NETWORKING:

VPC (VIRTUAL PRIVATE CLOUD):

Amazon Virtual Private Cloud (Amazon VPC) enables you to launch AWS resources into a virtual network that you've defined. You can launch your AWS resources, such as Amazon EC2 instances, into your VPC. You can specify an IP address range for the VPC, add subnets, associate security groups, and configure route tables.

Charastics of VPC:

1. VPC is region specific.
2. We can create upto 5 VPCs per region by default.
3. If you want to increase the VPC limit you want write email to AWS support team.
4. At present VPC is free of cost.

Creating a VPC:

From Amazon VPC dashboard → Choose Your VPCs, Create VPC → Specify the VPC details.

1. Name tag: Choose the desired name for VPC. Ex: “WestronUnion-VPC”
2. IPv4 CIDR block: Specify an IPv4 CIDR block for the VPC. **Ex**: 192.168.0.0/16. If you kept 16 as net mask there will be 2^(32-16)=2^16 ip addresses will be available inside this VPC, But first 4 and last 1 ip addresses are reserved for AWS.
3. Tenancy: Select a tenancy option. If you choose “*dedicated tenancy*” you want to install dedicated instances only with in the VPC, We don’t have any dedicated instances so we want to choose “*default tendency*”.
4. Next “*Yes Create*”.

SUBNET:

Subnet is a smaller network inside VPC.

Charastics of Subnet:

1. Subnet is availability zone specific
2. We can create as many subnets as possible within the VPC cidr netmask limit.
3. There is no cost for subnet, it is free.

AWS (AMAZON WEB SERVICES)

Creating a Subnet:

From Amazon VPC console → choose Subnets → Create Subnet → Specify the subnet details.

1. Name tag: Choose the desired name for VPC. Ex: "*WestronUnion-Subnet*"
2. VPC: Choose the VPC for which you're creating the subnet.
3. Availability Zone: Optionally choose an Availability Zone in which your subnet will reside, or leave the default No Preference to let AWS choose an Availability Zone for you.
4. IPv4 CIDR block: Specify an IPv4 CIDR block for your subnet. Ex: "*192.168.0.0/28*".
5. Next "*Create*".

Enabling network connectivity to the VPC:

To establish network connectivity we have the following options:

1. By using AWS Internet Gateway
2. By using VPN
3. By direct connect (A dedicated physical cable between our network and AWS)

AWS Internet Gateway(IGW):

An internet gateway is a horizontally scaled, redundant, and highly available VPC component that allows communication between instances in your VPC and the internet.

Creating AWS Internet Gateway:

AWS Console → VPC Dashboard → Internet Gateways → Create Internet Gateway

1. Name Tag: Just give the name what you desired. Ex: "*WestronUnion-IGW*"

By default the state of Internet Gateway is detached. So we want to attach to the VPC. For attaching IGW to VPC just follow the steps:

Select IGW → From Actions → Attach to VPC → Select your VPC → Attach

Note:

1. One Internet Gateway can attach to one VPC only.
2. Attaching IGW to the VPC does not give access for our subnets to internet, to grant internet access to the subnets we need to modify the route table of the subnet.

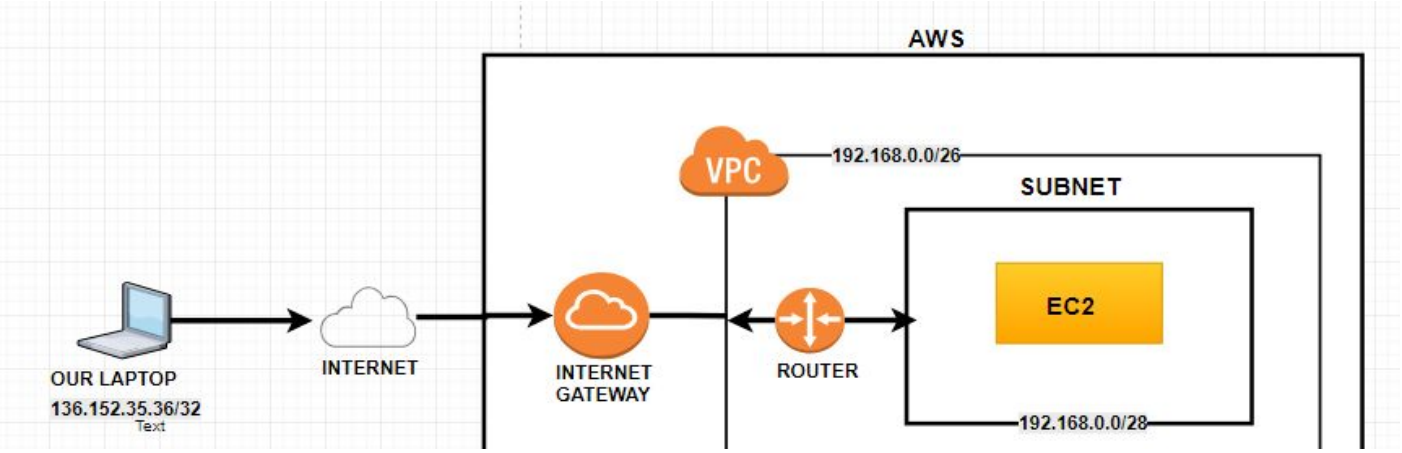
3. In Order to grant internet access to the subnet, we need to add a route to the internet in its route table.

Modifying Route Tables:

AWS Console → VPC Dashboard → Route Tables → Select route table associated with our VPC → Select routes → Click Edit → Add another route

Here we want to give two details:

1. In destination *0.0.0.0/0* (Means any resource in internet)
2. In the target → Select Internet Gateway what we created.



AWS EC2 (ELASTIC COMPUTE CLOUD):

EC2 stands for Elastic Compute Cloud. EC2 allow users to use virtual machines of different configurations as per their requirements. EC2 is a virtual server in the cloud which has, CPU, RAM, OS and some networking options.

Getting Started with AWS EC2:

From AWS console → EC2 → Instances → Launch Instance(Server).

For creating Instance in AWS we want to follow 8 steps:

1. Choose an Amazon Machine Image (AMI)
2. Choose an Instance Type
3. Configure Instance Details
4. Add Storage
5. Add Tags
6. Configure Security Group

7. Review Instance Launch
8. Select Existing Key Pair or Create New Key Pair

1. *Amazon Machine Image (AMI):*

AMI is a template containing operating system and few pre installed softwares. Ex: Amazon Linux, SUSE Linux, Red Hat Enterprise Linux, Ubuntu, Windows etc… As per our client requirement we want to select particular linux flavour. At present we want select only free tier eligible AMIs only otherwise we will get bill. Ex: *Amazon Linux*

2*. Instance Type:*

Instance type decides how many CPUs and how much RAM are allocated to the instance.

Ex: 1. t2.micro (1 CPU, 1GB RAM) → **Free Tier eligible**
2. t2.xlarge (4 CPU, 16GB RAM)
ETC…

3*. Configure Instance Details:*

In this option we want to select number of instances we want to create, VPC, Subnet, Auto assign public Ip enable or disable and etc…

Number of instances-1
Network- Select Your Vpc
Subnet- Select your Subnet
Auto-assign Public IP- Yes Enable
Remaining things default for present
Next Add STORAGE

4*. Add Storage:*

Storage is nothing but hard disk to the Server. As per client requirement we want to choose how much GB hard disk should be. Ex: For Linux minimum 8GB, for windows minimum 30GB. (**Note:** For free tire 30GB only free**)**

Note: In feature it is possible to increase hard disk size, but it is not possible to decrease the size.

Ex: Suppose you created a instance with hard disk size 10GB, after some days you may be increase

5*. Add Tags:*

A tag is a label that you or AWS assigns to an AWS resource. Each tag consists of a key and a value.

Ex: “Name” as a “Key” and “Value” as a “WEBSERVER”

1. Name: WestronUnionServer
2. ENV: DEV

***6**. Configure Security Group:*

Security Group is a virtual firewall to the EC2 instance.

1. First Select Create New Security Group
2. Name the Security Group
3. Give Description
4. At the type part select your protocol for which you want to open.
5. At Source part select any one of from Custom, anywhere or Myip.

Ex:

1. If you want connect to linux server you want to open *SSH(22)*
2. If you want connect to Windows server you want to open *RDP(3389)*
3. If you want connect to Jenkins/Tomcat server you want to open Custom Rule(at *Port Range* palace keep *8080*) + SSH(22)

4. If you want connect to Apache server you want to open *SSH(22)+ HTTP(80)*

***7**. Review Instance Launch:*

Finally review all your details and launch Instances

***8**. Select key pair:*

Create new key pair → Give key pair name → Download key pair → Launch

SSH Key Pair:

When you create New Key pair it generates public and private keys, public key is deployed in EC2 and private key is downloaded by us.

For remote server authentication we can use

1. Username with password

2. Username with private key (Compared to Username with password it is More secure)

Login to the remote server:

1. For connecting to linux remote servers we use SSH (22)(Secure Shell)
2. For connecting to Windows remote servers we use RDP(3389) (Remote Desktop Protocol)

To connect from our Windows laptop we can use

1. Putty
2. Git Bash
3. Linux Machine(Redhat/Ubuntu…)
4. Powershell
5. Mac Terminal(For Apple Laptops)

AWS (AMAZON WEB SERVICES)

From Git Bash:
Use the *chmod* command to make sure that your private key file isn't publicly viewable. For example, if the name of your private key file is my-key-pair.pem, use the following command:
chmod 400 suresh.pem
Command to connect Remote EC2 from our Laptop
ssh -i Privatekey_File_Location username@Pubiicip or public DNS
Where i indicates Identity file
Ex: *ssh -i /c/Users/Admin/Downloads/suresh.pem ec2-user@192.172.36*

Public IP And Elastic IP:
Public IP or Elastic IP are used to expose the instance to internet.
Public IP:
Public Ip is dynamic public, If we stop the server the public IP is released back to public IP pool, when we start the server AWS picks a new public IP from the pool.
Elastic IP:
Elastic IP is a static ip, even if we stop the server IP remains with us.
We can disassociate EIP from the instance and can associate with other instance.
Note: EIP is associated with non running EC2 instances are charged.
Allocating EIP and associating with EC2:
From AWS Console → EC2 → Elastic IPs → Allocate new address → Allocate
After Creation of New Elastic IP you want to attach the Elastic IP to the Instance
Select the EIP → From Actions → Select Resource Type as Instance → Choose Instance → Select Private of that Instance → Allocate

Public Subnet:
A subnet which is directly exposed to internet is called as public subnet. We make subnets public by adding *Internet Gateway*. Internet Gateway give both inbound and outbound access.
We use public subnets for internet facing servers like web servers.
Private Subnet
A subnet which is not directly exposed to internet is called as private subnet.
For example, database servers and any server which should not be exposed to internet should be part of private subnet.

AWS (AMAZON WEB SERVICES)

How to Give Internet Access to Private Subnet:

1. For giving internet access to private subnet we can use either NAT Instance or NAT Gateway.
2. NAT provides only outbound access but not inbound access.
3. NAT Instance and Nat Gateway are more secure than Internet Gateways.
4. In case of Internet Gateway every machine requires public ip which is a limited resource over internet.
5. In case of NAT only one public IP is required

NAT Instances:

Network Address Translation (NAT) instance in a public subnet in your VPC to enable instances in the private subnet to initiate outbound IPv4 traffic to the Internet or other AWS services, It allows only outbound traffic and restricts inbound traffic.

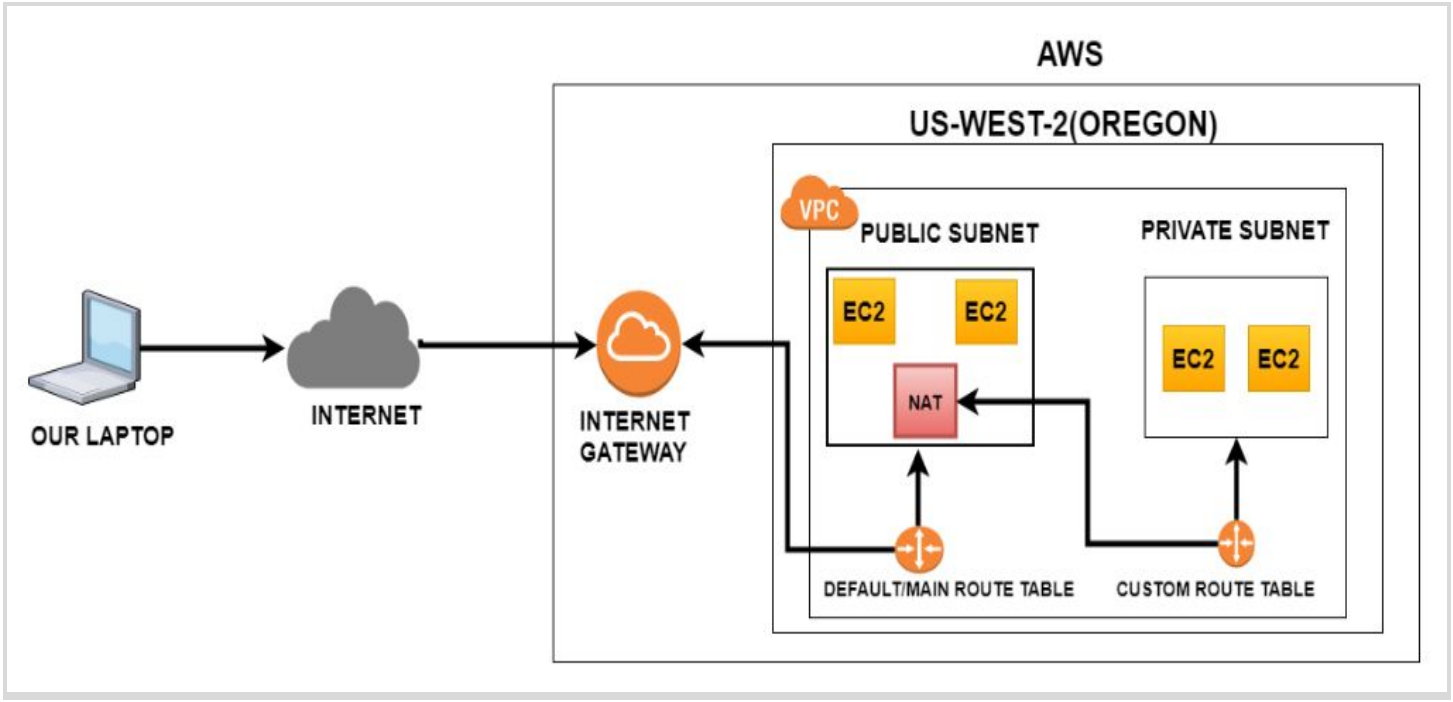
NAT must be launched in public subnet with public IP.

Source and destination check in Instance:

In AWS all EC2 instance are enabled with source/destination check i.e. EC2 must be the source and destination for the traffic otherwise the traffic is blocked.

In case of NAT it is not the source and destination of the traffic, so we need to disable the source and destination check.

Implementation of one Public, One Private Subnet and attaching NAT gateway to Private Subnet:



AWS (AMAZON WEB SERVICES)

1. *First Create VPC with CIDR → 192.168.0.0/26, and name it as "NAT-DEMO-VPC"*
2. *Next create Subnet-1 with CIDR → 192.168.0.0/27 and name it as "PUBLIC-SN"*
3. *Next create one more subnet-2 with CIDR → 192.168.0.32/27 and name it as "PRIVATE-SN"*
4. *Next name default route table as "Public route table"*
5. *Next create one more route table with name "Private Route Table"*
6. *Select "Public route table" → Subnet associations → Edit → add "PUBLIC-SN" → Save*
7. *Select "Private route table" → Subnet associations → Edit → add "PRIVATE-SN" → Save*
8. *Next create one internet gateway and name it as "NAT-DEMO-IGW"*
9. *Next attach that IGW to "NAT-DEMO-VPC"*
10. *Select "Public routetable" → Routes → Add another route → Destination should be 0.0.0.0/0 → Target should be "IGW" → Save.*
11. *From VPC Dashboard → NAT Gateways → Create NAT Gateway → Select "PUBLIC-SN" → Create new EIP → Create NAT gateway*
12. *Select "Private routetable" → Routes → Add another route → Destination should be 0.0.0.0/0 → Target should be "NAT" → Save*

Finally we created one public subnet by adding "IGW" to the Public Subnet and one private subnet by adding the "NAT Gateway" to the Private Subnet.
Now we can launch any "Web servers" which you want to expose to internet in "Public Subnet" and "Database servers" which you don't want to expose to internet in "Private Subnets".

Attaching NAT Instance to Private Subnet: If you want to attach NAT Instance instead of NAT gateway, we can achieve by using following steps.
Launch NAT instance in public subnet.
AWS Console → EC2 → Launch Instance → Select Community AMIs → Search for NAT and select any one from the list → Select t2.micro → Select "NAT-DEMO-VPC" → Select "PUBLIC-SN" → Enable auto assign public IP → Give tags → Select Security group → Review and launch → Select key pair and launch
Disable source and destination check for NAT.
Select Nat Instance → Actions → Networking → Change Source/Dest Check → And click disable → Select "Private routetable" → Routes → Destination-"0.0.0.0/0" → Target Should be "NAT Instance" → Save

Address: *MP Techno Solutions*, Above Titan ShowRoom, Marathahalli, Bangalore, Phone: 9606955170/1/2/3

AWS (AMAZON WEB SERVICES)

Comparison of NAT Instances and NAT Gateways:

Attribute	NAT gateway	NAT Instance
Availability	Highly available.	Use a script to manage failover between instances
Bandwidth	Can scale up to 45 Gbps.	Depends on the bandwidth of the instance type.
Maintenance	Fully Managed by AWS	Fully Managed by you
Cost	Charged depending on the number of NAT gateways you use, duration of usage, and amount of data that you send through the NAT gateways.	Charged depending on the number of NAT instances that you use, duration of usage, and instance type and size.
Public IP addresses	Choose the Elastic IP address to associate with a NAT gateway at creation.	Use an Elastic IP address or a public IP address with a NAT instance
SUITABLE FOR	Production Environments	Local Environments like Dev,TEST,UAT
Security groups	Cannot be associated with a NAT gateway.	You can associate
Network ACLs	Use a network ACL to control the traffic to and from the subnet in which your NAT gateway resides.	Use a network ACL to control the traffic to and from the subnet in which your NAT instance resides.

Enabling DNS Host names for VPC:

By default DNS Hostnames are disabled for every newly created VPC, so if you want to enable DNS Hostnames follow the steps.

From VPC Dashboard → Select VPC → Actions → Edit DNS Hostnames → Select Yes → Save

VPC-SECURITY:

We secure VPCs in two ways

1. Security Groups

2. Network ACLs (Access Control List)

Security Groups: A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. When you launch an instance in a VPC, you can assign up to five security groups to the instance. Security groups act at the instance level, not the subnet level.

Security Group Characteristics:

1. Every instance must have at least one security group
2. Same security group can be associated with multiple EC2 instances within the same VPC.
3. Every EC2 can have max 5 security groups.
4. Security groups are **stateful.**
    a. If a traffic is initiated from internet, this traffic is validated by inbound rules of security groups, no matter what is the outbound rule the traffics leaves out.
    b. If a traffic is initiated from EC2, this traffic is validated by outbound rules of security groups, no matter what is the inbound rules the traffics comes in.
5. Security groups does not have explicit Allow/Deny, Rules we add is to allow and others are implicitly denied.

Network ACLs:

A network access control list (NACL) is an optional layer of security for your VPC that acts as a firewall at **Subnet level**.

Network ACLs Charastics:

1. When VPC is created NACL is implicitly created, this is called as default or Main NACL.
2. Default NACL allows all inbound and outbound traffic.
3. Any subnet created is implicitly associated with default NACL.
4. Multiple subnets can be associated with one NACL.
5. One subnet can be associated with only one NACL.
6. Every rule can be explicitly allowed or denied.
7. Network ACLs are **stateless** that means inbound traffic is controlled by inbound and outbound traffic is controlled by outbound rule.
8. We have options to block an ip address/network using NACL.

9. In NACLs rules are evaluated i n **ascending** order.

VPC Peering:

VPC peering enables the communication between two resources which are present in two different VPCs.

`

Notes:

1. The owner of the requester VPC sends a request to the owner of the accepter VPC to create the VPC peering connection.
2. The accepter VPC can be owned by you, or another AWS account.
3. CIDR blocks of both VPCs must not coincide.
4. The owner of the accepter VPC accepts the VPC peering connection request to activate the VPC peering connection.
5. To enable traffic flow between both VPCs by using Private IP, the owner of each VPCs in the VPC peering connection must manually add the routes in both route tables and that points to other peered VPC.

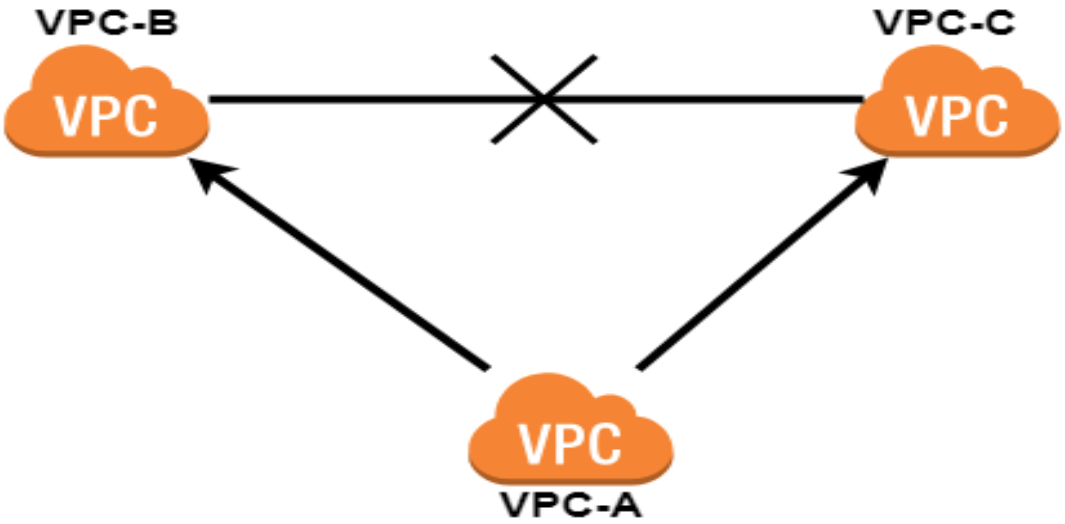
Implementation of VPC Peering Connection:

1. First you create two VPCs with CIDR 10.0.0.0/16 as VPC-1 and 192.168.0.0/16 as VPC-2
2. Create one public subnet in VPC-1 and name it as “Public Subnet” and one Private Subnet in VPC-2 and name it as “Private Subnet”
3. Create one Public instance in Public Subnet and name it as “Public instance” and create one Private Instance in Private subnet and name it as “Private Instance”
4. Now our aim is we want to access Private Instance, but both instances are in different VPCs, So to achieve this we want to create “VPC Peering” connection between both VPCs.
5. *From VPC Dashboard → VPC Peering Connections → Create Peering Connection → Select Tag Name → Select Requester VPC id → Select Accepter VPC id → Next Create Peering Connection*
6. *Next Select VPC Peering Connection from Accepter VPC Dashboard → Actions → Accept*

7. Next Modify routing tables in both the VPCs
8. To check this VPC Peering Connection first login to the Public Instance and from there login to the private instance with private ip.
9. If you login successfully, your both VPCs are Peered successfully.

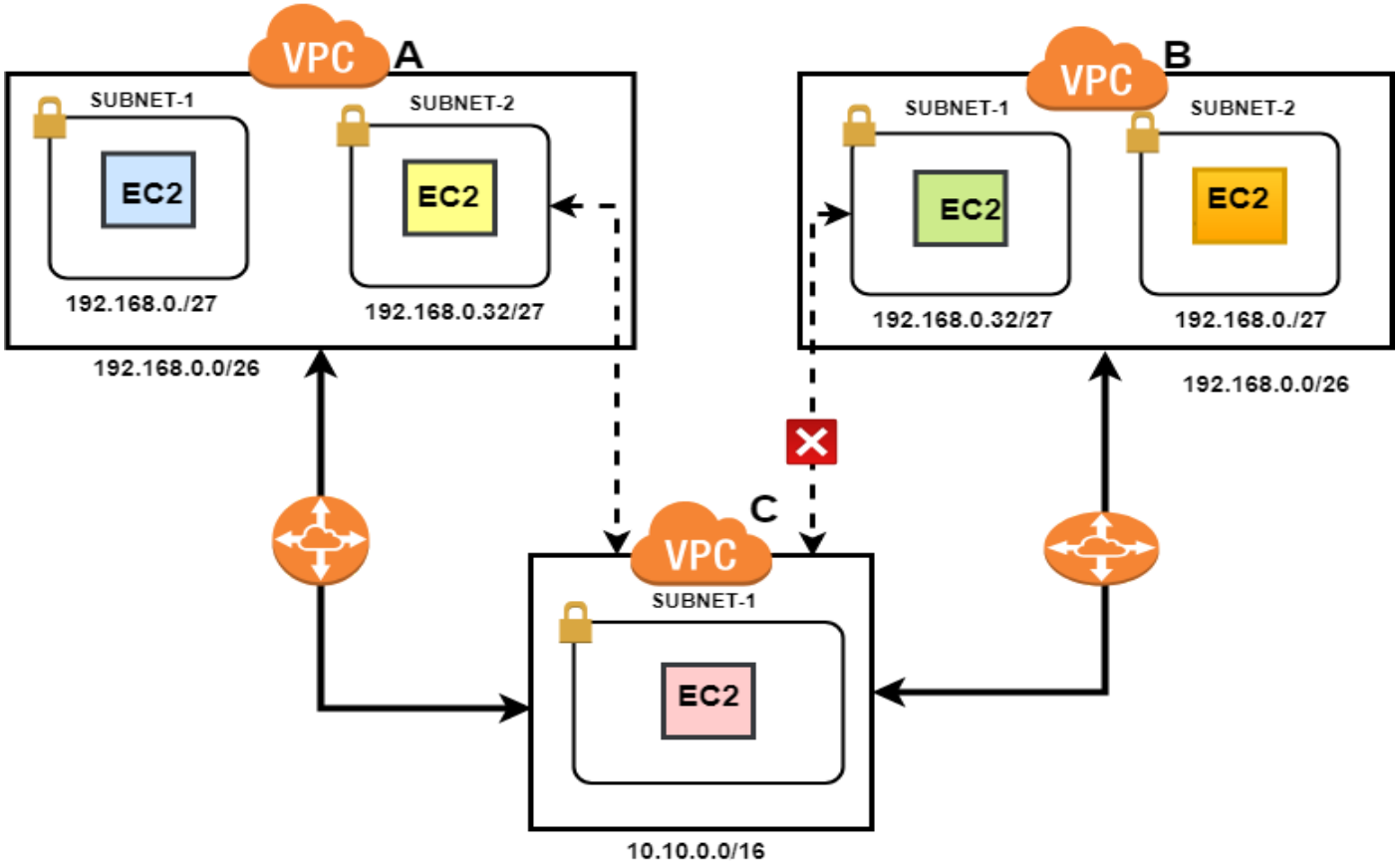
VPC Peering Limitations:

1. For creating VPC Peering connection, the CIDR blocks of both VPCs must not coincide.
2. You cannot have more than one VPC peering connection between the same two VPCs at the same time.
3. Active VPC peering connections per VPC default limit is 50, how ever we can increase upto by writing email to AWS Support team.
4. VPC peering does not support **transitive peering**, that means you created VPC peering between VPC-A and VPC-B, and created one more connection between VPC-A and VPC-C, it's doesn't meant that VPC-B and VPC-C are also Peered.



VPC PEERING DOES NOT SUPPORT TRANSITIVE PEERING

5. **Unicast reverse path forwarding** in VPC peering connections is not supported.



6. Any tags that you create for your VPC peering connection are only applied in the account or region in which you create them.

AWS EC2 UserData: If you want to run any scripts or if you want to install any packages in your instances **while launching time** EC2 UserData is used. In userdata there is no need to explicitly mention “sudo” because it is already in s”sudo” mode.

Real time sceneries:

1. If you want *Cloudwatch log agent* in all machines while launching of EC2 instances.
2. In *ansible* all control/target machines requires *Python* at this time to install python on all target/control machines EC2 UserData is used.
3. If you want to connect target machines by using *Chef/Puppet* target machines should contain *Chef/Puppet* agents, at that time EC2 UserData is used.
   Etc..

Implementation:

From EC2 Dashboard → Launch Instance → Step 3: Configure Instance Details → At the end of the page → advanced details → Userdata

There are two ways of introducing userdata

1. Enter directly.
2. Save all script in single file and upload.

Example:

If you want to install apache server in the instances, use following script.

```
#! /bin/bash
yum install httpd -y
service httpd start
chkconfig httpd on
echo " Welcome to Apache Server " > /var/www/html/index.html
```

Elastic Load Balancers (ELB):

- Elastic Load Balancing automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances, containers, and IP addresses.
- It can handle the varying load of your application traffic in a single Availability Zone or across multiple Availability Zones.

ELB Characteristics:

- ELB is region specific, that means you can load balance your instances within the same region.
- If you want to load balance your instances in different regions we can use ***Route53***
- ELB supports both internal and external load balancers, that means if you choose internal load balancers you can load balance only that instances which are not publicly accessible like *DB servers* if you didn't choose internal load balancers it wii load balance all servers which are publicly accessible like *Web Servers.*
- ELB does health checks on instances and routes traffic to healthy instances only, if it finds any instance unhealthy that instance is taken out of rotation, if ELB finds unhealthy instances are healthy, it brings them automatically into rotation.
- AWS maintains redundant copies of ELB, if any AZ fails it automatically failovers to the ELB in different AZ.
- We can also secure your ELB with Security Groups.

Types of ELB:

There are three types of ELSs are there, those are,

1. Classic Load Balancer
2. Application Load Balancer
3. Network Load Balancer

Classic Load Balancer:

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.

Implementation of Classic Load Balancer:

- Before starting of creation of Classic Load Balancer we want minimum 2 instances which are serving any web application.
- So we install 2 instances in same VPC in different availability zones and also install apache server and GIT on the top of the instances by using AWS EC2 UserData.
- Next we can login both servers and clone our project code from github by using "*git clone*" command.
  Ex: *git clone https://github.com/rcp-suresh/static*
- Next move the total content present in static folder to apache deployment directory.
  Ex: *mv /home/ec2-user/static/* /var/www/html/*
- Before stating the load balance between the servers first you check whether that two servers are serving properly by using public IPs.
- Next From EC2 DashBoard → Load Balancers → Create Load Balancer → Select Classic Load Balancer → Create

AWS (AMAZON WEB SERVICES)

STEP-1

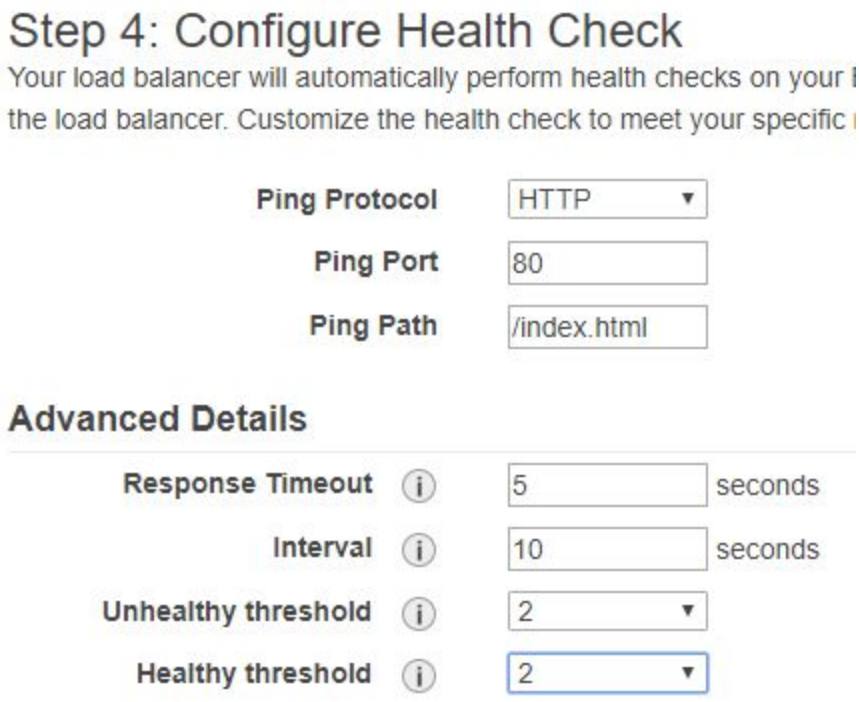
Load Balancer name: ApachE-ELB
Select VPC: Default
Create an internal load balancer: Don't Check
Enable advanced VPC configuration: Check the Box
Load Balancer Protocol: HTTP
Load Balancer Port: 80
Instance Protocol: HTTP
Load Balancer Port: 80
Select all subnets

STEP-2

Select Security Group

STEP-3

If you Select HTTPS in STEP-1 you want to upload SSL certificate, in our case we didn't select HTTPS so we can skip this step

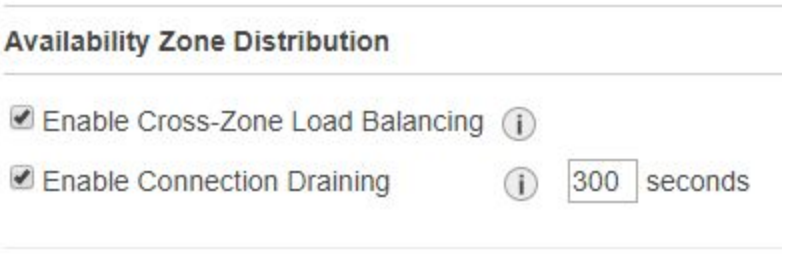


- Where **Healthy Threshold** is The number of consecutive successful health checks that must occur before declaring an EC2 instance healthy. Valid values: 2 to 10 and Default: 10.
- **Unhealthy Threshold:** The number of consecutive failed health checks that must occur before declaring an EC2 instance unhealthy. Valid values: 2 to 10 and Default: 2

- **Health Check Interval:** The amount of time between health checks of an individual instance, in seconds. Valid values: 5 to 300 and Default: 30
- **Response Timeout:** The amount of time to wait when receiving a response from the health check, in seconds. Valid values: 2 to 60 and Default: 5

STEP-5

- Add desired instances to load balance

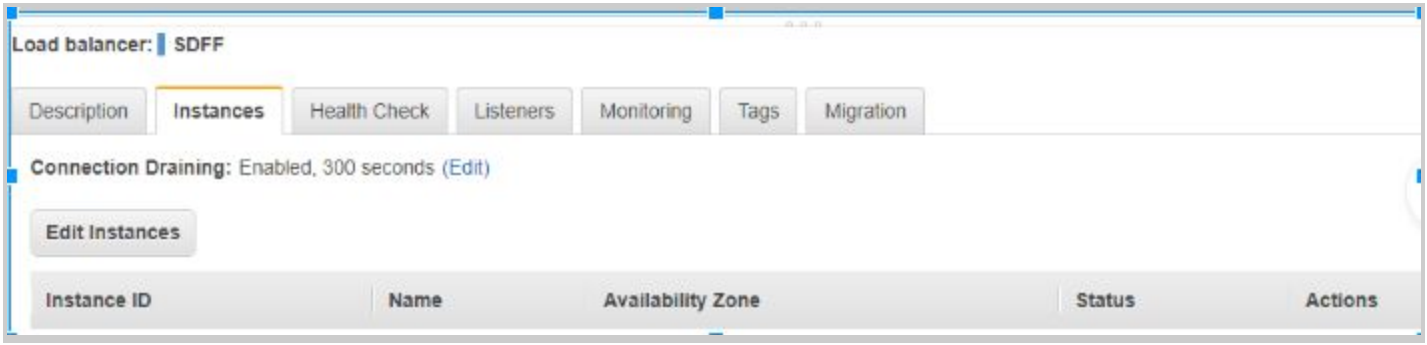


- **Crosszone Load balancing:**IF it is enabled all the instances in all subnets get equal load and If it is not enabled each subnet get gets equal loads.
- **Connection draining in ELB:** It is the time given to the unhealthy instances to complete the processing of all current open connections before taking the instances out of range.

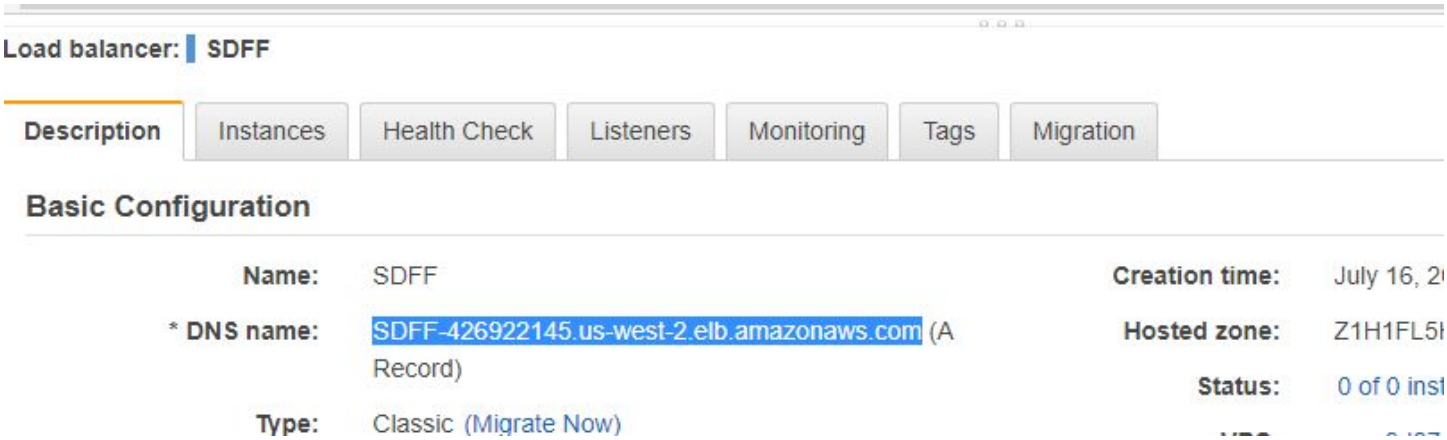
STEP-6

Add Tags

- Review and Create



Make sure that all instance status should be healthy.



From Description section take DNS name and access from web browser.

Stickiness In ELB: Stickiness enables the load balancer to bind a user's session to a specific instance. This ensures that all requests from the user during the session are sent to the same instance.

Types of Scaling:

We can scale our application by two methods

1. Vertical Scaling
2. Horizontal Scaling.

Vertical Scaling: If you want to improve server performance your instance should have higher configuration like number of CPUs and RAM, in Vertical scaling we can Increase instance size by replacing smaller instance type with higher instance type.

Implementation:

Select the Stopped Instance → Actions → Instance Settings → Change Instance Type → Choose Desired Instance Type → Apply

Horizontal Scaling: Horizontal scaling means adding of additional servers to the environment, this we can achieve by using Auto Scaling.

Auto Scaling:

Amazon EC2 Auto Scaling helps you ensure that you have the correct number of Amazon EC2 instances available to handle the load for your application. You create collections of EC2 instances, called Auto Scaling groups. You can specify the minimum number of instances in each Auto Scaling group, and Auto Scaling ensures that your group never goes below this size. You can specify the maximum number of instances

in each Auto Scaling group, and Auto Scaling ensures that your group never goes above this size. If you specify the desired capacity, either when you create the group or at any time thereafter, Auto Scaling ensures that your group has this many instances. If you specify scaling policies, then Auto Scaling can launch or terminate instances as demand on your application increases or decreases.

Note: Auto scaling service is free, however instances launched by auto scaling is chargeable.

Launch configurations: Your group uses a launch configuration as a template for its EC2 instances. When you create a launch configuration, you can specify information such as the AMI ID, instance type, key pair, security groups, and block device mapping for your instances.

Auto Scaling groups: Your EC2 instances are organized into groups so that they can be treated as a logical unit for the purposes of scaling and management. When you create a group, you can specify its minimum, maximum, and, desired number of EC2 instances.

Custom AMI(Amazon Machine Image): AMI is template which contains, Operating System and pre installed softwares

- By default AMI created by us is private
- We also can share with other AWS accounts
- We also can make it public (We can find under Community AMIs)
- When we share this image, this image is accessible from the same region.

Implementation of Auto Scaling:

Note: Before going to Implement Auto Scaling you must create the image from one of the server that image contains OS+Our Apache Application+Our Code.

From AWS Console → EC2 → Auto Scaling → Launch Configuration

STEP-1

Choose AMI → My AMIs → Choose your AMI

STEP-2 to STEP-6

Remaining things are same as like how we created aws ec2 instance

Select Key Pair → Create Launch Configuration

Next Create Auto Scaling Group

STEP-1

Configure Auto Scaling group details

Give the group name “APACHE GROUP”

Select Group Size → That means Select Initially how many instances should starts with

Select VPC

Select all subnets

In advanced Section → Enable Load Balancer Option → Then select load balancer

In Instance protection → select scale in → it enables instance terminate protection

Remaining Things are default

STEP-2

Configure scaling policies

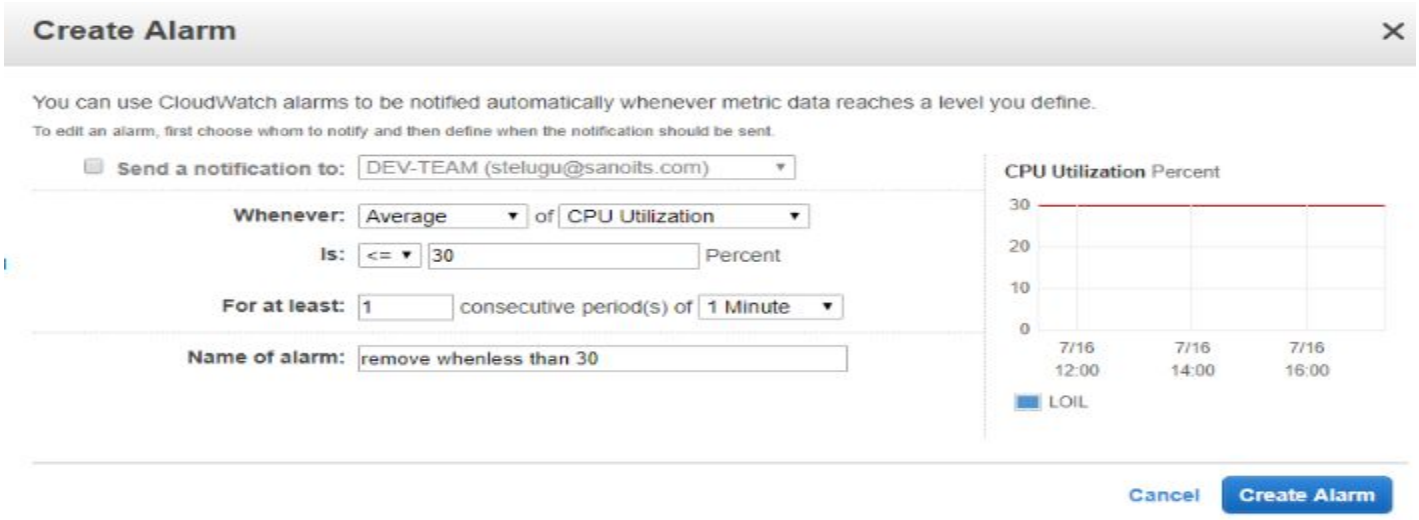
Select "Use scaling policies to adjust the capacity of this group"

Select Minimum size and maximum size of Auto Scaling Group

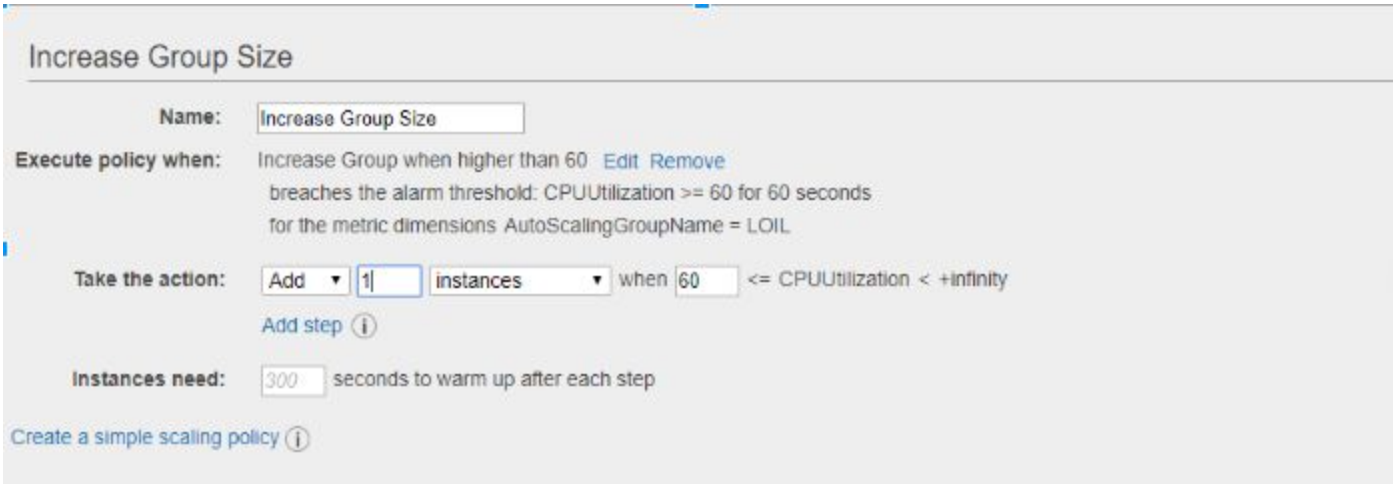
Select min-1 and max-5

Next select "Scale the Auto Scaling group using step or simple scaling policies"

Increase Group Size → Name it → Add new alarm



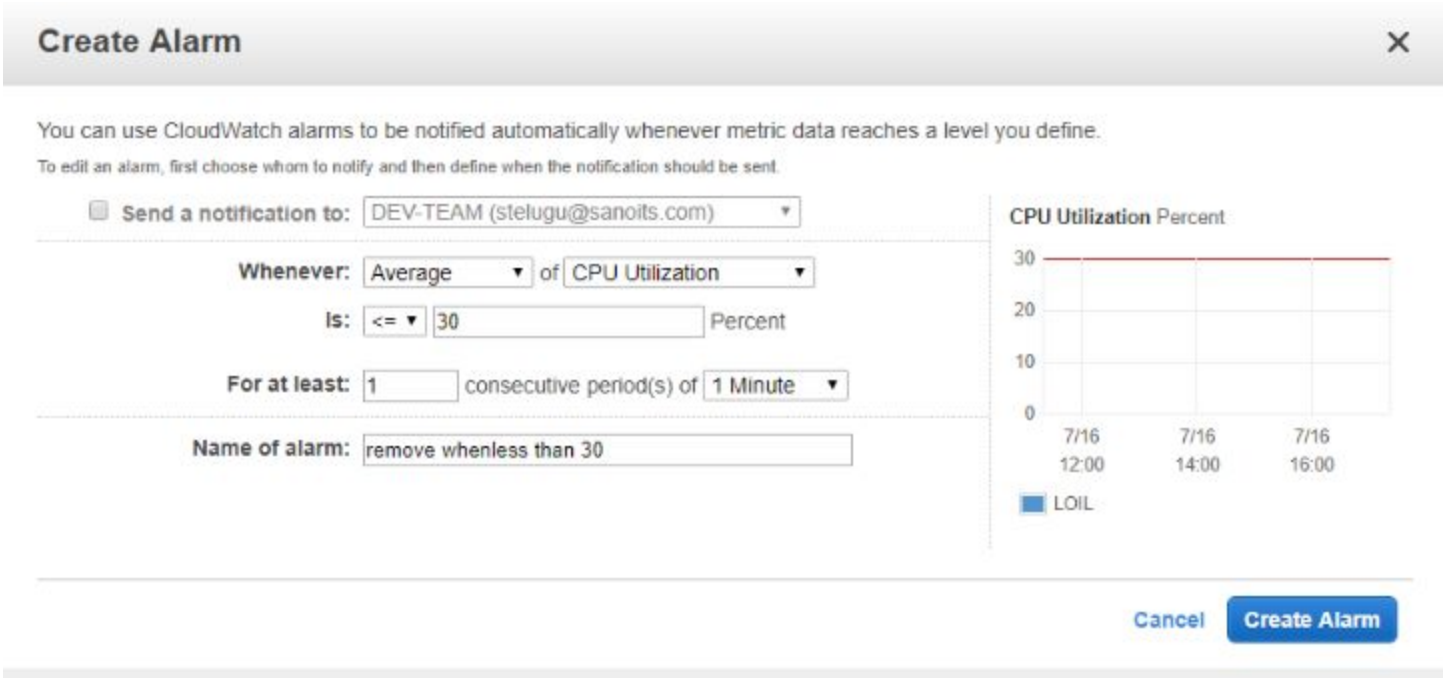
Create alarm



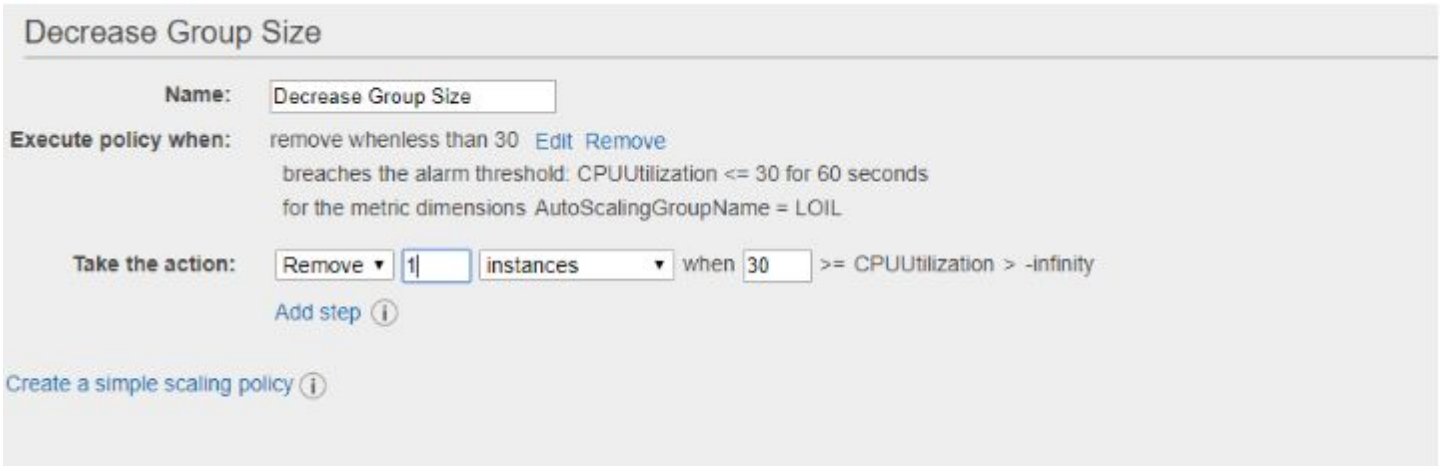
In Decrease Group Size

Name it

Add new alarm



Create alarm



Like this you can create multiple rules

STEP-3

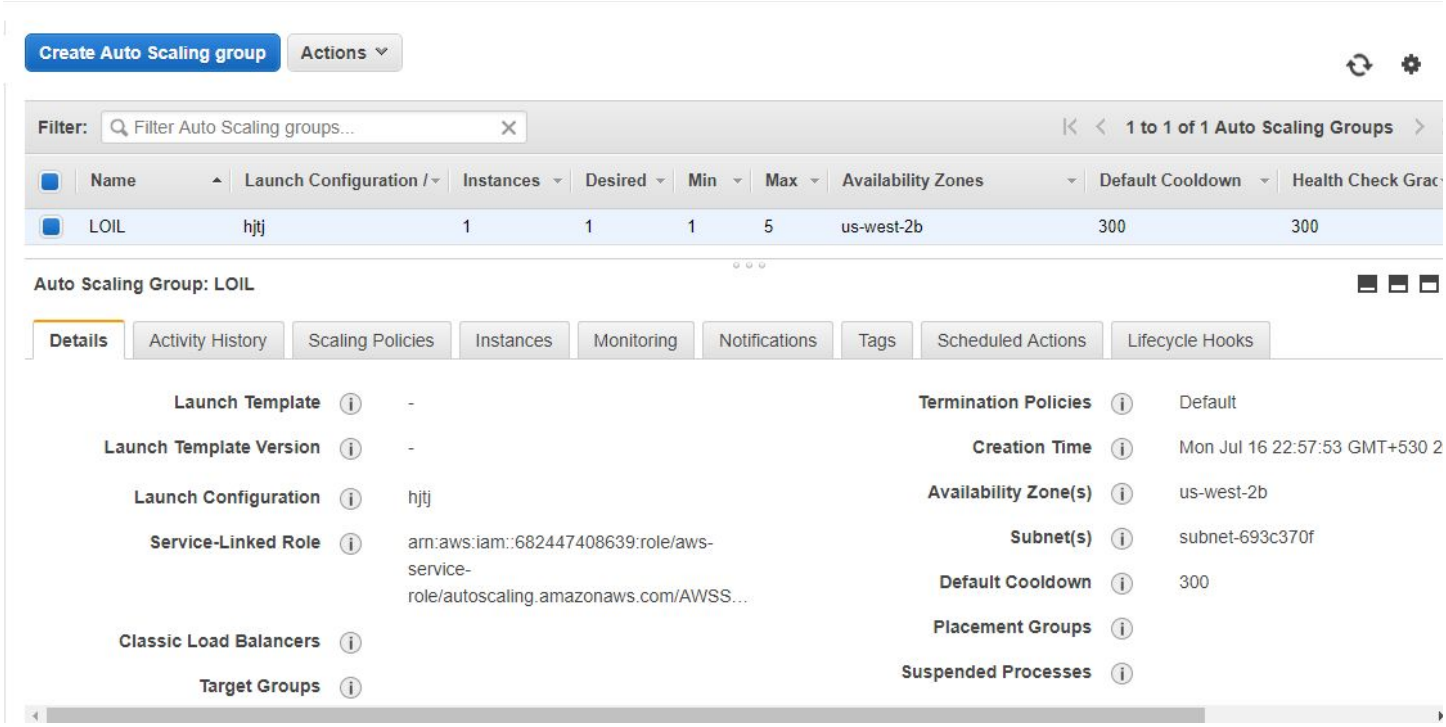
If you want latest notifications you can assign this to SNS topic, so when ever any changes present in this auto scaling group you will get notifications via email.

STEP-4

Give Tags

Next → Review → Create auto scaling Group

Finally you will get



Testing:

If you want to check whether this auto scaling group is working properly as per your rules there is an option to manually increase the load and decrease the load.

Select the Auto Scaling Group → Scaling Policies → Select Increase Group Size → Actions → Execute → Enter 90

90% is above 60% so it will create the one instance as per our Scaling Policies.

AWS (AMAZON WEB SERVICES)

Amazon Elastic Block Store (Amazon EBS):

Amazon Elastic Block Store (Amazon EBS) provides block level storage volumes for use with EC2 instances. EBS volumes are highly available and reliable storage volumes that can be attached to any running instance that is in the same Availability Zone.

Features of Amazon EBS:

1. You can mount multiple volumes on the same instance, but each volume can be attached to only one instance at a time.
2. You can only increase the size of EBS volumes, but not decrease.
3. To attach the volume both volume and EC2 must be in the same AZ
4. EBS volumes are created in a specific Availability Zone, and can then be attached to any instances in that same Availability Zone. To make a volume available outside of the Availability Zone, you can create a snapshot and restore that snapshot to a new volume anywhere in that region.
5. We can detach a volume from a EC2 and attach to other instance
6. We can take snapshots (backups), and we also can restore from snapshot.
7. EBS Volumes support incremental snapshots.

Adding new volume and mounting it to EC2:

From EC2 Dashboard → Volumes → Create Volume → Select volume type, size, and availability zone and tag → Create

Select the Volume → Actions → Attach Volume → Select Instance → Attach

Making an Amazon EBS Volume Available for Use on Linux:

1. Use the *lsblk* command to view your available disk devices and their mount points.

   Ex:

   ```
   [ec2-user ~]$ lsblk
   NAME  MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
   xvdf  202:80   0  100G  0 disk
   xvda1 202:1    0    8G  0 disk /
   ```

2. Use ***sudo file -s device*** command to check if filesystem is already created.

   Ex:

   ```
   [ec2-user ~]$ sudo file -s /dev/xvdf
   /dev/xvdf: data
   ```

3. If the output of the previous command shows simply data for the device, then there is no file system on the device and you must create one.

4. If you run this command on a device that contains a file system, then your output will be different.
   Ex:
   [ec2-user ~]$ **sudo file -s /dev/xvda1**
   /dev/xvda1: Linux rev 1.0 ext4 filesystem data, UUID=1701d228-e1bd-4094-a14c-8c64d6819362 (needs journal recovery) (extents) (large files) (huge files)
5. If there is no file system present we want to create new file system, for to create new file system use following command.
   sudo mkfs -t ext4 *device_name*
   Ex: *sudo mkfs -t ext4 /dev/xvdf*
6. For to create a mount point use following command,
   sudo mkdir *mount_point*
   Ex: s*udo mkdir /xyz*
7. Use the following command to mount the volume at the location you just created.
   sudo mount *device_name mount_point*
   Ex: *sudo mount /dev/xvdf /xyz*
8. Make the above changes persistent
   sudo vim /etc/fstab
   add the following entry to the end of file and save
   /dev/xvdf /xyz ext4 defaults 0 0

Amazon EBS Snapshots:

You can back up the data on your Amazon EBS volumes to Amazon S3 by taking point-in-time snapshots. Snapshots are incremental backups, which means that only the blocks on the device that have changed after your most recent snapshot are saved. This minimizes the time required to create the snapshot and saves on storage costs by not duplicating data.

Taking EBS snapshots:

EC2 Dashboard→ Volumes → Select the volume → Actions → Add tags to the Snapshot → Create snapshot

Restoring Volumes from snapshots:

EC2 Dashboard → Snapshots → Select the snapshot → Actions → Create volume

While restoring we can choose

- Different volume type
- Different availability zone
- Different size

AWS (AMAZON WEB SERVICES)

EBS Volume Types:

1. General Purpose SSD (Solid State Disk)
2. Provisioned IOPS SSD
3. Throughput optimized HDD
4. Cold HDD
5. Magnetic
6. Instance Store

Note:

1. For root volume we can choose any one from General Purpose SSD, Provisioned IOPS SSD and Magnetic. As a extra EBS volume we can choose any one from 5 types.
2. We can get upto 65000 IOPS by attaching multiple volumes.
3. By configuring RAID we can increase IOPS beyond 65000

General Purpose SSD (Solid State Disk): These volume type supports large set of workloads.

- EBS volumes accumulates credit points when it is idle.
- This volume types can perform above baseline when required, by spending credit points.
- This volume type is good fit for (Dev, Testing, Staging environments)
- Minimum 100 and max 10000/volume IPOS

Provisioned IOPS:

This volume type provides extreme performance, it designed for machine critical applications.

- Max IOPS, 20000/volume
- Oracle, Sybase, DB2, MySQL, mongodb etc.. are suitable for this type.
- Good choice for production workloads, not recommended for lower environments(dev, testing, staging) in order to save the cost.

Throughput Optimised HDD:

- This volume type is a good fit for processing big data.
- Like Data warehousing, Big Data, Informatica etc…
- Throughput and sequential reads are dominant performances attributes
- Gives faster access to data
- Use this type where we access data more frequently

Cold HDD:

- Works similar to Throughput Optimized HDD, storage cost is cheaper
- But this type has additional retrieval charges

- Good option for infrequently accessed data

Magnetic:

Magnetic is an old generation storage type, we should always consider other types except if our application is specially optimized for this type.

Instance Store:

- Is a temporary store
- Data stored on this store is lost when we power off the Instance
- Storage cost is very very cheap compared with EBS
- Use this types to store temporary data.

Launching EC2 with instance store

EC2 → Launch Instance → Community AMIs → filter by root device type(Instance store)

Recovery Of AWS Linux instance Key Pair:

- Suppose You created an instance with new key pair named as "*ec2key.pem*", after some days you missed that key pair but you want to access the instance. To achieve this follow below steps.
- First you create the New Instance in same availability zone where the missing key pair instance present in, with new key pair. Ex: "*oregon.pem*"
- After that stop the old instance and detach the volume and attach the that volume to the new instance.
- Next login the new instance check "*lsblk*" command whether it is attached or not.
- Next create a new directory in root folder in new instance with name "*recovery*"
  Ex: *mkdir /recovery*
- Next mount the old volume to the "*recovery*" directory.
  Ex: sudo mount /dev/xvdf1 /recovery
- Next copy the public key of the new instance which is present in "*/home/ec2-user/.ssh/authorised_keys"*
- And paste in old volume "*.ssh"* folder which is present in "*/recovery/home/ec2-user/.ssh/authorised_keys"*
- After that stop the new instance and detach the old volume from new instance and attach to the old instance.
  Note: While attaching you want to attach as a **root** volume but not EBS volume.
- Next you can login to the old instance with new key pair, that was created for the new instance.

AWS (AMAZON WEB SERVICES)