

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.

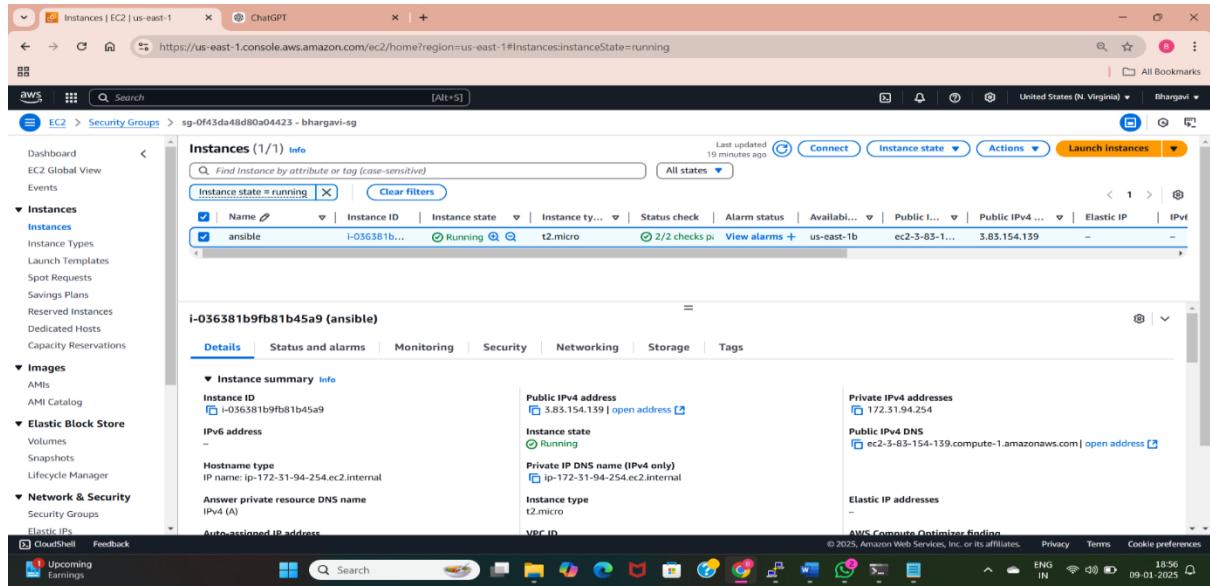
A playbook contains a list of **tasks** to define what actions need to be performed on the managed nodes (like installing software, starting services, etc.).

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

```

[ec2-user@ip-172-31-81-111:~]
> python 2.7.18
[ec2-user@ip-172-31-81-111 bin]$ ^C
[ec2-user@ip-172-31-81-111 bin]$ python3 --version
python 3.7.15
[ec2-user@ip-172-31-81-111 bin]$ cd
[ec2-user@ip-172-31-81-111 ~]$ sudo amazon-linux-extras install ansible2 -y
topic ansible2 has end-of-support date of 2023-09-30
Installing ansible
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Cleaning repos: amzn2-core amzn2extra-ansible2 amzn2extra-docker amzn2extra-kernel-5.10
17 metadata files removed
6 sqiqlite files removed
9 rpmdb files removed
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core
amzn2extra-ansible2
amzn2extra-docker
amzn2extra-kernel-5.10
(1/9): amzn2-core/2/x86_64/group_gz | 3.6 kB 00:00:00
(2/9): amzn2-core/2/x86_64/updateinfo | 2.9 kB 00:00:00
(3/9): amzn2extra-docker/2/x86_64/primary_db | 3.0 kB 00:00:00
(4/9): amzn2extra-kernel-5.10/2/x86_64/updateinfo | 2.7 kB 00:00:00
(5/9): amzn2extra-ansible2/2/x86_64/updateinfo | 1.0 MB 00:00:00
(6/9): amzn2extra-ansible2/2/x86_64/primary_db | 1.17 kB 00:00:00
(7/9): amzn2extra-docker/2/x86_64/updateinfo | 4.7 kB 00:00:00
(8/9): amzn2extra-kernel-5.10/2/x86_64/primary_db | 37 kB 00:00:00
(9/9): amzn2-core/2/x86_64/primary_db | 20 kB 00:00:00
Resolving Dependencies
--> Running transaction check
--> Package ansible.noarch 0:2.9.23-1.amzn2 will be installed
--> Processing Dependency: python2-setuptools for package: ansible-2.9.23-1.amzn2.noarch
--> Processing Dependency: python2-httplib2 for package: ansible-2.9.23-1.amzn2.noarch
--> Processing Dependency: python-keyczar for package: ansible-2.9.23-1.amzn2.noarch
--> Processing Dependency: python-paramiko for package: ansible-2.9.23-1.amzn2.noarch
--> Processing Dependency: sshpass for package: ansible-2.9.23-1.amzn2.noarch
--> Running transaction check
--> Package python-keyczar.noarch 0:0.71c-2.amzn2 will be installed
--> Package python2-crypto.x86_64 0:2.6.1-13.amzn2.0.3 will be installed
--> Processing Dependency: libitmCrypt.so.1.0.1 (64bit) for package: python2-crypto-2.6.1-13.amzn2.0.3.x86_64
--> Package python2-httplib2.noarch 0:1.19.1-3.amzn2 will be installed
--> Package python2-paramiko.noarch 0:2.16.1-13.amzn2.0.3 will be installed
--> Processing Dependency: python2-paramiko for package: python2-paramiko-1.16.1-3.amzn2.0.3.noarch
--> Package sshpass.x86_64 0:1.06-1.amzn2.0.1 will be installed
--> Running transaction check
--> Package libtomcrypt.x86_64 0:1.18.2-1.amzn2.0.1 will be installed
--> Processing Dependency: libtommath >= 1.0 for package: libtomcrypt-1.18.2-1.amzn2.0.1.x86_64

```

- ♣ 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.

```

[cmd] Command Prompt
Microsoft Windows [Version 10.0.22631.4692]
(c) Microsoft Corporation. All rights reserved.

C:\Users\bhava>cd downloads
C:\Users\bhava\Downloads>cd aws class
C:\Users\bhava\Downloads\AWS CLASS>scp -i new-key.pem new-key.pem ec2-user@3.83.221.224:/root
scp: stat local "new_key.pem": No such file or directory

C:\Users\bhava\Downloads\AWS CLASS>scp -i new-key.pem new-key.pem ec2-user@3.83.221.224:/root
Warning: Identity file "new_key.pem" not accessible: No such file or directory
The authenticity of host '3.83.221.224' (3.83.221.224) can't be established.
ED25519 key fingerprint is SHA256:007Raf+faolac4VSqJscQHQgR9jTrwElPR2HVaAGyx8Y.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
Warning: Permanently added '3.83.221.224' (ED25519) to the list of known hosts.
ec2-user@3.83.221.224: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
scp: Connection closed

C:\Users\bhava\Downloads\AWS CLASS>scp -i new-key.pem new-key.pem ec2-user@3.83.221.224:/root
scp: dest open "/root/new-key.pem": Permission denied
scp: failed to upload file new-key.pem to /root

C:\Users\bhava\Downloads\AWS CLASS>scp -i new-key.pem new-key.pem ec2-user@3.83.221.224:/home/ec2-user
          100% 1678      6.8KB/s   00:00

C:\Users\bhava\Downloads\AWS CLASS>

```

- ♣ 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.

```

[ec2-user@ip-172-31-81-111:~]
https://pip.pypa.io/warnings/venv
[ec2-user@ip-172-31-81-111 bin]$ python3 -m pip show boto3
Name: boto3
Version: 1.33.13
Summary: The AWS SDK for Python
Home-page: https://github.com/boto/boto3
Author: Amazon Web Services
Author-email:
License: Apache License 2.0
Location: /usr/local/lib/python3.7/site-packages
Requires: botocore, jmespath, s3transfer
Required-by:
[ec2-user@ip-172-31-81-111 bin]$ sudo pip3 install botocore
Requirement already satisfied: botocore in /usr/local/lib/python3.7/site-packages (1.33.13)
Requirement already satisfied: jmespath<2.0.0,>=0.9.4 in /usr/local/lib/python3.7/site-packages (from botocore) (0.10.1)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1.3 in /usr/local/lib/python3.7/site-packages (from botocore) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-packages (from botocore) (1.26.20)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-packages (from python-dateutil<3.0.0,>=2.1>botocore) (1.17.0)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: h
https://pip.pypa.io/warnings/venv
[ec2-user@ip-172-31-81-111 bin]$
[?] login as: ec2-user
[?] Authenticating with public key "imported-openssh-key"
Last login: Thu Jan  9 03:39:36 2025 from 49.43.201.231
  _/\_   Amazon Linux 2
  ~ \_\_  AL2 End of Life is 2025-06-30.
  ~ \_\_ \_\_ A newer version of Amazon Linux is available!
  ~ \_\_ \_\_ / Amazon Linux 2023, GA and supported until 2028-03-15.
  ~ \_\_ \_\_ / https://aws.amazon.com/linux/amazon-linux-2023/
[ec2-user@ip-172-31-81-111 ~]$ ll
total 4
-rw-r--r-- 1 ec2-user ec2-user 1678 Jan  9 03:54 new-key.pem
[ec2-user@ip-172-31-81-111 ~]$ sudo chmod 600 new-key.pem
[ec2-user@ip-172-31-81-111 ~]$ cd /etc/ansible/
[ec2-user@ip-172-31-81-111 ansible]$ ls
total 24
-rw-r--r-- 1 root root 19985 Jul  1 2021 ansible.cfg
-rw-r--r-- 1 root root 1016 Jul  1 2021 hosts
drwxr-xr-x 2 root root  6 Jul  1 2021 roles
[ec2-user@ip-172-31-81-111 ansible]$ sudo vi ansible.cfg
[ec2-user@ip-172-31-81-111 ansible]$ sudo vi hosts
[ec2-user@ip-172-31-81-111 ansible]$ cd
[ec2-user@ip-172-31-81-111 ~]$ 

```

System tray icons: 17°C, Haze, ENG IN, 09:29, 09-01-2025.

- 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.

```

[ec2-user@ip-172-31-27-18:~]
1 # config file for ansible -- https://ansible.com/
2 #
3
4 # nearly all parameters can be overridden in ansible-playbook
5 # or with command line flags. ansible will read ANSIBLE_CONFIG,
6 # ansible.cfg in the current working directory, .ansible.cfg in
7 # the home directory or /etc/ansible/ansible.cfg, whichever it
8 # finds first
9
10 [defaults]
11
12 # some basic default values...
13
14 inventory      = /etc/ansible/hosts
15 #library       = /usr/share/my_modules/
16 #module_utils   = /usr/share/my_module_utils/
17 #remote_tmp     = ~/.ansible/tmp
18 #local_tmp      = ~/.ansible/tmp
19 #plugin_filters = /etc/ansible/plugin_filters.yml
20 #forks          = 5
21 #poll_interval  = 15
22 #sudo_user      = root
23 #ask_sudo_pass  = True
24 #ask_pass        = True
25 #transport      = smart
26 #remote_port    = 22
27 #module_lang    = C
28 #module_set_locale = False
29
30 # plays will gather facts by default, which contain information about
31 # the remote system.
32 #
33 # smart - gather by default, but don't regather if already gathered
34 # implicit - gather by default, turn off with gather_facts: False
35 # explicit - do not gather by default, must say gather_facts: True
36 #gathering = implicit
37
38 # This only affects the gathering done by a play's gather_facts directive,
39 # by default gathering retrieves all facts subsets
40 # all - gather all subsets
-- INSERT --

```

System tray icons: 84°F, Sunny, ENG IN, 15:07, 06-01-2025.

- 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.

```

ec2-user@ip-172-31-81-111:/etc/ansible
35 # explicitly - do not gather by default, must say gather_facts: True
36 #gather_facts = implicit
37
38 # This only affects the gathering done by a play's gather_facts directive,
39 # by default gathering retrieves all facts subsets
40 # gather_all_subsets = False
41 # gather_subset = None
42 # hardware - gather hardware facts (longest facts to retrieve)
43 # virtual - gather min and virtual facts
44 # facter - import facts from facter
45 # chail - import facts from chail
46 # You can combine them using commas (ex: network,virtual)
47 # You can negate them using ! (ex: !hardware,!facter,!chail)
48 # A minimal set of facts is always gathered.
49 # gather_subset = all
50
51 # Some hardware related facts are collected
52 # with a maximum timeout of 10 seconds. This
53 # option lets you increase or decrease that
54 # timeout to something more suitable for the
55 # environment.
56 # gather_timeout = 10
57
58 # Ansible facts are available inside the ansible_facts.* dictionary
59 # namespace. This setting maintains the behaviour which was the default prior
60 # to 2.0 where these variables were imported directly into the main namespace, each with a
61 # prefix of 'ansible'.
62 # This variable is set to True by default for backwards compatibility. It
63 # will be changed to a default of 'False' in a future release.
64 # ansible_facts =
65 # inject_facts_as_vars = True
66
67 # additional paths to search for roles in, colon separated
68 roles_path = /etc/ansible/roles
69
70 # uncomment this to disable SSH key host checking
71 #host_key_checking = False
72
73 # change the default callback, you can only have one 'stdout' type enabled at a time.
74 stdout_callback = skippy
75
76
77 # Ansible ships with some plugins that require whitelisting.
78 # This is used to avoid running all of them by default.
79 # These settings lists them so you can enable them on your system.
80 # Custom plugins should not need this unless plugin author specifies it.
81
82 # enable callback plugins, they can output to stdout but cannot be 'stdout' type.
:set nu

```

17°C Haze ENG IN 09:47 09-01-2025

- 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.

```

ec2-user@ip-172-31-81-111:-
[localhost]
127.0.0.1 ansible_host=172.31.81.111 ansible_python_interpreter=/usr/bin/python3 ansible_user=ec2-user ansible_ssh_private_key_file=/home/ec2-user/new-key.pem

# This is the default ansible 'Hosts' file.
# It should live in /etc/ansible/hosts

# Comments begin with the '#' character
# Groups of hosts are defined by [header] elements
# - You can enter hostnames or IP addresses
# - A hostname/IP can be a member of multiple groups

# AA 1: Unspecified hosts, specify before any group Headers.

## www.example.com
## 127.0.0.1
## 192.168.1.10
## 192.168.1.110

# AA 2: A collection of hosts belonging to the 'unspecified' group
## [unspecified]
## alpha.example.com
## beta.example.com
## 192.168.1.100
## 192.168.1.110

# AA 3: If you have multiple hosts following a pattern you can specify
# them like this!
## www.(81|104).example.com

# AA 4: A collection of database servers in the 'dbservers' group
## [dbservers]
## www.internetsyndicate.net
## null.internetsyndicate.net
## 19.25.1.98
## 19.25.1.97

# AA's second example of host ranges, this time there are no
# trailing !
## 192.168.10.100-105.example.com

-- INSERT (paste) --

```

20°C Haze ENG IN 09:47 09-01-2025

- 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 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 <ansible-playbook vpc.yml>. It will automates the creation and installations which mentioned in the playbook.

```

PLAY RECAP
localhost : ok=0    changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-81-111 ~]$ ^C
[ec2-user@ip-172-31-81-111 ~]$ sudo vi /etc/ansible/hosts
[ec2-user@ip-172-31-81-111 ~]$ sudo vi /etc/ansible/hosts
[ec2-user@ip-172-31-81-111 ~]$ sudo vi /etc/ansible/hosts
[ec2-user@ip-172-31-81-111 ~]$ cd
[ec2-user@ip-172-31-81-111 ~]$ sudo ansible-playbook vpc.yml

PLAY [localhost] ****
TASK [ec2_vpc_net] ****
changed: [127.0.0.1]
TASK [ec2_vpc_iow] ****
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] ****
fatal: [127.0.0.1]: FAILED! => {"changed": false, "msg": "boto required for this module"}

PLAY RECAP
127.0.0.1 : ok=7    changed=7    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-81-111 ~]$ ^C
[ec2-user@ip-172-31-81-111 ~]$ sudo pip3 install boto3 or botocore
Requirement already satisfied: boto3 in /usr/local/lib/python3.7/site-packages (1.33.13)
Requirement already satisfied: botocore in /usr/local/lib/python3.7/site-packages (1.33.13)
ERROR: No matching distribution found for or
[ec2-user@ip-172-31-81-111 ~]$ sudo pip3 install botocore
Requirement already satisfied: botocore in /usr/local/lib/python3.7/site-packages (1.33.13)

```

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.

Name	VPC ID	State	Block Public Access	DNS hostnames
bhargavi_vpc	vpc-09d00b7e78b600d0	Available	Off	Enabled
	vpc-03f5fe5fd53d1d03	Available	Off	rtb-097c9de1de2e2bb8c

Details		Main route table	
VPC ID	vpc-09d00b7e78b600d0	State	Available
DNS resolution	Enabled	Tenancy	default
Main network ACL	acl-02d9cc61a13c725cf	Default VPC	No
IPv6 CIDR (Network border group)	-	Network Address Usage metrics	Disabled
		Route 53 Resolver DNS Firewall rule groups	-

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.

VPC dashboard

Internet gateways (1/2) **Info**

Name	Internet gateway ID	State	VPC ID	Owner
igw-040ac3a04fe0e294a	Attached	vpc-03f5fe3fd53d1d03	905418365089	
bhargavi_igw	Attached	vpc-09d00b7e78b600d00 bhargavi_vpc	905418365089	

igw-00b5040d3a74e3f3c / bhargavi_igw

Details

Internet gateway ID igw-00b5040d3a74e3f3c	State Attached	VPC ID vpc-09d00b7e78b600d00 bhargavi_vpc	Owner 905418365089
--	-------------------	--	-----------------------

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.

VPC dashboard

Subnets (1/8) **Info**

Name	Subnet ID	State	VPC	IPv4 CIDR	IPv6 ...	IPv6 CIDR association ID	Available IPv4 ac...
subnet-01657ffa4537...	vpc-03f5f...	Available	vpc-03f5f...	172.31.48.0/20	-	-	4091
bhargavi_pub	subnet-0f3377454d81...	Available	vpc-09d0...	11.0.1.0/24	-	-	250
subnet-080407bd5e37...	vpc-03f5f...	Available	vpc-03f5f...	172.31.0.0/20	-	-	4091
subnet-0bf303e5b007...	vpc-03f5f...	Available	vpc-03f5f...	172.31.64.0/20	-	-	4091
subnet-0b9c3ee5856f...	vpc-03f5f...	Available	vpc-03f5f...	172.31.80.0/20	-	-	4090
subnet-0f875dbfebb8...	vpc-03f5f...	Available	vpc-03f5f...	172.31.16.0/20	-	-	4091
bhargavi_pvt	subnet-0f4648fc1ffd...	Available	vpc-09d0...	11.0.2.0/24	-	-	251
	subnet-0db0d55fcff67...	Available	vpc-03f5f...	172.31.32.0/20	-	-	4091

subnet-0f3377454d81d400c / bhargavi_pub

Details

Subnet ID subnet-0f3377454d81d400c	Subnet ARN arn:aws:ec2:us-east-1:905418365089:subnet-0f3377454d81d400c	State Available	Block Public Access Off
IPv4 CIDR 11.0.1.0/24	-	IPv6 CIDR -	IPv6 CIDR association ID -
Availability Zone 250	Available IPv4 addresses 250	Network border group -	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.

The screenshot shows the AWS VPC Subnets console. On the left, there's a navigation sidebar with sections for EC2 Global View, Virtual private cloud (Your VPCs, Subnets, Route tables, Internet gateways, Egress-only internet gateways, Carrier gateways, DHCP option sets, Elastic IPs, Managed prefix lists, NAT gateways, Peering connections), Security (Network ACLs, Security groups), and PrivateLink and Lattice (Getting started, Updated). The main area displays a table titled "Subnets (1/8) Info" with columns: Name, Subnet ID, State, VPC, IPv4 CIDR, IPv6 CIDR, IPv6 association ID, and Available IPv4 addresses. A specific subnet named "bhargavi_pvt" is selected, showing its details: Subnet ID (subnet-0f4648fc1ffd720c), Subnet ARN (arn:aws:ec2:us-east-1:905418365089:subnet-0f4648fc1ffd720c), State (Available), IPv4 CIDR (11.0.2.0/24), Available IPv4 addresses (251), and Network border group (VPC). The bottom right corner shows the AWS logo, the date (09-01-2025), and the time (10:50).

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.

The screenshot shows the AWS VPC Route Tables console. The left sidebar is identical to the previous one. The main area displays a table titled "Route tables (1/4) Info" with columns: Name, Route table ID, Explicit subnet associations, Edge associations, Main, VPC, and Owner ID. A route table named "bhargavi_pub_rt" is selected, showing its details: Route table ID (rtb-074df972fc6fa0b39), Main (No), VPC (vpc-09d00b7e78b600d00), and Owner ID (905418365089). The bottom right corner shows the AWS logo, the date (09-01-2025), and the time (10:50).

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.

Route tables (1/4) Info

Name	Route table ID	Explicit subnet associations	Main	VPC	Owner...
-	rtb-0be7de17f6670a...	-	Yes	vpc-03f5fe3fd53d1d03	905418...
-	rtb-09c9de1de2e2b...	-	Yes	vpc-09d00b7e78b600d00 bhargavi_vpc	905418...
<input checked="" type="checkbox"/> bhargavi_pub_rt	rtb-074df972fc5fa0b...	subnet-03377454d81d400c / bhargavi_pub	No	vpc-09d00b7e78b600d00 bhargavi_vpc	905418...
<input checked="" type="checkbox"/> bhargavi_pvt_rt	rtb-0b7bf05890d0d2...	subnet-0f4648cf1ffd720c / bhargavi_pvt	No	vpc-09d00b7e78b600d00 bhargavi_vpc	905418...

rtb-0b7bf05890d0d2fa4 / bhargavi_pvt_rt

Details **Routes** **Subnet associations** **Edge associations** **Route propagation** **Tags**

Details

Route table ID rtb-0b7bf05890d0d2fa4	Main <input checked="" type="checkbox"/>	Explicit subnet associations subnet-0f4648cf1ffd720c / bhargavi_pvt
VPC vpc-09d00b7e78b600d00 bhargavi_vpc	Owner ID 905418365089	Edge associations -

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.

sg-0bc7f208dfb945410 - bhargavi_sg

Details

Security group name bhargavi_sg	Security group ID sg-0bc7f208dfb945410	Description allow	VPC ID vpc-09d00b7e78b600d00
Owner 905418365089	Inbound rules count 1 Permission entry	Outbound rules count 1 Permission entry	

Inbound rules (1)

Name	Security group r...	IP version	Type	Protocol	Port range	Source	Description
-	sgr-0c820fda1af7d...	IPv4	All traffic	All	All	0.0.0.0/0	allow all traffic

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

```

ec2-user@ip-172-31-81-111:~$ Requirement already satisfied: botocore in /usr/local/lib/python3.7/site-packages (1.33.13)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/local/lib/python3.7/site-packages (from botocore) (1.0.1)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/local/lib/python3.7/site-packages (from botocore) (2.9.0.post0)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/local/lib/python3.7/site-packages (from botocore) (1.26.20)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-packages (from python-dateutil<3.0.0,>=2.1->botocore) (1.17.0)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
[ec2-user@ip-172-31-81-111 ~]$ sudo pip3 install botocore
[ec2-user@ip-172-31-81-111 ~]$ sudo pip3 install boto
Collecting boto
  Downloading boto-2.49.0-py2.py3-none-any.whl.metadata (7.3 kB)
Downloaded boto-2.49.0-py2.py3-none-any.whl (1.4 MB)
    Installing collected packages: boto
      Successfully installed boto-2.49.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
[ec2-user@ip-172-31-81-111 ~]$ sudo ansible-playbook vpc.yml

PLAY [localhost] *****
TASK [ec2_vpc_net] *****
ok: [127.0.0.1]
TASK [ec2_vpc_igw] *****
ok: [127.0.0.1]
TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]
TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]
TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]
TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]
TASK [ec2_group] *****
ok: [127.0.0.1]
TASK [ec2] *****
changed: [127.0.0.1]

PLAY RECAP *****
127.0.0.1 : ok=8    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-81-111 ~]$ 

```

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

Region	Instance Type	Public IP Address	Private IP Address	State
us-east-1	t2.micro	ec2-44-197-170-166	ip-11-0-1-70.ec2.internal	Running
us-east-1	t2.micro	ec2-44-197-170-166	ip-11-0-1-70.ec2.internal	Running

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 configurations 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.

```

ec2-user@ip-172-31-81-111:~$ ansible-playbook vpc.yml
PLAY [localhost] *****
TASK [ec2_group] *****
An exception occurred during task execution. To see the full traceback, use -vvv. The error was: botocore.exceptions.ClientError: An error occurred (DependencyViolation) when calling the DeleteSecurityGroup operation: Resource sg-0bc7f208dfb945410 has a dependent object: AllowAllTraffic (id: 1127.0.0.1). FAILED! => {"botocore_version": "1.33.13", "changed": false, "error": {"code": "DependencyViolation", "message": "resource sg-0bc7f208dfb945410 has a dependent object: AllowAllTraffic (id: 1127.0.0.1)"}, "msg": "Unable to delete security group 'allow_all_traffic' 'allow'. 'GroupName': 'bharavisi_sg', 'IpPermissions': [{"IpProtocol": '-1', 'IpRanges': []}, {"UserIdGroupPairs": []}], 'OwnerId': '905419365099', 'GroupId': 'sg-0bc7f208dfb945410', 'IpPermissionsEgress': [{"IpProtocol": '-1', 'IpRanges': ['0.0.0.0/0']}, {"IpRanges": []}, {"UserIdGroupPairs": []}], 'Tags': [{"Name": "Name", "Value": "bharavisi_sg"}, "VpcId": "vpc-09a00b7e7bb600d00"}]: An error occurred (DependencyViolation) when calling the DeleteSecurityGroup operation: resource sg-0bc7f208dfb945410 has a dependent object: AllowAllTraffic (id: 1127.0.0.1). Response status code: 400, request id: ab6efbaf-4509-4304-b630-b7c19ee21996, retry attempts: 0)
PLAY RECAP *****
127.0.0.1 : ok=6   changed=0   unreachable=0   failed=1    skipped=0   rescued=0   ignored=0

[ec2-user@ip-172-31-81-111 ~]$ sudo vi vpc.yml
[ec2-user@ip-172-31-81-111 ~]$ sudo ansible-playbook vpc.yml
PLAY [localhost] *****
TASK [ec2_vpc_net] *****
ok: [127.0.0.1]

TASK [ec2_vpc_igw] *****
ok: [127.0.0.1]

TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]

TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]

TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]

TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]

TASK [ec2_group] *****
ok: [127.0.0.1]

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

PLAY RECAP *****
127.0.0.1 : ok=8   changed=1   unreachable=0   failed=0    skipped=0   rescued=0   ignored=0
[ec2-user@ip-172-31-81-111 ~]$ 

```

Now run the ansible playbook using <ansible-playbook 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-0f4648cfclffd720c" ]
    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.

```

ec2-user@ip-172-31-81-111:~ TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]

TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]

TASK [ec2_group] *****
ok: [127.0.0.1]

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

PLAY RECAP
127.0.0.1 : ok=8    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-81-111 ~]$ sudo vi vpc.yml
[ec2-user@ip-172-31-81-111 ~]$ sudo ansible-playbook vpc.yml

PLAY [localhost] *****

TASK [ec2_vpc_net] *****
ok: [127.0.0.1]

TASK [ec2_vpc_igw] *****
ok: [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] *****
ok: [127.0.0.1]

PLAY RECAP
127.0.0.1 : ok=8    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-81-111 ~]$ █

```

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.

```
ec2-user@ip-172-31-81-111:~$ sudo vi vpc.yml
[ec2-user@ip-172-31-81-111 ~]$ sudo ansible-playbook vpc.yml

PLAY [localhost]
  TASK [ec2_vpc_net]
    ok: [127.0.0.1]

  TASK [ec2_vpc_igw]
    changed: [127.0.0.1]

  TASK [ec2_vpc_subnet]
    ok: [127.0.0.1]

  TASK [ec2_vpc_route_table]
    ok: [127.0.0.1]

  TASK [ec2_vpc_route_table]
    ok: [127.0.0.1]

  TASK [ec2_group]
    ok: [127.0.0.1]

  TASK [ec2]
    ok: [127.0.0.1]

PLAY RECAP
127.0.0.1 : ok=6    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-81-111 ~]$ sudo vi vpc.yml
[ec2-user@ip-172-31-81-111 ~]$ sudo ansible-playbook vpc.yml

PLAY [localhost]
  TASK [ec2_vpc_net]
    ok: [127.0.0.1]

  TASK [ec2_vpc_igw]
    changed: [127.0.0.1]

  TASK [ec2_vpc_subnet]
    ok: [127.0.0.1]

  TASK [ec2_vpc_route_table]
    ok: [127.0.0.1]

  TASK [ec2_vpc_route_table]
    ok: [127.0.0.1]

  TASK [ec2_group]
    ok: [127.0.0.1]

  TASK [ec2]
    ok: [127.0.0.1]

PLAY RECAP
127.0.0.1 : ok=6    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

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.

```

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

TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]

TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]

TASK [ec2_group] *****
ok: [127.0.0.1]

TASK [ec2] *****
ok: [127.0.0.1]

PLAY RECAP
127.0.0.1 : ok=8    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-81-111 ~]$ sudo vi vpc.yml
[ec2-user@ip-172-31-81-111 ~]$ sudo ansible-playbook vpc.yml

PLAY [localhost] *****

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

TASK [ec2_vpc_igw] *****
ok: [127.0.0.1]

TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]

TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]

TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]

TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]

TASK [ec2_group] *****
ok: [127.0.0.1]

TASK [ec2] *****
ok: [127.0.0.1]

PLAY RECAP
127.0.0.1 : ok=8    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-81-111 ~]$ 

```

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.

```
- hosts: localhost
  become: yes
  gather_facts: false #if you want to hide the host key verification option then use gather_facts: false #####
  vars_files:
    - secret.yaml
  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 ipnames
    dns_hostnames: yes
    tenancy: default
    state: present # to delete Vpc then replace absent instead of present
    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
## 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_ip: yes
    resource_tags:
      Name: "[[ pubsubnet.name ]]"
    register: pubsubnet_result
# create an vpc pri 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: "[[ privzone ]]"        # az is the availability zone
    state: present
-- INSERT --
```

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

```

# AWS credentials
aws_access_key: "AKIA5PTZD3CQW63UY2XC"
aws_secret_key: "300srurc+1lf8mrhnyQee69xG0zJPBUV4ZvW"
title: "Bhargavi"

# VPC configuration
vpc_name: bhargavi_vpc
vpc_cidr_block: "172.0.0.0/16"
region: "us-east-1"

# Internet Gateway configuration
igw_name: bhargavi_igw

# Public Subnet configuration
pubsubnet_name: bhargavi_pub
pubsubnet_cidr_block: "172.0.0.0/24"
pubroute: "0.0.0.0/0"

# Private Subnet configuration
privsubnet_name: bhargavi_pvt
privsubnet_cidr_block: "172.0.0.0/24"
privzone: "us-east-1b"

# Public Route Table configuration
pubroute_table_name: bhargavi_pub_rt

# Private Route Table configuration
privroute_table_name: bhargavi_pvt_rt

# Security Group configuration
security_group_name: bhargavi_sg
destination_cidr_block: "0.0.0.0/0"
port22_cidr_block: "0.0.0.0/0"

# EC2 instance configuration
image_id: "ami-0ca9fb66e076afe32"
type: "t2.micro"
instance_name: Bhargavi-outline-ec2

```

56,1 Bot

25°C Sunny

Search

ENG IN 12:43 09-01-2025

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.

```

PLAY [localhost] *****
TASK [Gathering Facts]
ok: [localhost]
TASK [Installing git]
changed: [localhost]
PLAY RECAP *****
localhost : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-83-204 ~]$ sudo vi outline.yml
[ec2-user@ip-172-31-83-204 ~]$ sudo vi secret.yaml
[ec2-user@ip-172-31-83-204 ~]$ sudo vi secret.yaml
[ec2-user@ip-172-31-83-204 ~]$ sudo ansible-vault encrypt secret.yaml
New Vault password:
Confirm New Vault password:
Vault password:
Vault password successful
[ec2-user@ip-172-31-83-204 ~]$ ansible-playbook outline.yml --ask-vault-pass
Vault password:
[WARNING]: * Failed to parse /etc/ansible/hosts with yaml plugin: We were unable to read either as JSON nor YAML, these are the errors we got from each: JSON: No JSON object could be decoded. Syntax Error while loading YAML: did not find expected document start> The error appears to be in '/etc/ansible/hosts': line 2, column 1, but may be elsewhere in the file, depending on the exact syntax problem> the offending line appears to be: (localhost) ansible_python_interpreter=/usr/bin/python3 ansible_user=ec2-user
ansible_ssh_private_key_file=/home/ec2-user/.new-key.pem ^ here
[WARNING]: * Unable to parse /etc/ansible/hosts with inventory plugin: /etc/ansible/hosts:2: Expected key-value host variable assignment, got: 172.31.83.204
[WARNING]: * No inventory was found, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available
[WARNING]: implicit host matching was attempted while trying to read the file '/home/ec2-user/outline.yml': [Errno 13] Permission denied: '/home/ec2-user/outline.yml'
[ec2-user@ip-172-31-83-204 ~]$ cd /etc/ansible/
[ec2-user@ip-172-31-83-204 ~]$ hash -a md5
[ec2-user@ip-172-31-83-204 ~]$ cd /etc/ansible/
[ec2-user@ip-172-31-83-204 ansible]$ sudo vi hosts
[ec2-user@ip-172-31-83-204 ansible]$ cd
[ec2-user@ip-172-31-83-204 ~]$ ansible-playbook outline.yml --ask-vault-pass
Vault password:
[WARNING]: * Failed to parse /etc/ansible/hosts with yaml plugin: We were unable to read either as JSON nor YAML, these are the errors we got from each: JSON: No JSON object could be decoded. Syntax Error while loading YAML: did not find expected <document_start>. The error appears to be in '/etc/ansible/hosts': line 2, column 1, but may be elsewhere in the file, depending on the exact syntax problem> the offending line appears to be: (localhost) ansible_python_interpreter=/usr/bin/python3 ansible_user=ec2-user ansible_ssh_private_key_file=/home/ec2-user/.new-key.pem ^ here
[WARNING]: * Failed to parse /etc/ansible/hosts with inventory plugin: /etc/ansible/hosts:2: Expected key-value host variable assignment, got: 172.31.83.204
[WARNING]: Unable to parse /etc/ansible/hosts as an inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'
[WARNING]: implicit host matching was attempted while trying to read the file '/home/ec2-user/outline.yml': [Errno 13] Permission denied: '/home/ec2-user/outline.yml'
[ec2-user@ip-172-31-83-204 ~]$ cat
[ec2-user@ip-172-31-83-204 ~]$ sudo chmod 644 /etc/ansible/hosts
[ec2-user@ip-172-31-83-204 ~]$ sudo chmod 600 /home/ec2-user/secret.yaml

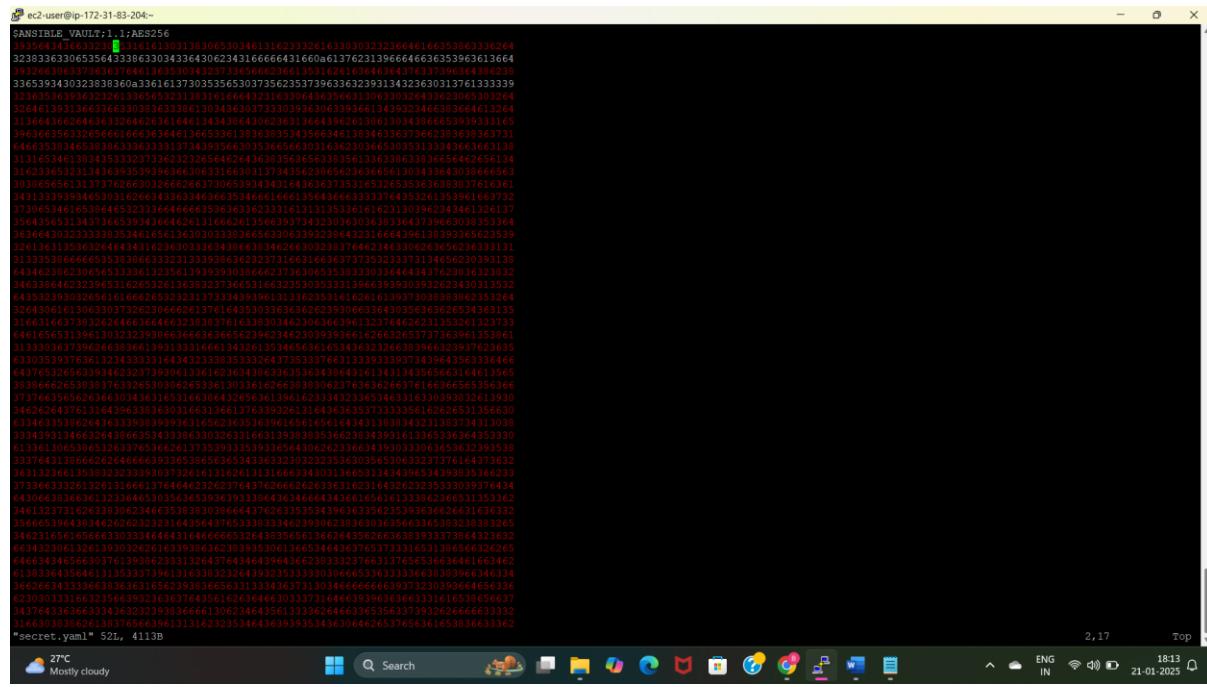
```

27°C Mostly cloudy

Search

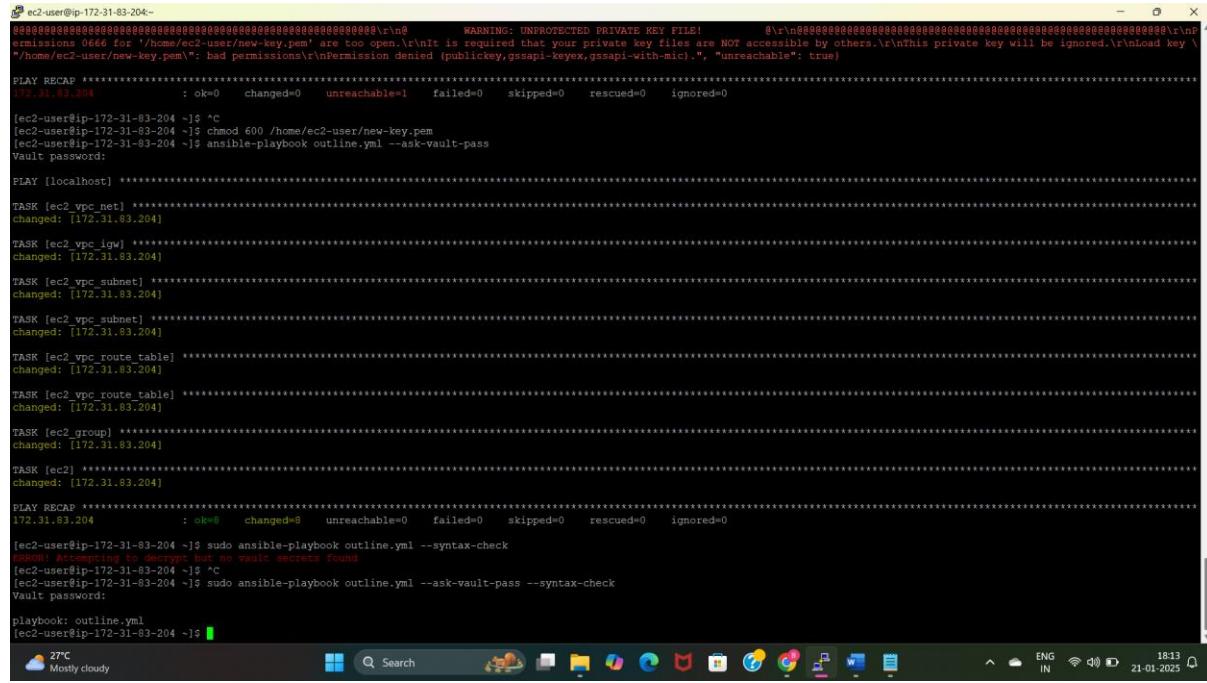
ENG IN 18:11 21-01-2025

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”.



```
ec2-user@ip-172-31-83-204-:~$ ansible vault -e secret.yaml
[...]
[...]
```

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



```
ec2-user@ip-172-31-83-204-:~$ ansible-vault --ask-vault-pass
[...]
[...]
```

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.

The screenshot shows the AWS Management Console VPC dashboard. In the center, there is a table titled "Your VPCs (1/2) Info" with one item listed:

Name	VPC ID	State	Block Public Access	DNS hostnames
bhargavi_vpc	vpc-06e4c2ec3c519737b	Available	Off	Enabled

Below the table, under "vpc-06e4c2ec3c519737b / bhargavi_vpc", there are tabs for "Details", "Resource map", "CIDRs", "Flow logs", "Tags", and "Integrations". The "Details" tab is selected, showing the following configuration details:

- VPC ID:** vpc-06e4c2ec3c519737b
- State:** Available
- DNS resolution:** Enabled
- Main network ACL:** ad-0c52b627e3a04cd1e
- IPv6 CIDR (Network border group):** -
- State:** Available
- Tenancy:** default
- Default VPC:** No
- IPv4 CIDR:** 11.0.0/16
- Network Address Usage metrics:** Disabled
- Block Public Access:** Off
- DHCP option set:** dopt-0fa14cc7d400cb9e2
- Route 53 Resolver DNS Firewall rule groups:** -
- Main route table:** rtb-0e604858fac72b2cf
- IPv6 pool:** -
- Owner ID:** 905418365089

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.

The screenshot shows the AWS Management Console Internet gateways section. In the center, there is a table titled "Internet gateways (1/2) Info" with two items listed:

Name	Internet gateway ID	State	VPC ID	Owner
bhargavi_igw	igw-03f94051d1e912c32	Attached	vpc-06e4c2ec3c519737b bhargavi_vpc	905418365089
-	igw-040ae3a04fe0e294a	Attached	vpc-03f5fe3fdc53d1d03	905418365089

Below the table, under "igw-03f94051d1e912c32 / bhargavi_igw", there are tabs for "Details" and "Tags". The "Details" tab is selected, showing the following configuration details:

- Internet gateway ID:** igw-03f94051d1e912c32
- State:** Attached
- VPC ID:** vpc-06e4c2ec3c519737b | bhargavi_vpc
- Owner:** 905418365089

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.

The screenshot shows the AWS VPC Subnets page. The left sidebar includes sections for EC2 Global View, Virtual private cloud (Your VPCs, Subnets, Route tables, Internet gateways, Egress-only internet gateways, Carrier gateways, DHCP option sets, Elastic IPs, Managed prefix lists, NAT gateways, Peering connections), Security (Network ACLs, Security groups), and PrivateLink and Lattice. The main content area displays a table of subnets with columns for Name, Subnet ID, State, VPC, IPv4 CIDR, IPv6 CIDR, IPv6 CIDR association ID, and Available IPv4 ac. Two subnets are listed: 'bhargavi_pub' (subnet-0331f16f9f4d61d25) and 'bhargavi_pvt' (subnet-05d86522d1eadaa4d). Both are in an 'Available' state. The 'Details' tab for 'bhargavi_pub' is selected, showing its ARN, IPv4 CIDR (11.0.1.0/24), and availability zone (250). The 'Details' tab for 'bhargavi_pvt' is also visible.

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 screenshot is identical to the one above, showing the AWS VPC Subnets page with the same subnets and details. The 'bhargavi_pvt' subnet is now selected in the table, and its 'Details' tab is active, displaying its ARN, IPv4 CIDR (11.0.2.0/24), and availability zone (251).

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.

VPC dashboard

Route tables (1/4) Info

Name	Route table ID	Explicit subnet associations	Edge associations	Main	VPC	Owner
rtb-0be7de17f6670a...	-	-	-	Yes	vpc-03f5fe3fd53d1d03	905418...
rtb-064fd7ac68ab1cae9 / bhargavi_pub_rt	rtb-064fd7ac68ab1cae9	subnet-03311f16f9f4d61d25 / bhargavi_pub	-	No	vpc-06e4c2ec3c519737b bhargavi_vpc	905418...
rtb-0a604838fec72b...	-	-	-	Yes	vpc-06e4c2ec3c519737b bhargavi_vpc	905418...
rtb-0715cbd1d7e59b...	rtb-0715cbd1d7e59b...	subnet-05d86522d1eada4d / bhargavi_pvt	-	No	vpc-06e4c2ec3c519737b bhargavi_vpc	905418...

rtb-064fd7ac68ab1cae9 / bhargavi_pub_rt

Details

Route table ID: rtb-064fd7ac68ab1cae9
Main: No
VPC: vpc-06e4c2ec3c519737b | bhargavi_vpc
Owner ID: 905418365089

Explicit subnet associations: subnet-03311f16f9f4d61d25 / bhargavi_pub
Edge associations: -

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.

EC2 > Security Groups > sg-0aa1c1b84ae6f1cb5 - bhargavi_sg

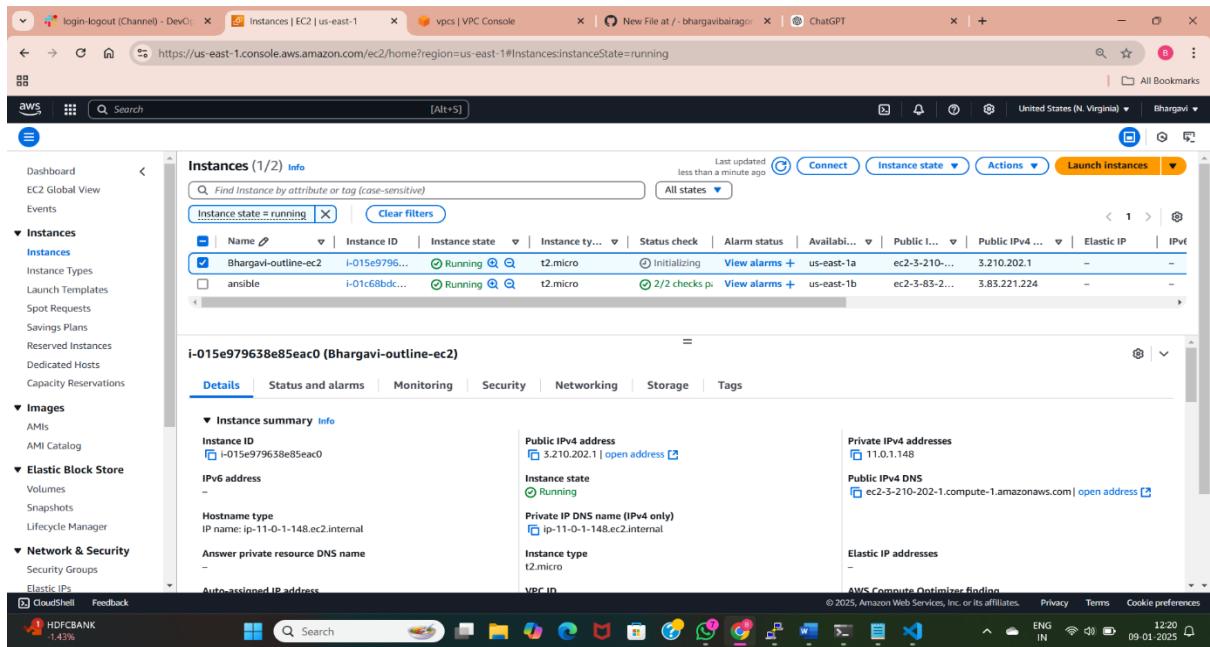
Details

Security group name: bhargavi_sg
Security group ID: sg-0aa1c1b84ae6f1cb5
Owner: 905418365089
Inbound rules count: 1 Permission entry
Outbound rules count: 1 Permission entry
VPC ID: vpc-06e4c2ec3c519737b

Inbound rules (1)

Name	Security group r...	IP version	Type	Protocol	Port range	Source	Description
sgr-00d7647188745...	IPv4	All traffic	All	All	0.0.0.0/0	allow	allow all traffic

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



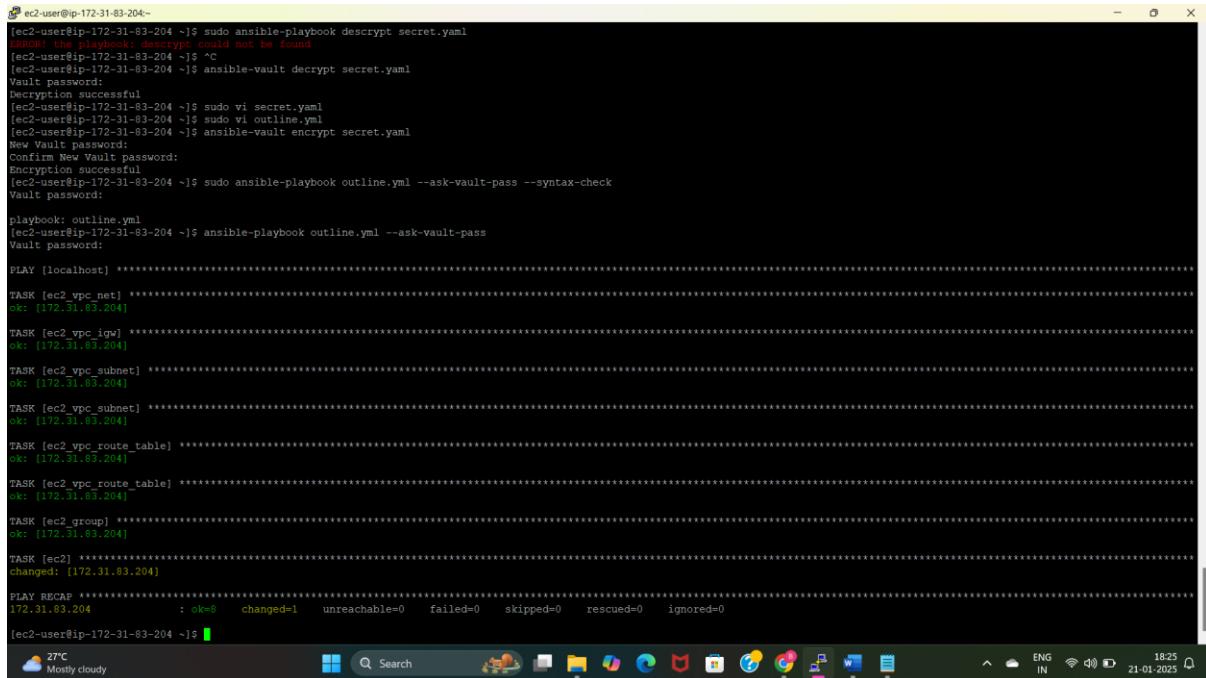
We can check with the aws console, ec2 service whether the instance is created or not. This is the new instance created using the playbook. We can check with the configurations which matches the inline variable like instance name, network settings, security group. After creating ec2 instance, security 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 ##

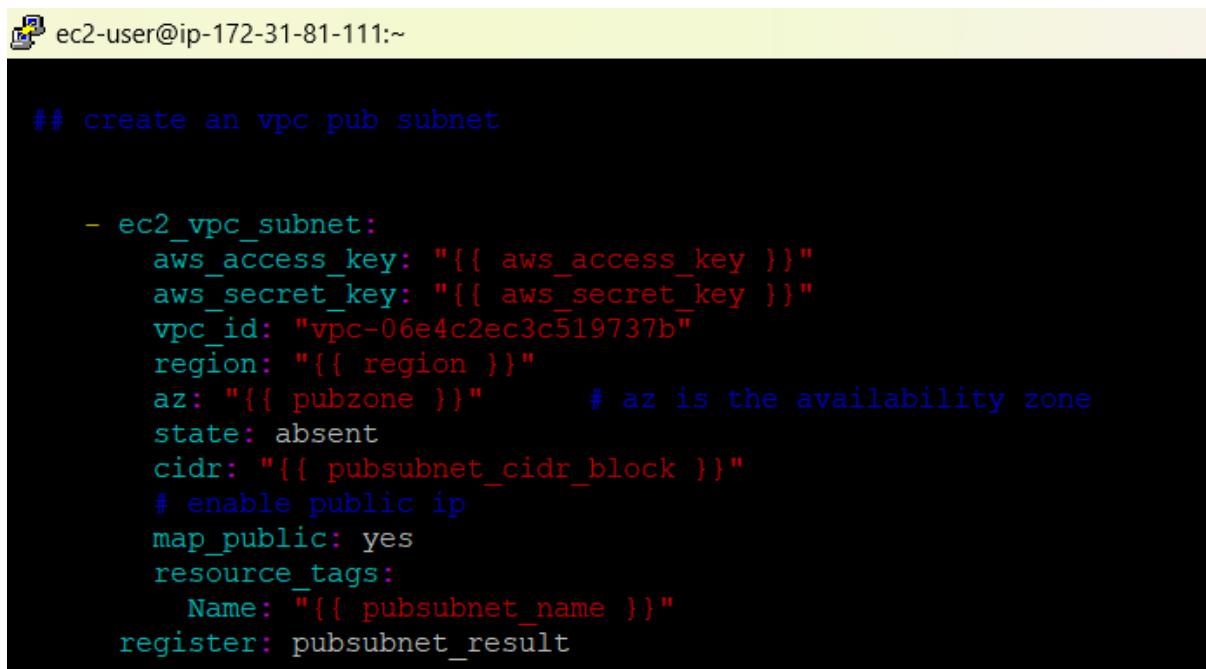
- ec2:
    image: ami-0ca9fb66e076a6e32
    instance_id: "i-015e979638e85eac0"
    instance_type: "{{ type }}"
    region: "{{ region }}"
    wait: yes
    count: 1
    state: absent
    vpc_subnet_id: "subnet-0331f16f9f4d61d25"
    assign_public_ip: yes
    group_id: "sg-0aa1c1b84ae6f1cb5"
    aws_access_key: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



```
ec2-user@ip-172-31-83-204~
[ec2-user@ip-172-31-83-204 ~]$ sudo ansible-playbook decrypt secret.yaml
[WARNING! The playbook: decrypt could not be found
[ec2-user@ip-172-31-83-204 ~]$ `c
[ec2-user@ip-172-31-83-204 ~]$ ansible-vault decrypt secret.yaml
Vault password:
Decryption successful
[ec2-user@ip-172-31-83-204 ~]$ sudo vi secret.yaml
[ec2-user@ip-172-31-83-204 ~]$ sudo vi outline.yml
[ec2-user@ip-172-31-83-204 ~]$ ansible-vault encrypt secret.yaml
New Vault password:
Confirm New Vault password:
Encryption successful
[ec2-user@ip-172-31-83-204 ~]$ sudo ansible-playbook outline.yml --ask-vault-pass --syntax-check
Vault password:
playbook: outline.yml
[ec2-user@ip-172-31-83-204 ~]$ ansible-playbook outline.yml --ask-vault-pass
Vault password:
PLAY [localhost] *****
TASK [ec2_vpc_net] *****
ok: [172.31.83.204]
TASK [ec2_vpc_igw] *****
ok: [172.31.83.204]
TASK [ec2_vpc_subnet] *****
ok: [172.31.83.204]
TASK [ec2_vpc_subnet] *****
ok: [172.31.83.204]
TASK [ec2_vpc_route_table] *****
ok: [172.31.83.204]
TASK [ec2_vpc_route_table] *****
ok: [172.31.83.204]
TASK [ec2_group] *****
ok: [172.31.83.204]
TASK [ec2] *****
changed: [172.31.83.204]
PLAY RECAP
172.31.83.204 : ok=8    changed=1    unreachable=0   failed=0    skipped=0   rescued=0   ignored=0
[ec2-user@ip-172-31-83-204 ~]$
```

Now run the ansible playbook using <ansible-playbook 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.

```

ec2-user@ip-172-31-83-204~
[ec2-user@ip-172-31-83-204 ~]$ ansible-playbook outline.yml --ask-vault-pass
Vault password:
PLAY [localhost] *****
TASK [ec2_vpc_net] *****
ok: [172.31.83.204]
TASK [ec2_vpc_igw] *****
ok: [172.31.83.204]
TASK [ec2_vpc_subnet] *****
changed: [172.31.83.204]
TASK [ec2_vpc_subnet] *****
changed: [172.31.83.204]
TASK [ec2_vpc_route_table] *****
changed: [172.31.83.204]
TASK [ec2_vpc_route_table] *****
changed: [172.31.83.204]
TASK [ec2_group] *****
ok: [172.31.83.204]
TASK [ec2] *****
fatal: [172.31.83.204]: FAILED! => {"msg": "The task includes an option with an undefined variable. The error was: 'dict object' has no attribute 'subnet'\n\nThe error appears to be in '/home/ec2-user/outline.yml': line 116, column 7, but maybe elsewhere in the file depending on the exact syntax problem.\n\nThe offending line appears to be:\n\n      - ec2:\n        ^ here\n"}
PLAY RECAP
172.31.83.204 : ok=7    changed=4    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-83-204 ~]$ sudo vi outline.yml
[ec2-user@ip-172-31-83-204 ~]$ ansible-playbook outline.yml --ask-vault-pass
Vault password:
PLAY [localhost] *****
TASK [ec2_vpc_net] *****
ok: [172.31.83.204]
TASK [ec2_vpc_igw] *****
changed: [172.31.83.204]
TASK [ec2_vpc_subnet] *****
ok: [172.31.83.204]
TASK [ec2_vpc_subnet] *****
ok: [172.31.83.204]

```

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.

```

[ec2-user@ip-172-31-83-204 ~]
[ec2-user@ip-172-31-83-204 ~]$ sudo vi outline.yml
[ec2-user@ip-172-31-83-204 ~]$ ansible-playbook outline.yml --ask-vault-pass
Vault password:

PLAY [localhost] *****
TASK [ec2_vpc_net] *****
ok: [172.31.83.204]

TASK [ec2_vpc_ipw] *****
changed: [172.31.83.204]

TASK [ec2_vpc_subnet] *****
ok: [172.31.83.204]

TASK [ec2_vpc_subnet] *****
ok: [172.31.83.204]

TASK [ec2_vpc_route_table] *****
ok: [172.31.83.204]

TASK [ec2_vpc_route_table] *****
ok: [172.31.83.204]

TASK [ec2_group] *****
ok: [172.31.83.204]

TASK [ec2] *****
fatal: [172.31.83.204]: FAILED! => {"msg": "The task includes an option with an undefined variable. The error was: 'dict object' has no attribute 'subnet'\n\nThe error appears to be in '/home/ec2-user/outline.yml'\nline 116, column 7, but maybe elsewhere in the file depending on the exact syntax problem.\n\nThe offending line appears to be:\n\n      - ec2:\n        - subnet\n\n      ^\n\n"}
PLAY RECAP
172.31.83.204 : ok=7    changed=1    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-83-204 ~]$ sudo vi outline.yml
[ec2-user@ip-172-31-83-204 ~]$ ansible-playbook outline.yml --ask-vault-pass
Vault password:

PLAY [localhost] *****
TASK [ec2_vpc_net] *****
ok: [172.31.83.204]

TASK [ec2_vpc_ipw] *****
ok: [172.31.83.204]

TASK [ec2_vpc_subnet] *****
ok: [172.31.83.204]

```

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:~]
---
- 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: present # 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.

```

ec2-user@ip-172-31-83-204~
TASK [ec2_vpc_route_table] *****
ok: [172.31.83.204]
TASK [ec2_vpc_route_table] *****
ok: [172.31.83.204]
TASK [ec2_group] *****
ok: [172.31.83.204]
TASK [ec2] *****
ok: [172.31.83.204]
PLAY RECAP
172.31.83.204 : ok=8    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-83-204 ~]$ sudo vi outline.yml
[ec2-user@ip-172-31-83-204 ~]$ ansible-playbook outline.yml --ask-vault-pass
Vault password:

PLAY [localhost] *****
TASK [ec2_vpc_net] *****
changed: [172.31.83.204]
TASK [ec2_vpc_igw] *****
ok: [172.31.83.204]
TASK [ec2_vpc_subnet] *****
ok: [172.31.83.204]
TASK [ec2_vpc_subnet] *****
ok: [172.31.83.204]
TASK [ec2_vpc_route_table] *****
ok: [172.31.83.204]
TASK [ec2_vpc_route_table] *****
ok: [172.31.83.204]
TASK [ec2_group] *****
ok: [172.31.83.204]
TASK [ec2] *****
ok: [172.31.83.204]
PLAY RECAP
172.31.83.204 : ok=8    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-83-204 ~]$ 

```

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.

```

ec2-user@ip-172-31-94-254~
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-packages (from python-dateutil<3.0.0,>=2.1->botocore) (1.17.0)
[ec2-user@ip-172-31-94-254 bin]$ python3 --version
Python 3.7.16
[ec2-user@ip-172-31-94-254 bin]$ d
-bash: d: command not found
[ec2-user@ip-172-31-94-254 bin]$ cd /etc/ansible/
[ec2-user@ip-172-31-94-254 ansible]$ sudo vi hosts
[ec2-user@ip-172-31-94-254 ansible]$ cd
[ec2-user@ip-172-31-94-254 ~]$ sudo ansible-playbook without.yml

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] *****
fatal: [127.0.0.1]: FAILED! => {"changed": false, "msg": "boto required for this module"}
PLAY RECAP
127.0.0.1 : ok=8    changed=7    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-94-254 ~]$ cd
[ec2-user@ip-172-31-94-254 ~]$ python3 -m pip show boto
Name: boto
Version: 2.49.0
Summary: Amazon Web Services Library
Home-page: https://github.com/boto/boto/

```

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.

```

ec2-user@ip-172-31-94-254:~$ Home-page: https://github.com/boto/boto/
Author: Mitch Garnaat
Author-email: mitch@garnaat.com
License: MIT
Location: /home/ec2-user/.local/lib/python3.7/site-packages
Requires:
Required-by:
[ec2-user@ip-172-31-94-254 ~]$ sudo pip3 install boto
WARNING: Running pip install with root privileges is generally not a good idea. Try 'pip3 install --user' instead.
Collecting boto
  Downloading boto-2.49.0-py2.py3-none-any.whl (1.4 MB)
    100% |██████████| 1.4 MB 22.4 MB/s
Installing collected packages: boto
Successfully installed boto-2.49.0
[ec2-user@ip-172-31-94-254 ~]$ sudo ansible-playbook without.yml

PLAY [localhost] *****

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

TASK [ec2_vpc_net] *****
ok: [127.0.0.1]

TASK [ec2_vpc_igw] *****
ok: [127.0.0.1]

TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]

TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]

TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]

TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]

TASK [ec2_group] *****
ok: [127.0.0.1]

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

PLAY RECAP *****
127.0.0.1 : ok=9    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-94-254 ~]$ 

```

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

Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR	DHCP option set	Main route table
bhargavi-vpc	vpc-07c41ab4db9095efc	Available	10.0.0.0/16	-	dopt-0fa14cc7d400cb9e2	rtb-060a27d723a491cf
	vpc-03f5fe4fd53d1d03	Available	172.31.0.0/16	-	dopt-0fa14cc7d400cb9e2	rtb-0be7de17f6670a5be

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.

VPC dashboard

Internet gateways (1/2) Info

Name	Internet gateway ID	State	VPC ID	Owner
igw-040ac3a04fe0e294a	Attached	vpc-03f5fe3dc53d1d03	905418365089	
bhargavi-igw	Attached	vpc-07c41ab4db9095efc bhargavi-vpc	905418365089	

igw-0ba0ab3594bb067e9 / bhargavi-igw

Details

Internet gateway ID igw-0ba0ab3594bb067e9	State Attached	VPC ID vpc-07c41ab4db9095efc bhargavi-vpc	Owner 905418365089
--	-------------------	--	-----------------------

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.

VPC dashboard

Subnets (1/8) Info

Name	Subnet ID	State	VPC	IPv4 CIDR	IPv6 ...	IPv6 CIDR association ID	Available IPv4 ac
subnet-01657ffa4537...	Available	vpc-03f5f...	172.31.48.0/20	-	-	-	4091
subnet-0a0407fd5e37...	Available	vpc-03f5f...	172.31.0.0/20	-	-	-	4091
subnet-0bf303e5b0007...	Available	vpc-03f5f...	172.31.64.0/20	-	-	-	4091
subnet-0b9c3ee5856f...	Available	vpc-03f5f...	172.31.80.0/20	-	-	-	4090
subnet-0f875d1fbefb8...	Available	vpc-03f5f...	172.31.16.0/20	-	-	-	4091
bhargavi-pvt	Available	vpc-07c4...	10.0.16.0/20	-	-	-	4091
subnet-0bd05d5cf6f6...	Available	vpc-03f5f...	172.31.32.0/20	-	-	-	4091
bhargavi-pub	Available	vpc-07c4...	10.0.0.0/20	-	-	-	4090

subnet-08163e10df309551b / bhargavi-pub

Details

Subnet ID subnet-08163e10df309551b	Subnet ARN arn:aws:ec2:us-east-1:905418365089:subnet/subnet-08163e10df309551b	State Available	Block Public Access Off
IPv4 CIDR 10.0.0.0/20	-	IPv6 CIDR -	IPv6 CIDR association ID -
Availability Zone -	Available IPv4 addresses 4090	Network border group -	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.

The screenshot shows the AWS VPC Subnets console. On the left, there's a navigation sidebar with sections like EC2 Global View, Virtual private cloud, Security, and PrivateLink and Lattice. The main area displays a table titled "Subnets (1/8) Info" with columns for Name, Subnet ID, State, VPC, IPv4 CIDR, IPv6 CIDR, IPv6 CIDR association ID, and Available IPv4 addresses. One row for "bhargavi-pvt" is selected, showing its details: Subnet ID: subnet-0b386a87cf1434996, Subnet ARN: arn:aws:ec2:us-east-1:905418365089:subnet-0b386a87cf1434996, State: Available, IPv4 CIDR: 10.0.16.0/20, Available IPv4 addresses: 4091. Below the table, a specific subnet detail page for "subnet-0b386a87cf1434996 / bhargavi-pvt" is shown with tabs for Details, Flow logs, Route table, Network ACL, CIDR reservations, Sharing, and Tags. The "Details" tab is selected, displaying information such as Subnet ID, Subnet ARN, IPv4 CIDR, Availability Zone, State, Block Public Access (Off), IPv6 CIDR, IPv6 CIDR association ID, and Network border group.

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.

The screenshot shows the AWS VPC RouteTables console. The left sidebar includes sections for EC2 Global View, Virtual private cloud, Security, and PrivateLink and Lattice. The main content area lists route tables with columns for Name, Route table ID, Explicit subnet associations, Edge associations, Main, and VPC. One route table, "bhargavi-pub-rt", is selected, showing its details: Route table ID: rtb-04972e32c83959522, Main: No, Owner ID: 905418365089, and Explicit subnet associations: subnet-08163e10df309551b / bhargavi-pub. Below the table, a specific route table detail page for "rtb-04972e32c83959522 / bhargavi-pub-rt" is displayed with tabs for Details, Routes, Subnet associations, Edge associations, Route propagation, and Tags. The "Details" tab is selected, showing the route table ID, Main status, Owner ID, and Explicit subnet associations.

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 configurations matches with the inline variable.

The screenshot shows the AWS CloudWatch Metrics interface. A single log entry is displayed in the log stream:

```

igw: [1588441600000] {
    "region": "us-east-1",
    "vpc_id": "vpc-00000000000000000000000000000000",
    "subnet_id": "subnet-00000000000000000000000000000000",
    "route_table_id": "rtb-00000000000000000000000000000000",
    "source_ip": "10.0.0.1",
    "destination_ip": "0.0.0.0/0",
    "protocol": "0/0",
    "port": "0/0",
    "version": "1.0"
}

```

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 configurations 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: S0JsTuFtC+l1fj6HrbhyQse69bxGGzJPBUv4Zvw
        cidr_block: 10.0.0.0/16
        name: bhargavi-vpc
        region: us-east-1
        state: absent # to delete Vpc then replace absent instead of present
        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: S0JsTuFtC+l1fj6HrbhyQse69bxGGzJPBUv4Zvw
    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.

```

ec2-user@ip-172-31-94-254:~$ ansible-playbook without.yml
PLAY [localhost] *****
TASK [Gathering Facts] *****
ok: [127.0.0.1]
TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]
TASK [ec2_group] *****
ok: [127.0.0.1]
TASK [ec2] *****
changed: [127.0.0.1]
PLAY RECAP *****
127.0.0.1 : ok=9    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-94-254 ~]$ sudo vi without.yml
[ec2-user@ip-172-31-94-254 ~]$ sudo ansible-playbook without.yml
PLAY [localhost] *****
TASK [Gathering Facts] *****
ok: [127.0.0.1]
TASK [ec2_vpc_net] *****
ok: [127.0.0.1]
TASK [ec2_vpc_igw] *****
ok: [127.0.0.1]
TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]
TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]
TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]
TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]
TASK [ec2_group] *****
ok: [127.0.0.1]
TASK [ec2] *****
changed: [127.0.0.1]
PLAY RECAP *****
127.0.0.1 : ok=9    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-94-254 ~]$ 

```

Now run the ansible playbook using <ansible-playbook 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.

```

ec2-user@ip-172-31-94-254:~$ 
TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]

TASK [ec2_group] *****
ok: [127.0.0.1]

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

PLAY RECAP *****
127.0.0.1 : ok=9    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-94-254 ~]$ sudo vi without.yml
[ec2-user@ip-172-31-94-254 ~]$ sudo ansible-playbook without.yml

PLAY [localhost] *****

TASK [Gathering Facts]
ok: [127.0.0.1]

TASK [ec2_vpc_net] *****
ok: [127.0.0.1]

TASK [ec2_vpc_igw] *****
ok: [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] *****
ok: [127.0.0.1]

PLAY RECAP *****
127.0.0.1 : ok=9    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-94-254 ~]$ 

```

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.

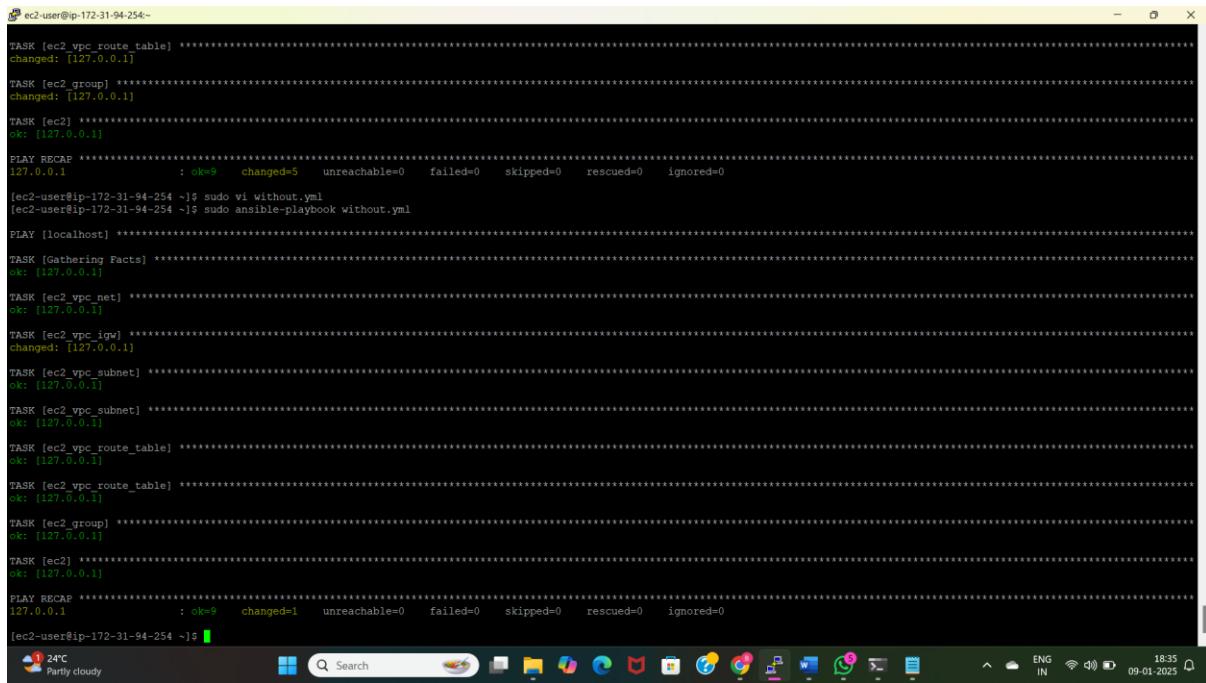
```

#####
# 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.



```
ec2-user@ip-172-31-94-254~
TASK [ec2_vpc_route_table] *****
changed: [127.0.0.1]

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

TASK [ec2] *****
ok: [127.0.0.1]

PLAY RECAP *****
127.0.0.1 : ok=9    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-94-254 ~]$ sudo vi without.yml
[ec2-user@ip-172-31-94-254 ~]$ sudo ansible-playbook without.yml

PLAY [localhost] *****

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

TASK [ec2_vpc_net] *****
ok: [127.0.0.1]

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

TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]

TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]

TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]

TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]

TASK [ec2_group] *****
ok: [127.0.0.1]

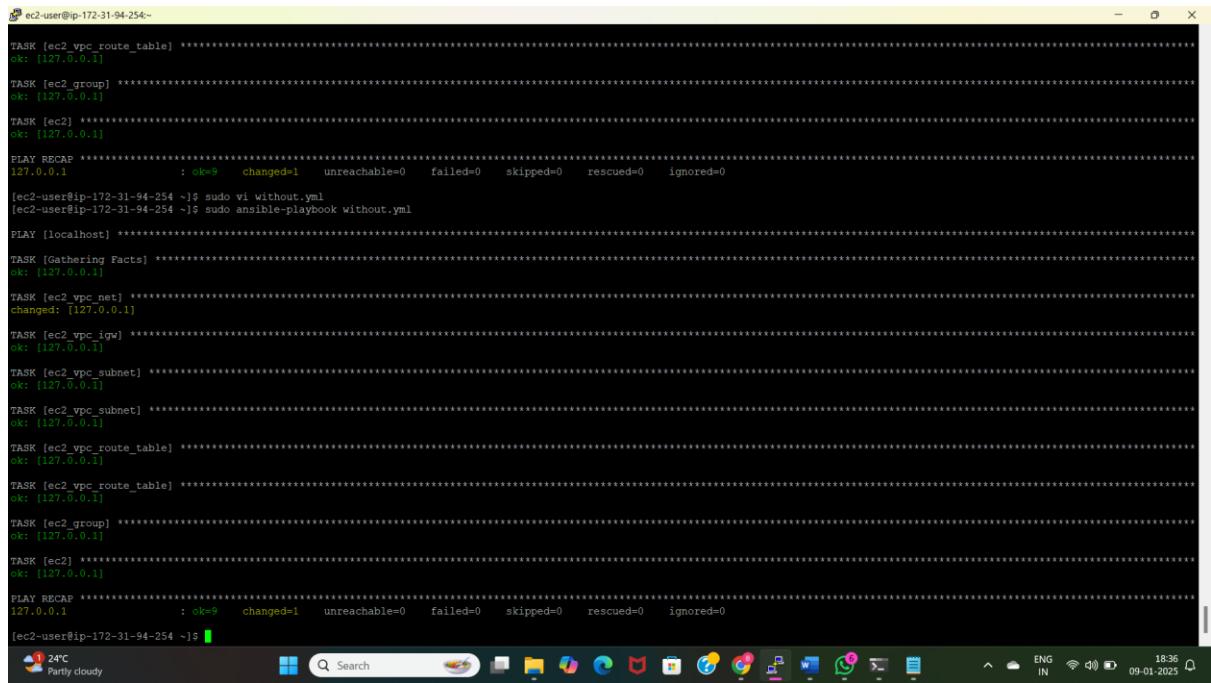
TASK [ec2] *****
ok: [127.0.0.1]

PLAY RECAP *****
127.0.0.1 : ok=9    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-94-254 ~]$
```

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 present
        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.



```
ec2-user@ip-172-31-94-254~
TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]
TASK [ec2_group] *****
ok: [127.0.0.1]
TASK [ec2] *****
ok: [127.0.0.1]
PLAY RECAP *****
127.0.0.1 : ok=9    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-94-254 ~]$ sudo vi without.yml
[ec2-user@ip-172-31-94-254 ~]$ sudo ansible-playbook without.yml
PLAY [localhost] *****
TASK [Gathering Facts] *****
ok: [127.0.0.1]
TASK [ec2_vpc_net] *****
changed: [127.0.0.1]
TASK [ec2_vpc_igw] *****
ok: [127.0.0.1]
TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]
TASK [ec2_vpc_subnet] *****
ok: [127.0.0.1]
TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]
TASK [ec2_vpc_route_table] *****
ok: [127.0.0.1]
TASK [ec2_group] *****
ok: [127.0.0.1]
TASK [ec2] *****
ok: [127.0.0.1]
PLAY RECAP *****
127.0.0.1 : ok=9    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-94-254 ~]$
```

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.