Using ***AWS Cloud Formation*** to create a simple client-side VPN.

So what is cloud Formation actually?

AWS cloud Formation is one of the many services available in aws to streamline user interaction with various other services within the AWS ecosystem. It helps the client to provision, set-up, configure the resources(Ex: EC2 instance, RDB instance etc..) and policies attached automatically with ease from a pre-defined template that defined all the said specifications, so that the client can focus on actually focus on maintaining and improving the resources than actually spend resources and time on deploying them individually.

Understanding the use-case:

For a scalable web application with a backend database, you typically use an Auto Scaling group, Elastic Load Balancing, and Amazon RDS. Manually provisioning and configuring these resources can be complex and time-consuming.

Instead, you can use a CloudFormation template to streamline this process. A template defines all your resources and their properties. When you create a CloudFormation stack from the template, CloudFormation provisions and configures the Auto Scaling group, load balancer, and database for you. This approach allows you to manage your resources as a single unit, simplifying deployment and maintenance. You can also easily delete the stack, which removes all associated resources. CloudFormation enables quick replication and efficient management of your infrastructure.

How cloud formation works?

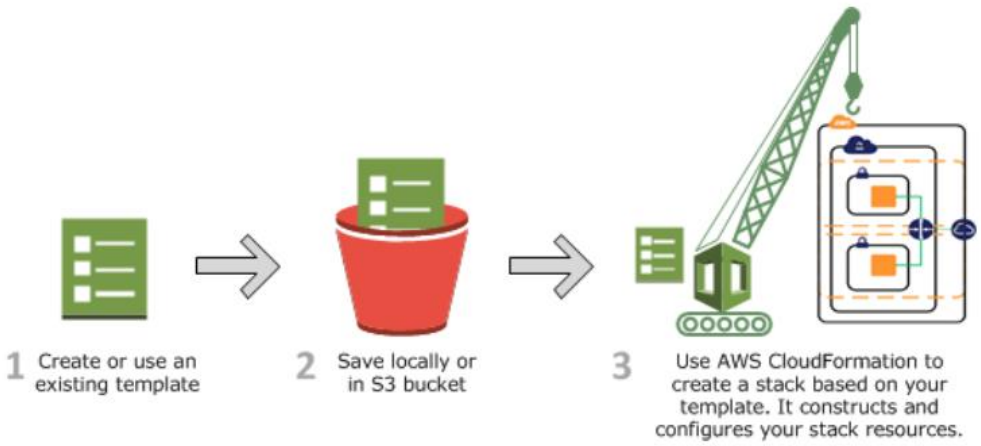
Cloud formation consists of two main key concepts:

. Templates

. Stacks

When creating a stack, CloudFormation makes service calls to AWS to provision and configure resources, acting within your permission limits. For instance, to create or terminate EC2 instances, you need corresponding permissions, managed via AWS IAM.

The template defines all resource actions. For example, a template specifying a t2.micro EC2 instance prompts CloudFormation to call the EC2 create instance API with that instance type.



Example of how a JSON template looks like :

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "A simple EC2 instance",
    "Resources": {
        "MyEC2Instance": {
            "Type": "AWS::EC2::Instance",
            "Properties": {
                "ImageId": "ami-0ff8a91507f77f867",
                "InstanceType": "t2.micro"
            }
        }
    }
}
```

Explantion:

{

"AWSTemplateFormatVersion": "2010-09-09",

. This line specifies the version of the AWS CloudFormation template format you are using. "2010-09-09" is the latest version as of this writing.

"Description": "A simple EC2 instance",

. This line provides a brief description of what the CloudFormation stack will do. In this case, it describes the creation of a simple EC2 instance.

"Resources": {

This line begins the section where you define the AWS resources that CloudFormation will create

"MyEC2Instance": {

. This line specifies a logical name for the resource within the template. "MyEC2Instance" is a unique identifier for this EC2 instance in the template.

"Type": "AWS::EC2::Instance",

. This line defines the type of AWS resource to create. Here, "AWS::EC2::Instance" specifies that the resource is an EC2 instance.

"Properties": {

. This line starts the section where you define properties for the EC2 instance.

"ImageId": "ami-0ff8a91507f77f867",

. This line specifies the Amazon Machine Image (AMI) ID that the EC2 instance will use. The AMI ID "ami-0ff8a91507f77f867" refers to a specific Amazon Linux AMI.

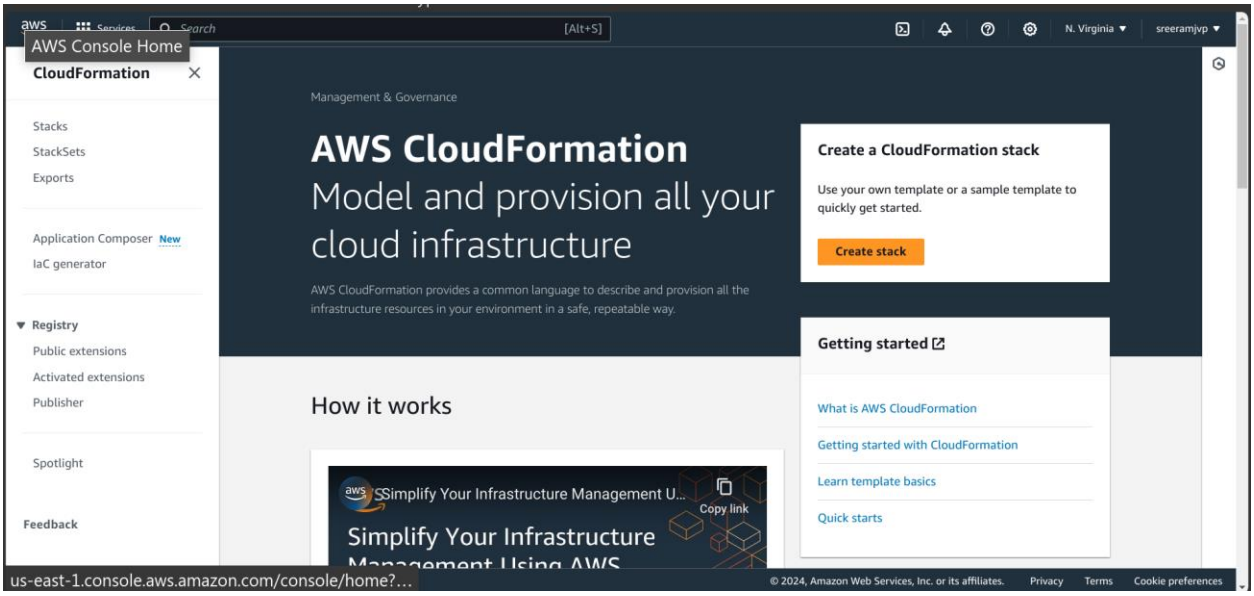
"InstanceType": "t2. micro"

. This line specifies the instance type for the EC2 instance. "t2.micro" is a small instance type suitable for low-traffic applications and testing.

Creating a client side vpn from pre-defined template:

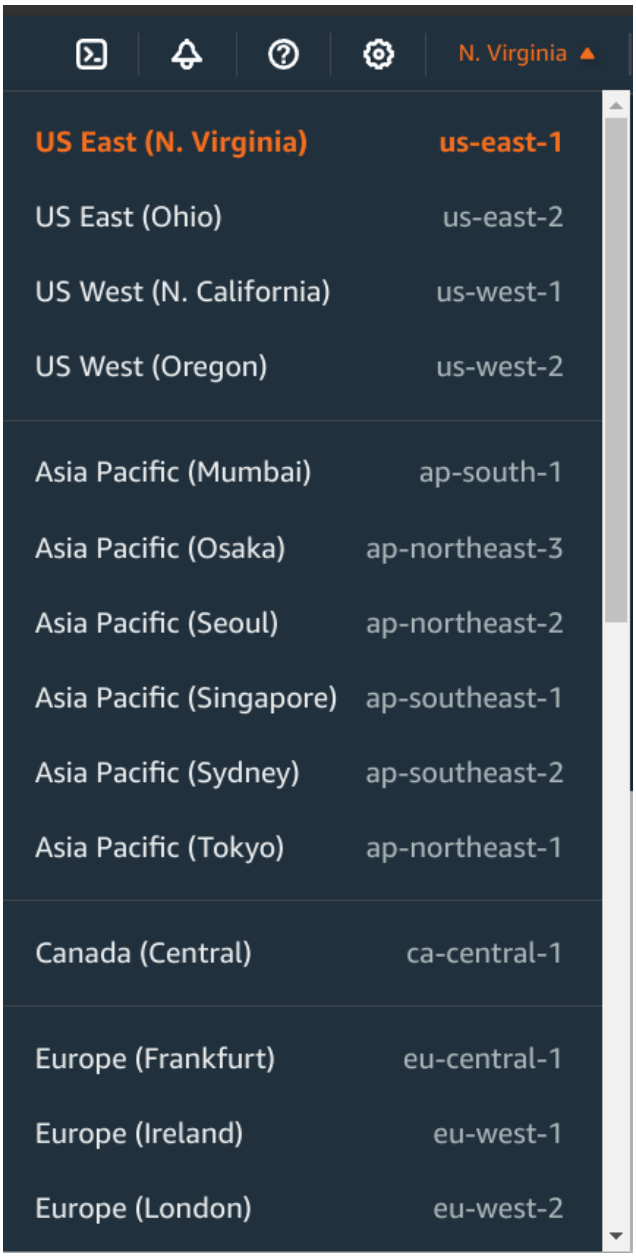
(Step-1):

Home page of AWS cloud formation:



(Step-2)

Region selection:



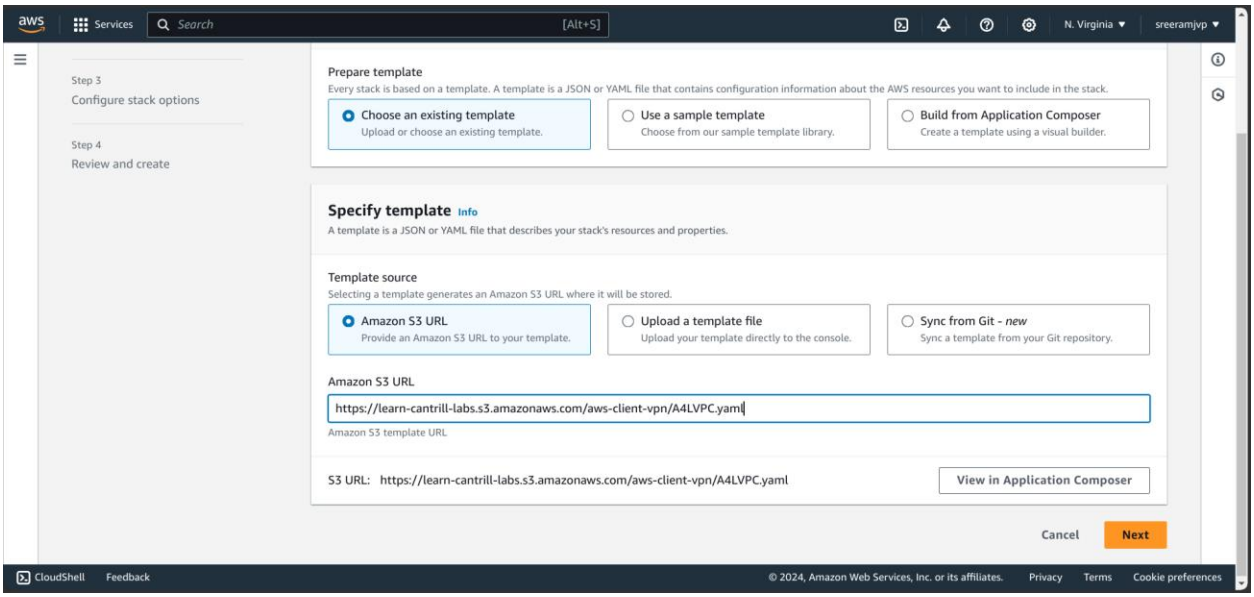
(I will be selecting us-east-1 region)

(Step-3):

Stacks are created using *Templates,* which can be done in three ways:

1. Using a pre-defined template or
2. Create a scratch template using *application composer*

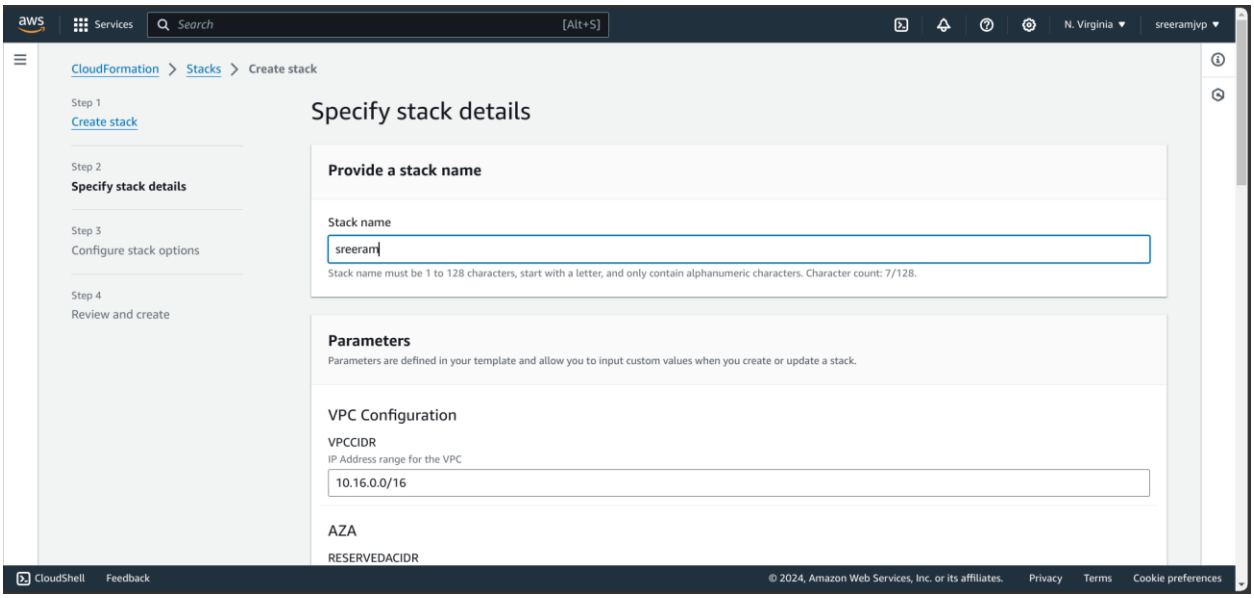
We will be using a pre-defined template, loaded from an amazon S3 URL.*(s3, stands for simple storage service)*.

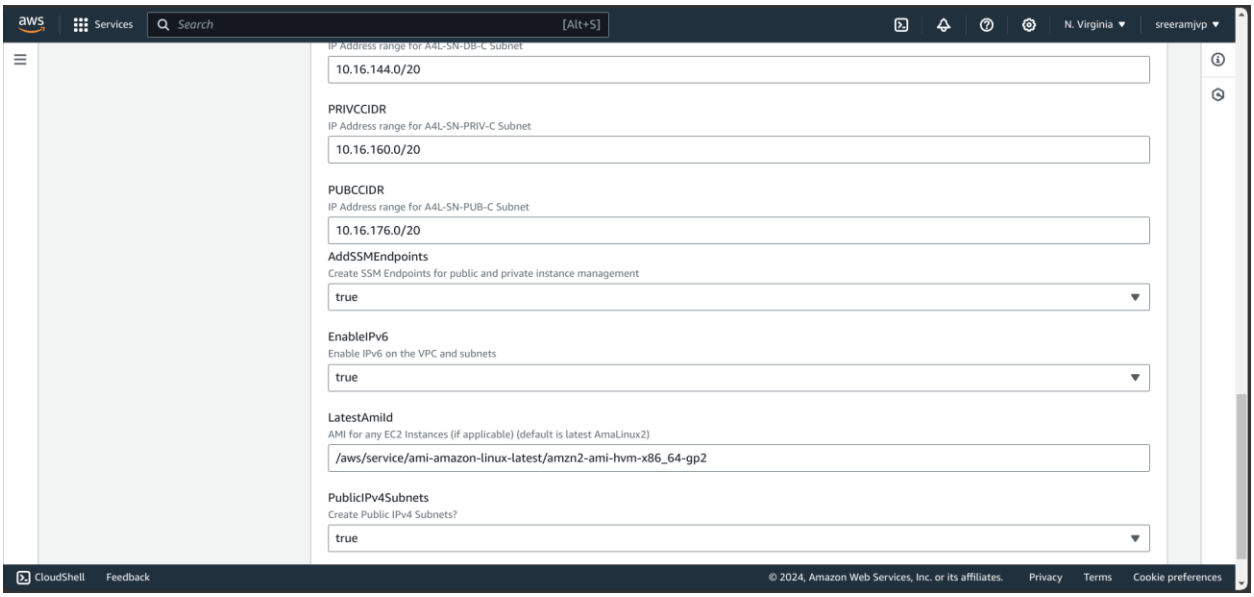


The template is loaded from https://learn-cantrill-labs.s3.amazonaws.com/aws-client-vpn/A4LVPC.yaml.

(Step-4):

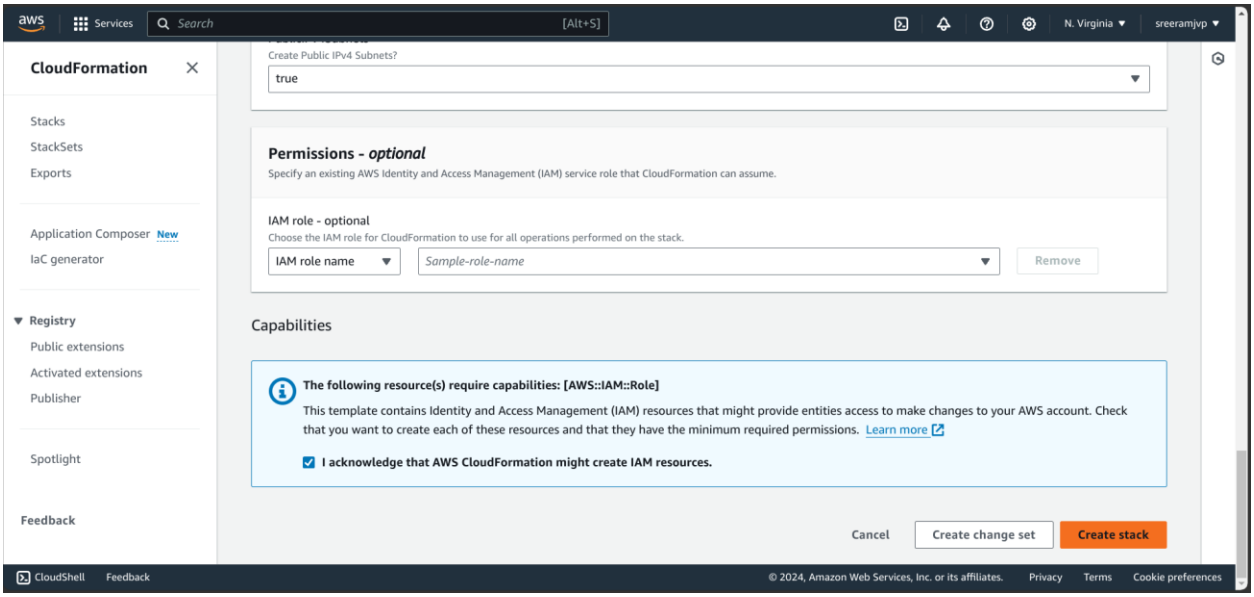
Click next.





As seen, all of the required resources and policies have been defined by the sample template.

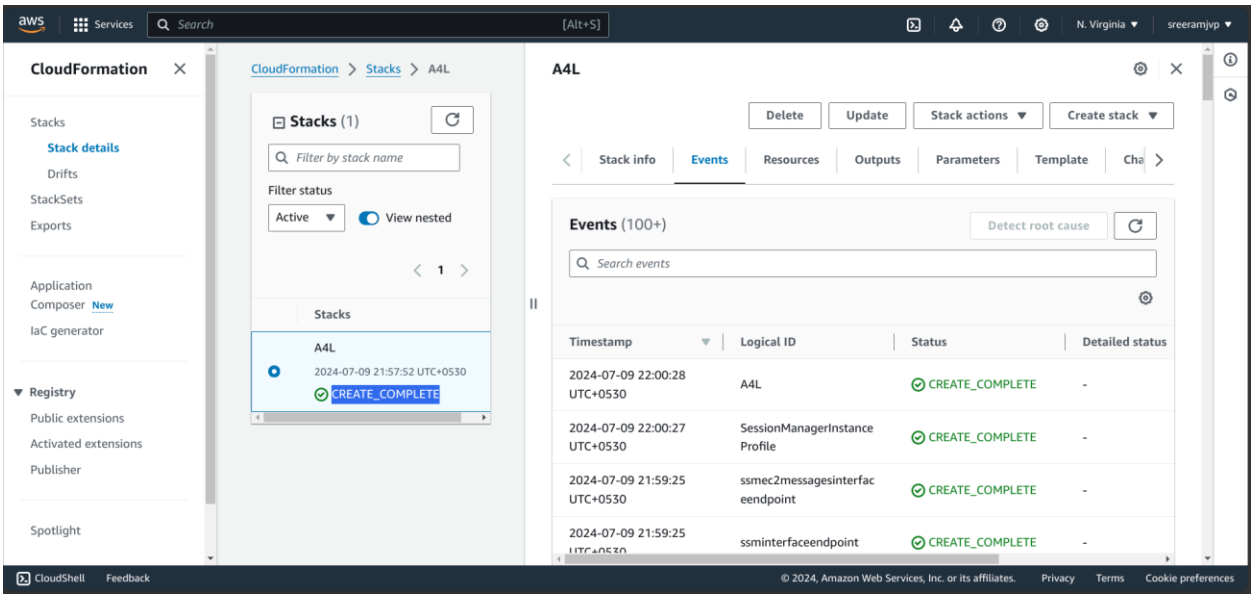
Authorizing IAM role for access and permissions:



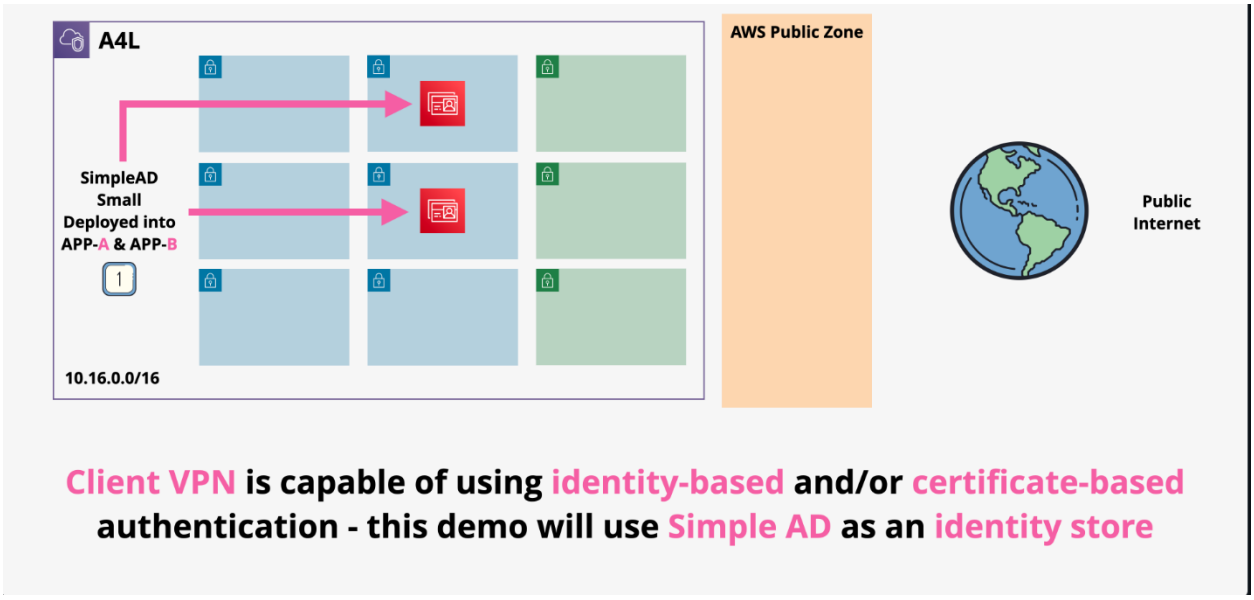
Click ***Create Stack.***

(Step-5):

Wait till the status changed from CREATE_IN_PROGRESS *to* CREATE_COMPLETE.



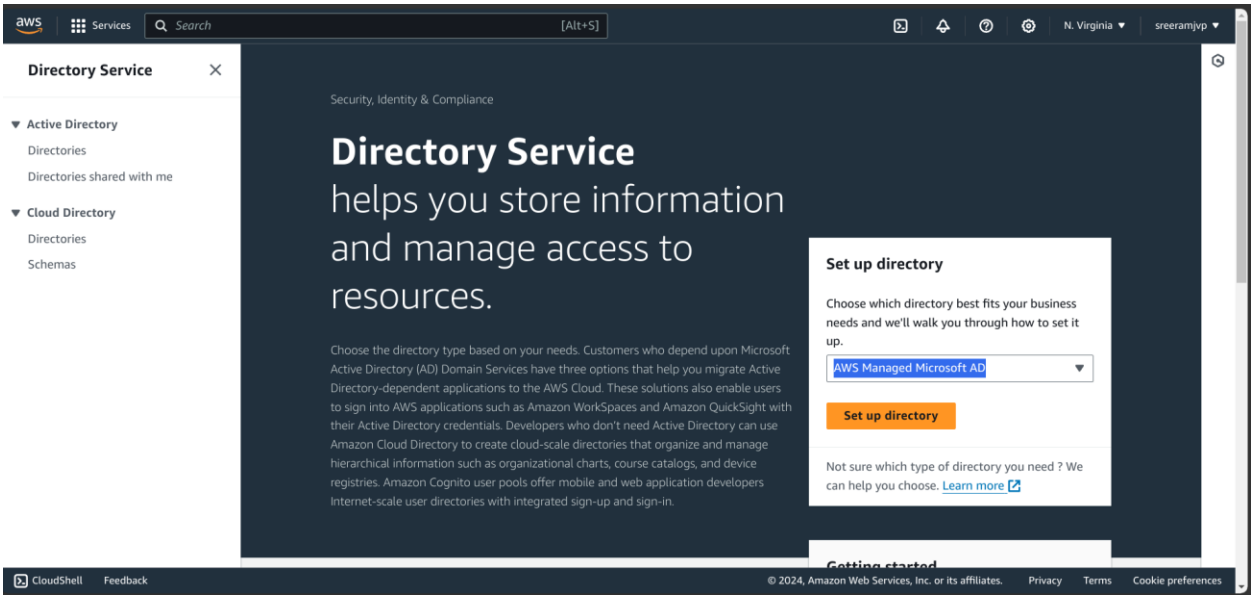
Stage:1



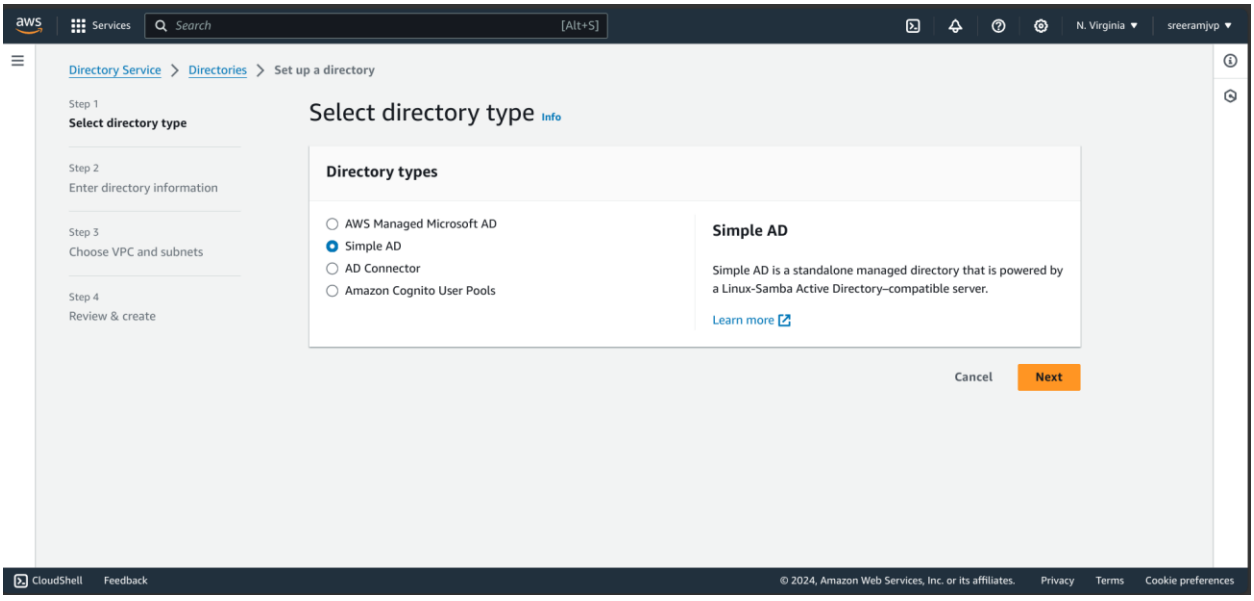
We will be using AWS directory service in simple AD mode. AWS Directory Service in Simple AD mode refers to a specific configuration of AWS Directory Service that provides basic Active Directory (AD) features without requiring a full Microsoft AD setup. It is designed to be simple to set up and manage, making it suitable for smaller organizations or less complex use cases.

(Step-6)

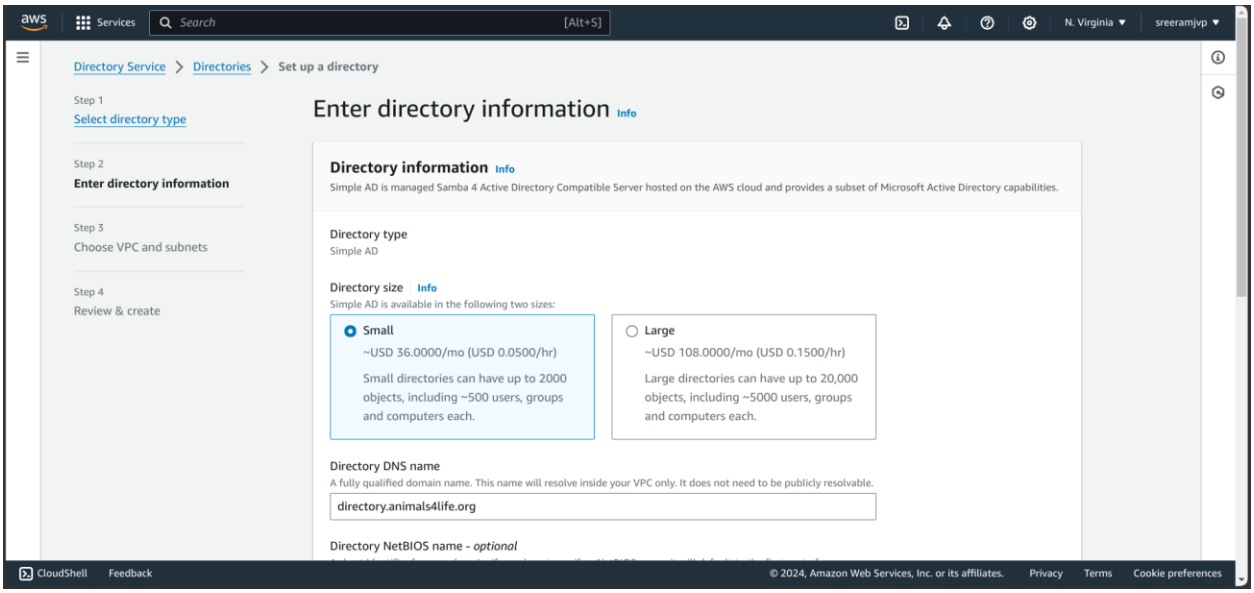
Choose AWS Directory service.

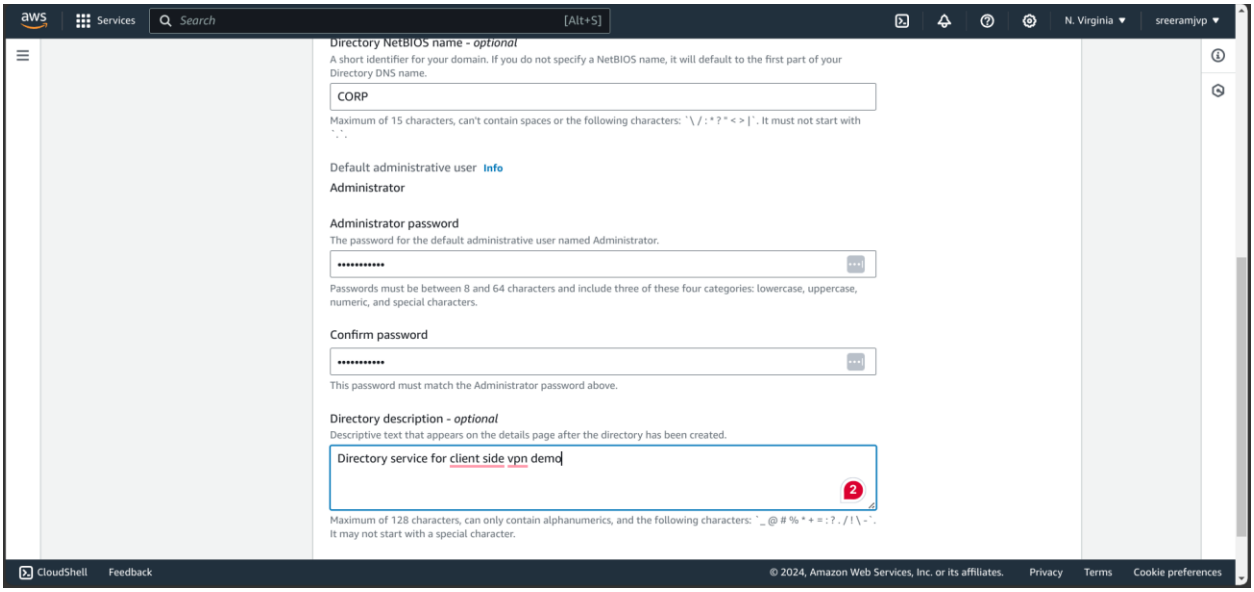


Select simple AD in the drop down menu:

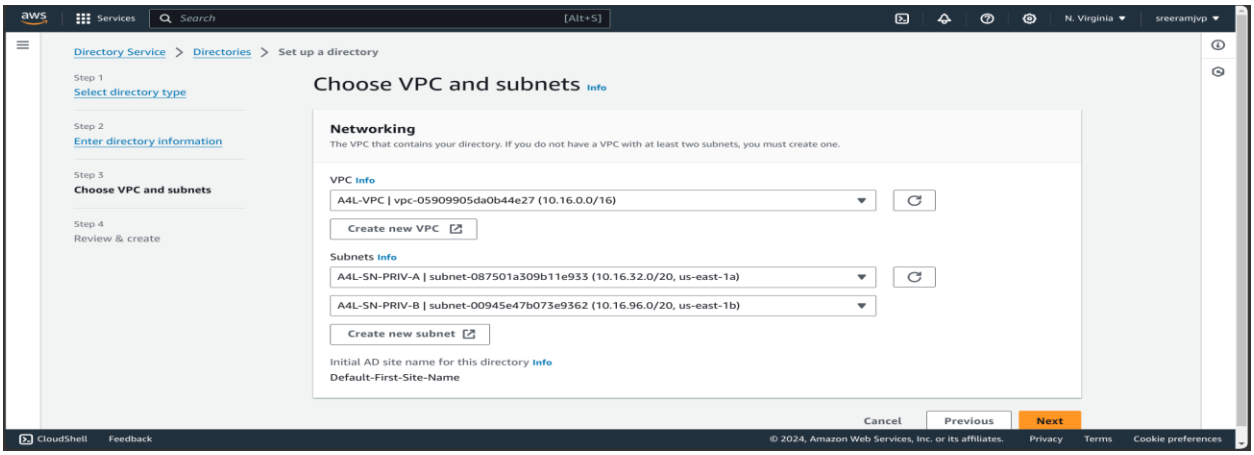


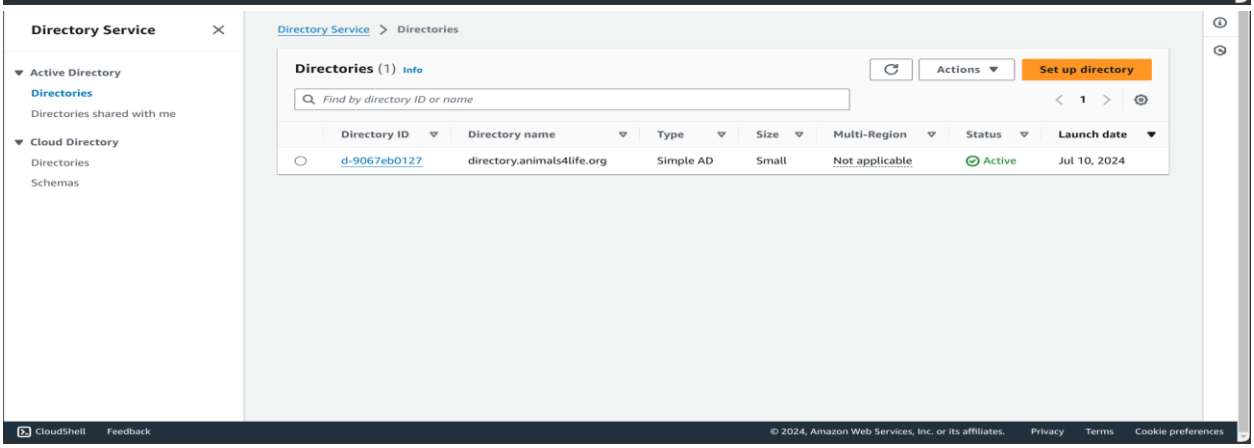
Directory information:





Choosing vpc and subnets:





Creating a Server Certificate for Client VPN

Prerequisites:

- Ensure the directory is in the active state.

Authentication Method:

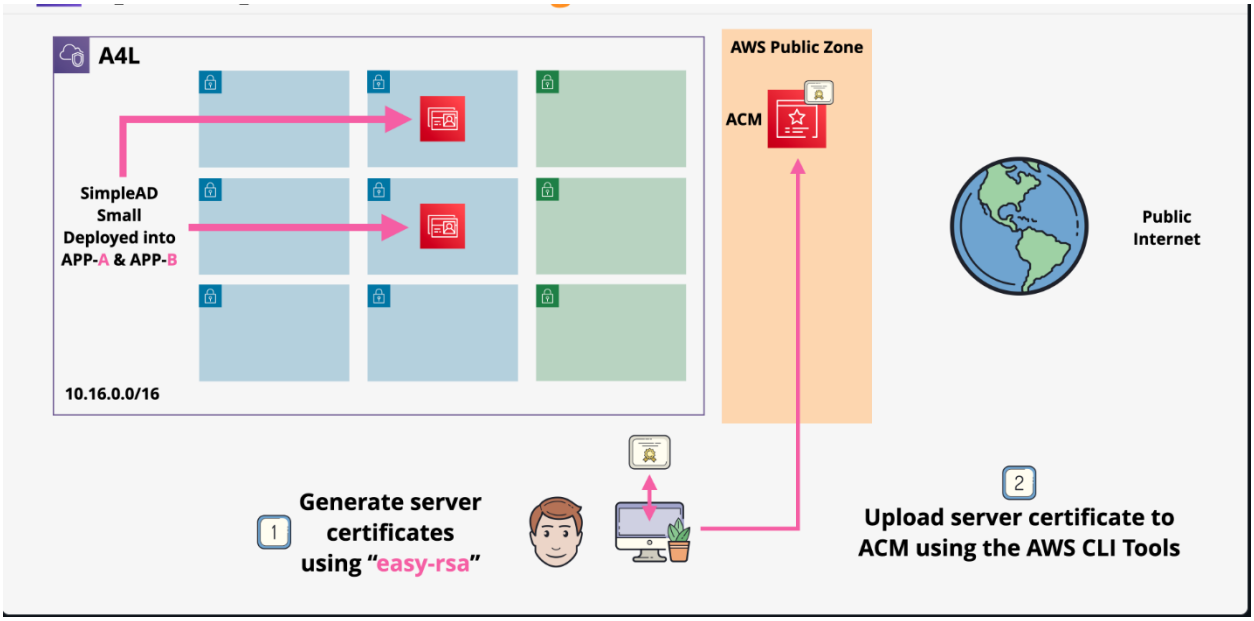
There are two ways we can implement authentication.

. Using Certificate based authentication.

- Using Identity-based Authentication.

Our choice method in the implementation is *Identity-based Authentication*.

We are at this level:



(Step-7):

" *A service certificate is generated IB Access Management*"

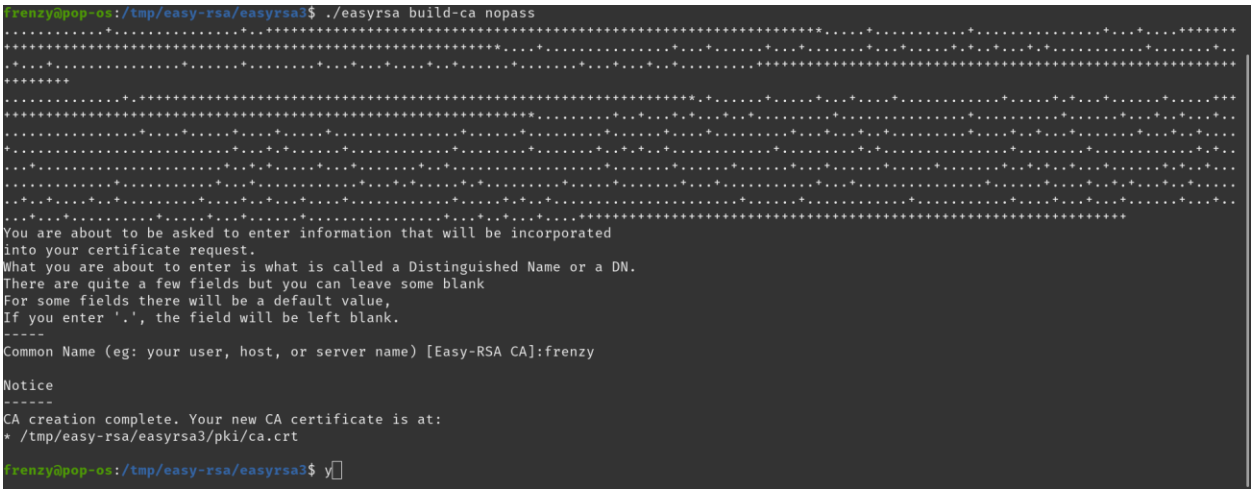
Moving to local machine.....

OS used: Linux(debian)>

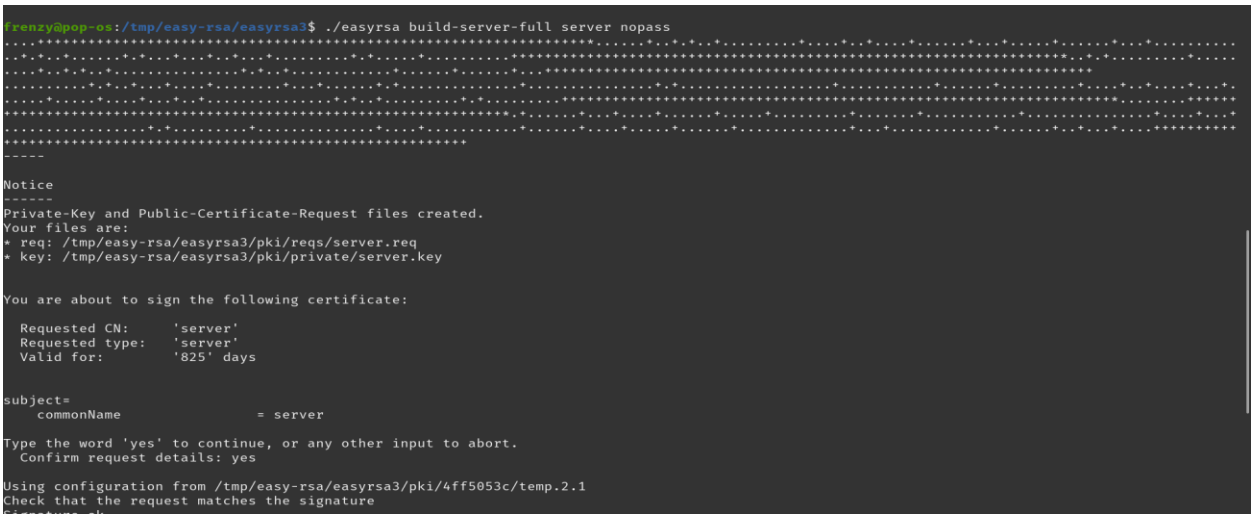
Steps:

```
frenzy@pop-os:~$ cd /tmp
frenzy@pop-os:/tmp$ git clone https://github.com/OpenVPN/easy-rsa.git
Cloning into 'easy-rsa'...
remote: Enumerating objects: 7095, done.
remote: Counting objects: 100% (1871/1871), done.
remote: Compressing objects: 100% (829/829), done.
remote: Total 7095 (delta 1088), reused 1439 (delta 1033), pack-reused 5224
Receiving objects: 100% (7095/7095), 52.45 MiB | 878.00 KiB/s, done.
Resolving deltas: 100% (3337/3337), done.
```

```
frenzy@pop-os:/tmp$ cd easy-rsa/easyrsaa3
bash: cd: easy-rsa/easyrsaa3: No such file or directory
frenzy@pop-os:/tmp$ cd easy-rsa/easyrsa3
frenzy@pop-os:/tmp/easy-rsa/easyrsa3$ ./easyrsa init-pki

Notice
------
'init-pki' complete; you may now create a CA or requests.

Your newly created PKI dir is:
* /tmp/easy-rsa/easyrsa3/pki


Using Easy-RSA configuration:
* undefined
```



Server buiding:



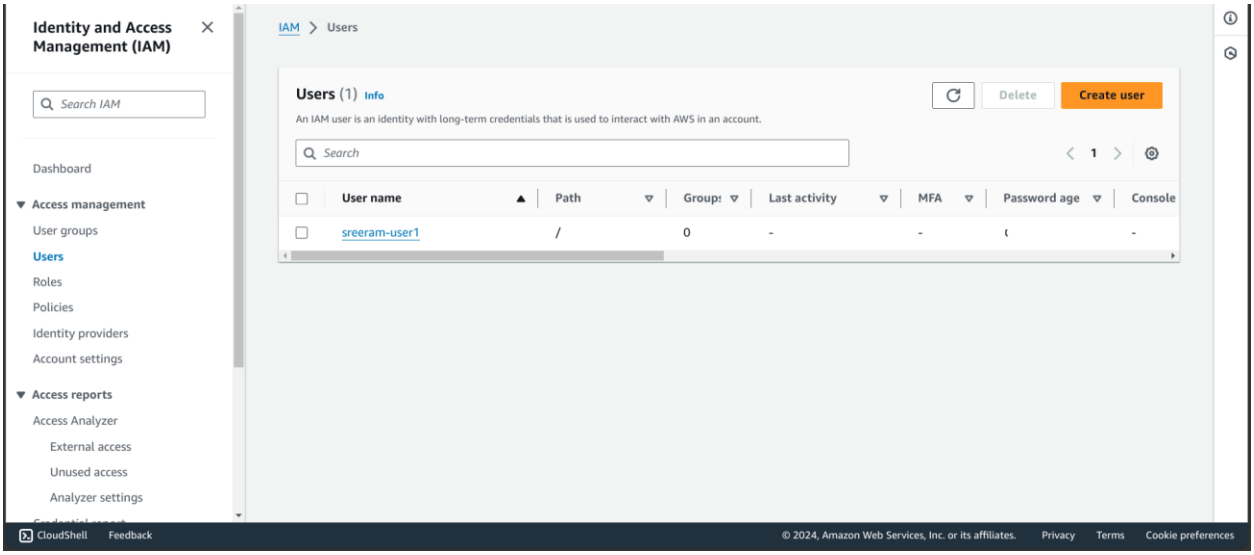
Name: *“server”*

Server certificate succefully completed….

```
You are about to sign the following certificate:

  Requested CN:     'server'
  Requested type:   'server'
  Valid for:        '825' days


subject=
    commonName                = server

Type the word 'yes' to continue, or any other input to abort.
  Confirm request details: yes

Using configuration from /tmp/easy-rsa/easyrsa3/pki/4ff5053c/temp.2.1
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'server'
Certificate is to be certified until Oct 13 13:09:16 2026 GMT (825 days)

Write out database with 1 new entries
Data Base Updated

Notice
------
Certificate created at:
* /tmp/easy-rsa/easyrsa3/pki/issued/server.crt


Notice
------
Inline file created:
* /tmp/easy-rsa/easyrsa3/pki/inline/server.inline

frenzy@pop-os:/tmp/easy-rsa/easyrsa3$
```

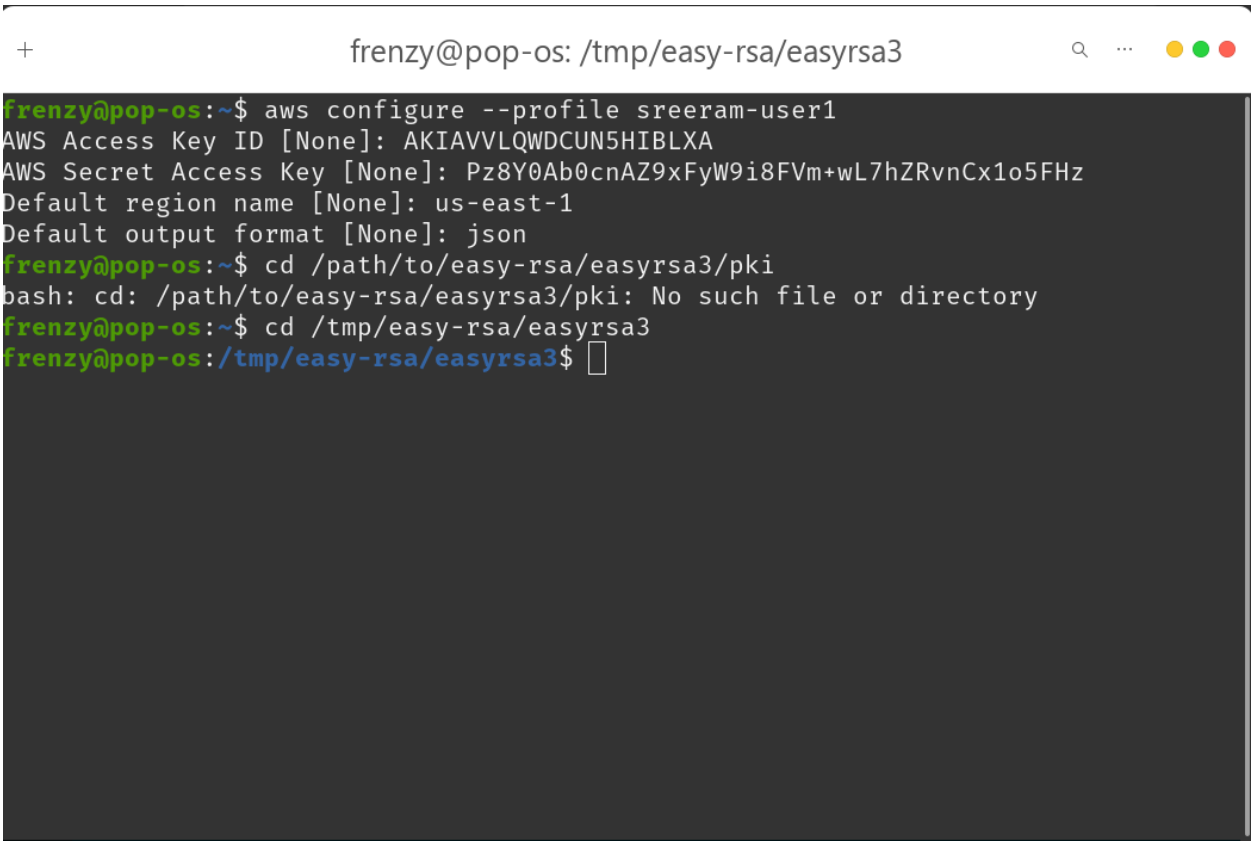
“Pki folder contains all the required certificates. All the contents in easyrsa folder are to uploaded to ACM, so that client vpn could authenticate and access.”

#Attaching a user profile to ACM

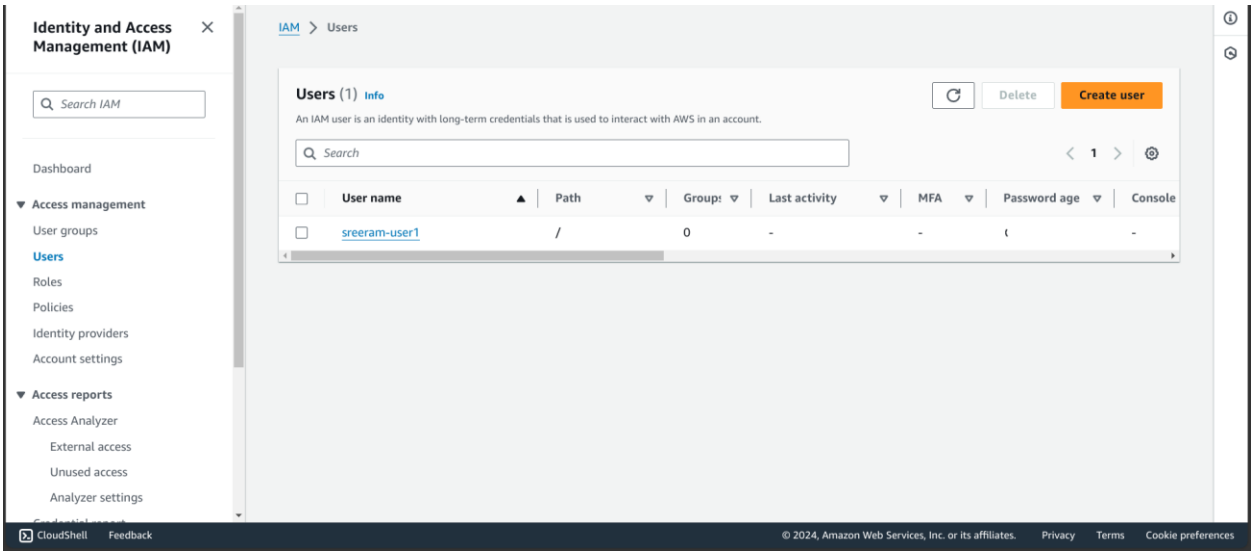


Created a user-profile using aws console, with IAM.

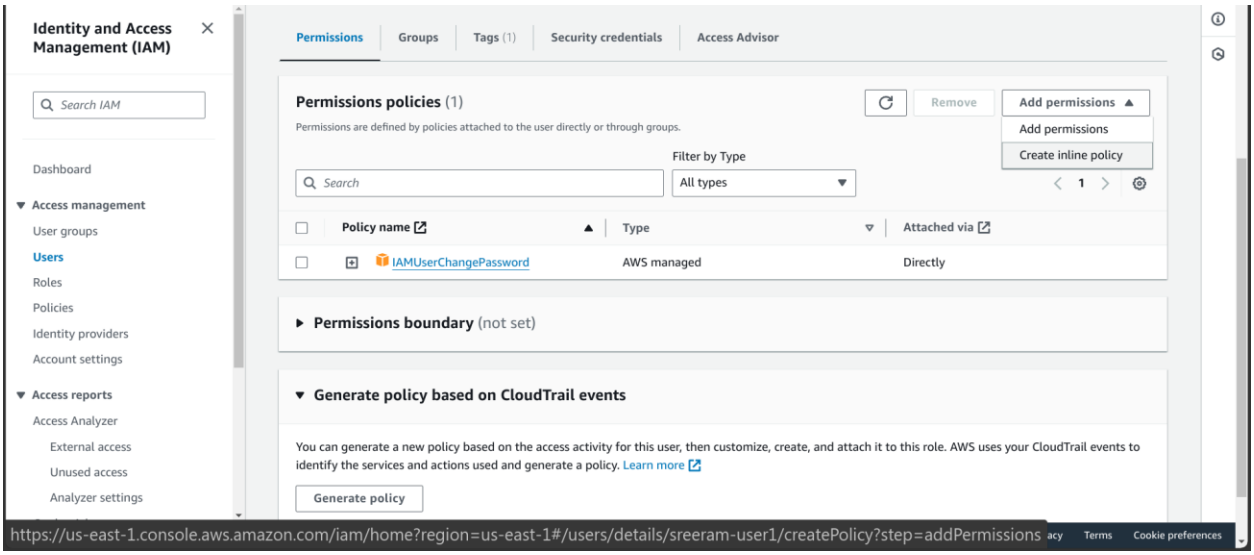
. Accessing user-profile ‘sreeram-user1’ using aws acm profile.



Adding custom policy to allow write ACM server certificates upload:

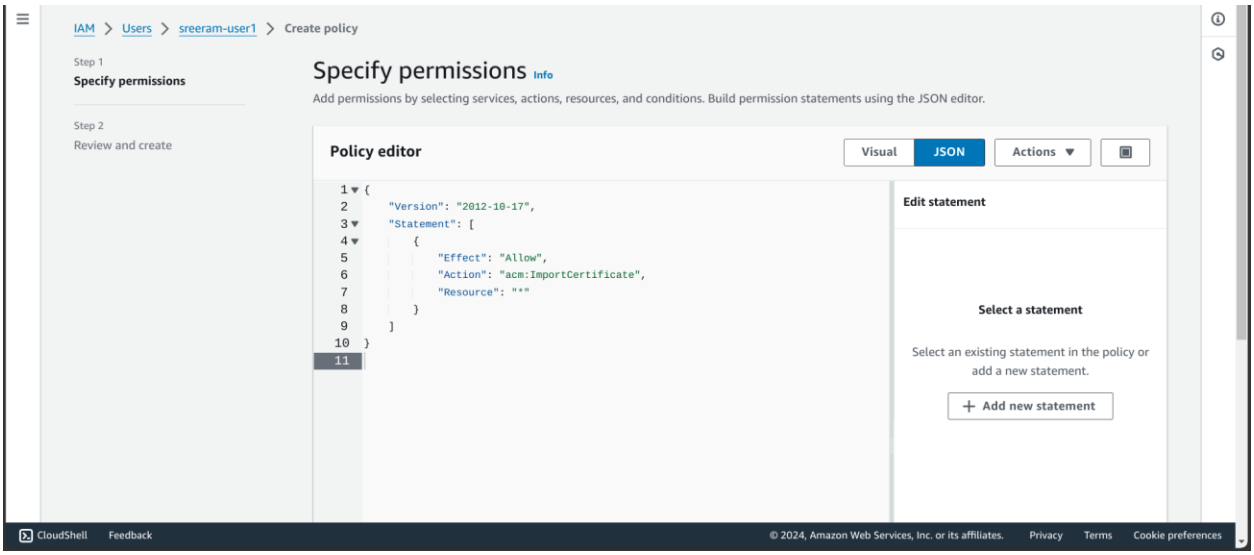


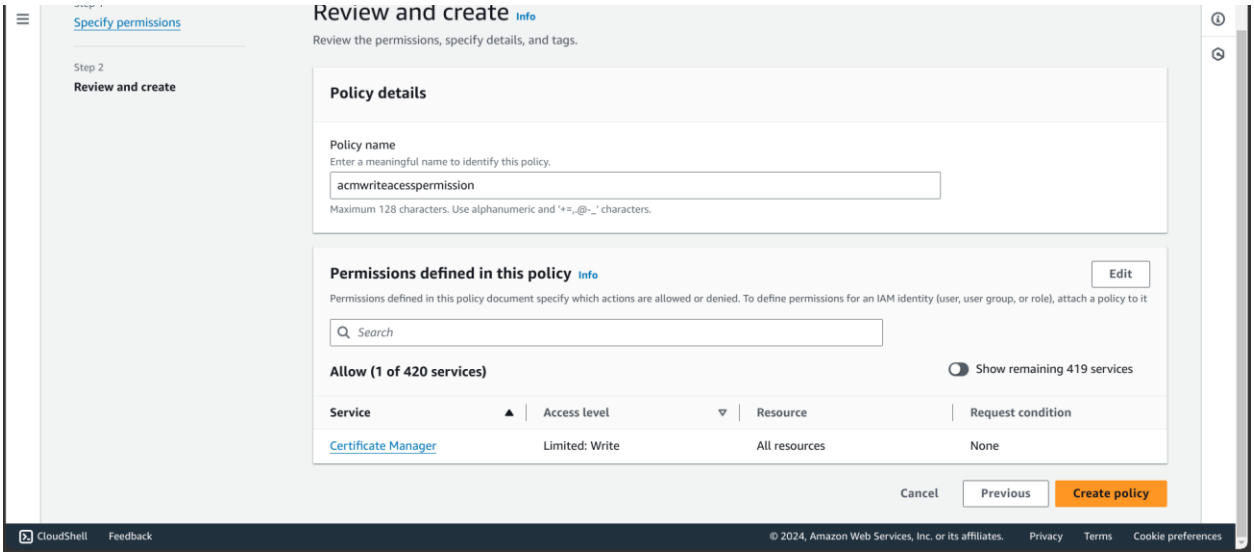
#Creating a new inline policy:



Adding policy in form of json, which would only allow AWS CLI to access and write into the simple AD directory service using ACM.

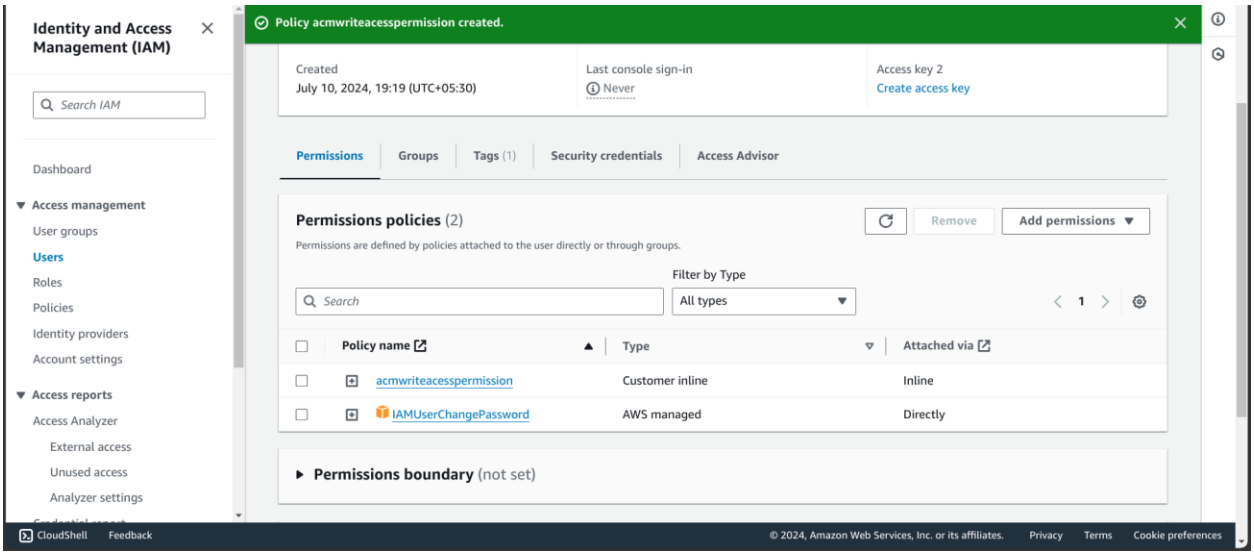
 # “Sreeram-user1” is only allowed to accessed through AWS CLI, for which the access keys and secrey access keys have been generated....





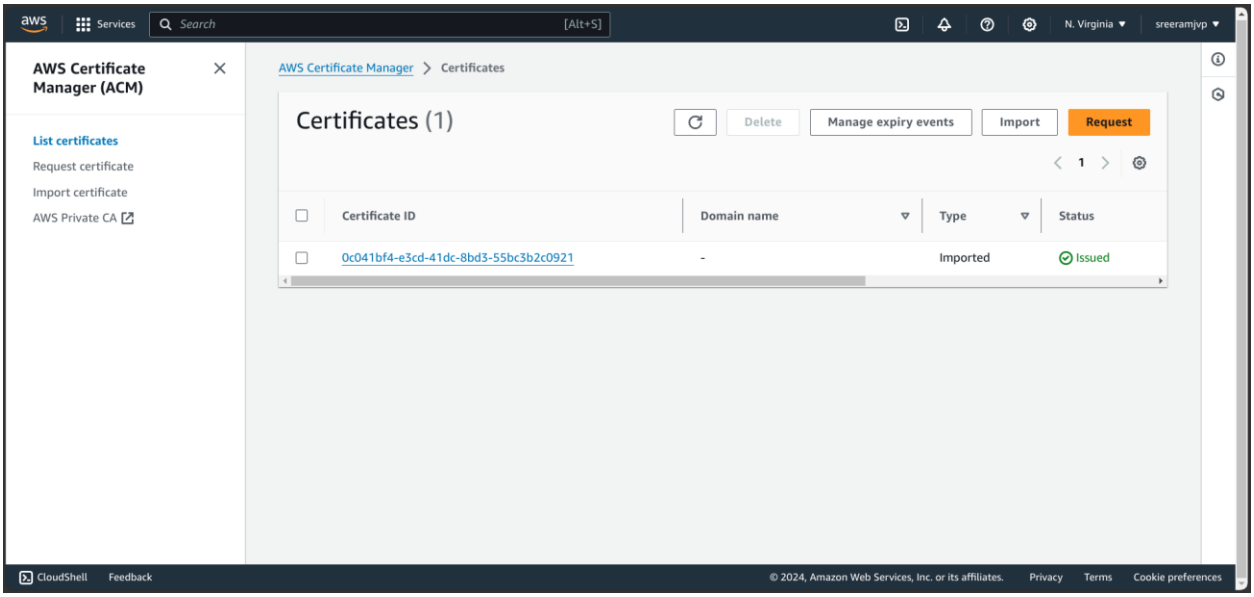
.....

Policy has been successfully created.....



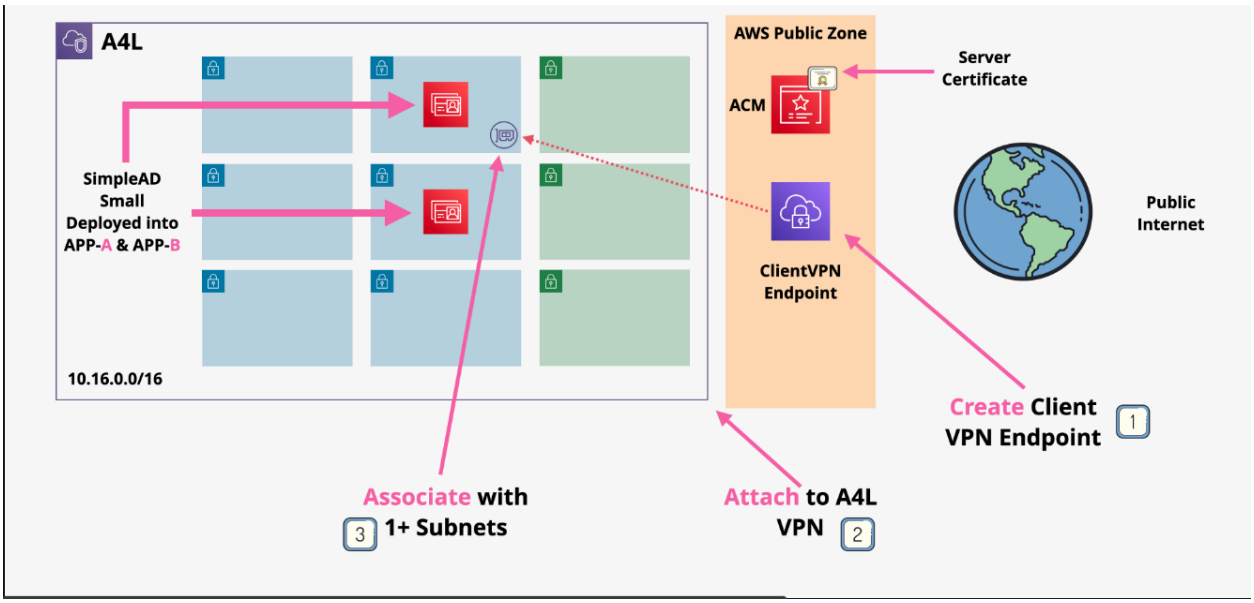
Uploaded the server certificates to "sreeram-user1" profile using AWS ACM successfully...

```
frenzy@pop-os:/tmp/easy-rsa/easyrsa3$ aws acm import-certificate --certificate fileb://pki/issued/server.crt --private-key fileb://pki/private/ser
ver.key --certificate-chain fileb://pki/ca.crt --profile sreeram-user1
{
    "CertificateArn": "arn:aws:acm:us-east-1:389467740328:certificate/0c041bf4-e3cd-41dc-8bd3-55bc3b2c0921"
}
frenzy@pop-os:/tmp/easy-rsa/easyrsa3$
```



(Step-8):

We are here.....

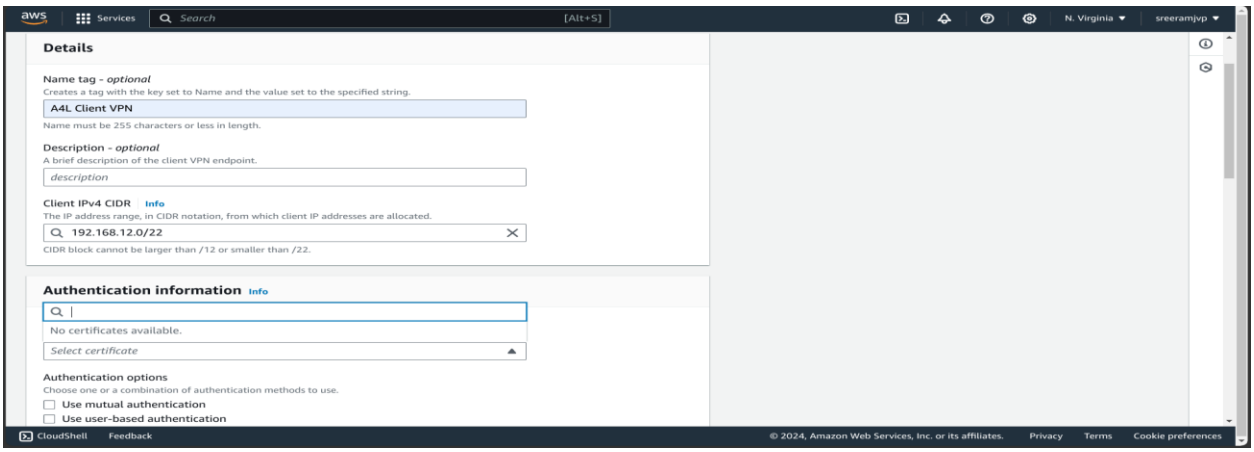


This stage involves creating a vpn end point and association using the VPC service within the aws console:

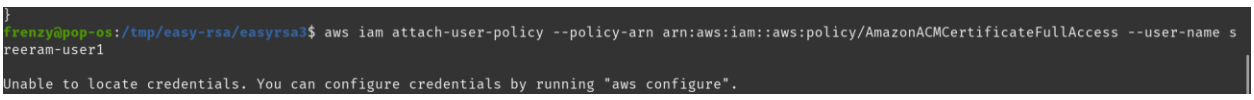
1. *Choose vpc service*
2. *Select client VPN endpoint*

Challenges faced:

As specified through the inline policy , acmwriteaccess is alone not enough for locating the server ARN certificates.



Unable to update or add inline policy using aws cli:



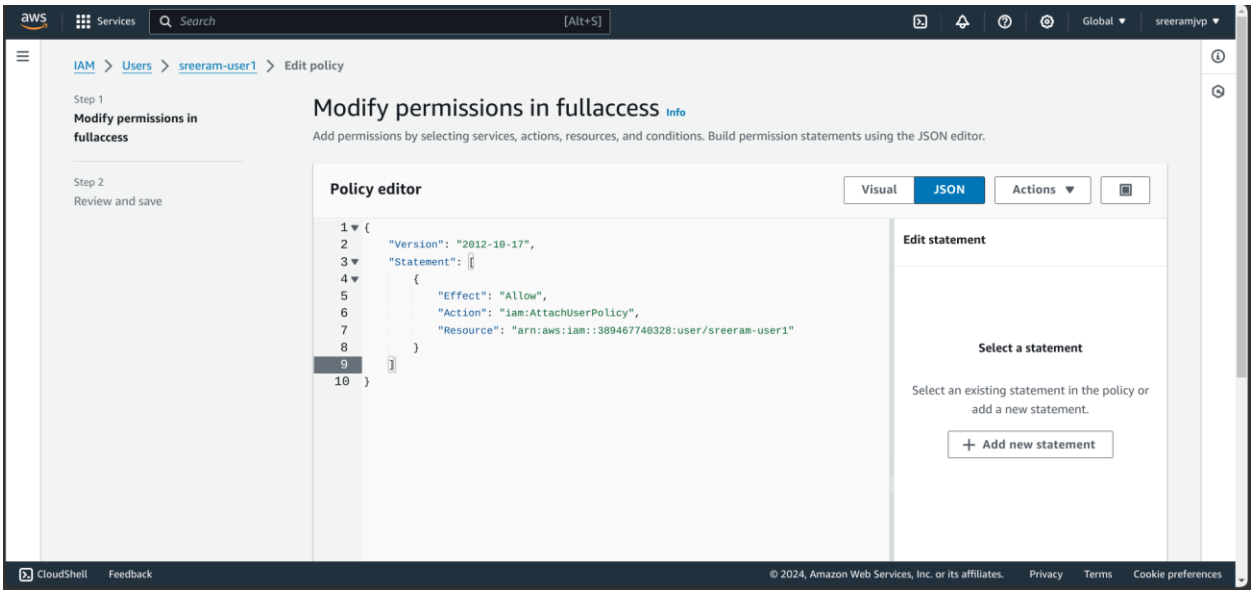
Solution;

Used ‘aws configure’ to authenticate request from my local machine.

Lack of policy permissions for sreeram-user1 to manifest policy in Identity access management:

```
frenzy@pop-os:/tmp/easy-rsa/easyrsa3$ aws iam attach-user-policy --policy-arn arn:aws:iam::aws:policy/IAMFullAccess --user-name sreeram-user1

An error occurred (AccessDenied) when calling the AttachUserPolicy operation: User: arn:aws:iam::389467740328:user/sreeram-user1 is not authorized
 to perform: iam:AttachUserPolicy on resource: user sreeram-user1 because no identity-based policy allows the iam:AttachUserPolicy action
```

To solve this, I have added an inline policy that allows attaching policy to user-1



On adding this policy, we are now able to attach AWS managed policy to allow fullaccess to the acm service for “sreeram-user1”:

Succes....

```
frenzy@pop-os:/tmp/easy-rsa/easyrsa3$ aws iam attach-user-policy --policy-arn arn:aws:iam::aws:policy/IAMFullAccess --user-name sreeram-user1

Unknown output type: JSON
frenzy@pop-os:/tmp/easy-rsa/easyrsa3$
```

To solve this, we need to add a new inline policy that allows acm discoverability(i.e. ALLOWFULLACCESS), to ensure its visible to the VPC service.

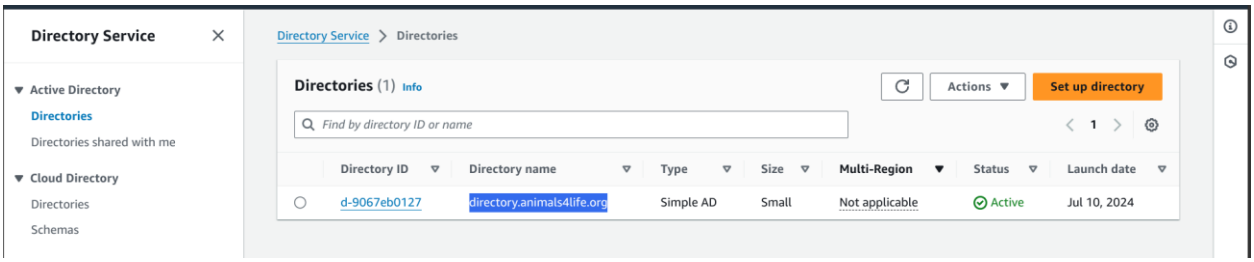
Now lets configure the cliend VPN end points...

But no so fast...

We still cant get out vpc to recognise the ACM certificate:

On further investigation, with reference to https://repost.aws/questions/QUbHwO-HGfTcWCrSc5fBZmKw/server-certificate-not-showing-for-vpn-endpoint, the main error stems from not including domain name in the server build command in easyrsa.

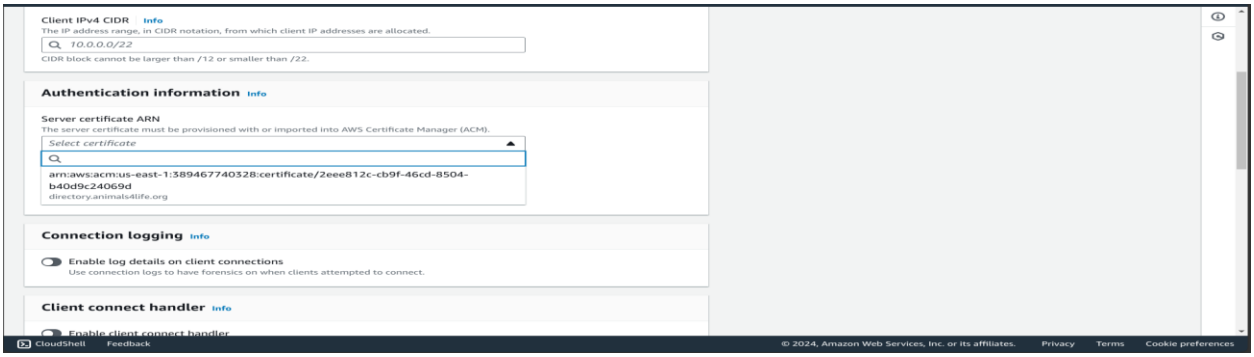
Our domain name, that we have created in simple AD, using Directory service, is
"directory.animals4life.org"



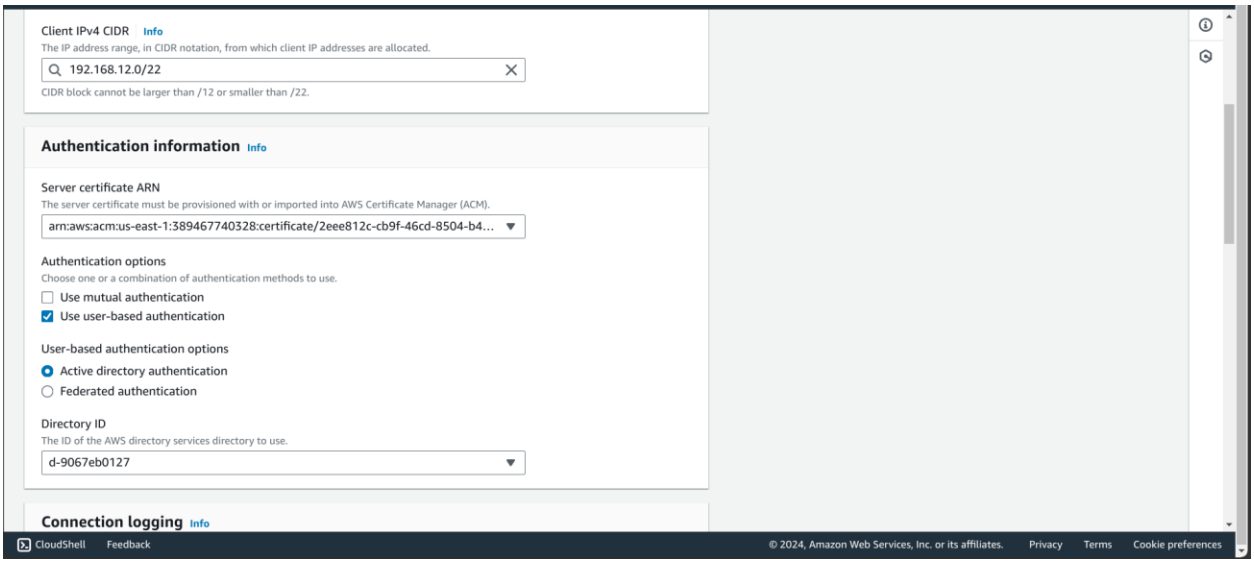
On repeating the steps from creating sever certificate on our local machine, we have modified the command to :

"frenzy@pop-os:/tmp/easy-rsa/easyrsa3$ ./easyrsa build-server-full directory.animals4life.org nopass"

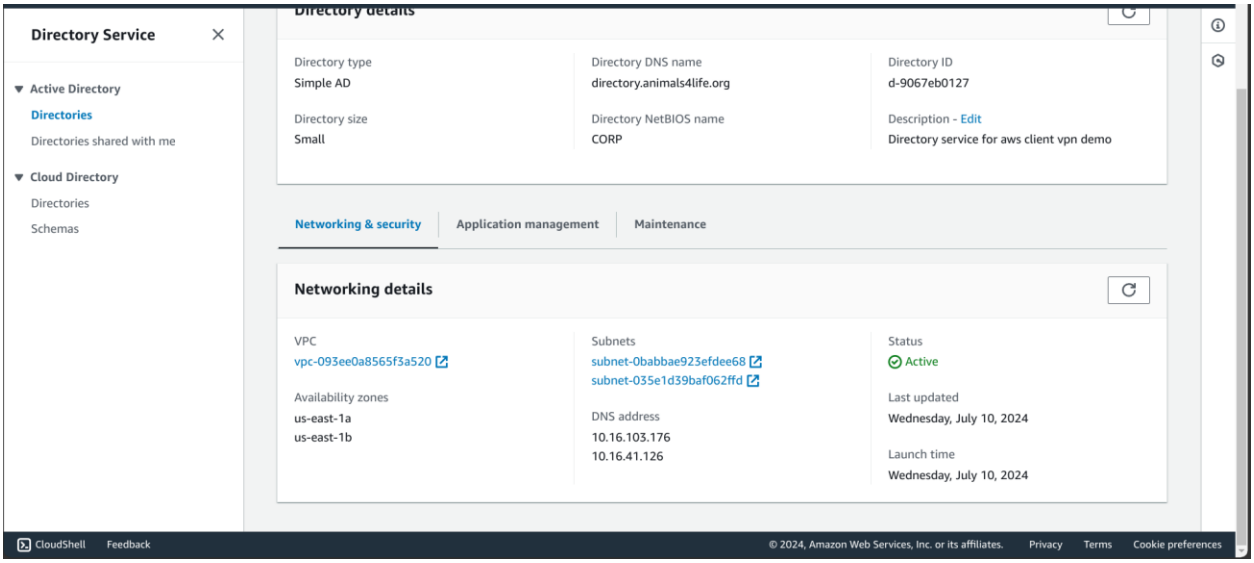
SUCCESS!



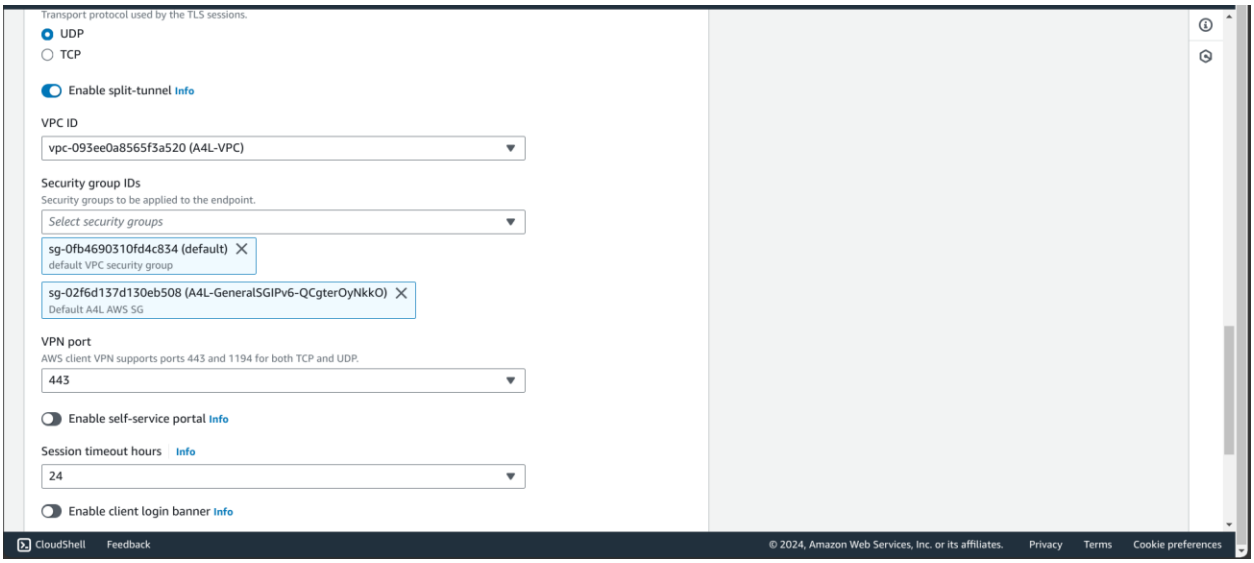
Lets move to client vpn endpoint creation!



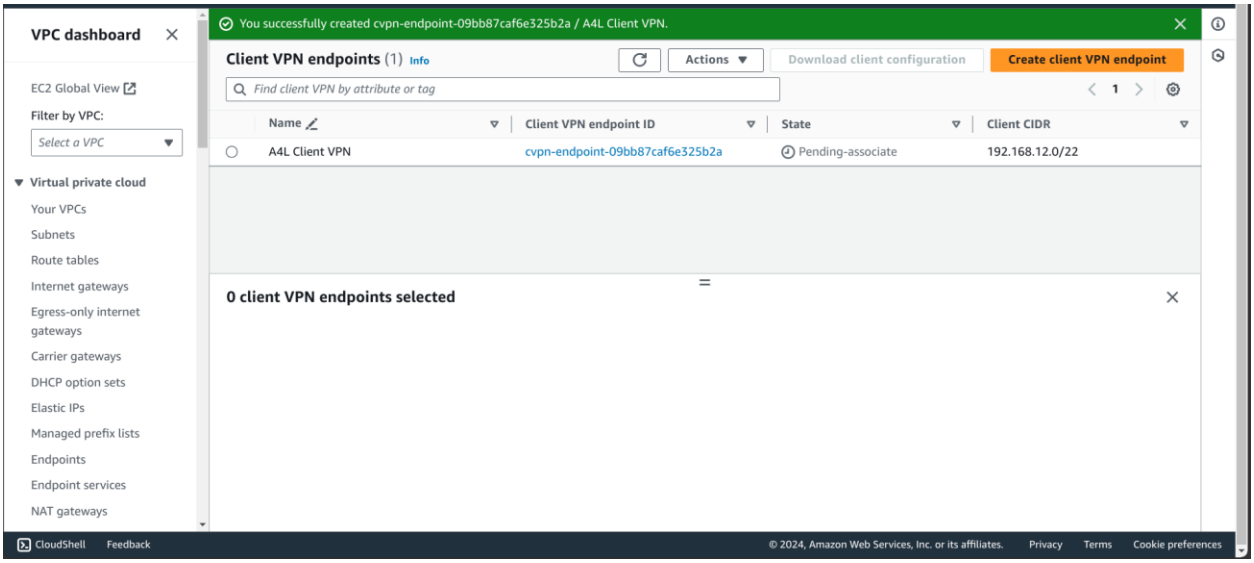
For DNS server , it is essential when a user needs to resolve any connectivity issues.
We will use simple AD service for dns addressing:



The dns service is provided to our client end vpn within the vpc by the ad service itself!
We would also want our user to access the internet resources without the vpn . The vpn is configured in such a way that, it allows the connection to attach itself only to the resources that lie inside within the VPC.
To achieve that functionality, *split tunnel* is enabled.



SUCCESS!!



The state remains pending association as it still needs to be attached with a subnet within a VPC.

Client VPN Endpoint State

- **Pending-associate**: This state indicates that the Client VPN endpoint is waiting to be associated with a subnet in your VPC (Virtual Private Cloud).

Association with a Subnet

- **Subnet Association**: You need to link the Client VPN endpoint with a subnet in your VPC. This subnet will host the network interfaces for the Client VPN.

Network Interfaces

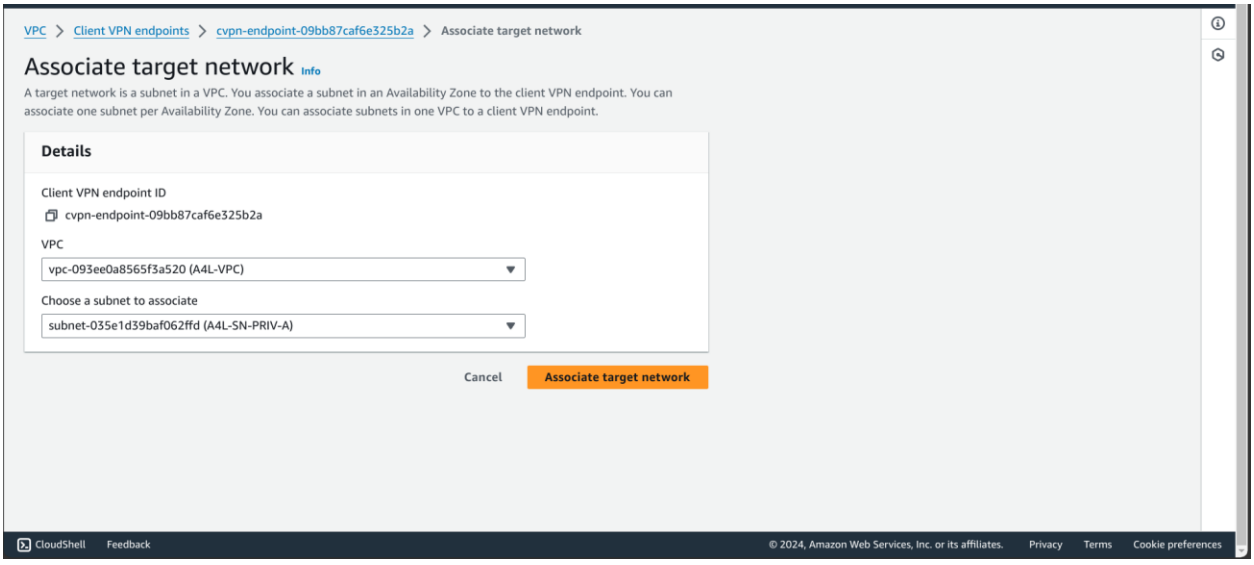
- **Client VPN Interfaces**: These are virtual network interfaces created in the associated subnet. They facilitate communication between the VPN clients and the VPC.

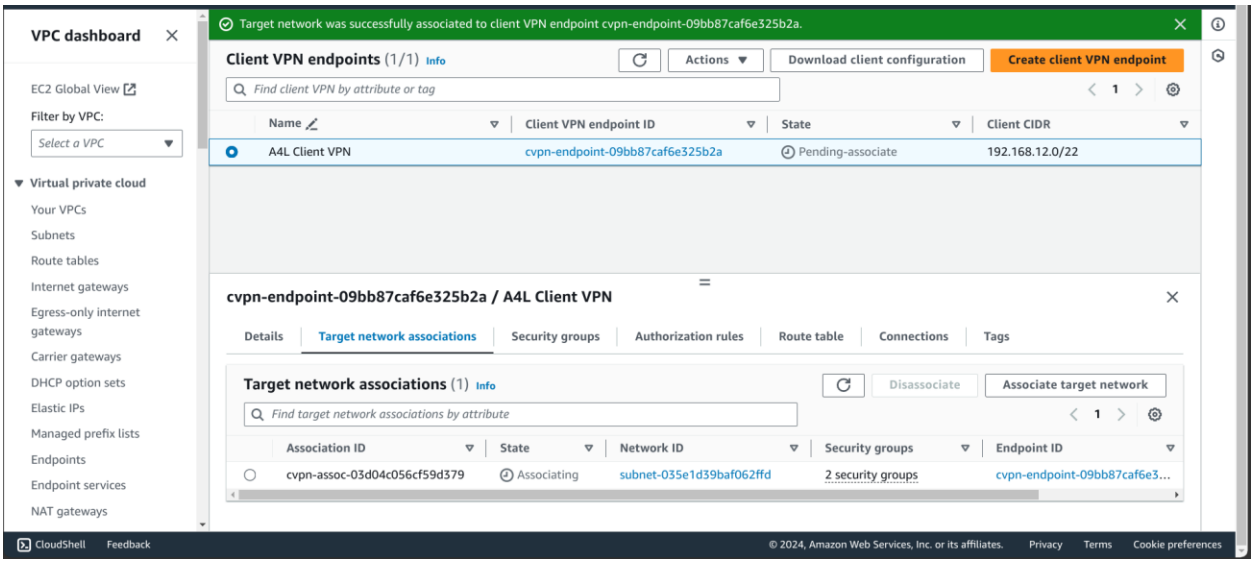
Traffic Flow

- **Traffic from Clients to VPC**: Traffic from the VPN clients will exit through these interfaces into the VPC.
- **Traffic from VPC to Clients**: Traffic destined for the VPN clients from within the VPC will enter through these interfaces.

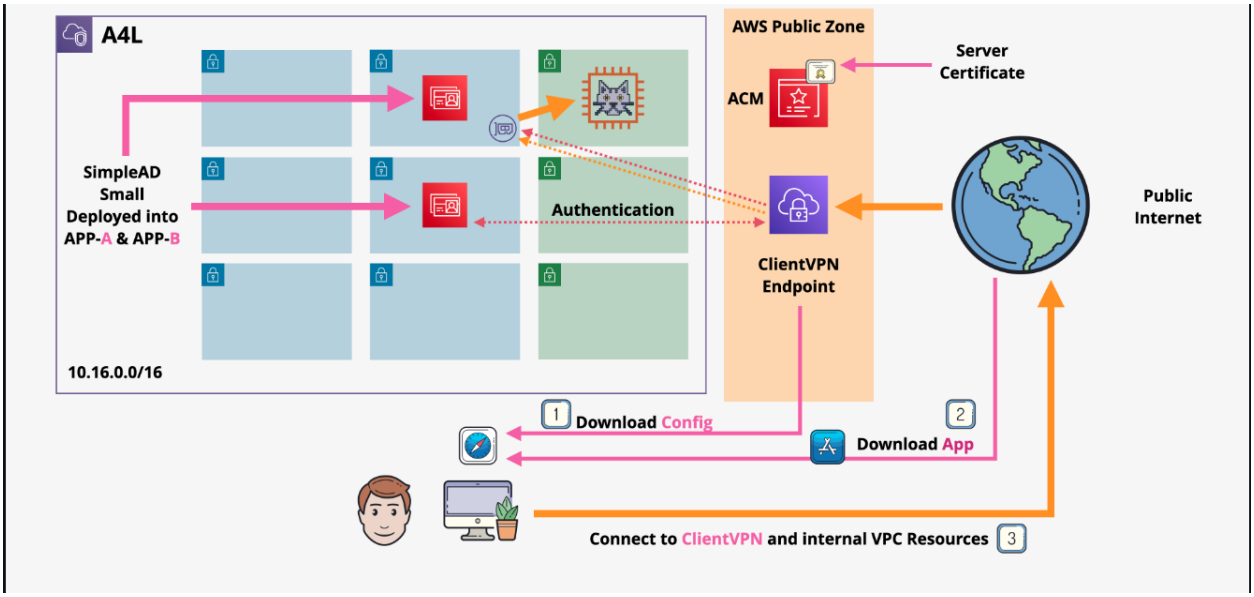
(Step-9)

Creating association...



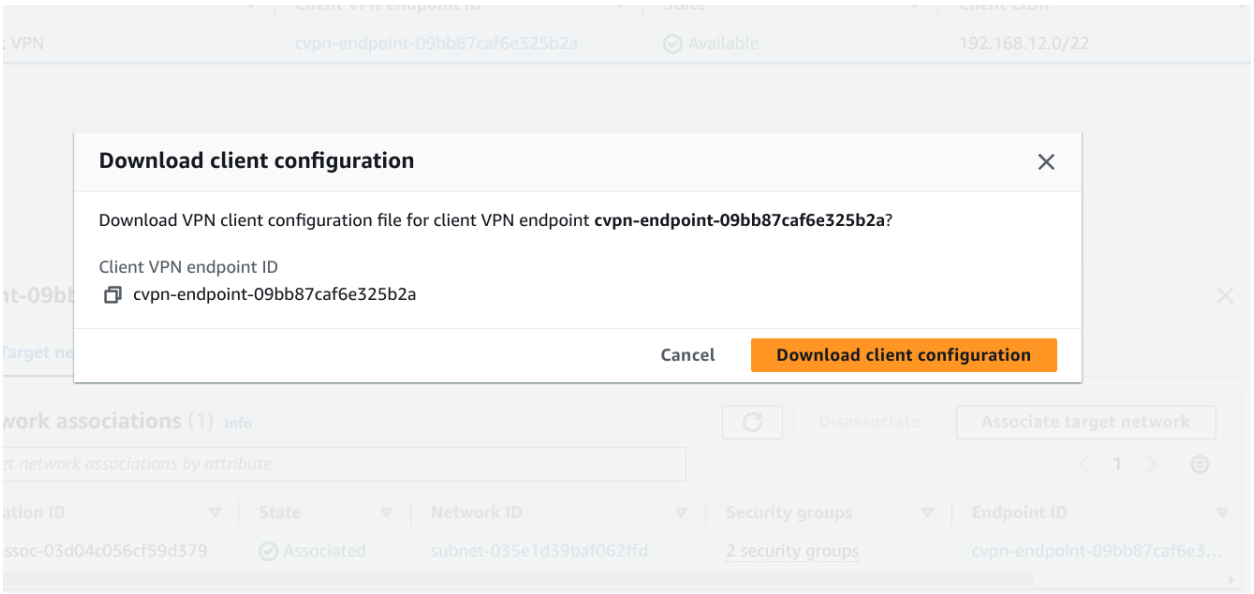


We are here....



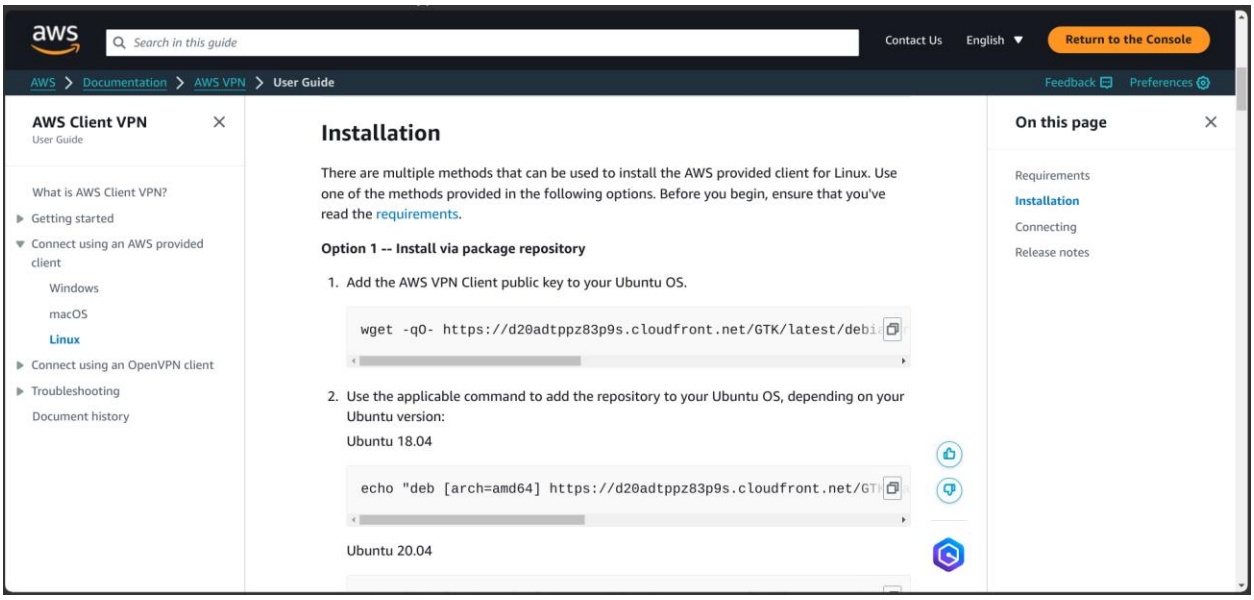
Wait for the target association status to change to “Associated”...

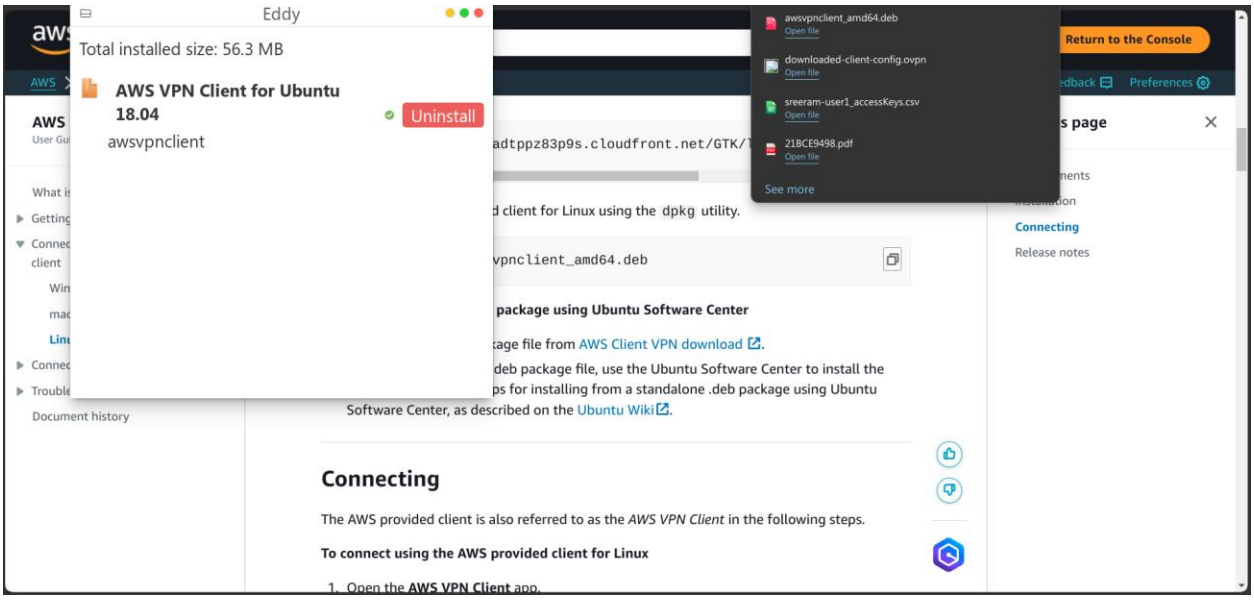
After that, download client configuration settings...



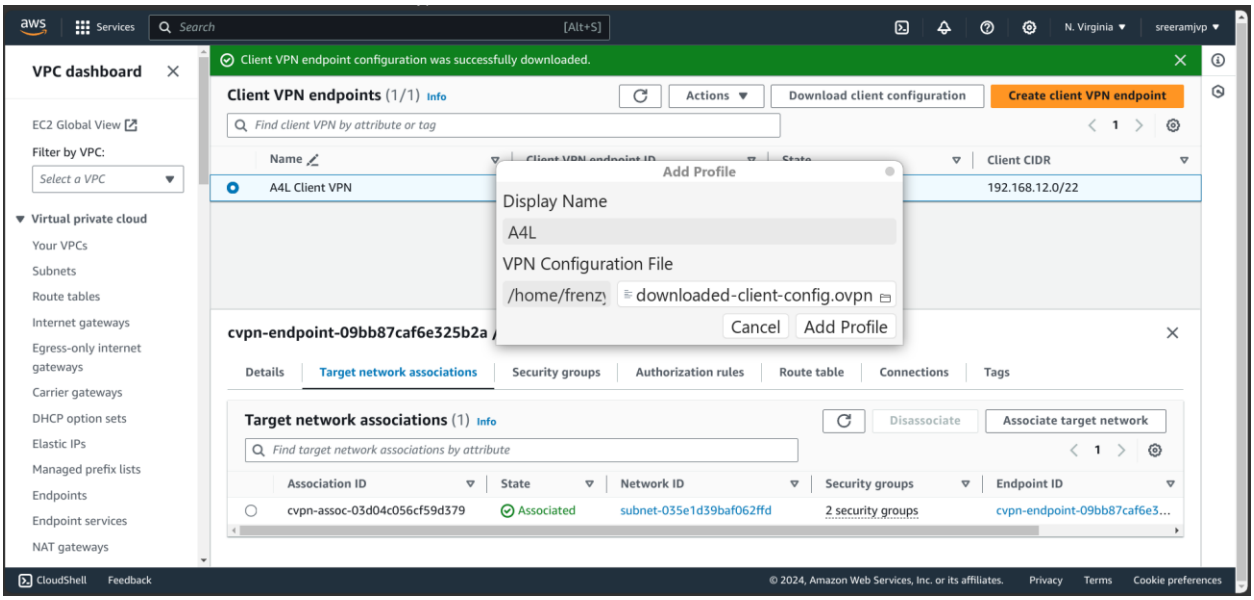
....

We are required to download the client vpn ...

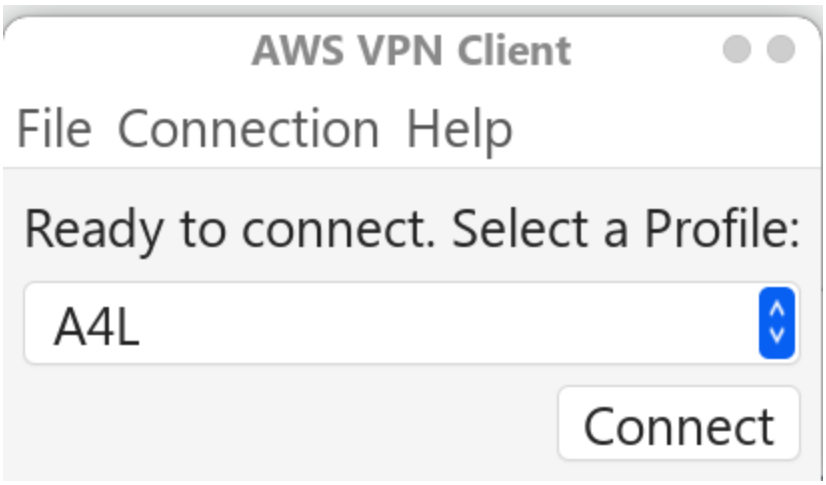




Associate openVPN profile to the AWS VPN CLIENT...



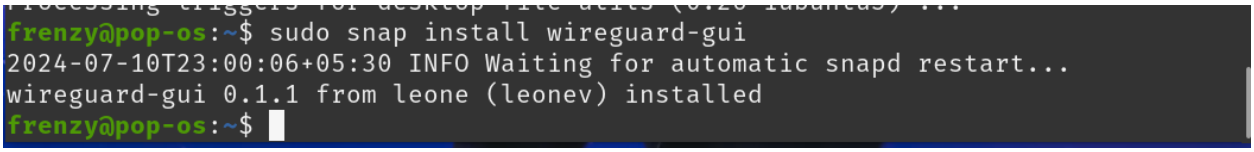
Click on connect...and enter the simple ad credentials that was created at the beginning..



Well....there seems to be another problem with the dependencies associated with the .deb file. On further digging, its been found that, AWS is no longer supporting AWS client vpn for Ubuntu version, and the client would only run on older versions of lib, which would be a no no due to security concerns.

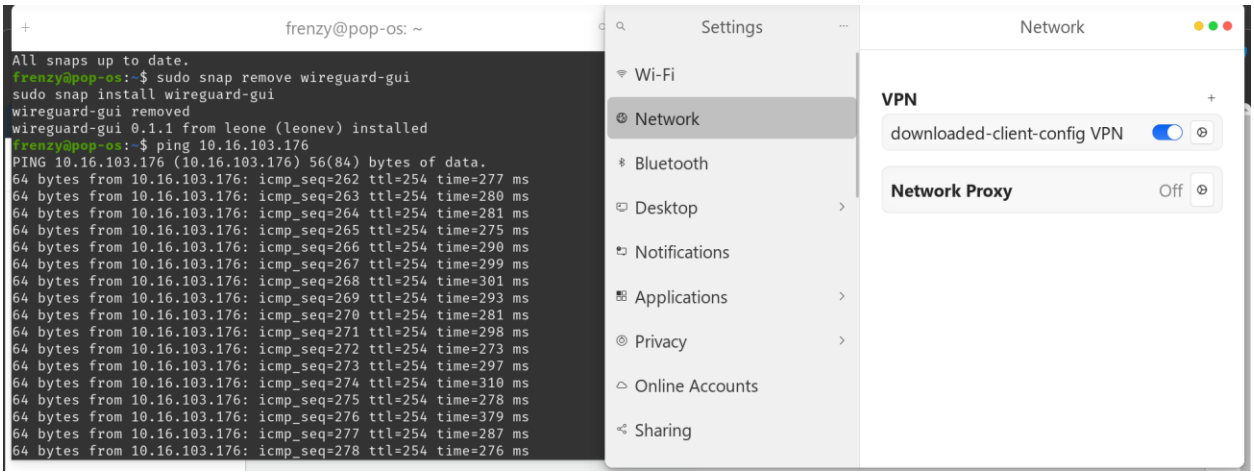
The alternative would be to use, wireguard with gui installed using snapd, which is a proprietary software packaging by *canonical softwares*.

Wireguard is similar to AWS client vpn, and is supported and updated. The repository is updated .

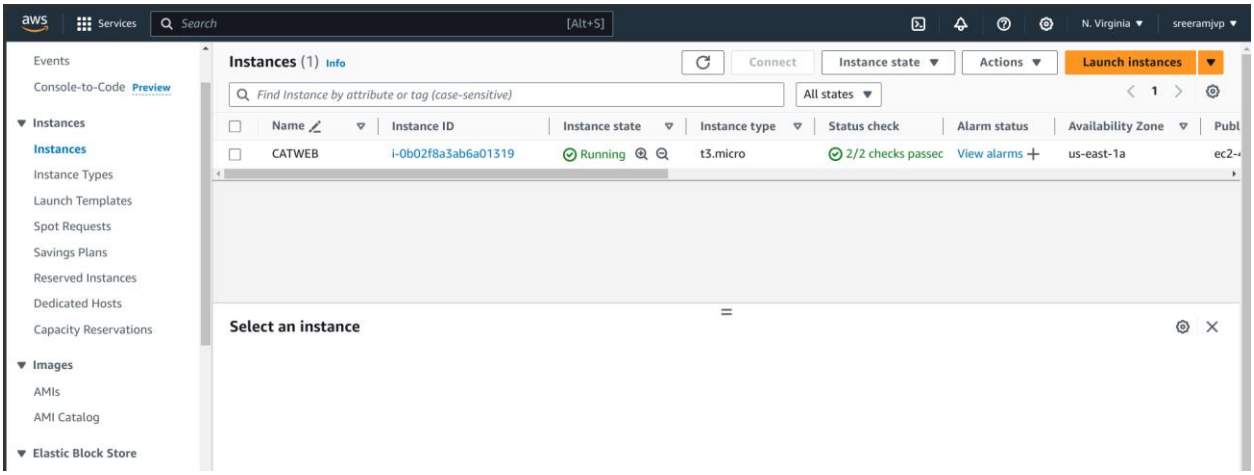


After adding in user-name and password , go to the vpn client settings and select authorization roles, and approve,

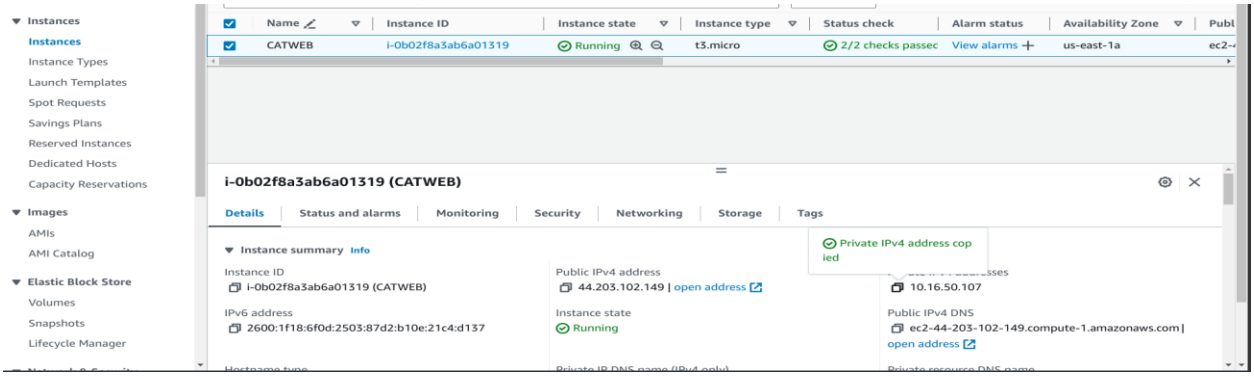
SUCCESS!!!



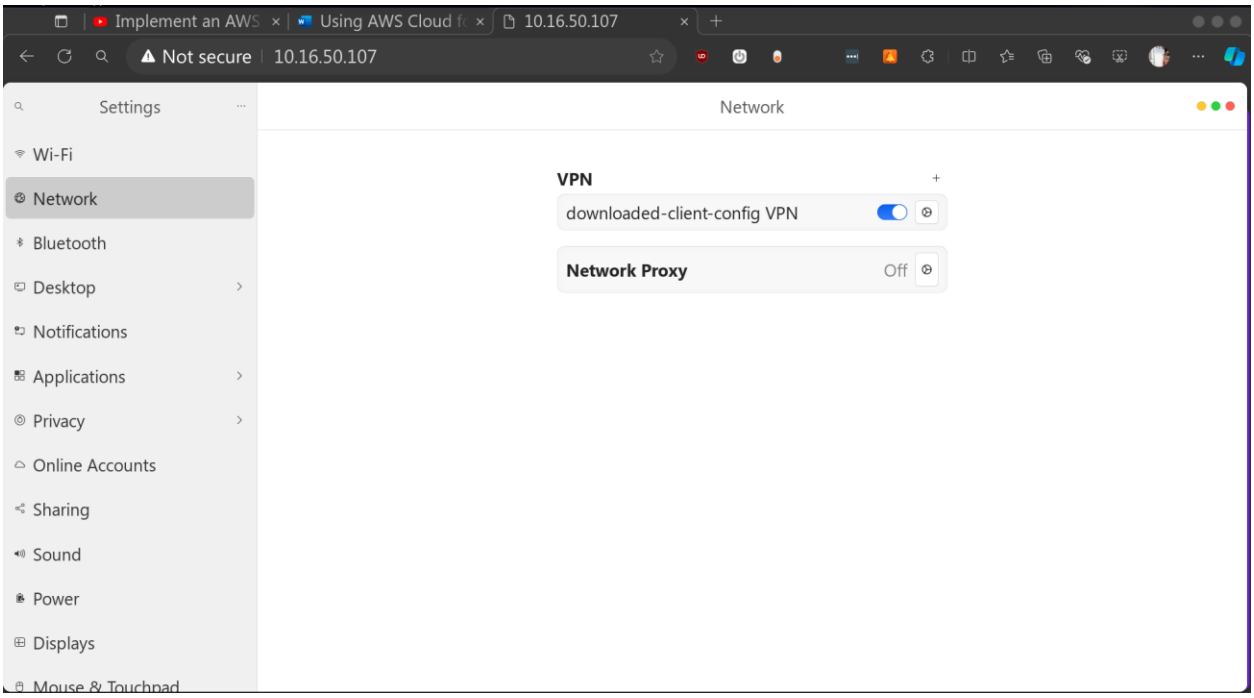
Another way to verify if our VPN is working is to select Ec2 instance:

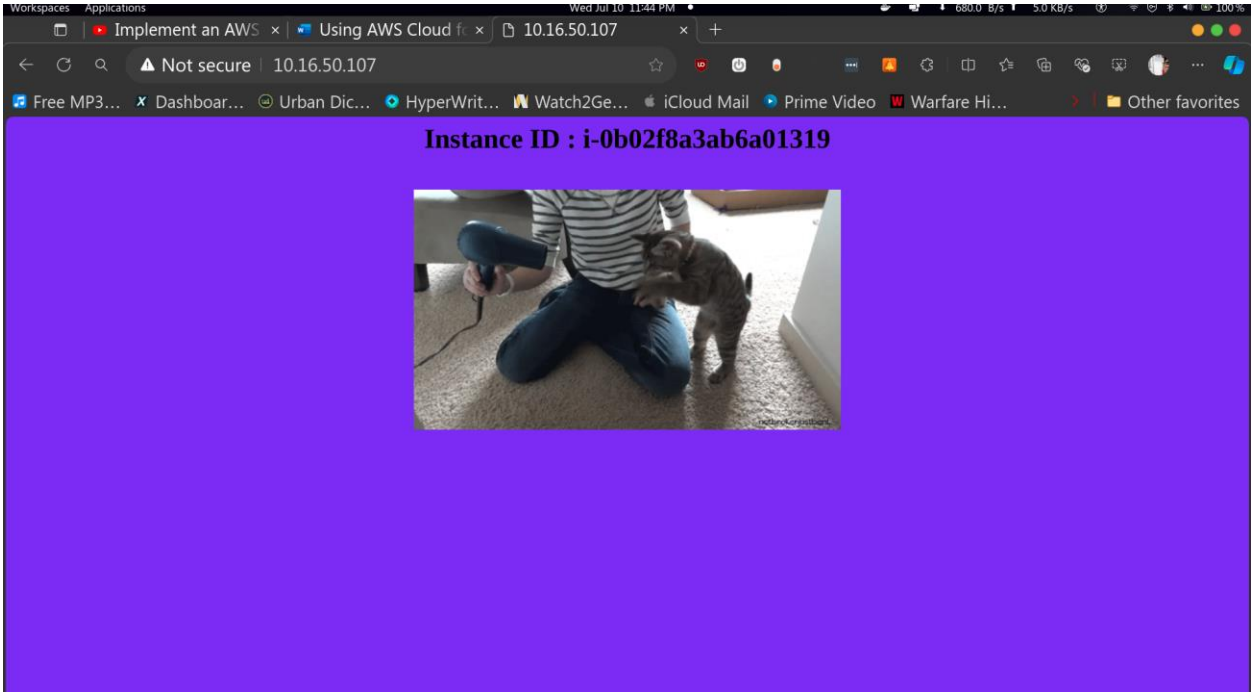


We can see there is a an ec2 instance running hosting a static webiste. Choose the ec2 instance and copy the private IPV4 instance ip address:

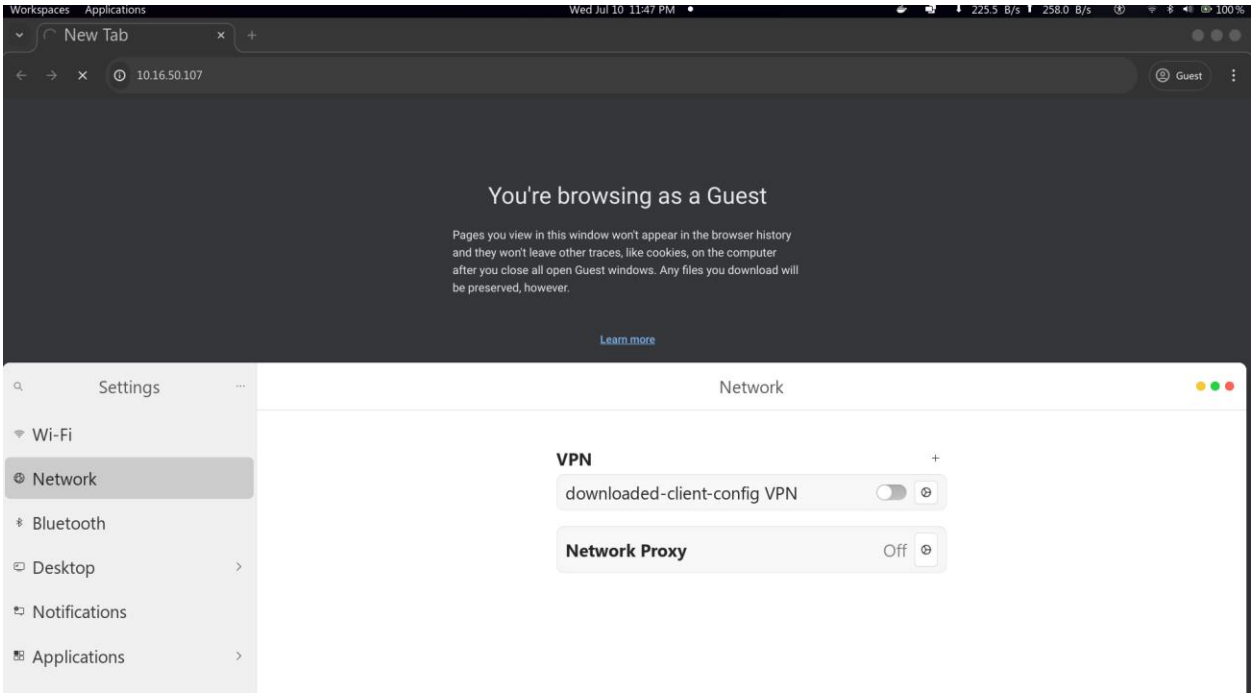


VPN ON:


Implement an AWS
Using AWS Cloud f
10.16.50.107
Not secure
10.16.50.107
Settings
Network
Wi-Fi
Network
Bluetooth
Desktop
Notifications
Applications
Privacy
Online Accounts
Sharing
Sound
Power
Displays
Mouse & Touchpad
VPN
downloaded-client-config VPN
Network Proxy
Off


Workspaces
Applications
Wed Jul 10 11:44 PM
Implement an AWS
Using AWS Cloud f
10.16.50.107
Not secure
10.16.50.107
Free MP3...
Dashboar...
Urban Dic...
HyperWrit...
Watch2Ge...
iCloud Mail
Prime Video
Warfare Hi...
Other favorites
Instance ID : i-0b02f8a3ab6a01319

With vpn off:



.....

“Successfully set up a client side vpn using simple AD service, AWS cli, directory service, vpc, vpn, and verified it using an ec2 instance within the vpc network.”