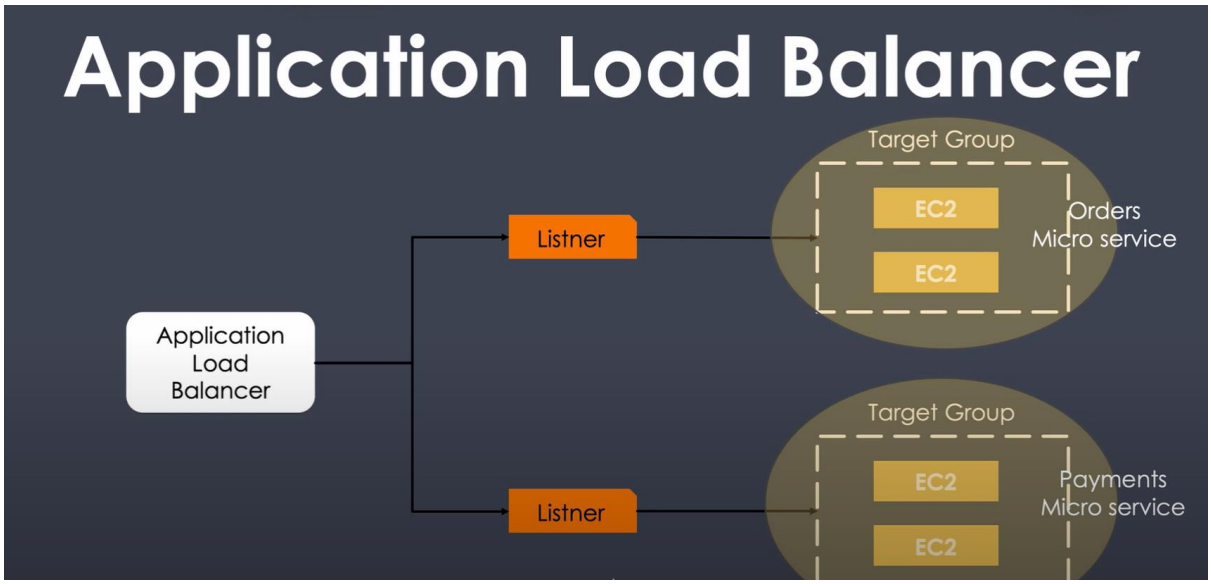
Hence, in AWS ELB tutorial, we studied the load balancer distributes traffic evenly across the Availability Zones that you enable for your load balancer. AWS Elastic Load Balance is one of the effective ways to manage and deploy the application as it is fast, reliable, and efficient.

Load Balancer simplifies and improves the security of your application and hence provides security which makes it a reliable function. Furthermore, if you have any doubt regarding AWS ELB, feel free to ask in the comment box.

2. AWS ELB setup

Application Load Balancer setup:





Listener make routing decisions like when to route traffic to orders and when to route traffic to Payments.

Routing decisions are made based on the path or port (rules)

Steps:

Step1: Create Two instances: Orders & Payments

- Create an Ec2 instance with Linux image.
- Durind the instance creation, use the below code in the User instructions to create an html file in Advanced Settings→ User data for Orders.

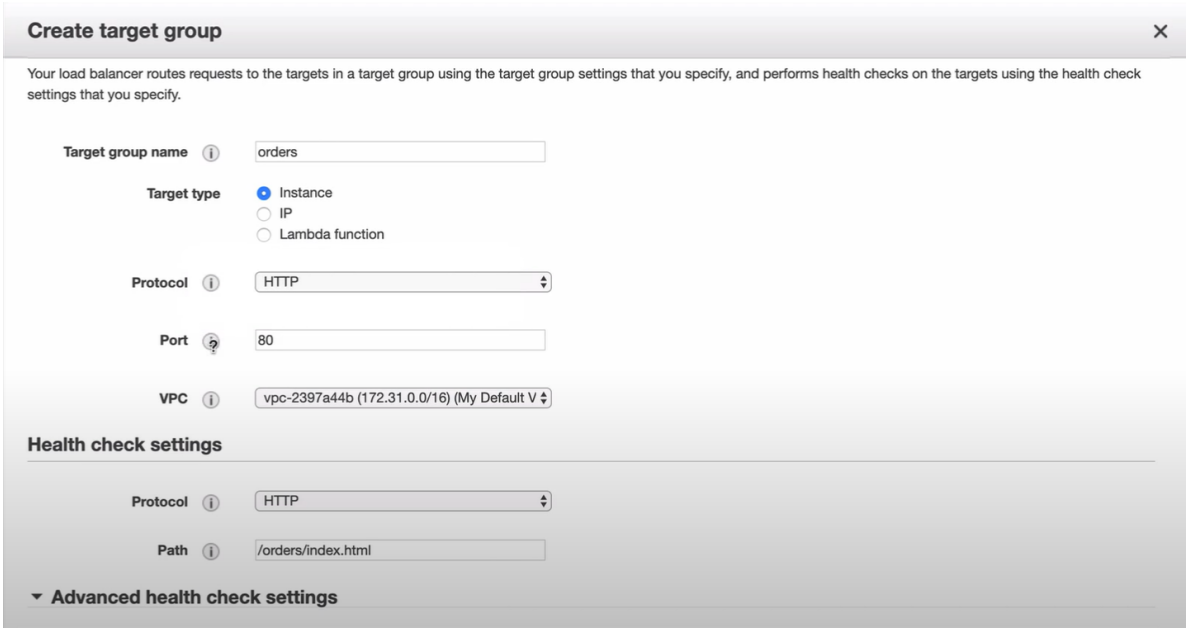
```
apache.sh
#!/bin/bash

yum install httpd -y

systemctl enable httpd

mkdir /var/www/html/orders/

echo "<h1>This is Orders App</h1>" > /var/www/html/orders/index.html

systemctl start httpd
```

- Launch instance
- Similarly, launch instance for Payments application by giving the user data as shown below.

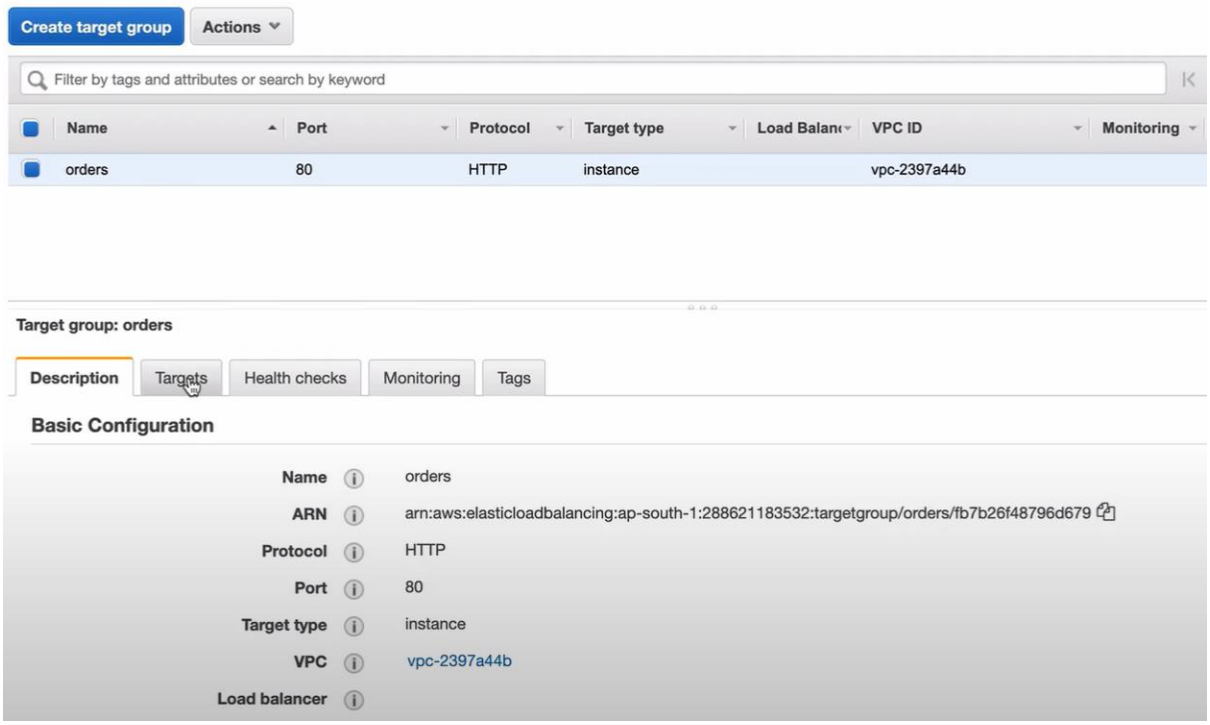
```
apache.sh
#!/bin/bash

yum install httpd -y

systemctl enable httpd

mkdir /var/www/html/orders/

echo "<h1>This is Orders App</h1>" > /var/www/html/payments/index.html

systemctl start httpd
```

Step2: Target Group creation

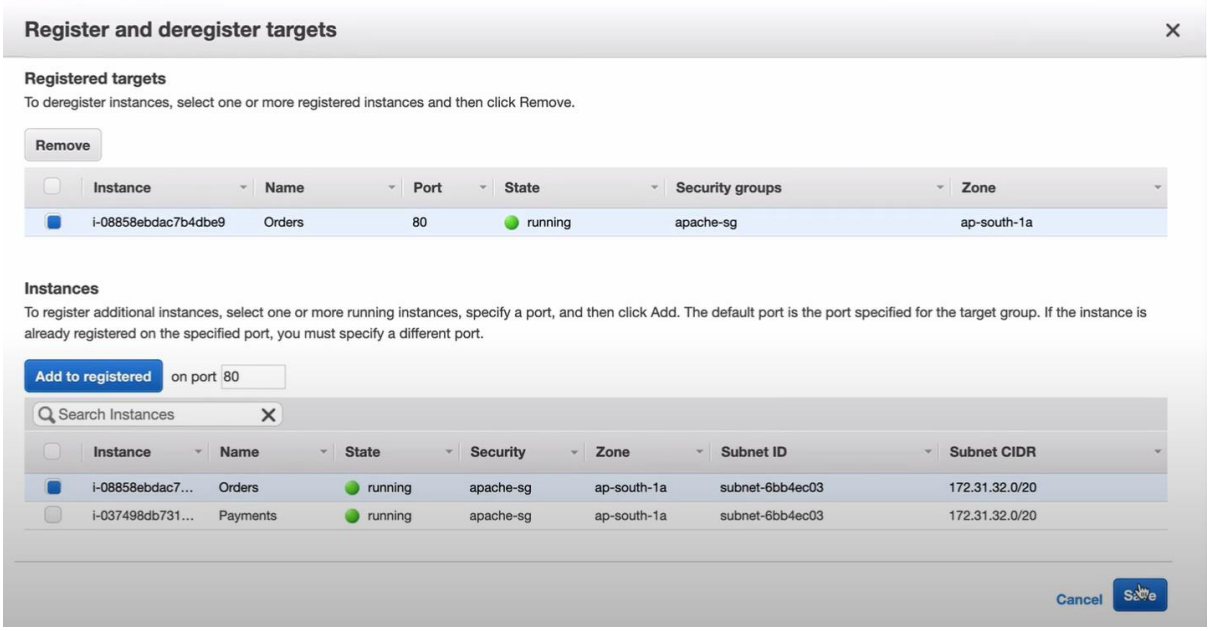
- Choose Load Balancing→ Target groups from AWS console Left Pane and **Create**
- Enter Details like Target group name as required, Target Type as **Instance**, Protocol as **HTTP,** port as **80,** VPC as required.
- Under Health check settings, update path as **"/orders/index.html**" path of the application and click **Create**



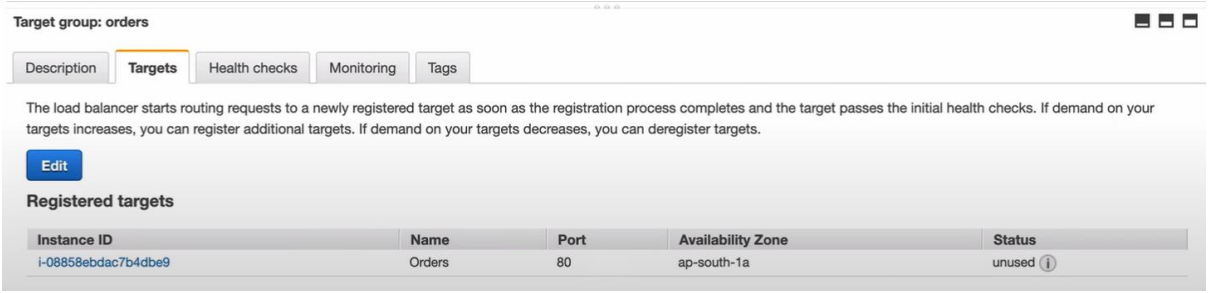
- Create another Target group for Payments Micro-service and update the Path as **"/payments/index.html"**
- Once the Target groups are created, associate the Targets for the Target group.



- Click **Edit** & register the instances as Target in the Target group & Save

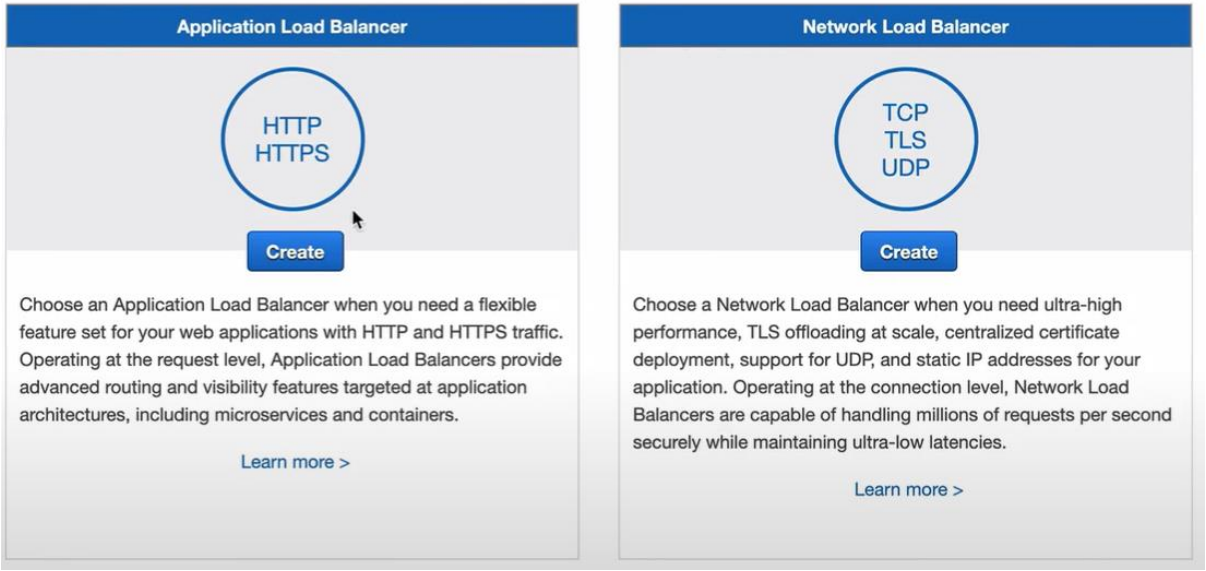


- Similarly, register the "Payments" instance as Target for "Payment" Target group
- You can check the status of Target groups

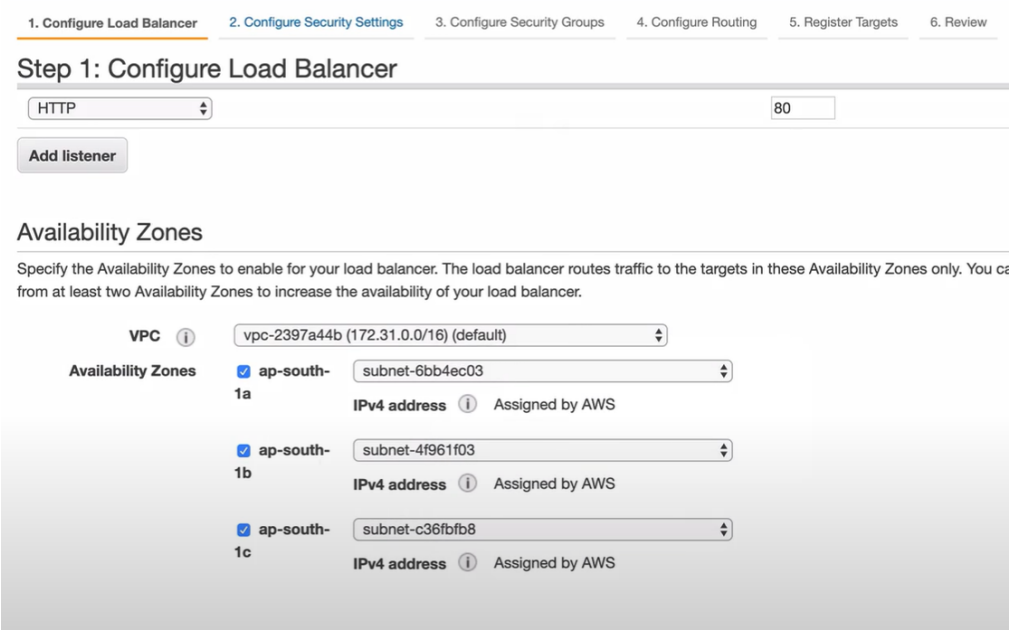


Step3: Load Balancer Creation

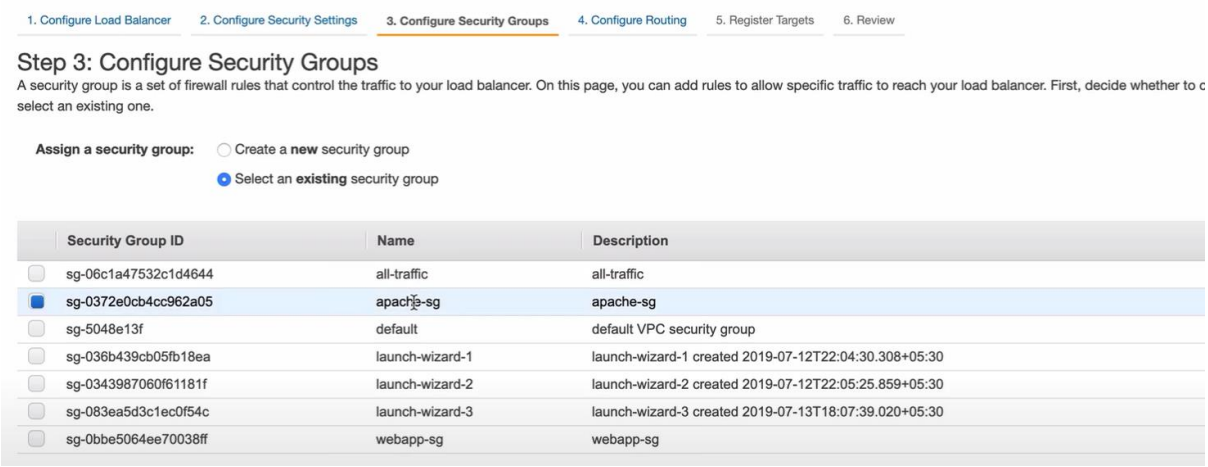
- Select Load Balancing→ Load Balancer from AWS Console LP & **Create Load Balancer**



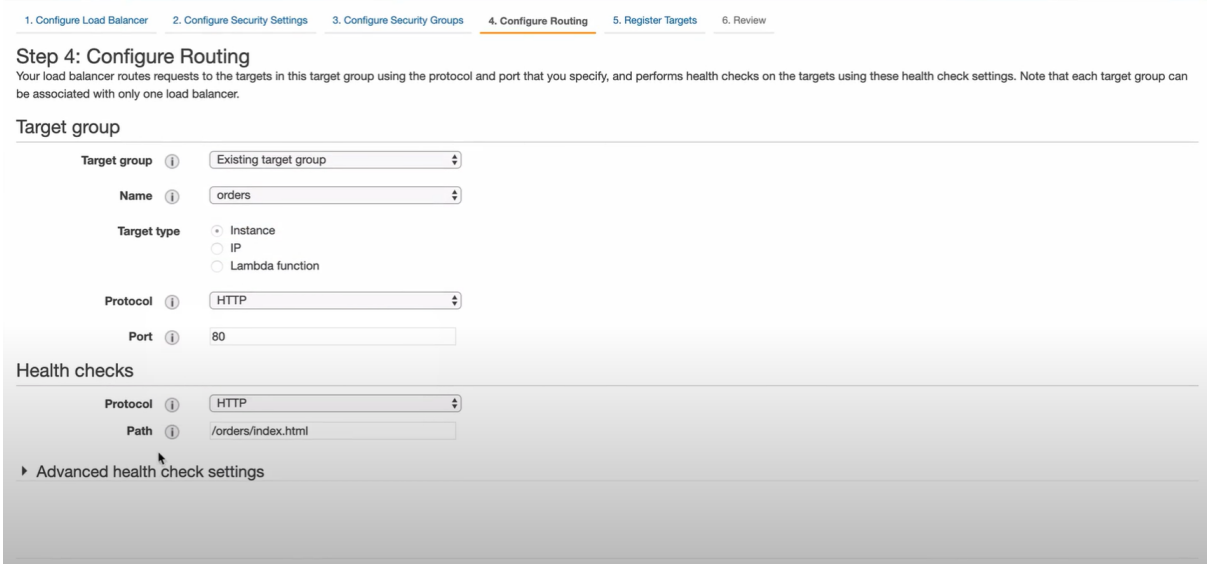
- In **Configure Load Balancer**, Enter as Required, Scheme as **Internet-facing,** IP address type as IPv4, Select default VPC from Availability zones and select the availability zones as required.
  Note: The instances must reside on one of the availability zones listed.



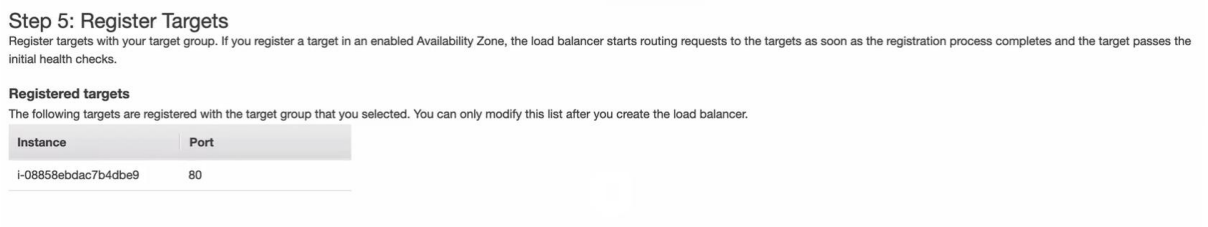
- **Configure Security group** as required



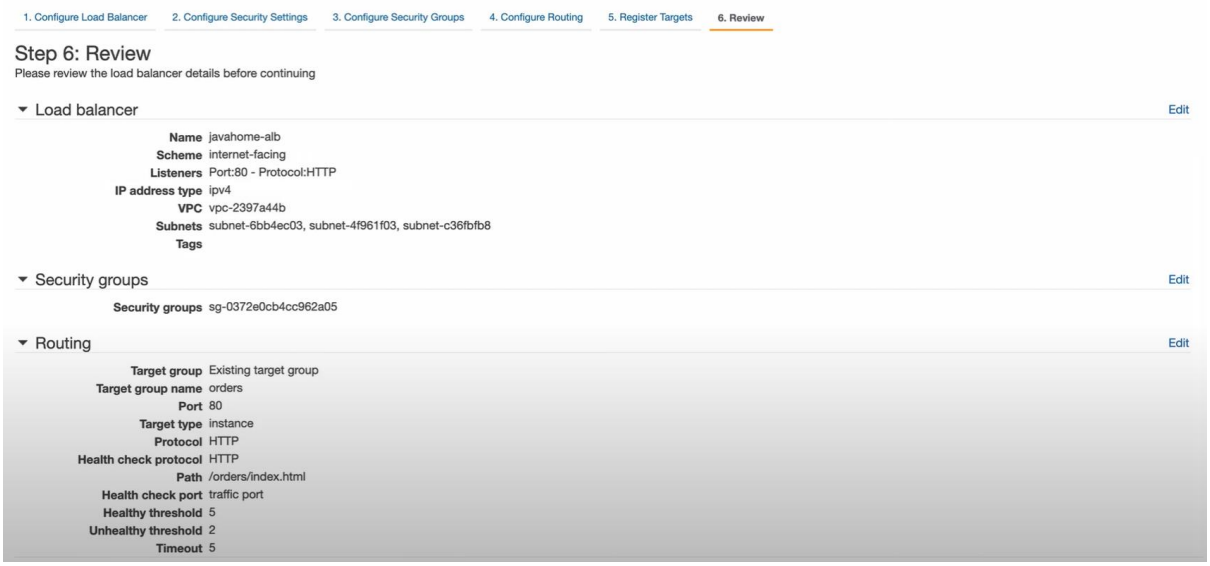
- **Configure Routing**, Select the Target group created for **Orders.**



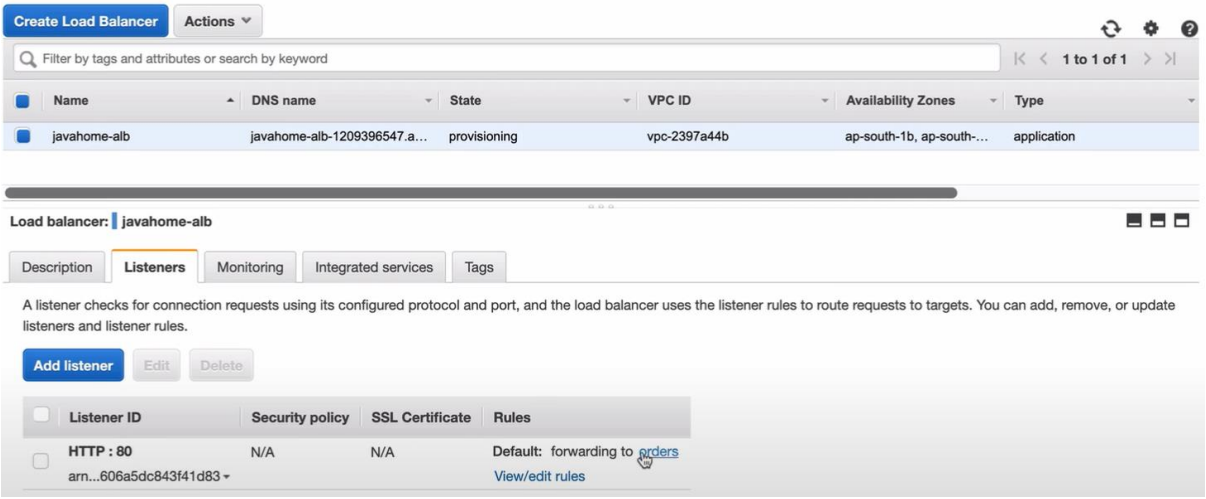
- **Register Targets**



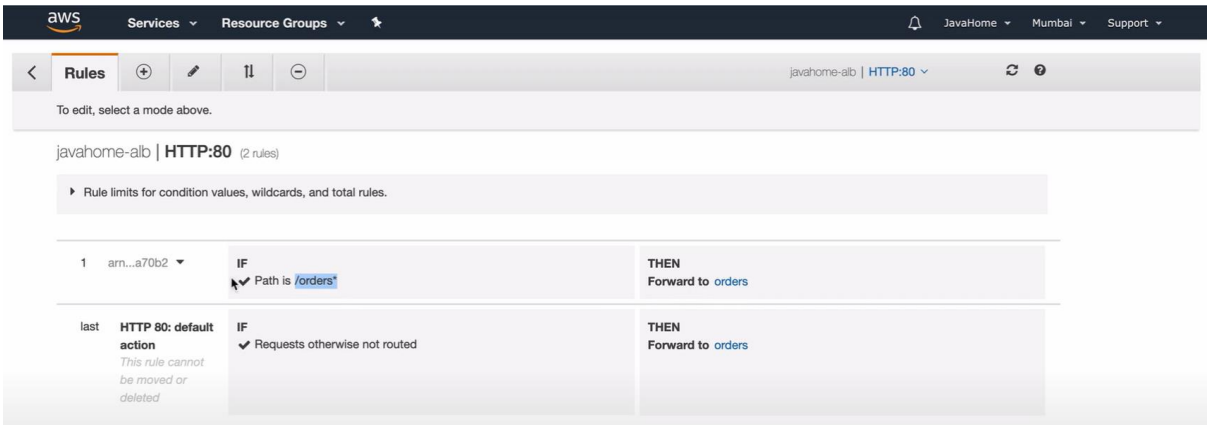
- **Review & Create**



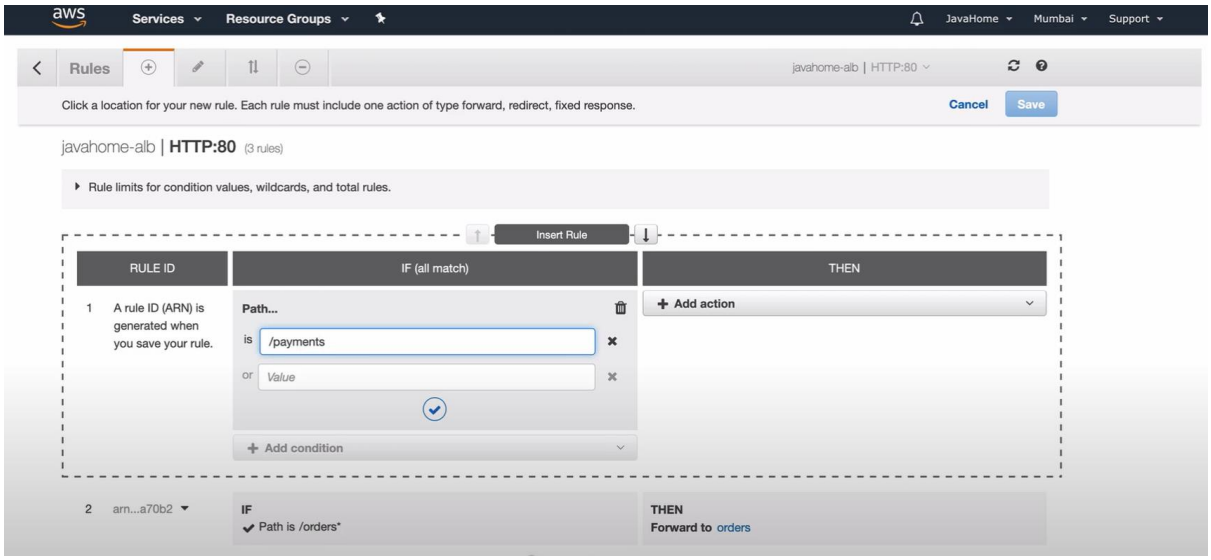
- Go to **Listeners**, it will have a default Listener and Rule



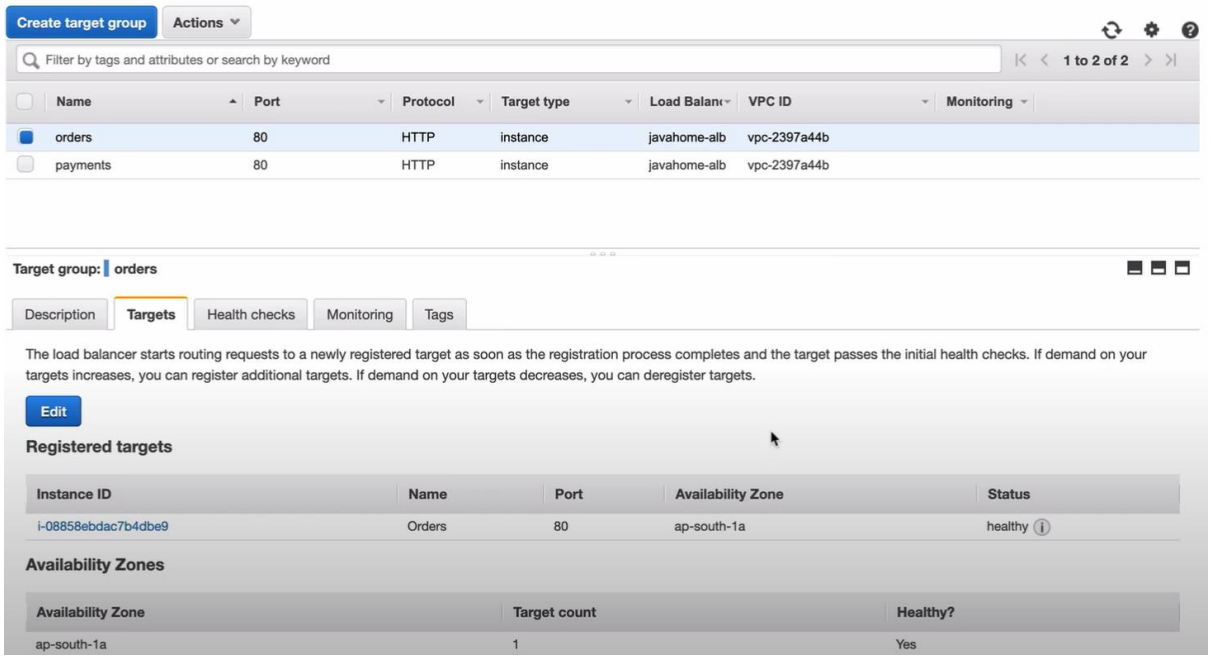
- Edit the rule to forward to Target group- Orders if path is **/orders/***



- Add another rule to forward to Target group- Payments if path is **/payments/***



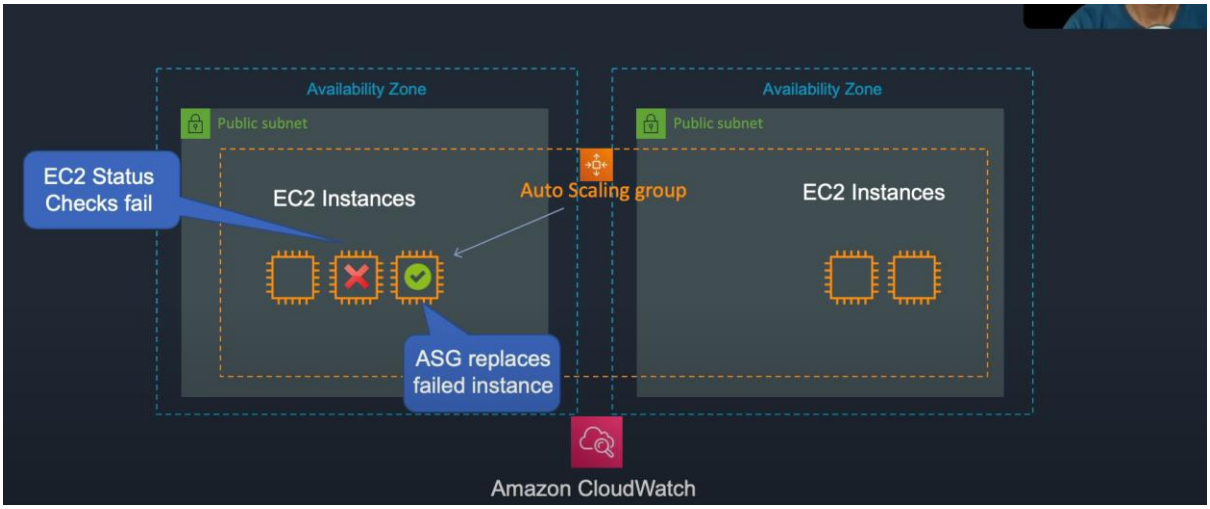
- Now,check the Target group status, it would be updated as Healthy



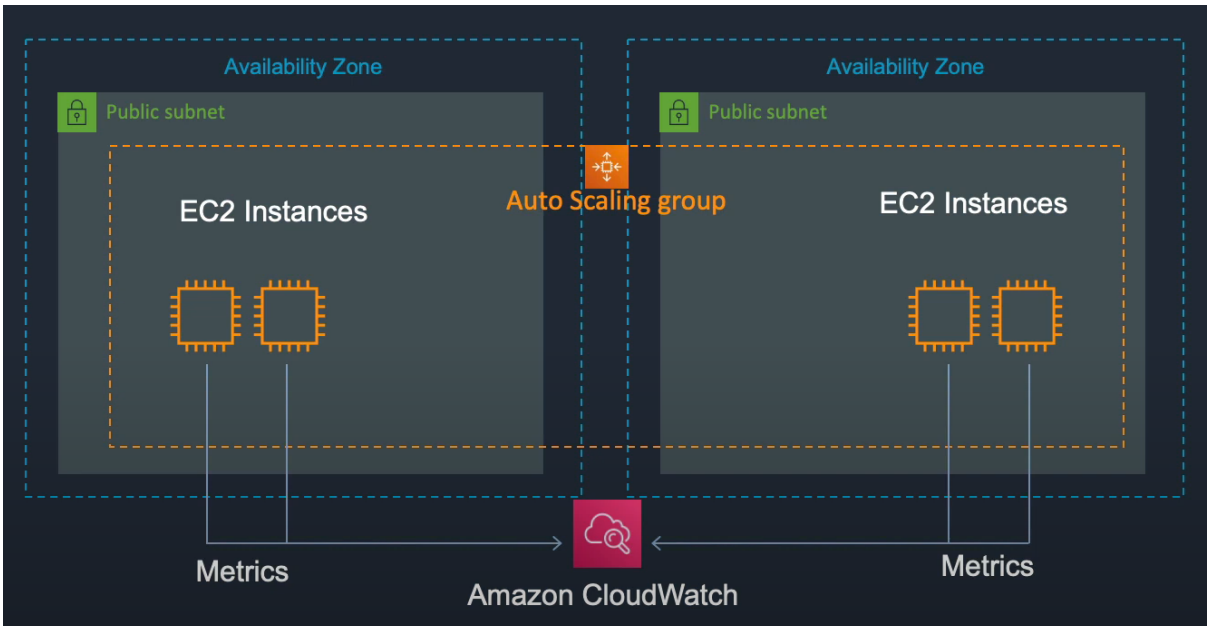
3. AWS Scaling-group setup

Working:

- Based on the Health checks

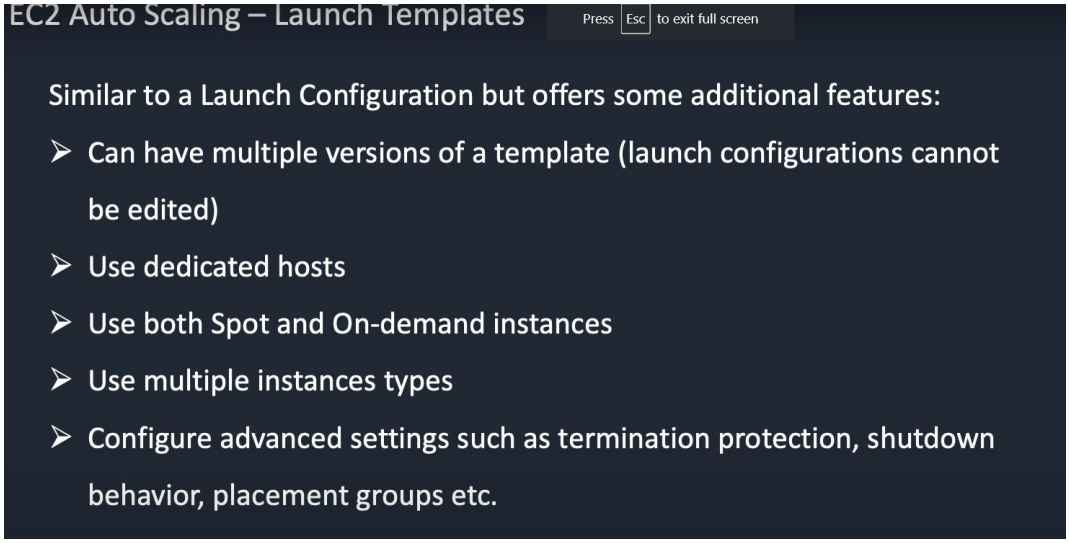


- Based on the Metrics using CloudWatch



Launch Templates:

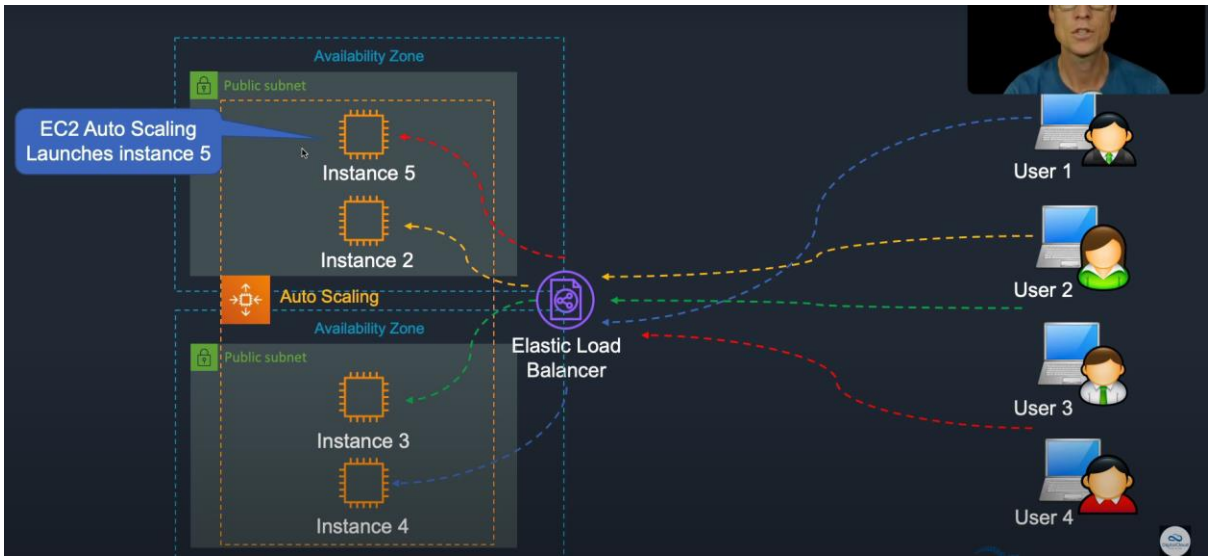
- Features:



Steps:

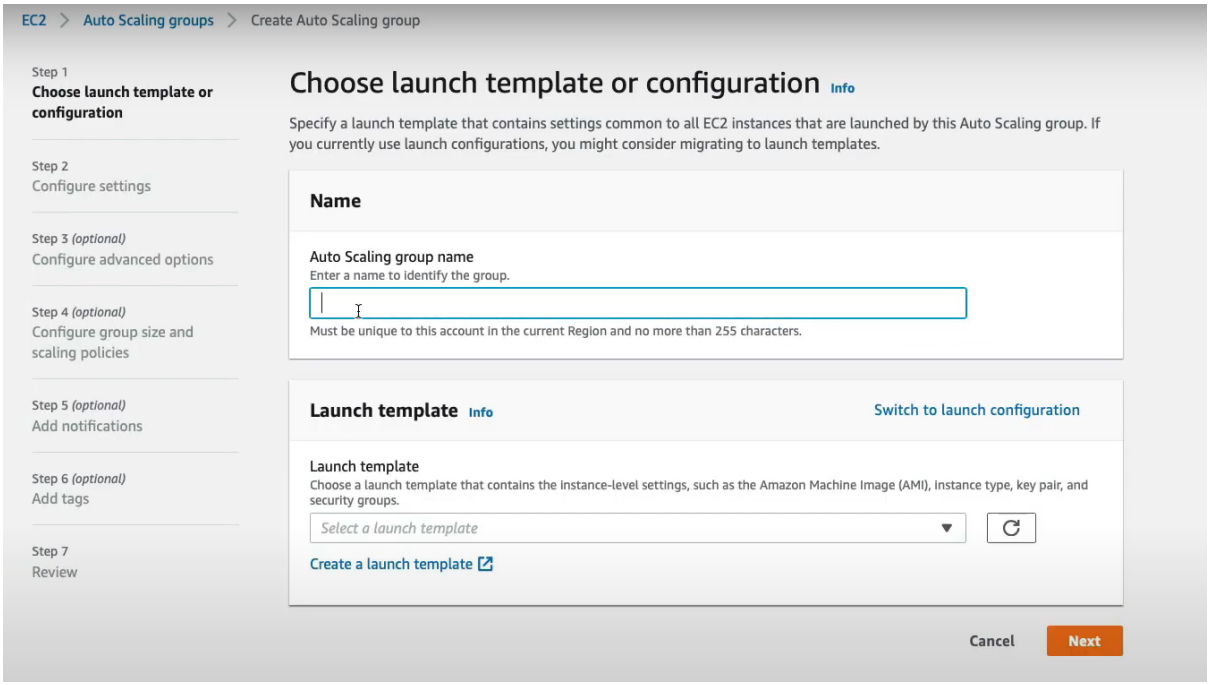
Step1: Create Image of an instance

- Create an Image by selecting an EC2 instance, Actions→Image→ Create Image
- Then Select AMI from AWS LP to see the image being created.
- Create ELB with Target group but need not add any Targets in the Target group as it will be taken care as a part of Auto-scaling group
- The ELB should be pointing to Public subnets available in the different availability Zones

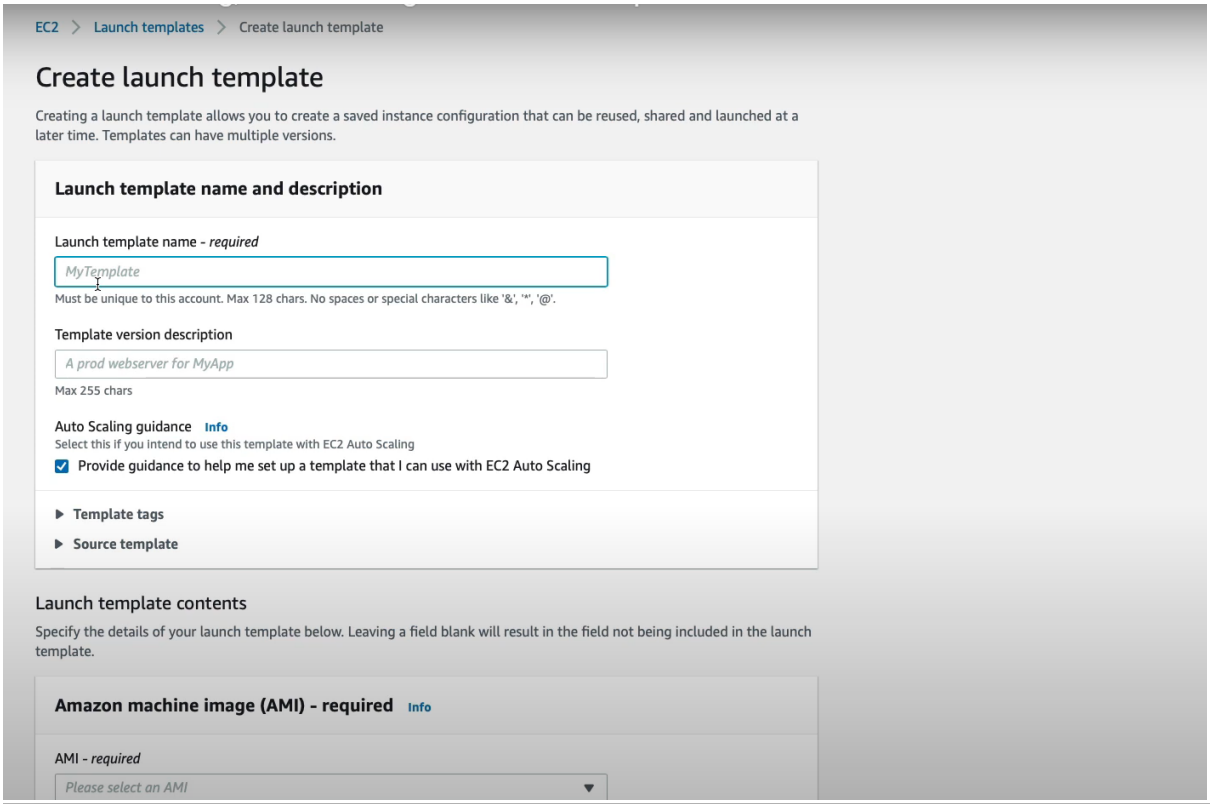


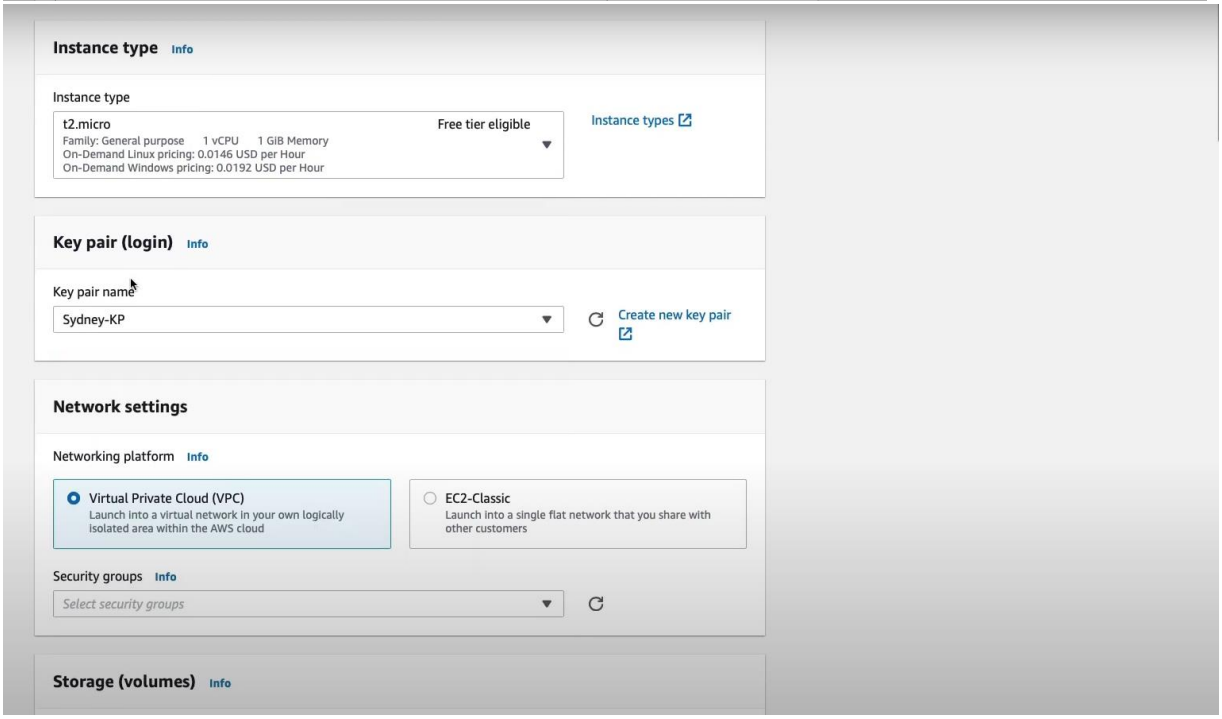
Step2: Create Autoscaling group using Launch template

- Select the Auto Scaling→ Auto Scaling Group from AWS LP
- Click Create Auto-scaling group
- Enter Name of the ASG and click **Create a launch template**

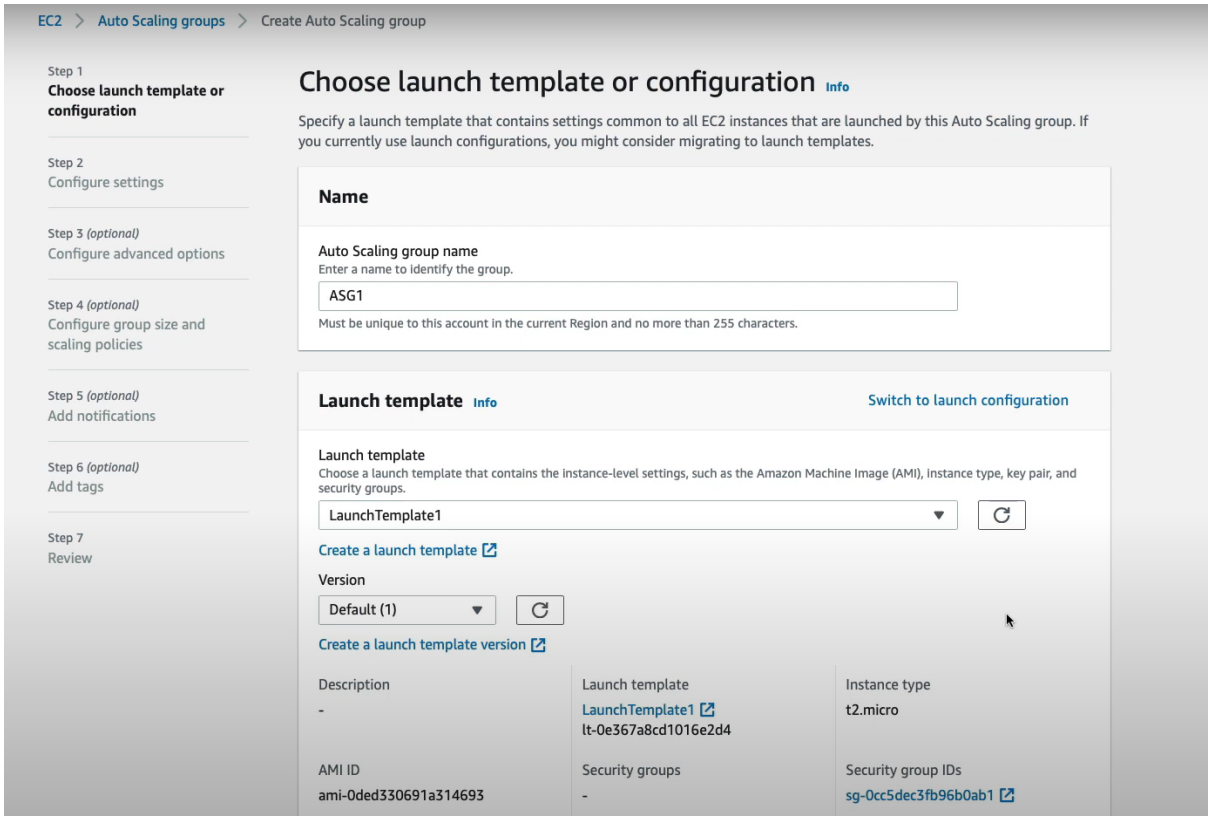


- Enter Launch template name as required, in select AMI→ Select the Image created from EC2 Instance, Instance Type, Key Pair, Select VPC, Security group, IAM Instance profile, and then click Launch Template.

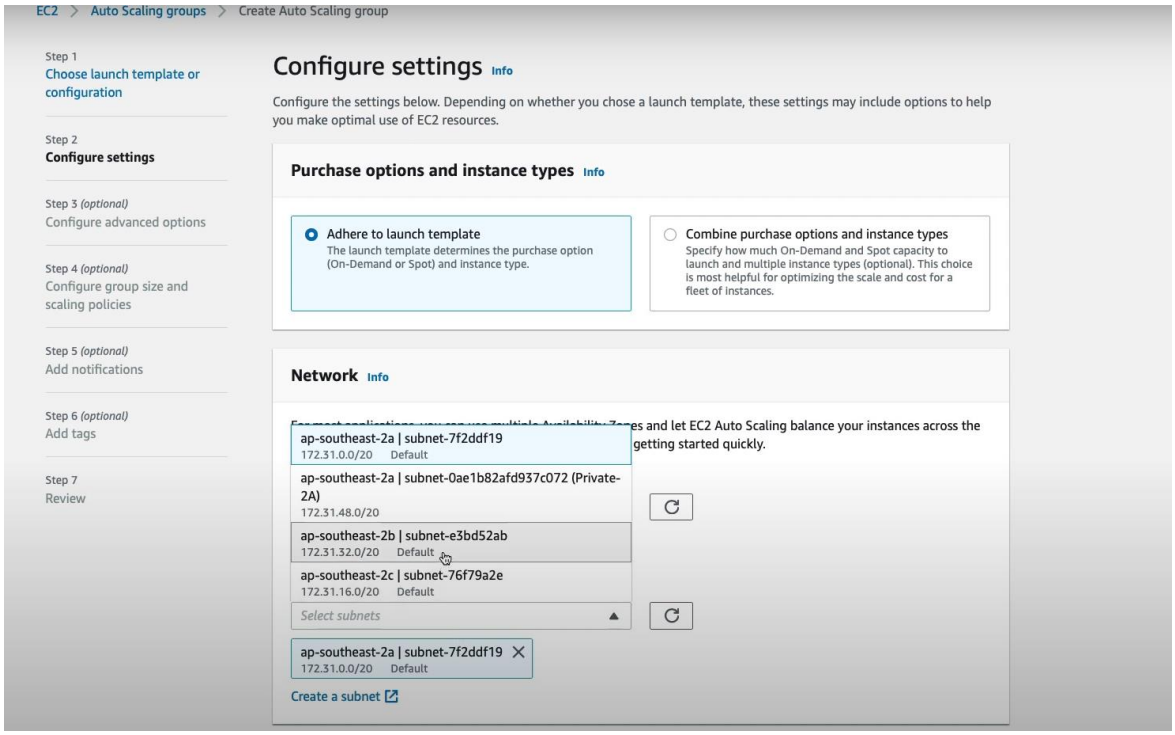




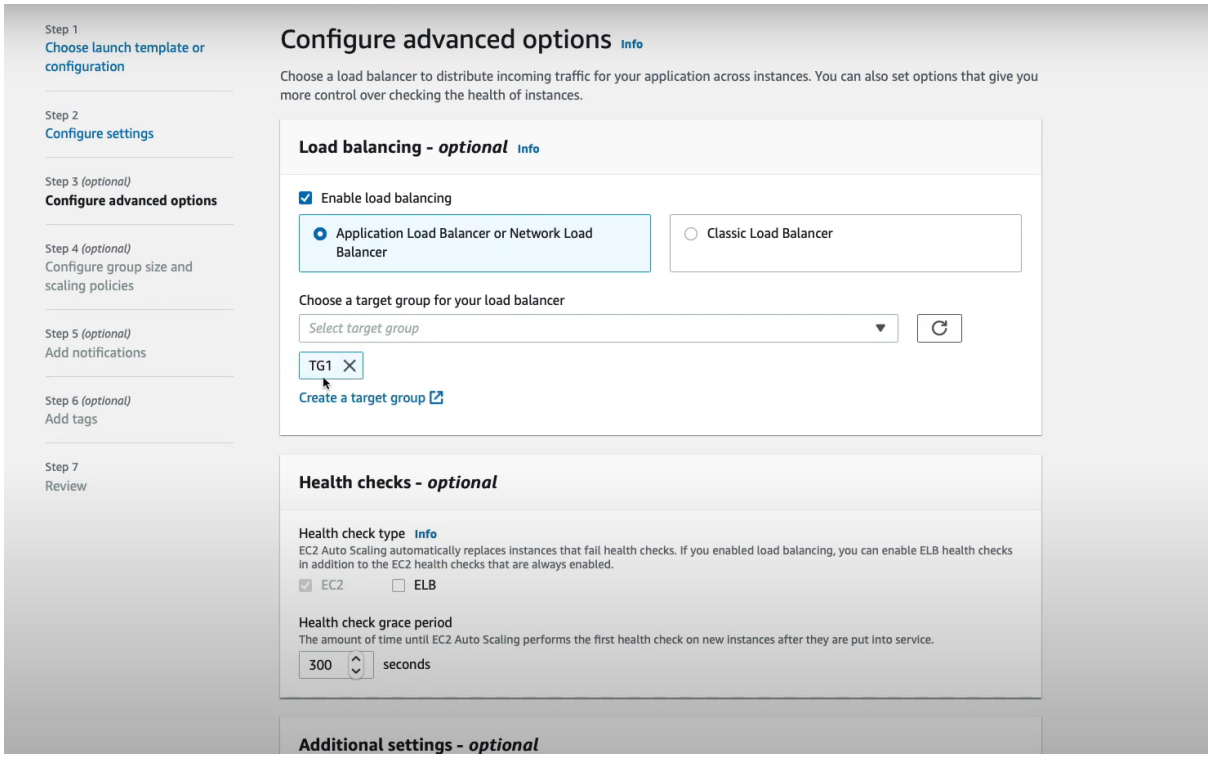
- Go back one step, choose the newly create Launch template and click Next



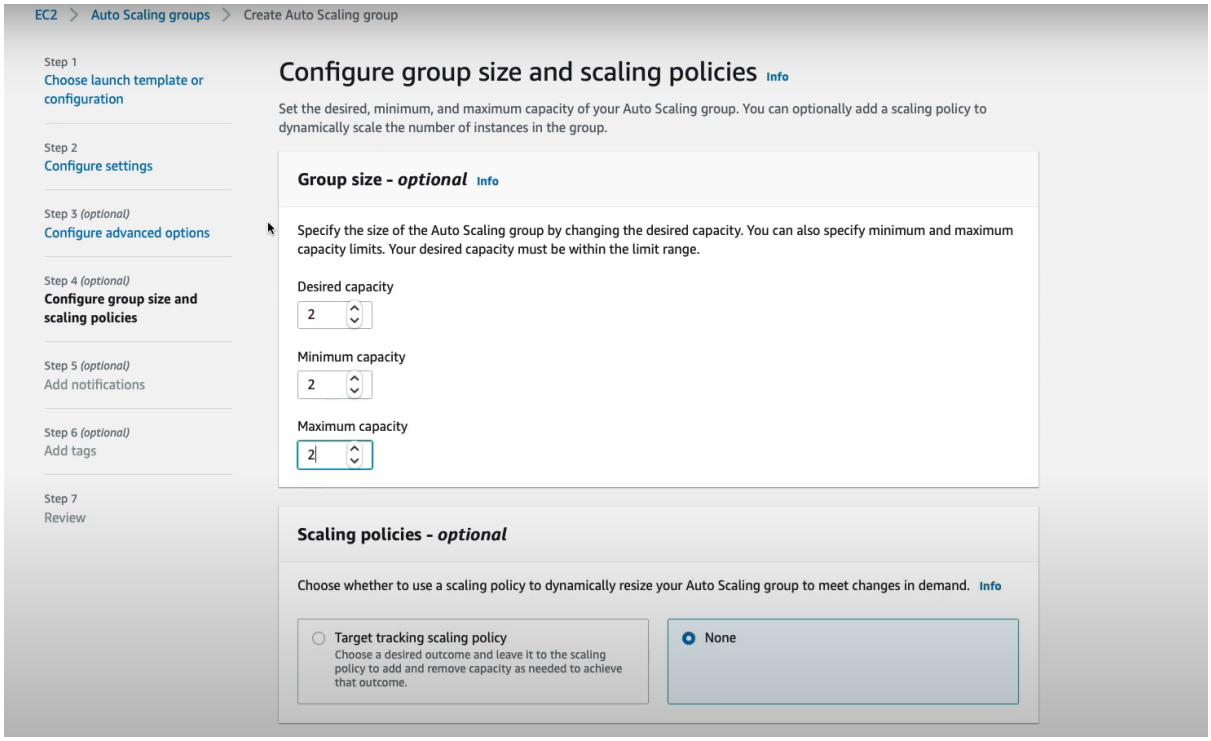
- You can choose Instances distribution in **Configure settings** & choose the Public subnets from different availability zones from the VPC



- Attach the Target group created in the **Configure Advanced** options.

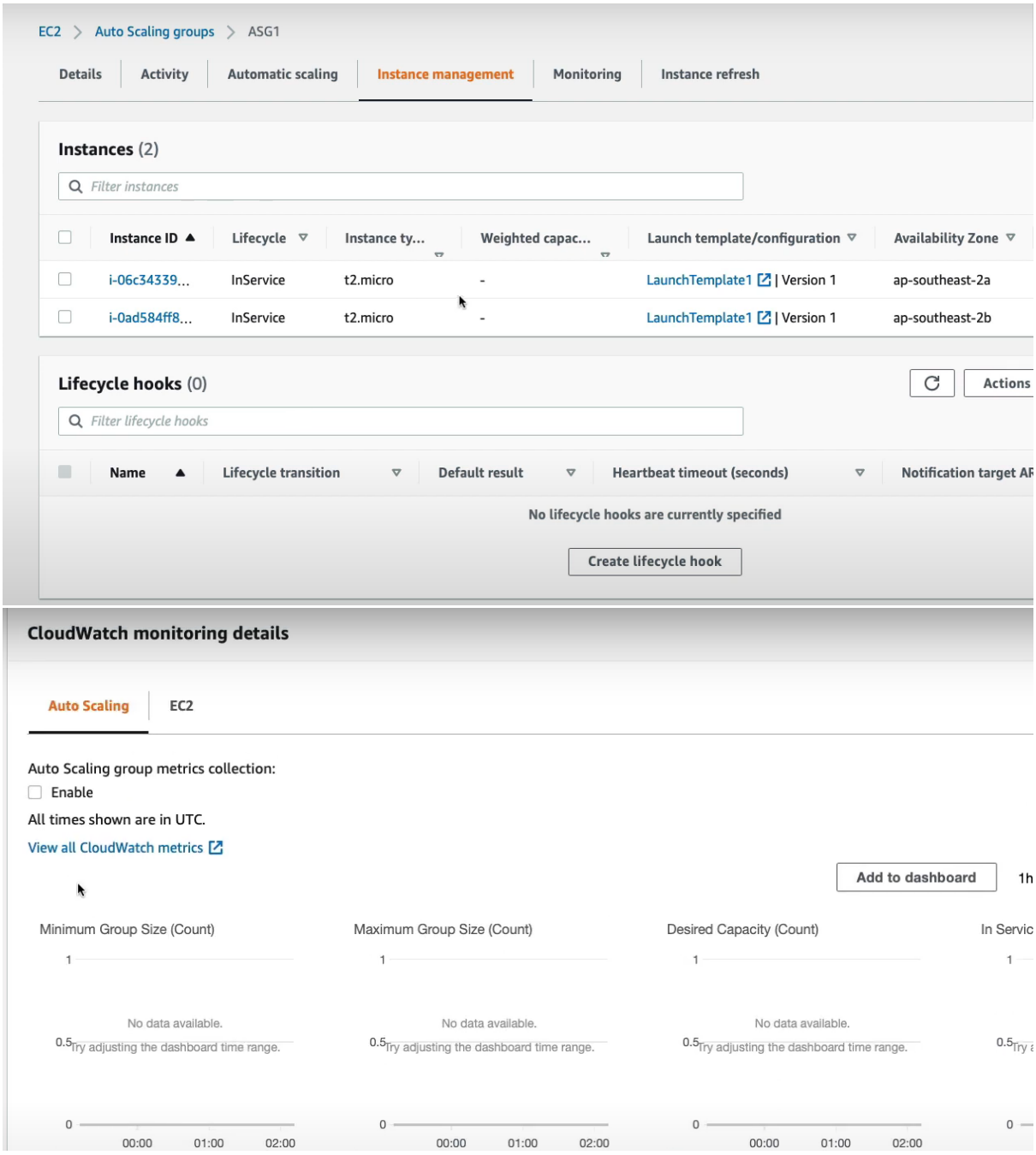


- You can enter the Desired, Minimum and Maximum scaling capacity for the Auto Scaling group.

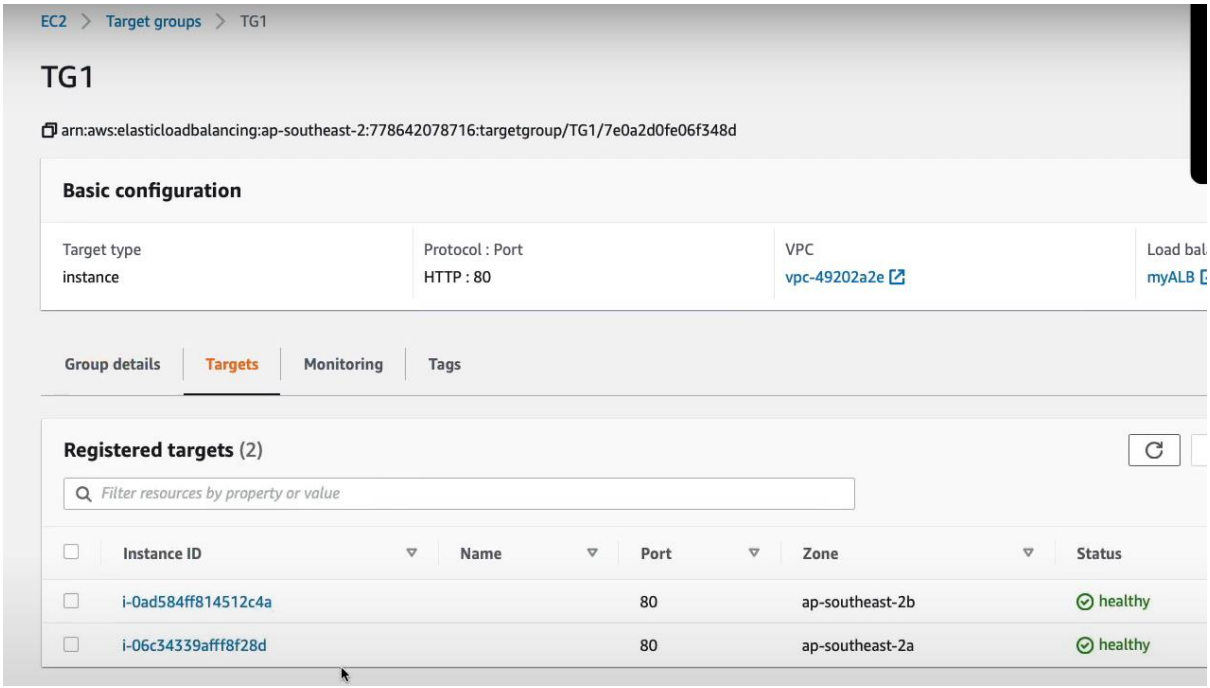


- You can add notifications, Tags if required else click Next until you get option to Create Autoscaling group.

- Once this is done, open the Auto-scaling group to view the details



- If you go to Target groups, you can see the Targets created in the target group automatically.



4. AWS EKS Setup

- Setup kubectl
  a. Download kubectl version 1.20
  b. Grant execution permissions to kubectl executable
  c. Move kubectl onto /usr/local/bin
  d. Test that your kubectl installation was successful

  curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl

  chmod +x ./kubectl

  mv ./kubectl /usr/local/bin

  kubectl version --short --client

- Setup eksctl
  a. Download and extract the latest release

b. Move the extracted binary to /usr/local/bin
c. Test that your eksclt installation was successful

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp

sudo mv /tmp/eksctl /usr/local/bin

eksctl version

- Create an IAM Role and attache it to EC2 instance
  Note: create IAM user with programmatic access if your bootstrap system is outside of AWS
  IAM user should have access to
  IAM
  EC2
  VPC
  CloudFormation

- Create your cluster and nodes
  eksctl create cluster --name cluster-name \
  --region region-name \
  --node-type instance-type \
  --nodes-min 2 \
  --nodes-max 2 \
  --zones <AZ-1>,<AZ-2>

  example:
  eksctl create cluster --name valaxy-cluster \
  --region ap-south-1 \
  --node-type t2.small \

- To delete the EKS clsuter
  eksctl delete cluster valaxy --region ap-south-1

- Validate your cluster using by creating by checking nodes and by creating a pod
  kubectl get nodes
  kubectl run pod tomcat --image=tomcat

5. AWS ROUTE53 DNS Records

It's a global service. You need to buy a domain in order to work with Route53, Go to Route53 Service & Click on register domain. Enter the domain name & check availability, Add to cart & click on continue.
Route53 can use basically:
Public domain names you own (or buy) or Private domain names that can be resolved by your instances in your VPCs.
Route53 has many features such as **Load balancing, Health checks, Routing policy like Simple, Failover, Geolocation, Latency, Weighted, Multi value.**
You pay $0.50 per month per hosted zone.

In AWS Route53, we have many types of records. Let's talk about various records.

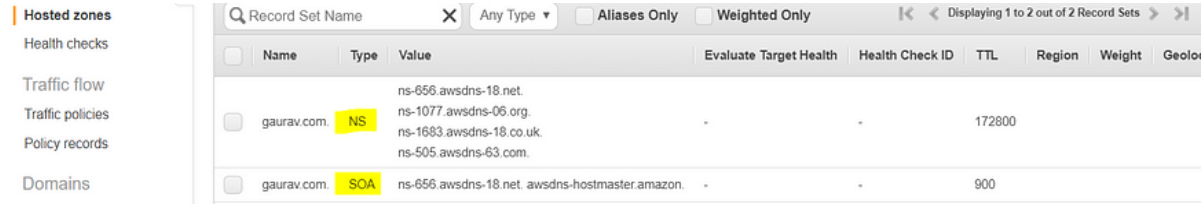
1- SOA (Start of Authority Records)

Basic SOA stores information about below things.
Name of Server that supplied the data for zone.
The administrator of that zone & current version of data file.
Eg:
ns-2048.awsdns-64.net. hostmaster.example.com. 1 7200 900 1209600 86400
Route53 Name server that create SOA record: ns-2048.awsdns-64.net.
Email Address of Administrator: *hostmaster.example.com*

2- NS Record (Name Server Records)

NS records is basically your name server records which are used by top level domain servers to direct traffic to content DNS server which contains the authoritative records.

So whenever we create a hosted zone in Route53, Two types of records automatically created, one is SOA & second is NS.

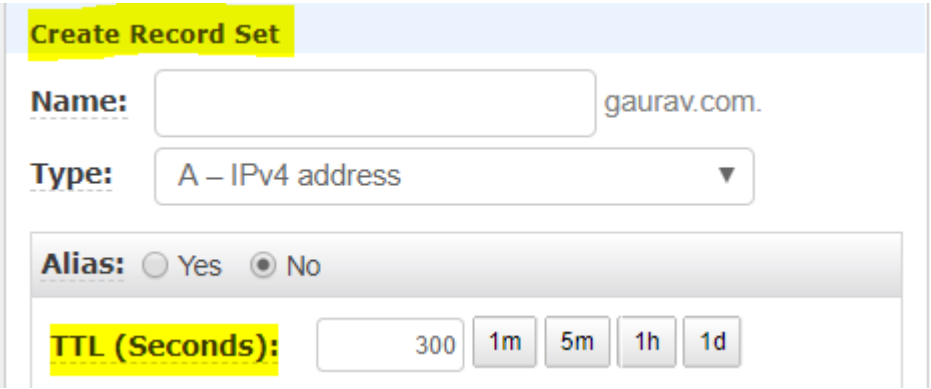


Hosted Zone(SOA & NS Records)

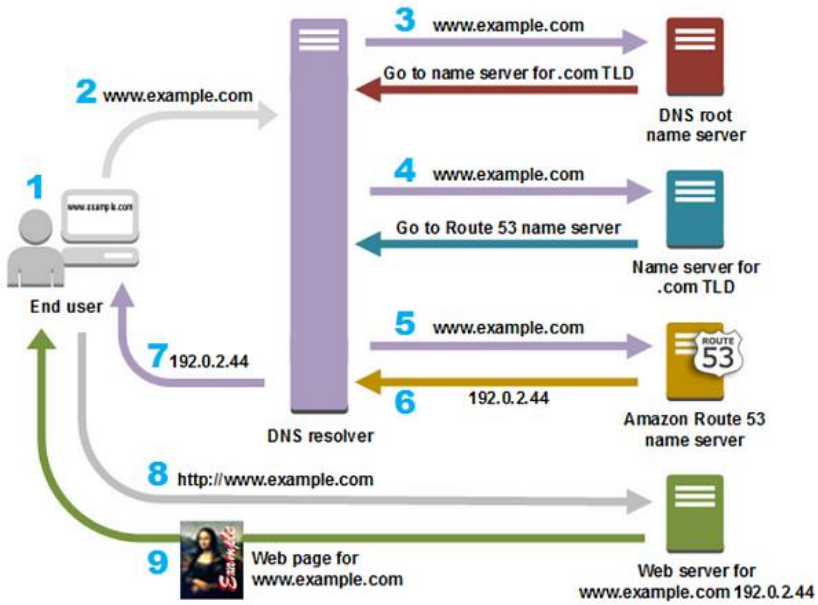
Before we head over to important type of records used in AWS, let's talk about one important concept TTL(Time to Live)

TTL (Time to Live):

TTL is mandatory for each DNS record. So TTL is length that a DNS records is cached on either the resolving server or user own Laptop. The Lower the TTL, the faster changes to DNS records. Whenever you created record set, you need to define TTL for it.



TTL(Time to Live)



How Amazon Route 53 Routes Traffic for Your Domain (Source AWS Official)
Let's talk about the most common records in AWS.

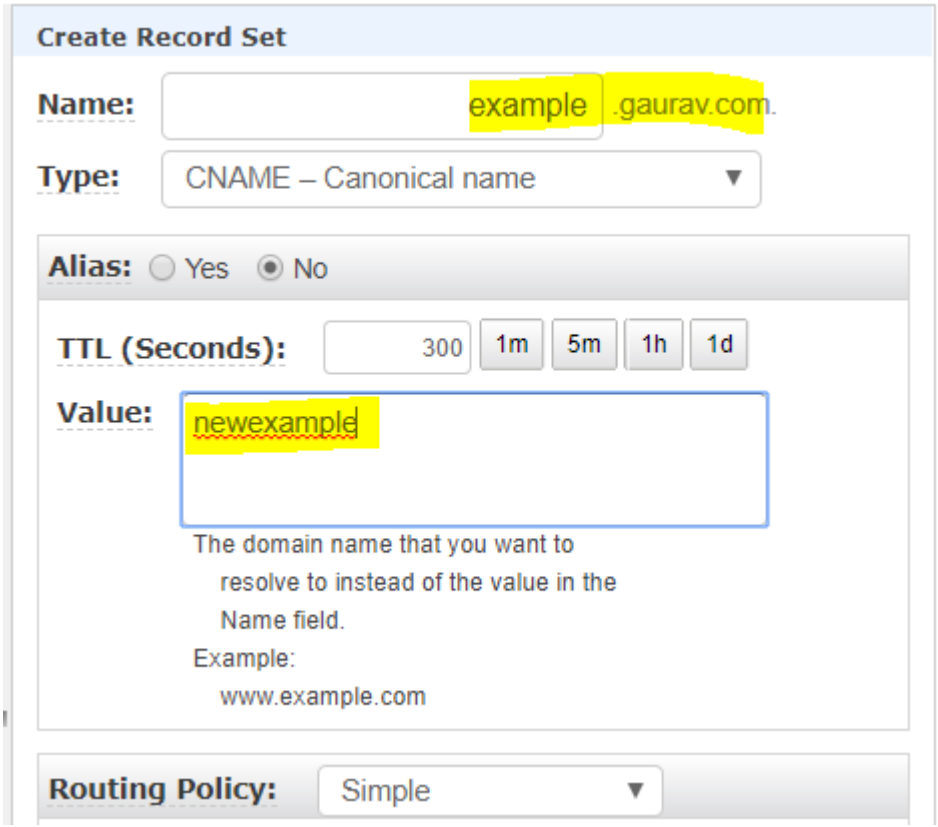
When you create basic records, you specify the following values.
Name
Type
Alias
TTL (Time to Live)
Value
Routing Policy

1- A Record (URL to IPv4)

The "A" record stands for Address record. The A record is used by computer to translate the name of the domain to an IP address.
Eg: (http://medium.com might point to http://126.78.98.90)

2- CNAME (Canonical Records- URL to URL)

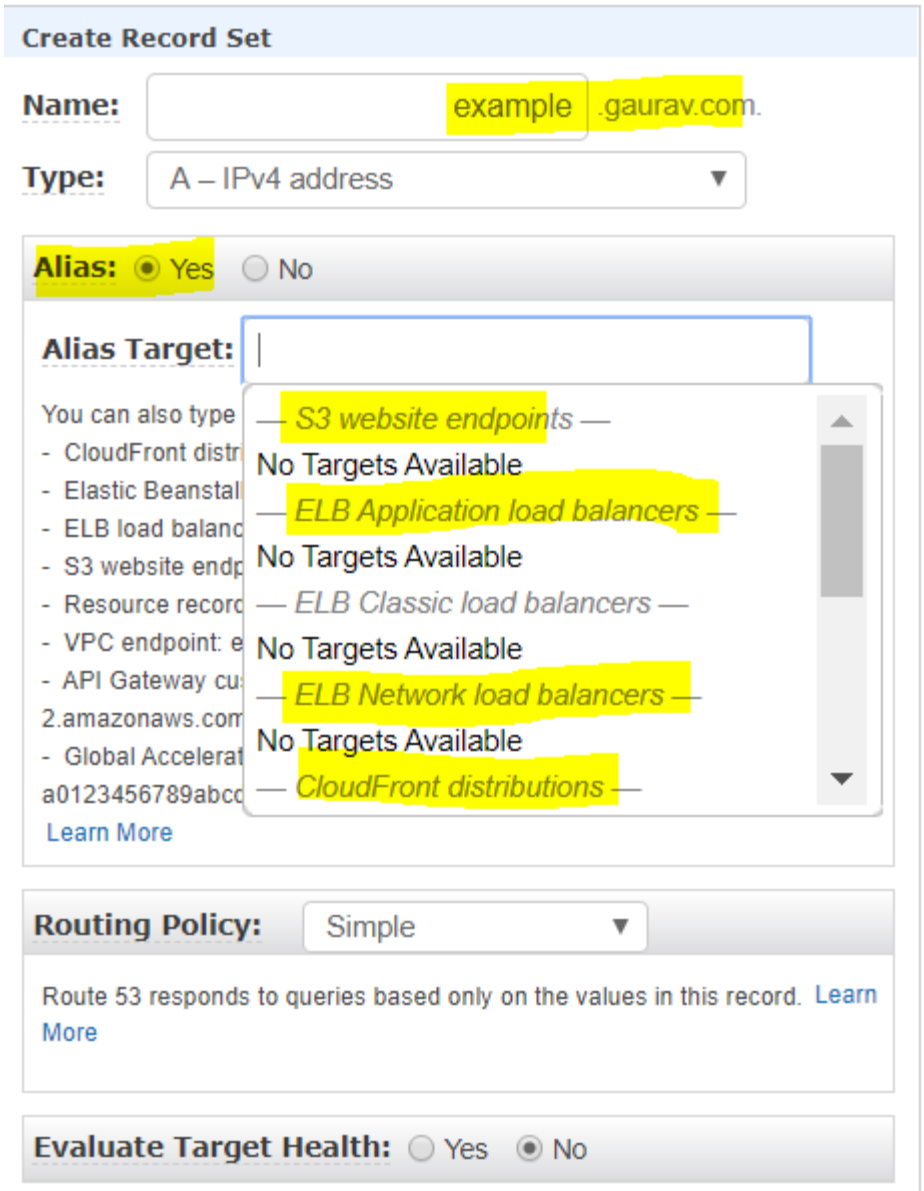
CNAME Points a URL to any other URL. (gaurav.gupta.com => gkg.example.com), We use it only for Non-Root Domain(aka. something.mydomain.com)



Resolving example.gaurav.com to newexample.gaurav.com

3- Alias Record:

Alias record points a URL to an AWS Resource, Alias record are used to map resource record sets in your hosted zone to Elastic Load Balancer, CloudFront or S3 Buckets websites.



Alias Record

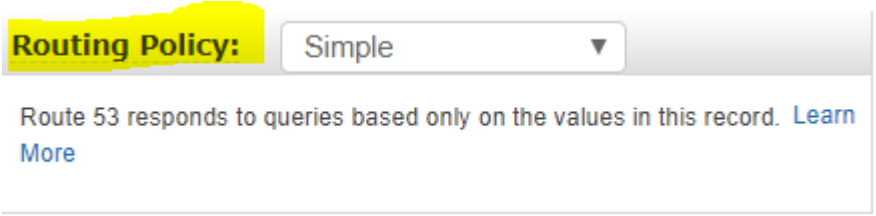
4- AAAA: (URL to IPv6)

An **AAAA record** maps a domain name to the IP address (Version 6) of the computer hosting the domain. An **AAAA record** is used to find the IP address of a computer connected to the internet from a name.

5- MX Record (Main Exchange Record)

A mail Exchanger **record** (**MX record**) specifies the mail server responsible for accepting email messages on behalf of a domain name. It is a resource **record** in the Domain Name System (**DNS**). It is

possible to configure several **MX records**, typically pointing to an array of mail servers for load balancing and redundancy.
This is all about various type records, let's talk about Routing policy.
So what is routing policy.
When you create a record, you choose a routing policy, which determines how Amazon Route 53 responds to queries.

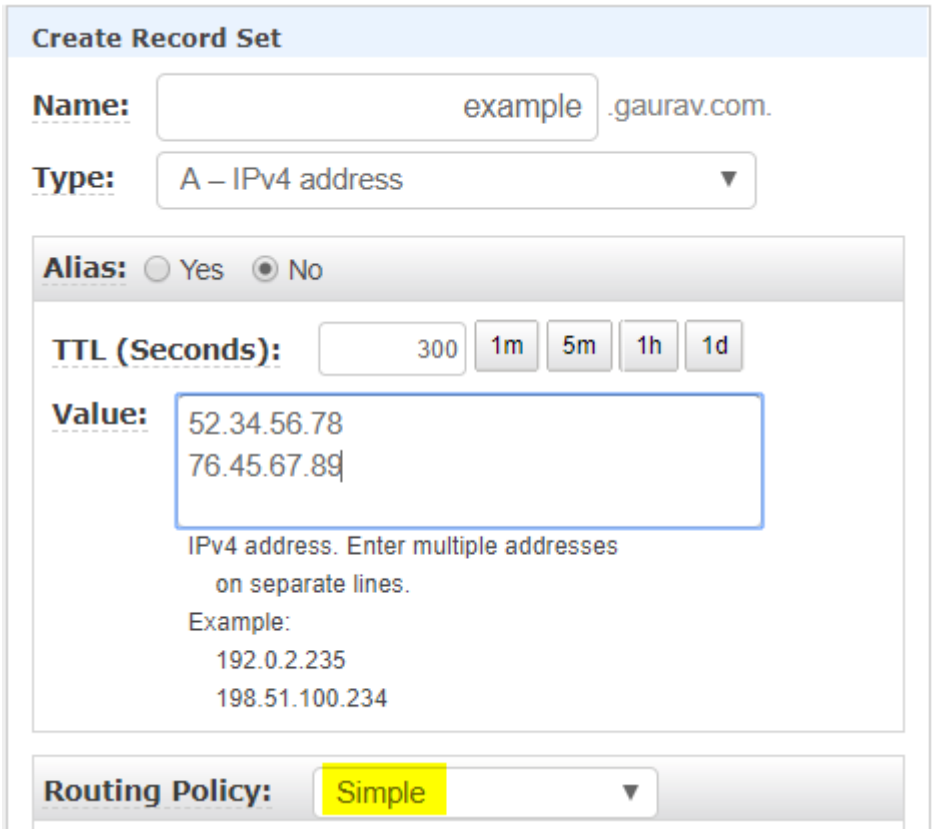


6. Routing Policy

There is total 6 types of routing policy in Route53, let's talk about one by one.

1- Simple Routing Policy:

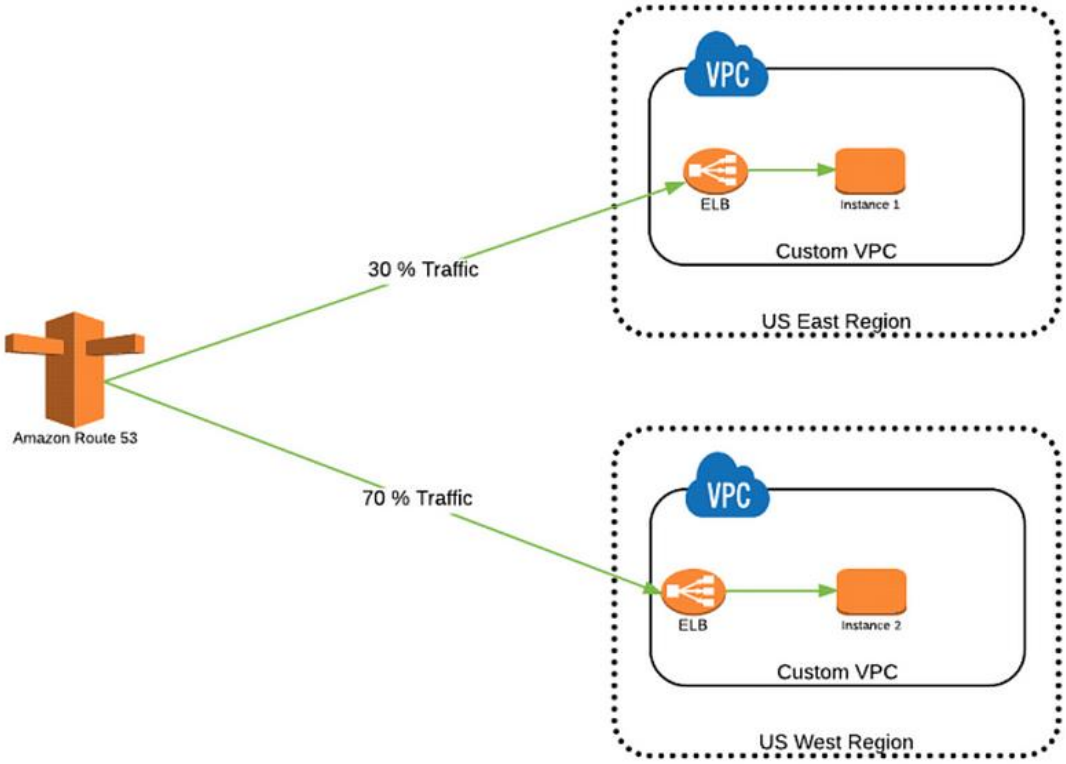
In case of simple routing policy, you can have only one record with multiple IP addresses. If you specify multiple values in record, Route53 returns all values in random order to the user.
Maps a domain to one URL, Use when you need to redirect to a single resource. You can't attach health checks to simple routing policy. If multiple values are returned, a random one is chosen by the client.



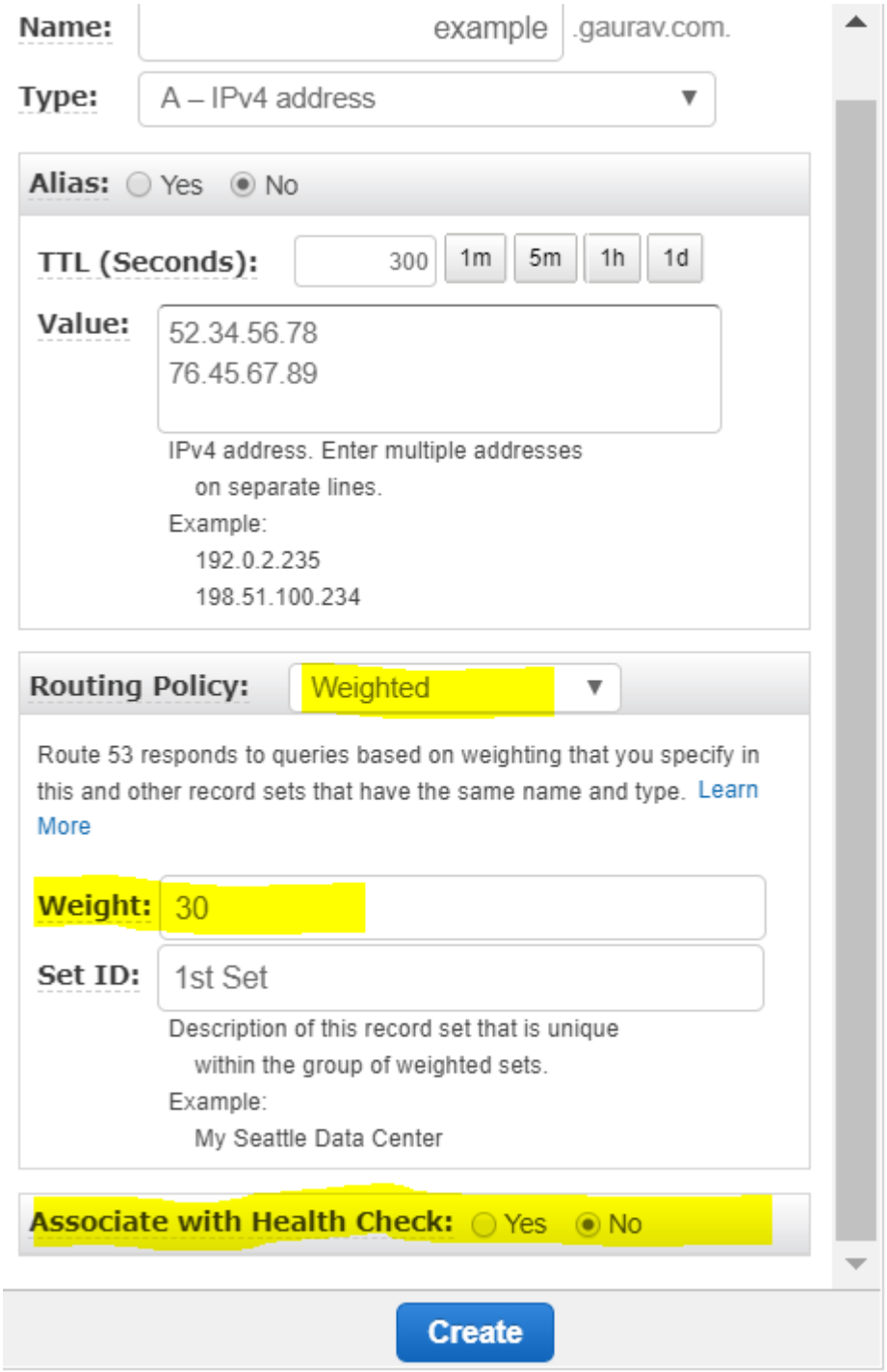
Simple Routing Policy

2- Weighted Routing Policy

Weighted Routing Policy controls the what percentage % of the requests that go to specific endpoint. It's helpful to test 1% of traffic on new app version. It is also helpful to split traffic between two regions. We can associate Health checks with it.

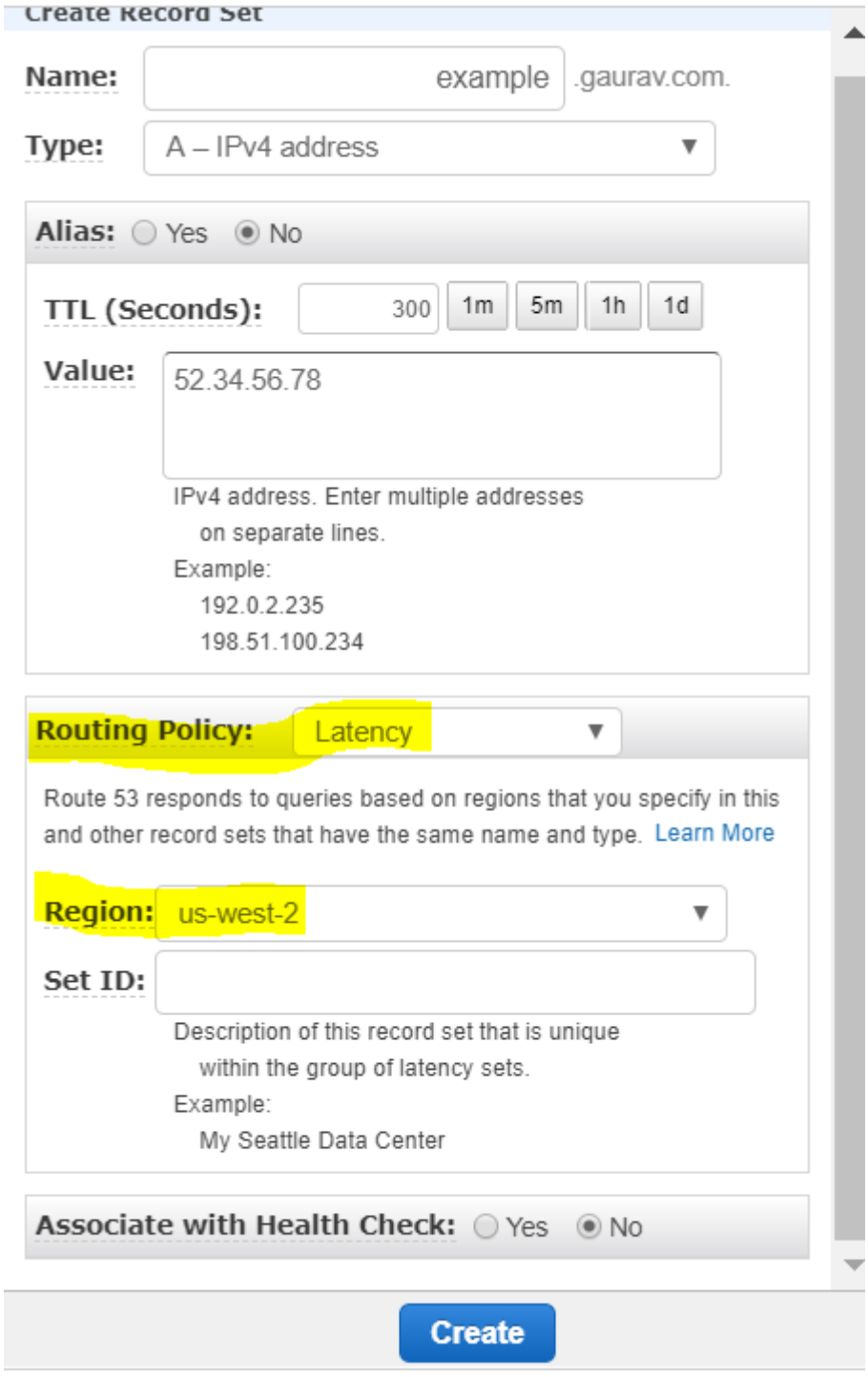


Weighted Policy



3- Latency Routing Policy

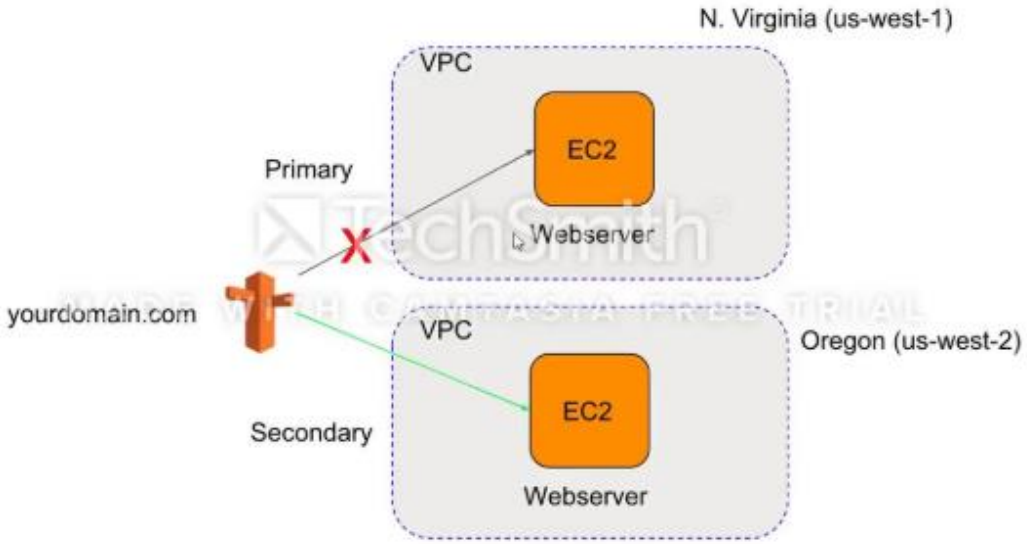
It allows you to route your traffic based on lowest network latency for your end user. It redirects to the server that has the least latency close to us also helpful when latency of users is a priority. Latency is evaluated in terms of user to designated AWS Region. For example: Germany may be directed to the US (if that's the lowest latency)

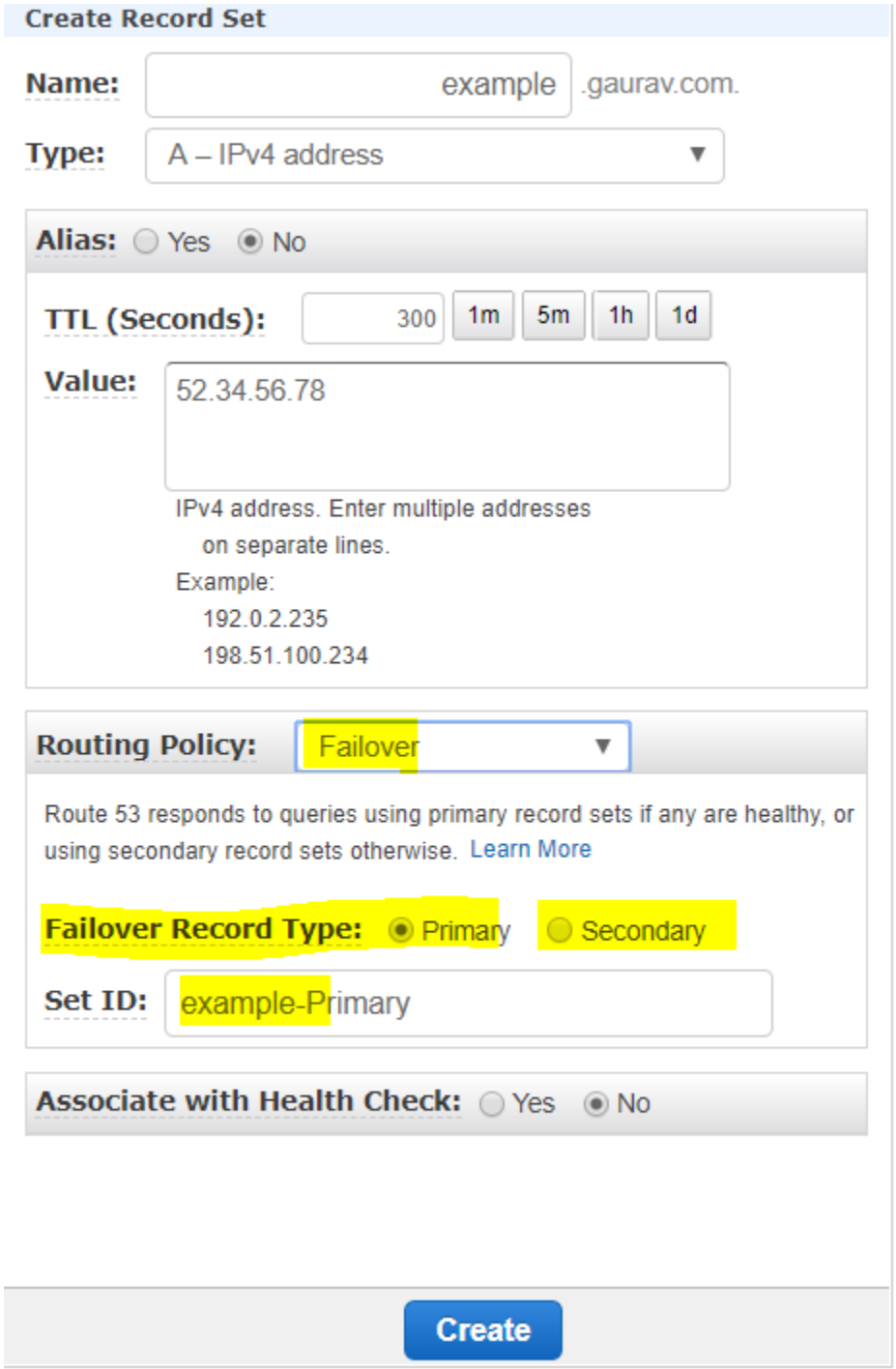


4- Failover Routing Policy

Failover routing lets you **route** traffic to a resource when the resource is healthy or to a different resource when the first resource is unhealthy. The primary and secondary records can **route** traffic to anything from an Amazon S3 bucket that is configured as a website to a complex tree of records.

You can associate health check with this type of policy.



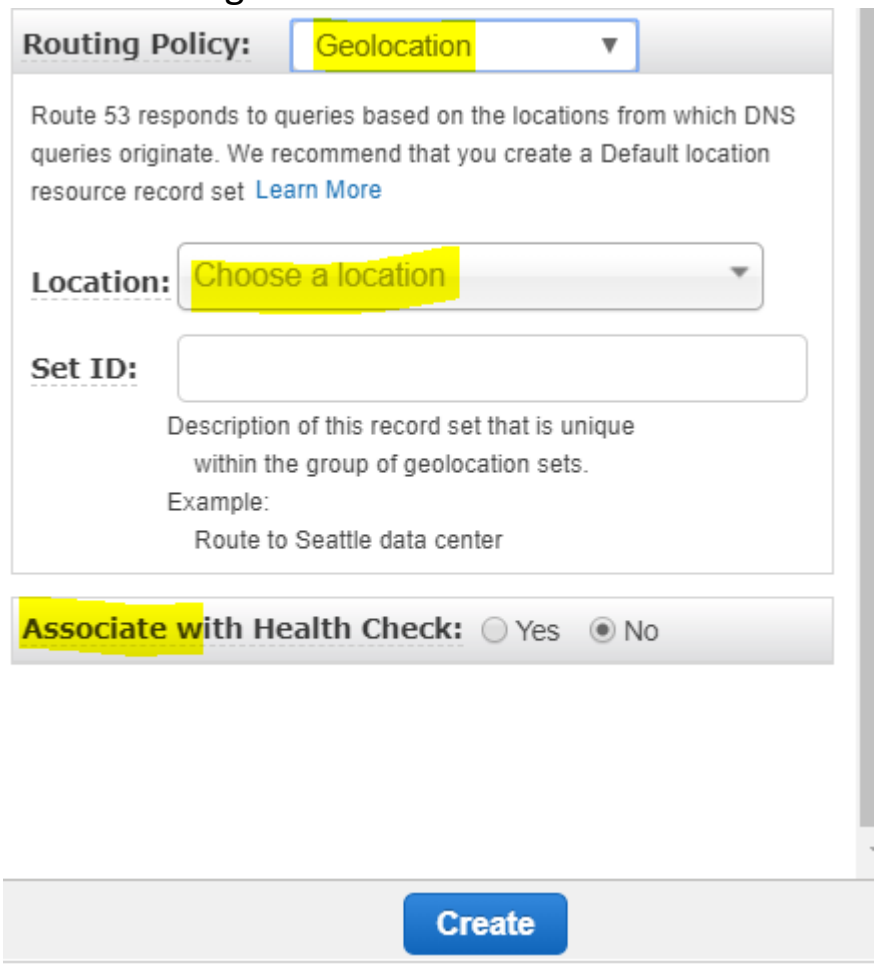


Failover Routing Policy

5- Geo Location Routing Policy

Geolocation routing lets you choose the resources that serve your traffic based on the geographic location of your users, meaning the location that DNS queries originate from. For example, you might want all queries from Europe to be **routed** to an ELB load balancer in the Frankfurt region.

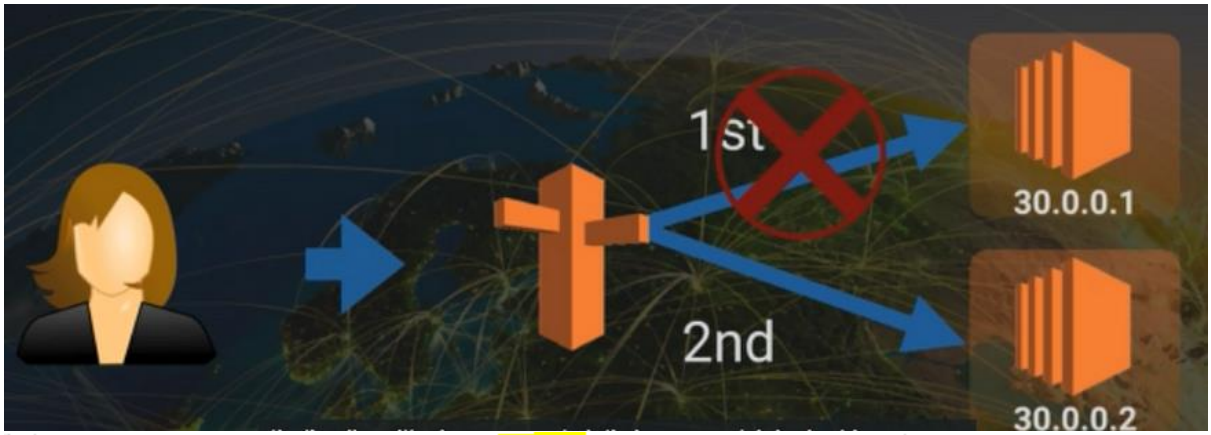
This is routing based on user location. Health check associated.

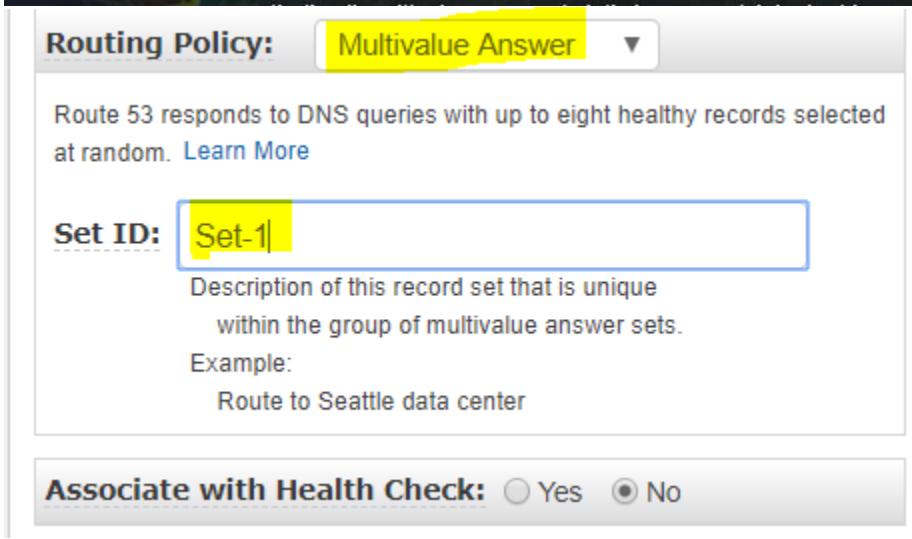


Geolocation Policy

6- Multi Value Routing Policy

It helps distribute DNS responses across **multiple** resources. For example, use **multivalue** answer **routing** when you want to associate your **routing** records with a **Route** 53 health check.
Use multivalue answer routing when you need to return multiple values for a DNS query and route traffic to multiple IP addresses. Up to 8 healthy records are returned for each Multi Value query. Multi Value is not a substitute for having an ELB.





MultiValue Routing

7. Amazon Elastic Disaster Recovery

Definition:

Minimizes downtime, data loss with fast reliable recovery of on-premises & cloud-based applications using affordable storages, minimal compute & point-in-time recovery

Recovery-why?

- Disaster happened at one Availability zone, at that point of time, we have to point/route all the requests to the recovery environment where data is already replicated at the backend.
    - Users will not get impacted & the new data will get routed to the new server whichever we consider it as recover server called as **Failover(**to launch recovery

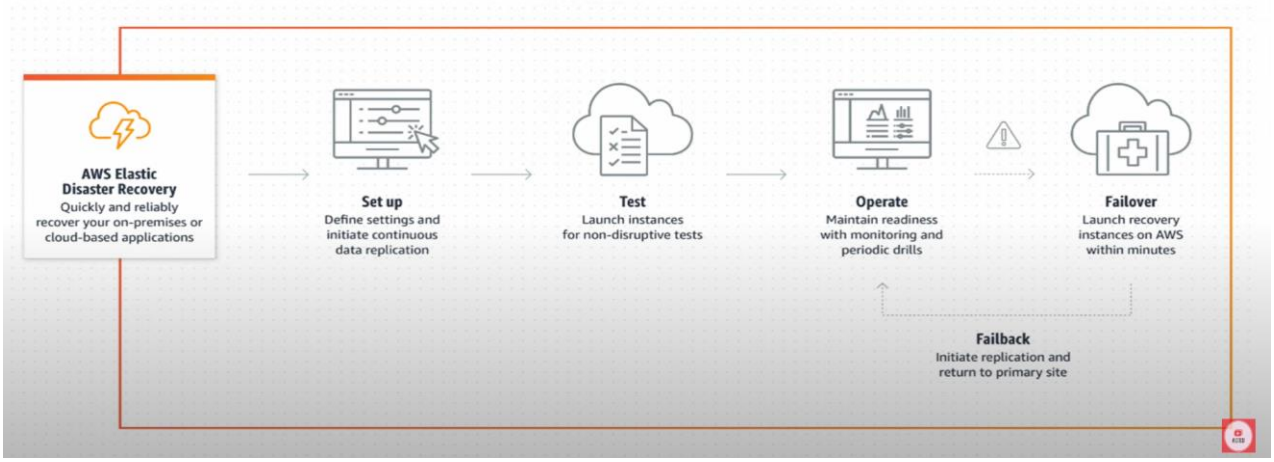
instances having previous data & new requests will be routed to new server)

- Disaster is over, **failback** from recovery server to the original source server.

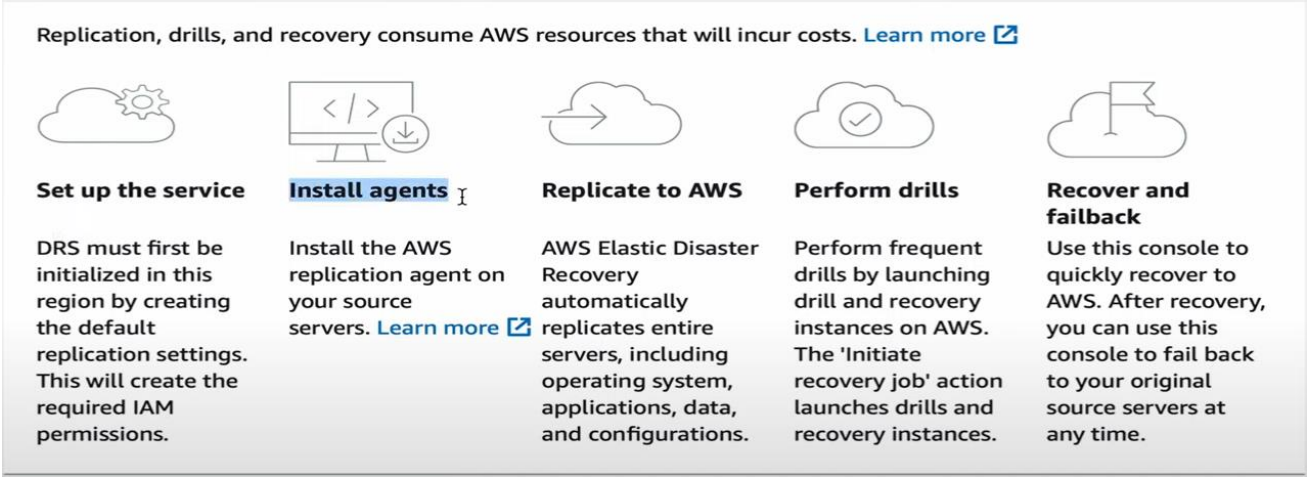
Process:

- Create an replication agent which acts as a media b/w source server & DR environment & tries to replicate(in time interval) . In the backend, it takes the snapshot of the source environment if any disaster happens.
- During Failover, the latest sanpshot can be taken for recovering our instances & source environment if any disaster happens.
- Available for Region-to-region; within a region; on-premises system

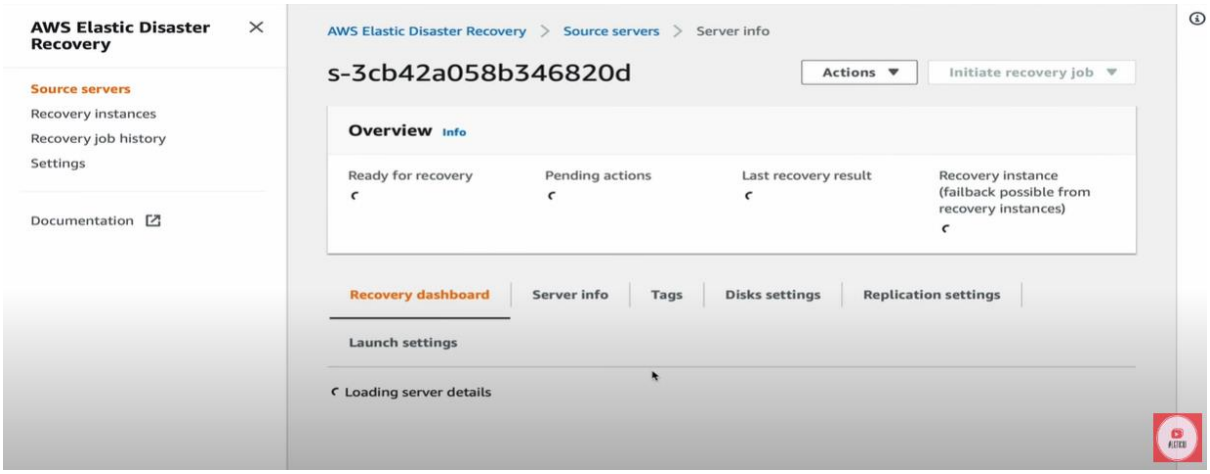
Architecture:



How it works?



EDR Dashboard:



- Server Info→ RAM, Recommended Instance type, Hardware
- Tag→ Name
- Disk settings→ Disk name & storage
- Replication settings→
- Launch settings→ Ec2 Launch Template
    - Create template
        - Size/Tier
        - Key pair
        - SG
        - Subnet
        - Storage→gp3
    - Template version→ Set as “default version”

Steps:

Step1: Settings

- Go to AWS→ Services→ Storage→ Amazon Elastic Disaster Recovery→Settings→Edit default Application settings
- Subnet→ Public/Private
- Replication server instance type→ t3.micro
- Volume→ SSD gp3 cheaper than gp2
- EBS Encryption→ Default
- Security Group→ Custom SG
- Data Routing:
    - VPN, Direct Connect, VPC peering
    - Public IP creation
- Point in Time→ Snapshot retention(1-365)

Step2: IAM Role setup

- Role setup→ IAM→ Role→ Add or Create Role
    - Entity→EC2
    - Tag→ <name>
- Policy/Permission: AWSElasticDisasterRecoveryRecoveryInstancePolicy
- Attach the role to your user
- Assign the role to the running instances→ Actions→ Security→ Modify IAM Role→ Enter Role & Save it

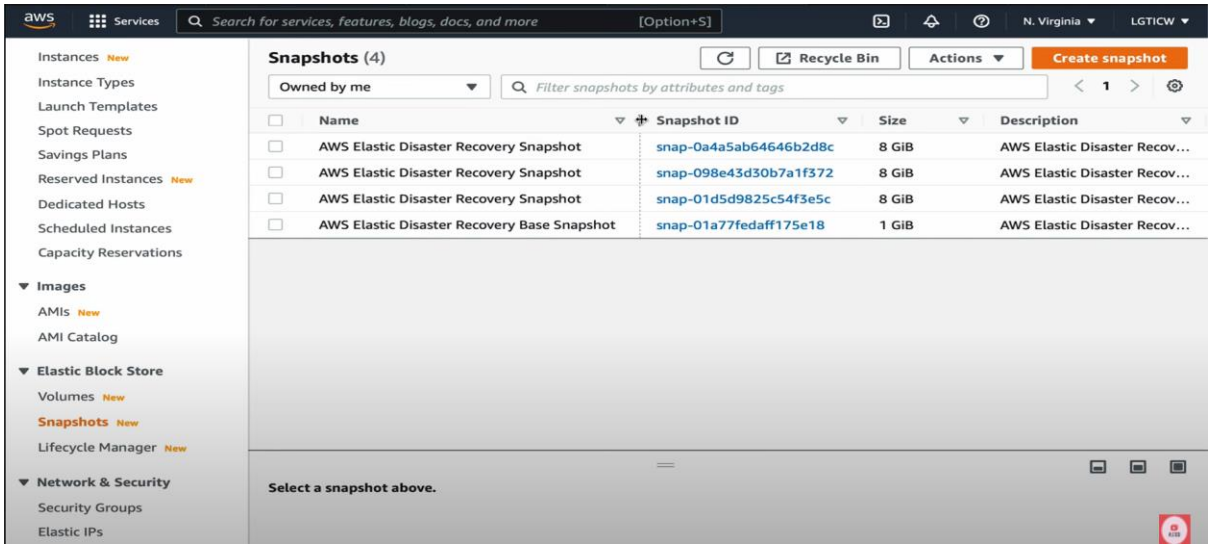
Step3: Install Agent

- Install Agent→ Pull the code/file to CLI using wget (refer Amazon documentation)
- Execute the file as root user
    - Enter AWS region
    - Enter Access Key & Secret Key [IAM→Users→Security credentials]

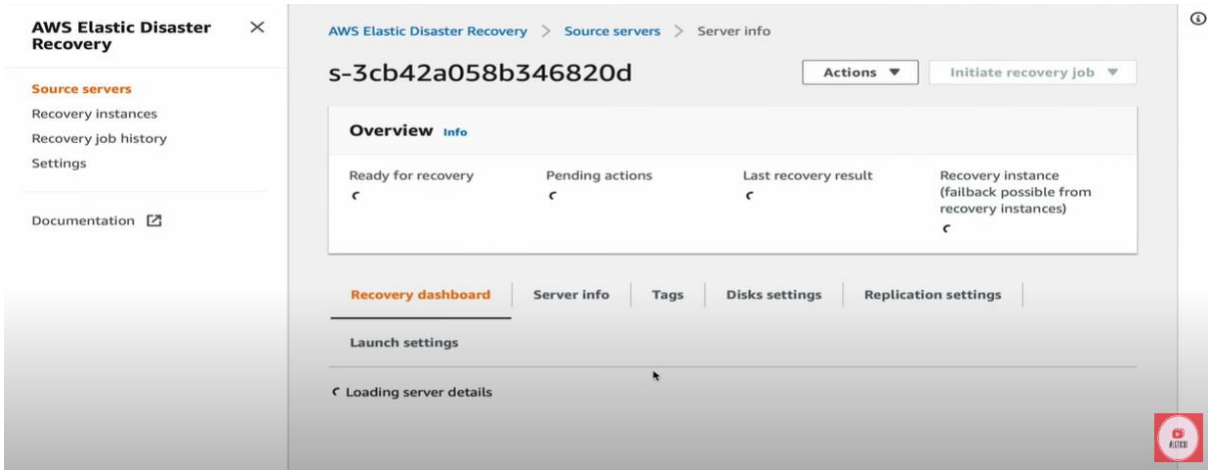
- Choose the disks to replicate→ Enter to replicate all
- Once this is done, replication agent will be downloaded & installed, then in AWS EDR dashboard, we can see the hostname with source servers Private IP
- Data replication will be initiated
- We will have one new instance as **"AWS Disaster Recovery Replication Server"** along with source instance.

Step4: Recovery Job

- Once replication is done & snapshot is created(Source server is ready for recover) and the same can be viewed in EDR Dashboard

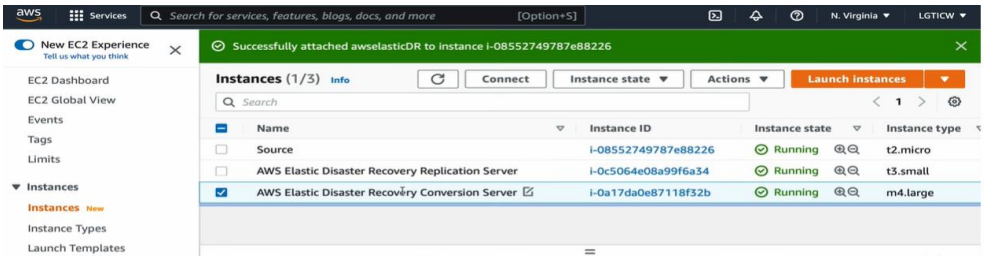


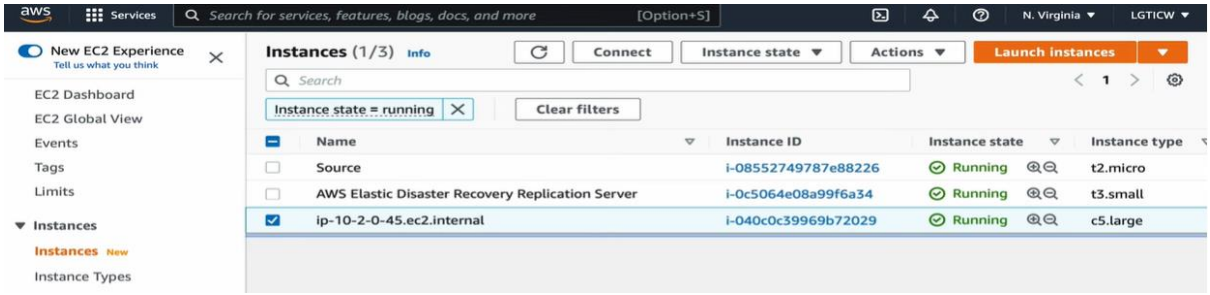
- Select Hostname→Initiate Recovery job→Initiate recovery



- Select the required PIT
- Initiate recovery

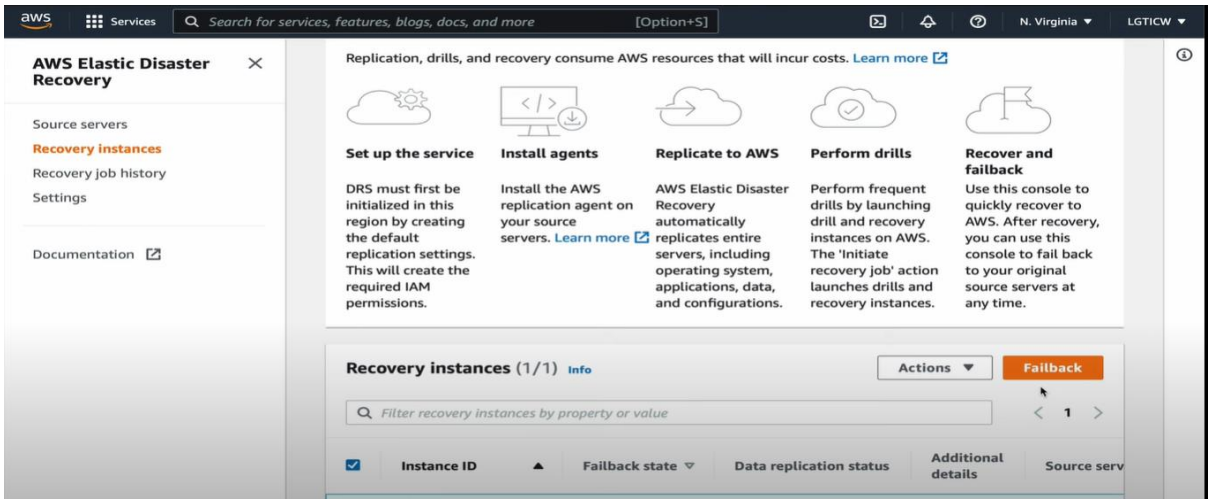
- This will create a conversion server in EC2 instances as "**AWS Disaster Recovery Conversion Server"**
  - This is not the replicates server
  - This will stop & terminated once the replicated server instance is created/initiated





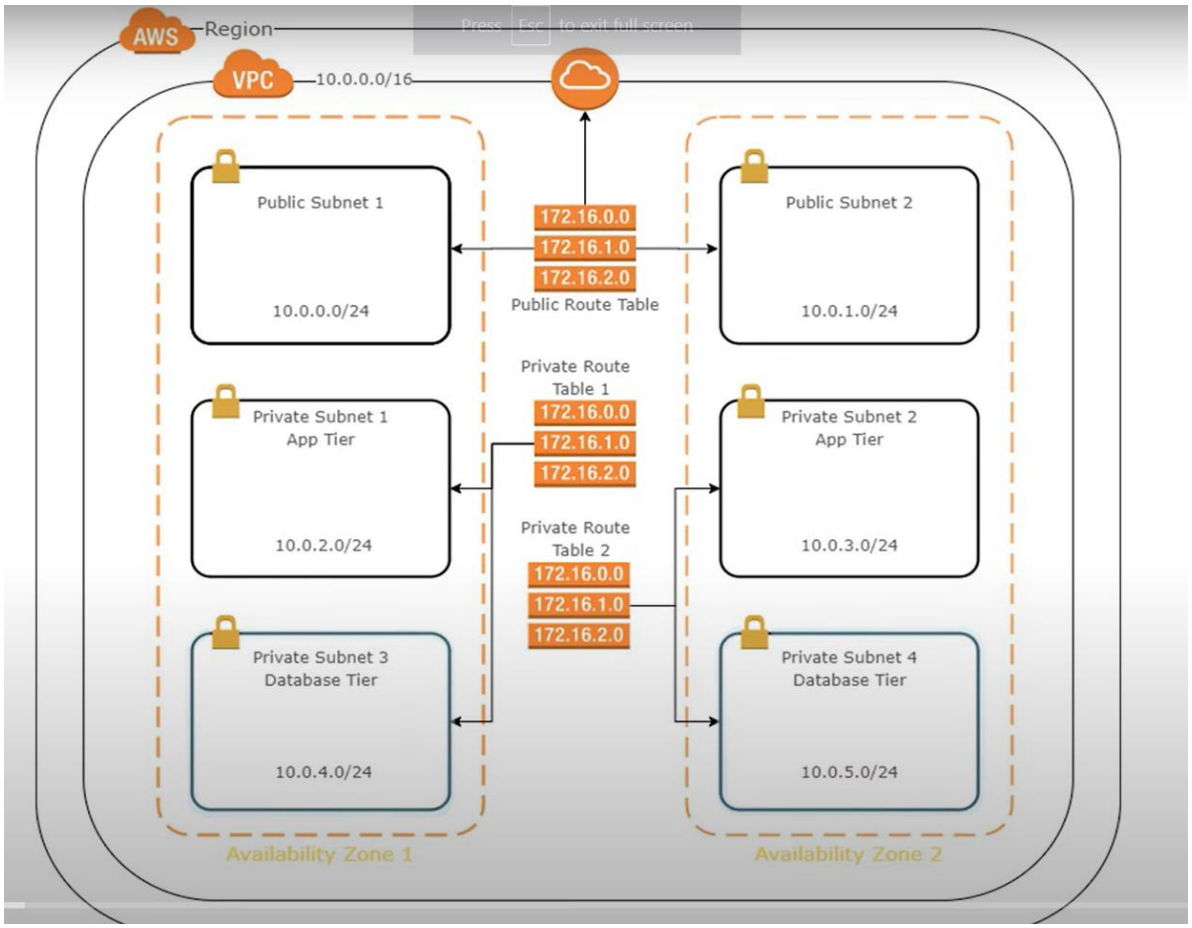
Step5: Failback

- Install agents in the replicated server
- Go to Recovery instances→ Select → Failback



8. VPC

Architecture:

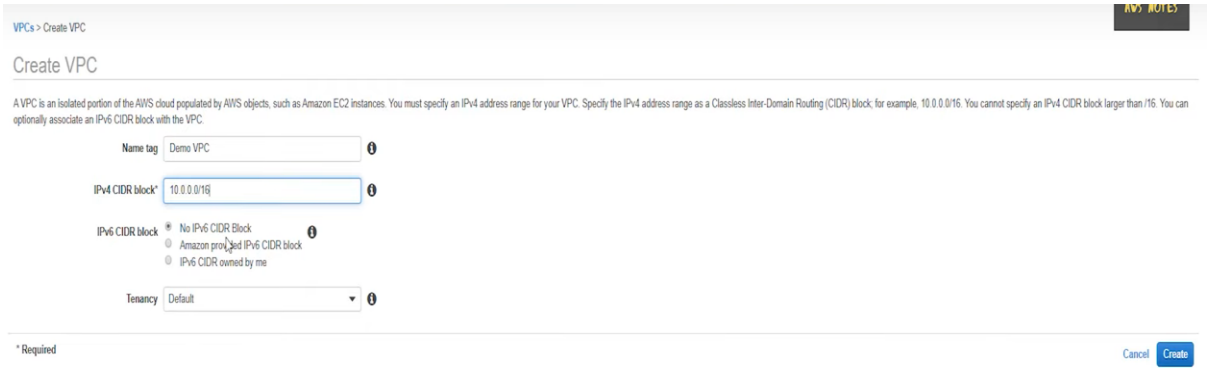


Process:

Steps:

Step1: Create VPC

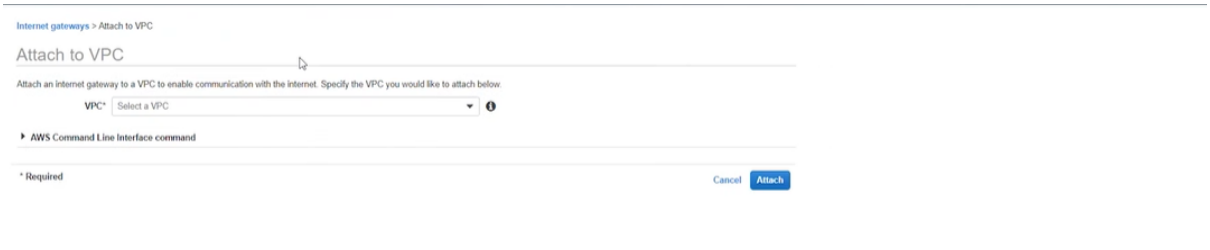
- Select region
- Services→ Networking & Content delivery→ VPC
- Create VPC
- Enter CIDR block

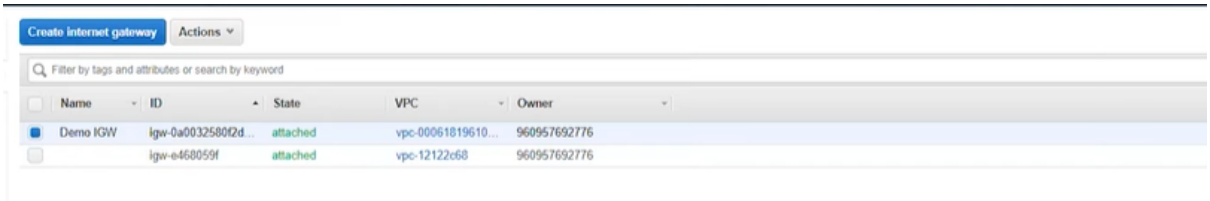


- Create
- The system will automatically create Route table which is private

Step2: Create Internet Gateway and attach it to VPC

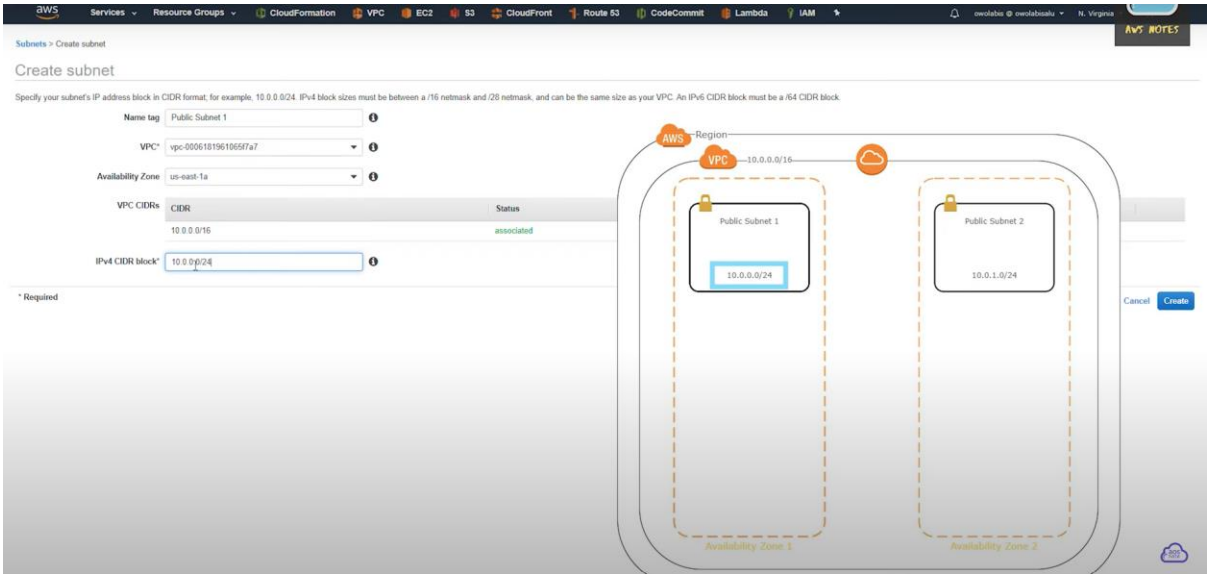
- Select Internet Gateway→Create
- Give name “Demo IGW”
- Close
- Then Select created gateway→Actions→Attach to VPC
- Enter the VPC
- Click “Attach”





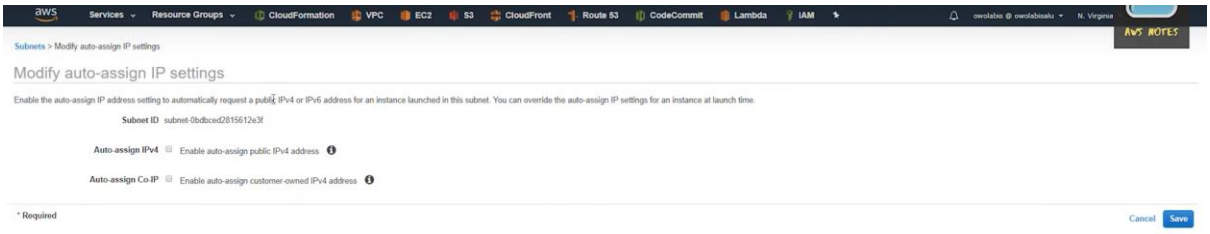
Step3: Create Subnets in Different Availability Zones

- Click on “Subnets” in Left Pane
- Fill the details like required VPC, Availability zone, Name of subnet(Public subnet1), CIDR block

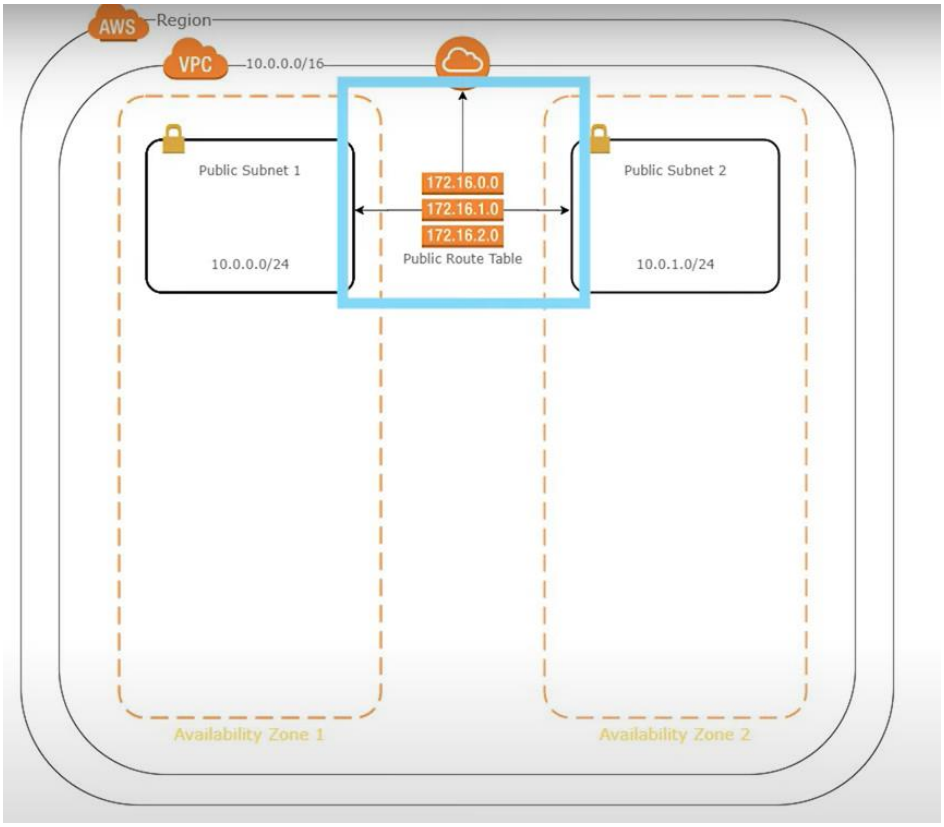


- Repeat the previous step to create another subnet(“Public subnet2”) and change the Availability Zone alone.

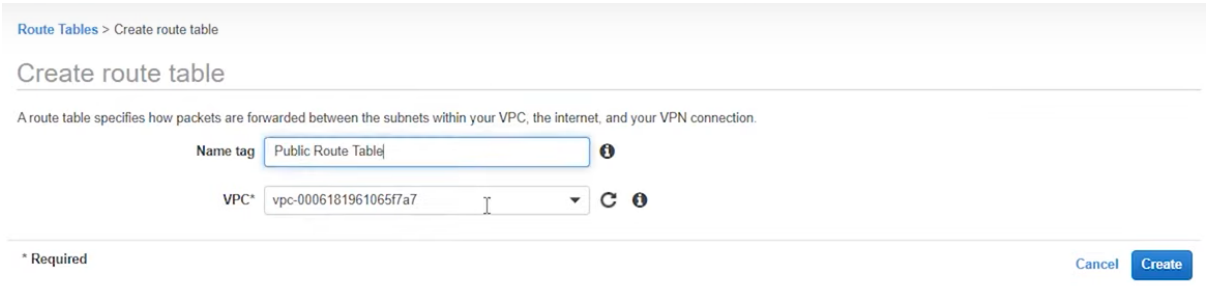
- To assign IP automatically to this subnet, select the Subnet→Action→Modify auto-assign IP settings & check-in **Enable** IpV4



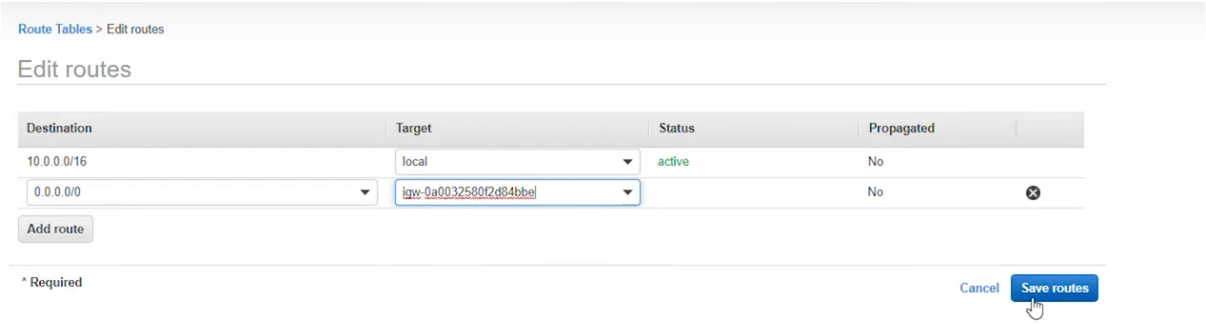
Step4: Create and Assign public route table to public subnet 1 & 2



- Select “Route table” in the Left pane of AWS console and select Create
- Enter details like Name & Enter VPC “Demo VPC” & **Create**

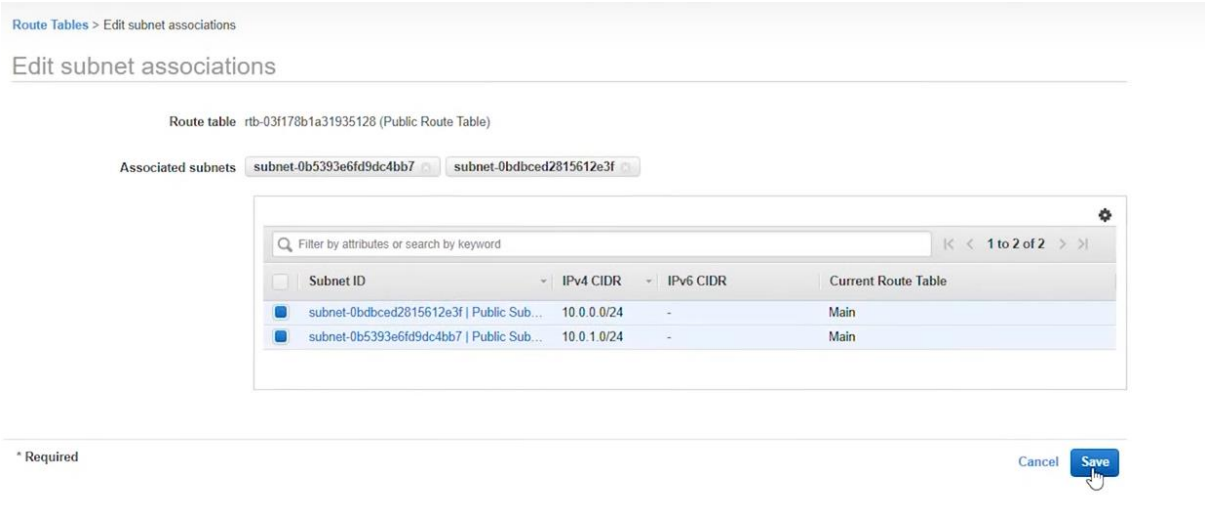


- Select created Route Table→ Routes
- Enter Route as “0.0.0.0/16” and Target as “Internet Gateway” and **Save Routes**

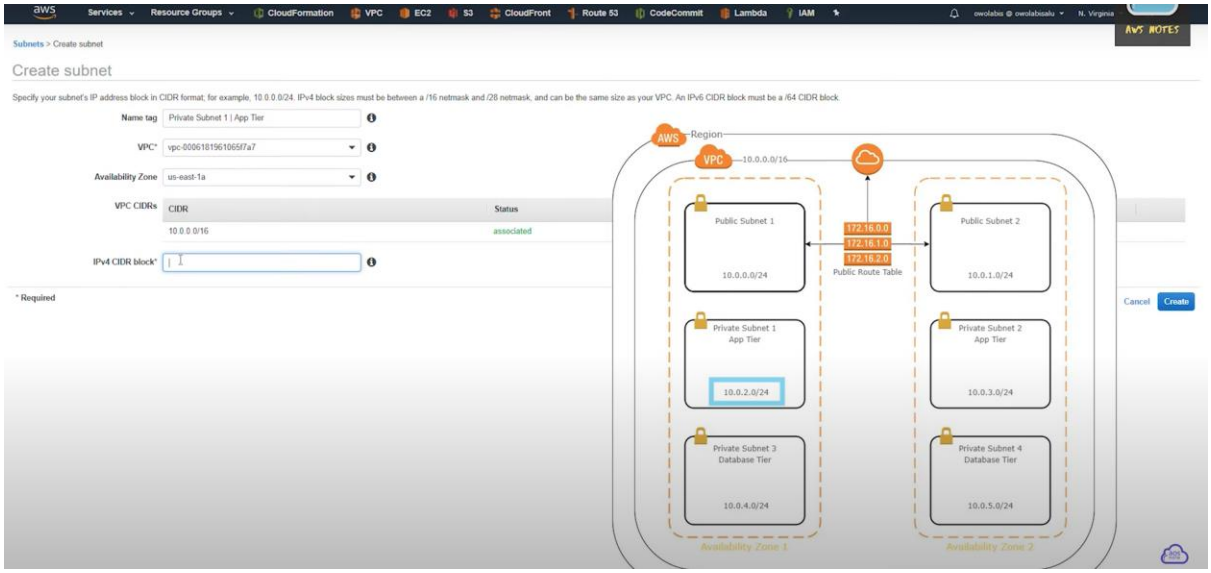


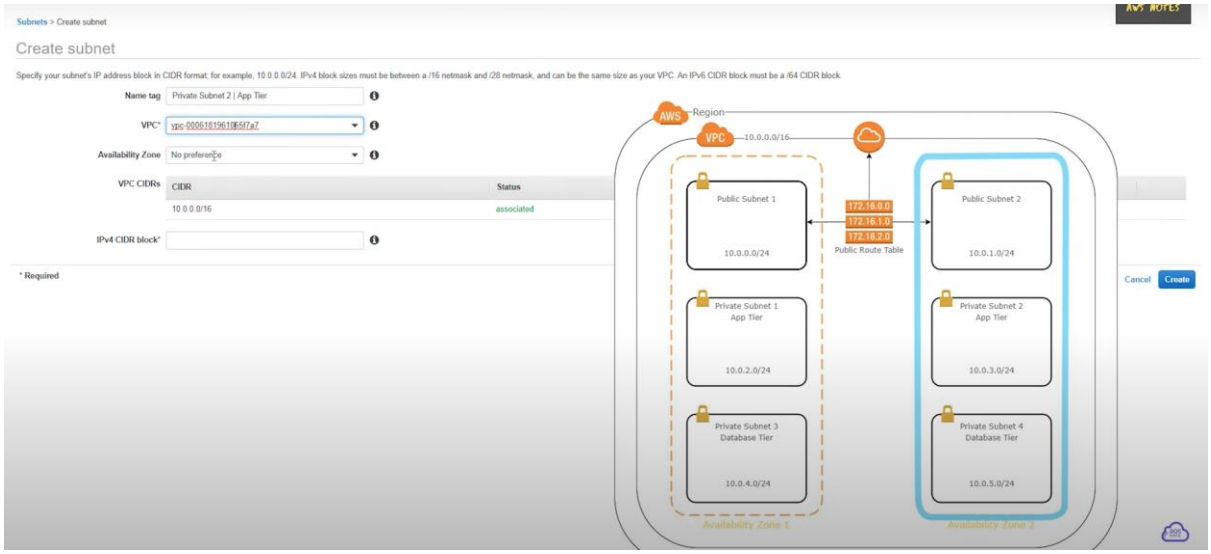
- To associate the subnets, Select Route table → Subnet Associations → Edit

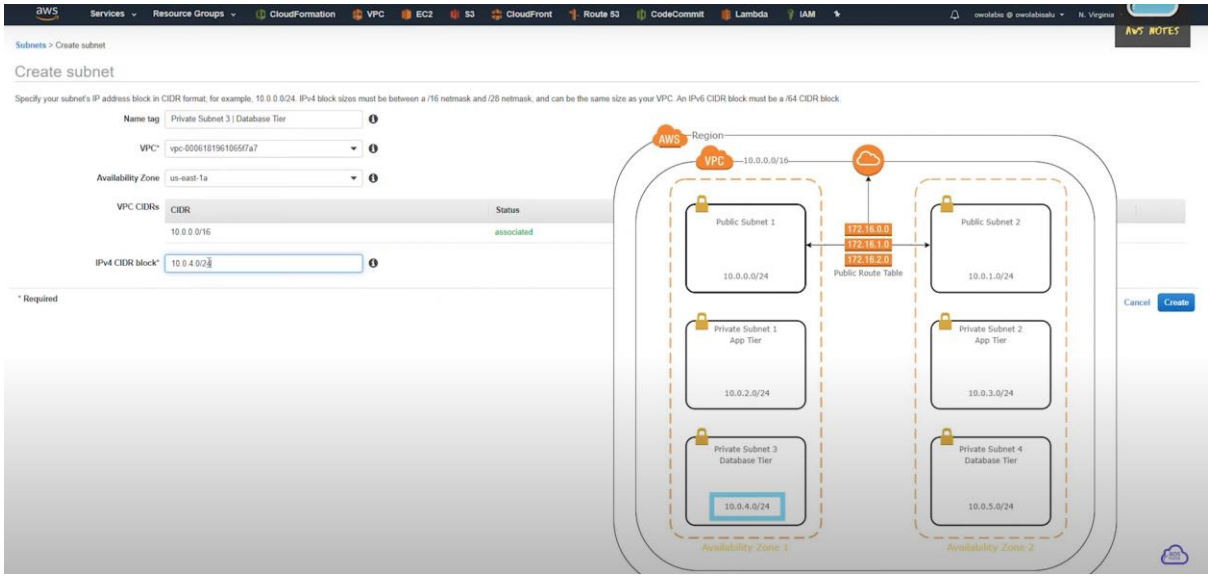
- Check-in the created Subnets and click **Save**

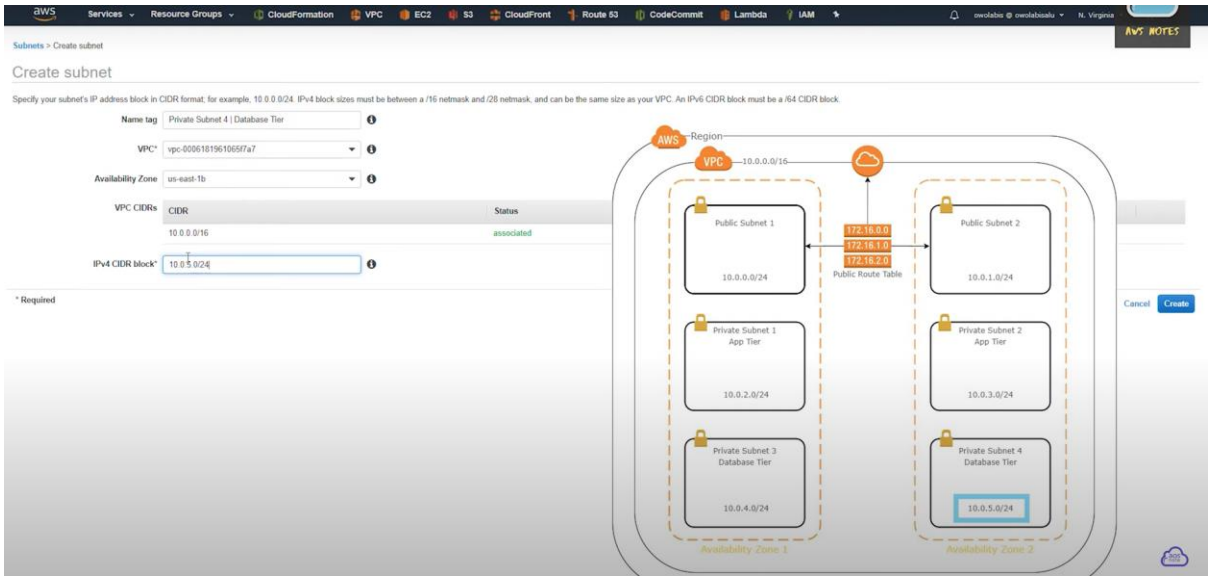


- Create the two private subnets by following the same steps, but it doesn't have any Internet gateway associated with it

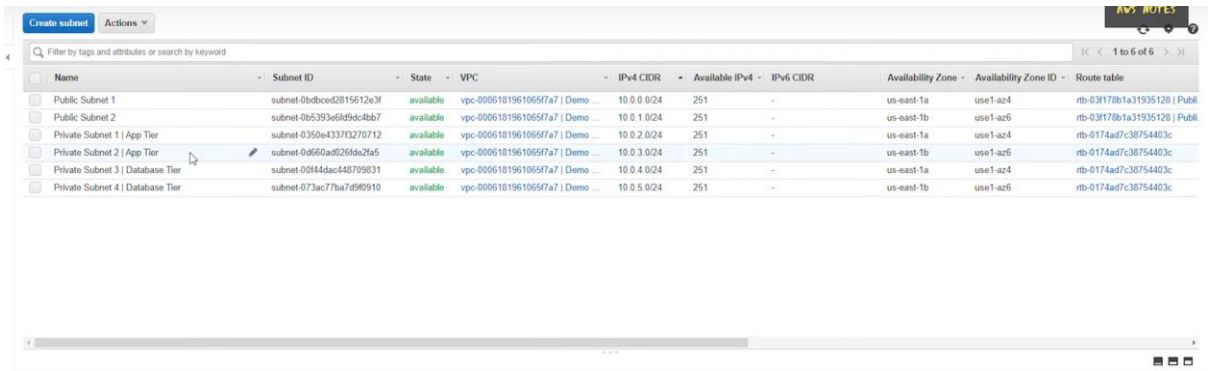

Services
Resource Groups
CloudFormation
VPC
EC2
S3
CloudFront
Route 53
CodeCommit
Lambda
IAM
Subnets > Create subnet
Create subnet
Specify your subnet's IP address block in CIDR format; for example, 10.0.0.0/24. IPv4 block sizes must be between a /16 netmask and /28 netmask, and can be the same size as your VPC. An IPv6 CIDR block must be a /64 CIDR block.
Name tag
Private Subnet 1 | App Tier
VPC*
vpc-0006181961065f7a7
Availability Zone
us-east-1a
VPC CIDRs
CIDR
Status
10.0.0.0/16
associated
IPv4 CIDR block*
* Required
Cancel
Create
AWS
Region
VPC
10.0.0.0/16
Public Subnet 1
10.0.0.0/24
172.16.0.0
172.16.1.0
172.16.2.0
Public Route Table
Public Subnet 2
10.0.1.0/24
Private Subnet 1
App Tier
10.0.2.0/24
Private Subnet 2
App Tier
10.0.3.0/24
Private Subnet 3
Database Tier
10.0.4.0/24
Private Subnet 4
Database Tier
10.0.5.0/24
Availability Zone 1
Availability Zone 2


Subnets > Create subnet
Create subnet
Specify your subnet's IP address block in CIDR format; for example, 10.0.0.0/24. IPv4 block sizes must be between a /16 netmask and /28 netmask, and can be the same size as your VPC. An IPv6 CIDR block must be a /64 CIDR block.
Name tag
Private Subnet 2 | App Tier
VPC*
vpc-0006181961065f7a7
Availability Zone
No preference
VPC CIDRs
CIDR
Status
10.0.0.0/16
associated
IPv4 CIDR block*
* Required
Cancel
Create
AWS
Region
VPC
10.0.0.0/16
Public Subnet 1
10.0.0.0/24
172.16.0.0
172.16.1.0
172.16.2.0
Public Route Table
Public Subnet 2
10.0.1.0/24
Private Subnet 1
App Tier
10.0.2.0/24
Private Subnet 2
App Tier
10.0.3.0/24
Private Subnet 3
Database Tier
10.0.4.0/24
Private Subnet 4
Database Tier
10.0.5.0/24
Availability Zone 1
Availability Zone 2

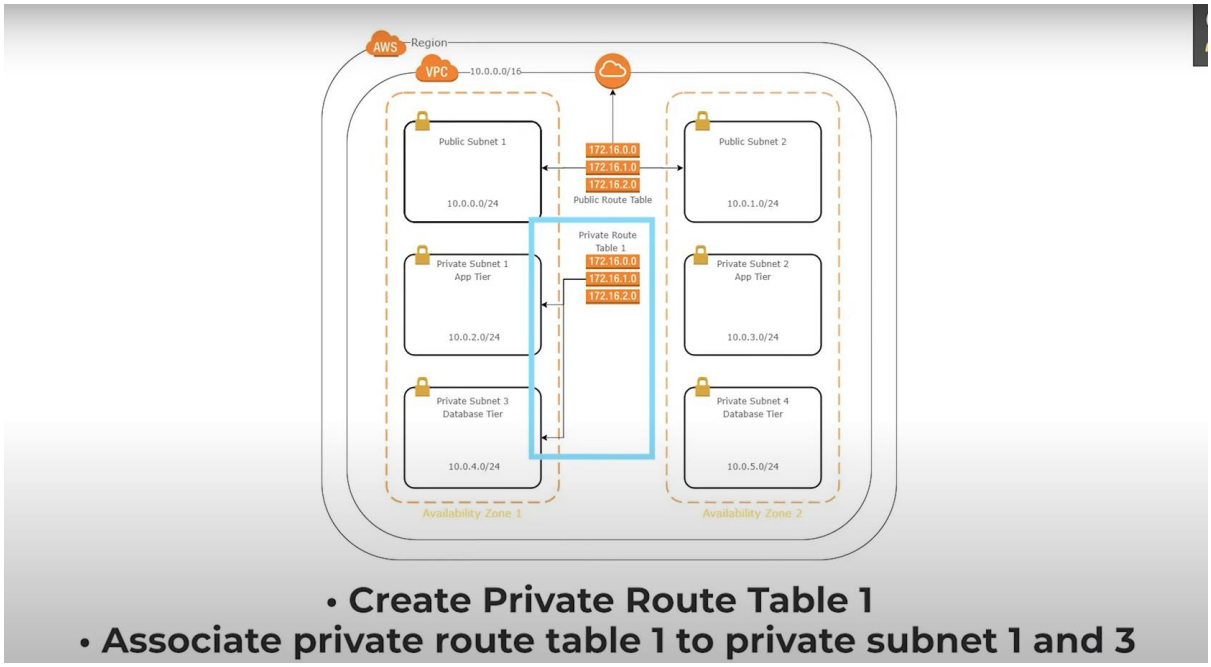


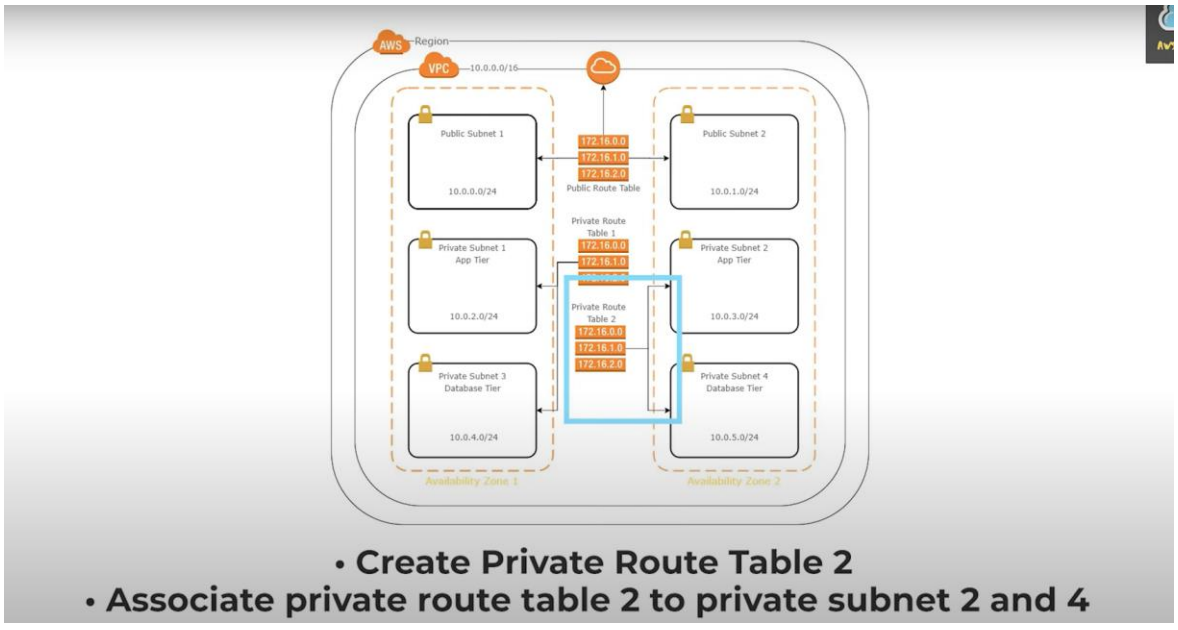


After Subnet creation:



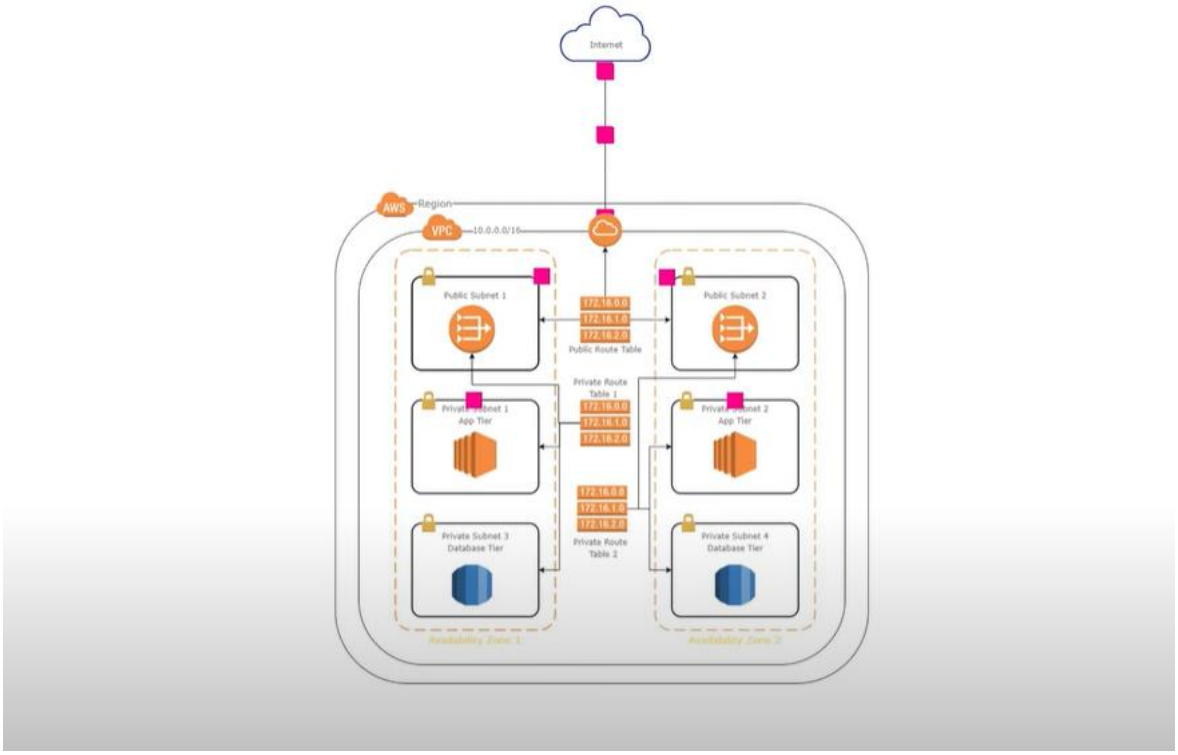
Step5: Create Two Private Route tables and attach it to 4 subnets



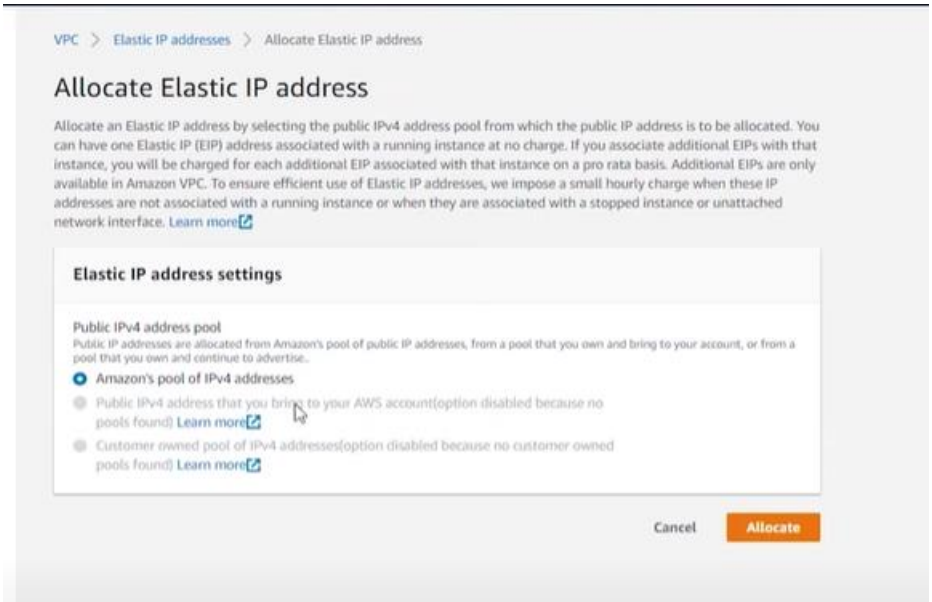


- Follow the step 4 till the Subnet Associations

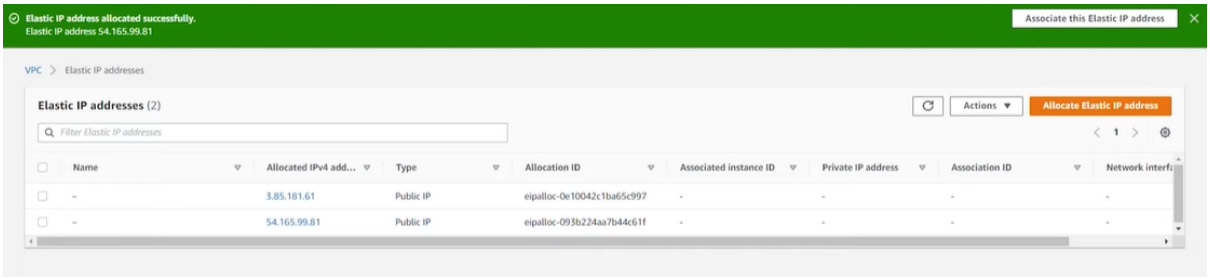
Step6: Create two Elastic IP's



- Select Services→Networking & Content Delivery→ VPC
- Click on "Elastic IP's" in the Left pane of AWS dashboard
- Click "Allocate Elastic IP address"

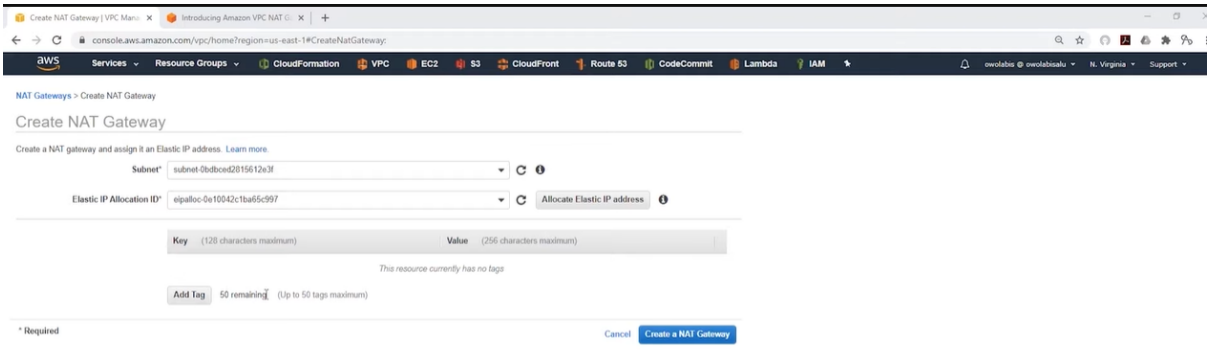


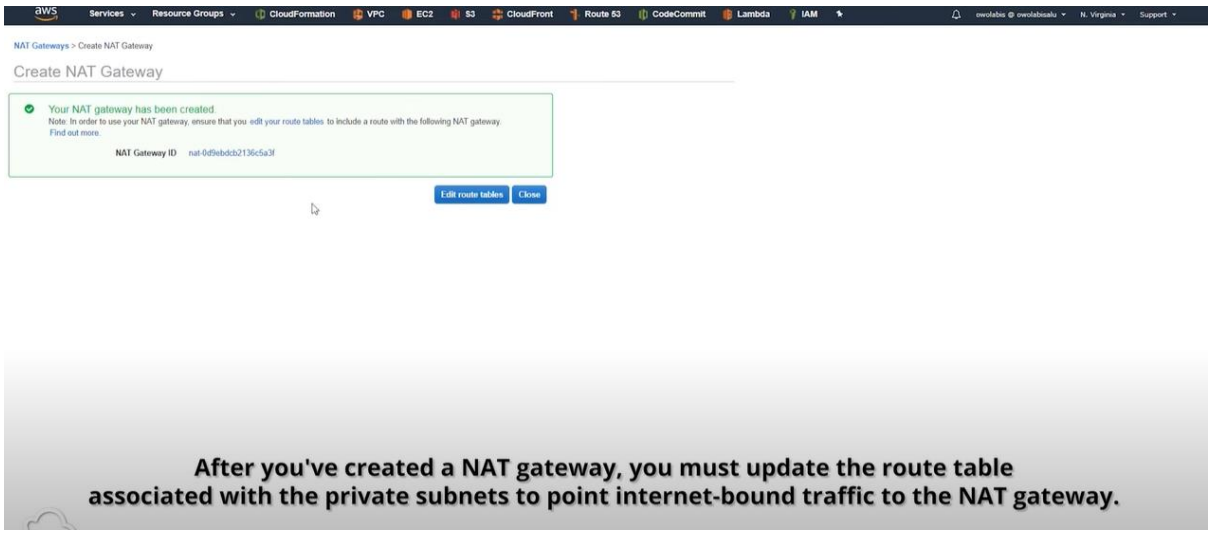
- Click "Allocate"
- Repeat the same step to create another Elastic IP



Step7: Create NAT Gateway

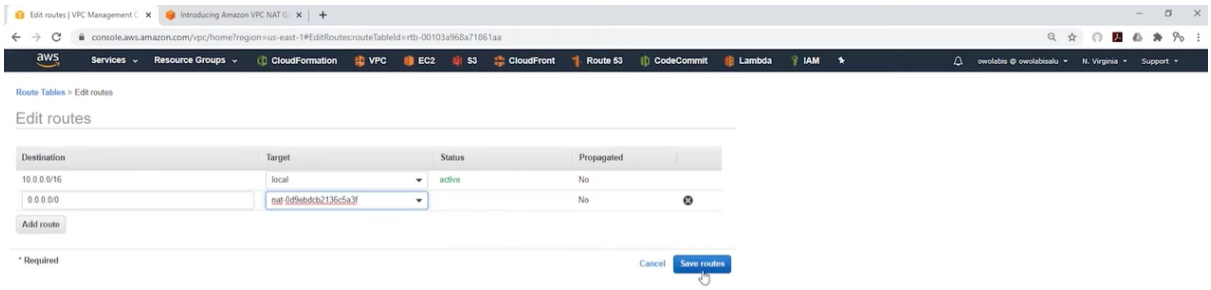
- Select “NAT Gateway” from LP and click **Create**
- Enter the details like Public subnet tag and Elastic IP for NAT gateway & click **Create**





- Click “Edit Routes”
- Select the Private Route table 1 and click “Routes” & Edit

- Enter the Open IPv4 and Target as “NAT Gateway” and click Save



- Repeat the same steps to create another NAT gateway and assign it to Private Route Table 2.