PROJECT -5

AWS RESOURCE CREATION USING ANSIBLE

Ansible:

Ansible is an open-source automation tool used for configuration management, application deployment, and task automation (e.g., orchestration). It simplifies IT operations by automating repetitive tasks, enabling teams to scale and manage their infrastructure efficiently. Ansible is agentless, meaning it does not require any additional software to be installed on the target machines, and it uses SSH for communication.

Key Features of Ansible:

1. **Agentless**:

   No need for installing agents on remote machines. Ansible uses SSH to connect and execute commands.

2. **Simple and Easy to Learn**:

   Playbooks are written in YAML, a human-readable format that makes configuration and automation tasks easy to understand.

3. **Cross-Platform**:

   Supports a wide range of operating systems, including Linux, macOS, and Windows.

4. **Idempotent**:

   Ensures the desired state is achieved without causing unintended changes when tasks are repeated.

5. **Scalable**:

   Can manage a small number of servers or scale up to thousands.

6. **Extensible**:

   Comes with a wide variety of pre-built modules for common tasks, and you can create custom modules if needed.

Components of Ansible:

1. **Control Node**:

   The machine where Ansible is installed and from which commands are run.

2. **Managed Nodes**:

   The servers or devices that Ansible manages. These do not require any Ansible software installed.

3. **Inventory**:

   A file that lists the target machines (IP addresses or hostnames) to manage. It can be in **INI** or **YAML** format.

4. **Playbooks**:

YAML files that define a series of tasks to automate processes.

5. **Modules**:

   Predefined units of code used to execute tasks like installing packages, copying files, or configuring services.

6. **Plugins**:

   Extend Ansible's functionality, such as callback plugins, connection plugins, and inventory plugins.

How Ansible Works:

1. The user creates an inventory file and writes playbooks.
2. The control node sends the playbooks to the managed nodes via **SSH**.
3. Ansible executes the tasks on the managed nodes using **modules**.
4. It ensures the desired state is reached without manual intervention.

Playbooks

Playbooks in Ansible are files written in YAML that define the tasks you want to automate.

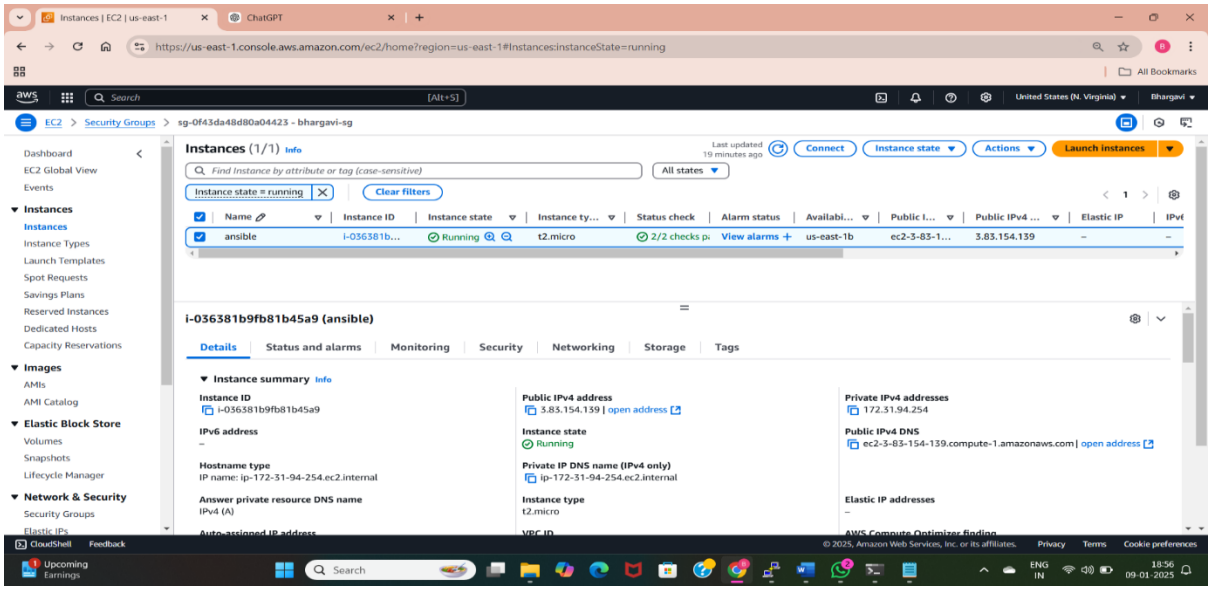
A playbook contains a list of **tasks** to define what actions need to be performed on the managed nodes (like installing software, starting services, etc.).

Key Features of Playbooks:

- **Human-readable**: Uses YAML, which is easy to read and write.
- **Reusable**: Can be used multiple times for different systems.
- **Structured**: Allows you to define tasks in a clear and logical order.

ANSIBLE_PLAYBOOK SCRIPT WITH INLINE VAIRABLES:

To begin using Ansible to manage your infrastructure, first we need to launch an instance with the appropriate operating system and instance type.

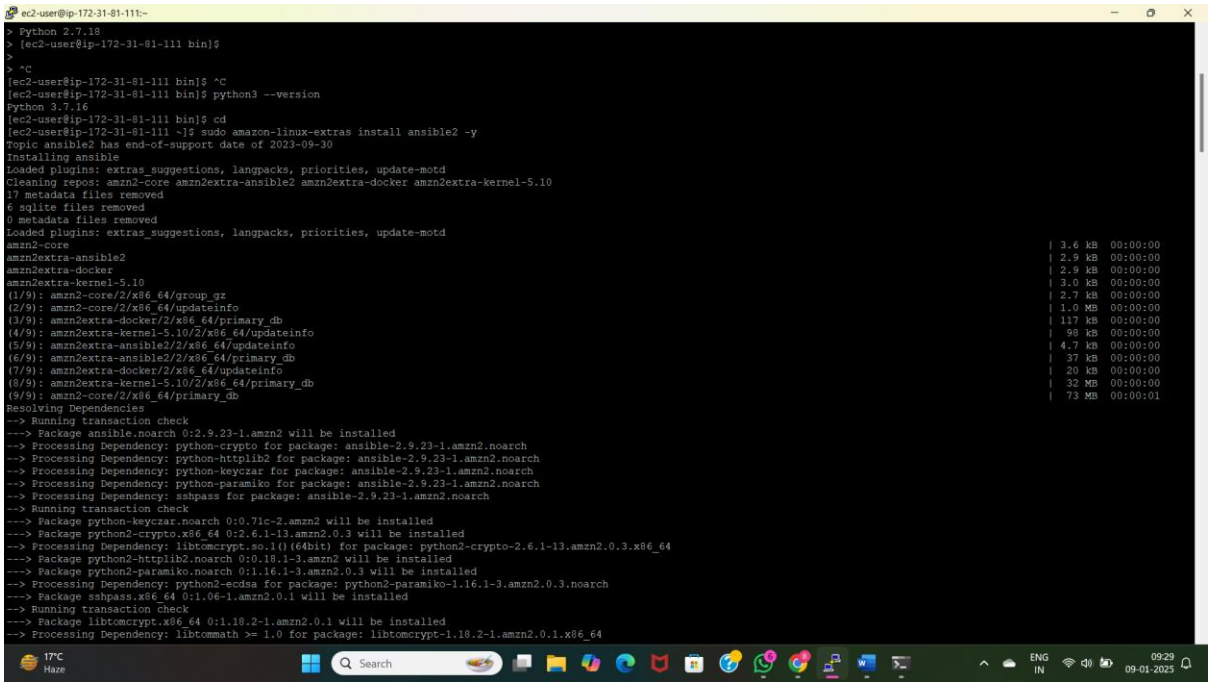


Here I have launched a ec2 instance by navigating into the ec2 service in aws console and select launch instance, give name for the instance, select application os(I have selected amazon-linux kernel 5.10 version), select instance type, provide network(either default vpc or manually created vpc), select the security group to allow the traffic/access from the outside, increase the storage if you have more load and launch the instance.

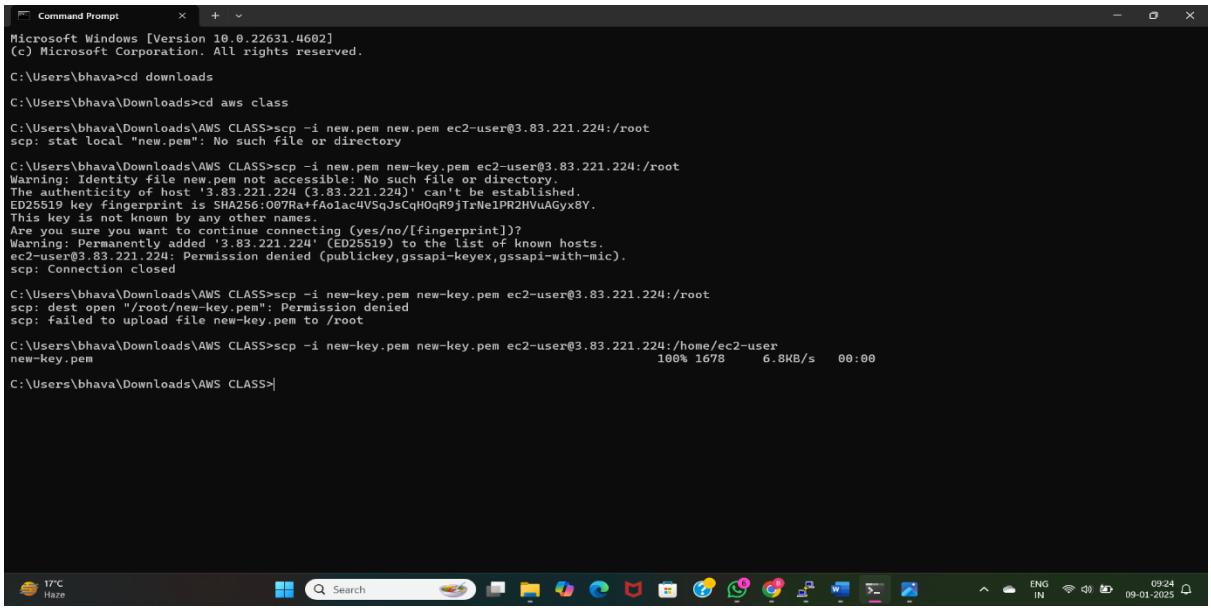
Install Required Software:

To use Ansible effectively with AWS, you need several key software tools installed on your local system.

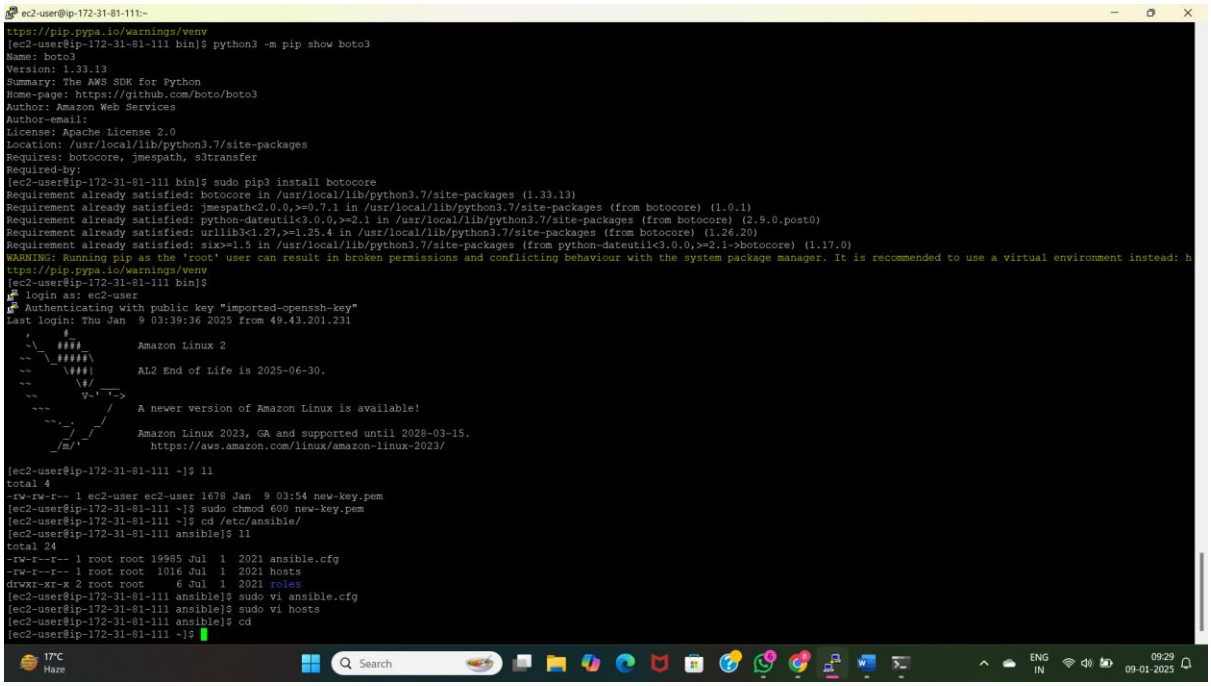
- ♣ python3
- ♣ pip3
- ♣ boto
- ♣ boto3
- ♣ botocore



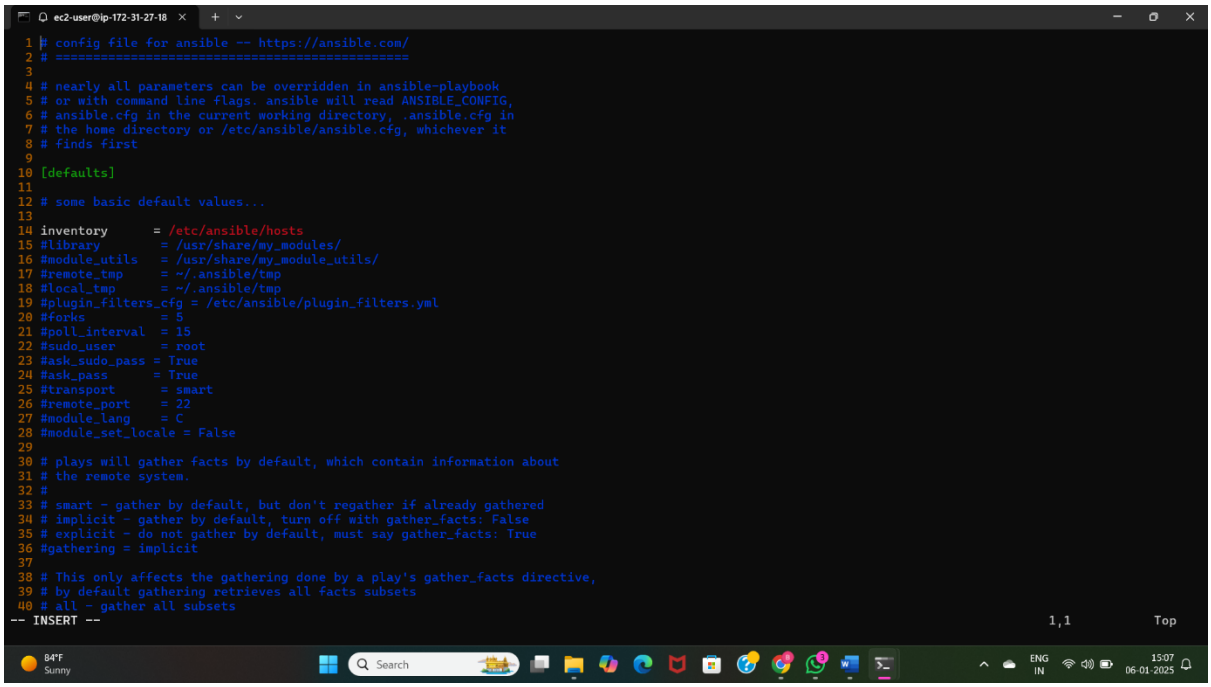
- Python is the underlying runtime for Ansible. Ansible modules and scripts rely on Python for execution. Here am using amazon-linux by default python is installed in it.
- Pip is Python's package manager and is required to install additional libraries like boto and boto3 using command <sudo pip3 install boto boto3 botocore>.
- Boto3 is the AWS SDK for Python and allows Ansible to communicate with AWS services.
- Ansible uses Boto3 under the hood for tasks like creating EC2 instances, VPCs, and other resources.



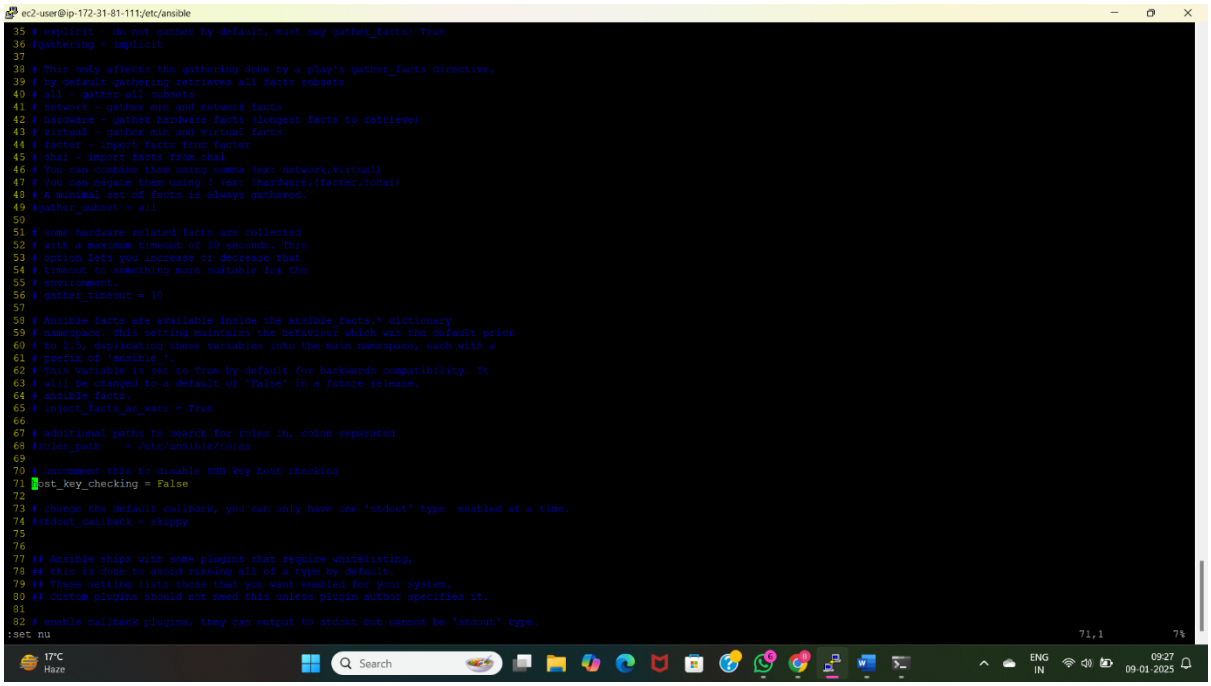
- Copy the private key from local host to ec2 instance using <scp -i pemfile pemfile ec2-user@publicip of instace: /to directory > by navigating to the path where we have stored the .pem file in the command prompt and run above command.
- We can see the successful copying of the pem file from local machine to your ec2 instance.



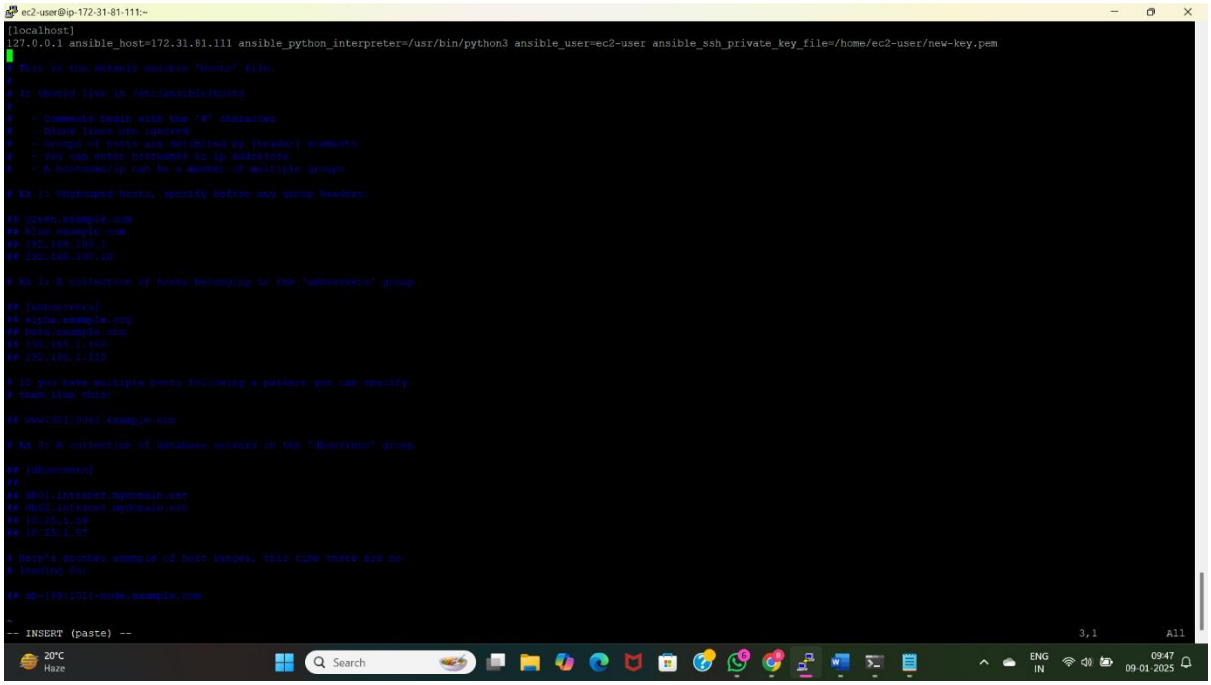
- After copying the private key check in the terminal whether the key is copied or not.
- If it is copied successfully give execution permissions for the private key.



- To configure Ansible to access the inventory file, you need to modify the ansible.cfg
- we need to go to the ansible.cfg file using <sudo vi /etc/ansible/ansible.cfg>, find inventory setting and remove hash(by default this line is commented as #) before inventory to give access to the host file.
- An inventory file is where you define the managed nodes (servers or devices) that Ansible will work on.



- By default, Ansible enables host key checking. This means Ansible will verify that the remote server's SSH host key matches what is stored in the known_hosts(where we have stored private key) file on your local machine.
- If the host key is not found, or if it has changed (which could indicate a security issue), the connection will fail.
- When you set host_key_checking = False, Ansible **skips** this verification process. So in the same file find for line number 71 by default it is also commented using the # symbol, uncomment the line host_key_checking=False.



- Now we need to specify the host in the inventory file using [localhost]: This defines a group of hosts named localhost. In Ansible, groups are used to logically organize hosts. This local host is used to refer the local machine where we are using ansible.

- ♣ 127.0.0.1 is refers to local machine will be used to initiate the connection to the remote EC2 instance.
- ♣ ansible_host=172.31.94.254 this is the actual instance that we want to manage and ansible_python_interpreter=/usr/bin/python3 using this we are saying ansible to use python3 for remote machine.
- ♣ ansible_user=ec2-user here we are setting the user as ec2-user for ssh connection.
- ♣ ansible_ssh_private_key_file=/home/ec2-user/new-key.pem specifies the private SSH key that Ansible will use to authenticate and connect to the EC2 instance.

ec2-user@ip-172-31-81-111:~

```
---
- hosts: localhost
  become: yes
  gather_facts: false  #if you want to hide the host key verification option then use gather_facts: false #####

### Inline Variable ###

  vars:
    - aws_access_key: "AKIA5FTZD3CQW63UY2XC"
    - aws_secret_key: "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"
      title: "ARTH"
    - vpc_name: bhargavi_vpc
    - igw_name: bhargavi_igw
    - pubsubnet_name: bhargavi_pub
    - pvtsubnet_name: bhargavi_pvt
    - pubroute_table_name: bhargavi_pub_rt
    - pvtroute_table_name: bhargavi_pvt_rt
    - security_group_name: bhargavi_sg
    - vpc_cidr_block: '11.0.0.0/16'
    - pubsubnet_cidr_block: '11.0.1.0/24'
    - pvtsubnet_cidr_block: '11.0.2.0/24'
    - destination_cidr_block: '0.0.0.0/0'
    - port22_cidr_block: '0.0.0.0/0'
    - region: "us-east-1"
    - pubzone: "us-east-1a"
    - pvtzone: "us-east-1b"
    - image_id: "ami-0ca9fb66e076a6e32"
    - type: "t2.micro"
    - instance_name: bhargavi-ec

  tasks:
```

This is the playbook which defines inline variables. An inline variable in Ansible refers to a variable that is defined directly within the playbook, task, or role, rather than being defined in an external file. It's commonly used for simple, one-time values or temporary use in the playbook.

- ➲ Access keys and secret access keys for AWS access.
- ➲ Title, VPC name, and CIDR block for network configuration.
- ➲ Internet Gateway name, and names and CIDR blocks for public and private subnets.
- ➲ Regions, public and private route table names.
- ➲ SSH port access, allowing SSH connectionEC2 instance settings such as the region, Amazon Machine Image (AMI) ID, instance type, and instance name.

```
  ### VPC Creation ####

    - ec2_vpc_net:
        aws_access_key: "{{ aws_access_key }}"
        aws_secret_key: "{{ aws_secret_key }}"
        cidr_block: "{{ vpc_cidr_block }}"
        name: "{{ vpc_name }}"
        region: "{{ region }}"
        # enable dns support
        dns_support: yes
        # enable dns hostnames
        dns_hostnames: yes
        tenancy: default
        state: present  # to delete Vpc then replace absent instead of presnt
      register: vpc_result

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

    - ec2_vpc_igw:
        aws_access_key: "{{ aws_access_key }}"
        aws_secret_key: "{{ aws_secret_key }}"
        vpc_id: "{{ vpc_result.vpc.id }}"
        region: "{{ region }}"
        state: present
        tags:
          Name: "{{ igw_name }}"
      register: igw_result
```

This playbook is used to create a VPC (Virtual Private Cloud) with an Internet Gateway. In this playbook, we can see that inline variables are used to define various parameters such as access keys, VPC settings.

To create any service, install any command we need to specify the state. Present state specifies creation of the instances, vpc, subnets, route tables.

```
## create an vpc pub subnet


   - ec2_vpc_subnet:
       aws_access_key: "{{ aws_access_key }}"
       aws_secret_key: "{{ aws_secret_key }}"
       vpc_id: "{{ vpc_result.vpc.id }}"
       region: "{{ region }}"
       az: "{{ pubzone }}"      # az is the availability zone
       state: present
       cidr: "{{ pubsubnet_cidr_block }}"
       # enable public ip
       map_public: yes
       resource_tags:
         Name: "{{ pubsubnet_name }}"
     register: pubsubnet_result

   ## create an vpc pvt subnet


   - ec2_vpc_subnet:
       aws_access_key: "{{ aws_access_key }}"
       aws_secret_key: "{{ aws_secret_key }}"
       vpc_id: "{{ vpc_result.vpc.id }}"
       region: "{{ region }}"
       az: "{{ pvtzone }}"      # az is the availability zone
       state: present
       cidr: "{{ pvtsubnet_cidr_block }}"
       # enable public ip
       map_public: no
       resource_tags:
         Name: "{{ pvtsubnet_name }}"
     register: pvtsubnet_result
```

This playbook is used to create public and private subnets. In this playbook, we can see that inline variables are used to define various parameters such as access keys, VPC id, subnet details.

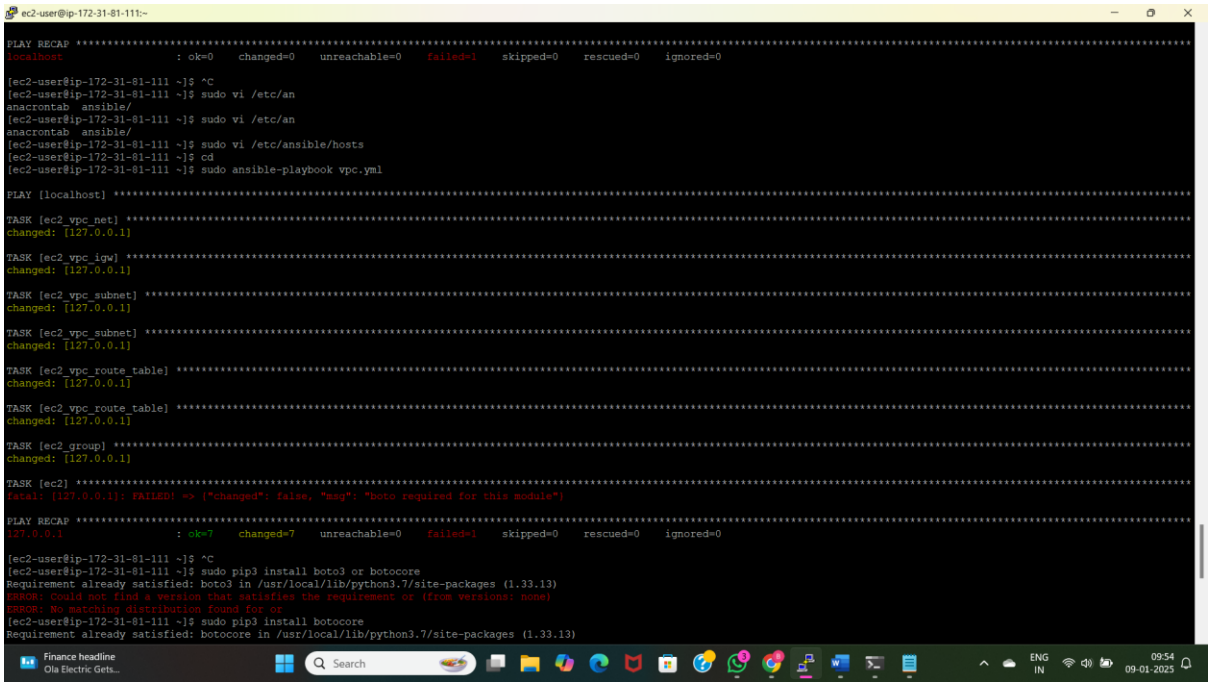
```
## create an vpc pub route table



    - ec2_vpc_route_table:
        aws_access_key: "{{ aws_access_key }}"
        aws_secret_key: "{{ aws_secret_key }}"
        vpc_id: "{{ vpc_result.vpc.id }}"
        region: "{{ region }}"
        state: present
        tags:
          Name: "{{ pubroute_table_name }}"
        subnets: [ "{{ pubsubnet_result.subnet.id }}" ]
        # create routes
        routes:
          - dest: 0.0.0.0/0
            gateway_id: "{{ igw_result.gateway_id }}"
      register: public_route_table

    ## create an vpc pvt route table


    - ec2_vpc_route_table:
        aws_access_key: "{{ aws_access_key }}"
        aws_secret_key: "{{ aws_secret_key }}"
        vpc_id: "{{ vpc_result.vpc.id }}"
        region: "{{ region }}"
        state: present
        tags:
          Name: "{{ pvtroute_table_name }}"
        subnets: [ "{{ pvtsubnet_result.subnet.id }}" ]
      register: private_route_table
```

This playbook is used to create public and private route table. In this playbook, we can see that inline variables are used to define various parameters such as access keys, VPC id, subnet details, route tables.

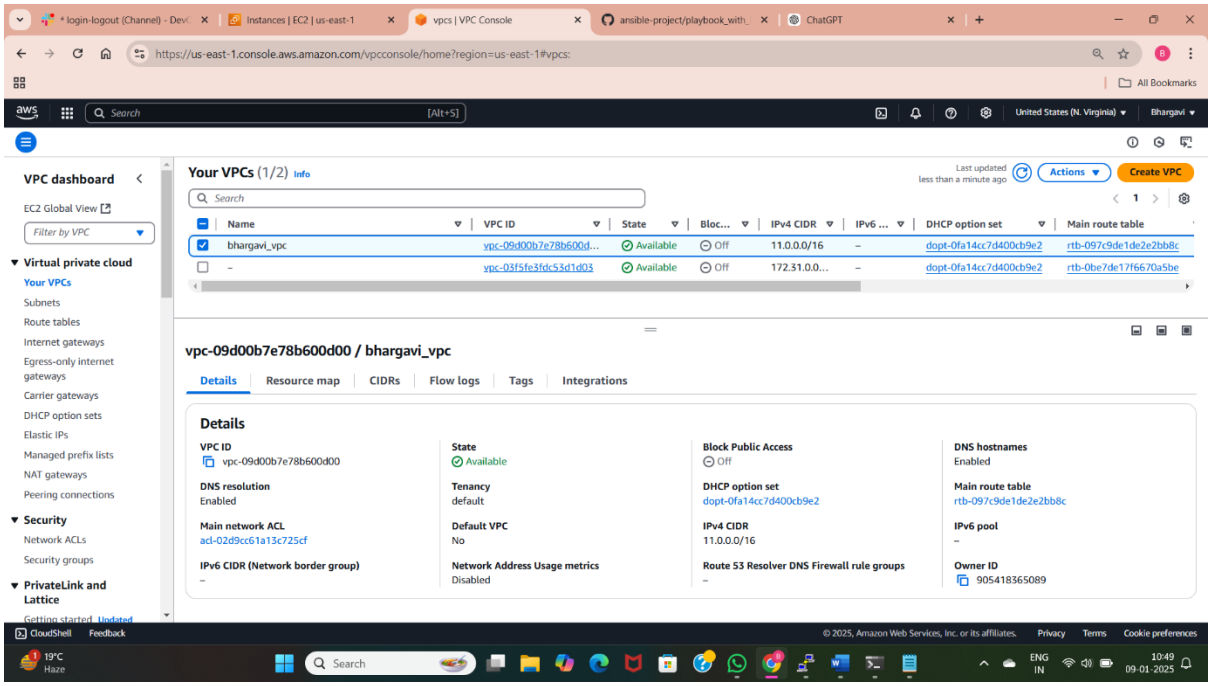
```
 - ec2_group:
     aws_access_key: "{{ aws_access_key }}"
     aws_secret_key: "{{ aws_secret_key }}"
     vpc_id: "{{ vpc_result.vpc.id }}"
     region: "{{ region }}"
     state: present
     name: "{{ security_group_name }}"
     description: allow
     tags:
       Name: bhargavi-sg
     rules:
     - proto: all
       cidr_ip: 0.0.0.0/0
       rule_desc: allow all traffic
   register: security_group_results


## tasks file for ec2-launch ##


 - ec2:
     image: ami-0ca9fb66e076a6e32
     instance_type: "{{ type }}"
     region: "{{ region }}"
     wait: yes
     count: 1
     state: present
     vpc_subnet_id: "{{ pubsubnet_result.subnet.id }}"
     assign_public_ip: yes
     group_id: "{{ security_group_results.group_id }}"
     aws_access_key: "{{ aws_access_key }}"
     aws_secret_key: "{{ aws_secret_key }}"
     instance_tags:
             Name: "{{ instance_name }}"
```

This playbook is used to create security group and ec2 instance. In this playbook, we can see that inline variables are used to define various parameters such as access keys, VPC id, public subnet details and EC2 instance configurations.

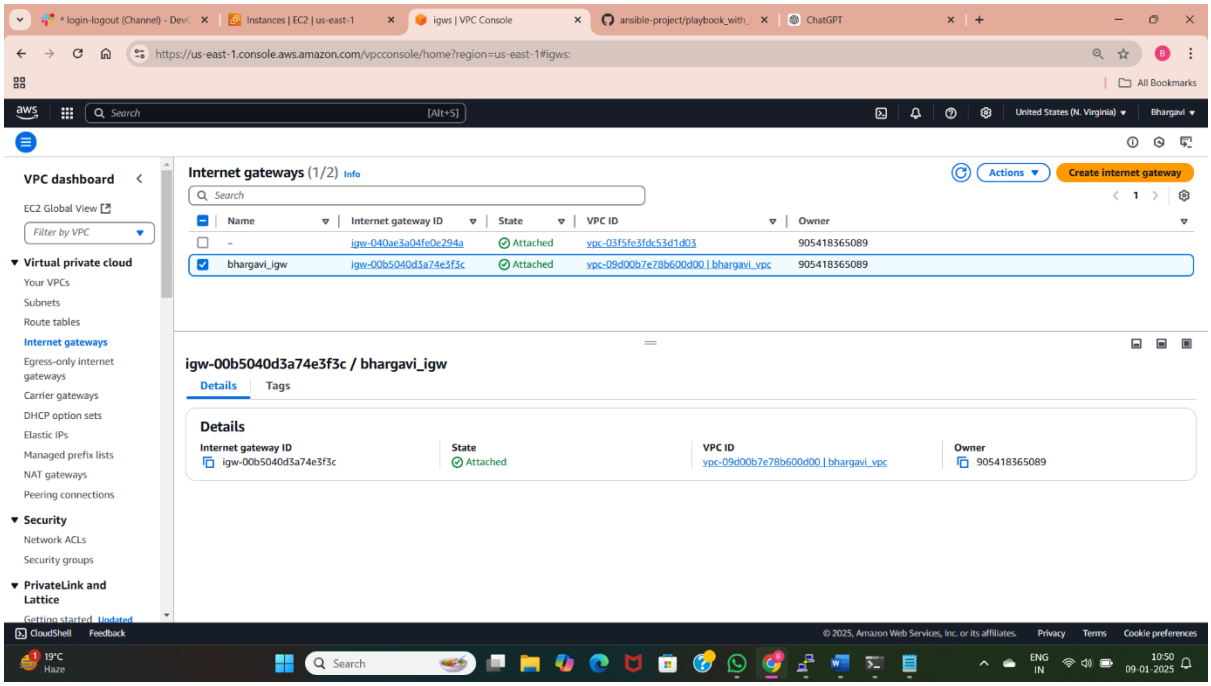
- ➔ After writing the playbook check for the syntax error using <ansible-playbook vpc.yml -syntax-check>.
- ➔ If there is no error run the playbook using the <ansile-playbook vpc.yml>. It will automates the creation and installations which mentioned in the playbook.



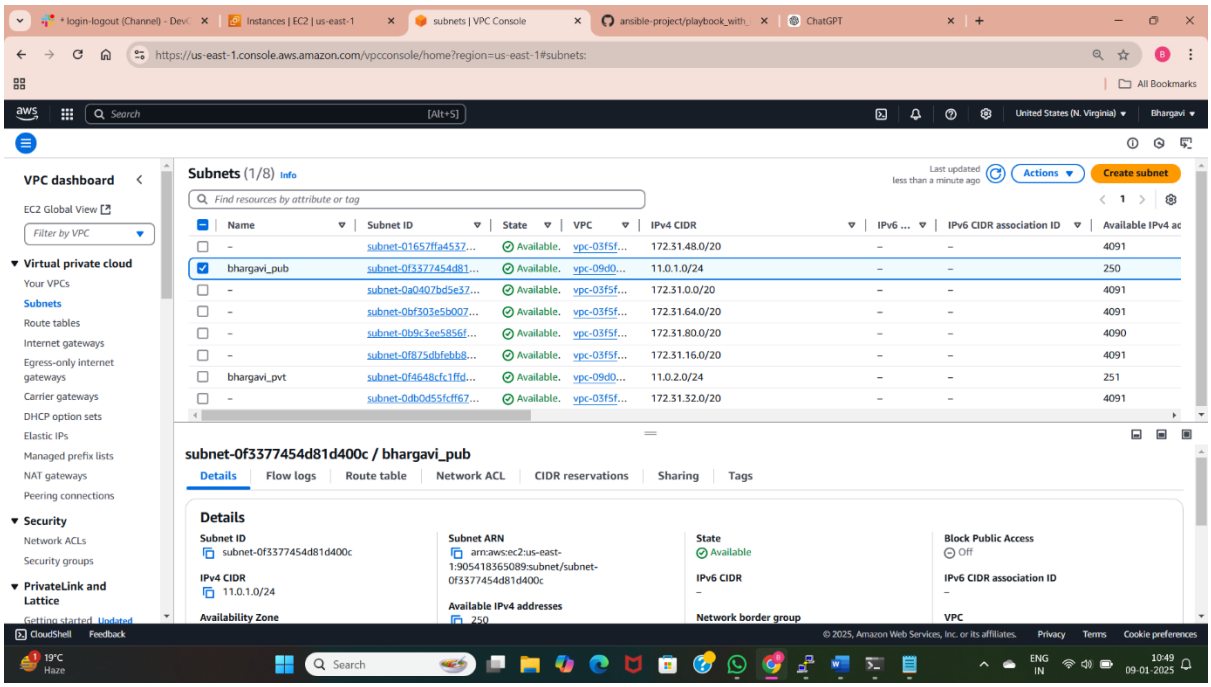
We can see that the ansible is connected to the local host, vpc, internet gateway, subnets, route table and ec2 security group is created successfully and there is an error indicating that boto is required for creating instance.



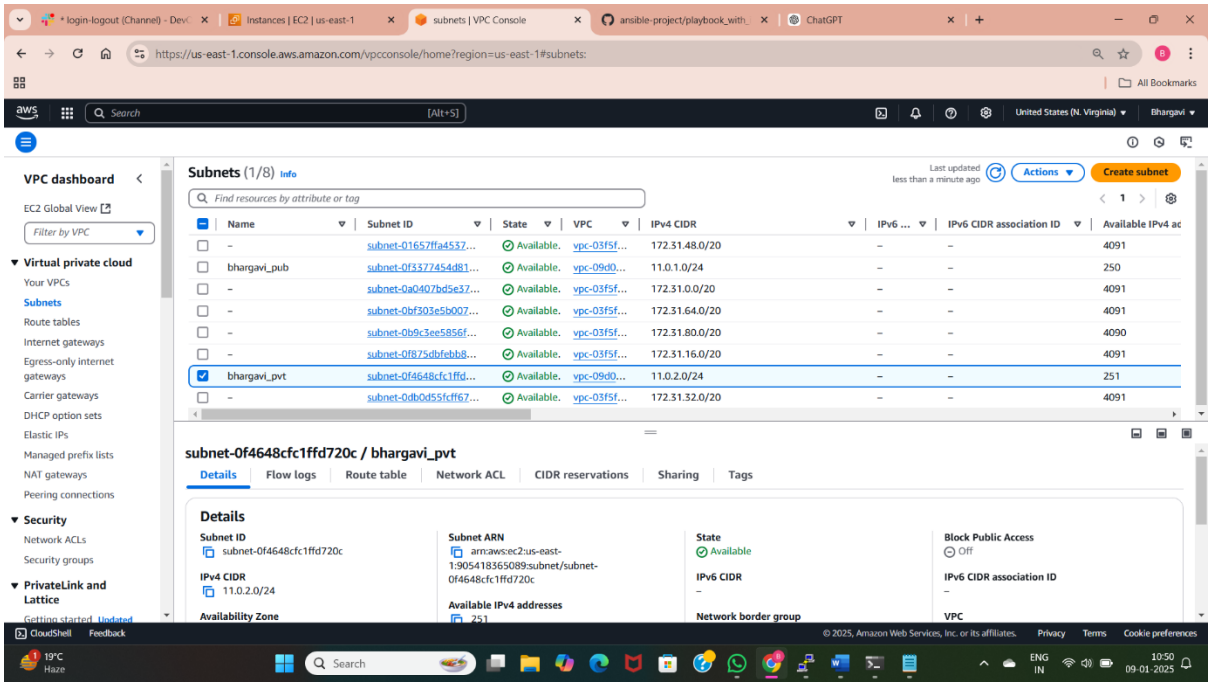
We can verify the VPC creation by checking the AWS Management Console. In the console, we can see the configuration of the VPC that was created using the Ansible playbook.



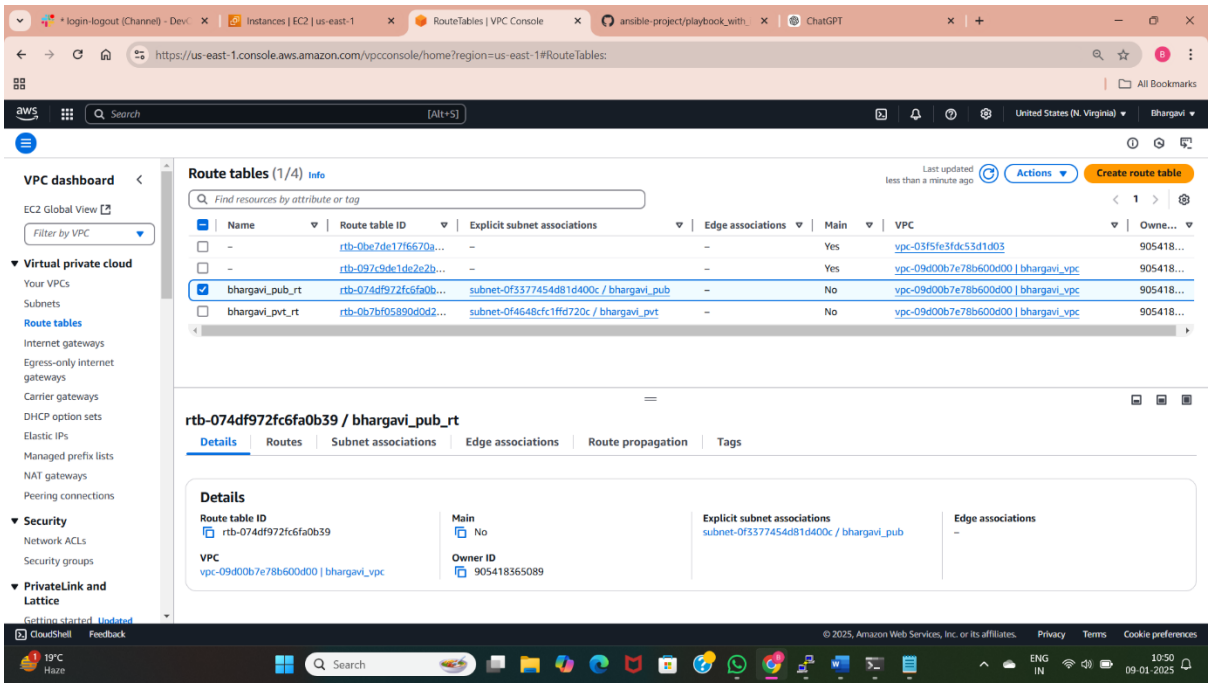
Navigate to the VPC section to check if the Internet Gateway has been created successfully and is attached to the VPC, it is created and attached to VPC.



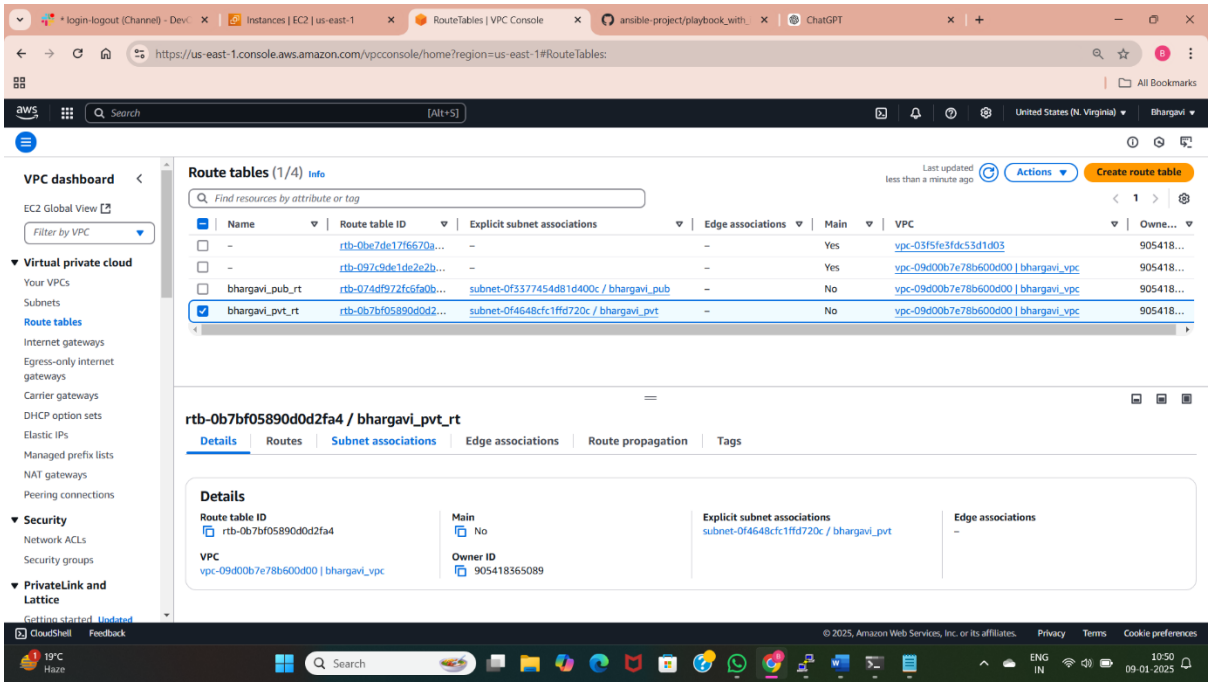
We can also navigate to the subnets section in VPC and check with the creation of public and private subnets with the subnet id's, availability zone and cidr block.



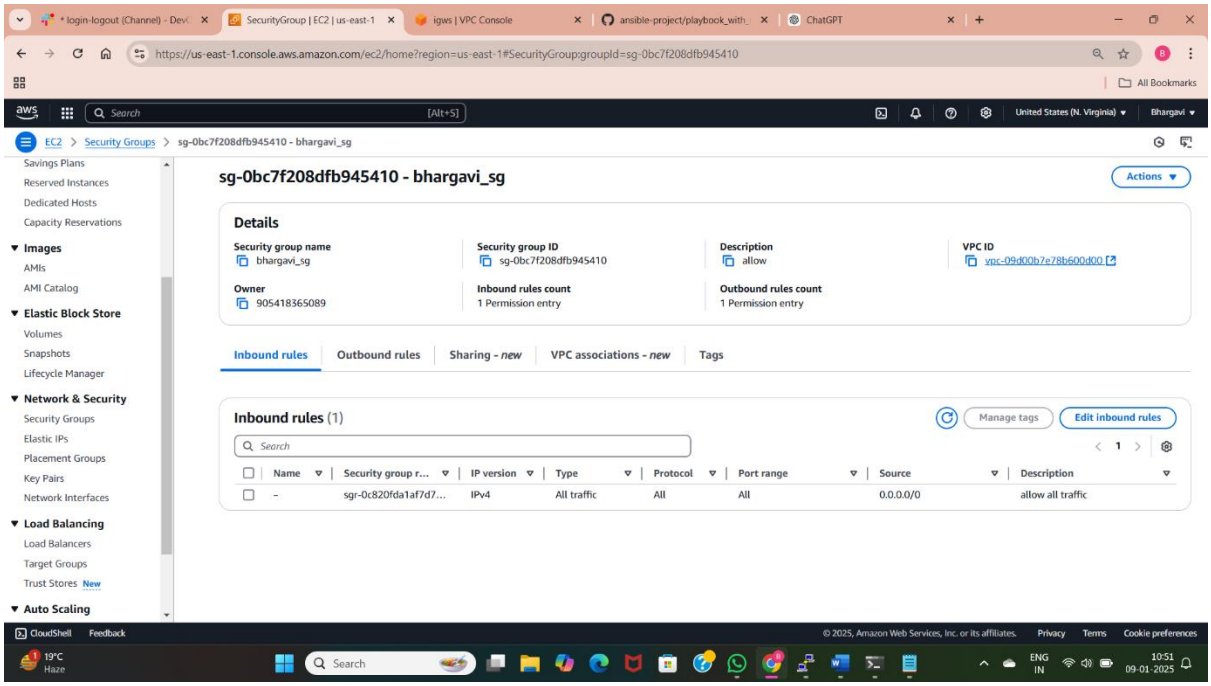
We can also navigate to the subnets section in VPC and check with the creation of public and private subnets with the subnet id's, availability zone and cidr block.



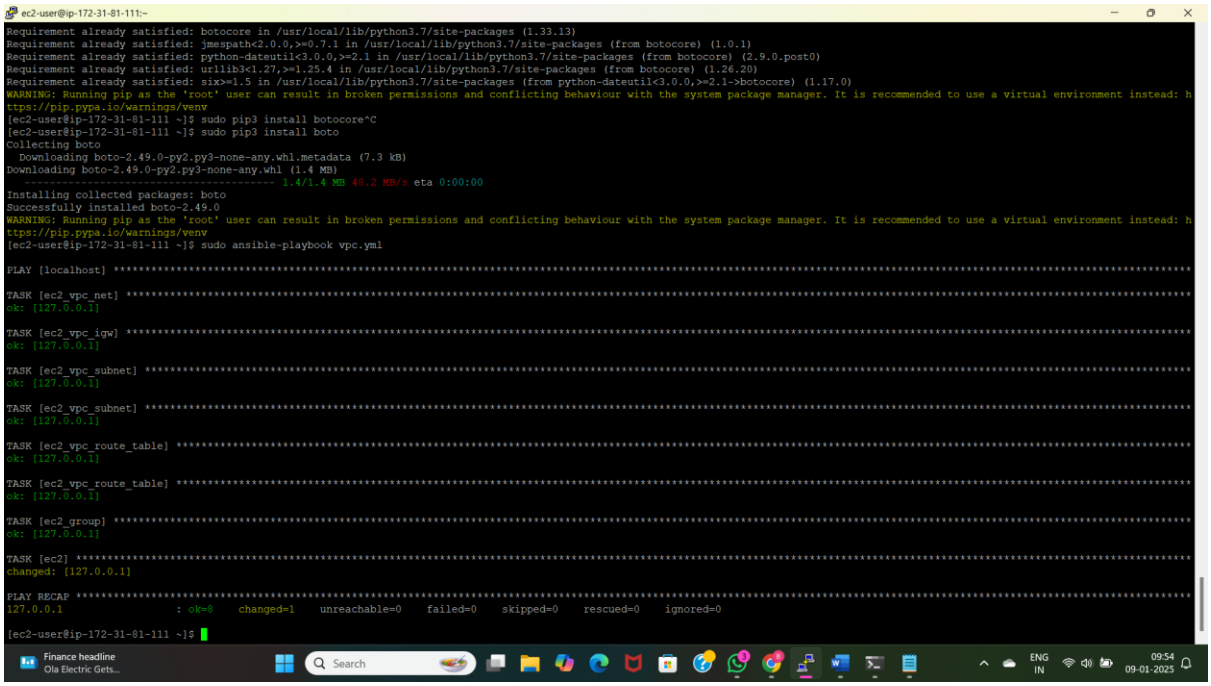
We can also navigate to the route tables section in VPC and check with the creation of public and private route tables with the route table id's, availability zone, subnet association and vpc.



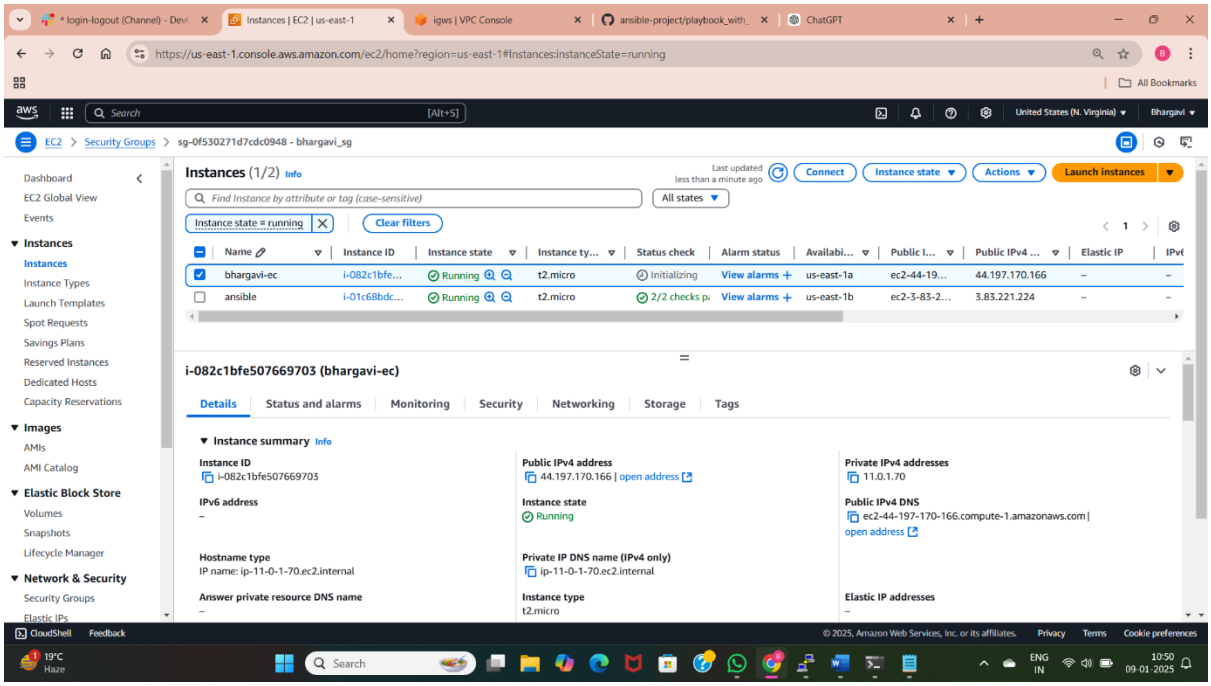
We can also navigate to the route tables section in VPC and check with the creation of public and private route tables with the route table id's, availability zone, subnet association and vpc.



This is the security group which is created using the ansible playbook. Here we can see the security group name and configuartions matches with the inline variable.



After installing boto in the server we can see ec2 instance is created successfully with ansible playbook.

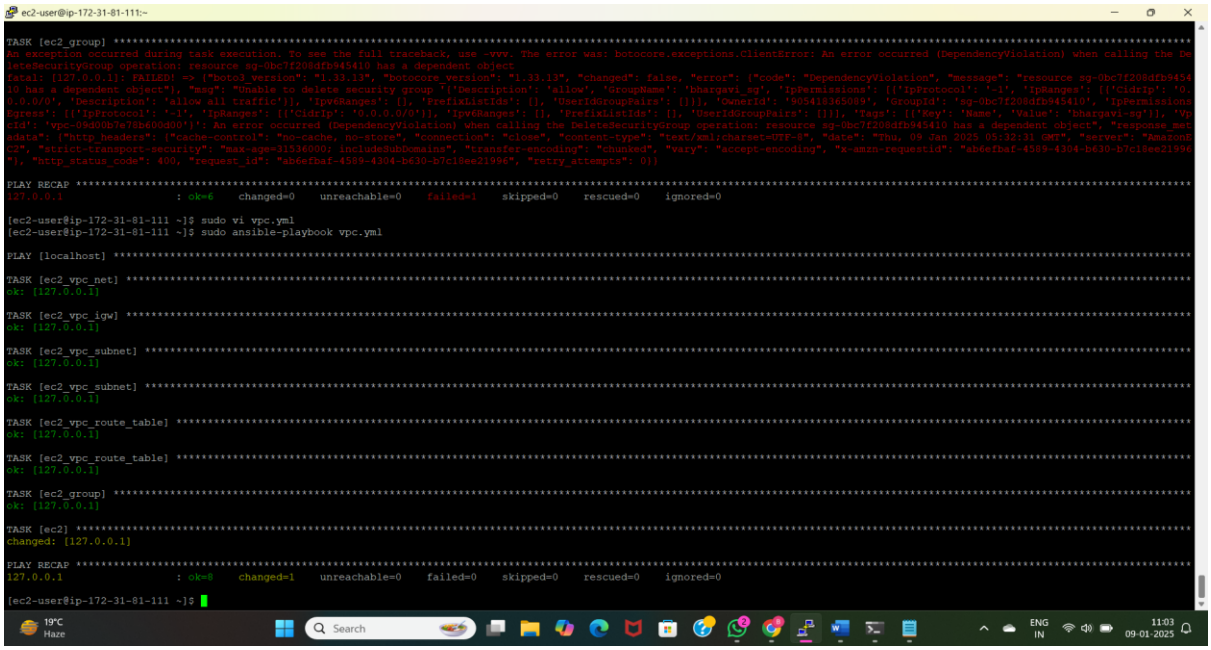


We can check with the aws console, ec2 service whether the instance is created or not. This is the new instance created using the playbook. We can check with the configuations which matches the inline variable like instance name, network settings, security group. After creating ec2 instance, security group, VPC with igw, subnet and route tables with the ansible playbook we need to delete all these using ansible playbook. For that we need to mention the specifications in the playbook.

While creating we need to create VPC first, but while deleting the services first we need to delete the ec2 instance first because it uses the security group and public subnet.

```
## tasks file for ec2-launch ##


 - ec2:
      image: ami-0ca9fb66e076a6e32
      instance_id: "i-082c1bfe507669703"
      instance_type: "{{ type }}"
      region: "{{ region }}"
      wait: yes
      count: 1
      state: absent
      vpc_subnet_id: "subnet-0f3377454d81d400c"
      assign_public_ip: yes
      group_id: "sg-0bc7f208dfb945410"
      aws_access_key: "{{ aws_access_key }}"
      aws_secret_key: "{{ aws_secret_key }}"
      instance_tags:
            Name: "{{ instance_name }}"
```

This is the yaml file for deleting the ec2 instance. Here we need to mention the instance id which we want to delete and the subnet id, security group id which is used by the instance. In place of state mention absent so it will terminates the instance.



Now run the ansible playbook using <ansible-playbbok vpc.yml> to delete the ec2 instance. After running this command we can see the ec2 instance state is changed to termination. We can see there is 1 change in the instance state.

ec2-user@ip-172-31-81-111:~

```
## create an vpc pub subnet


  - ec2_vpc_subnet:
      aws_access_key: "{{ aws_access_key }}"
      aws_secret_key: "{{ aws_secret_key }}"
      vpc_id: "vpc-09d00b7e78b600d00"
      region: "{{ region }}"
      az: "{{ pubzone }}"      # az is the availability zone
      state: absent
      cidr: "{{ pubsubnet_cidr_block }}"
      # enable public ip
      map_public: yes
      resource_tags:
        Name: "{{ pubsubnet_name }}"
    register: pubsubnet_result
```

Now we need to delete the public subnet by specifying the vpc id in the subnet. Here we need to mention the vpc id because by using that id it will delete the subnet which is created using that vpc. To delete the public subnet in place of state mention absent so it will delete the subnet.

ec2-user@ip-172-31-81-111:~

```
  ## create an vpc pvt subnet


  - ec2_vpc_subnet:
      aws_access_key: "{{ aws_access_key }}"
      aws_secret_key: "{{ aws_secret_key }}"
      vpc_id: "vpc-09d00b7e78b600d00"
      region: "{{ region }}"
      az: "{{ pvtzone }}"      # az is the availability zone
      state: absent
      cidr: "{{ pvtsubnet_cidr_block }}"
      # enable public ip
      map_public: no
      resource_tags:
        Name: "{{ pvtsubnet_name }}"
    register: pvtsubnet_result
```

Now we need to delete the private subnet by specifying the vpc id in the subnet. Here we need to mention the vpc id because by using that id it will delete the subnet which is created using that vpc. To delete the public subnet in place of state mention absent so it will delete the subnet.

ec2-user@ip-172-31-81-111:~

```
## create an vpc pub route table


    - ec2_vpc_route_table:
        aws_access_key: "{{ aws_access_key }}"
        aws_secret_key: "{{ aws_secret_key }}"
        vpc_id: "vpc-09d00b7e78b600d00"
        region: "{{ region }}"
        state: absent
        tags:
          Name: "{{ pubroute_table_name }}"
        subnets: [ "subnet-0f3377454d81d400c" ]
        # create routes
        routes:
          - dest: 0.0.0.0/0
            gateway_id: "igw-00b5040d3a74e3f3c"
      register: public_route_table
```

Now we need to delete the public route table by specifying the vpc , internet gateway id's in the subnet. Here we need to mention the vpc id and igw id because these resources were created for the specified VPC and igw is used to routing the traffic to internet. By setting the state to "absent", Ansible will ensure that the route table is deleted.

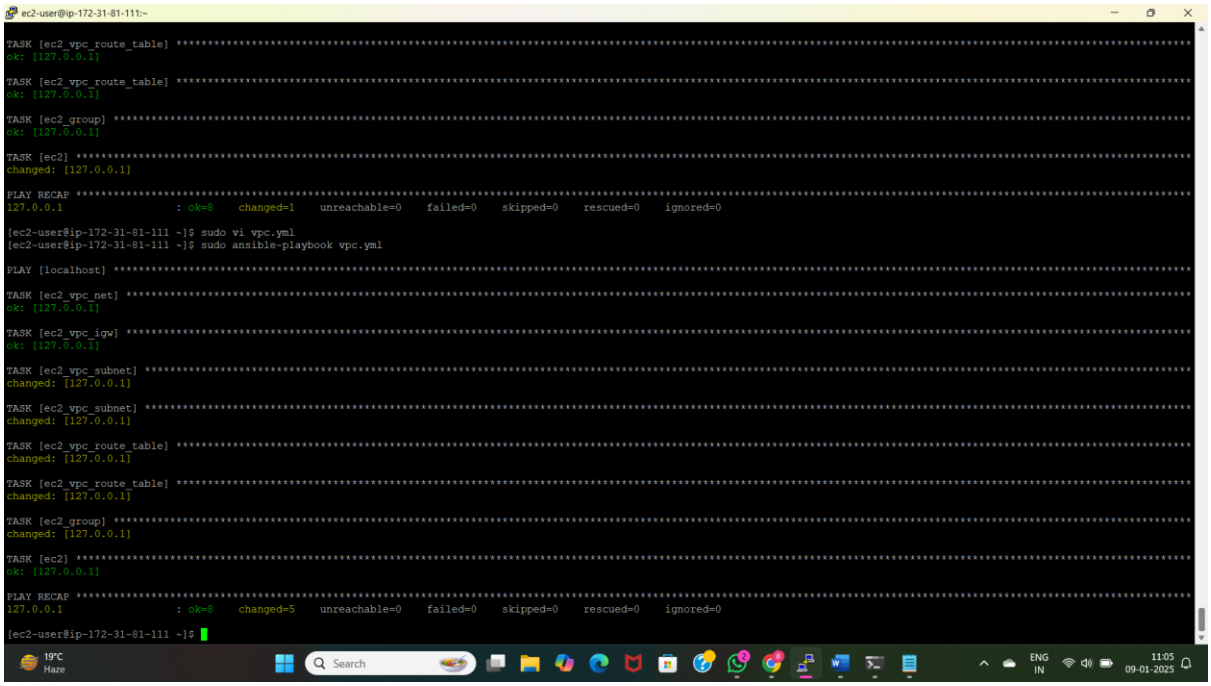
ec2-user@ip-172-31-81-111:~

```
    ## create an vpc pvt route table


    - ec2_vpc_route_table:
        aws_access_key: "{{ aws_access_key }}"
        aws_secret_key: "{{ aws_secret_key }}"
        vpc_id: "vpc-09d00b7e78b600d00"
          # igw_id: "igw-00b5040d3a74e3f3c"
        region: "{{ region }}"
        state: absent
        tags:
          Name: "{{ pvtroute_table_name }}"
        subnets: [ "subnet-0f4648cfc1ffd720c" ]
      register: private_route_table
```

Now we need to delete the private route table we need to specify the VPC ID and Subnet ID, as the private route table is associated with a specific subnet and VPC. By setting the state to "absent", Ansible will ensure that the route table is deleted.

```
## create an vpc security groups


 - ec2_group:
     aws_access_key: "{{ aws_access_key }}"
     aws_secret_key: "{{ aws_secret_key }}"
     vpc_id: "vpc-09d00b7e78b600d00"
     region: "{{ region }}"
     state: absent
     name: "{{ security_group_name }}"
     description: allow
     tags:
       Name: bhargavi-sg
     rules:
     - proto: all
       cidr_ip: 0.0.0.0/0
       rule_desc: allow all traffic
   register: security_group_results
```

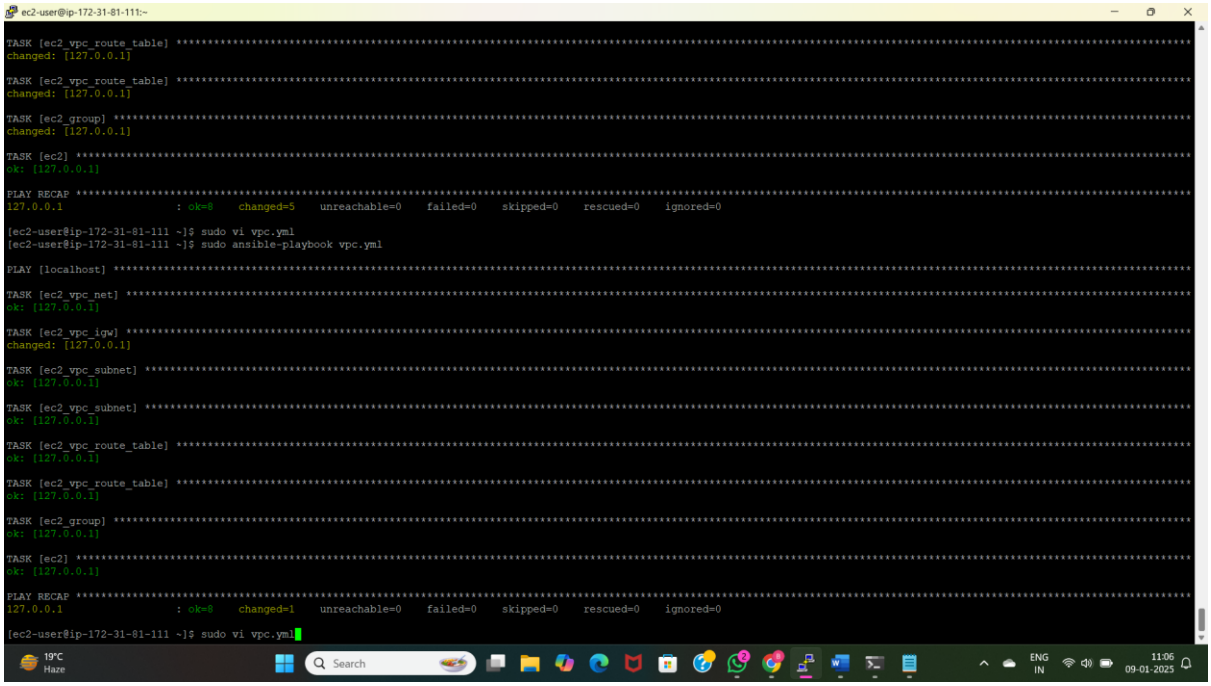
To delete security group first we need to check whether the instance is deleted or not, if instance is not deleted aws will not allow us to delete this security group because it is used by instance and it is in running state. Here we need to specify the VPC ID because security group is created using that network.



To delete the route table, subnets, and security group, simply run the Ansible playbook. The playbook will handle the deletion process for the specified resources, including the route table, public and private subnets, and security group associated with the VPC.

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

    - ec2_vpc_igw:
        aws_access_key: "{{ aws_access_key }}"
        aws_secret_key: "{{ aws_secret_key }}"
        vpc_id: "vpc-09d00b7e78b600d00"
        region: "{{ region }}"
        state: absent
        tags:
          Name: "{{ igw_name }}"
      register: igw_result
```

Now we need to delete the internet gateway. For that make whatever resources are using this internet gateway should be deleted first(like route table). In place of vpc id mention the vpc which is attached to this igw and place absent at state.

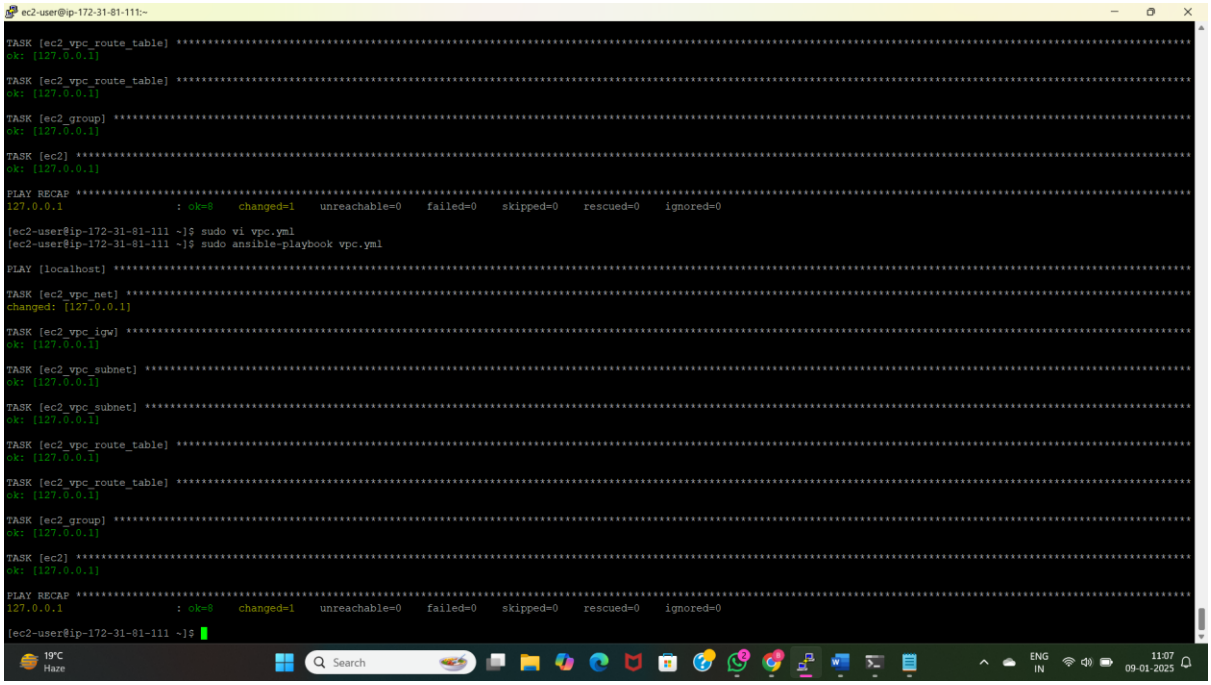


To delete the internet gateway simply run the Ansible playbook. The playbook will handle the deletion process for the internet gateway.

ec2-user@ip-172-31-81-111:~

```
### VPC Creation ####

  - ec2_vpc_net:
      aws_access_key: "{{ aws_access_key }}"
      aws_secret_key: "{{ aws_secret_key }}"
      cidr_block: "{{ vpc_cidr_block }}"
      name: "{{ vpc_name }}"
      region: "{{ region }}"
      # enable dns support
      dns_support: yes
      # enable dns hostnames
      dns_hostnames: yes
      tenancy: default
      state: absent  # to delete Vpc then replace absent instead of presnt
    register: vpc_result
```

After deleting all resources which are created by this vpc then only we can able to delete this vpc. Now to delete the vpc keep absent in place of present in the ansible playbook.



To delete the VPC simply run the Ansible playbook. The playbook will handle the deletion process for the VPC.

We can see the terminal it showing changed which means vpc is deleted automatically using playbook.

ANSIBLE_PLAYBOOK SCRIPT WITH OUTLINE VAIRABLES:

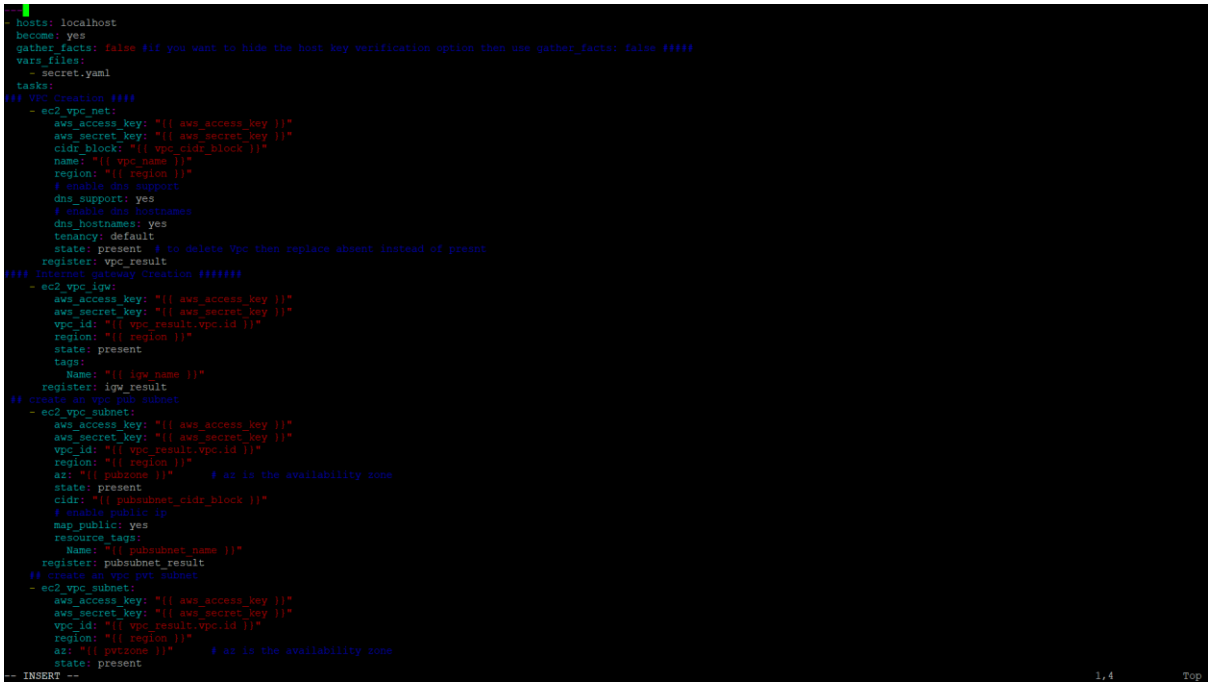
A variable that is defined outside of the playbook in a separate file and then included in the playbook or referenced during execution. This concept is commonly used to make playbooks more modular and reusable.

Ansible Vault:

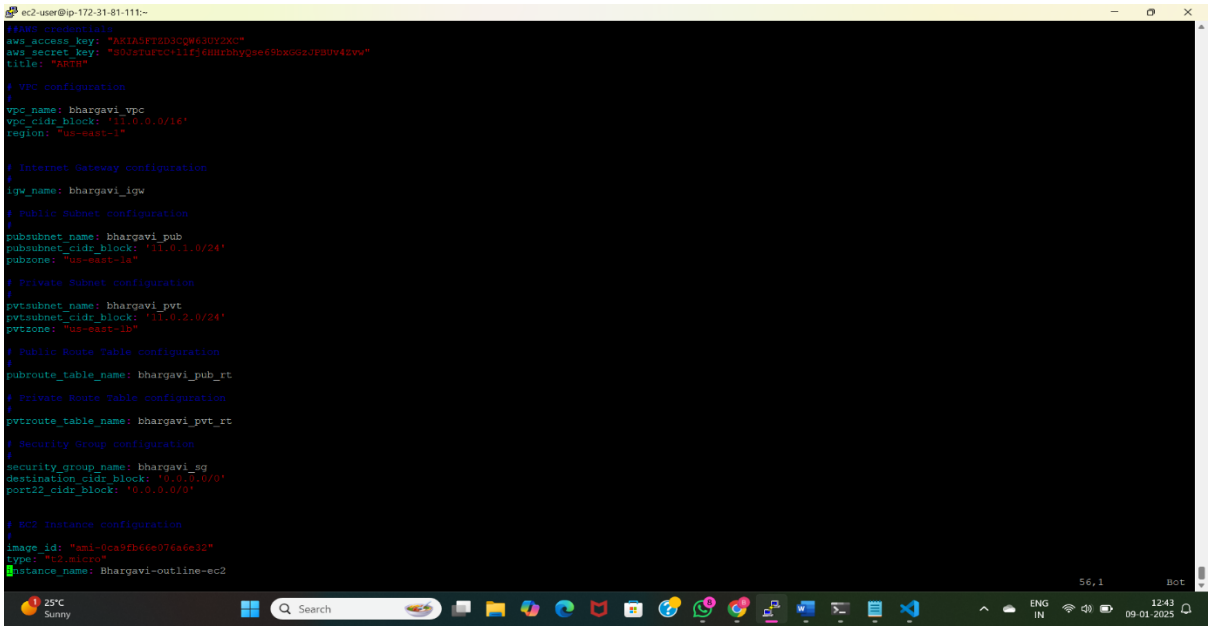
Ansible Vault is a feature in Ansible that allows you to encrypt sensitive data such as passwords, API keys, certificates, and other sensitive configuration information. This ensures that your sensitive data is not exposed when storing playbooks or variables in version control systems like Git.

Key Features of Ansible Vault

- Data Encryption: Encrypts sensitive files or strings.
- File-Level Protection: Encrypts entire files (e.g., variable files).
- String Encryption: Encrypts specific strings directly within playbooks or variable files.
- Role Integration: Easily integrates with roles and playbooks.
- Password Management: Supports password-based encryption, where the vault password is required to decrypt or use the file.



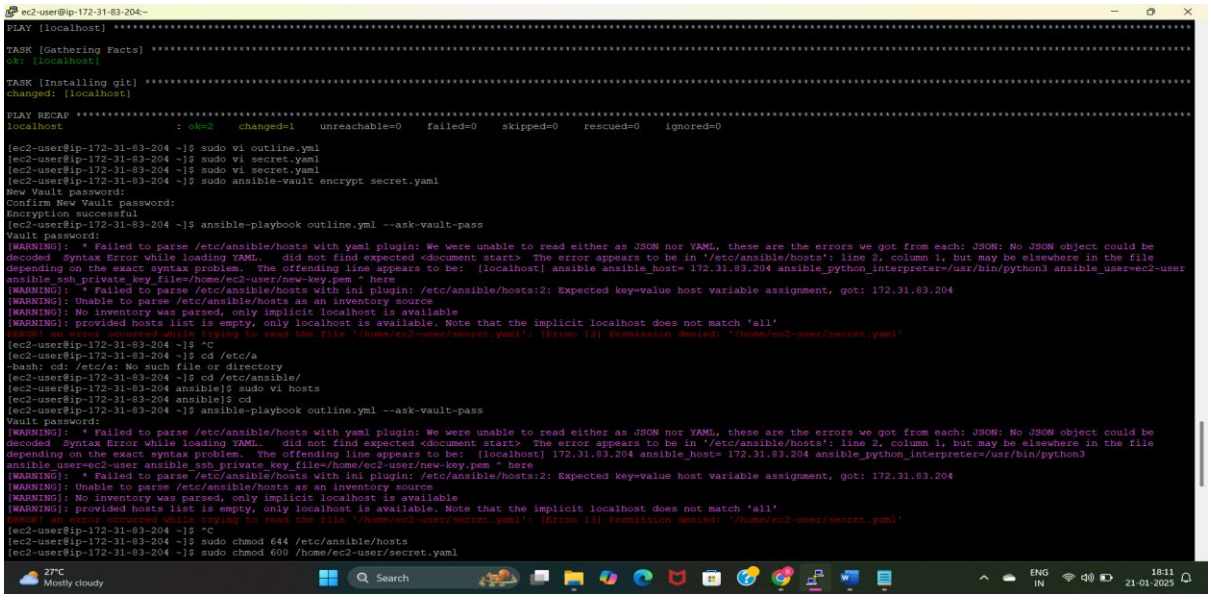
This is the outline.yml playbook which creates aws resources using ansible playbook.



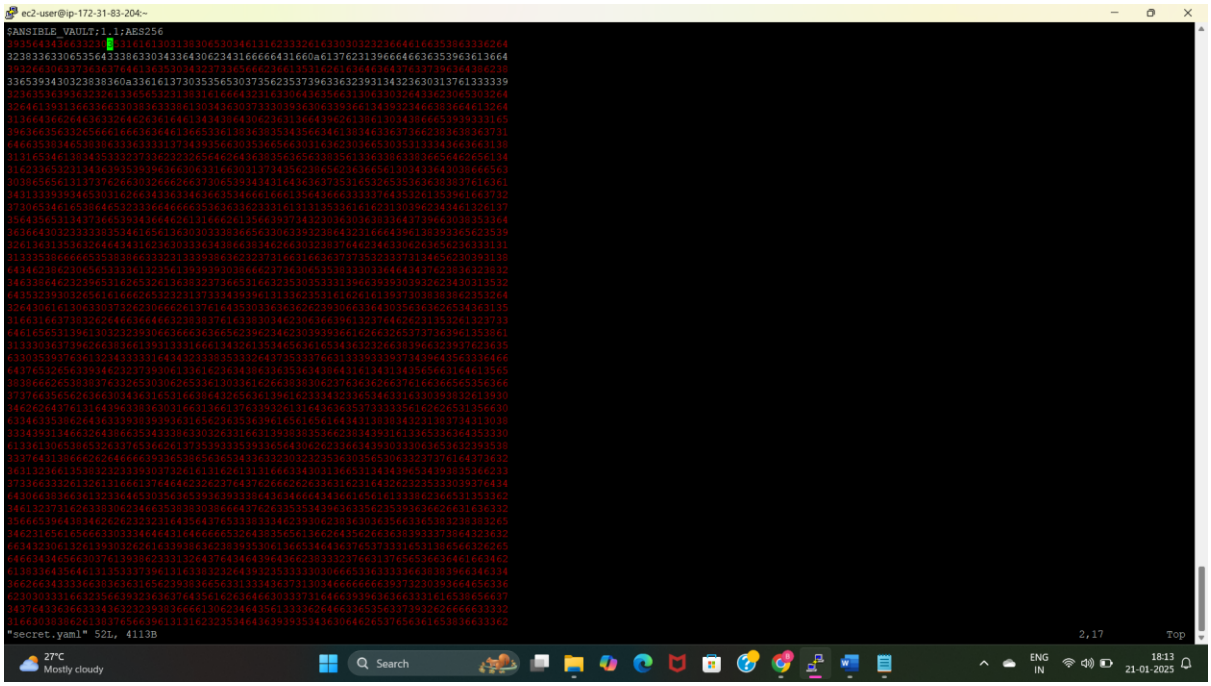
This is the outline variable file which is written in a secrets.yml file. It is not included in the actual playbook. But the outline.yml file references these variables. In this file am storing variables related to aws

- Aws credentials
- VPC configurations
- Internet gateway configurations
- Private route table configurations
- Security group configurations
- Ec2 instance configurations

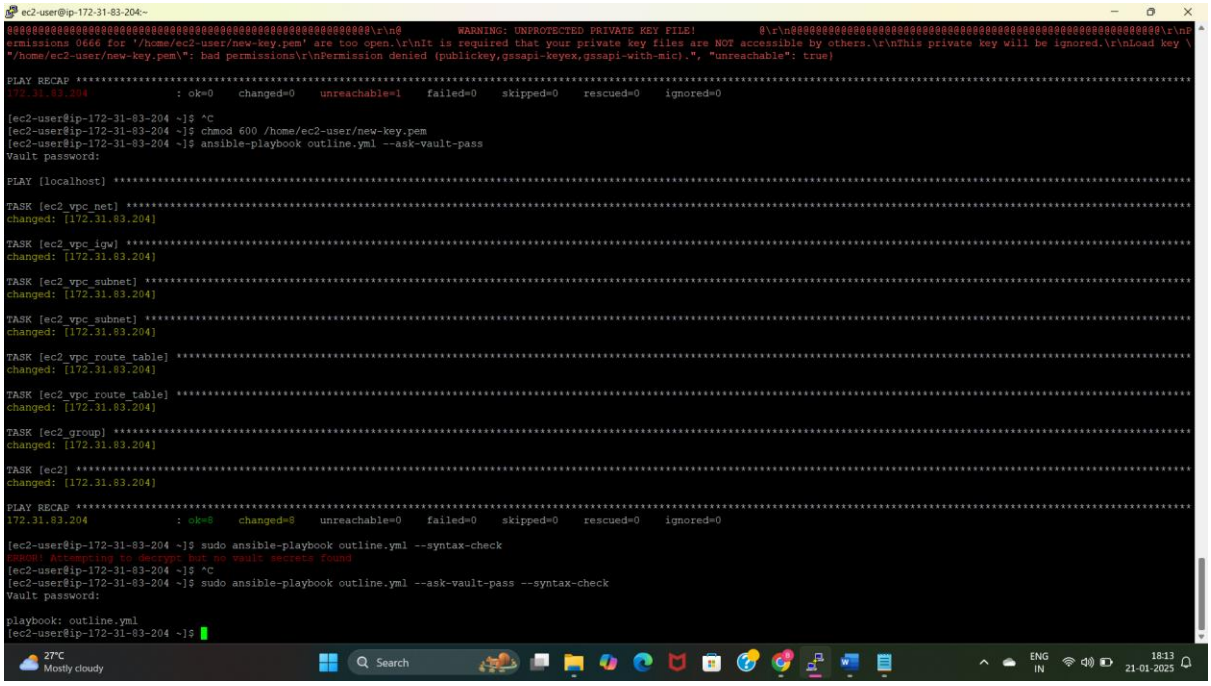
I have used a ansible playbook with the name outline.yml, if I run this playbook(ansible-playbook outline.yml) it automatically takes the variables from the above created secrets.yml file.



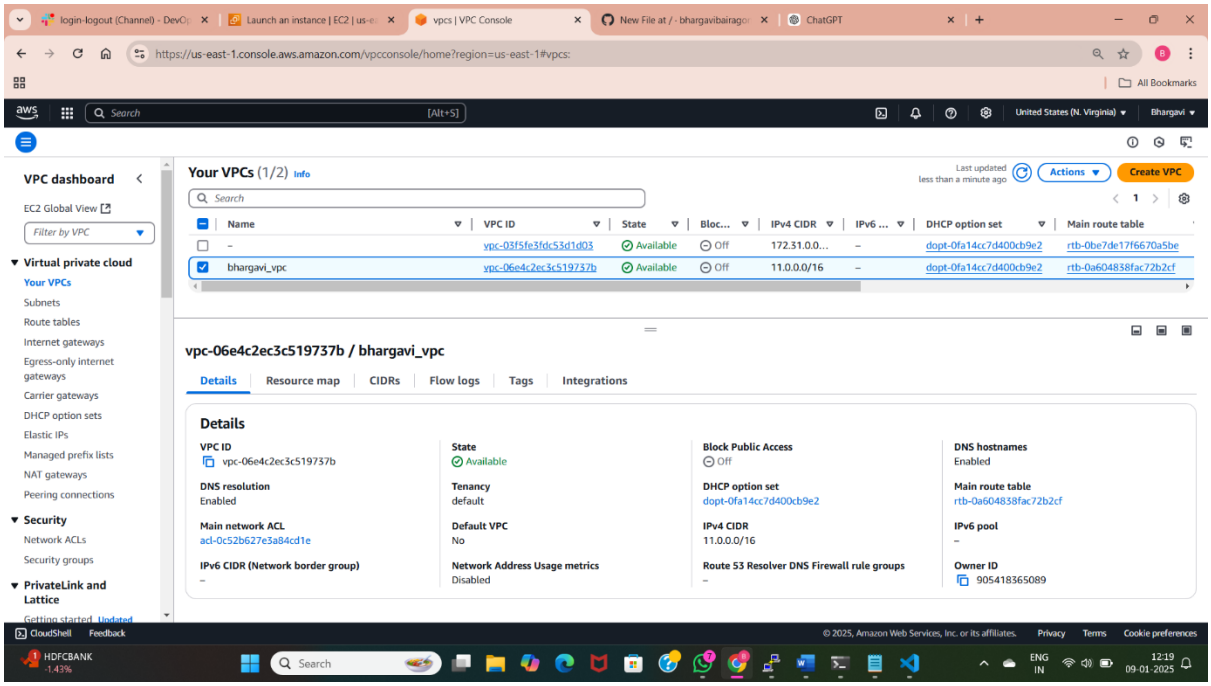
Encrypt the secret.yaml using vault “ansible-vault encrypt secret.yaml”. Run the playbook to create aws resources using “ansible-playbook outline.yml –ask-vault-pass”.



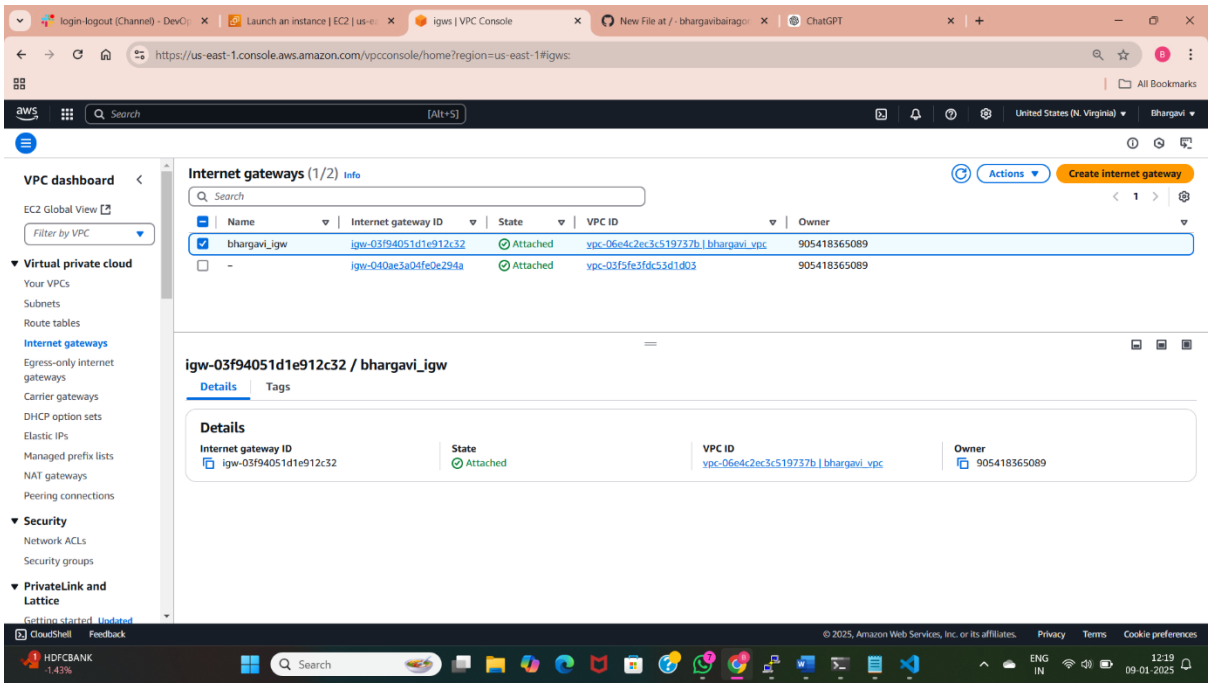
This is the encrypted secret.yaml file. We cannot see the information stored in this file.



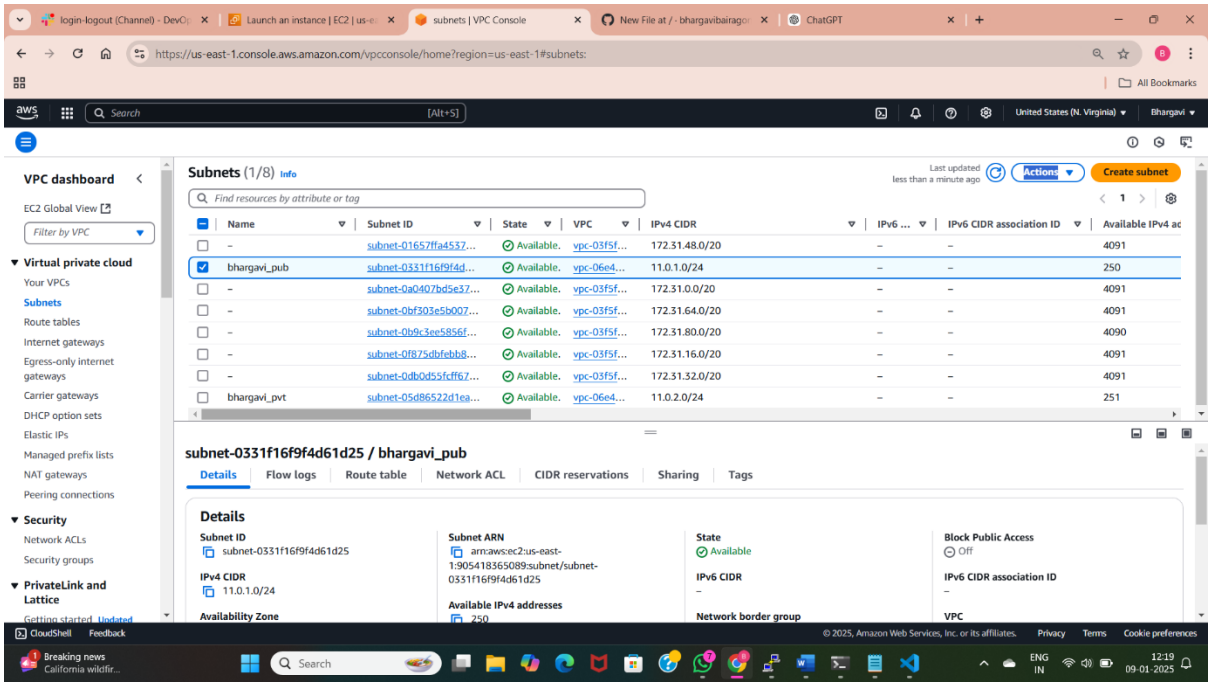
We can see that the ansible is connected to the local host, vpc, internet gateway, subnets, route table and ec2 security group, ec2 instance is created successfully.



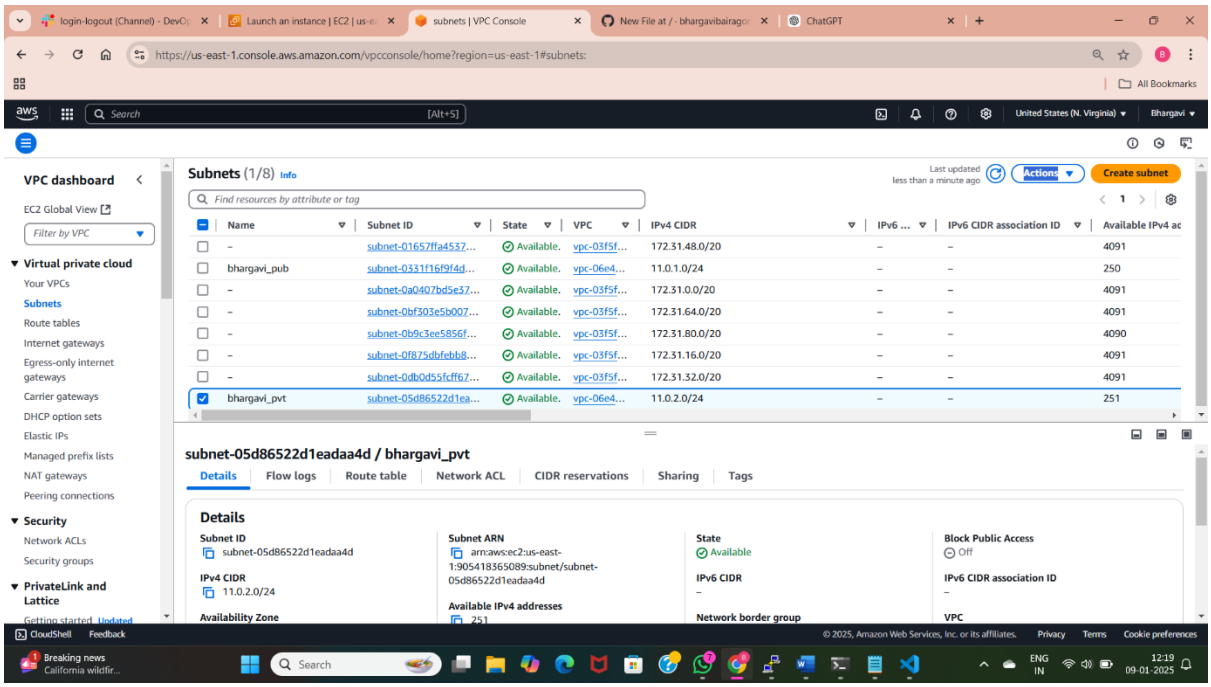
We can verify the VPC creation by checking the AWS Management Console. In the console, we can see the configuration of the VPC that was created using the Ansible playbook.



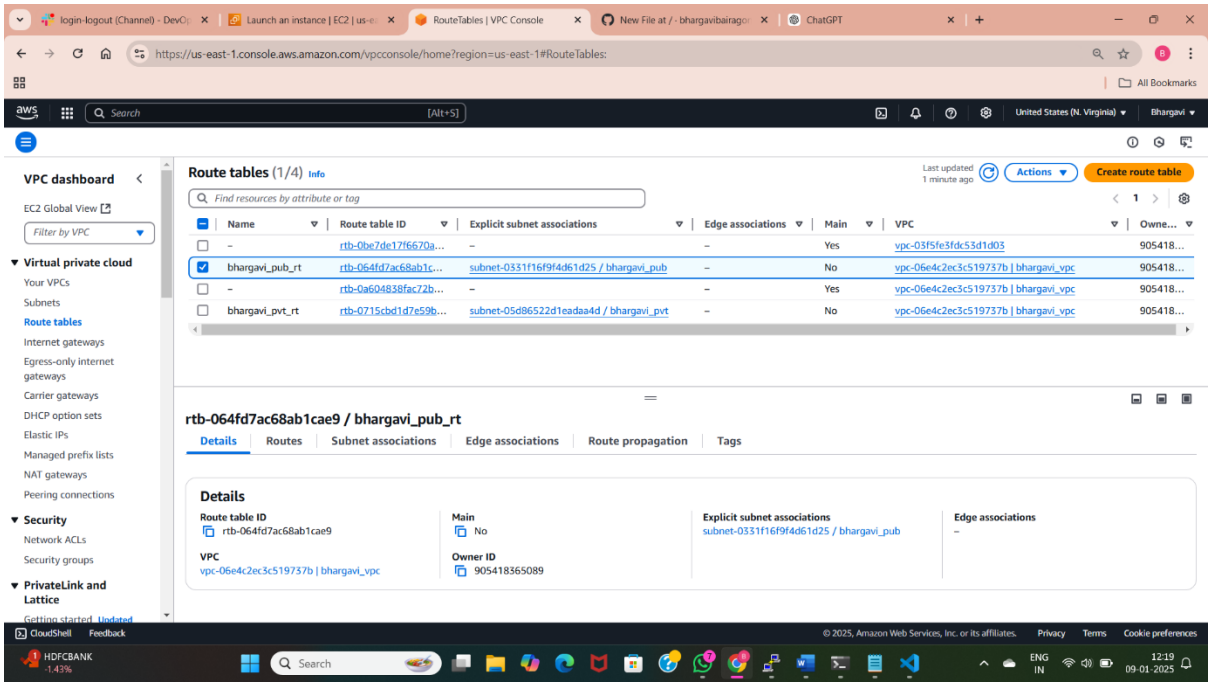
Navigate to the VPC section to check if the Internet Gateway has been created successfully and is attached to the VPC, it is created and attached to vpc.



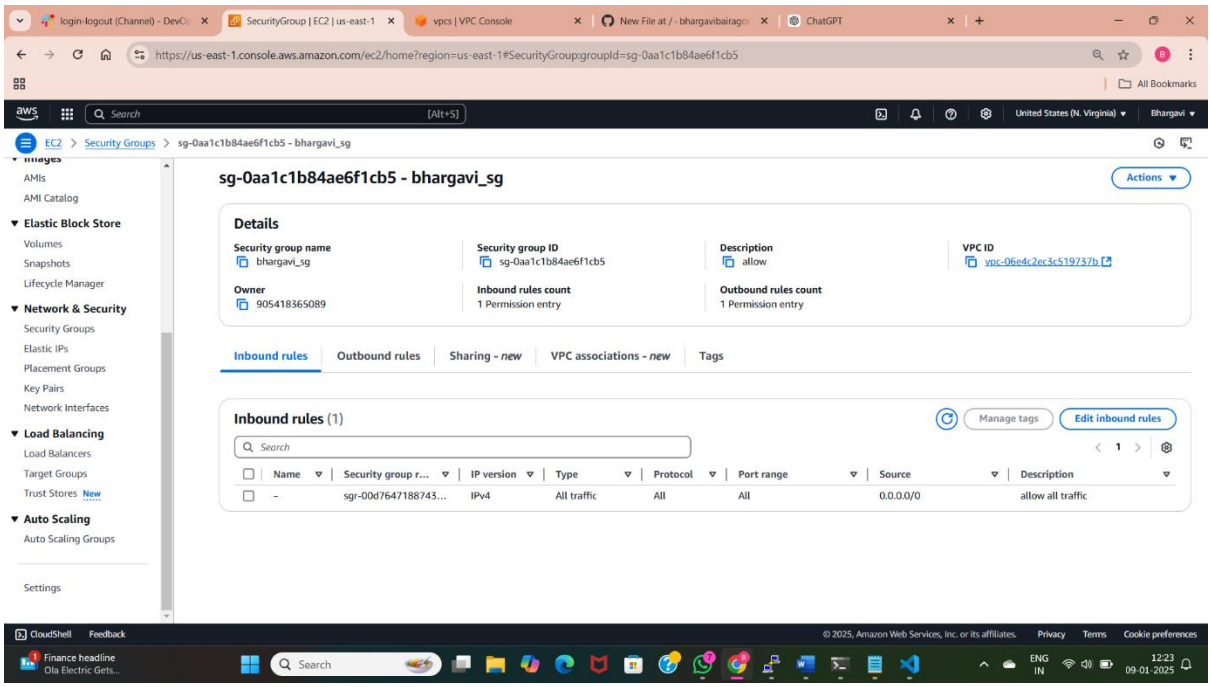
We can also navigate to the subnets section in VPC and check with the creation of public and private subnets with the subnet id's, availability zone and cidr block.



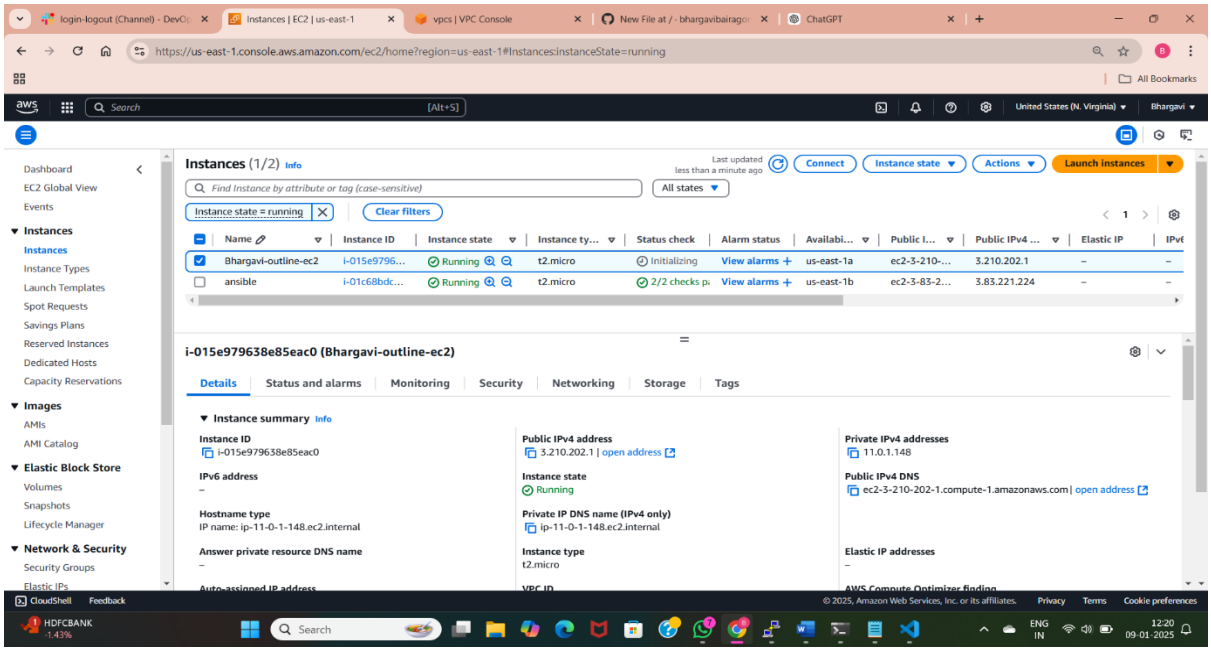
We can also navigate to the subnets section in VPC and check with the creation of public and private subnets with the subnet id's, availability zone and cidr block.



We can also navigate to the subnets section in VPC and check with the creation of public and private subnets with the subnet id's, availability zone and cidr block.



This is the security group which is created using the ansible playbook. Here we can see the security group name and configuartions matches with the inline variable.

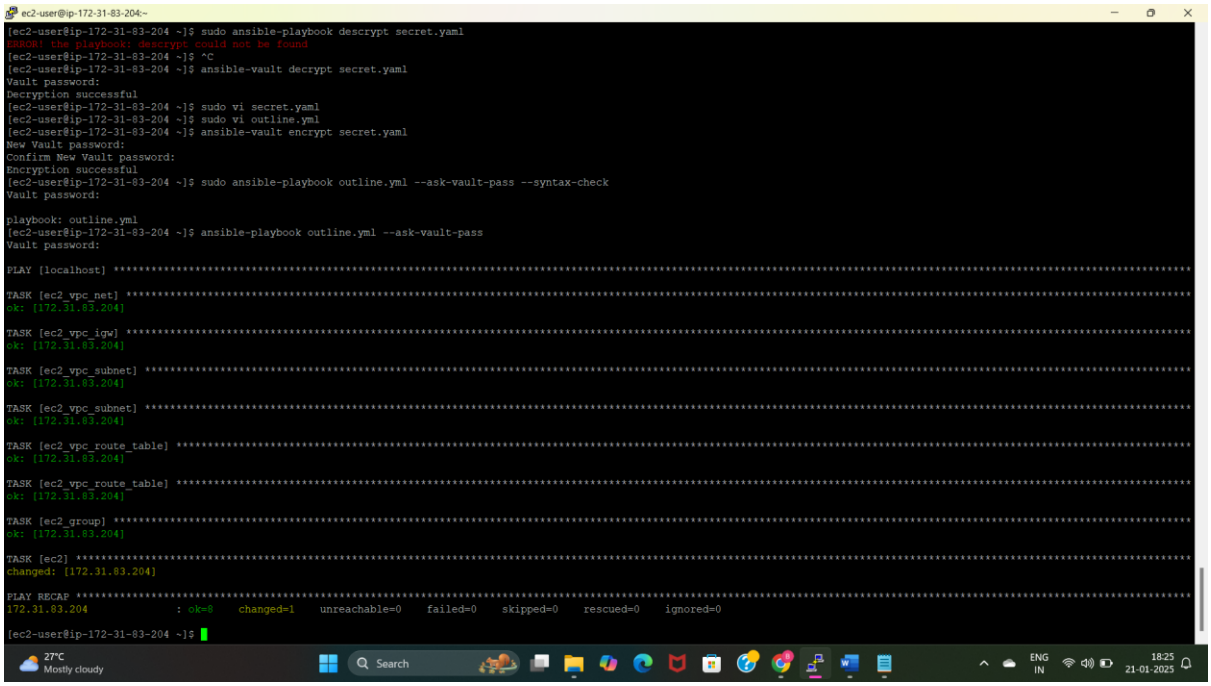


We can check with the aws console, ec2 service whether the instance is created or not. This is the new instance created using the playbook. We can check with the configuations which matches the inline variable like instance name, network settings, security group. After creating ec2 instance, security group, VPC with igw, subnet and route tables with the ansible playbook we need to delete all these using ansible playbook. For that we need to mention the specifications in the playbook.

While creating we need to create VPC first, but while deleting the services first we need to delete the ec2 instance first because it uses the security group and public subnet.

```
## tasks file for ec2-launch ##


  - ec2:
      image: ami-0ca9fb66e076a6e32
      instance_id: "i-015e979638e85eac0"
      instance_type: "{{ type }}"
      region: "{{ region }}"
      wait: yes
      count: 1
      state: absent
      vpc_subnet_id: "subnet-0331f16f9f4d61d25"
      assign_public_ip: yes
      group_id: "sg-0aa1c1b84ae6f1cb5"
      aws_access_key: "{{ aws_access_key }}"
      aws_secret_key: "{{ aws_secret_key }}"
      instance_tags:
            Name: "{{ instance_name }}"
```

This is the yaml file for deleting the ec2 instance. Here we need to mention the instance id which we want to delete and the subnet id, security group id which is used by the instance. In place of state mention absent so it will terminates the instance.



Now run the ansible playbook using <ansible-playbbok vpc.yml> to delete the ec2 instance. After running this command we can see the ec2 instance state is changed to termination. We can see there is 1 change in the instance state.

ec2-user@ip-172-31-81-111:~

```
## create an vpc pub subnet


    - ec2_vpc_subnet:
        aws_access_key: "{{ aws_access_key }}"
        aws_secret_key: "{{ aws_secret_key }}"
        vpc_id: "vpc-06e4c2ec3c519737b"
        region: "{{ region }}"
        az: "{{ pubzone }}"      # az is the availability zone
        state: absent
        cidr: "{{ pubsubnet_cidr_block }}"
        # enable public ip
        map_public: yes
        resource_tags:
          Name: "{{ pubsubnet_name }}"
      register: pubsubnet_result
```

Now we need to delete the public subnet by specifying the vpc id in the subnet. Here we need to mention the vpc id because by using that id it will delete the subnet which is created using that vpc. To delete the public subnet in place of state mention absent so it will delete the subnet.

ec2-user@ip-172-31-81-111:~

```
## create an vpc pvt subnet


- ec2_vpc_subnet:
    aws_access_key: "{{ aws_access_key }}"
    aws_secret_key: "{{ aws_secret_key }}"
    vpc_id: "vpc-06e4c2ec3c519737b"
    region: "{{ region }}"
    az: "{{ pvtzone }}"      # az is the availability zone
    state: absent
    cidr: "{{ pvtsubnet_cidr_block }}"
    # enable public ip
    map_public: no
    resource_tags:
      Name: "{{ pvtsubnet_name }}"
  register: pvtsubnet_result
```

Now we need to delete the private subnet by specifying the vpc id in the subnet. Here we need to mention the vpc id because by using that id it will delete the subnet which is created using that vpc. To delete the public subnet in place of state mention absent so it will delete the subnet.

ec2-user@ip-172-31-81-111:~

```
## create an vpc pub route table


 - ec2_vpc_route_table:
     aws_access_key: "{{ aws_access_key }}"
     aws_secret_key: "{{ aws_secret_key }}"
     vpc_id: "vpc-06e4c2ec3c519737b"
     region: "{{ region }}"
     state: absent
     tags:
       Name: "{{ pubroute_table_name }}"
     subnets: [ "subnet-0331f16f9f4d61d25" ]
     # create routes
     routes:
       - dest: 0.0.0.0/0
         gateway_id: "igw-03f94051d1e912c32"
   register: public_route_table
```

Now we need to delete the public route table by specifying the vpc , internet gateway id's in the subnet. Here we need to mention the vpc id and igw id because these resources were created for the specified VPC and igw is used to routing the traffic to internet. By setting the state to "absent", Ansible will ensure that the route table is deleted.

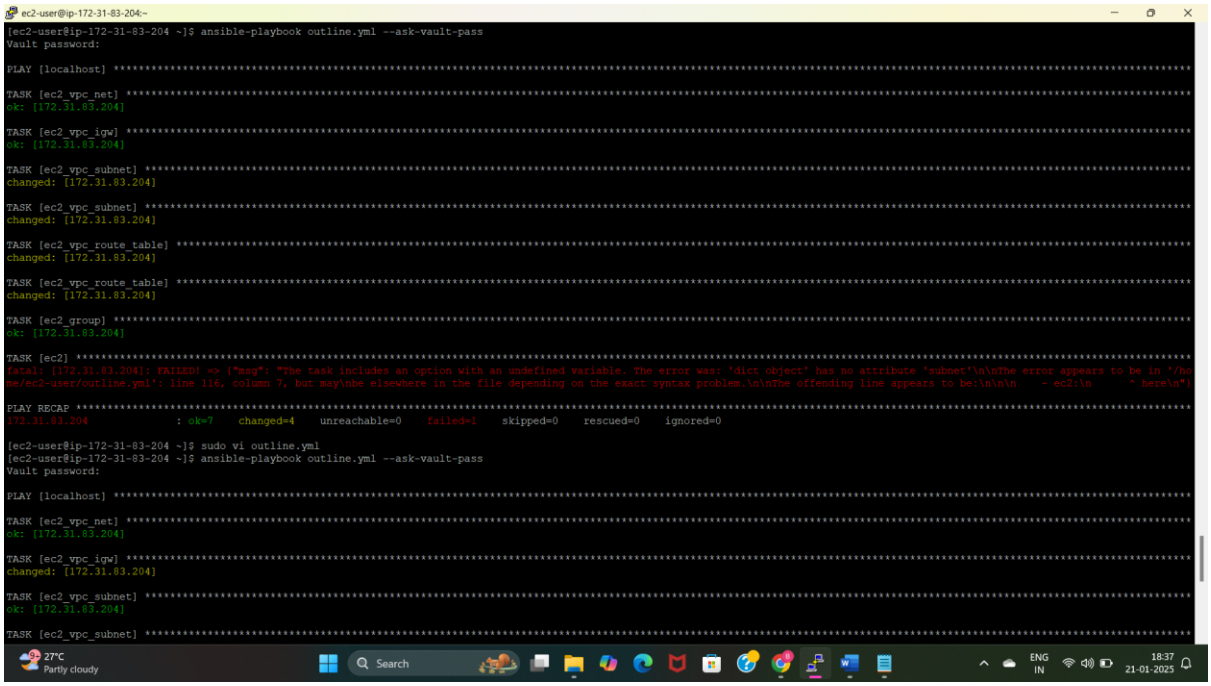
ec2-user@ip-172-31-81-111:~

```
## create an vpc pvt route table


- ec2_vpc_route_table:
    aws_access_key: "{{ aws_access_key }}"
    aws_secret_key: "{{ aws_secret_key }}"
    vpc_id: "vpc-06e4c2ec3c519737b"
    region: "{{ region }}"
    state: absent
    tags:
      Name: "{{ pvtroute_table_name }}"
    subnets: [ "subnet-05d86522d1eadaa4d" ]
  register: private_route_table
```

Now we need to delete the private route table we need to specify the VPC ID and Subnet ID, as the private route table is associated with a specific subnet and VPC. By setting the state to "absent", Ansible will ensure that the route table is deleted.

```
## create an vpc security groups


- ec2_group:
    aws_access_key: "{{ aws_access_key }}"
    aws_secret_key: "{{ aws_secret_key }}"
    vpc_id: "vpc-06e4c2ec3c519737b"
    region: "{{ region }}"
    state: absent
    name: "{{ security_group_name }}"
    description: allow
    tags:
      Name: bhargavi-sg
    rules:
    - proto: all
      cidr_ip: 0.0.0.0/0
      rule_desc: allow all traffic
  register: security_group_results
```

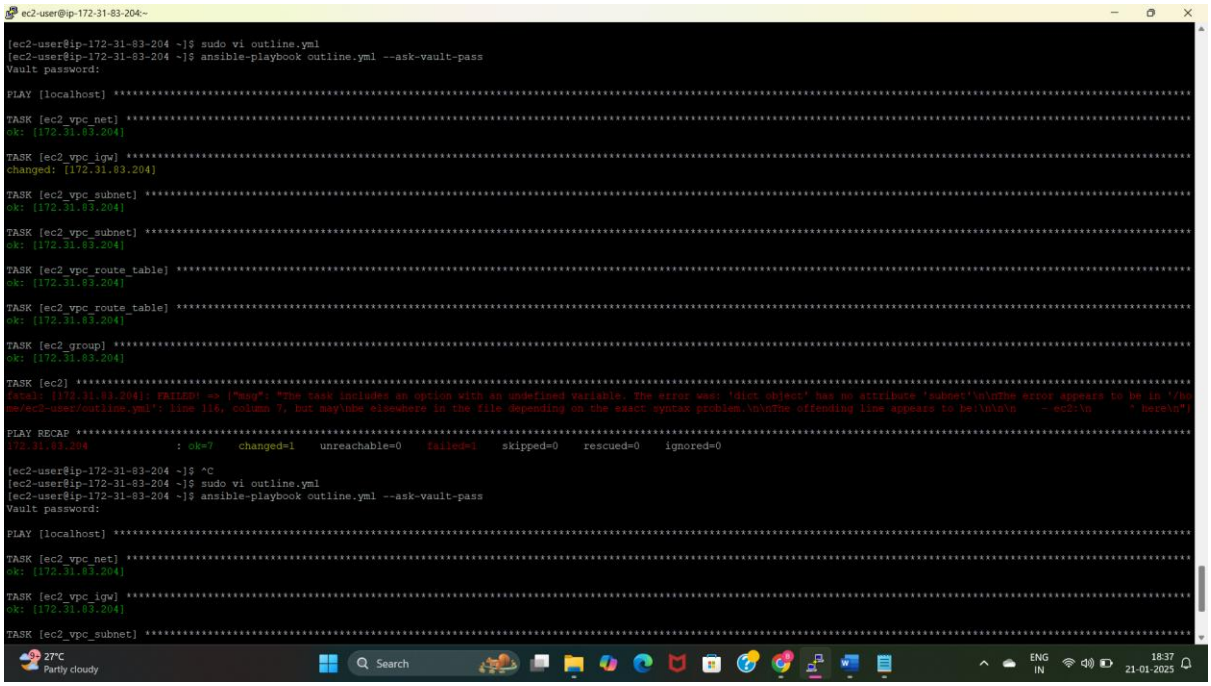
To delete security group first we need to check whether the instance is deleted or not, if instance is not deleted aws will not allow us to delete this security group because it is used by instance and it is in running state. Here we need to specify the VPC ID because security group is created using that network.



Now we need to delete the private route table we need to specify the VPC ID and Subnet ID, as the private route table is associated with a specific subnet and VPC. By setting the state to "absent", Ansible will ensure that the route table is deleted.

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

    - ec2_vpc_igw:
        aws_access_key: "{{ aws_access_key }}"
        aws_secret_key: "{{ aws_secret_key }}"
        vpc_id: "vpc-06e4c2ec3c519737b"
        region: "{{ region }}"
        state: absent
        tags:
          Name: "{{ igw_name }}"
      register: igw_result
```

Now we need to delete the internet gateway. For that make whatever resources are using this internet gateway should be deleted first(like route table). In place of vpc id mention the vpc which is attached to this igw and place absent at state.

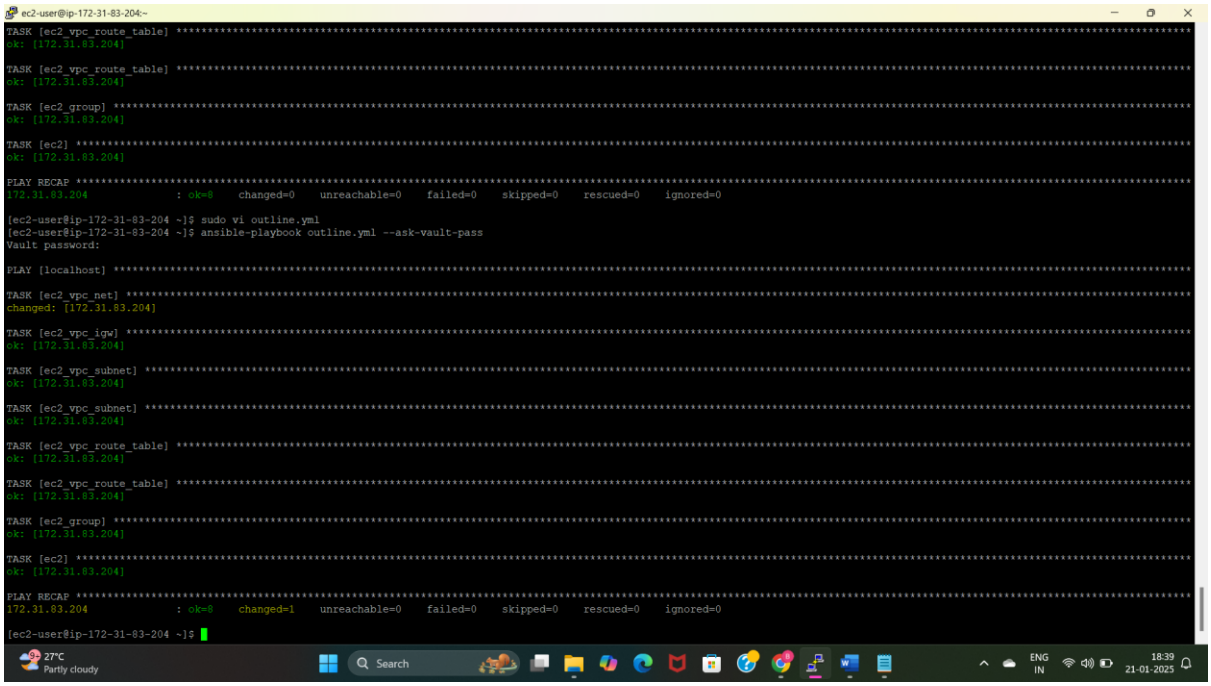


To delete the internet gateway simply run the Ansible playbook. The playbook will handle the deletion process for the internet gateway.

ec2-user@ip-172-31-81-111:~

```yaml
---
- hosts: localhost
  become: yes
  gather_facts: false #if you want to hide the host key verification option then use gather_facts: false ####
  vars_files:
    - secrets.yml
  tasks:

### VPC Creation ####

    - ec2_vpc_net:
        aws_access_key: "{{ aws_access_key }}"
        aws_secret_key: "{{ aws_secret_key }}"
        cidr_block: "{{ vpc_cidr_block }}"
        name: "{{ vpc_name }}"
        region: "{{ region }}"
        # enable dns support
        dns_support: yes
        # enable dns hostnames
        dns_hostnames: yes
        tenancy: default
        state: absent  # to delete Vpc then replace absent instead of presnt
      register: vpc_result
```

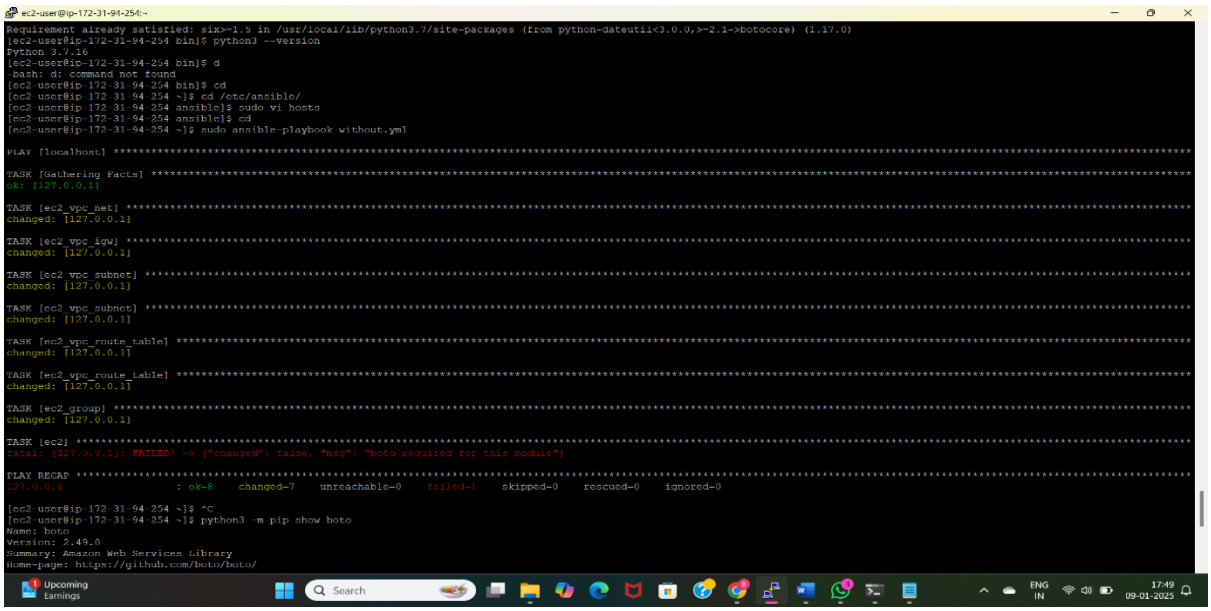
After deleting all resources which are created by this vpc then only we can able to delete this vpc. Now to delete the vpc keep absent in place of present in the ansible playbook.



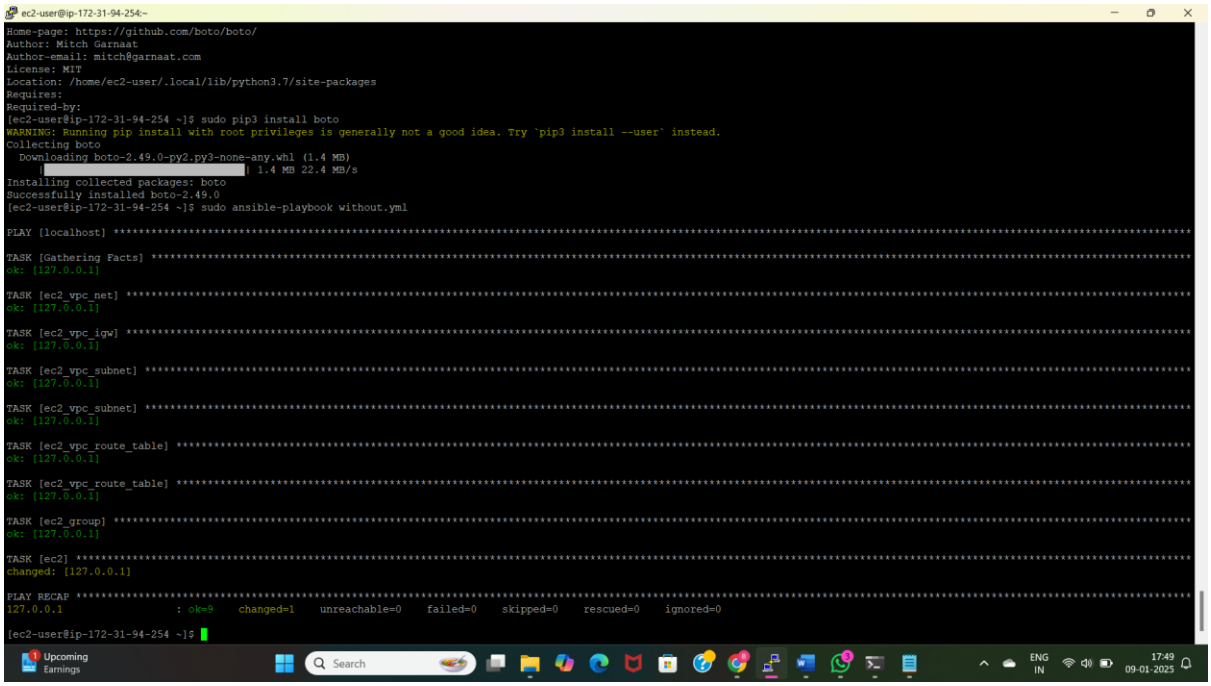
To delete the VPC simply run the Ansible playbook. The playbook will handle the deletion process for the VPC. We can see the terminal it showing changed which means vpc is deleted automatically using playbook.

ANSIBLE_PLAYBOOK SCRIPT WITHOUT VAIRABLES:

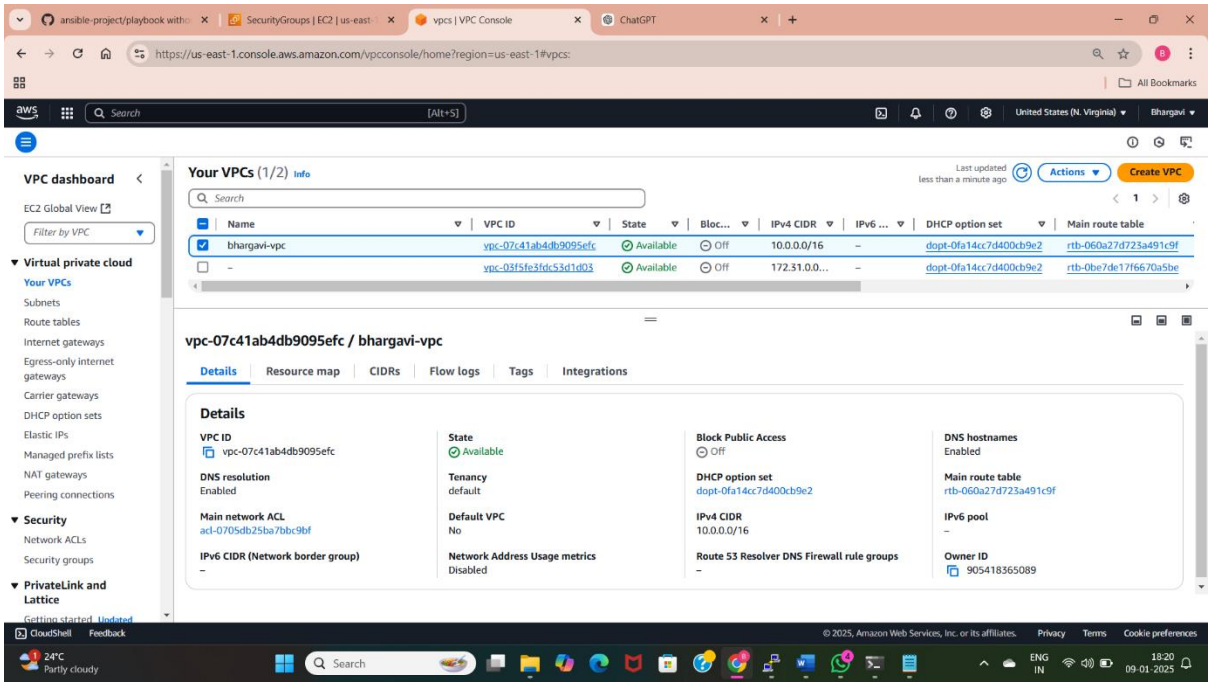
An Ansible Playbook without variables is a simple playbook that performs tasks without the use of external or predefined variables. This type of playbook is typically used for straightforward automation tasks where the configuration is static or predefined within the playbook itself, rather than being dynamic or customizable through external files or parameters.



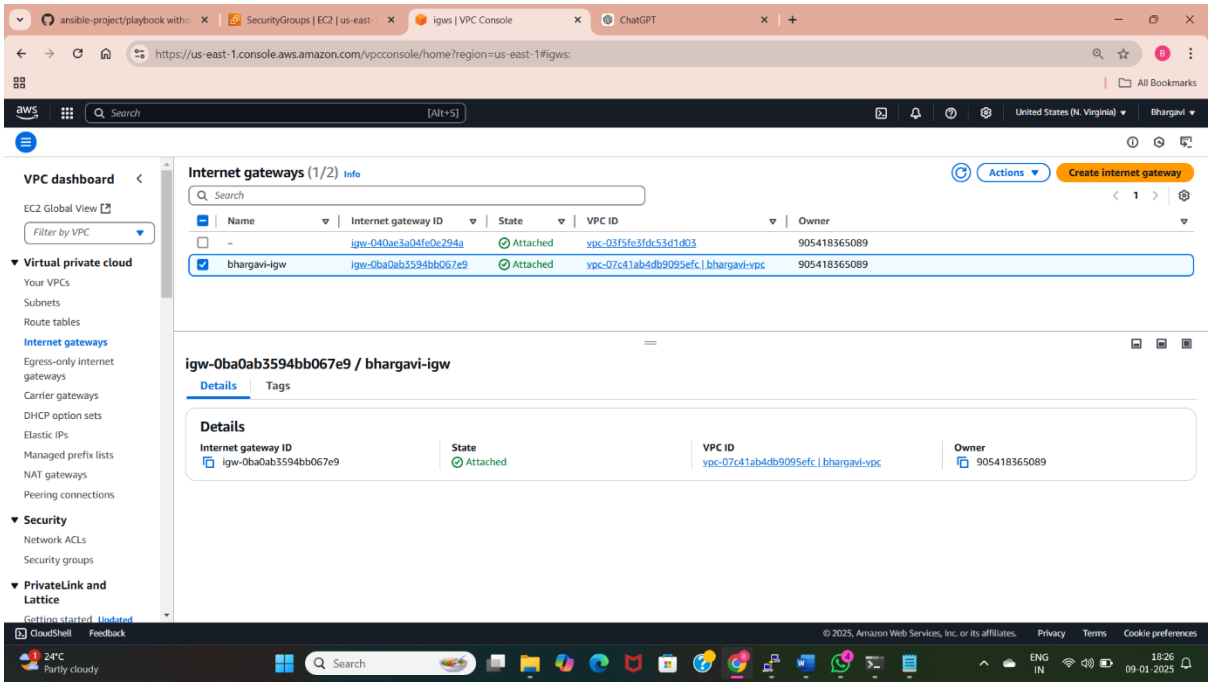
We can see that the ansible is connected to the local host, vpc, internet gateway, subnets, route table and ec2 security group is created successfully and there is an error indicating that boto is required for creating instance.



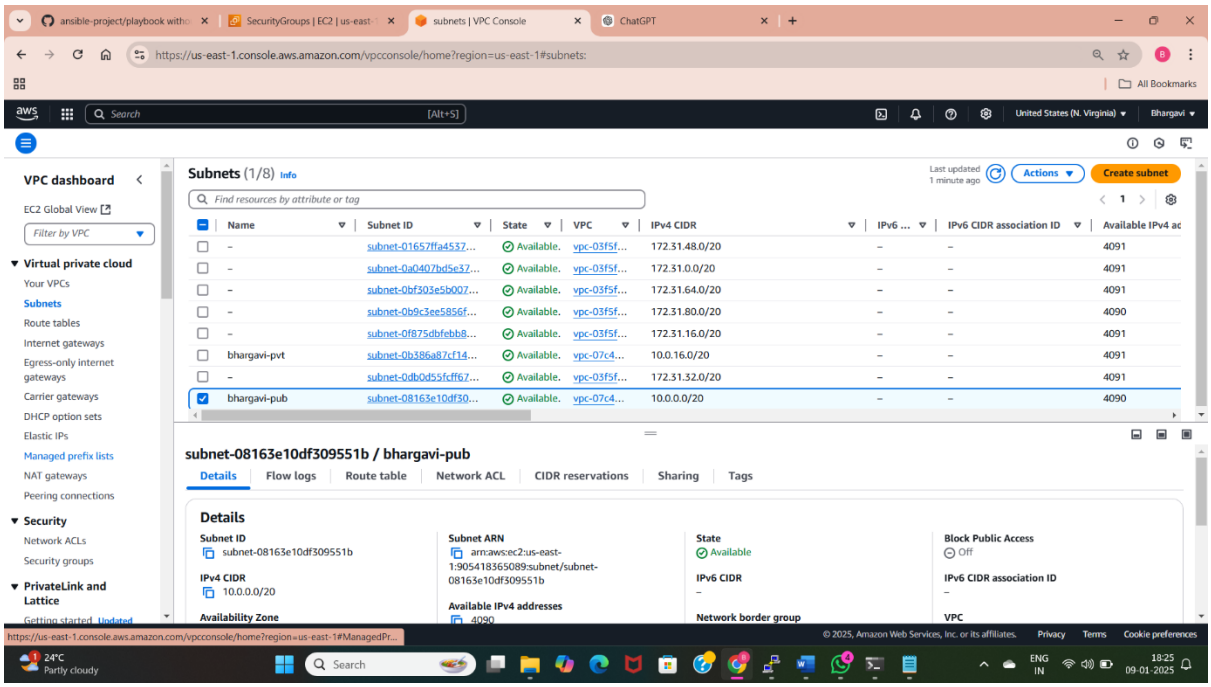
After installing boto in the server we can see ec2 instance is created successfully with ansible playbook.



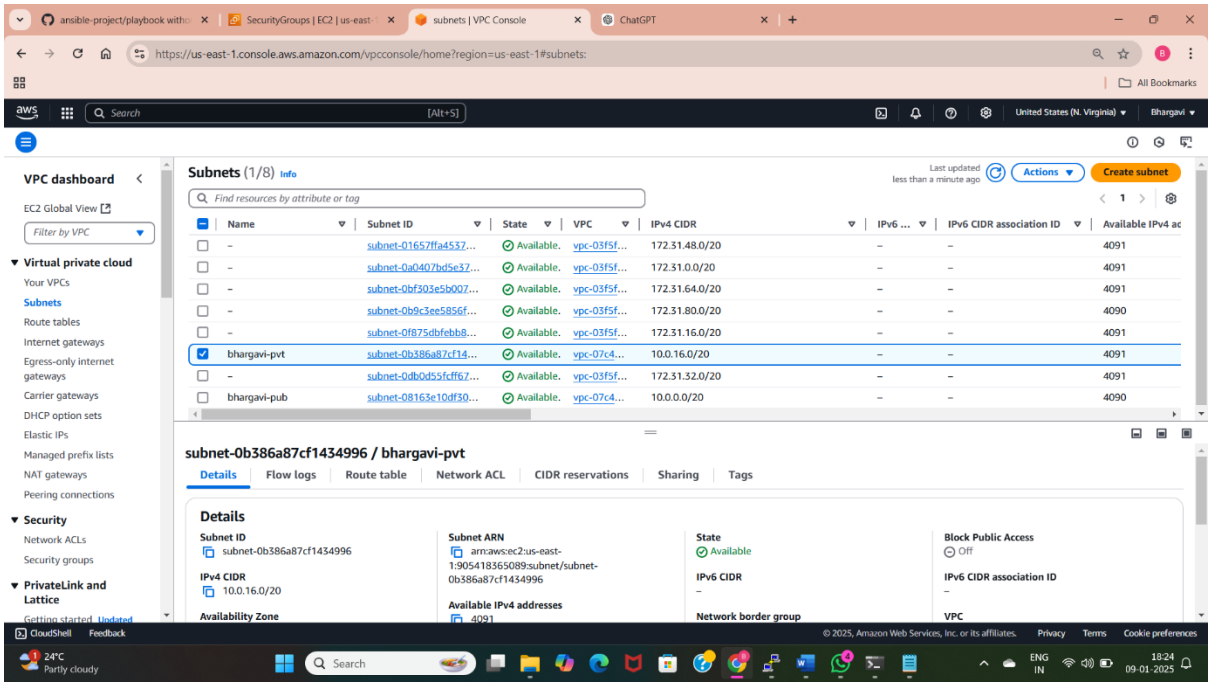
We can verify the VPC creation by checking the AWS Management Console. In the console, we can see the configuration of the VPC that was created using the Ansible playbook.



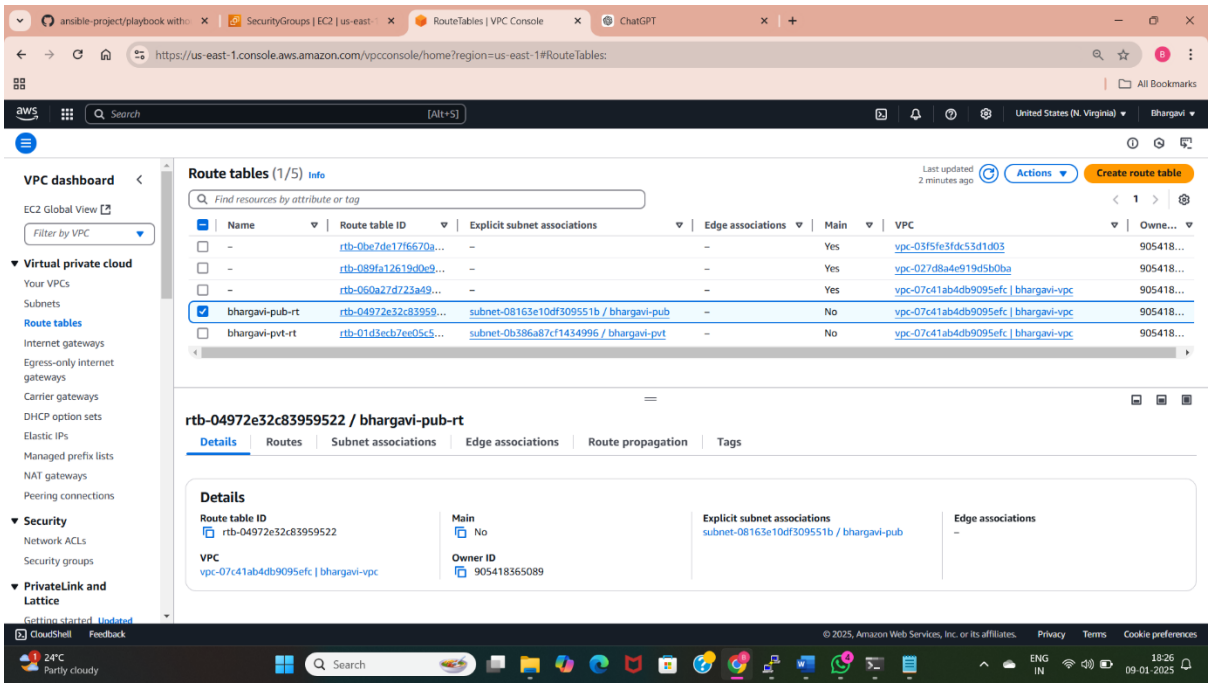
Navigate to the VPC section to check if the Internet Gateway has been created successfully and is attached to the VPC, it is created and attached to VPC.



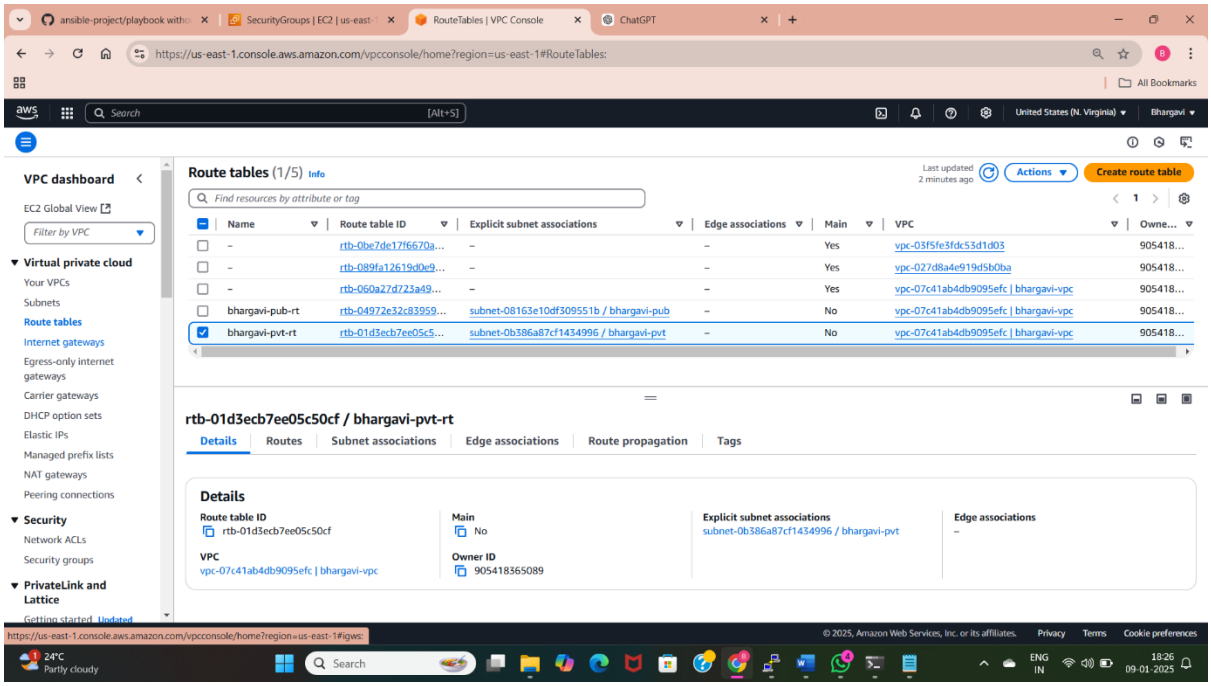
We can also navigate to the subnets section in VPC and check with the creation of public and private subnets with the subnet id's, availability zone and cidr block.



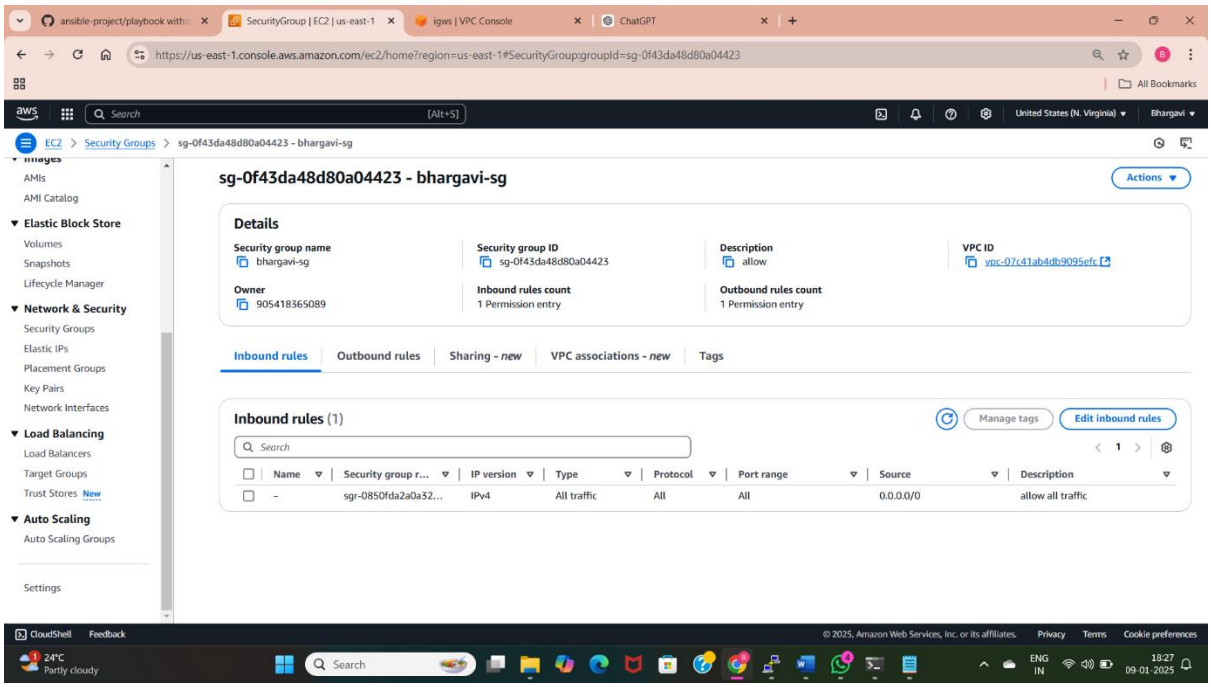
We can also navigate to the subnets section in VPC and check with the creation of public and private subnets with the subnet id's, availability zone and cidr block.



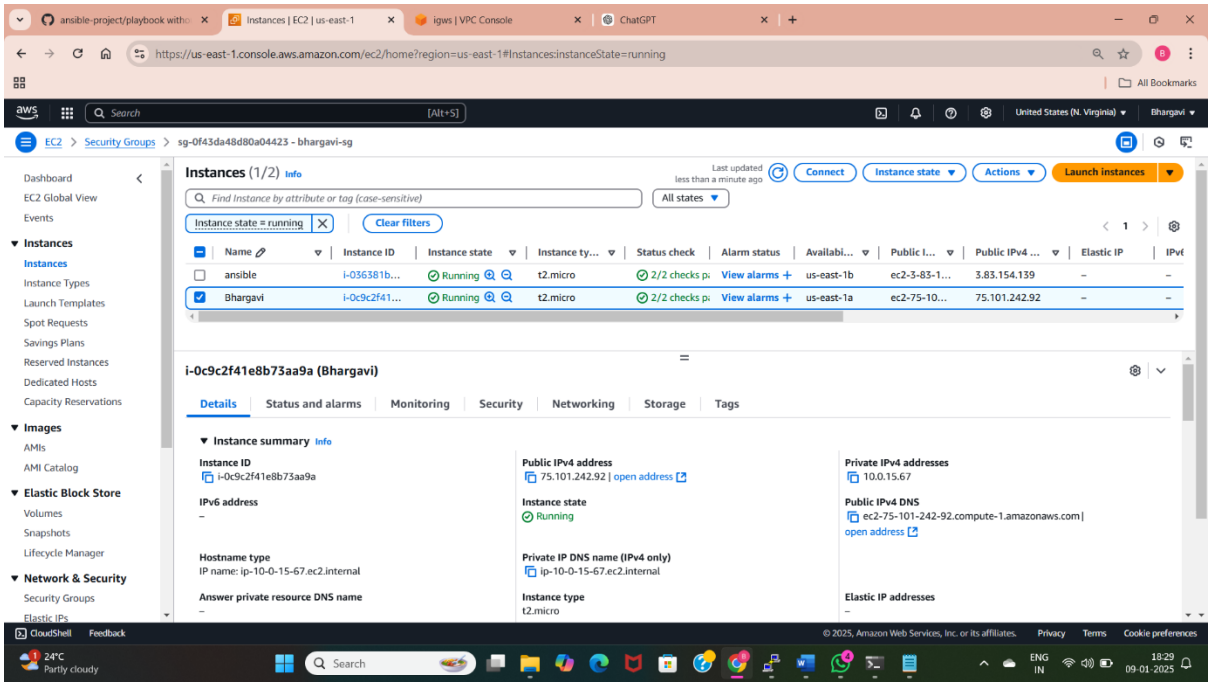
We can also navigate to the subnets section in VPC and check with the creation of public and private subnets with the subnet id's, availability zone and cidr block.



We can also navigate to the route tables section in VPC and check with the creation of public and private route tables with the route table id's, availability zone, subnet association and vpc.



This is the security group which is created using the ansible playbook. Here we can see the security group name and configuartions matches with the inline variable.



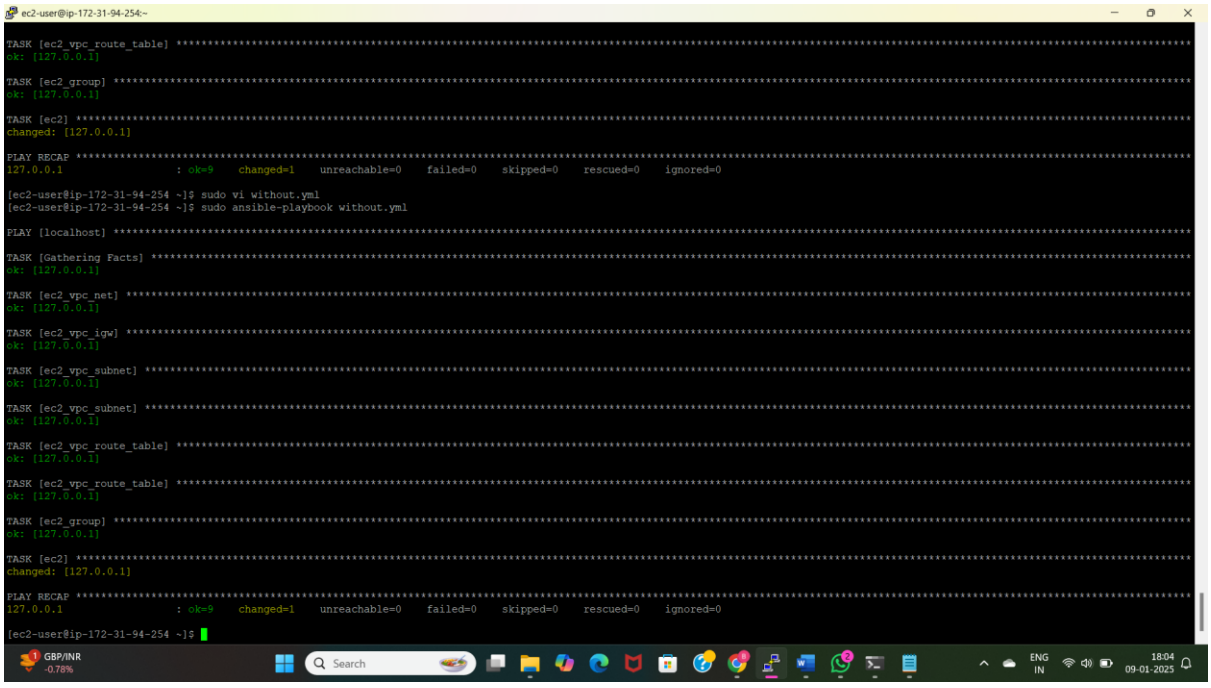
We can check with the aws console, ec2 service whether the instance is created or not. This is the new instance created using the playbook. We can check with the configuations which matches the inline variable like instance name, network settings, security group. After creating ec2 instance, security group, VPC with igw, subnet and route tables with the ansible playbook we need to delete all these using ansible playbook. For that we need to mention the specifications in the playbook.

While creating we need to create VPC first, but while deleting the services first we need to delete the ec2 instance first because it uses the security group and public subnet.

```
[ec2-user@ip-172-31-94-254 ~]$ sudo cat without.yml
---
- hosts: localhost
  become: yes
  tasks:
    - ec2_vpc_net:
        aws_access_key: AKIA5FTZD3CQW63UY2XC
        aws_secret_key: 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
        cidr_block: 10.0.0.0/16
        name: bhargavi-vpc
        region: us-east-1
        state: absent  # to delete Vpc then replace absent instead of presnt
      register: vpc_result
```

```
## tasks file for ec2-launch ##


 - ec2:
      image: ami-0ca9fb66e076a6e32
      instance_id: i-0c9c2f41e8b73aa9a
      instance_type: t2.micro
      region: us-east-1
      wait: yes
      count: 1
      state: absent
      vpc_subnet_id: "subnet-08163e10df309551b"
      assign_public_ip: yes
      group_id: "sg-0f43da48d80a04423"
      aws_access_key: AKIA5FTZD3CQW63UY2XC
      aws_secret_key: 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
      instance_tags:
              Name: Bhargavi
```

This is the yaml file for deleting the ec2 instance. Here we need to mention the instance id which we want to delete and the subnet id, security group id which is used by the instance. In place of state mention absent so it will terminates the instance.



Now run the ansible playbook using <ansible-playbbok vpc.yml> to delete the ec2 instance. After running this command we can see the ec2 instance state is changed to termination. We can see there is 1 change in the instance state.

```
## create an vpc pub subnet


  - ec2_vpc_subnet:
      aws_access_key: AKIA5FTZD3CQW63UY2XC
      aws_secret_key: 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
      vpc_id: "vpc-07c41ab4db9095efc"
      region: us-east-1
      az: us-east-1a      # az is the availability zone
      state: absent
      cidr: 10.0.0.0/20
      # enable public ip
      map_public: yes
      resource_tags:
        Name: bhargavi-pub
    register: pubsubnet_result
```

Now we need to delete the public subnet by specifying the vpc id in the subnet. Here we need to mention the vpc id because by using that id it will delete the subnet which is created using that vpc. To delete the public subnet in place of state mention absent so it will delete the subnet.

```
## create an vpc pvt subnet


- ec2_vpc_subnet:
    aws_access_key: AKIA5FTZD3CQW63UY2XC
    aws_secret_key: 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
    vpc_id: "vpc-07c41ab4db9095efc"
    region: us-east-1
    az: us-east-1b      # az is the availability zone
    state: absent
    cidr: 10.0.16.0/20
    # enable public ip
    map_public: no
    resource_tags:
      Name: bhargavi-pvt
  register: pvtsubnet_result
```

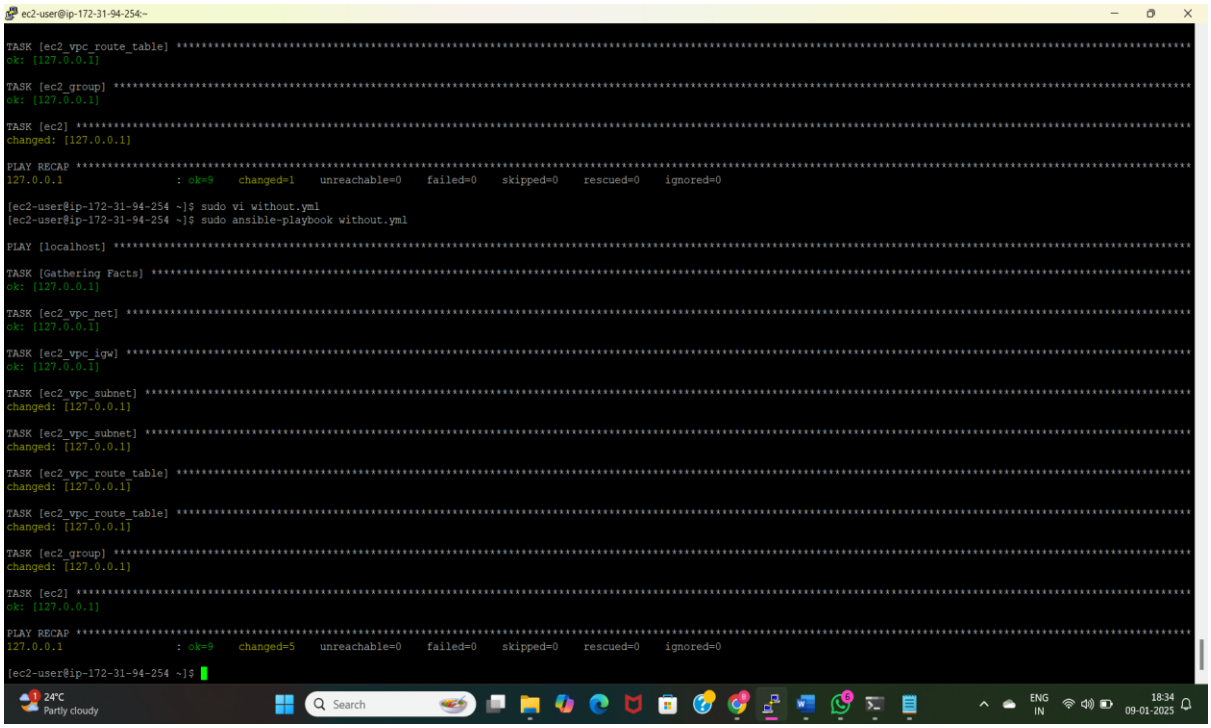
Now we need to delete the private subnet by specifying the vpc id in the subnet. Here we need to mention the vpc id because by using that id it will delete the subnet which is created using that vpc. To delete the public subnet in place of state mention absent so it will delete the subnet.

```
  - ec2_vpc_route_table:
      aws_access_key:
      aws_secret_key:
      vpc_id: "vpc-07c41ab4db9095efc"
      region: us-east-1
      state: absent
      tags:
        Name: bhargavi-pvt-rt
      subnets: [ "subnet-0b386a87cf1434996" ]
    register: private_route_table
```

Now we need to delete the private route table we need to specify the VPC ID and Subnet ID, as the private route table is associated with a specific subnet and VPC. By setting the state to "absent", Ansible will ensure that the route table is deleted.

```
## create an vpc security groups


 - ec2_group:
     aws_access_key: AKIA5FTZD3CQW63UY2XC
     aws_secret_key: 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
     vpc_id: "vpc-07c41ab4db9095efc"
     region: us-east-1
     state: absent
     name: bhargavi-sg
     description: allow
     tags:
       Name: bairagoni-sg
     rules:
     - proto: all
       cidr_ip: 0.0.0.0/0
       rule_desc: allow all traffic
   register: security_group_results
```

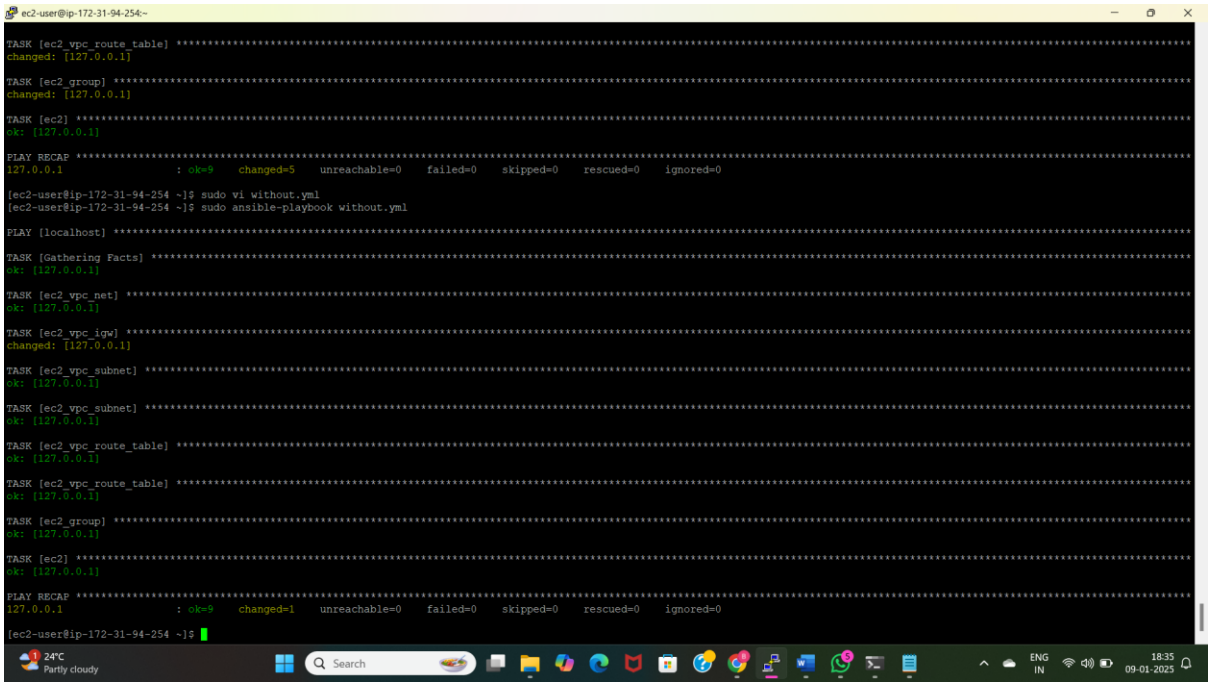
To delete security group first we need to check whether the instance is deleted or not, if instance is not deleted aws will not allow us to delete this security group because it is used by instance and it is in running state. Here we need to specify the VPC ID because security group is created using that network.



Now we need to delete the private route table we need to specify the VPC ID and Subnet ID, as the private route table is associated with a specific subnet and VPC. By setting the state to "absent", Ansible will ensure that the route table is deleted.

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

    - ec2_vpc_igw:
        aws_access_key: AKIA5FTZD3CQW63UY2XC
        aws_secret_key: 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
        vpc_id: "vpc-07c41ab4db9095efc"
        region: us-east-1
        state: absent
        tags:
          Name: bhargavi-igw
      register: igw_result
```

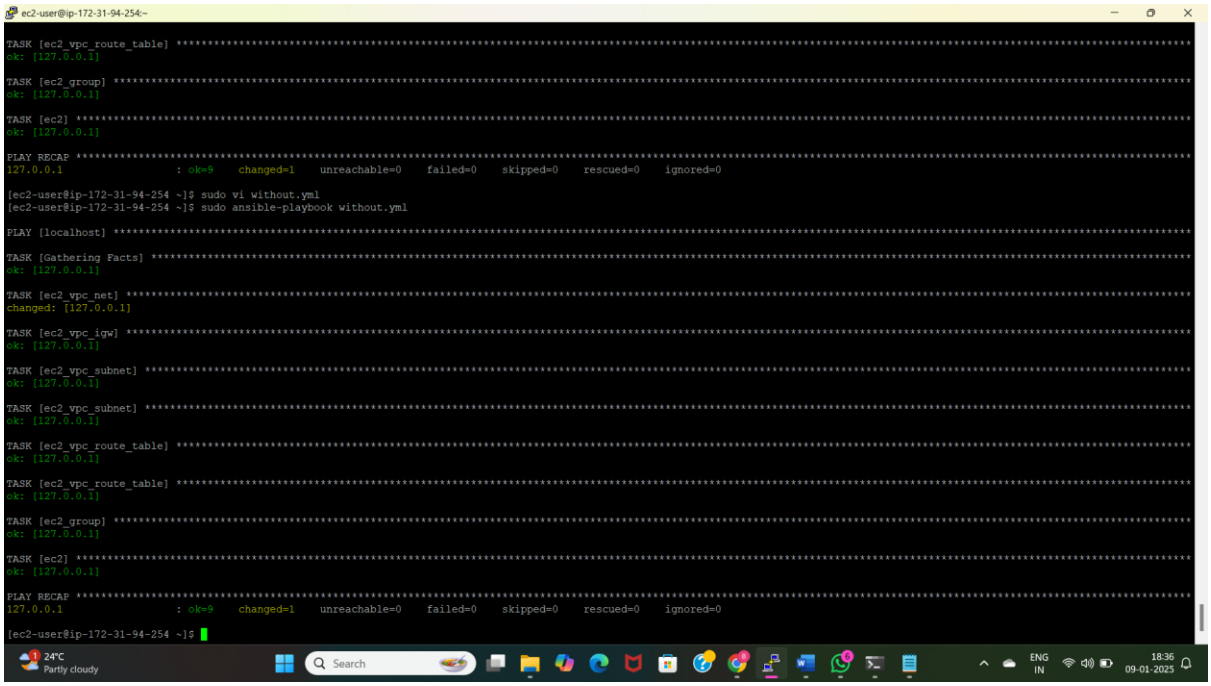
Now we need to delete the internet gateway. For that make whatever resources are using this internet gateway should be deleted first(like route table). In place of vpc id mention the vpc which is attached to this igw and place absent at state.



To delete the internet gateway simply run the Ansible playbook. The playbook will handle the deletion process for the internet gateway.

```
[ec2-user@ip-172-31-94-254 ~]$ sudo cat without.yml
---
- hosts: localhost
  become: yes
  tasks:
    - ec2_vpc_net:
        aws_access_key: AKIA5FTZD3CQW63UY2XC
        aws_secret_key: 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
        cidr_block: 10.0.0.0/16
        name: bhargavi-vpc
        region: us-east-1
        state: absent  # to delete Vpc then replace absent instead of presnt
      register: vpc_result
```

After deleting all resources which are created by this vpc then only we can able to delete this vpc. Now to delete the vpc keep absent in place of present in the ansible playbook.



To delete the VPC simply run the Ansible playbook. The playbook will handle the deletion process for the VPC. We can see the terminal it showing changed which means vpc is deleted automatically using playbook.