PROJECT – 5

CREATING AN EC2 INSTANCE USING ANSIBLE

This project involves automating the creation and configuration of an EC2 instance on AWS using Ansible, a popular IT automation tool. The main goal is to streamline the process of provisioning and managing cloud infrastructure with minimal manual intervention.

What is Ansible?

Ansible is an **open-source IT automation tool** used for configuration management, application deployment, task automation, and orchestration. It allows users to define the desired state of their infrastructure in a **declarative way** using YAML-based playbooks.

Ansible operates in an **agentless** manner, meaning it doesn't require any special software or agents to be installed on the managed systems. It uses **SSH (Secure Shell)** for communication and works seamlessly across multiple operating systems.

Key Features of Ansible

1. **Agentless Architecture**: No need to install agents on managed systems.
2. **Declarative Syntax**: Uses simple, human-readable YAML files for defining automation tasks.
3. **Idempotency**: Ensures that tasks only apply changes when needed, avoiding redundant operations.
4. **Extensibility**: Supports plugins and custom modules for enhanced functionality.

Advantages of Ansible

1. **Ease of Use**: Simple to learn and implement due to its YAML-based syntax and intuitive commands.
2. **Agentless**: Eliminates the need for agents, reducing overhead and simplifying setup.
3. **Cross-Platform Support**: Works with various operating systems, including Linux, macOS, and Windows.
4. **Idempotency**: Ensures tasks do not cause unintended side effects if rerun multiple times.
5. **Extensive Community Support**: A large, active community contributes modules, plugins, and updates.

6. **Integration with Cloud Providers**: Easily integrates with AWS, Azure, GCP, and other platforms for managing cloud infrastructure.
7. **Scalability**: Handles both small-scale automation and large-scale orchestration efficiently.
8. **Security**: Leverages SSH for secure communications without the need to open additional ports.

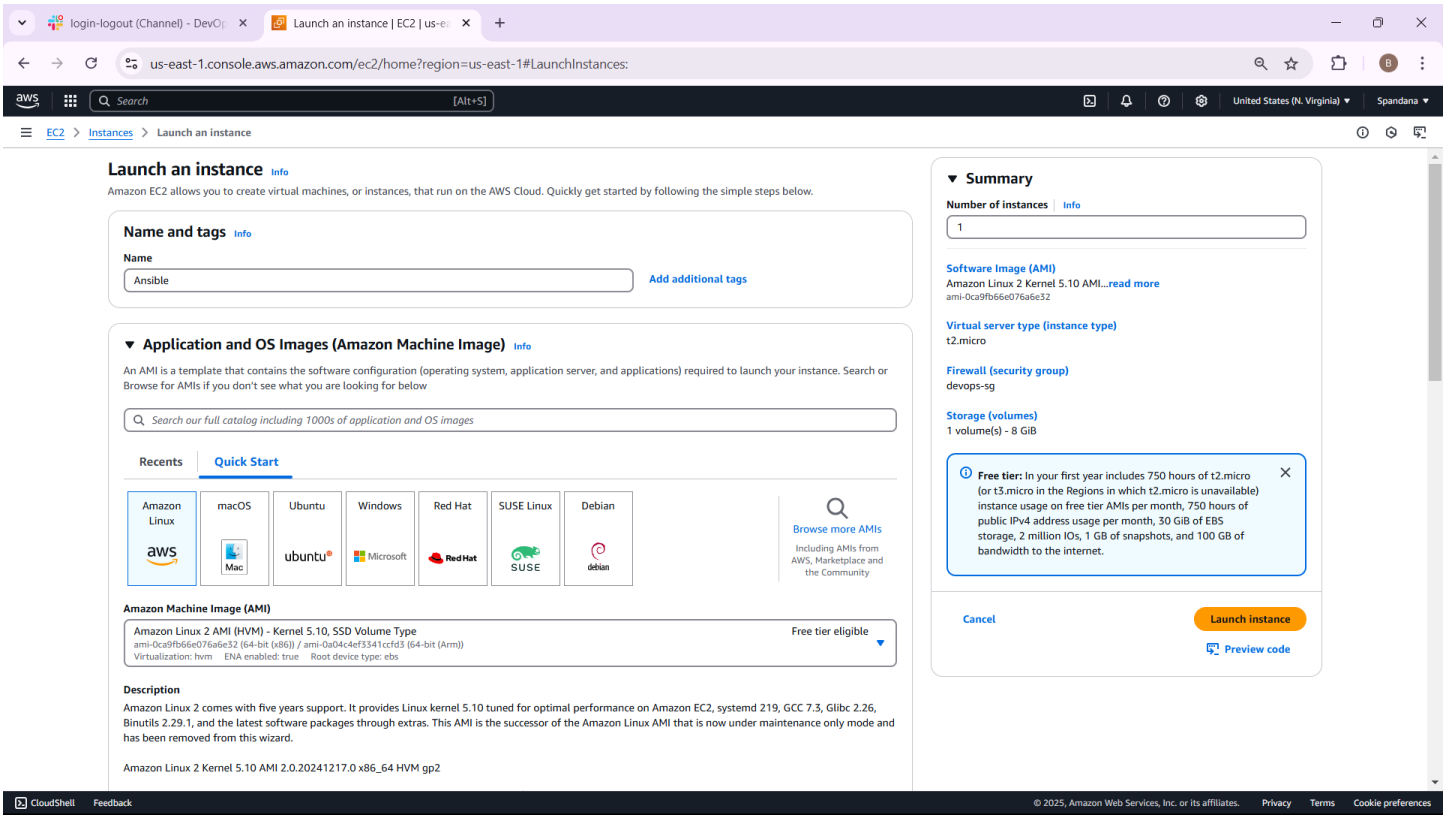
Disadvantages of Ansible

1. **Performance on Large-Scale Deployments**: Being agentless, Ansible can be slower compared to agent-based tools when managing a large number of nodes.
2. **Steeper Learning Curve for Complex Scenarios**: While basic tasks are straightforward, advanced configurations and custom module development can be challenging.
3. **Limited UI Options**: The Ansible Tower (paid version) provides a UI, but the free version is command-line driven.
4. **Dependency on Python**: Requires Python on managed systems, which may require installation in some environments.
5. **Not Real-Time**: Designed for task execution and configuration management but not for real-time monitoring or alerting.
6. **Complex Debugging**: Debugging playbooks can be difficult, especially for complex setups or when using third-party modules.

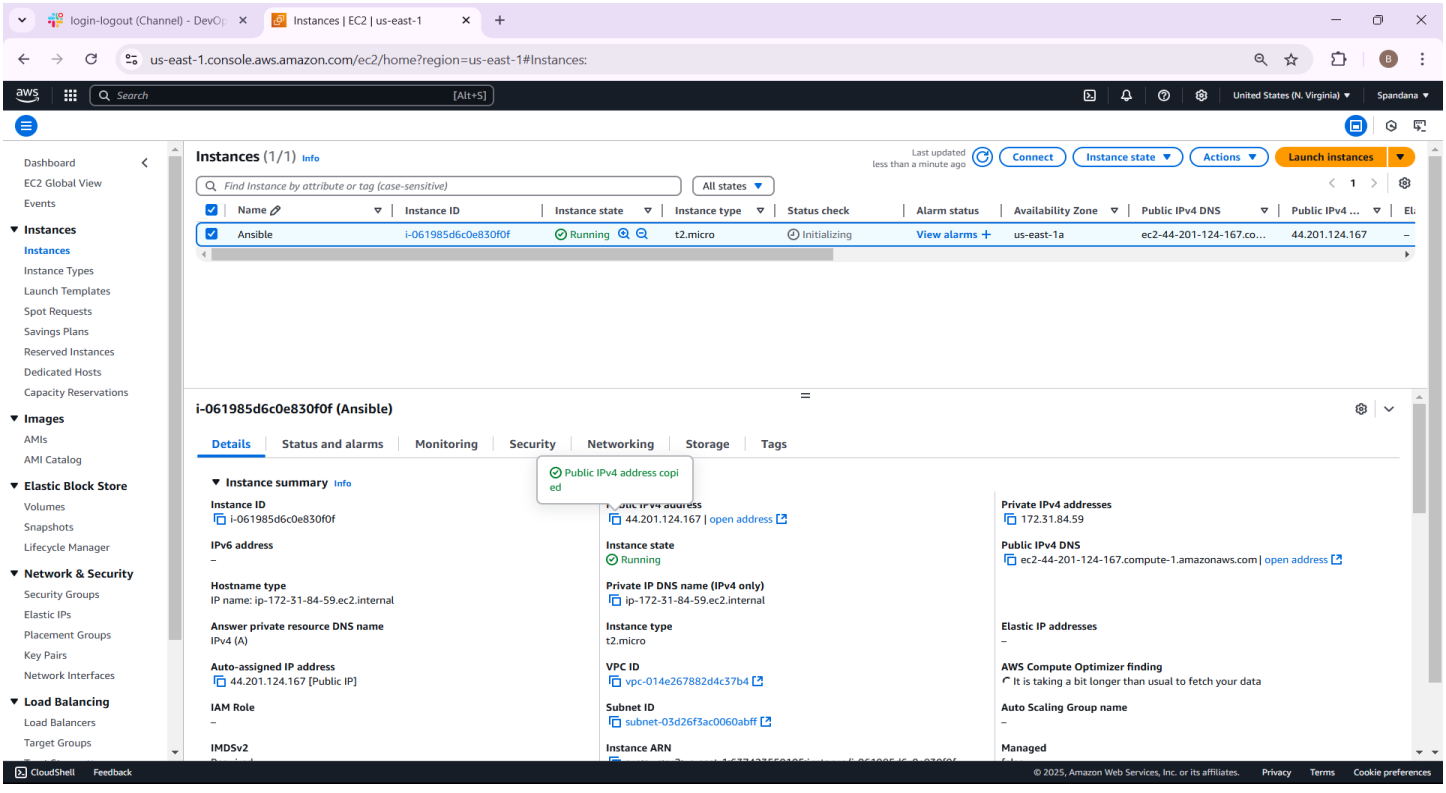
Use Cases for Ansible

1. **Configuration Management**: Managing server configurations consistently across environments.
2. **Application Deployment**: Automating deployments for web and enterprise applications.
3. **Orchestration**: Managing interdependent services across multiple systems.
4. **Cloud Provisioning**: Automating the creation and configuration of cloud infrastructure.

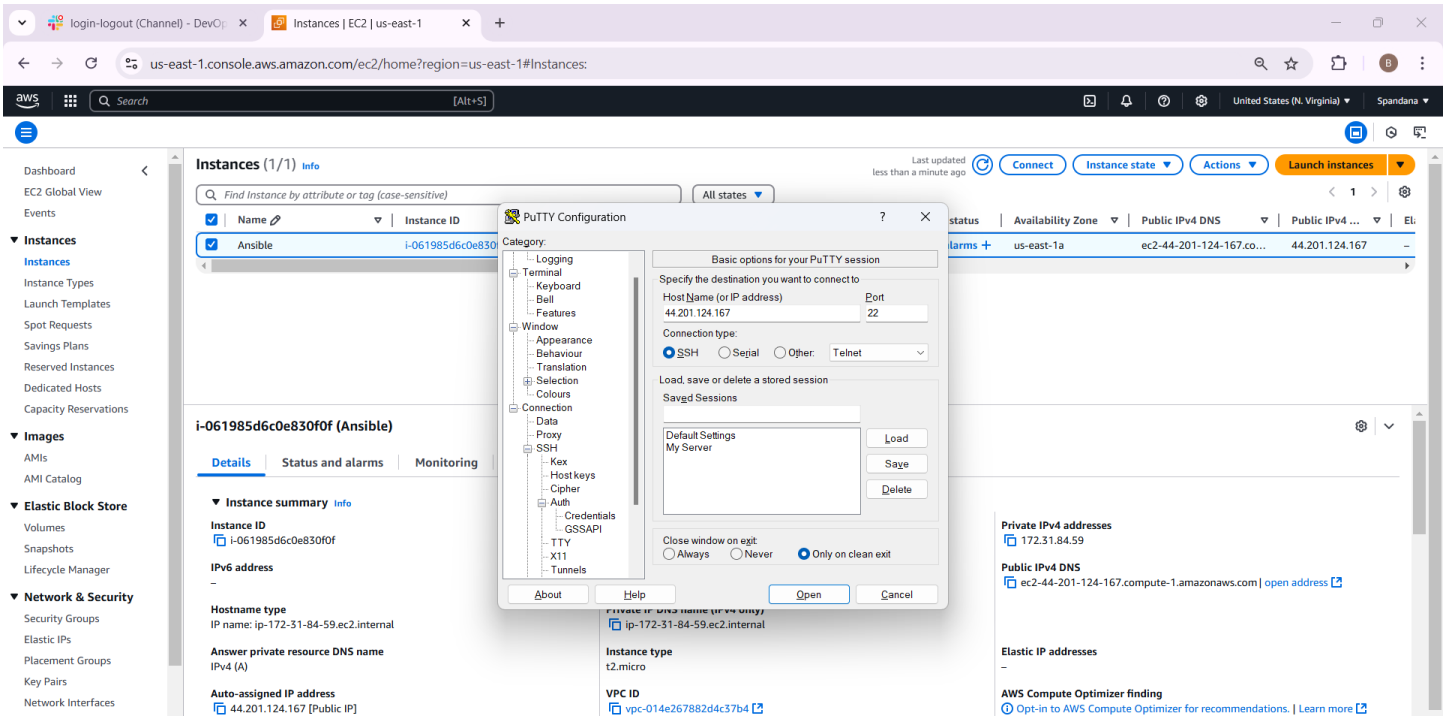
First we need to launch an instance with our own configuration and I named as Ansible and we need to wait it until it get into running state as shown in below image. After configuration we need to click on launch instance.



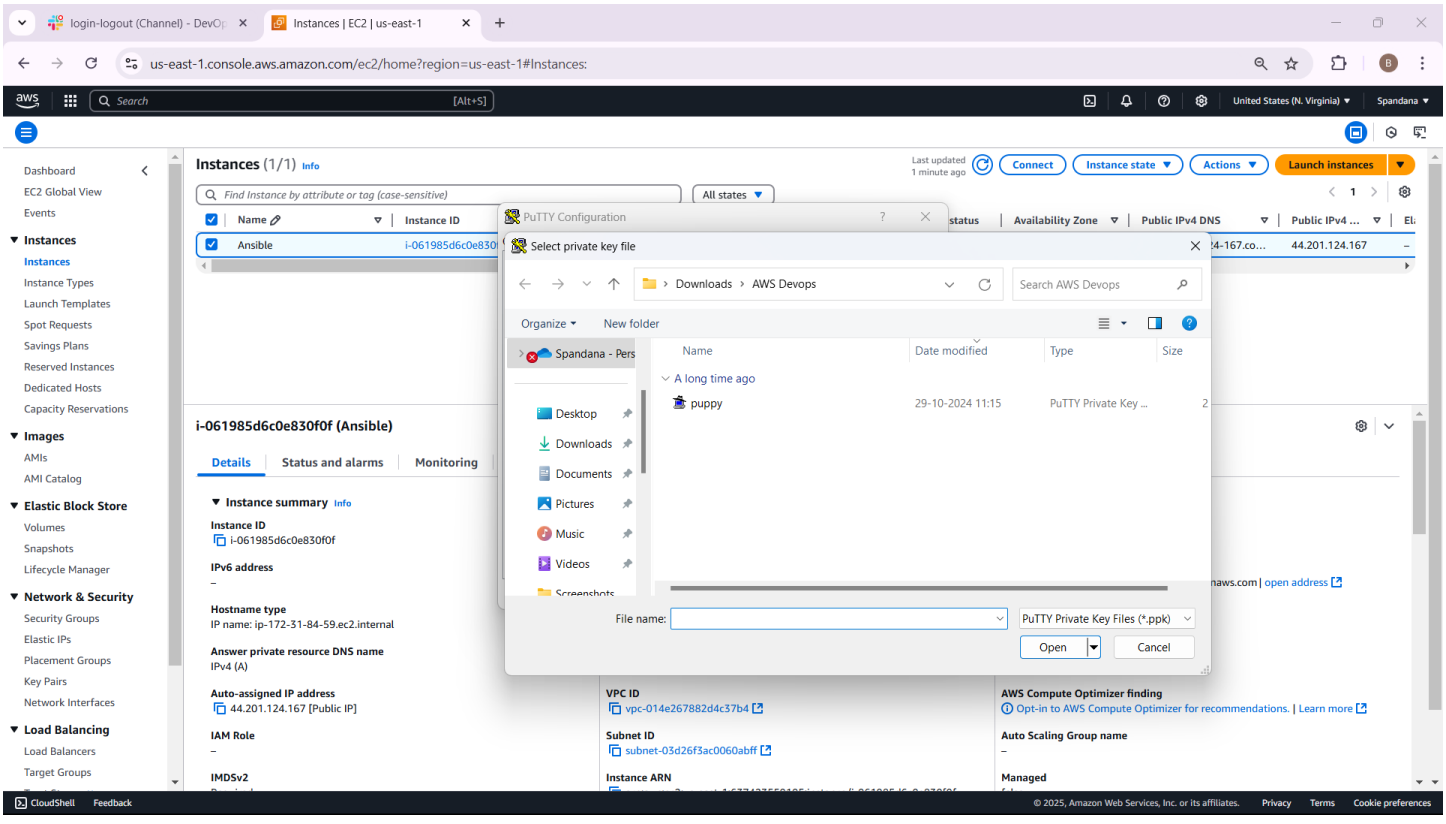
In the below image we can able to see instance state should be in running state as shown in below image.



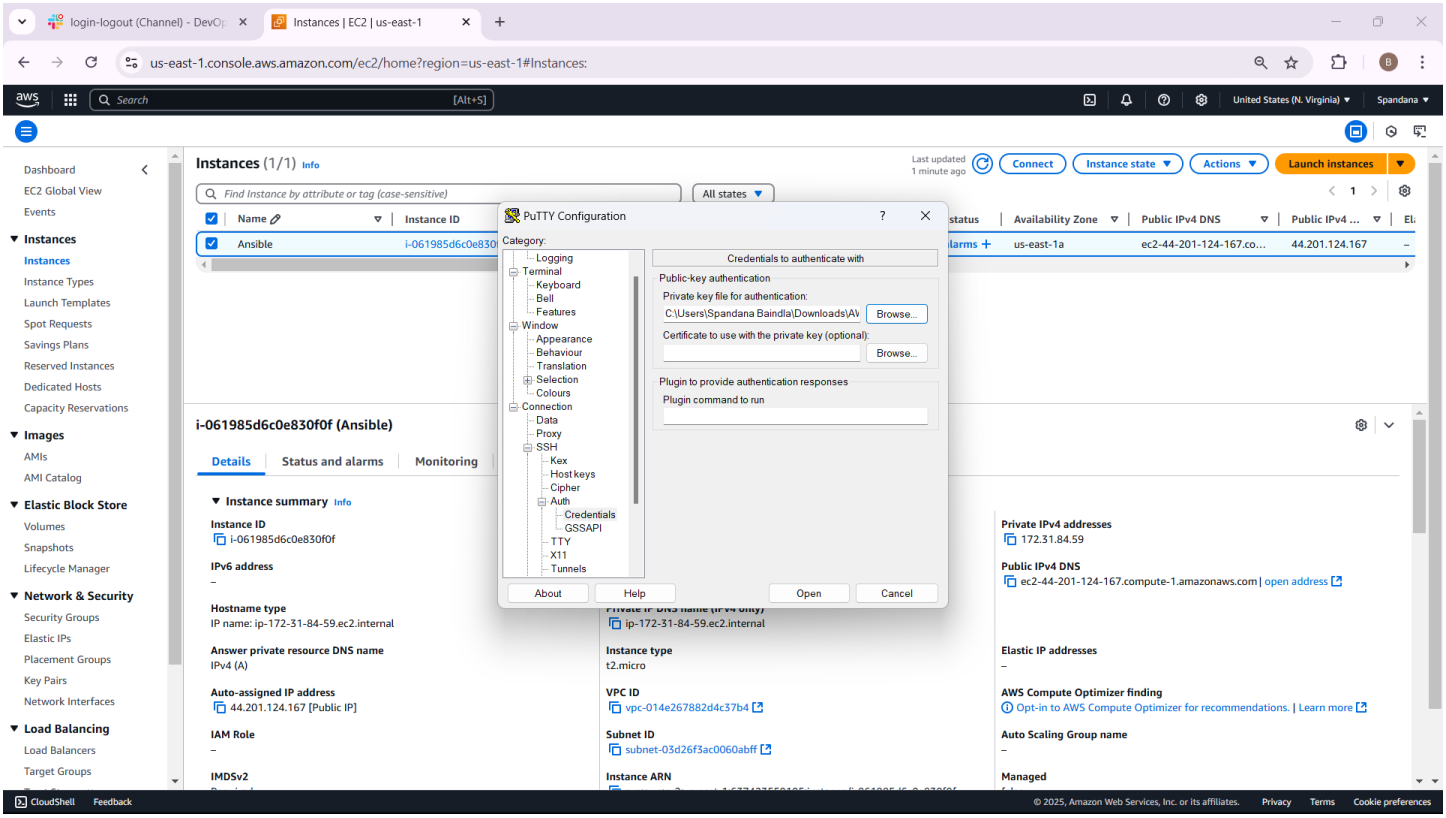
Then copy the Public IP of an instance and connect through PuTTY. First we need to copy and paaste the public IP of an instance in Host Name place and in the left hand side we can able to see SSH > Auth > Credentials click on Open as shown in below image.



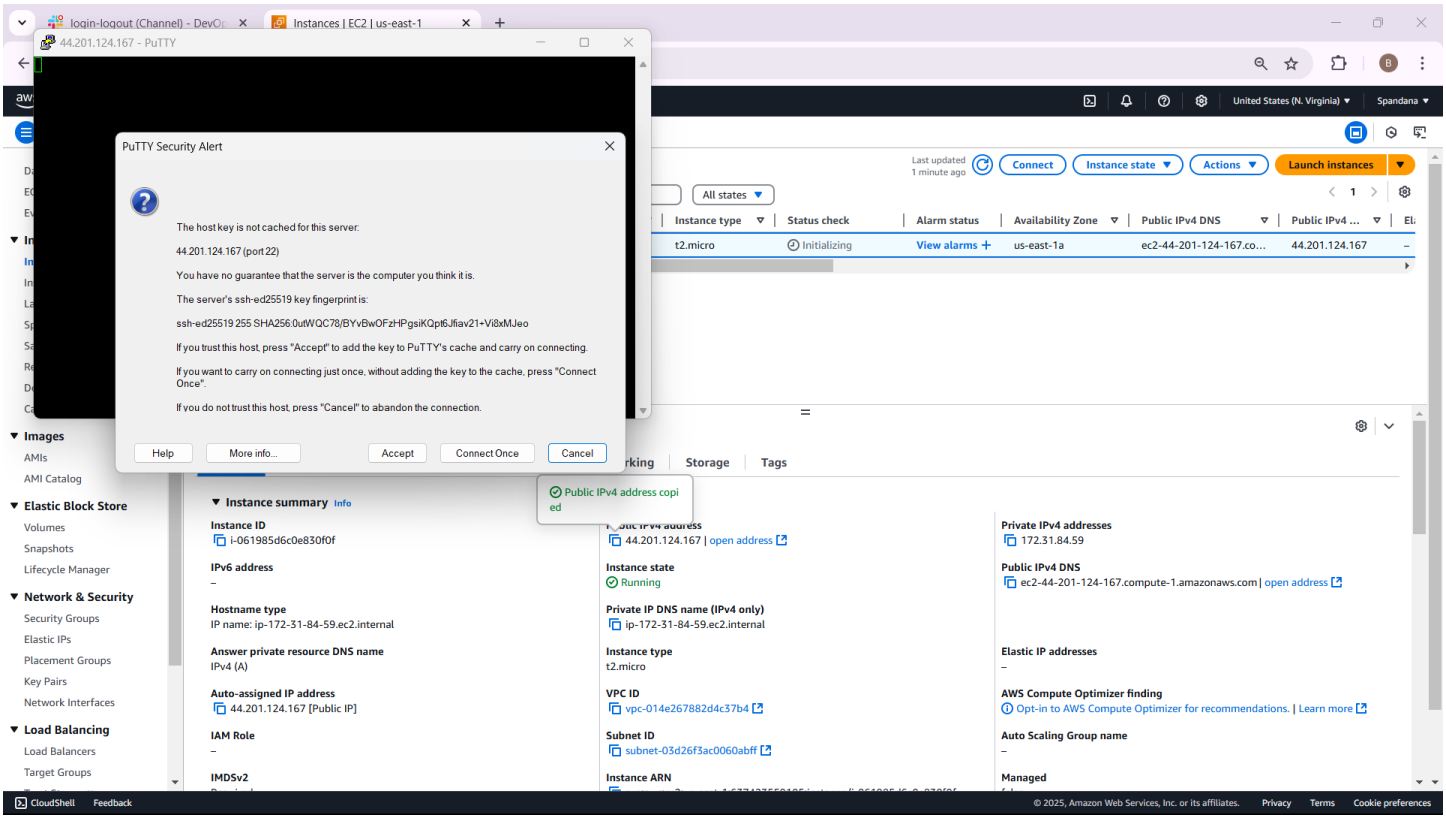
Then we need to select .ppk file that is shown in below image.



Click on browse and select .ppk file and click on open as shown below.



Then we will navigate to terminal as shown below and it shows a popup message for security purpose.



Install Required Software on Your Local Computer

- Install the following software
    - Python :
    - pip
    - boto and boto3
    - Ansible

1. Python

What is Python?

Python is a high-level, general-purpose programming language widely used for scripting, automation, and backend development.

Uses in the Project:

- **Base Requirement for Ansible:** Ansible is written in Python, so Python is essential for its execution.
- **AWS SDKs (boto/boto3):** Python enables the use of AWS SDKs like boto and boto3, which facilitate interactions with AWS services.

2. pip

What is pip?

pip is the package manager for Python, used to install and manage Python libraries and dependencies.

Uses in the Project:

- Installing required Python libraries such as boto, boto3, and any dependencies for Ansible modules.

3. boto and boto3

What are boto and boto3?

- **boto:** The original AWS SDK for Python, used to interact with AWS services.
- **boto3:** The improved and latest version of the AWS SDK for Python, offering more features and better performance.

Uses in the Project:

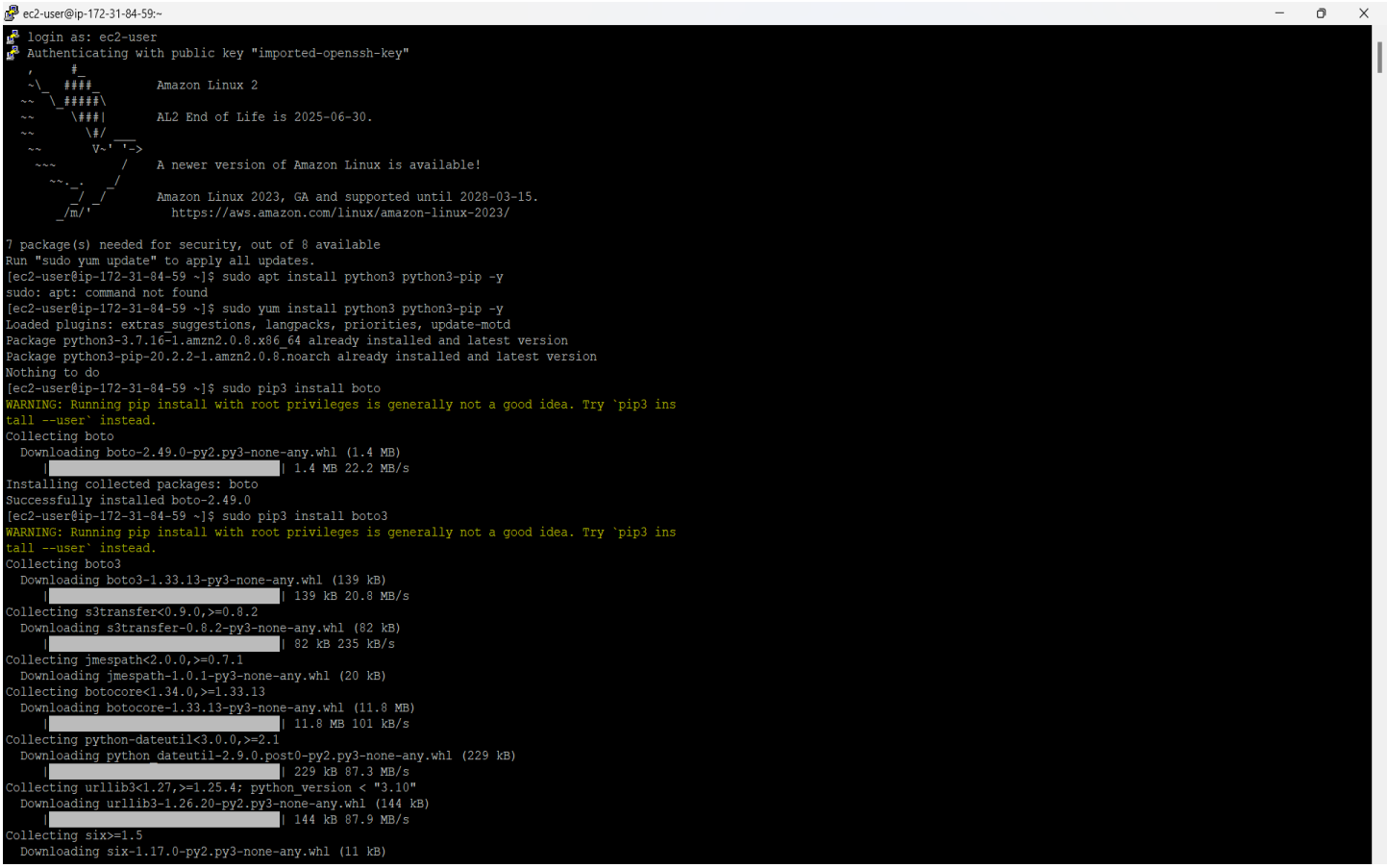
- **boto3** is used by Ansible to interact with AWS services for tasks like:

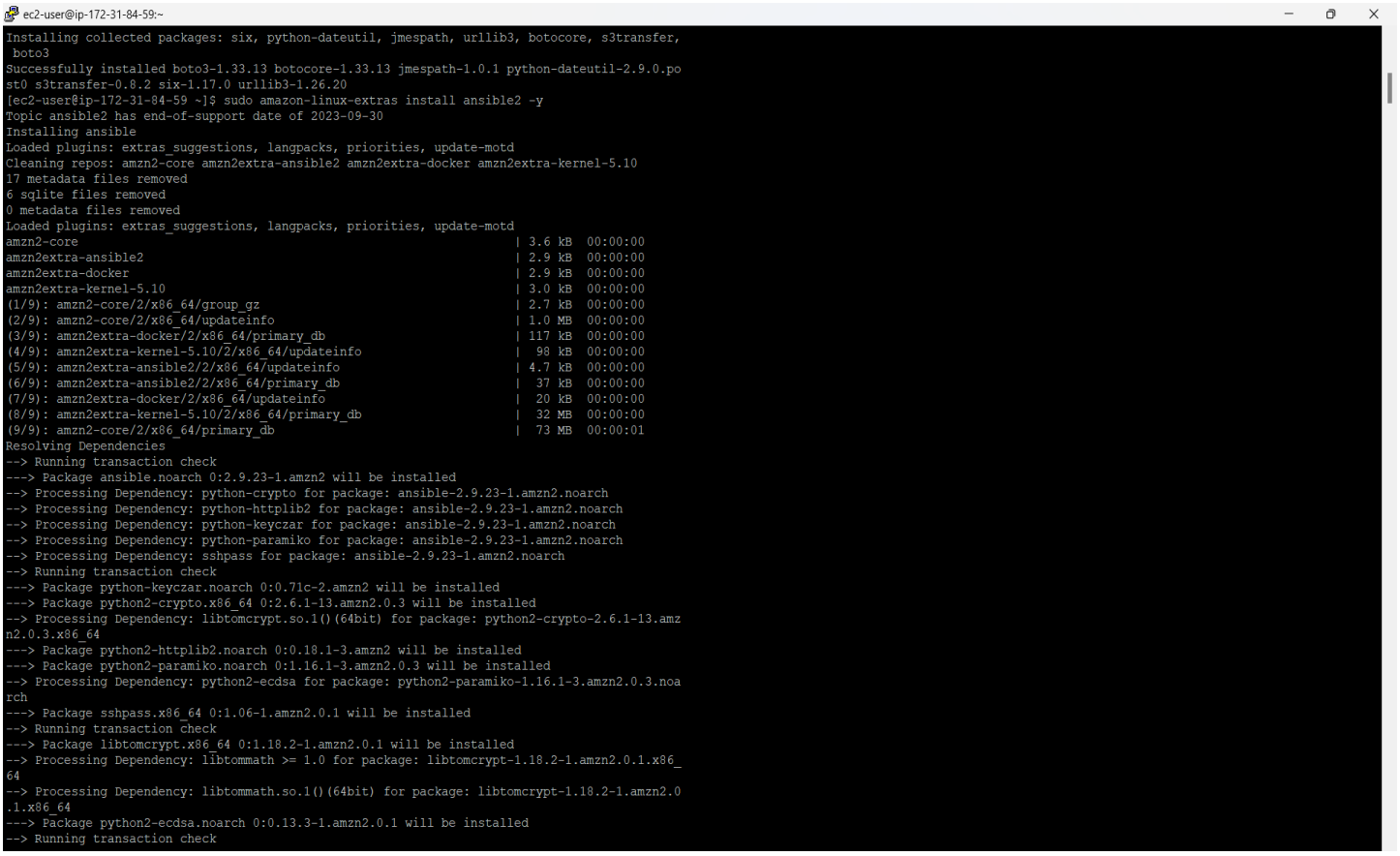
- Launching EC2 instances.
- Configuring networking (e.g., VPC, subnets).
- Managing resources like security groups and key pairs.

First we need to install installs **Python 3** and **pip** (Python's package manager) on your EC2 instance:

sudo yum install python3 python3-pip -y

- sudo: Runs the command with administrative (root) privileges.
- yum: Package manager for Amazon Linux to install software.
- install: Installs the specified software.
- python3: Installs Python 3.
- python3-pip: Installs pip for Python 3.
- -y: Automatically confirms the installation without asking for your permission.





To install and run Ansible on an EC2 instance, you need to enable and install the Ansible package. Amazon Linux provides a set of pre-packaged tools, and Ansible is one of them. You can use the command**:**

- **sudo amazon-linux-extras install ansible2 -y**

    1. amazon-linux-extras is a repository of additional software packages.
    2. install ansible2 installs the version 2 of Ansible.
    3. The -y flag automatically confirms the installation without prompting you.
    4.

To use Ansible for managing AWS resources, you need to install the boto3 and botocore libraries. These Python libraries act as a bridge between Ansible and AWS, allowing Ansible to communicate with AWS services.

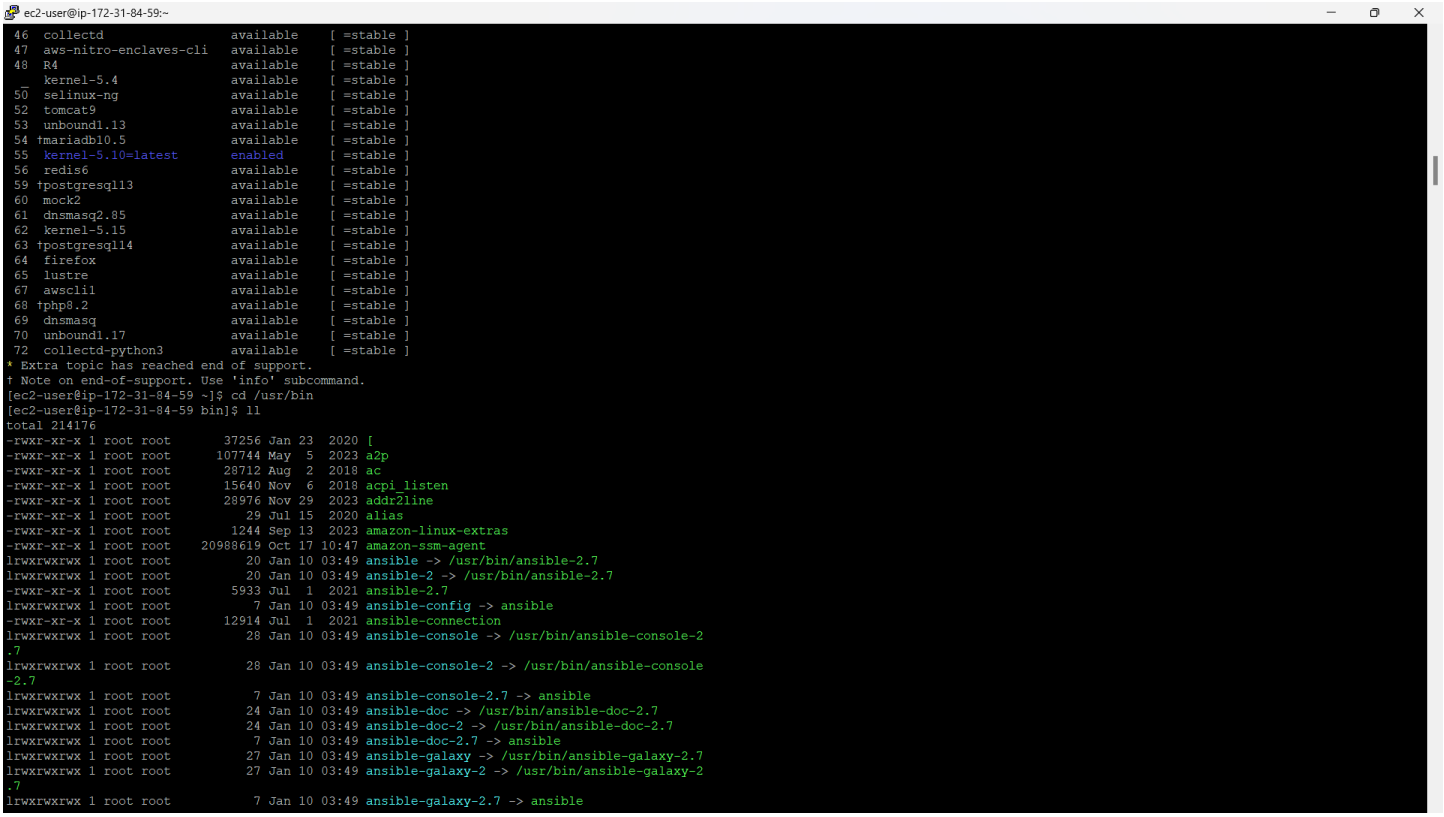
Navigate to the /usr/bin directory

- cd /usr/bin
    - This is where Python and related tools are installed on the EC2 instance.
    - It ensures you're in the correct environment for Ansible to function properly.

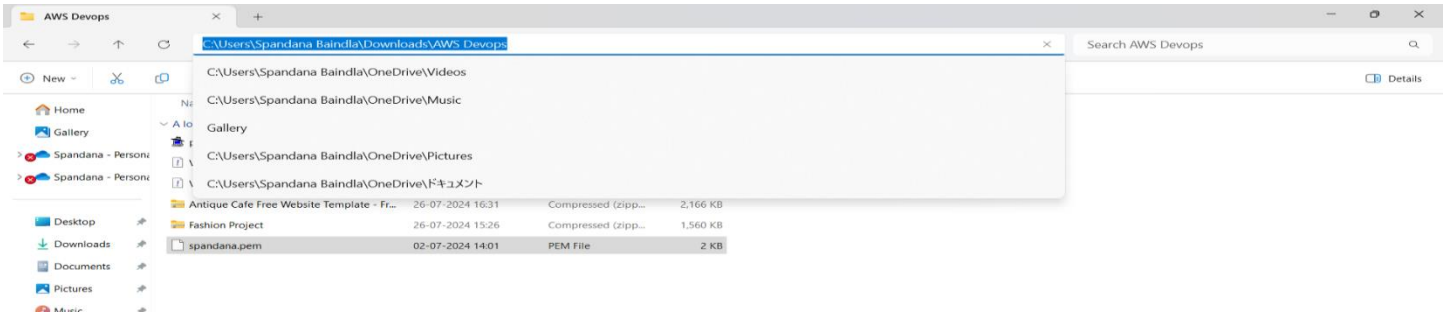
Install boto3 and botocore:

- sudo pip3 install boto3
- sudo pip3 install botocore
    - pip3 is the package manager for Python 3.
    - boto3 is the AWS SDK for Python, enabling interaction with AWS services.
    - botocore is a low-level core library used by boto3 for handling AWS service requests.

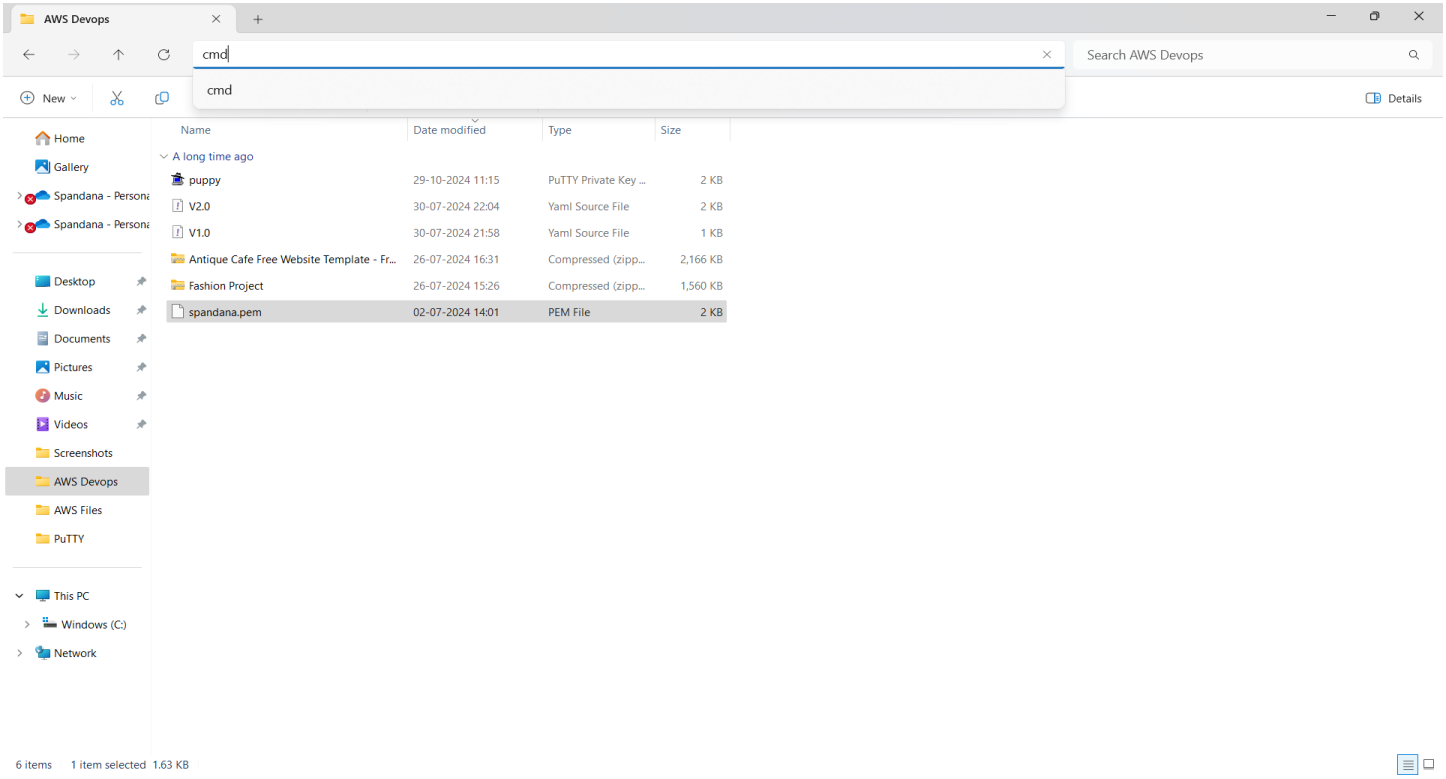
By running this command, you ensure that Ansible has the necessary tools to create and manage AWS resources, such as EC2 instances and VPCs.



First we need to go to .pem file folder where we are stored and select that .pem file as shown in below image.



Type cmd in that path it will directly navigate to terminal.



You successfully used the Secure Copy Protocol (SCP) to transfer a .pem file from your local machine to your AWS EC2 instance.

1. **Command Executed:**

   scp -i spandana.pem spandana.pem ec2-user@44.201.124.167:/home/ec2-user/

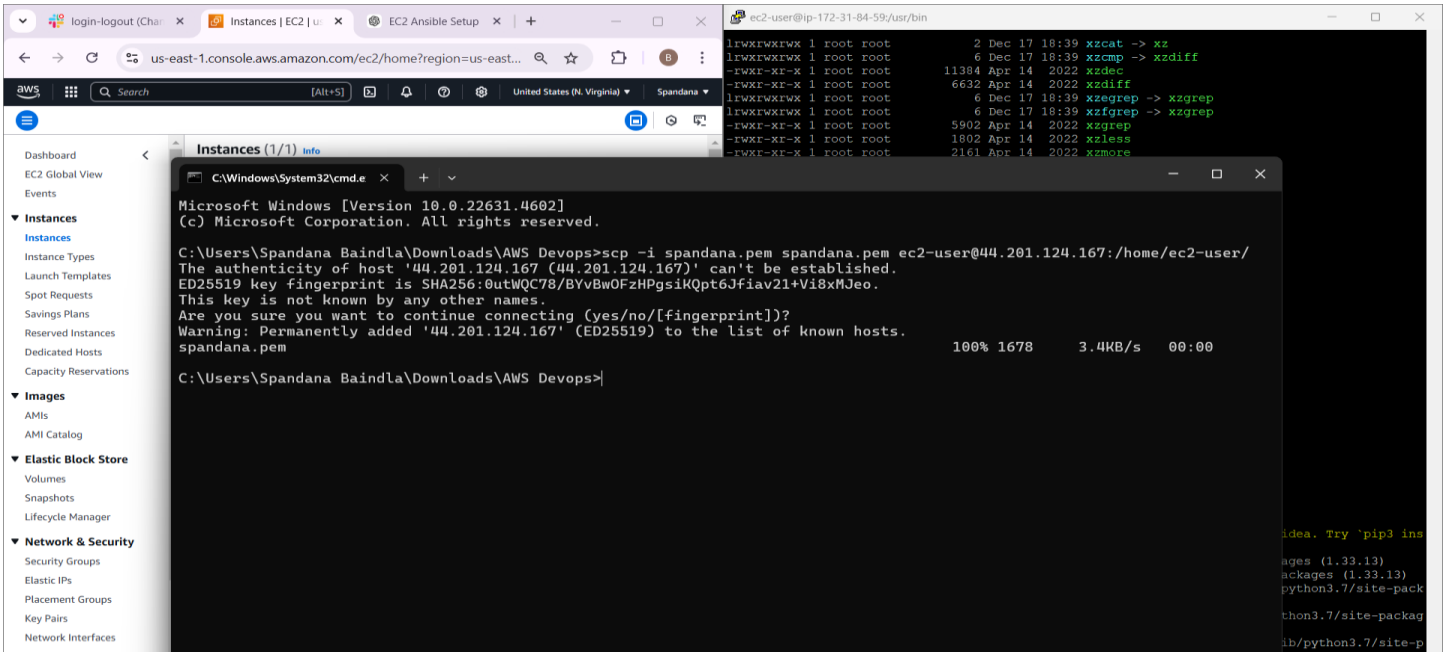
   - scp: Securely copies files between a local machine and a remote server.
   - -i spandana.pem: Specifies the private key file for SSH authentication.
   - spandana.pem: The file you are copying.
   - ec2-user@44.201.124.167: The remote server's user (ec2-user) and public IP address.
   - /home/ec2-user/: The destination directory on the EC2 instance.

2. **Host Authenticity Check:**
    - Since it's your first connection to the EC2 instance, the authenticity of the host is not yet verified.
    - The message asks for confirmation:
      Are you sure you want to continue connecting (yes/no/[fingerprint])?
    - You responded with yes, so the host's key was added to your known hosts file.
3. **File Transfer Confirmation:**
    - The file spandana.pem was successfully transferred to the EC2 instance.
    - Details:
        - 100% 1678: Indicates the file size (1,678 bytes) and the completion status.
        - 3.4KB/s: The transfer speed.
        - 00:08: Time taken for the transfer.

Your spandana.pem file is now securely copied to /home/ec2-user/ on the EC2 instance. Make sure to update its permissions using:

- **sudo chmod 600 /home/ec2-user/spandana.pem**

This process allows you to securely copy your private key to your EC2 instance, which will be useful for SSH connections and managing your EC2 environment.



In Ansible, the inventory file (usually located at /etc/ansible/hosts) contains the details of the hosts or machines that Ansible will manage. In your case, you're updating the file to include your EC2 instance.

Explanation of the Inventory File Entry

[localhost]

127.0.0.1 ansible_host=172.31.84.59 ansible_python_interpreter=/usr/bin/python3 ansible_user=ec2-user ansible_ssh_private_key_file=/home/ec2-user/spandana.pem

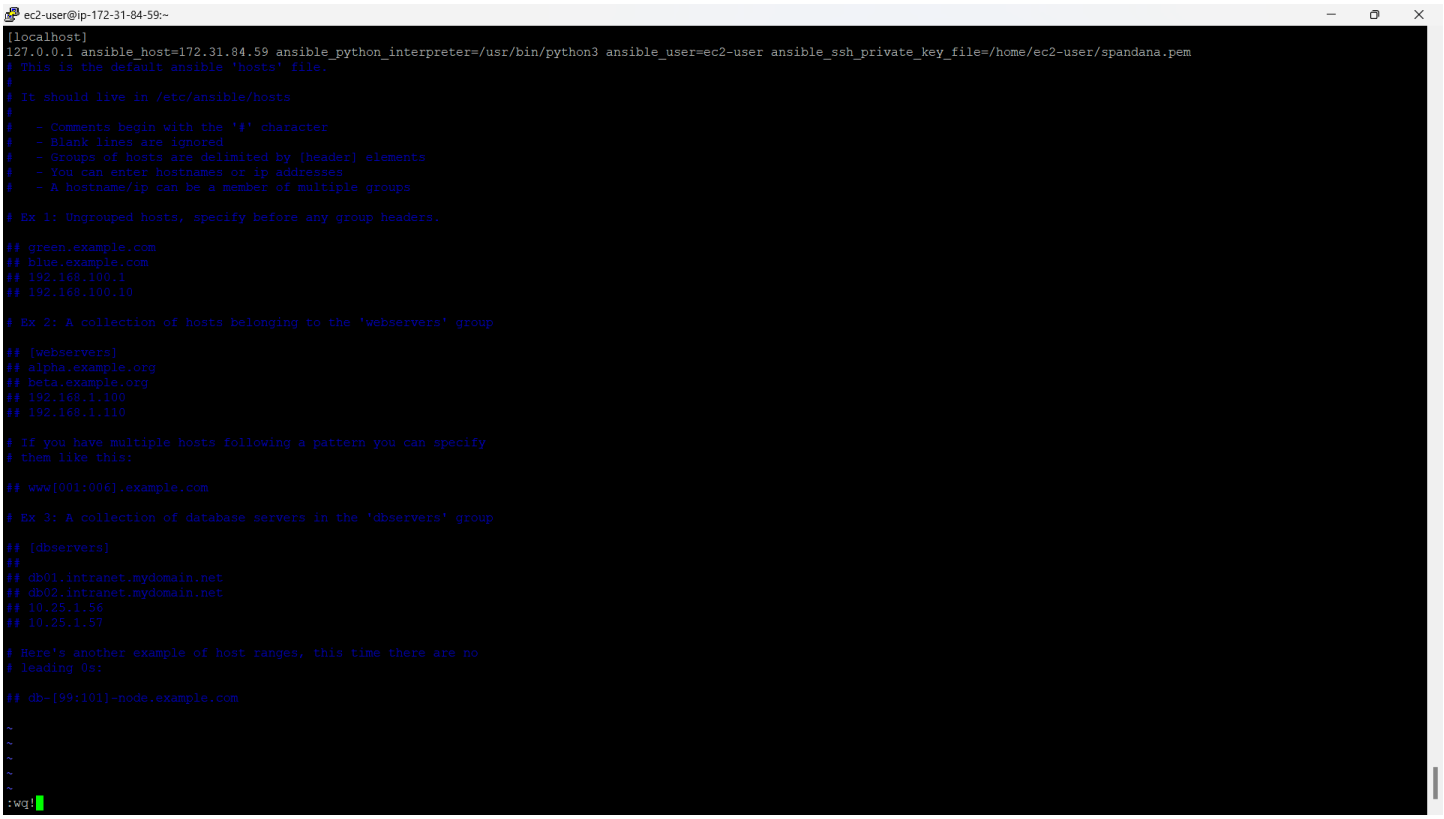
1. [localhost]:
   This is the group name. In this case, it's labeled localhost, but you can use any name. It's essentially an identifier for the group of hosts that you want to manage.
2. 127.0.0.1:
   This is the loopback address (local address) for the machine where you're running Ansible (your local machine). Ansible will use this to connect to the EC2 instance.
3. ansible_host=172.31.84.59:
   This defines the actual IP address of the EC2 instance you're managing. Ansible will connect to 172.31.84.59 to run commands on the EC2 instance.
4. ansible_python_interpreter=/usr/bin/python3:
   Specifies the Python interpreter that Ansible should use on the EC2 instance. Since Ansible runs on Python, this tells it to use Python 3, which is installed on the instance.
5. ansible_user=ec2-user:
   This specifies the user Ansible should use when connecting to the EC2 instance. ec2-user is the default SSH user for Amazon Linux-based EC2 instances.
6. ansible_ssh_private_key_file=/home/ec2-user/spandana.pem:
   This tells Ansible to use the private key file (spandana.pem) to authenticate the connection when connecting to the EC2 instance. This file should have the correct permissions set (e.g., chmod 600).

Why is This Important?

This configuration enables Ansible to:

- Find and connect to your EC2 instance using its private IP address.
- Use Python 3 on the instance for execution.

- Authenticate the connection securely using the specified .pem key.



We need to create two files vpc.yaml file and secrets.yaml

Ansible_Playbook script with outline vairables

vpc.yaml file

1. Setting Up the Playbook:

- Host and Privilege: The playbook runs on localhost and uses elevated privileges (become: yes).
- Facts Gathering: The gather_facts: false disables automatic system fact-gathering to speed up execution.
- Variables: The playbook includes a secret.yaml file containing sensitive information like AWS access keys, secret keys, and other required variables.

2. VPC Creation:

The ec2_vpc_net module creates a Virtual Private Cloud (VPC) with specific configurations:

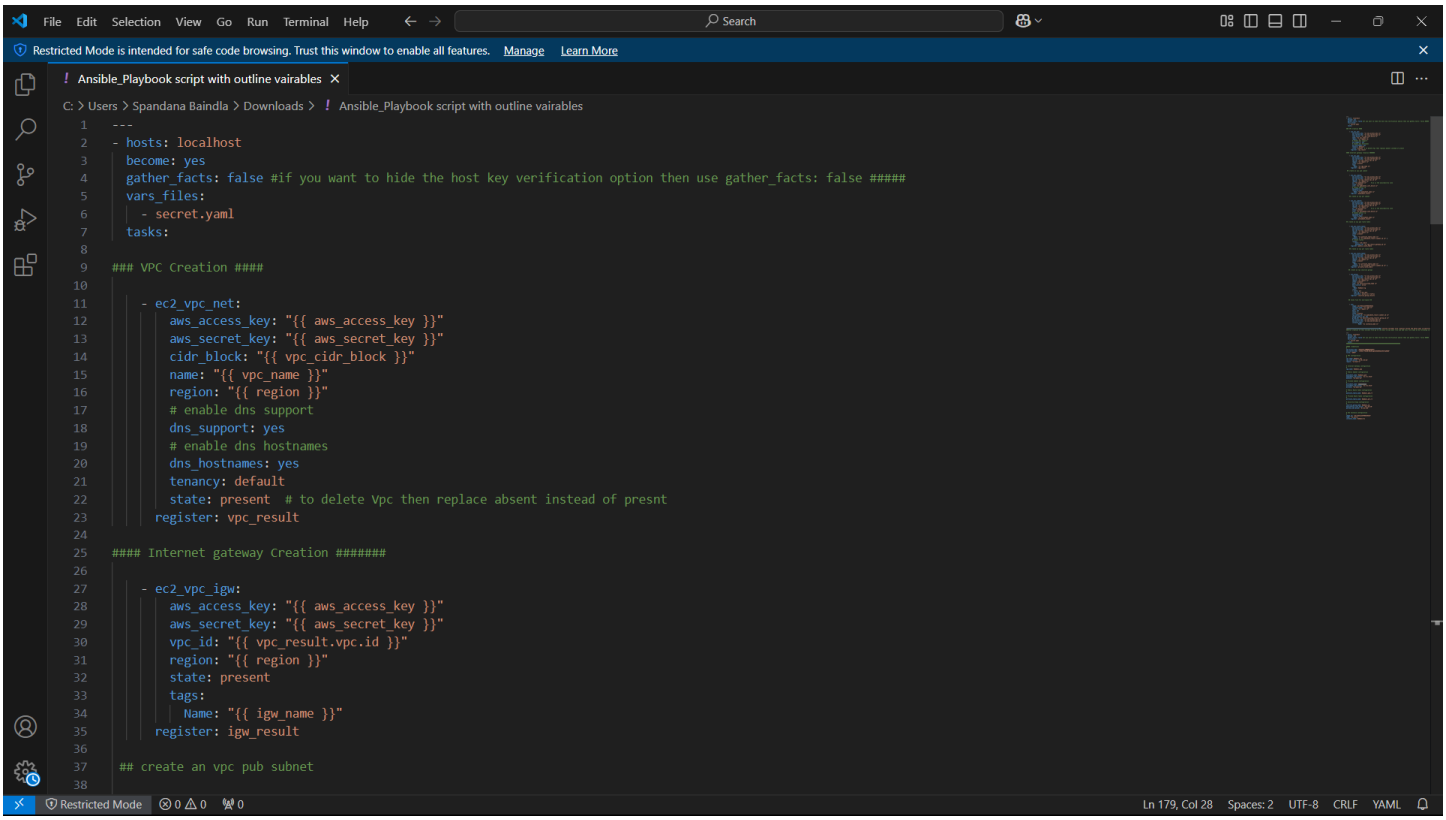
- AWS Credentials: Uses the AWS access and secret keys from the variables.
- CIDR Block: Defines the network range of the VPC using the vpc_cidr_block variable.
- Name and Region: The VPC is named based on vpc_name and created in the specified region.

- DNS Settings:
    - Enables DNS support to resolve domain names within the VPC.
    - Enables DNS hostnames to assign DNS names to AWS resources like EC2 instances.
- Tenancy: Set to default, which means instances can share hardware with other AWS accounts.
- State: If state: present, the VPC is created; if absent, the specified VPC is deleted.
- Output: The vpc_result variable stores the details of the created VPC, including its ID.

3. Internet Gateway (IGW) Creation:

The ec2_vpc_igw module creates and attaches an Internet Gateway (IGW) to the VPC:

- AWS Credentials: Uses the same AWS access and secret keys.
- VPC Association: The vpc_id is dynamically set using the VPC ID from the vpc_result variable.
- Region: Specifies the AWS region where the IGW is created.
- State: If state: present, the IGW is created and attached; if absent, it is detached and deleted.
- Tags: Assigns a name tag to the IGW for identification, using the igw_name variable.



1. Public Subnet Creation

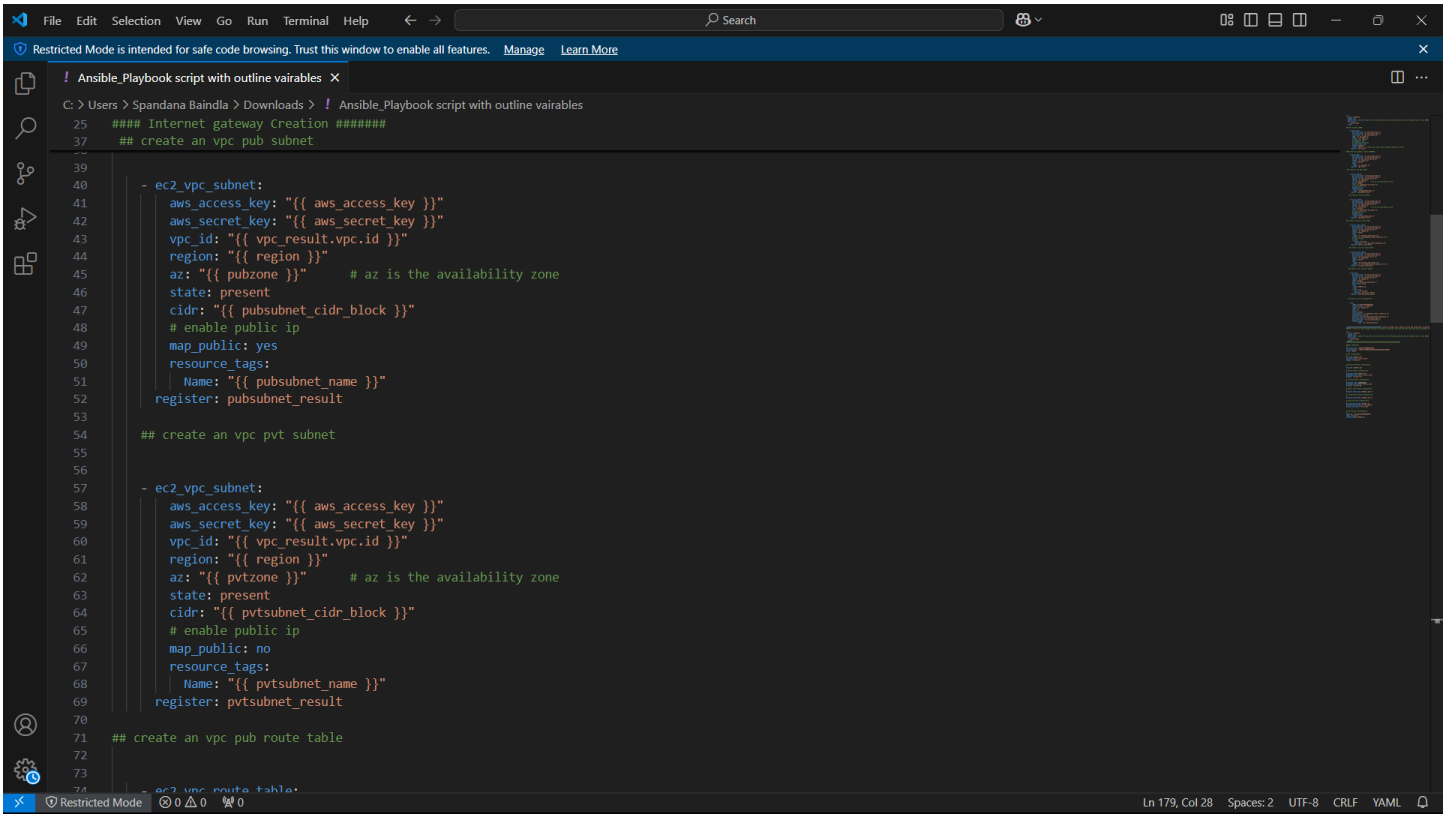
The public subnet is created with the following properties:

- AWS Credentials: Uses the AWS access key (aws_access_key) and secret key (aws_secret_key).
- VPC Association: The subnet is associated with the VPC created earlier, identified by the vpc_id obtained from vpc_result.vpc.id.
- Region and Availability Zone: The region is specified with region, and the availability zone (AZ) is defined using the pubzone variable.
- State: Setting state: present ensures the subnet is created (use absent to delete it).
- CIDR Block: The subnet's IP range is defined with the pubsubnet_cidr_block variable.
- Public IP Mapping: map_public: yes allows EC2 instances launched in this subnet to have public IPs by default, enabling them to communicate with the internet.
- Tagging: The subnet is tagged with a name using pubsubnet_name.
- Output: The details of the created subnet are stored in the pubsubnet_result variable.

2. Private Subnet Creation

The private subnet is created similarly to the public subnet, but with a key difference:

- Public IP Mapping Disabled: map_public: no ensures that EC2 instances launched in this subnet will not have public IPs by default, keeping them private and isolated from direct internet access.
- Availability Zone: The private subnet is created in the pvtzone AZ.
- Other Parameters: The vpc_id, region, state, cidr (set using pvtsubnet_cidr_block), and resource_tags (tagged using pvtsubnet_name) are specified in the same way as the public subnet.
- Output: The details of the private subnet are stored in the pvtsubnet_result variable.



This part of the playbook creates and configures public and private route tables for the VPC. Below is a detailed explanation:

1. Public Route Table Creation

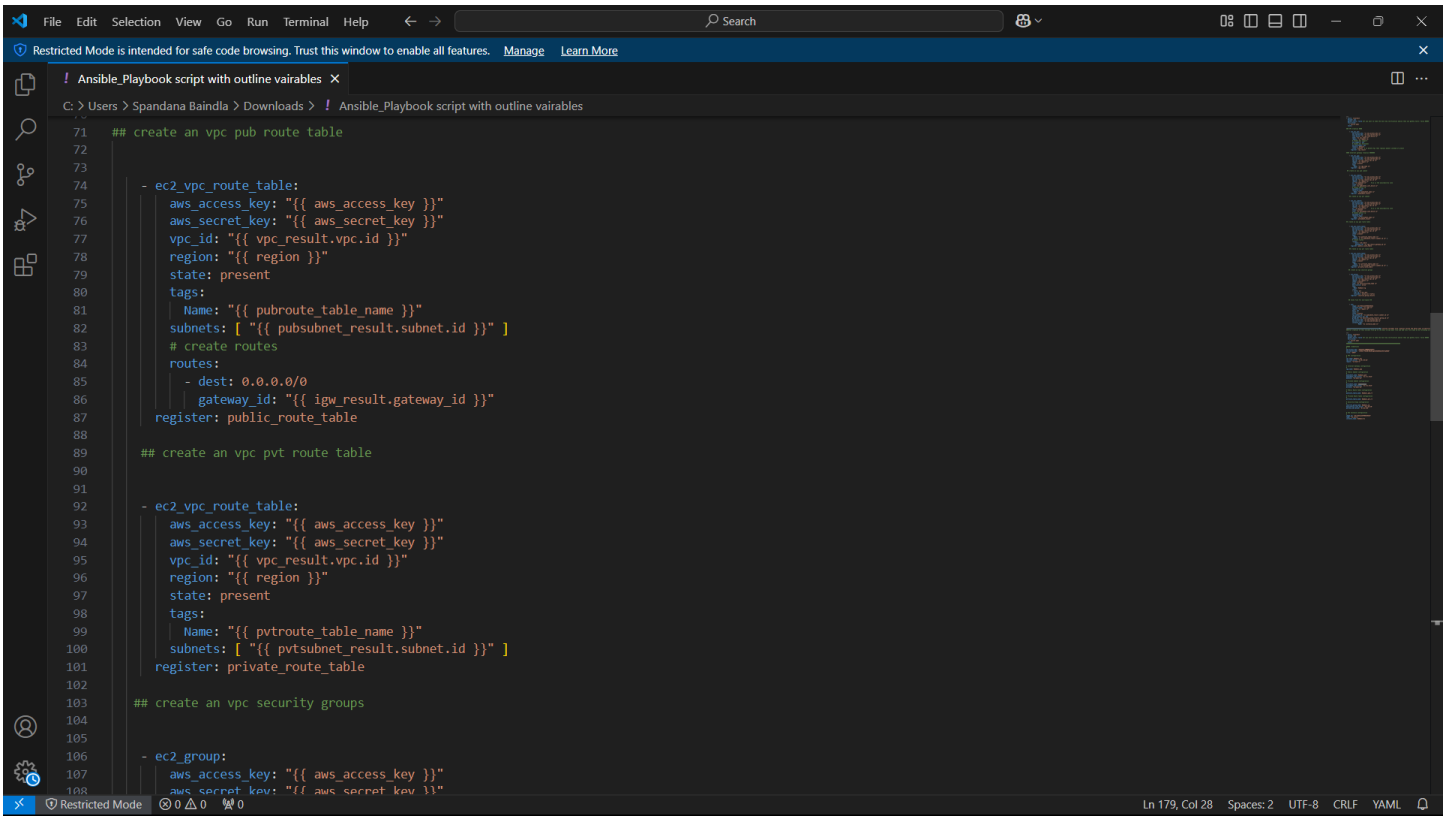
The public route table is created using the ec2_vpc_route_table module with the following details:

- AWS Credentials: Specifies the aws_access_key and aws_secret_key.
- VPC Association: Links the route table to the VPC created earlier using the vpc_id from vpc_result.vpc.id.
- Region: Sets the AWS region for the resource.
- State: Ensures the route table is created (state: present).
- Tagging: Tags the route table with a name using pubroute_table_name for easier identification.
- Subnet Association: Associates the public route table with the public subnet using the subnets parameter and the ID of the public subnet stored in pubsubnet_result.subnet.id.
- Routing Configuration: Adds a route to enable internet access:
    - Destination: 0.0.0.0/0 (the default route for all outbound internet traffic).
    - Gateway: Uses the Internet Gateway (IGW) created earlier, identified by igw_result.gateway_id.
- Output: Stores the details of the public route table in the public_route_table variable.

2. Private Route Table Creation

The private route table is created similarly but with notable differences:

- Subnet Association: Associates the private route table with the private subnet using pvtsubnet_result.subnet.id.
- Routing Configuration: No internet route (routes is not defined) since this route table is meant for private subnets.
- Tagging: The route table is named using pvtroute_table_name for identification.
- Output: Details of the private route table are stored in the private_route_table variable.



1. Create a VPC Security Group

The ec2_group module is used to create a security group with the following specifications:

- AWS Credentials: Uses aws_access_key and aws_secret_key for authentication.
- VPC Association: The security group is created within the VPC, identified by vpc_id from vpc_result.vpc.id.
- Region: Specifies the AWS region where the security group will reside.

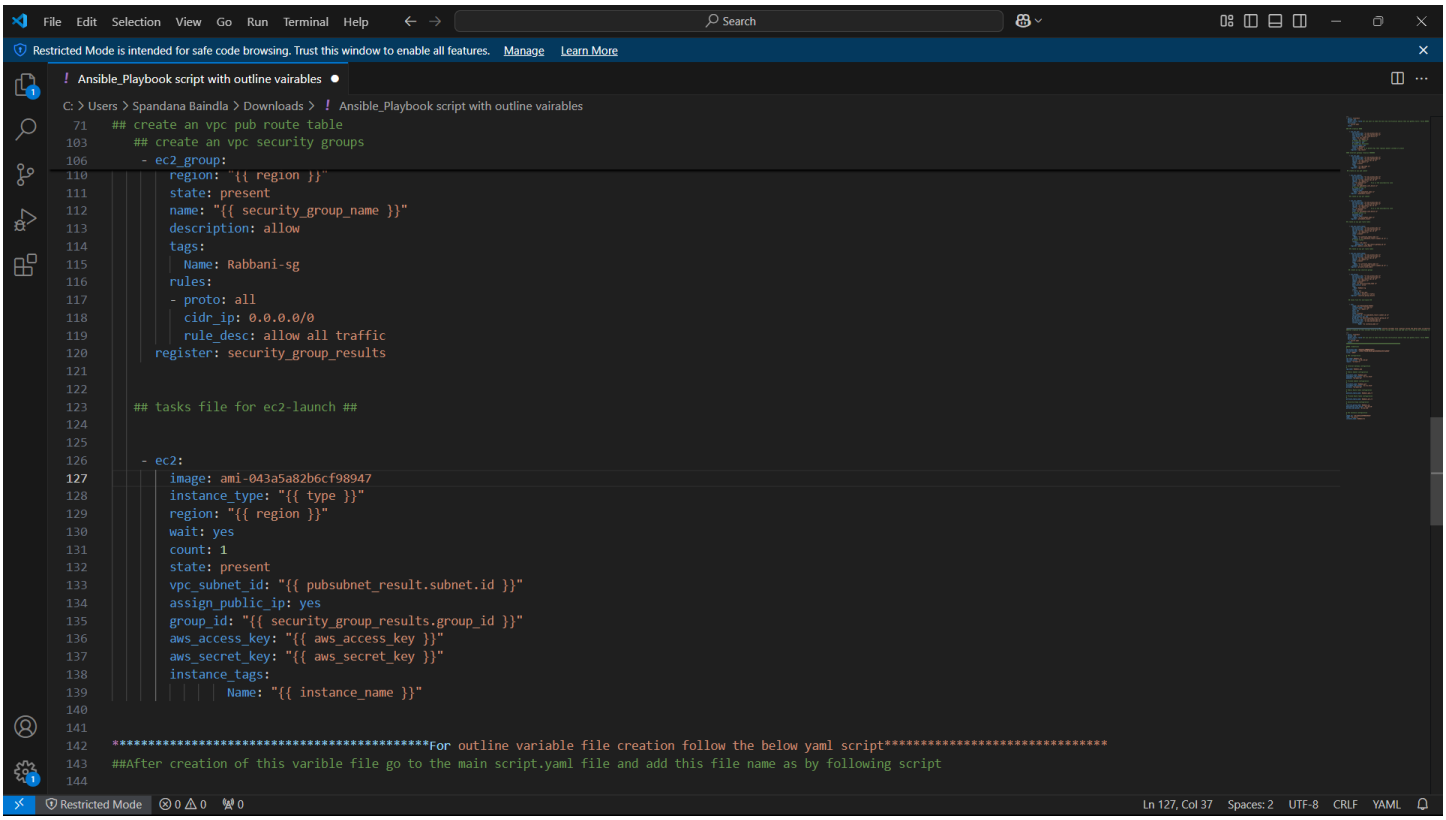
- State: Ensures the security group is created (state: present).
- Name and Description:
    - name specifies the security group name, set dynamically with security_group_name.
    - description provides a basic description of the group (e.g., "allow").
- Tags: Tags the security group with spandana-sg for easy identification in AWS resources.
- Rules: Allows all traffic (all protocols and all ports) from any IP address:
    - proto: all allows all protocols.
    - cidr_ip: 0.0.0.0/0 permits access from any IP address.
    - rule_desc: allow all traffic adds a descriptive note for the rule.
- Output: The created security group details, including the group_id, are stored in the security_group_results variable.

2. Launch an EC2 Instance

The ec2 module is used to launch an EC2 instance with these properties:

- AMI ID: The Amazon Machine Image (AMI) is defined as ami-043a5a82b6cf98947. This is the base image for the instance (you should verify its availability in the specified region).
- Instance Type: The instance type is set dynamically using the type variable, specifying the hardware configuration (e.g., t2.micro).
- Region: Indicates where the EC2 instance will be launched.
- Wait for Completion: wait: yes ensures the playbook waits until the instance is fully provisioned before proceeding.
- Count: Launches a single instance (count: 1).
- State: Ensures the instance is created (state: present).
- Subnet Association: Places the EC2 instance in the public subnet using vpc_subnet_id from pubsubnet_result.subnet.id.
- Public IP Assignment: assign_public_ip: yes ensures the instance gets a public IP for internet connectivity.

- Security Group Association: Associates the instance with the created security group using group_id from security_group_results.group_id.
- Tags: Tags the instance with a name using the instance_name variable.



Below we have secrets.yaml file

General Information

- Access Keys:
  - aws_access_key: "ARTM7"
  - aws_secret_key: "Oz7 "

    These keys are used to authenticate and authorize actions in the AWS account.
- Title: "ARTH" (It might be a project name or an identifier for the setup.)

VPC Configuration

- VPC Name: Spandana_vpc

  A virtual private cloud named "Spandana_vpc" will be created.

- CIDR Block: 15.0.0.0/16

  Specifies the IP range for the VPC.

- Region: us-east-1

  All resources will be provisioned in the "US East (N. Virginia)" region.

Internet Gateway Configuration

- Name: Spandana_igu

  An internet gateway will be created and attached to the VPC for external connectivity.

Public Subnet Configuration

- Name: Spandana_pub

  A public subnet will be created with the following:

  - CIDR Block: 11.0.1.0/24

    Allocates a specific range of IPs for the subnet.

  - Availability Zone: us-east-1a

    The subnet will reside in the specified availability zone.

Private Subnet Configuration

- Name: Spandana_pvt

  A private subnet will be created with the following:

  - CIDR Block: 11.0.2.0/24

    Allocates a specific range of IPs for the subnet.

  - Availability Zone: us-east-1b

    The subnet will reside in the specified availability zone.

Public Route Table Configuration

- Name: Spandana_pub_rt

  A public route table will be created and associated with the public subnet. It will include a route to the internet gateway, enabling internet access for resources in the public subnet.

Private Route Table Configuration

- Name: Spandana_pvt_rt

  A private route table will be created and associated with the private subnet. This subnet won't have direct internet access unless configured with a NAT gateway**.**

Security Group Configuration

- Name: Spandana_sg

  A security group named "Spandana_sg" will be created with the following rules:

  - Allow All Traffic: CIDR block 10.0.0.0/0

    This rule allows inbound and outbound traffic from all IPs within this range.

  - Allow SSH (Port 22): CIDR block 10.0.0.0/0

    This rule allows SSH access to the instances.

EC2 Instance Configuration

- Image ID: ami-0cafb660076a6e32

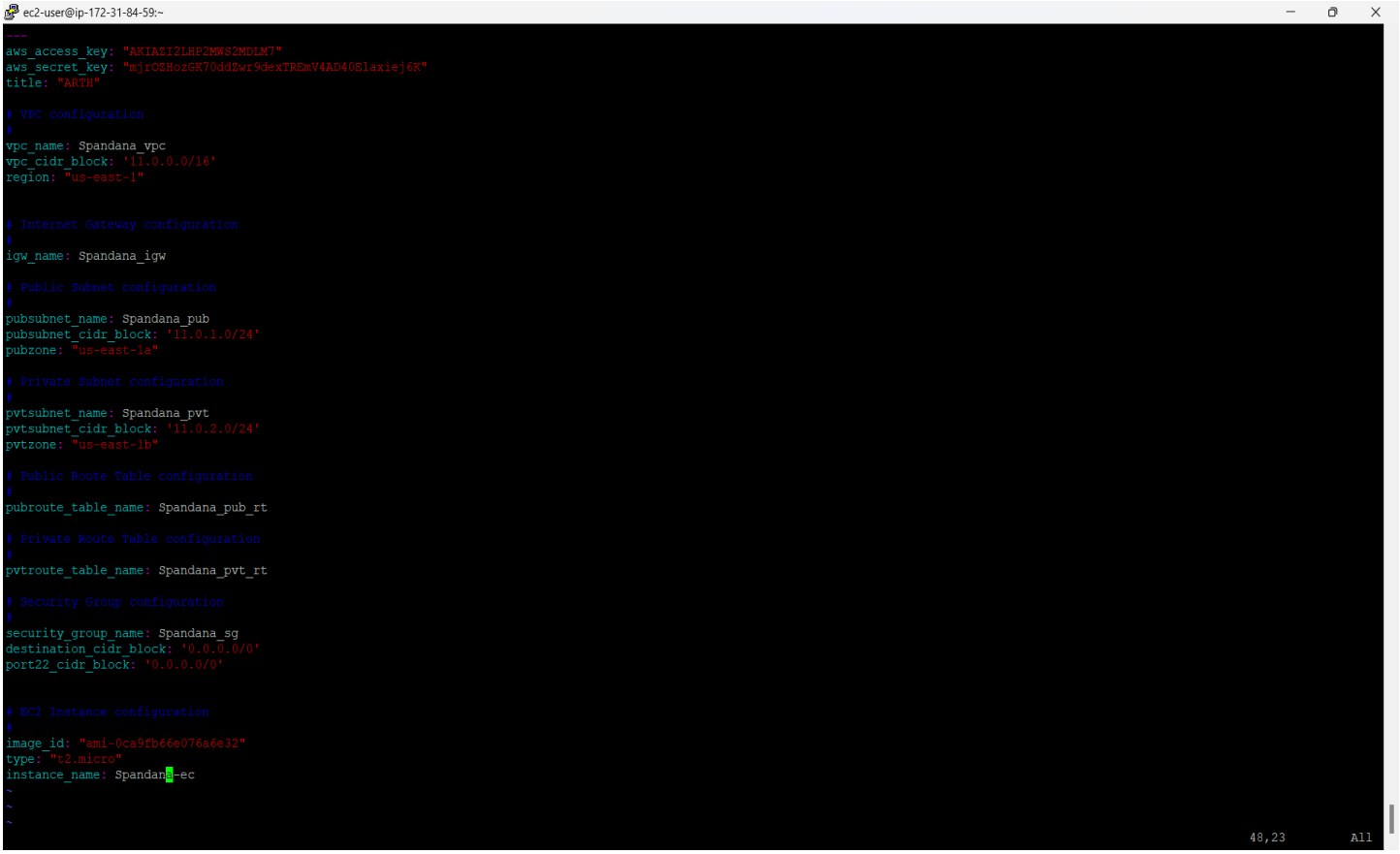
  The Amazon Machine Image (AMI) ID specifies the operating system and configuration for the instance.

- Instance Name: Spandana_ec2

  The instance will be tagged with this name.

- Instance Type:

  Not explicitly provided but commonly a type like t2.micro for testing**.**



I have run an **ansible-playbook vpc.yaml –syntax-check** for checking an syntax

Ansible-playbook:

This is the primary command to run an Ansible playbook.

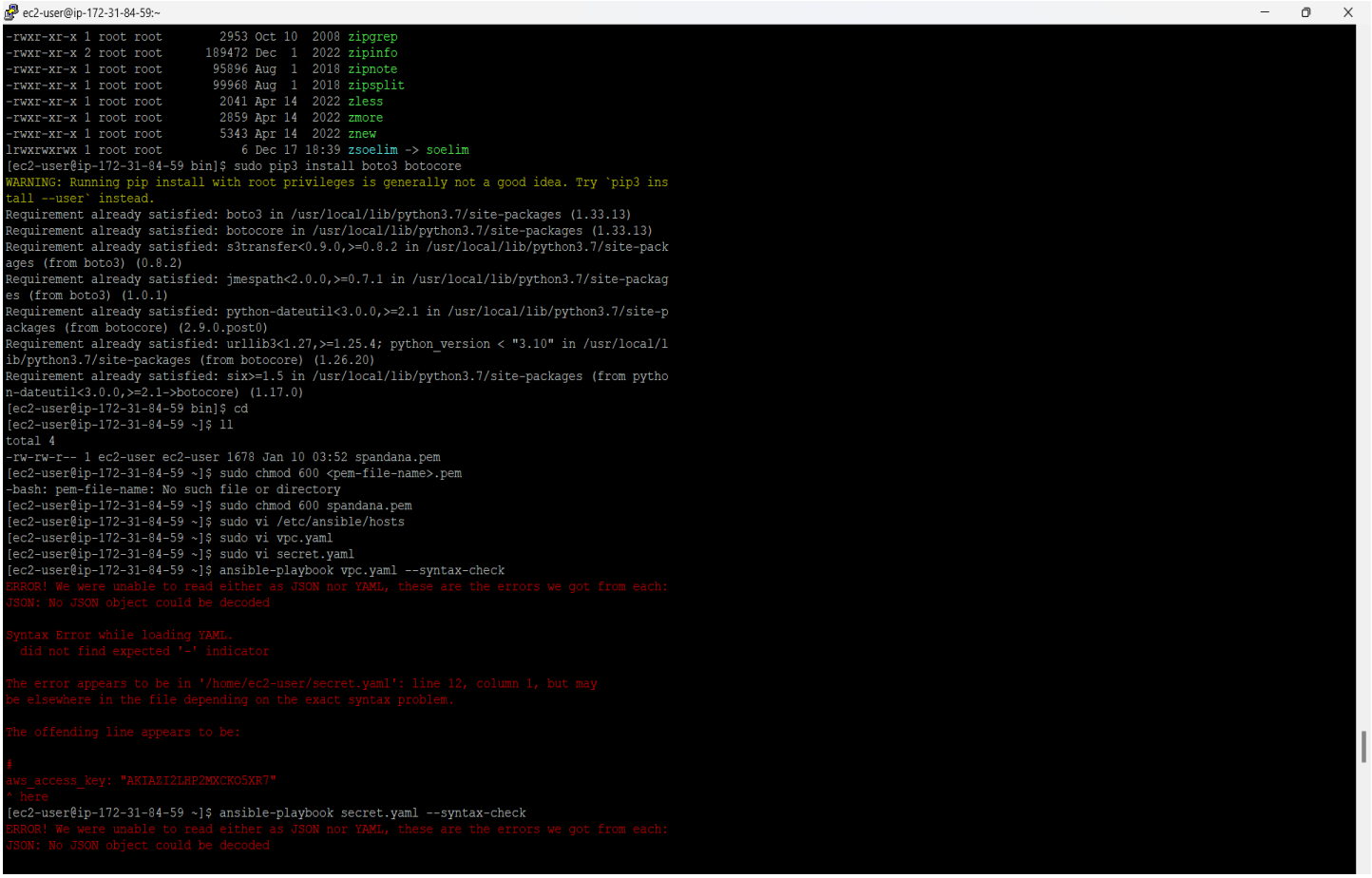
vpc.yaml:

This specifies the playbook file (vpc.yaml) you want to validate or execute. It contains tasks for configuring your infrastructure, such as creating a VPC and related components.

–syntax-check:

This option ensures that the playbook's syntax is correct without running any tasks or making changes. It helps to catch:

- YAML formatting errors
- Missing or misplaced parameters
- Syntax mistakes in the Ansible modules or tasks



Run **ansible-playbook vpc.yaml** Reading the Playbook Ansible reads the vpc.yaml file to understand. The hosts to run tasks on (e.g., localhost or remote servers). The tasks to execute and their sequence. Variables and configurations

ansible-playbook:

This is the Ansible CLI command to run playbooks. A playbook is a YAML file containing tasks, roles, and modules that automate IT processes like provisioning infrastructure, managing configurations, and deploying applications.

vpc.yaml:

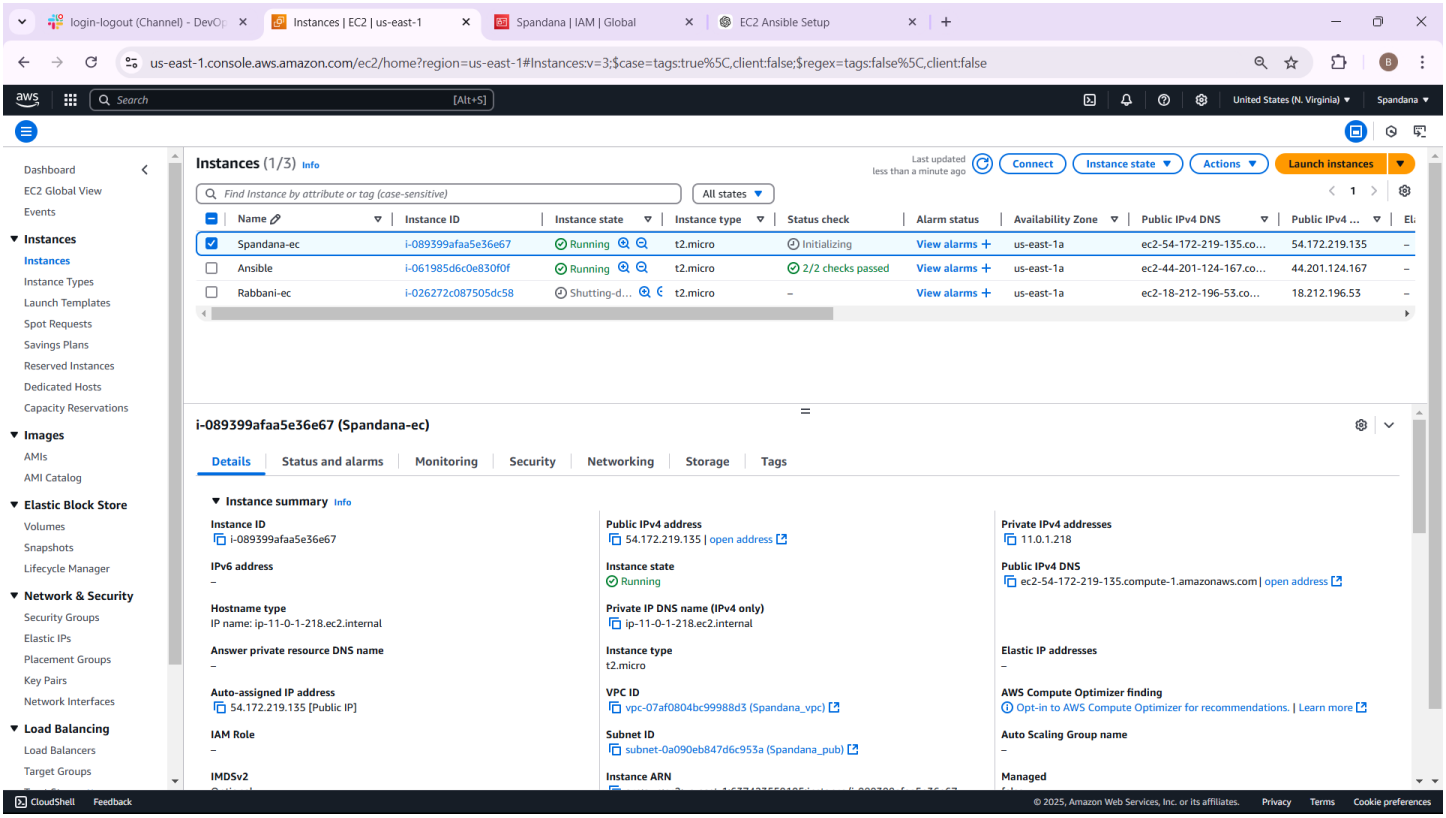
This is the name of the playbook file. In this case, it likely contains tasks to create and configure an AWS Virtual Private Cloud (VPC) and related components (e.g., subnets, route tables, security groups, and instances).

```
ec2-user@ip-172-31-84-59:~
[ec2-user@ip-172-31-84-59 ~]$ sudo vi secret.yaml
[ec2-user@ip-172-31-84-59 ~]$ ansible-playbook secret.yaml --syntax-check
ERROR! A playbook must be a list of plays, got a <class 'ansible.parsing.yaml.objects.AnsibleMapping'> instead

The error appears to be in '/home/ec2-user/secret.yaml': line 2, column 1, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

---
aws_access_key: "AKIA2I2LHP2MWS2MDLM7"
^ here
[ec2-user@ip-172-31-84-59 ~]$ ansible-playbook vpc.yaml --syntax-check

playbook: vpc.yaml
[ec2-user@ip-172-31-84-59 ~]$ ^C
[ec2-user@ip-172-31-84-59 ~]$ ansible-playbook vpc.yaml

PLAY [localhost] ***************************************************************************************************************************************************************************************

TASK [ec2_vpc_net] *************************************************************************************************************************************************************************************
The authenticity of host '172.31.84.59 (172.31.84.59)' can't be established.
ECDSA key fingerprint is SHA256:hNmV4gYrEPjkV9Exu5agMEGg6fzfY7fXmJM+oUqunWM.
ECDSA key fingerprint is MD5:6c:2f:ae:14:04:20:20:20:63:b2:a6:08:3e:57:4c:19.
Are you sure you want to continue connecting (yes/no)? yes
changed: [127.0.0.1]

TASK [ec2_vpc_igw] *************************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_subnet] **********************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_subnet] **********************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_route_table] *****************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_route_table] *****************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_group] ***************************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2] *********************************************************************************************************************************************************************************************
changed: [127.0.0.1]

PLAY RECAP *********************************************************************************************************************************************************************************************
127.0.0.1                  : ok=8    changed=8    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-84-59 ~]$
```

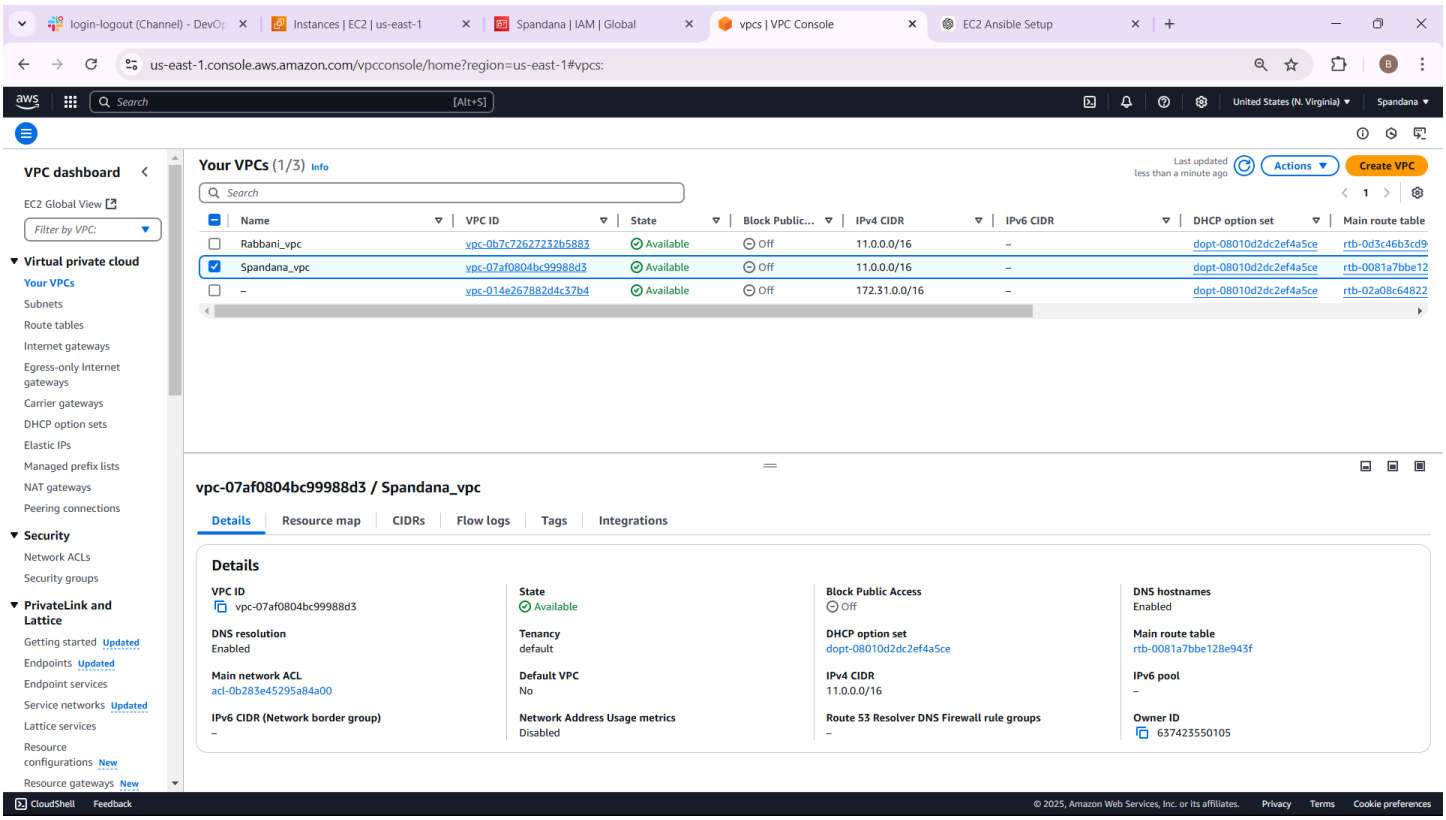
The output typically includes Play Recap: A summary of tasks and their status for each host as shown below.

```
ec2-user@ip-172-31-84-59:~
changed: [127.0.0.1]

TASK [ec2_vpc_route_table] *****************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_route_table] *****************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_group] ***************************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2] *********************************************************************************************************************************************************************************************
changed: [127.0.0.1]

PLAY RECAP *********************************************************************************************************************************************************************************************
127.0.0.1                  : ok=8    changed=8    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-84-59 ~]$ sudo vi vpc.yaml
[ec2-user@ip-172-31-84-59 ~]$ sudo vi secret.yaml
[ec2-user@ip-172-31-84-59 ~]$ sudo vi vpc.yaml
[ec2-user@ip-172-31-84-59 ~]$ ansible-playbook vpc.yaml

PLAY [localhost] ***************************************************************************************************************************************************************************************

TASK [ec2_vpc_net] *************************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_igw] *************************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_subnet] **********************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_subnet] **********************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_route_table] *****************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_route_table] *****************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_group] ***************************************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2] *********************************************************************************************************************************************************************************************
changed: [127.0.0.1]

PLAY RECAP *********************************************************************************************************************************************************************************************
127.0.0.1                  : ok=8    changed=8    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-84-59 ~]$
```

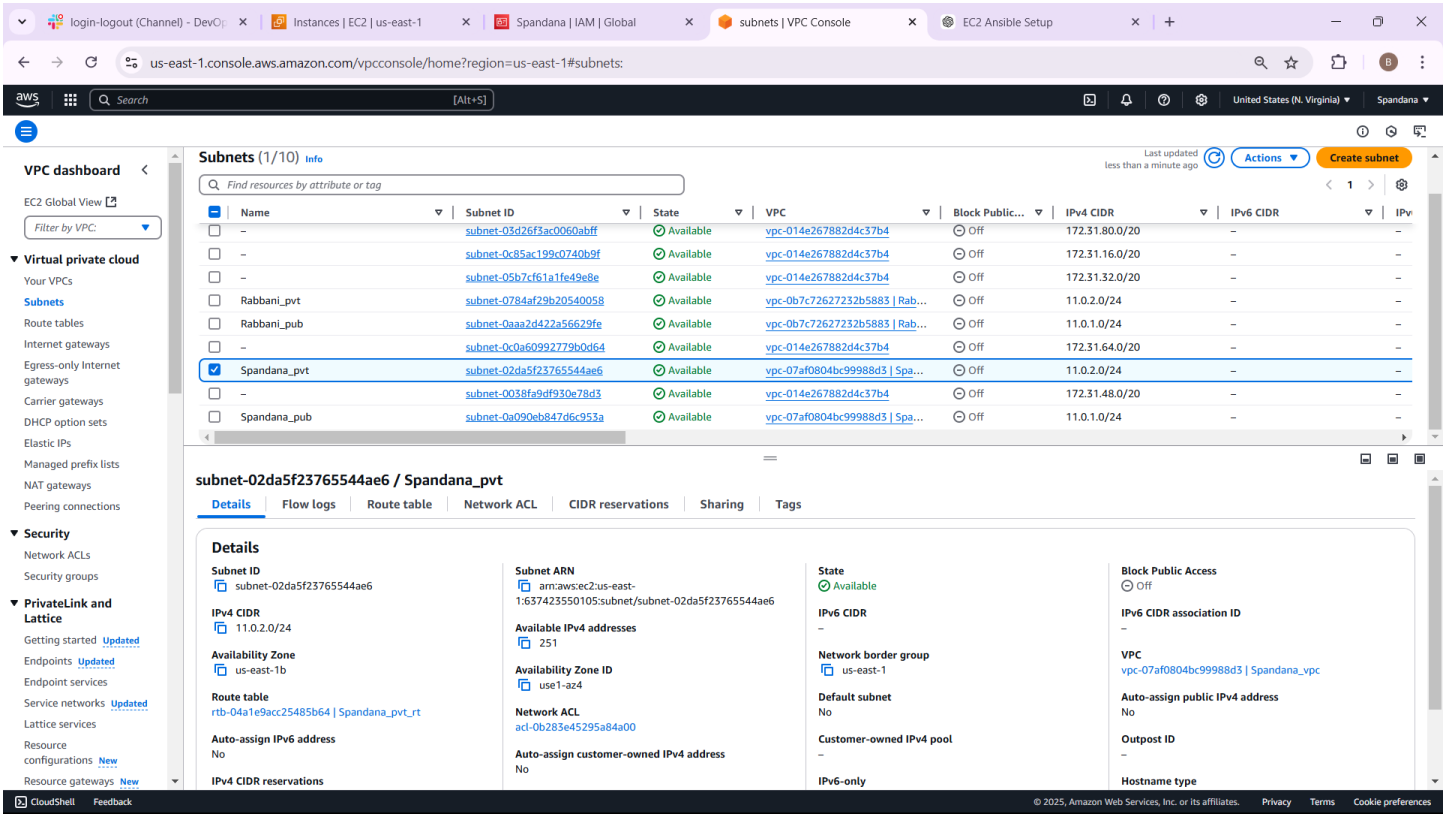
Ec2 instance has been created successfully as shown below.



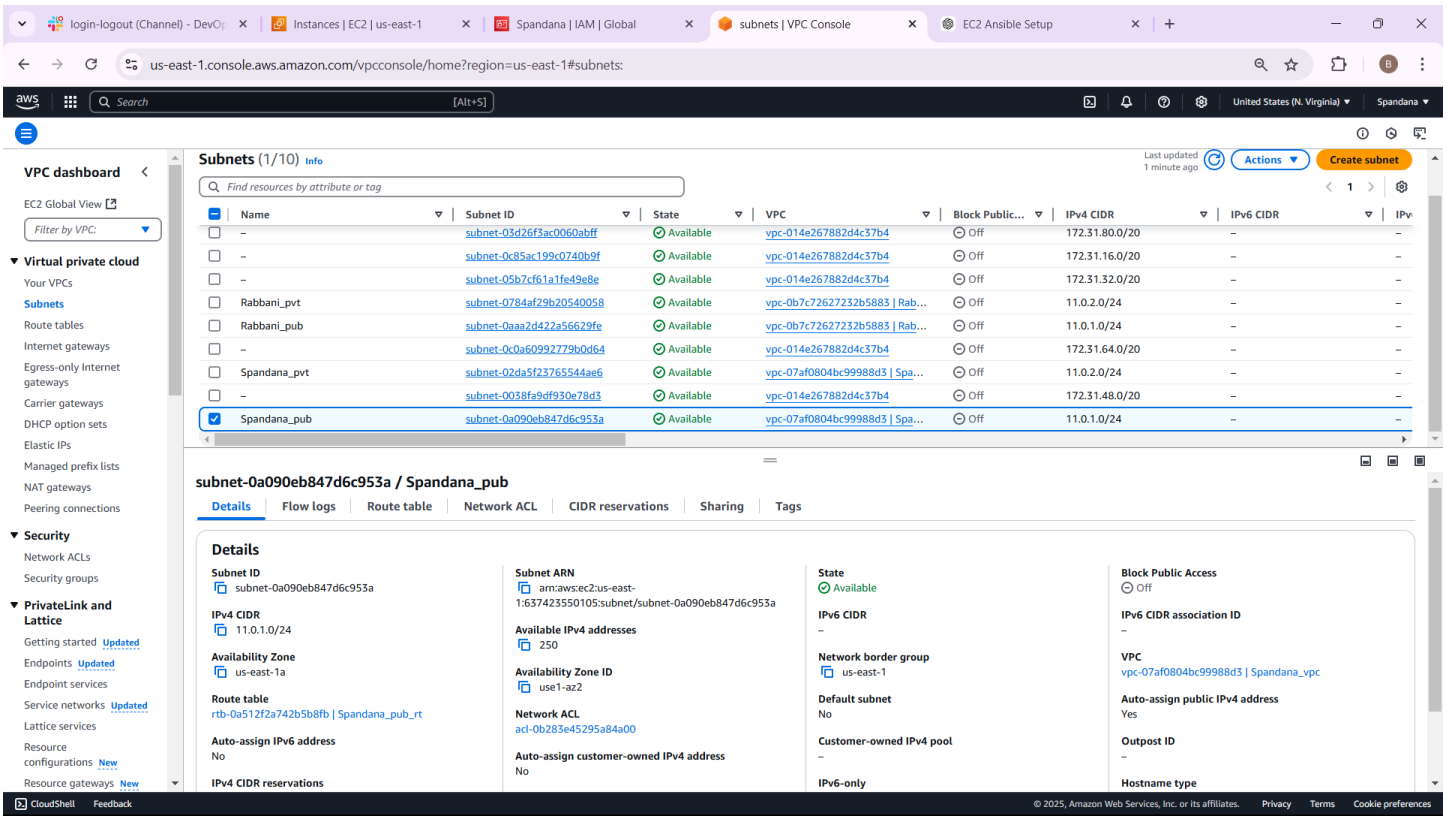
VPC has been created successfully as shown below.



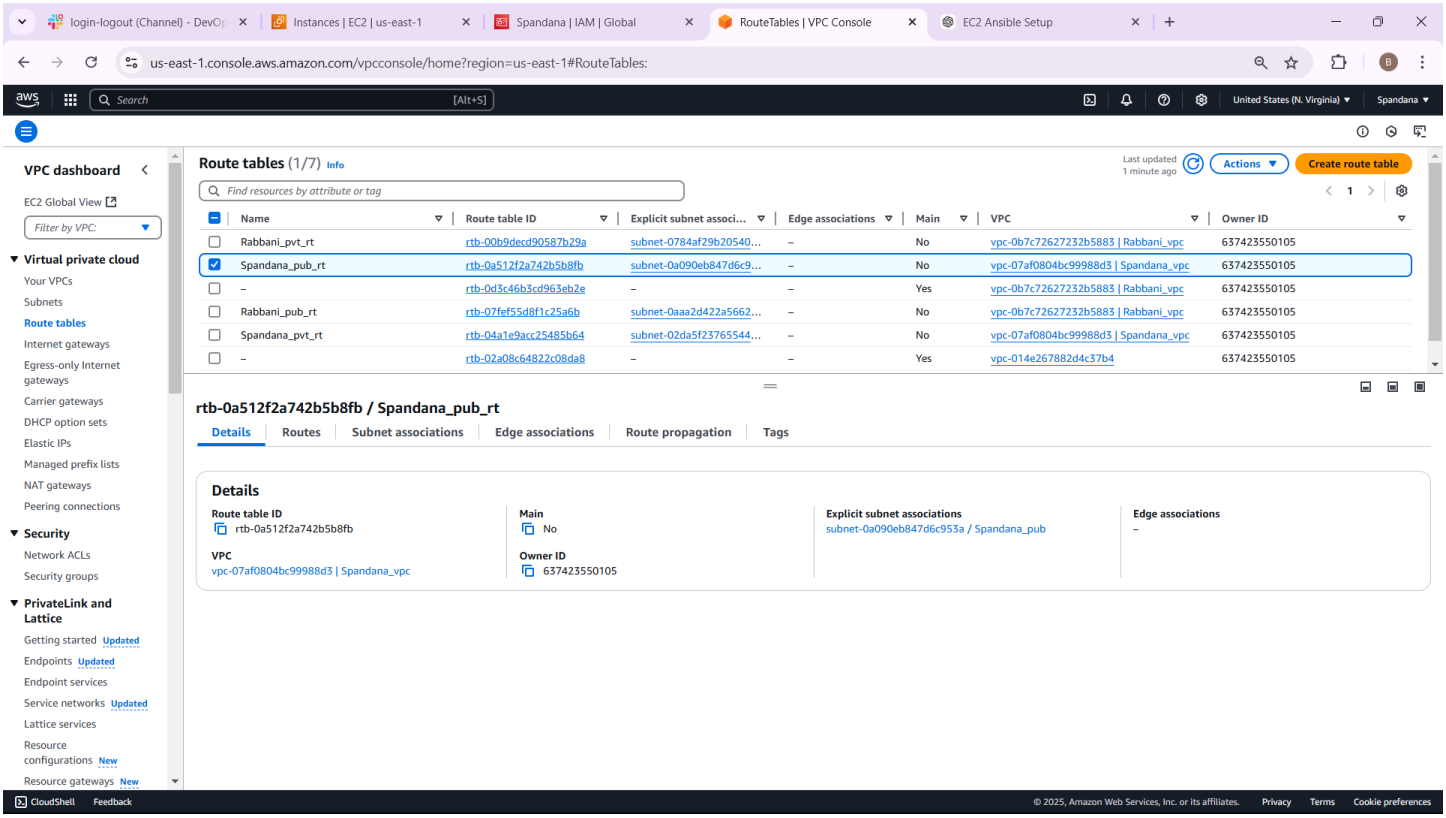
The below image is the private subnet of the vpc created successfully.



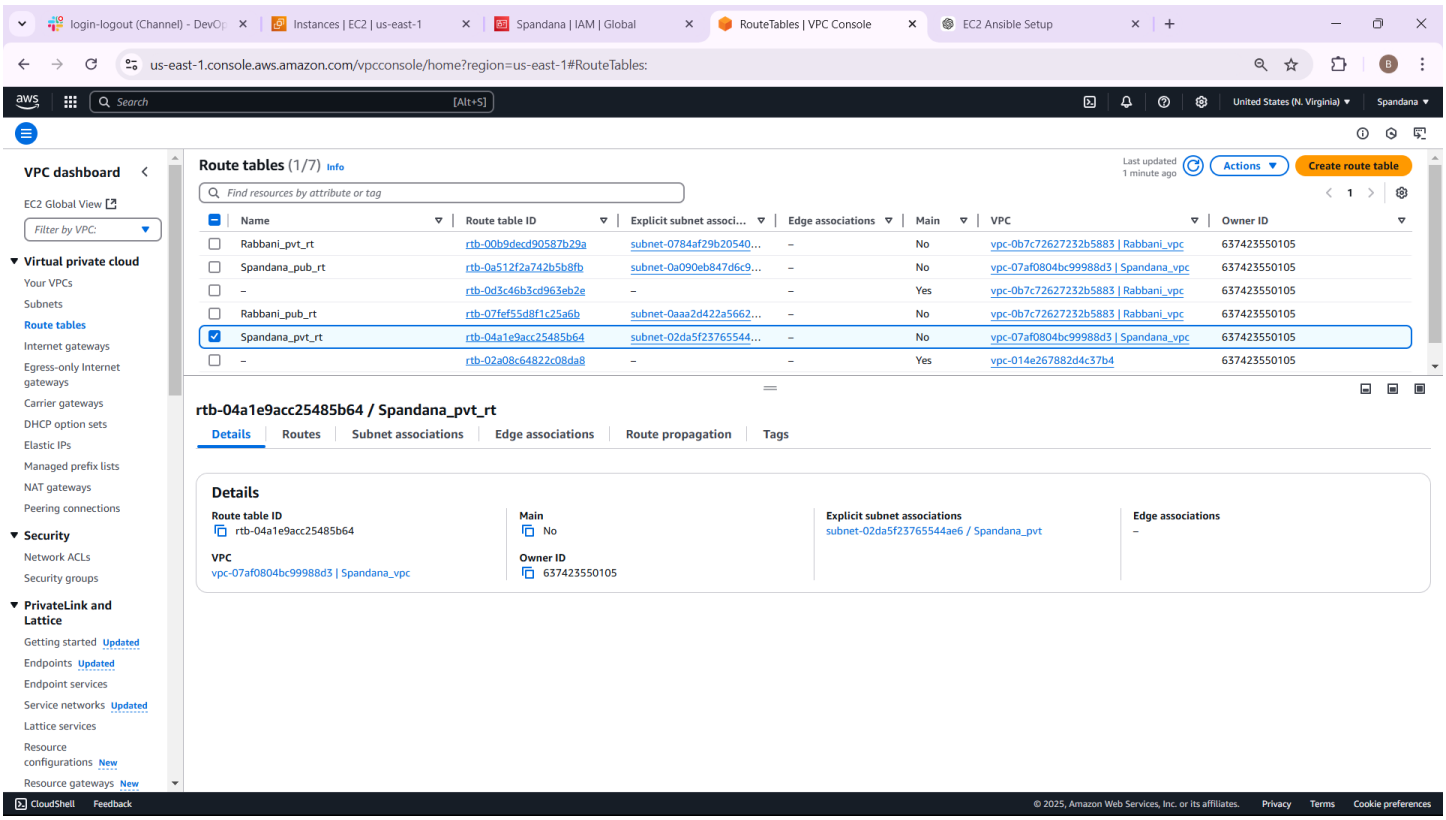
The below is the public subnet of the vpc.



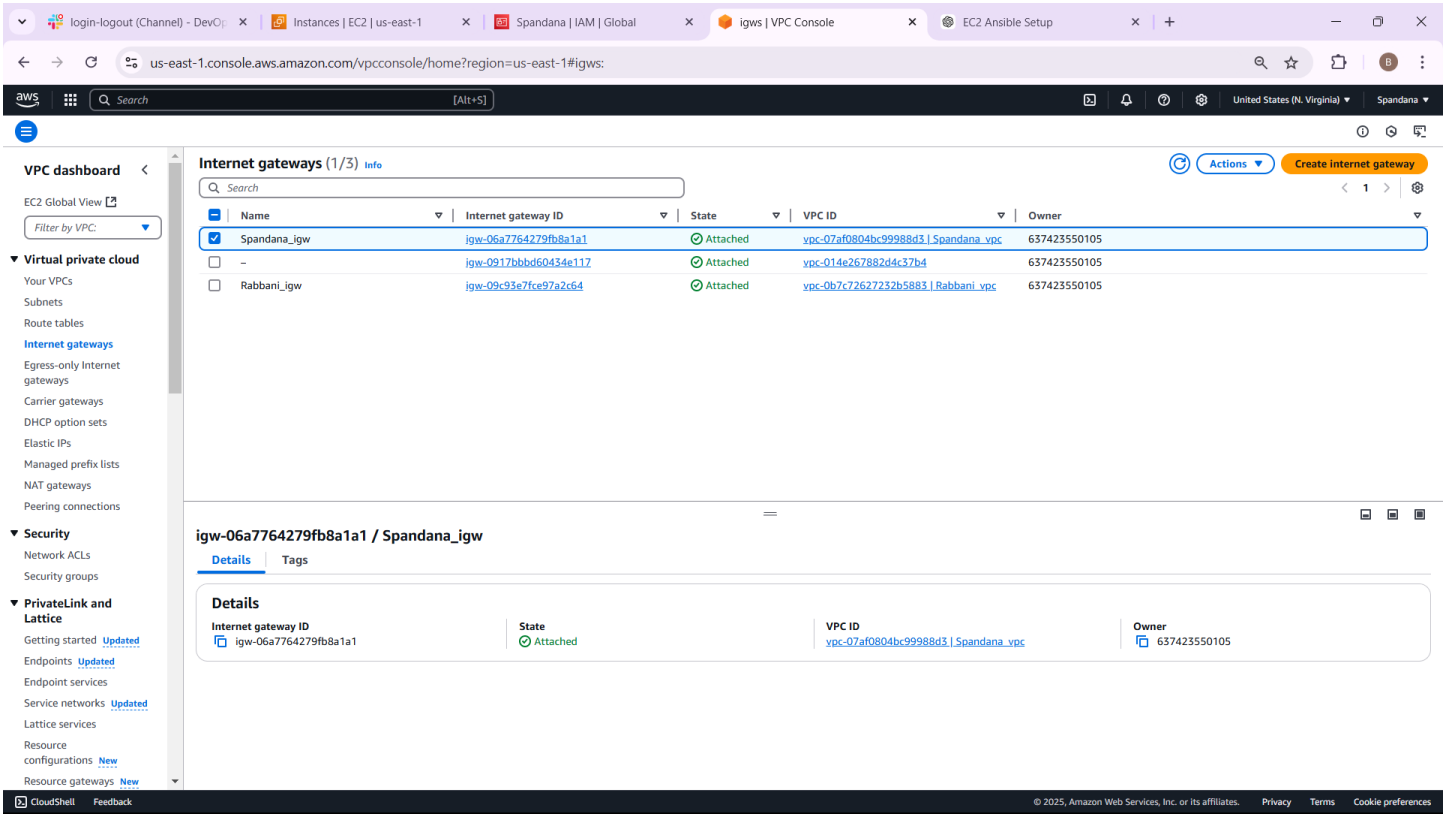
Successfully public subnet has been created as shown below.



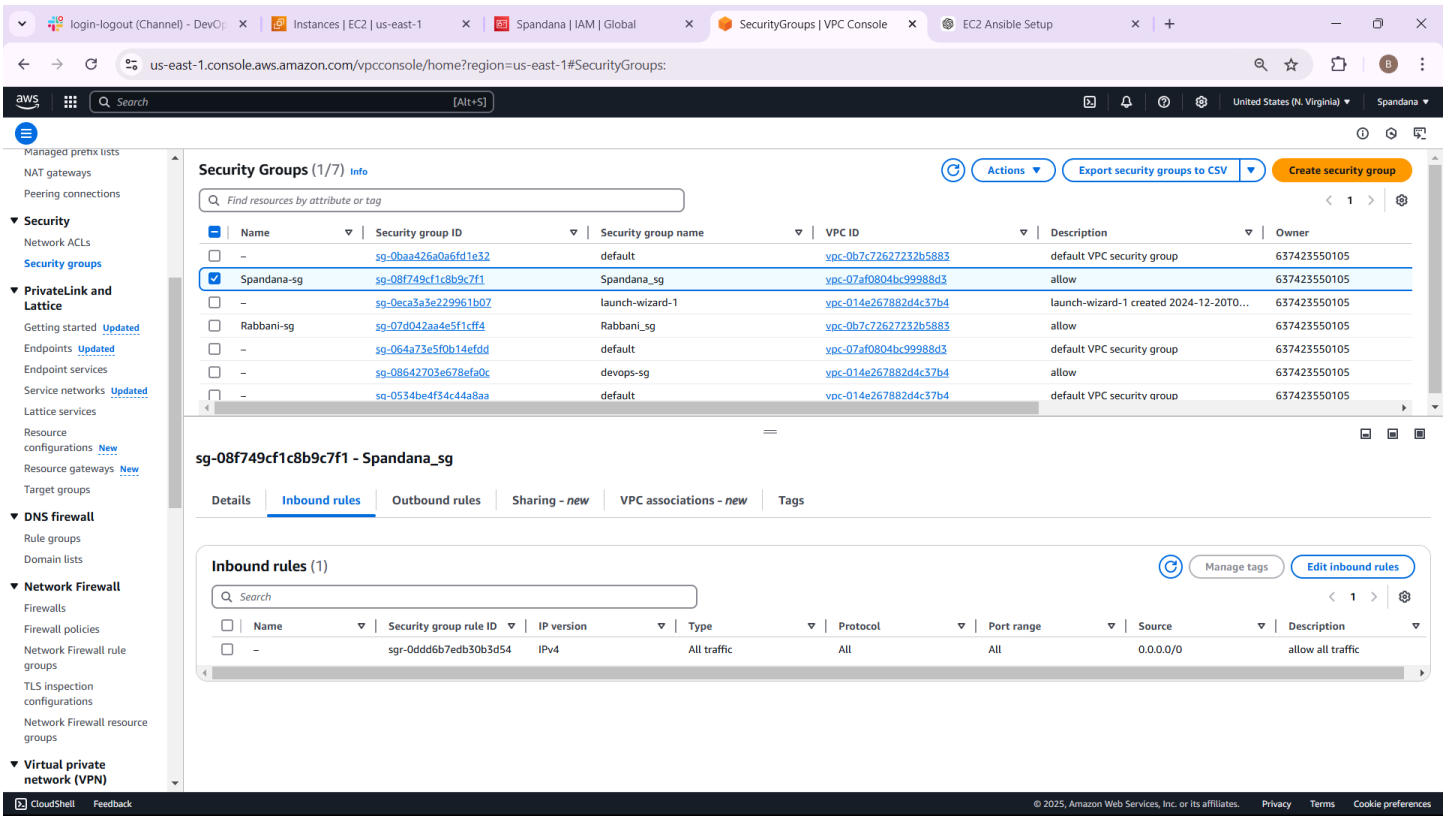
Successfully private subnet has been created successfully as shown below.



Successfully internet gateway has been created as shown below.



Security groups has been created successfully has shown below.



Ansible_Playbook script without vairables

I have created a VPC.yaml file in that I have wroat an script explained below.

```
-rw------- 1 ec2-user ec2-user 1678 Jan 10 03:52 spandana.pem
[ec2-user@ip-172-31-84-59 ~]$ sudo vim VPC.yaml
[ec2-user@ip-172-31-84-59 ~]$ ll
total 8
-rw------- 1 ec2-user ec2-user 1678 Jan 10 03:52 spandana.pem
-rw-r--r-- 1 root     root     3561 Jan 10 05:00 VPC.yaml
[ec2-user@ip-172-31-84-59 ~]$
```

Explanation of playbook:

VPC Creation script explanation :

ec2_vpc_net: This defines a task that interacts with AWS EC2 to create a VPC.

aws_access_key: The AWS Access Key is used to authenticate your connection with AWS services.

aws_secret_key: The AWS Secret Key, used along with the Access Key, is also part of the authentication for AWS.

cidr_block: This defines the IP range for the VPC using CIDR notation. Here, 10.0.0.0/16 means the VPC can have IPs from 10.0.0.1 to 10.0.255.254.

name: The name tag of the VPC, which will be Puppy-vpc.

region: The AWS region where the VPC will be created (us-east-1).

state: present This ensures the VPC will be created. If it already exists, no change will occur.

register: vpc_result The result of this task (e.g., the VPC ID) is saved in the variable vpc_result, which can be used later.

Internet Gateway creation script explanation :

- ec2_vpc_igw: This task creates an Internet Gateway (IGW) for the VPC.
- aws_access_key: The Access Key for AWS authentication.
- aws_secret_key: The Secret Key for AWS authentication.
- vpc_id: "{{ vpc_result.vpc.id }}" This dynamically references the ID of the VPC created in the previous task (vpc_result holds this ID).
- region: The region where the Internet Gateway will be created (us-east-1).
- state: present Ensures the Internet Gateway is created. If it already exists, nothing will change.

 tags: Assigns the tag Name: Puppyligw to the Internet Gateway.

 register: igw_result Saves the details of the created Internet Gateway (like its ID) in the igw_result variable

Creation of Public subnet script :

 ec2_vpc_subnet: This task creates a Subnet inside the VPC.

 aws_access_key: The Access Key for AWS authentication.

 aws_secret_key: The Secret Key for AWS authentication.

 vpc_id: "{{ vpc_result.vpc.id }}" References the VPC ID created earlier.

 region: us-east-1 The region for the subnet (us-east-1).

 az: us-east-1a The Availability Zone in which to create the subnet (us-east-1a).

 state: present Ensures the subnet is created.

 cidr: 10.0.0.0/20 The CIDR block for the subnet (10.0.0.0/20 means 4096 IP addresses, with the first 4 bits fixed).

 map_public: yes This indicates the subnet will automatically assign public IPs to instances (making it a public subnet).

 resource_tags: Adds a tag with the name Puppy-pub to the subnet.

```
ec2-user@ip-172-31-84-59:~
---
- hosts: localhost
  become: yes
  tasks:
    - ec2_vpc_net:
        aws_access_key: AKIAZI2LHP2MWS2MDLM7
        aws_secret_key: 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
        cidr_block: 10.0.0.0/16
        name: Puppy-vpc
        region: us-east-1
        state: present  # to delete Vpc then replace absent instead of present
      register: vpc_result

#### Internet gateway Creation ######

    - ec2_vpc_igw:
        aws_access_key: AKIAZI2LHP2MWS2MDLM7
        aws_secret_key: mjrOZHozGK70ddZwr9dexTREmV4AD40Elaxiej6K
        vpc_id: "{{ vpc_result.vpc.id }}"
        region: us-east-1
        state: present
        tags:
          Name: Puppyigw
      register: igw_result

 ## create an vpc pub subnet


    - ec2_vpc_subnet:
        aws_access_key: AKIAZI2LHP2MWS2MDLM7
        aws_secret_key: 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
        vpc_id: "{{ vpc_result.vpc.id }}"
        region: us-east-1
        az: us-east-1a      # az is the availability zone
        state: present
        cidr: 10.0.0.0/20
        # enable public ip
        map_public: yes
        resource_tags:
          Name: Puppy-pub
      register: pubsubnet_result

    ## create an vpc pvt subnet


    - ec2_vpc_subnet:
        aws_access_key: AKIAZI2LHP2MWS2MDLM7
        aws_secret_key: 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
        vpc_id: "{{ vpc_result.vpc.id }}"
        region: us-east-1
        az: us-east-1b      # az is the availability zone
-- INSERT --                                                    23,22         Top
```

Creation of private subnet :

- ec2_vpc_subnet: This task creates another subnet, but this one will be private.
- aws_secret_key: The Secret Key for AWS authentication.
- vpc_id: "{{ vpc_result.vpc.id }}" References the VPC ID created earlier.
- region: us-east-1 The region where the subnet will be created (us-east-1).
- az: us-east-1b The Availability Zone where this private subnet will be created (us-east-1b).
- state: present Ensures the subnet is created.
- cidr: 10.0.1.0/20 The CIDR block for this subnet (10.0.1.0/20), providing a range of IPs.
- map_public: no This means no public IP addresses will be assigned to instances in this subnet (making it a private subnet).
- resource_tags: Adds a tag with the name Puppy-pvt to the subnet.
- register: pvtsubnet_result The details of the private subnet are saved in pvtsubnet_result.

Creation of Public route table script:

- ec2_vpc_route_table: Creates a route table for a public subnet to route traffic, including to the internet through the internet gateway (IGW).
- aws_access_key: The AWS access key for authentication.
- aws_secret_key: The AWS secret key for authentication.
- vpc_id: "{{ vpc_result.vpc.id }}" References the VPC where the route table will be created.
- region: us-east-1 The region for the route table (us-east-1).
- state: present Ensures that the route table is created.
- tags: Tags the route table with the name Puppy-pub-rt.
- subnets: Specifies the subnet ID (pubsubnet_result.subnet.id) where this route table will be used. It associates the public subnet with the route table.
- routes: Defines the routes for the table.
    - dest: 0.0.0.0/0: This route means all traffic (0.0.0.0/0) will be forwarded to the internet gateway (gateway_id: "{{ igw_result.gateway_id }}").
- register: public_route_table Saves the result of this task (e.g., route table ID) to the public_route_table variable.

Creation of private route table :

- ec2_vpc_route_table: Creates a route table for the private subnet, which can route traffic inside the VPC but not directly to the internet.
- aws_access_key: The AWS access key for authentication.
- aws_secret_key: The AWS secret key for authentication.
- vpc_id: "{{ vpc_result.vpc.id }}" References the VPC where the route table will be created.
- region: us-east-1 The region for the route table (us-east-1).
- state: present Ensures the route table is created.
- tags: Tags the route table with the name Puppy-pvt-rt.
- subnets: Specifies the private subnet ID (pvtsubnet_result.subnet.id) where this route table will be used.
- register: private_route_table Saves the result of this task (e.g., route table ID) to the private_route_table variable.

ec2_security_group: This task creates a security group within the VPC to control access to instances.

aws_access_key: The AWS access key for authentication.

aws_secret_key: The AWS secret key for authentication.

vpc_id: "{{ vpc_result.vpc.id }}" References the VPC where the security group will be created.

region: us-east-1 The region for the security group (us-east-1).

group_name: Puppy-sg The name for the security group (Puppy-sg).

description: A brief description of the security group.

ip_permissions: Specifies the rules for allowing traffic.

- ip_protocol: tcp: Allows TCP traffic.
- from_port: '22', to_port: '22': Allows SSH access on port 22.
- ip_ranges: cidr_ip: 0.0.0.0/0: Allows access from any IP address.

register: security_group_result Saves the details of the created security group (like its ID) to security_group_result.

EC2 Instance Creation:

- **Image**: ami-0ca9fb66e076a6e32 refers to the Amazon Machine Image (AMI) ID for the operating system or application you want to launch.
- **Instance Type**: t2.micro is the type of EC2 instance. It determines the resource (CPU, memory) available to the instance. t2.micro is a low-cost option suitable for small workloads.

- **Region**: us-east-1 specifies the AWS region where the EC2 instance will be launched.
- **Wait**: yes means the playbook will wait until the EC2 instance is fully launched before moving on to the next step.
- **Count**: Defines how many instances to launch (though this is not assigned in your playbook, it defaults to 1).
- **State**: present means the instance will be created if it doesn't exist already.
- **VPC Subnet ID**: Specifies the subnet in which the instance should be launched. The subnet ID is dynamically retrieved from pubsubnet_result.subnet.id.
- **Security Group ID**: The security group to be applied to the instance. The security group ID is dynamically retrieved from security_group_results.group_id.
- **AWS Access & Secret Keys**: These are the credentials used to authenticate to AWS and perform the actions in your playbook. They're typically stored securely in a separate file, but in this case, they are written in the playbook directly (though this is not recommended for security reasons).
- **Instance Tags**: Tags are key-value pairs that help you organize and identify your resources. In this case, the tag "Name": "Puppy-Ec" is applied to the instance.

Then after all the script written in vpc.yaml and save the file through :wq! And run the below command to run the yaml file.

The command **ansible-playbook VPC.yaml --syntax-check** is used to check the syntax of the VPC.yaml playbook without actually running it.

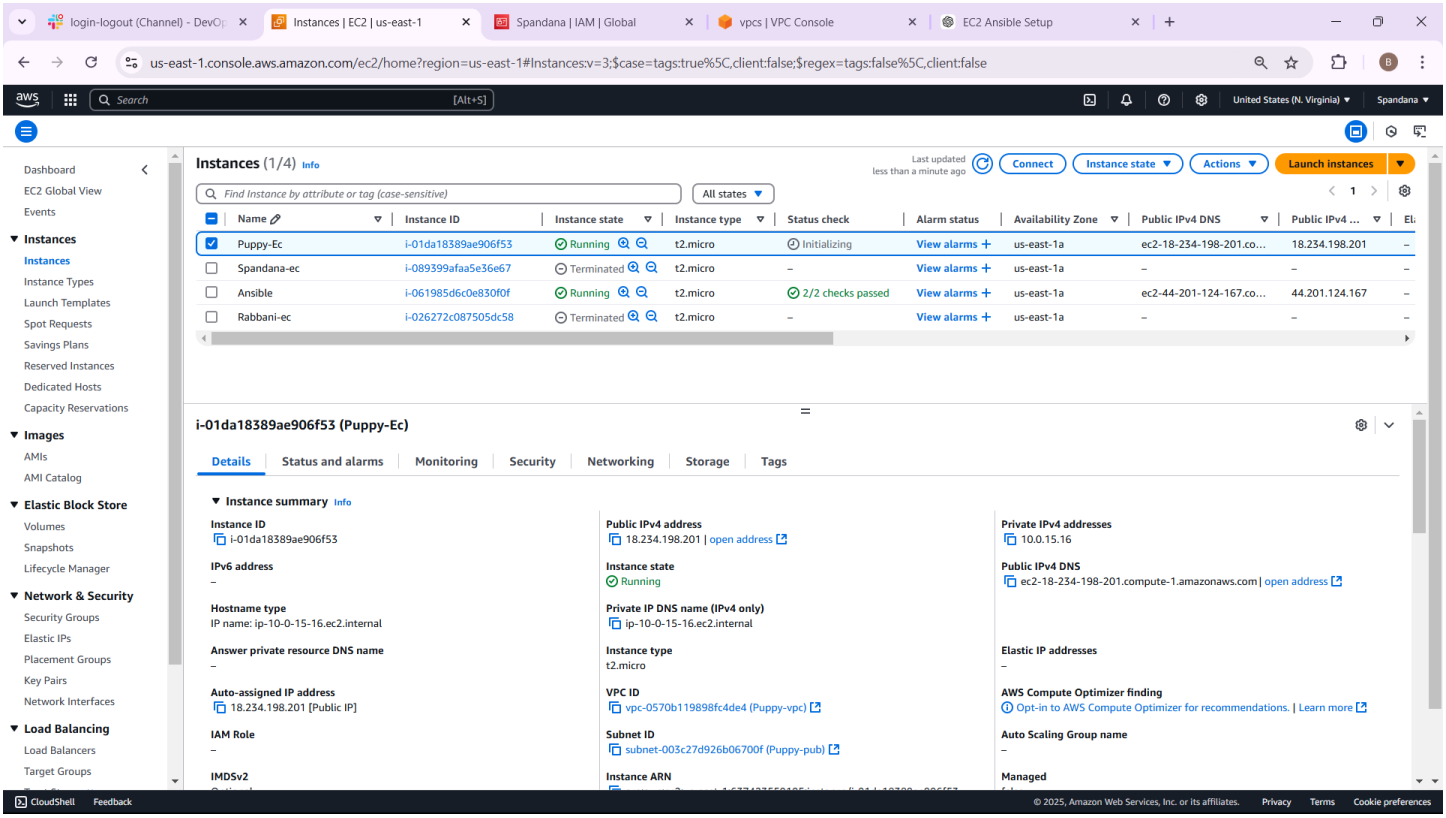
1. ansible-playbook: This is the command used to run Ansible playbooks.
2. VPC.yaml: Specifies the playbook file (VPC.yaml) to be checked.
3. --syntax-check: A flag that tells Ansible to perform a syntax validation on the playbook to ensure there are no errors in the code.

The command **ansible-playbook VPC.yaml** is used to execute the Ansible playbook defined in the VPC.yaml file. Here's what happens when you run it:

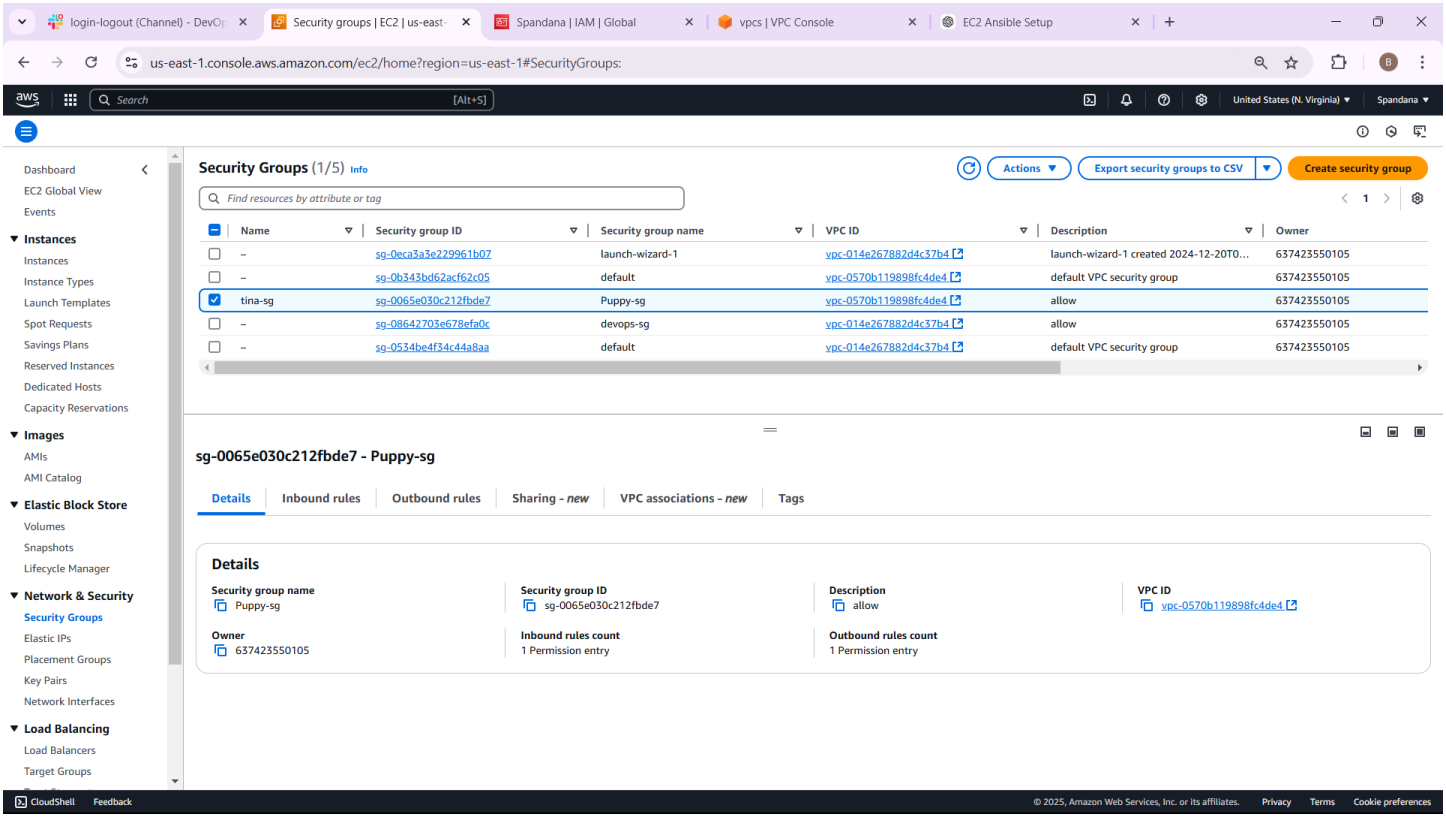
1. **ansible-playbook**: This is the command used to run Ansible playbooks, which are YAML files containing automation tasks.
2. **VPC.yaml**: This is the specific playbook file to be executed. It contains tasks that will be performed on the target machines.

```
ec2-user@ip-172-31-84-59:~
total 12
-rw-r--r-- 1 root     root      909 Jan 10 04:38 secret.yaml
-rw------- 1 ec2-user ec2-user 1678 Jan 10 03:52 spandana.pem
-rw-r--r-- 1 root     root     3941 Jan 10 04:38 vpc.yaml
[ec2-user@ip-172-31-84-59 ~]$ rm -rf secret.yaml vpc.yaml
[ec2-user@ip-172-31-84-59 ~]$
[ec2-user@ip-172-31-84-59 ~]$ ll
total 4
-rw------- 1 ec2-user ec2-user 1678 Jan 10 03:52 spandana.pem
[ec2-user@ip-172-31-84-59 ~]$ sudo vim VPC.yaml
[ec2-user@ip-172-31-84-59 ~]$ ll
total 8
-rw------- 1 ec2-user ec2-user 1678 Jan 10 03:52 spandana.pem
-rw-r--r-- 1 root     root     3561 Jan 10 05:00 VPC.yaml
[ec2-user@ip-172-31-84-59 ~]$ ansible-playbook VPC.yaml --syntax-check

playbook: VPC.yaml
[ec2-user@ip-172-31-84-59 ~]$ ansible-playbook VPC.yaml

PLAY [localhost] ***************************************************************************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************************************************************
ok: [127.0.0.1]

TASK [ec2_vpc_net] *************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_igw] *************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_subnet] **********************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_subnet] **********************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_route_table] *****************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_vpc_route_table] *****************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2_group] ***************************************************************************************************************************************************************
changed: [127.0.0.1]

TASK [ec2] *********************************************************************************************************************************************************************
changed: [127.0.0.1]

PLAY RECAP *********************************************************************************************************************************************************************
127.0.0.1                  : ok=9    changed=8    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-84-59 ~]$
```

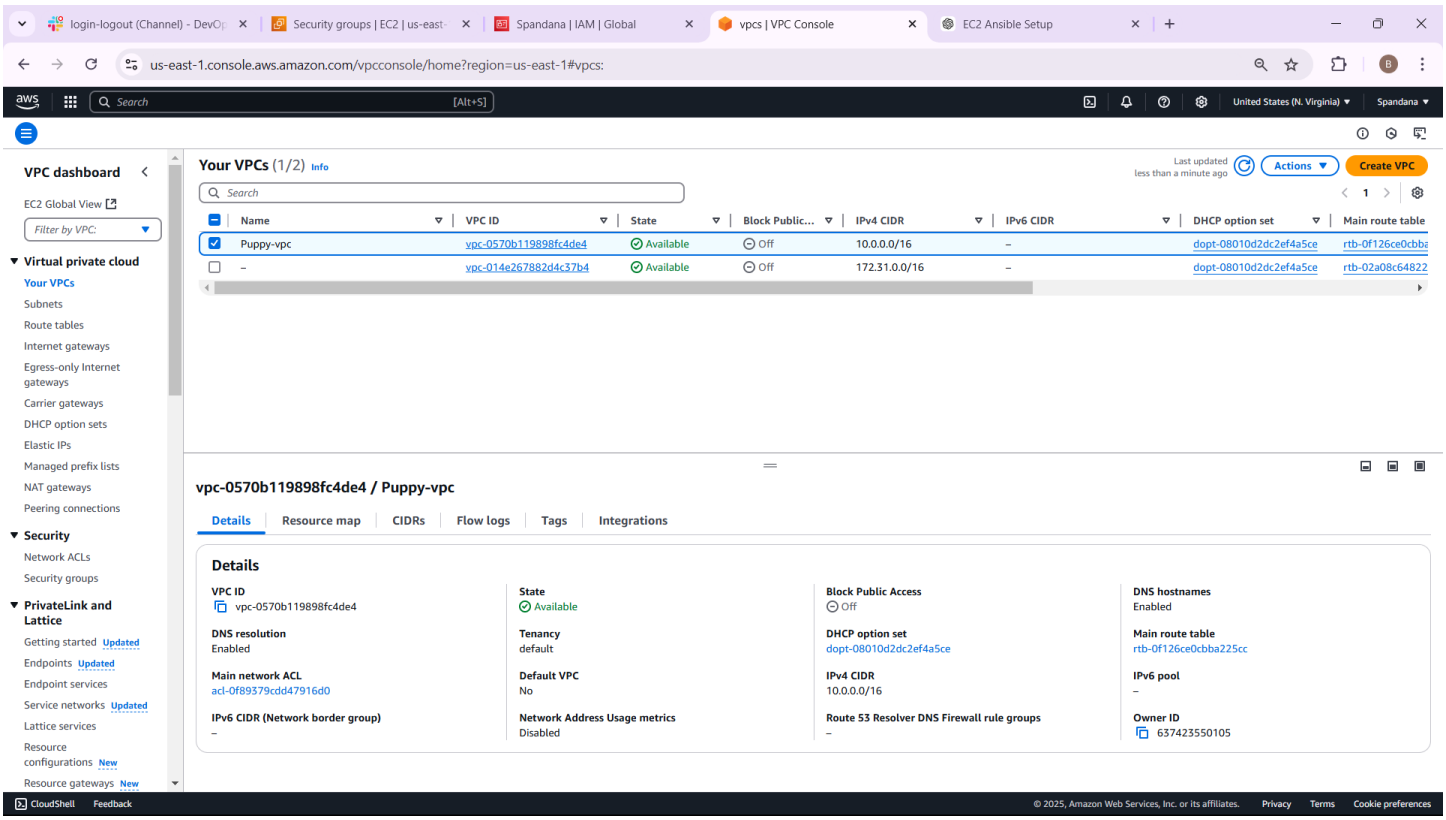
New Instance has been created successfully as shown below.



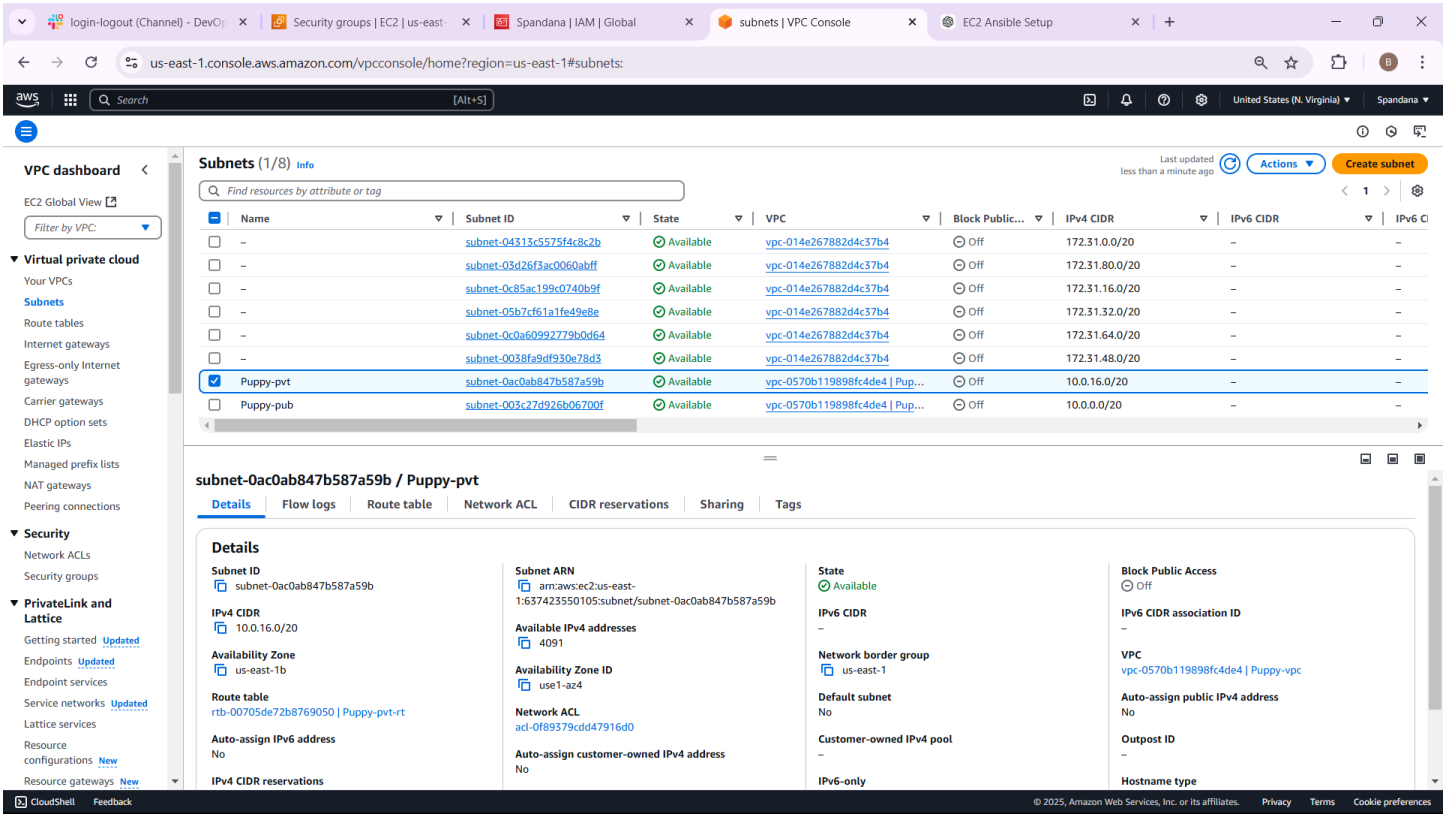
New security group has been created successfully.



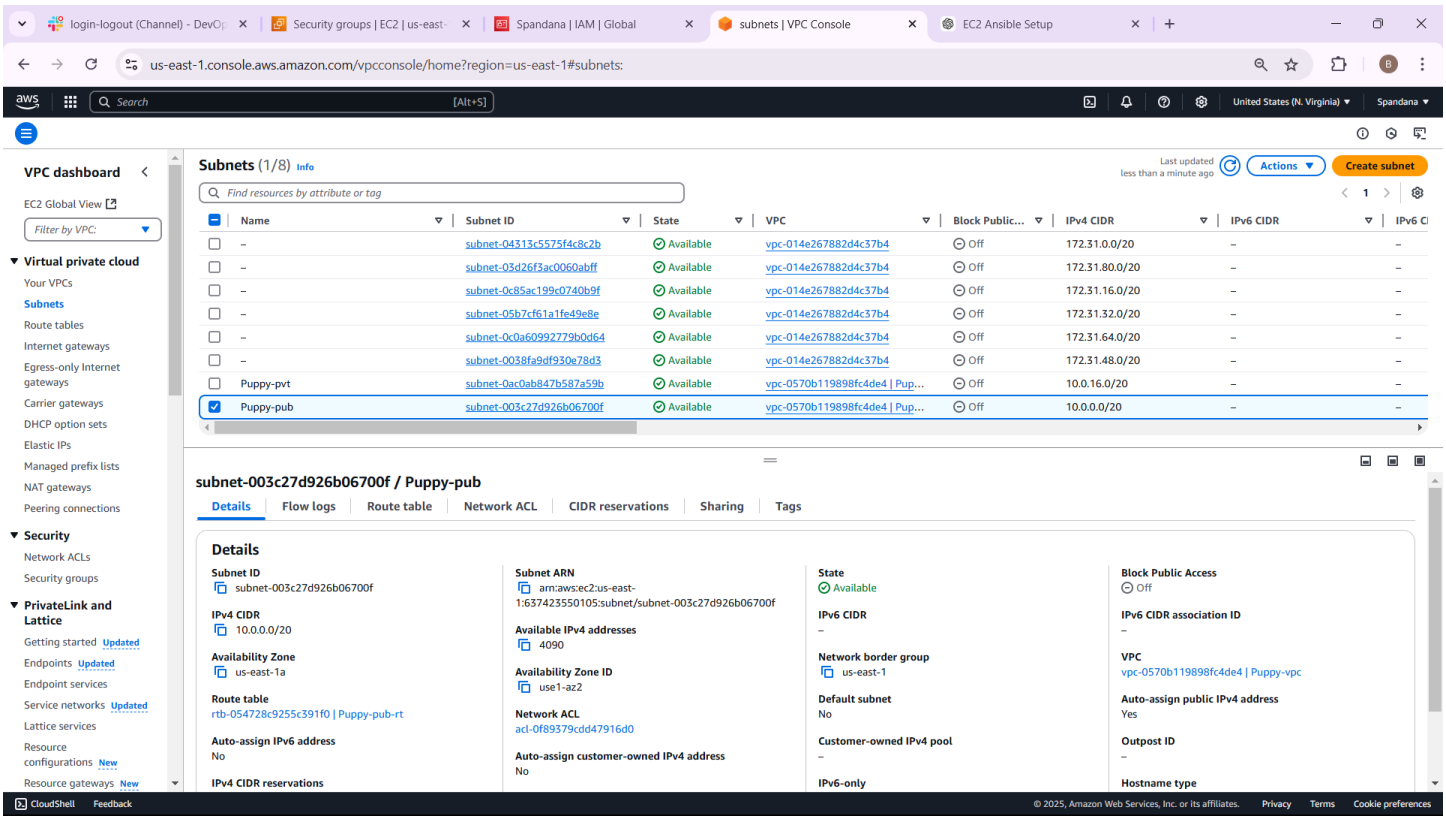
New vpc has been created successfully as shown below.



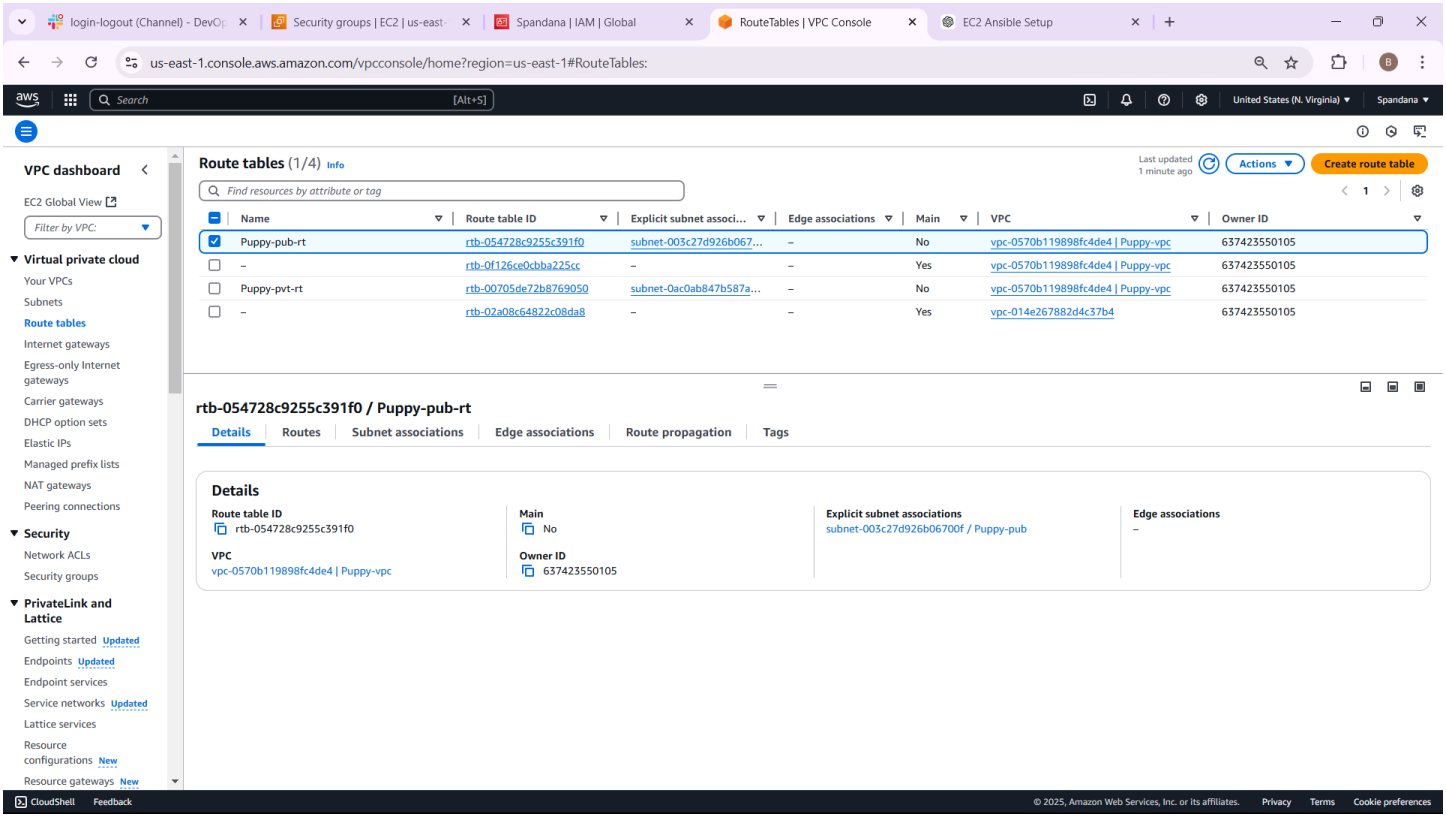
New Private subnet has been created successfully as shown below.



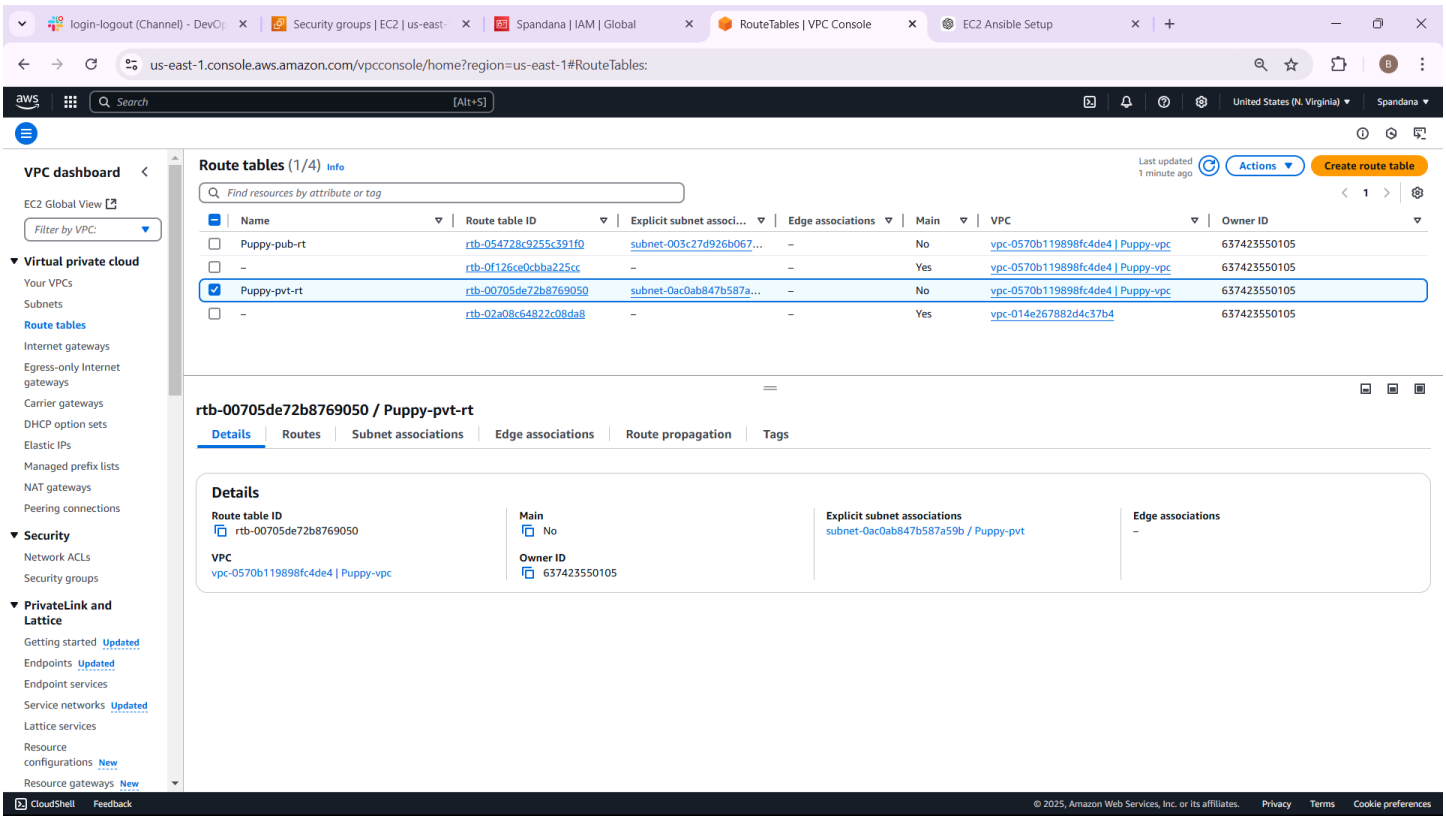
New public subnet has been created successfully has shown below.



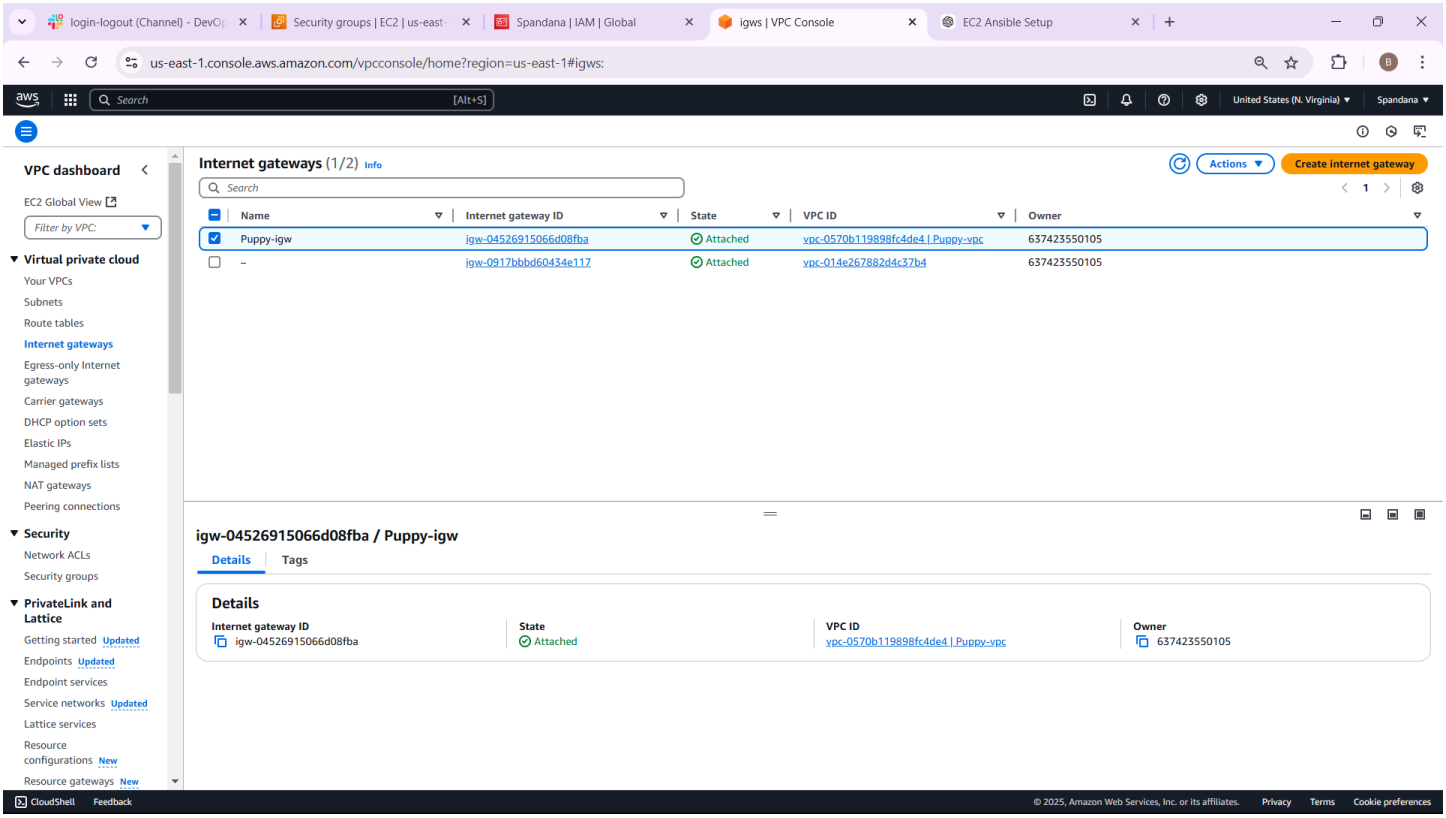
Public route table has been created successfully as shown below.



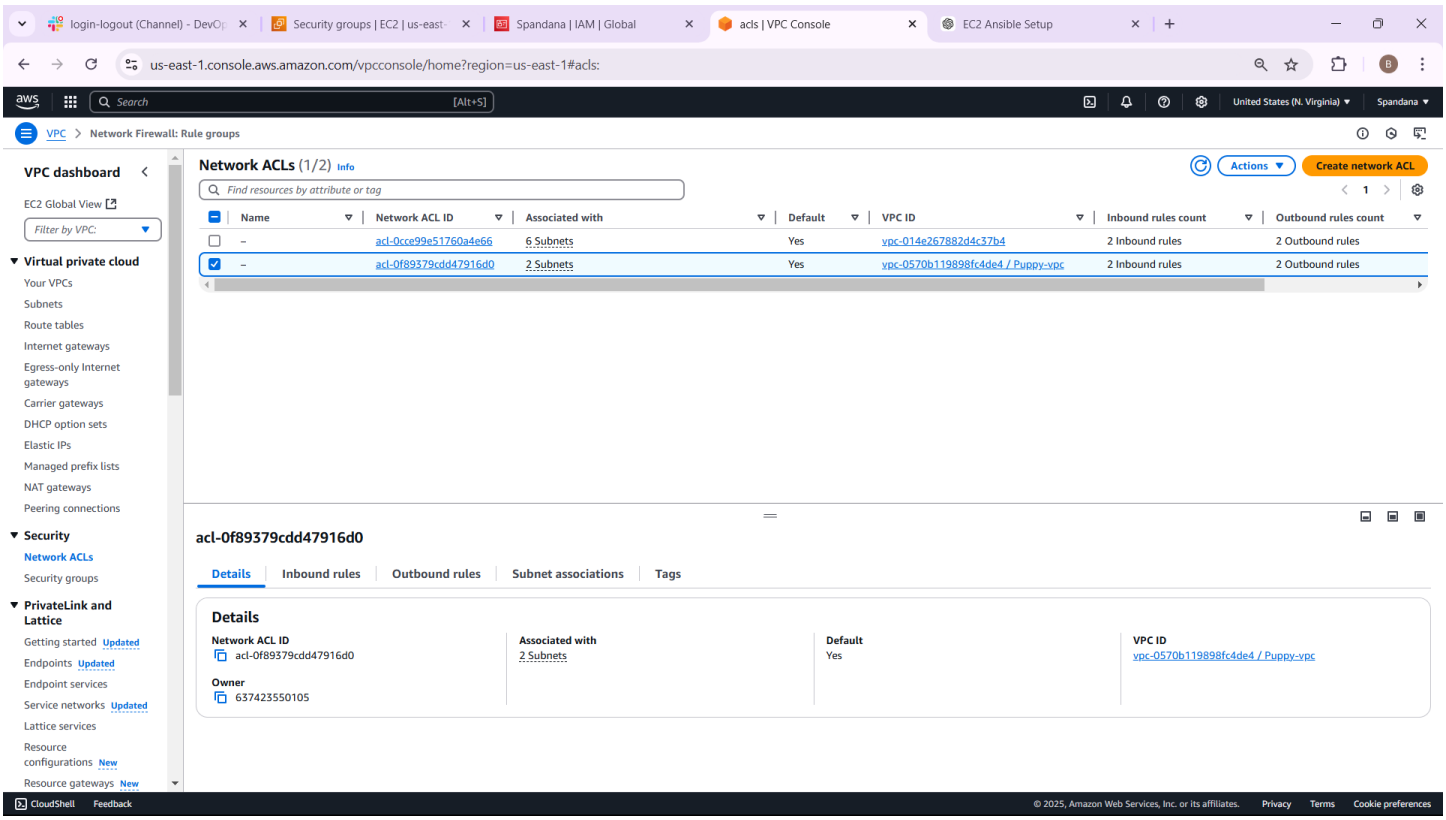
Private subnet has been created successfully as shown below.



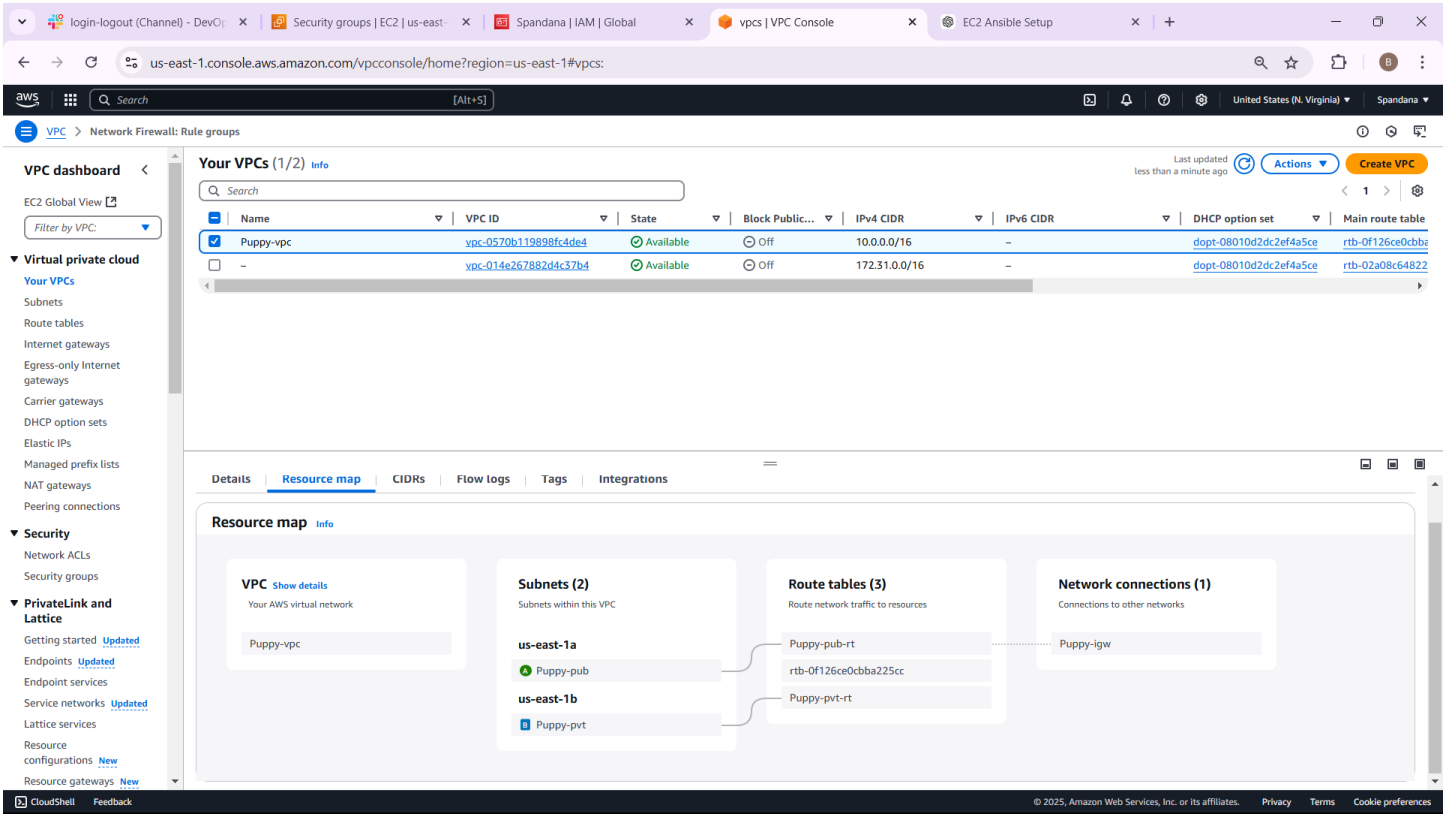
New internet gateway has been created successfully has shown below.



New network ACLs has been created successfully has shown below.



In the below image we can able to see that our resource map of an VPC.



Ansible_Playbook script with inline variables

I have created vpcinline.yaml file for inline variable script.

Explanation of vpcinline.yaml

1. hosts: localhost

- This indicates that the tasks defined in this playbook will be executed on the local machine (localhost**).**

2. become: yes

- This means that the playbook will run with elevated privileges (root or admin access) on the local machine.

3. gather_facts: false

- Disables the automatic gathering of system facts (e.g., OS details, memory, etc.) from the target hosts. It's set to false to avoid unnecessary overhead if it's not required.

4. vars (Variables)

- The vars section defines the variables that will be used later in the tasks. These variables store values that will be referenced throughout the playbook:

- VPC and Resources: Names and configurations of AWS resources like VPC (vpc_name), subnets (pubsubnet_name, pvtsubnet_name), route tables (pubroute_table_name, pvtroute_table_name), security group (security_group_name), etc.
- CIDR Blocks: These define the IP address ranges for VPC and subnets (vpc_cidr_block, pubsubnet_cidr_block, pvtsubnet_cidr_block).
- Region and Availability Zones: Specifies the AWS region (us-east-1) and the availability zones (pubzone, pvtzone).
- Instance Configuration: image_id, type, and instance_name are the EC2 instance configurations such as the AMI ID, instance type (t2.micro), and instance name (Baindla-ec).

5. tasks

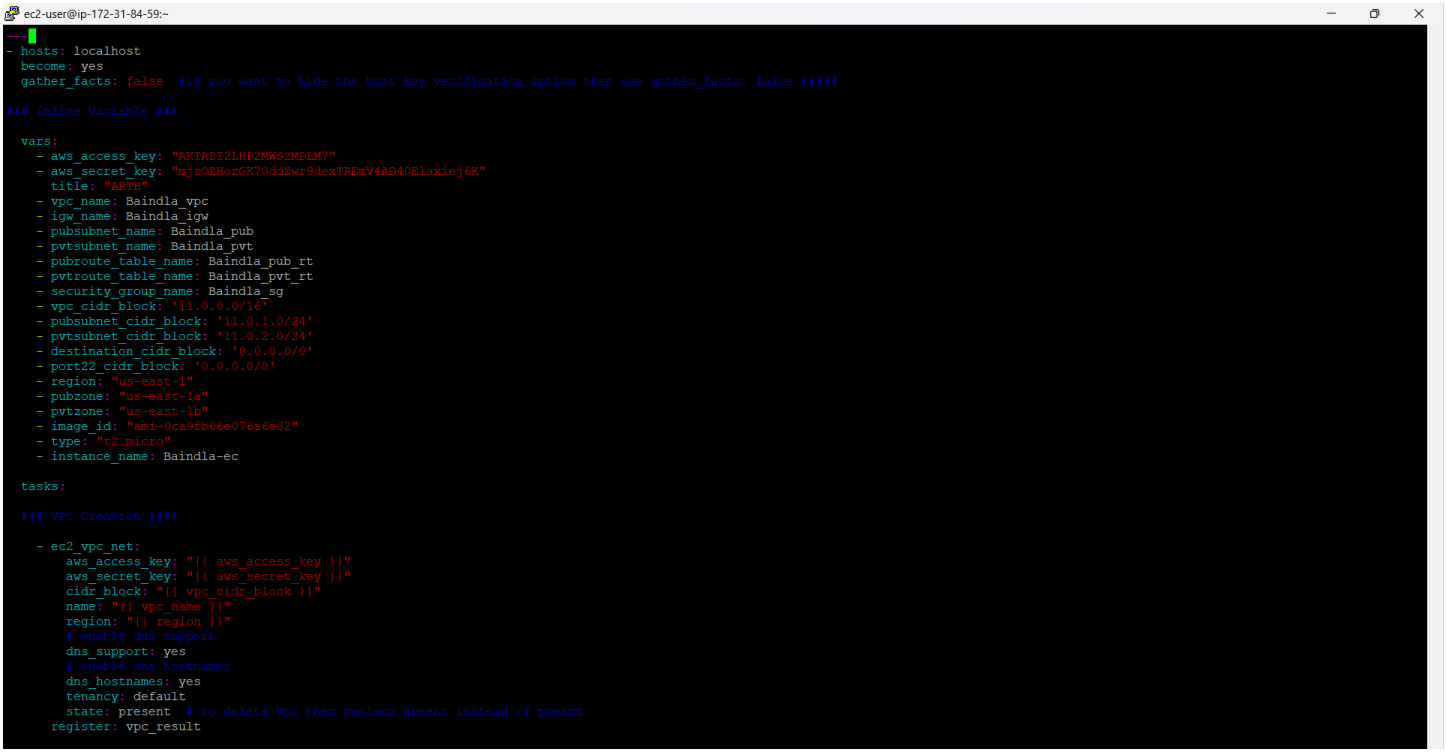
- This section will include the specific actions that need to be performed using the defined variables. These tasks would likely involve creating a VPC, subnets, routing tables, security groups, and EC2 instances based on the variables defined earlier.

VPC Creation Task Explanation:

This task creates an AWS Virtual Private Cloud (VPC) with the specified configurations.

1. **ec2_vpc_net:**
   - This is an Ansible module used to create or manage an AWS VPC.
2. **aws_access_key and aws_secret_key:**
   - These variables are used for authenticating with AWS to perform the operation. They are passed from the playbook's variables (aws_access_key, aws_secret_key).
3. **cidr_block:**
   - This specifies the IP address range for the VPC (in this case, vpc_cidr_block), which is 11.0.0.0/16 based on the variable. This will define the available range of private IP addresses for the resources inside this VPC.
4. **name:**
   - The name of the VPC being created. The value of vpc_name is dynamically provided from the variables (Baindla_vpc).

5. **region:**
    - This defines the AWS region where the VPC will be created (e.g., us-east-1).
6. **dns_support and dns_hostnames:**
    - These options are set to yes to enable DNS support and DNS hostnames for instances within the VPC. Enabling this allows instances to resolve DNS names in the VPC.
7. **tenancy:**
    - This defines the tenancy of the VPC. The value default means that instances in the VPC will use shared hardware. If you want dedicated hardware, you could set this to dedicated.
8. **state: present:**
    - This indicates that the task should ensure the VPC is created. If the state were set to absent, it would delete the VPC instead of creating it.
9. **register: vpc_result:**
    - The result of this task (the VPC details) will be stored in the variable vpc_result. This can be used later in the playbook for further actions, such as retrieving the VPC ID or using it in other tasks.



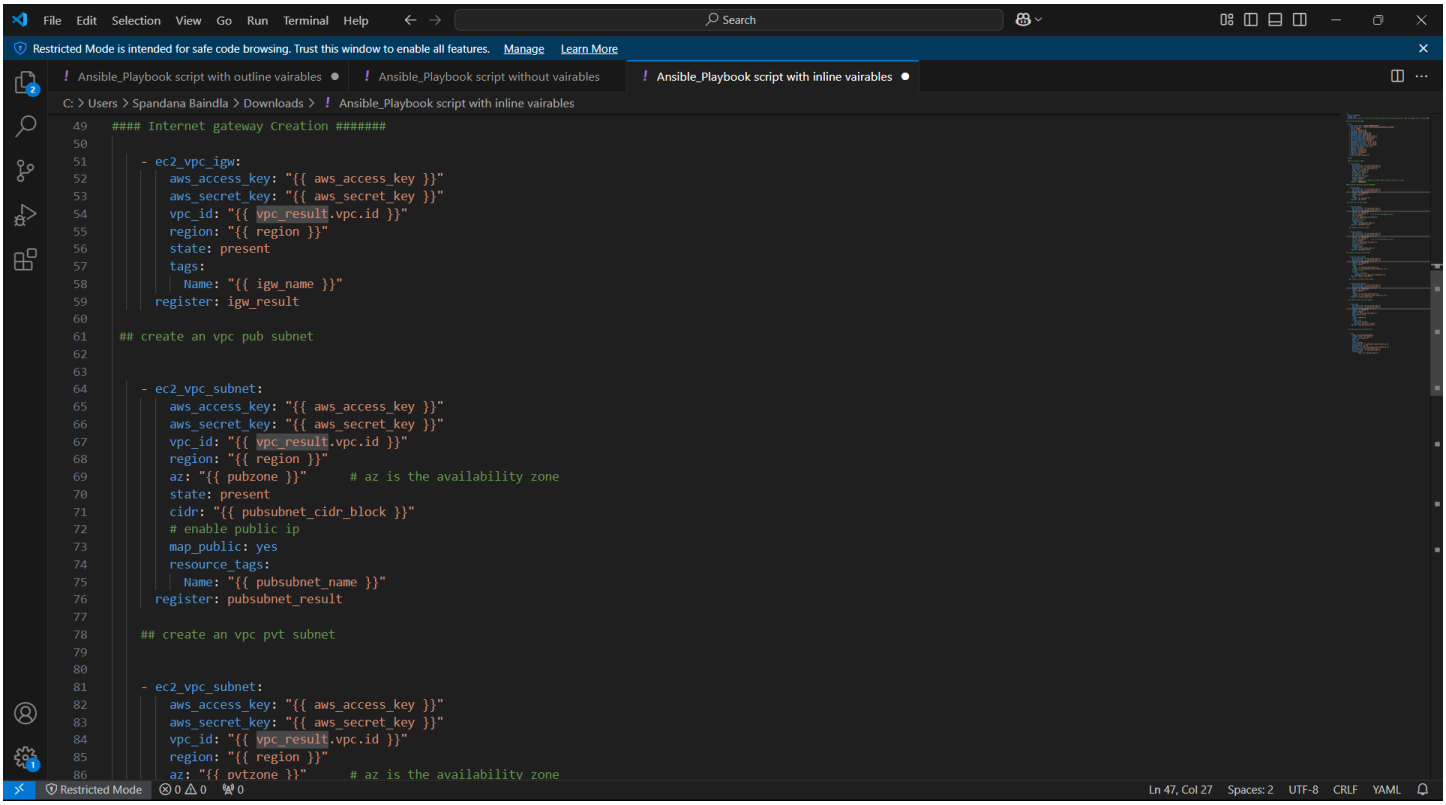
Internet Gateway Creation Task:

1. **ec2_vpc_igw**:
    - This Ansible module is used to create an Internet Gateway (IGW) for the VPC.
2. **aws_access_key** and **aws_secret_key**:
    - These provide AWS credentials to authenticate the request.
3. **vpc_id**:
    - The VPC ID is dynamically retrieved from the result of the previous task (i.e., vpc_result.vpc.id), which holds the information about the created VPC.
4. **region**:
    - Specifies the AWS region where the IGW will be created, passed from the region variable.
5. **state: present**:
    - Indicates that the task should ensure the creation of the Internet Gateway. If state: absent, it would delete the IGW.
6. **tags**:
    - Assigns the name of the Internet Gateway using the variable igw_name (e.g., Baindla_igw).
7. **register: igw_result**:
    - Stores the result of the task (details about the IGW) in the igw_result variable, which can be used later.

Public Subnet Creation Task:

1. **ec2_vpc_subnet**:
    - This module is used to create a subnet inside the VPC.
2. **aws_access_key** and **aws_secret_key**:
    - These credentials authenticate the request to AWS.
3. **vpc_id**:
    - The VPC ID from the previous task (vpc_result.vpc.id) is passed to create the subnet in the correct VPC.
4. **region**:
    - Specifies the region for the subnet creation, passed from the region variable.
5. **az**:
    - Specifies the Availability Zone for the subnet, passed from the pubzone variable (e.g., us-east-1a).
6. **state: present**:
    - Ensures that the subnet is created.

7. **cidr**:
    - Defines the CIDR block for the subnet using the pubsubnet_cidr_block variable (e.g., 11.0.1.0/24).
8. **map_public: yes**:
    - Enables the automatic assignment of public IP addresses to instances launched in this subnet. This is useful for creating a public-facing subnet.
9. **resource_tags**:
    - Assigns a tag with the name of the subnet, using the pubsubnet_name variable (e.g., Baindla_pub).
10. **register: pubsubnet_result**:
    - Stores the result of the subnet creation in the pubsubnet_result variable, which can be referenced later.



Create Private Subnet:

1. **ec2_vpc_subnet**:
    - **aws_access_key** and **aws_secret_key**: These are the AWS credentials for authentication.
    - **vpc_id**: The VPC ID for the subnet, taken from vpc_result.vpc.id (created in a previous task).
    - **region**: The AWS region, passed from the region variable.
    - **az**: Specifies the Availability Zone for the subnet, here pvtzone (e.g., us-east-1b).

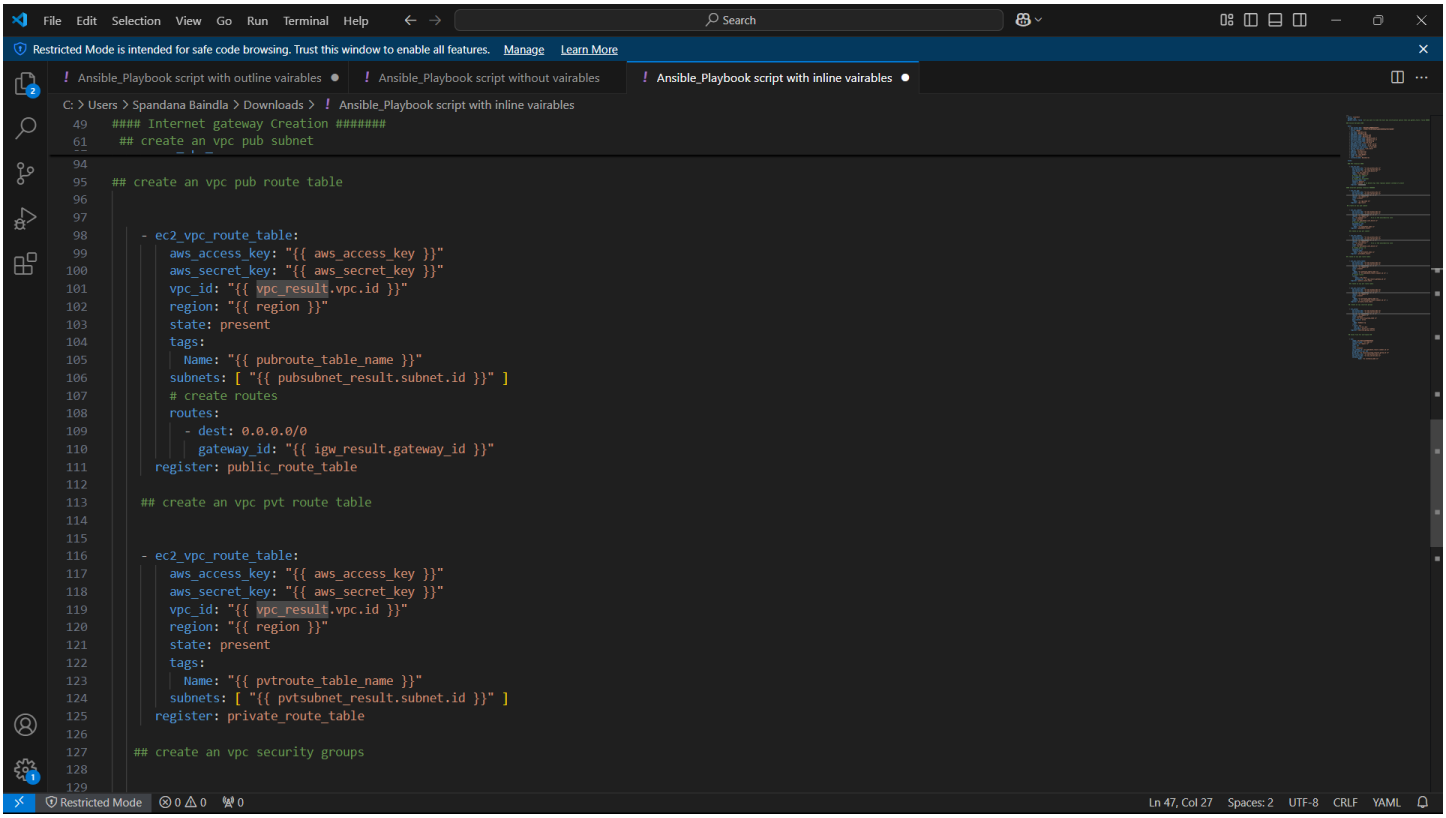
- **state: present**: Ensures that the subnet is created.
- **cidr**: The CIDR block for the subnet, passed from pvtsubnet_cidr_block (e.g., 11.0.2.0/24).
- **map_public: no**: This disables the automatic assignment of public IP addresses to instances in the private subnet.
- **resource_tags**: Tags the subnet with a name, taken from the pvtsubnet_name variable (e.g., Baindla_pvt).
- **register: pvtsubnet_result**: Registers the result of the task in the variable pvtsubnet_result to be used later.

Create Public Route Table:

2. **ec2_vpc_route_table**:
    - **vpc_id**: The VPC ID, which is retrieved from vpc_result.vpc.id.
    - **region**: The region in which the route table will be created.
    - **state: present**: Ensures the creation of the route table.
    - **tags**: Assigns a name to the route table, passed from pubroute_table_name (e.g., Baindla_pub_rt).
    - **subnets**: Associates the created public subnet (from pubsubnet_result.subnet.id) with this route table.
    - **routes**: Defines the routing rule, where 0.0.0.0/0 (all traffic) is routed through the IGW (Internet Gateway).
    - **register: public_route_table**: Stores the result of the route table creation in public_route_table.

Create Private Route Table:

3. **ec2_vpc_route_table** (for private subnet):
    - Similar to the public route table, but this one is associated with the private subnet (pvtsubnet_result.subnet.id) and does not necessarily include an IGW (used for private communication).
    - **register: private_route_table**: Registers the result of the private route table creation.



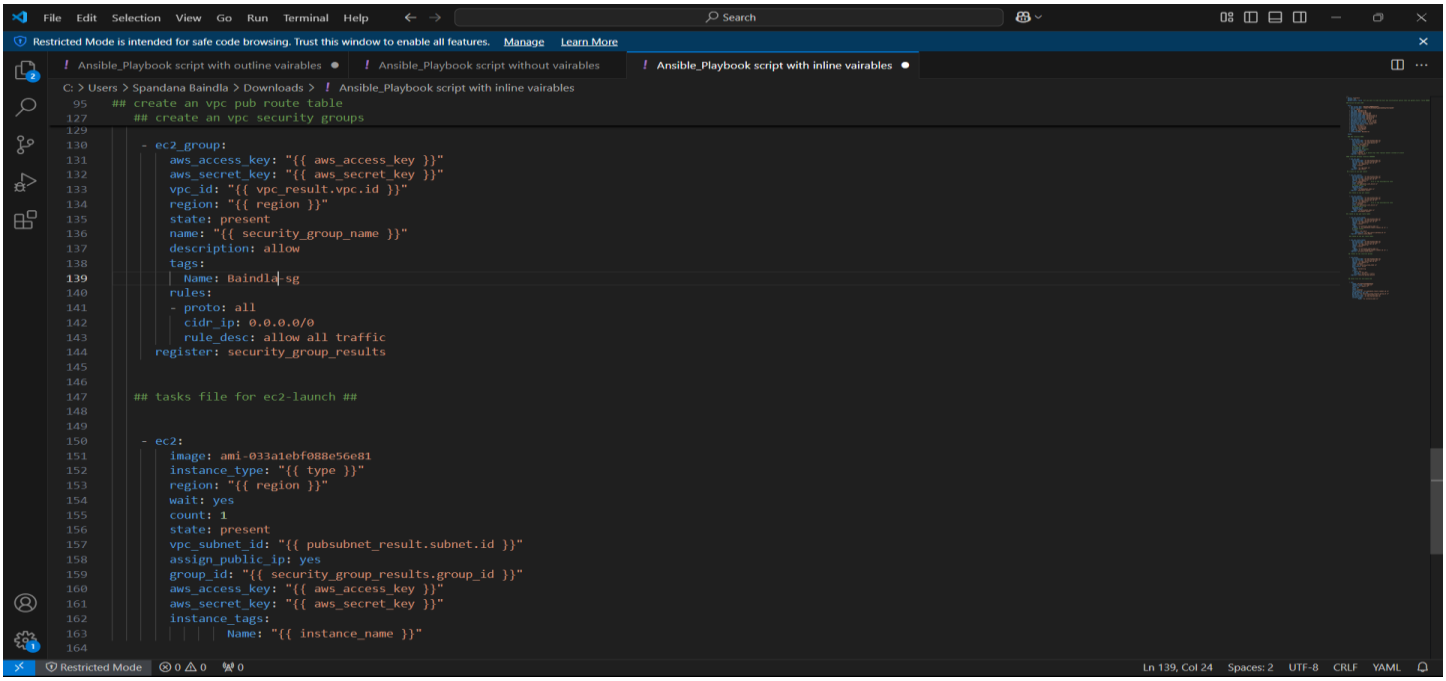
Create Security Group:

4. **ec2_group**:
    - **vpc_id**: The ID of the VPC, fetched from vpc_result.vpc.id.
    - **region**: The AWS region where the security group is created.
    - **state: present**: Ensures the security group is created.
    - **name**: The name of the security group, passed from the security_group_name variable (e.g., Baindla_sg).
    - **description**: A description of the security group, set as allow.
    - **rules**: Defines the security rules. The rule here allows all inbound traffic (proto: all and cidr_ip: 0.0.0.0/0).
    - **register: security_group_results**: Registers the result of security group creation in security_group_results.

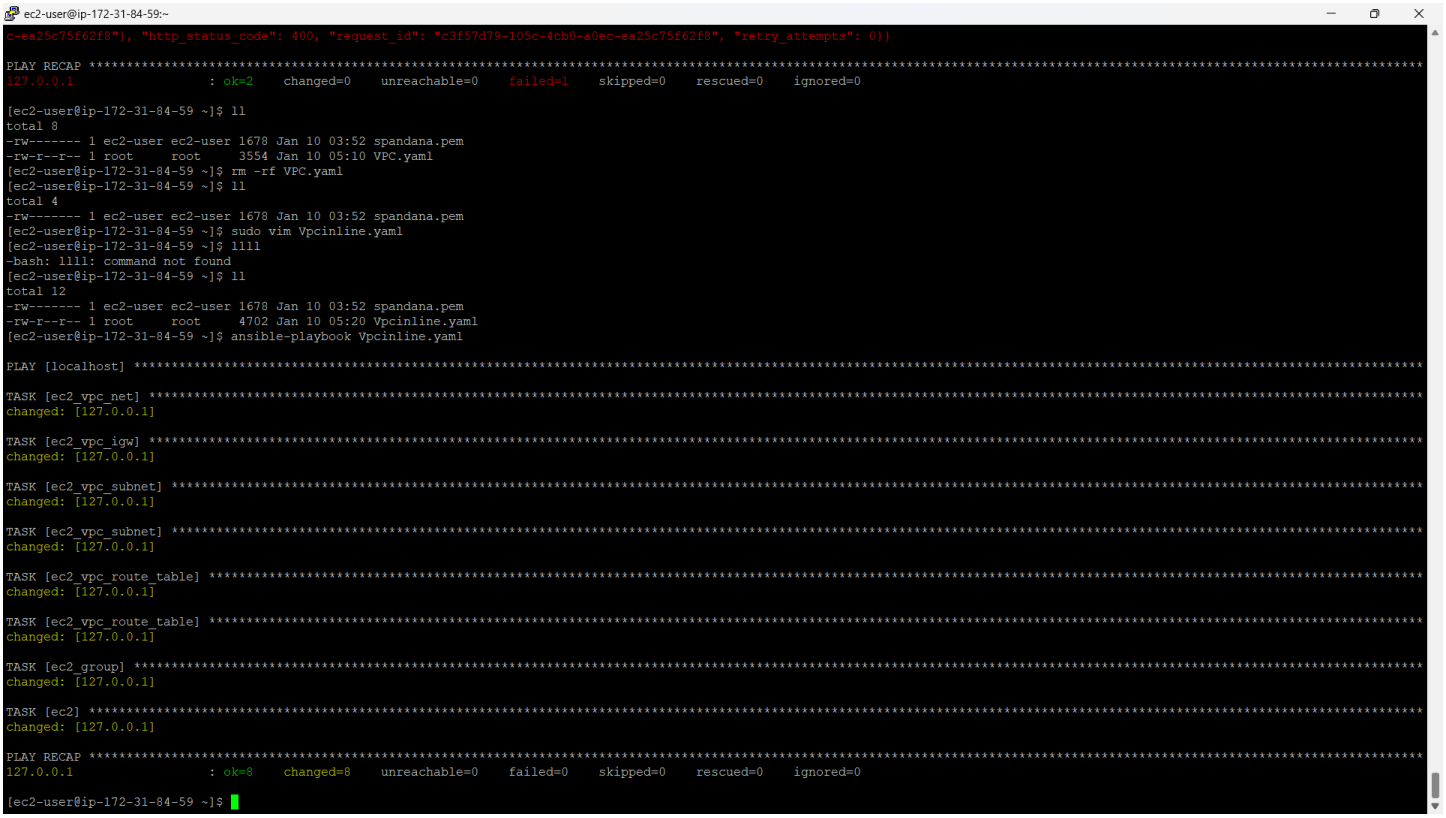
Launch EC2 Instance:

5. **ec2**:

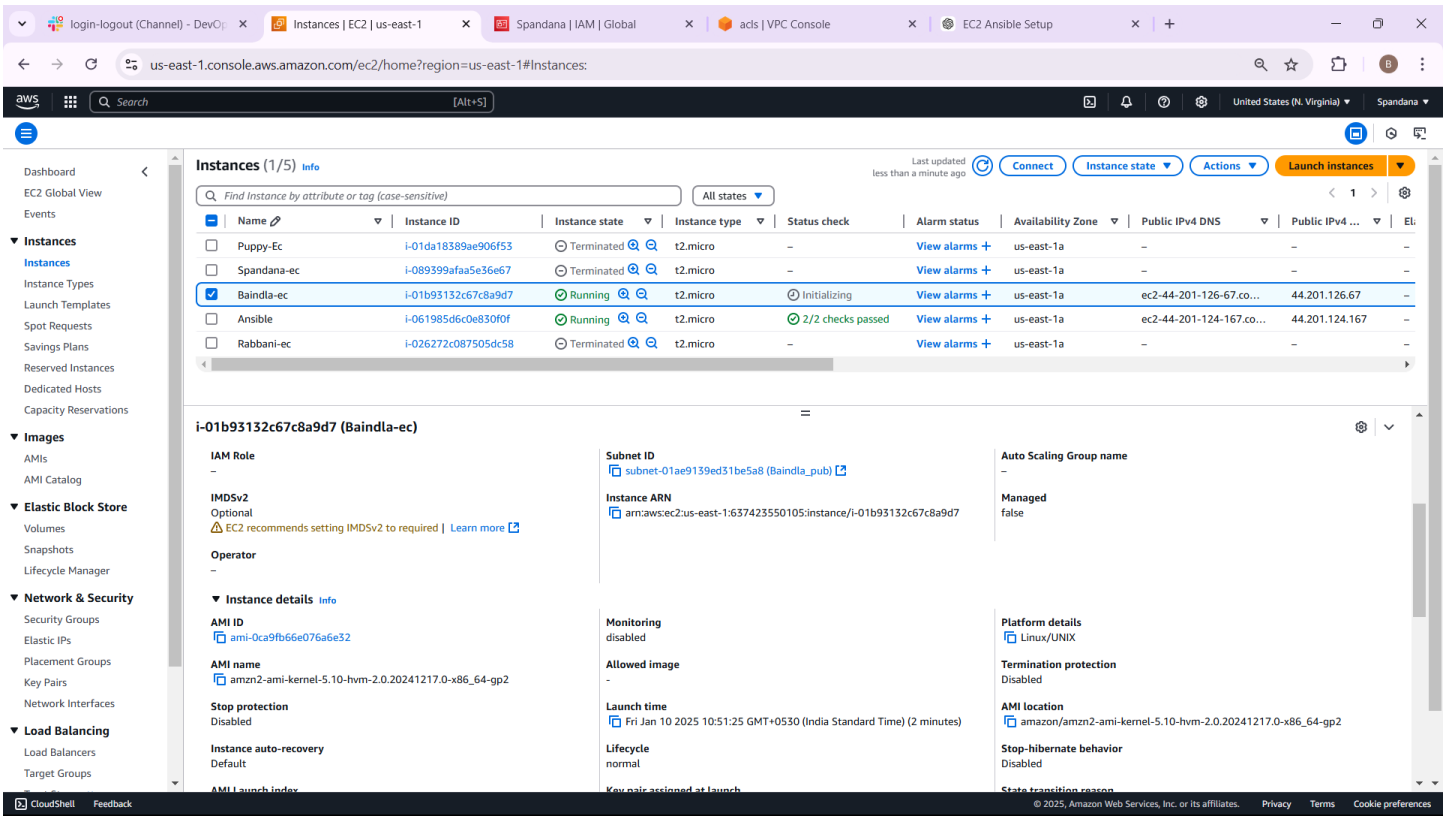
   - **image**: The AMI ID for the EC2 instance (e.g., ami-033a1ebf088e56e81).
   - **instance_type**: The instance type, passed from the type variable (e.g., t2.micro).
   - **region**: Specifies the AWS region where the EC2 instance is created.
   - **wait: yes**: Ensures the playbook waits for the instance to be in the "running" state before moving on.
   - **count: 1**: Launches a single EC2 instance.
   - **state: present**: Ensures the EC2 instance is created.
   - **vpc_subnet_id**: Associates the EC2 instance with the public subnet (from pubsubnet_result.subnet.id).
   - **assign_public_ip: yes**: Assigns a public IP address to the EC2 instance, making it accessible from the internet.
   - **group_id**: Associates the instance with the security group created earlier (security_group_results.group_id).
   - **instance_tags**: Tags the EC2 instance with the name from instance_name (e.g., Baindla-ec).



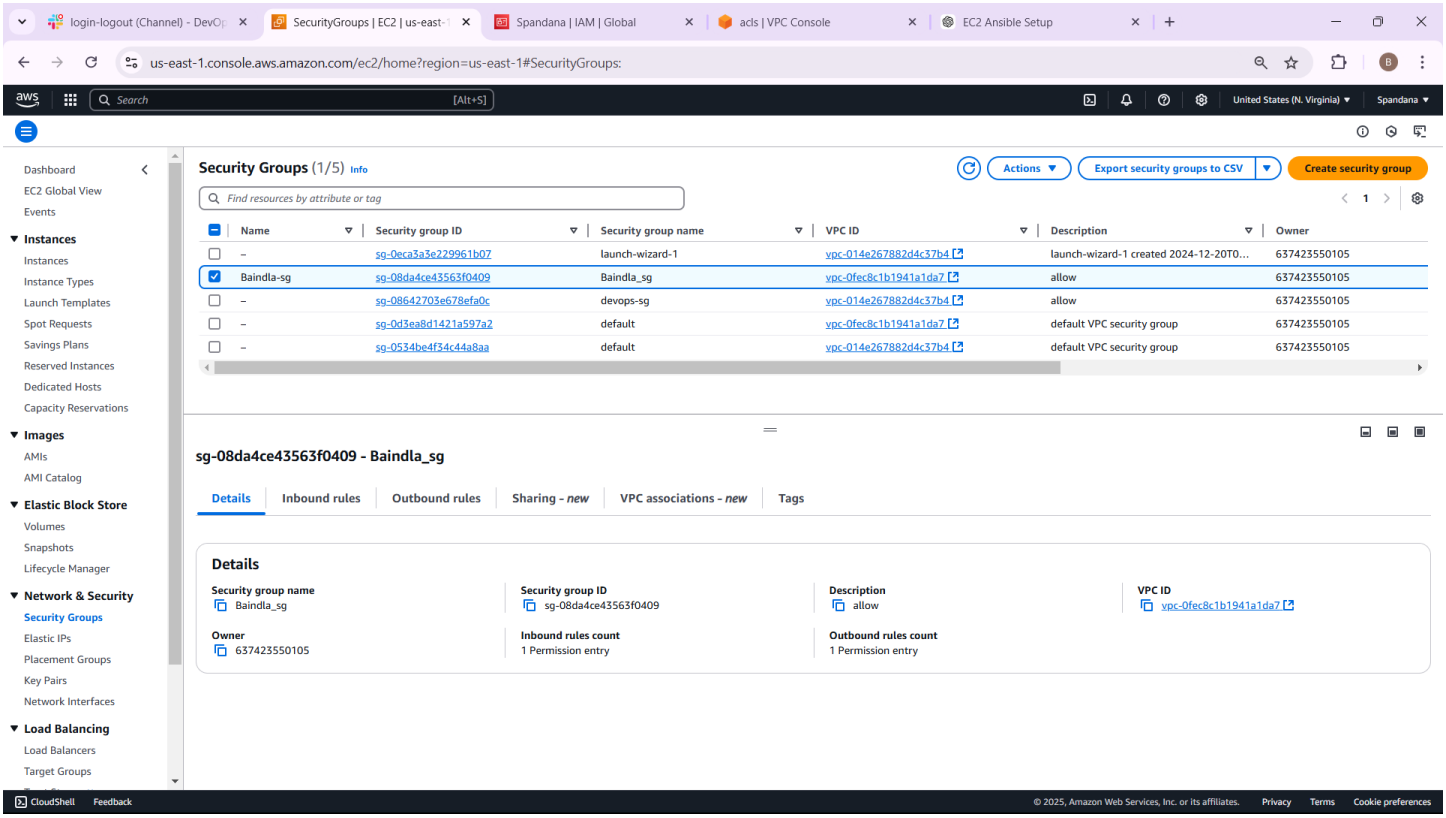
Then I have run the playbook using ansible-playbook vpcinline.yaml file.



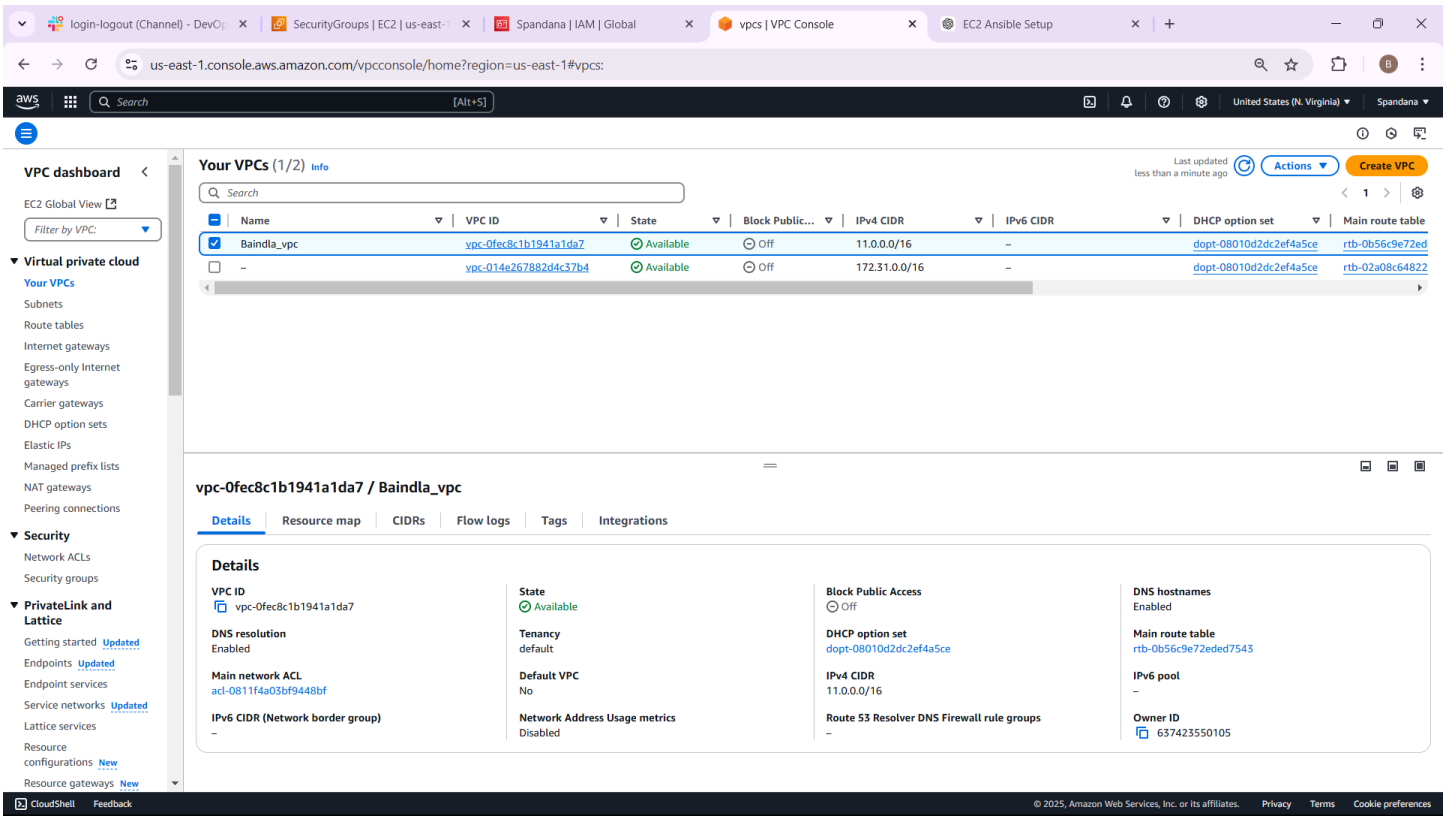
After successfully running of an above command new instance has been created successfully has shown below.



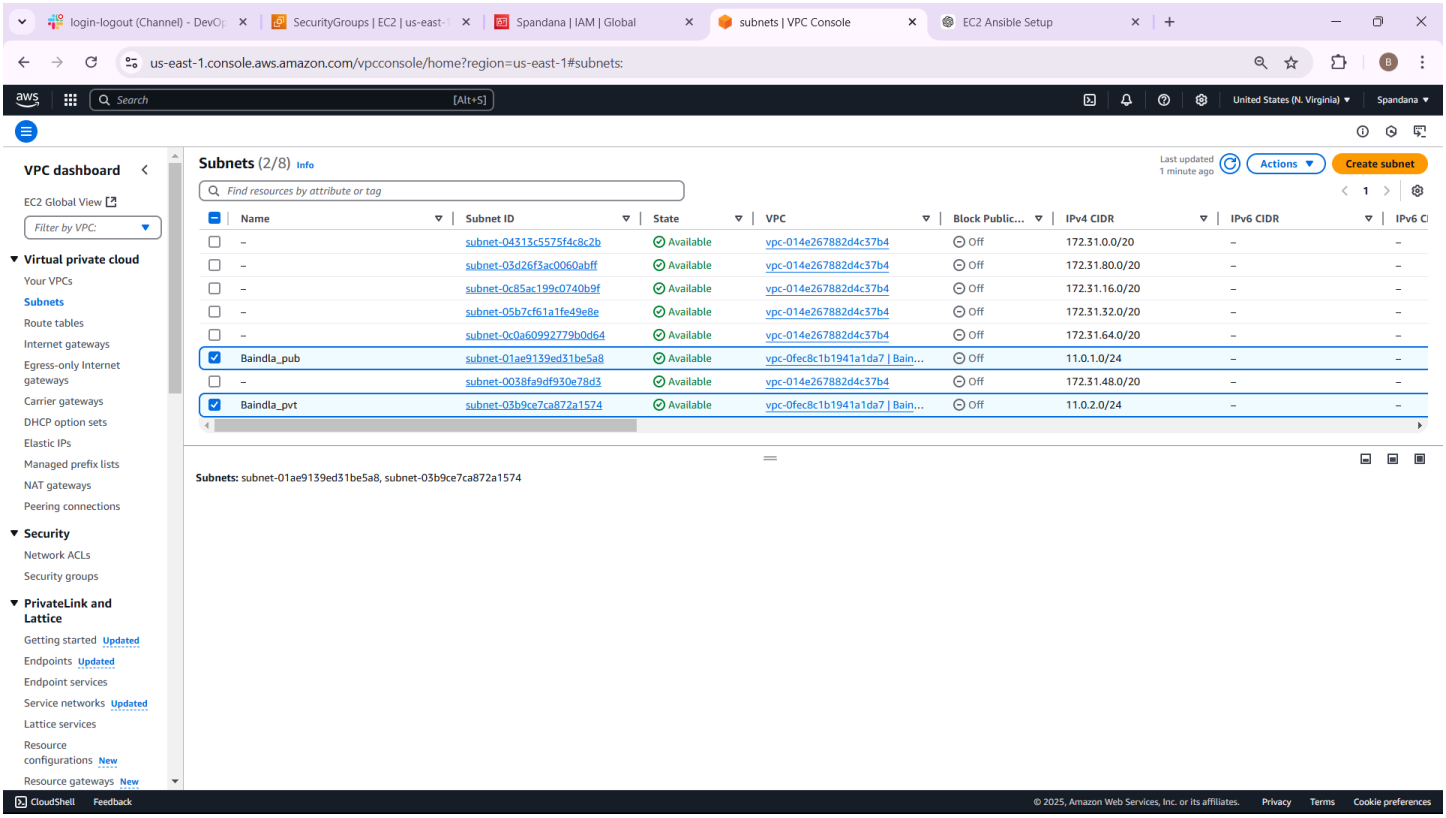
Successfully new security group has been created we can able to see below.



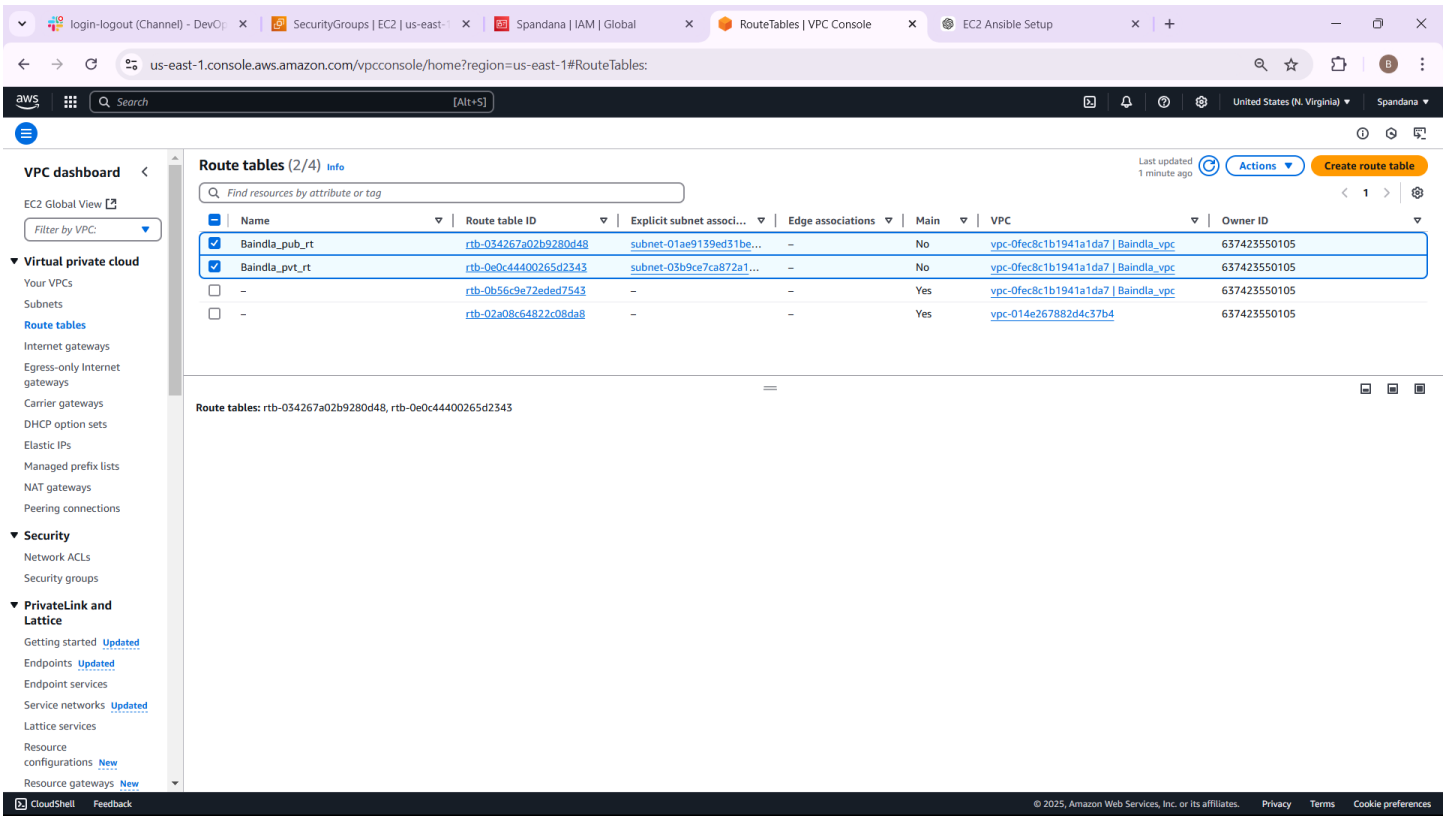
Successfully new VPC has been created.



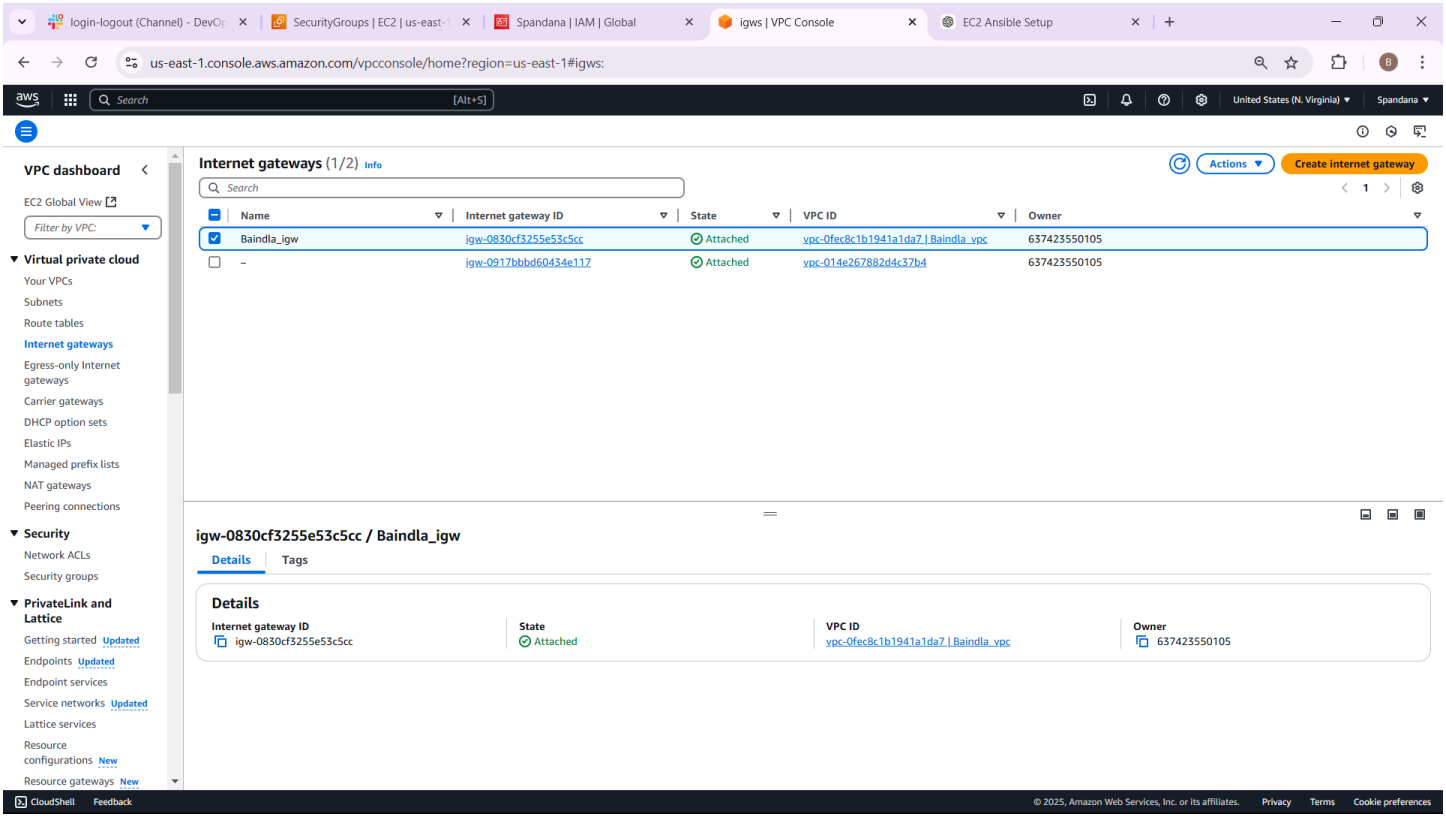
Successfully Public and Private subnets has been created.



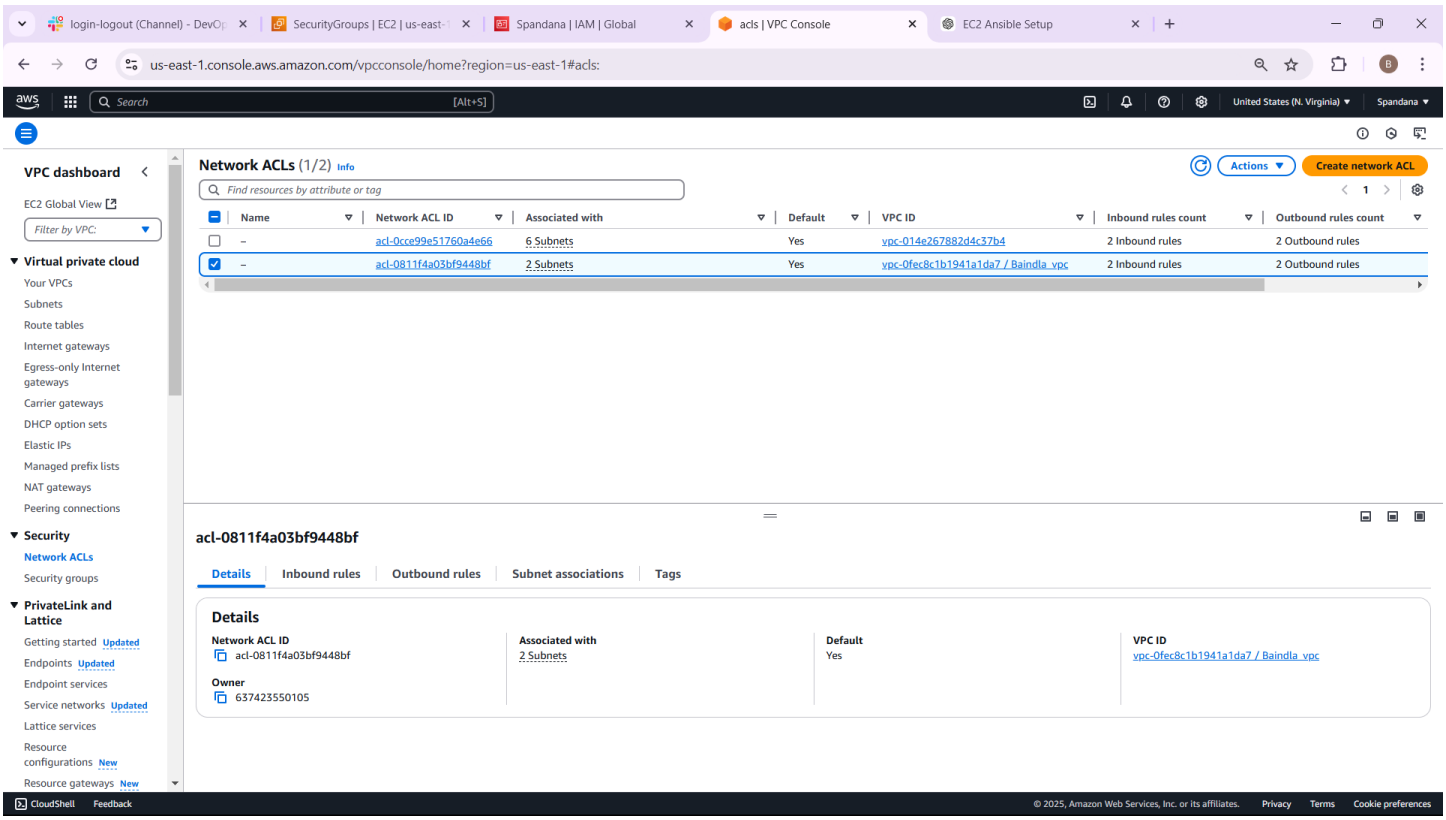
New public and private route table has been created successfully.



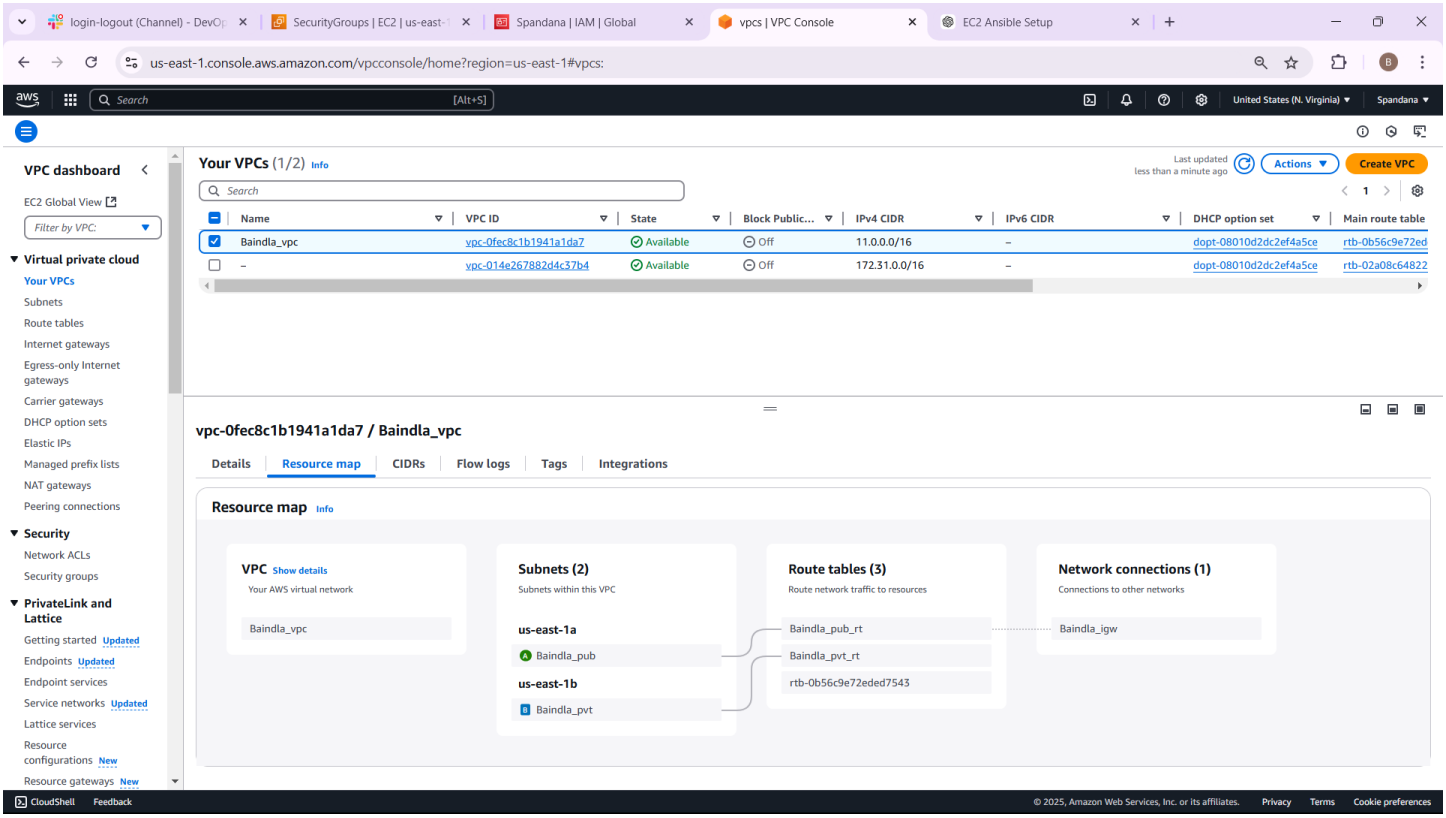
New internet gateway has been created successfully.



New network ACLs has been created successfully.



The below image is the Route Map for the Vpc.



To automatically delete the resources created by the Ansible playbook, you can modify the state attribute of the respective tasks from present to absent. This ensures that Ansible deletes the resources during the playbook run instead of creating them.

Important Notes:

- **Order of Deletion**: It's important to delete resources in a specific order to avoid dependency issues. For example, delete EC2 instances first, then security groups, route tables, subnets, internet gateways, and finally the VPC.
- **Dependencies**: Ensure that when deleting resources, there are no dependencies left (e.g., instances still running in the subnet, route tables not being associated properly).