

# PROJECT – 5

## CREATING AN EC2 INSTANCE USING ANSIBLE

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

### What is Ansible?

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

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

### Key Features of Ansible

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

### Advantages of Ansible

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

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

## Disadvantages of Ansible

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

## Use Cases for Ansible

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

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

The screenshot shows the 'Launch an instance' wizard. In the 'Name and tags' section, 'Ansible' is entered in the 'Name' field. Under 'Application and OS Images (Amazon Machine Image)', 'Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type' is selected. The 'Virtual server type (instance type)' is set to 't2.micro'. The 'Free tier' information indicates it's eligible for the first year. The 'Summary' box shows 1 instance being launched. The 'Launch instance' button is visible at the bottom right.

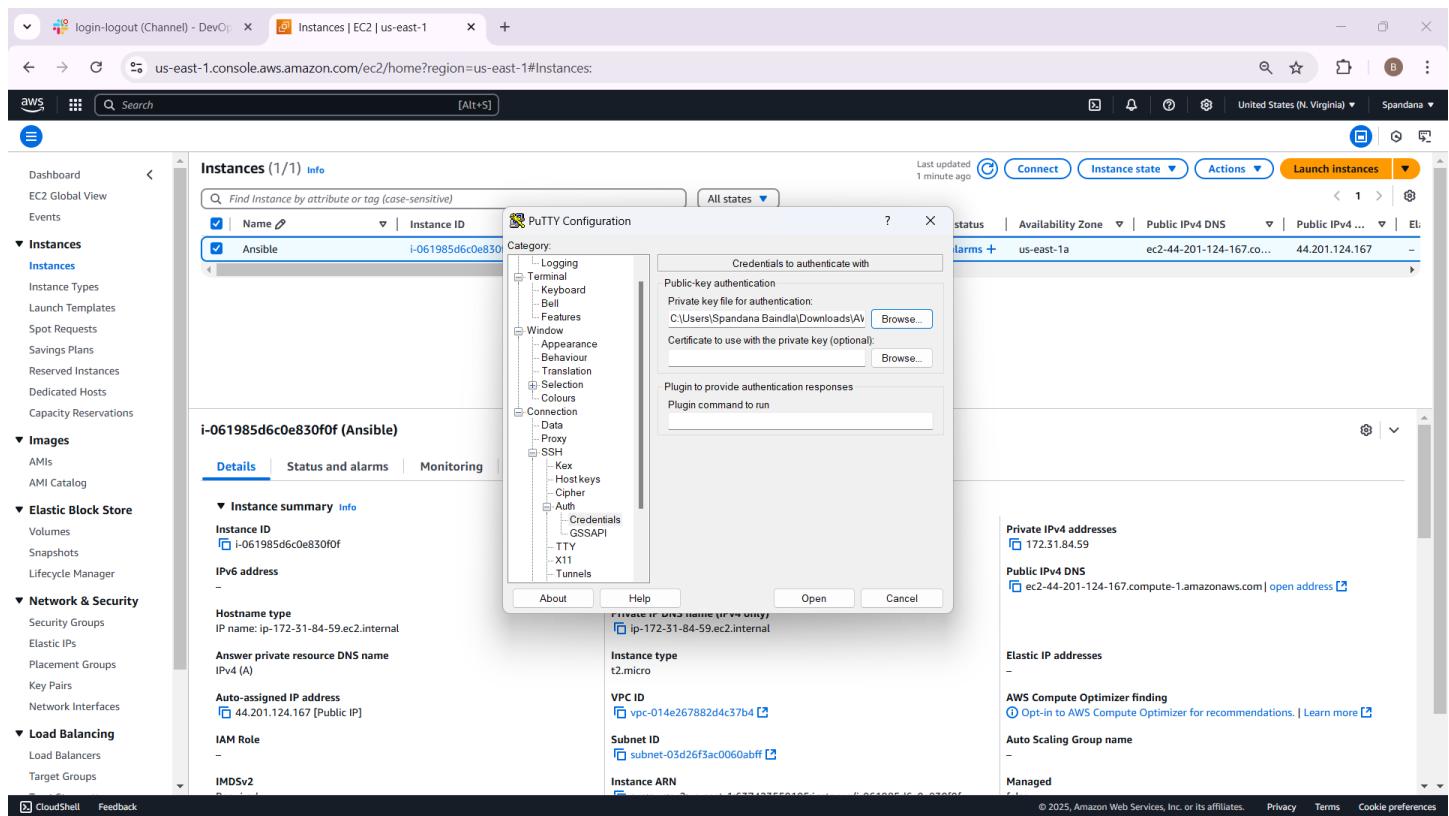
In the below image we can able to see instance state should be in running state as shown in below image.

The screenshot shows the 'Instances (1/1)' page. The instance 'Ansible' (ID: i-061985d6c0e830f0f) is listed with a status of 'Running'. The 'Actions' dropdown menu is open, showing options like 'Connect', 'Instance state', 'Actions', and 'Launch instances'. The instance details page for 'i-061985d6c0e830f0f (Ansible)' is shown, confirming the public IP address is 44.201.124.167 and the private IP address is 172.31.84.59.

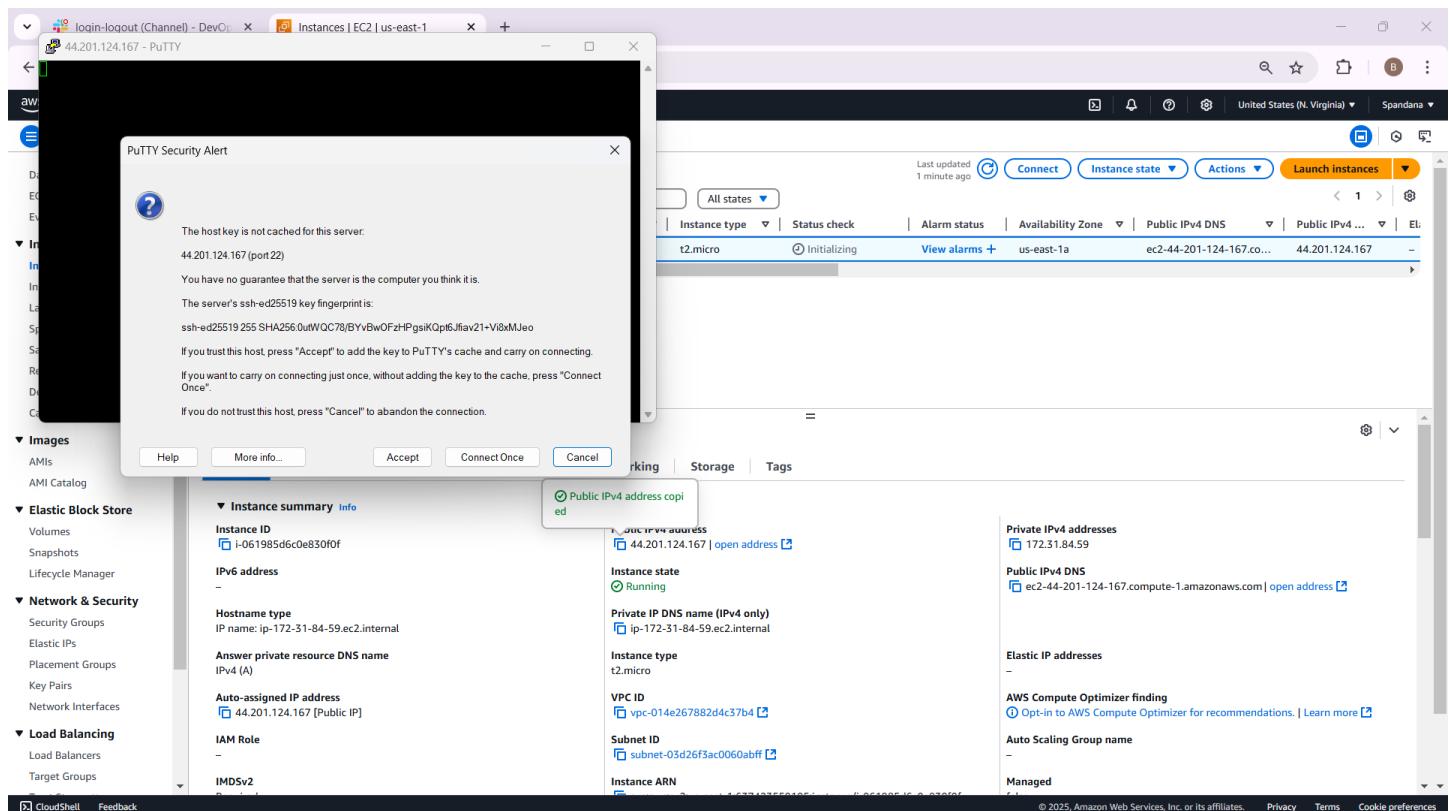
Then copy the Public IP of an instance and connect through PuTTY. First we need to copy and paste the public IP of an instance in Host Name place and in the left hand side we can see SSH > Auth > Credentials click on Open as shown in below image.

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

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



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



## Install Required Software on Your Local Computer

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

### 1. Python

#### What is Python?

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

#### Uses in the Project:

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

### 2. pip

#### What is pip?

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

#### Uses in the Project:

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

### 3. boto and boto3

#### What are boto and boto3?

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

#### Uses in the Project:

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

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

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

```
sudo yum install python3 python3-pip -y
```

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

```
ec2-user@ip-172-31-84-59:~$  
  login as: ec2-user  
  Authenticating with public key "imported-ssh-key"  
  ,  
  #  
  \### Amazon Linux 2  
  \### AL2 End of Life is 2025-06-30.  
  \#/  
  V~,-_>  
  / A newer version of Amazon Linux is available!  
  .-/ /  
  / Amazon Linux 2023, GA and supported until 2028-03-15.  
  /m/ https://aws.amazon.com/linux/amazon-linux-2023/  
  
7 package(s) needed for security, out of 8 available  
Run "sudo yum update" to apply all updates.  
[ec2-user@ip-172-31-84-59 ~]$ sudo apt install python3 python3-pip -y  
sudo: command not found  
[ec2-user@ip-172-31-84-59 ~]$ sudo yum install python3 python3-pip -y  
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd  
Package python3-3.7.16-1.amzn2.0.8.x86_64 already installed and latest version  
Package python3-pip-20.2.2-1.amzn2.0.8.noarch already installed and latest version  
Nothing to do  
[ec2-user@ip-172-31-84-59 ~]$ 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-py3-none-any.whl (1.4 MB)  
|██████████| 1.4 MB 22.2 MB/s  
Installing collected packages: boto  
Successfully installed boto-2.49.0  
[ec2-user@ip-172-31-84-59 ~]$ sudo pip3 install boto3  
WARNING: Running pip install with root privileges is generally not a good idea. Try 'pip3 install --user' instead.  
Collecting boto3  
  Downloading boto3-1.33.13-py3-none-any.whl (139 kB)  
|██████████| 139 kB 20.8 MB/s  
Collecting s3transfer<0.9.0,>=0.8.2  
  Downloading s3transfer-0.8.2-py3-none-any.whl (82 kB)  
|██████████| 82 kB 235 kB/s  
Collecting jmespath<2.0.0,>=0.7.1  
  Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)  
Collecting botocore<1.34.0,>=1.33.13  
  Downloading botocore-1.33.13-py3-none-any.whl (11.8 MB)  
|██████████| 11.8 MB 101 kB/s  
Collecting python-dateutil<3.0.0,>=2.1  
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)  
|██████████| 229 kB 87.3 MB/s  
Collecting urllib<1.27,>=1.25.4: python_version < "3.10"  
  Downloading urllib3-1.26.20-py2.py3-none-any.whl (144 kB)  
|██████████| 144 kB 87.9 MB/s  
Collecting six>=1.5  
  Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
```

```

ec2-user@ip-172-31-84-59:~ 
Installing collected packages: six, python-dateutil, jmespath, urllib3, botocore, s3transfer, boto3
Successfully installed boto3-1.33.13 botocore-1.33.13 jmespath-1.0.1 python-dateutil-2.9.0.p0
st0 s3transfer-0.8.2 six-1.17.0 urllib3-1.26.20
[ec2-user@ip-172-31-84-59 ~]$ 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 sqlite files removed
0 metadata files removed
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core | 3.6 kB 00:00:00
amzn2extra-ansible2 | 2.9 kB 00:00:00
amzn2extra-docker | 2.9 kB 00:00:00
amzn2extra-kernel-5.10 | 3.0 kB 00:00:00
(1/9): amzn2-core/2/x86_64/group.gz | 2.7 kB 00:00:00
(2/9): amzn2-core/2/x86_64/updateinfo | 1.0 MB 00:00:00
(3/9): amzn2extra-docker/2/x86_64/primary_db | 117 kB 00:00:00
(4/9): amzn2extra-kernel-5.10/2/x86_64/updateinfo | 98 kB 00:00:00
(5/9): amzn2extra-ansible2/2/x86_64/updateinfo | 4.7 kB 00:00:00
(6/9): amzn2extra-ansible2/2/x86_64/primary_db | 37 kB 00:00:00
(7/9): amzn2extra-docker/2/x86_64/updateinfo | 20 kB 00:00:00
(8/9): amzn2extra-kernel-5.10/2/x86_64/primary_db | 32 MB 00:00:00
(9/9): amzn2-core/2/x86_64/primary_db | 73 MB 00:00:01
Resolving Dependencies
--> Running transaction check
--> Package ansible.noarch 0:2.9.23-1.amzn2 will be installed
--> Processing Dependency: python-crypto for package: ansible-2.9.23-1.amzn2.noarch
--> Processing Dependency: python-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: libtomcrypt.so.1() (64bit) for package: python2-crypto-2.6.1-13.amz
n2.0.3.x86_64
--> Package python2-httplib2.noarch 0:0.18.1-3.amzn2 will be installed
--> Package python2-paramiko.noarch 0:1.16.1-3.amzn2.0.3 will be installed
--> Processing Dependency: python2-ecdsa for package: python2-paramiko-1.16.1-3.amzn2.0.3.no
rch
--> 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
--> Processing Dependency: libtommath.so.1() (64bit) for package: libtomcrypt-1.18.2-1.amzn2.0
.1.x86_64
--> Package python2-ecdsa.noarch 0:0.13.3-1.amzn2.0.1 will be installed
--> Running transaction check

```

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

#### ❖ sudo amazon-linux-extras install ansible2 -y

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

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

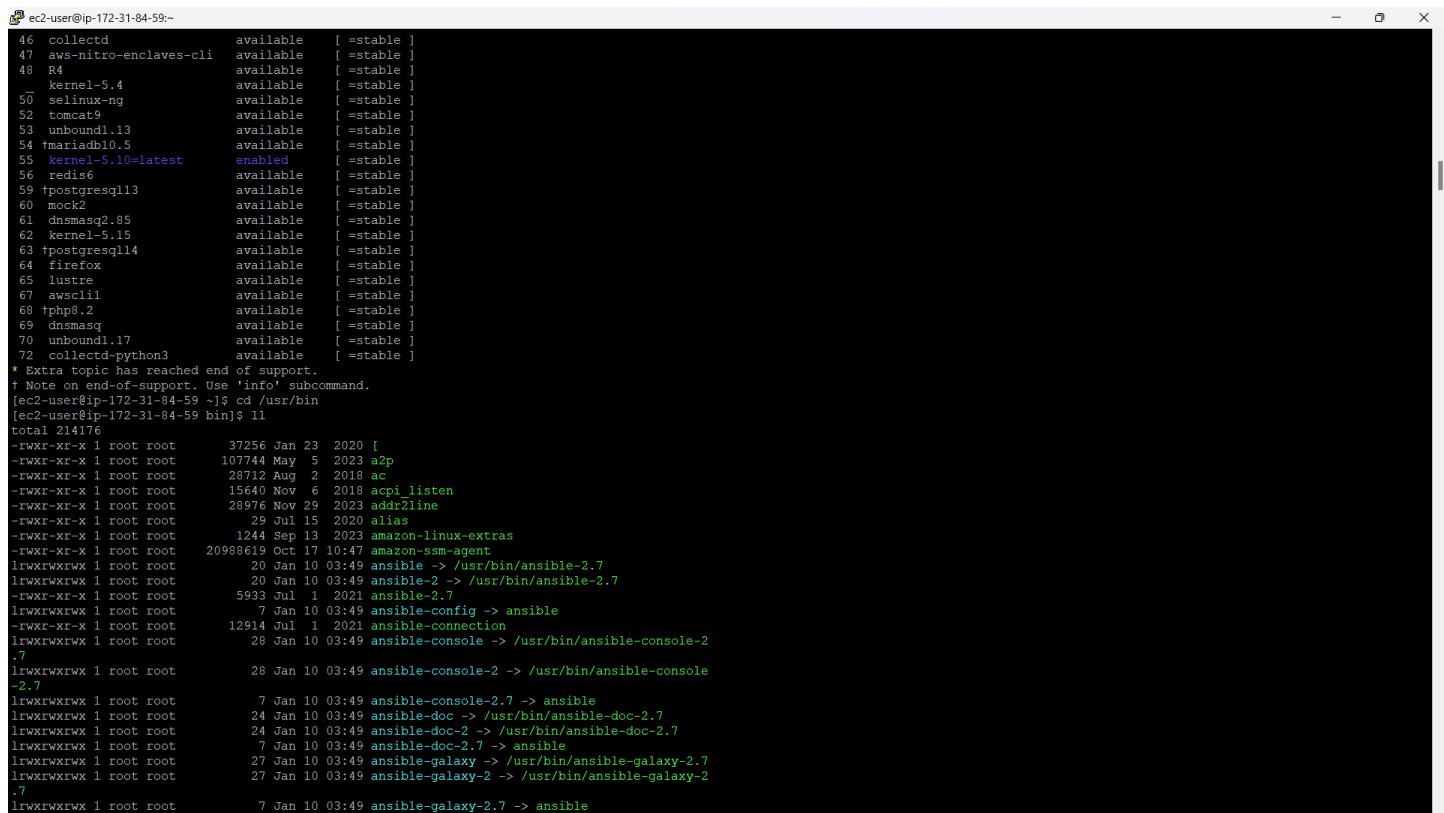
#### Navigate to the /usr/bin directory

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

## Install boto3 and botocore:

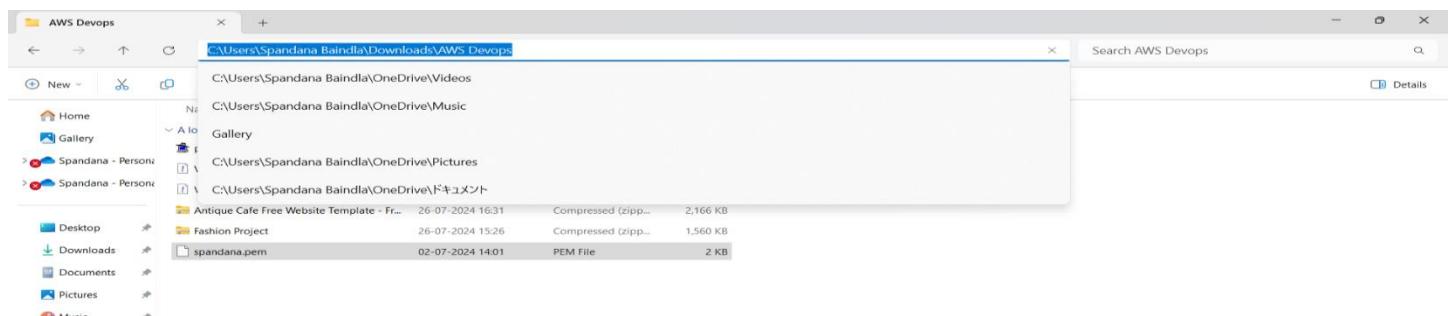
- sudo pip3 install boto3
- sudo pip3 install botocore
  - pip3 is the package manager for Python 3.
  - boto3 is the AWS SDK for Python, enabling interaction with AWS services.
  - botocore is a low-level core library used by boto3 for handling AWS service requests.

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

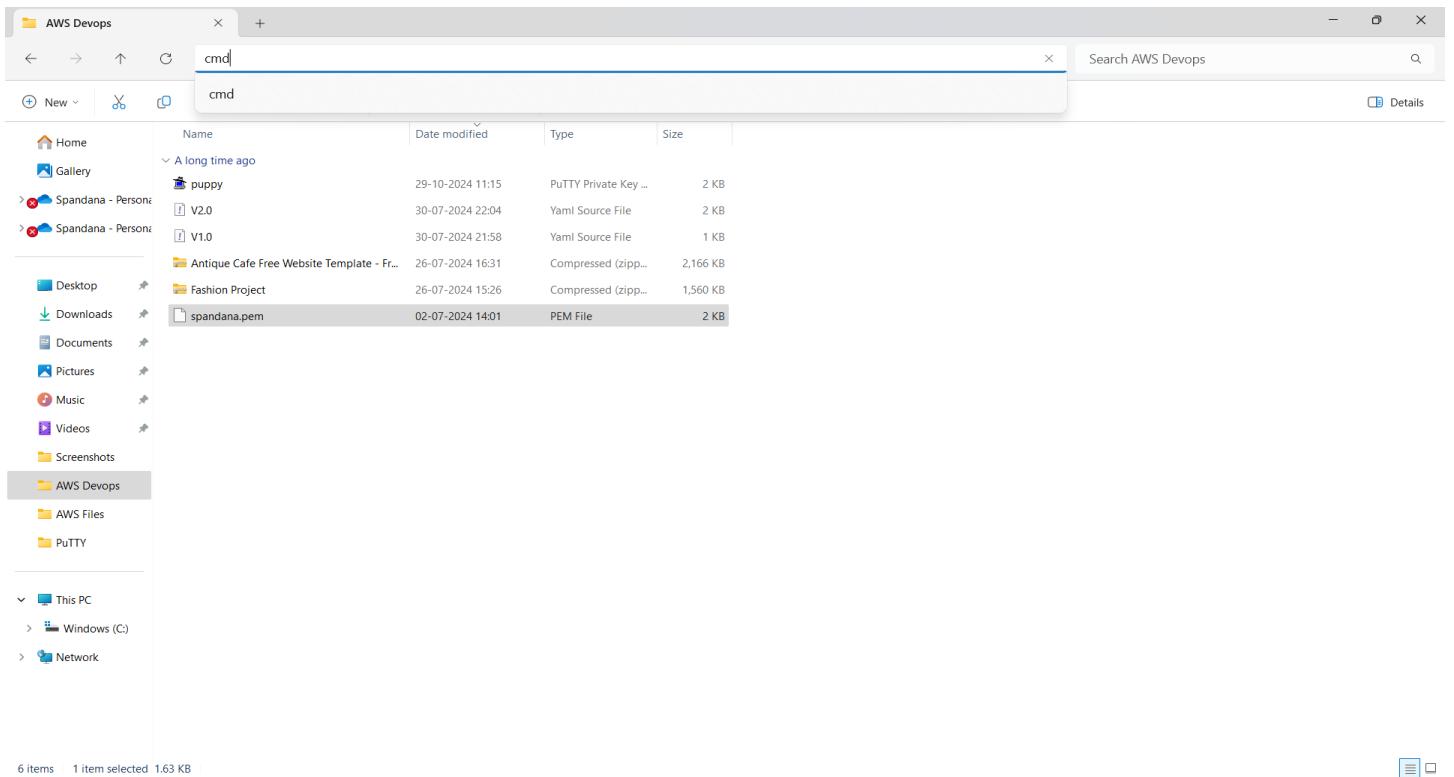


```
ec2-user@ip-172-31-84-59:~
46 collected      available  [ =stable ]
47 aws-nitro-enclaves-cli  available  [ =stable ]
48 R4             available  [ =stable ]
50 kernel-5.4     available  [ =stable ]
50 selinux-ng     available  [ =stable ]
52 tomcat9       available  [ =stable ]
53 unbound1.13   available  [ =stable ]
54 mariadb10.5   available  [ =stable ]
55 kernel-5.10=latest  enabled   [ =stable ]
56 redis          available  [ =stable ]
59 fpostgresql13 available  [ =stable ]
60 mock2          available  [ =stable ]
61 dnsmasq2.85   available  [ =stable ]
62 kernel-5.15   available  [ =stable ]
63 fpostgresql14 available  [ =stable ]
64 firefox        available  [ =stable ]
65 lustre         available  [ =stable ]
67 awscli1       available  [ =stable ]
68 tphp8.2        available  [ =stable ]
69 dnsmasq        available  [ =stable ]
70 unbound1.17   available  [ =stable ]
72 collected-python3 available  [ =stable ]
* Extra topic has reached end of support.
† Note on end-of-support. Use 'info' subcommand.
[ec2-user@ip-172-31-84-59 ~]$ cd /usr/bin
[ec2-user@ip-172-31-84-59 bin]$ ll
total 214176
-rwxr-xr-x 1 root root    37256 Jan 23 2020 [
-rwxr-xr-x 1 root root  107744 May  5 2023 a2p
-rwxr-xr-x 1 root root   28712 Aug  2 2018 ac
-rwxr-xr-x 1 root root   15640 Nov  6 2018 acpi_listen
-rwxr-xr-x 1 root root   28976 Nov 29 2023 addr2line
-rwxr-xr-x 1 root root    29 Jul 15 2020 alias
-rwxr-xr-x 1 root root   1244 Sep 13 2023 amazon-linux-extras
-rwxr-xr-x 1 root root 20988619 Oct 17 10:47 amazon-ssm-agent
lrwxrwxrwx 1 root root    20 Jan 10 03:49 ansible -> /usr/bin/ansible-2.7
lrwxrwxrwx 1 root root    20 Jan 10 03:49 ansible-2 -> /usr/bin/ansible-2.7
-rwxr-xr-x 1 root root   5933 Jul  1 2021 ansible-2.7
lrwxrwxrwx 1 root root    7 Jan 10 03:49 ansible-config -> ansible
-rwxr-xr-x 1 root root  12914 Jul  1 2021 ansible-connection
lrwxrwxrwx 1 root root   28 Jan 10 03:49 ansible-console -> /usr/bin/ansible-console-2
.7
lrwxrwxrwx 1 root root    28 Jan 10 03:49 ansible-console-2 -> /usr/bin/ansible-console
-2.7
lrwxrwxrwx 1 root root    7 Jan 10 03:49 ansible-console-2.7 -> ansible
lrwxrwxrwx 1 root root   24 Jan 10 03:49 ansible-doc -> /usr/bin/ansible-doc-2.7
lrwxrwxrwx 1 root root   24 Jan 10 03:49 ansible-doc-2 -> /usr/bin/ansible-doc-2.7
lrwxrwxrwx 1 root root    7 Jan 10 03:49 ansible-doc-2.7 -> ansible
lrwxrwxrwx 1 root root   27 Jan 10 03:49 ansible-galaxy -> /usr/bin/ansible-galaxy-2.7
lrwxrwxrwx 1 root root   27 Jan 10 03:49 ansible-galaxy-2 -> /usr/bin/ansible-galaxy-2
.7
lrwxrwxrwx 1 root root    7 Jan 10 03:49 ansible-galaxy-2.7 -> ansible
```

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



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



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

## 1. Command Executed:

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

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

## 2. Host Authenticity Check:

- Since it's your first connection to the EC2 instance, the authenticity of the host is not yet verified.
- The message asks for confirmation:  
Are you sure you want to continue connecting (yes/no/[fingerprint])?
- You responded with yes, so the host's key was added to your known hosts file.

## 3. File Transfer Confirmation:

- The file spandana.pem was successfully transferred to the EC2 instance.
- Details:
  - 100% 1678: Indicates the file size (1,678 bytes) and the completion status.
  - 3.4KB/s: The transfer speed.
  - 00:08: Time taken for the transfer.

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

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

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

The screenshot shows a CloudWatch Log Stream titled "ec2-user@ip-172-31-84-59:/usr/bin". It displays the command "scp -i spandana.pem spandana.pem ec2-user@44.201.124.167:/home/ec2-user/" followed by a warning about host fingerprint authentication. The user is prompted for confirmation ("Are you sure you want to continue connecting (yes/no/[fingerprint])?"). The response "Warning: Permanently added '44.201.124.167' (ED25519) to the list of known hosts." is shown, along with the transfer details: 100% 1678, 3.4KB/s, and 00:00. The log also shows the file being processed by various xz tools like xzcat, xzcmp, xzdiff, xzgrep, xzless, and xzmore.

```
ec2-user@ip-172-31-84-59:/usr/bin
[...]
lrwxrwxrwx 1 root root          2 Dec 17 18:39 xzcat -> xz
lrwxrwxrwx 1 root root          6 Dec 17 18:39 xzcmp -> xzdiff
-rwxr--r-- 1 root root        11384 Apr 14 2022 xzdiff
-rwxr--r-- 1 root root         6632 Apr 14 2022 xzdiff
lrwxrwxrwx 1 root root          4 Dec 17 18:39 xzgrep -> xzgrep
lrwxrwxrwx 1 root root          6 Dec 17 18:39 xzfgrep -> xzgrep
-rwxr--r-- 1 root root        5902 Apr 14 2022 xzgrep
-rwxr--r-- 1 root root         1802 Apr 14 2022 xzless
-rwxr--r-- 1 root root        2161 Apr 14 2022 xzmore

[...]
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.22631.4602]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Spandana Baindla\Downloads\AWS Devops>scp -i spandana.pem spandana.pem ec2-user@44.201.124.167:/home/ec2-user/
The authenticity of host '44.201.124.167 (44.201.124.167)' can't be established.
ED25519 key fingerprint is SHA256:0utWQC78/BVbWOfzHPgs1kQpt6Jfia21+Vi8MJe0.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
Warning: Permanently added '44.201.124.167' (ED25519) to the list of known hosts.
spandana.pem
100% 1678      3.4KB/s   00:00

C:\Users\Spandana Baindla\Downloads\AWS Devops>
```

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

### Explanation of the Inventory File Entry

#### [localhost]

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

##### 1. [localhost]:

This is the group name. In this case, it's labeled localhost, but you can use any name. It's essentially an identifier for the group of hosts that you want to manage.

##### 2. 127.0.0.1:

This is the loopback address (local address) for the machine where you're running Ansible (your local machine). Ansible will use this to connect to the EC2 instance.

##### 3. ansible\_host=172.31.84.59:

This defines the actual IP address of the EC2 instance you're managing. Ansible will connect to 172.31.84.59 to run commands on the EC2 instance.

##### 4. ansible\_python\_interpreter=/usr/bin/python3:

Specifies the Python interpreter that Ansible should use on the EC2 instance. Since Ansible runs on Python, this tells it to use Python 3, which is installed on the instance.

##### 5. ansible\_user=ec2-user:

This specifies the user Ansible should use when connecting to the EC2 instance. ec2-user is the default SSH user for Amazon Linux-based EC2 instances.

##### 6. ansible\_ssh\_private\_key\_file=/home/ec2-user/spandana.pem:

This tells Ansible to use the private key file (spandana.pem) to authenticate the connection when connecting to the EC2 instance. This file should have the correct permissions set (e.g., chmod 600).

### Why is This Important?

This configuration enables Ansible to:

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

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

```

ec2-user@ip-172-31-84-59:~ 
[localhost]
127.0.0.1 ansible_host=172.31.84.59 ansible_python_interpreter=/usr/bin/python3 ansible_user=ec2-user ansible_ssh_private_key_file=/home/ec2-user/spandana.pem
# This is the default ansible 'hosts' file.

# It should live in /etc/ansible/hosts

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

# Ex 1: Ungrouped hosts, specify before any group headers.

## green.example.com
## blue.example.com
## 192.168.100.1
## 192.168.100.10

# Ex 2: A collection of hosts belonging to the 'webservers' group

[webservers]
## alpha.example.org
## beta.example.org
## 192.168.1.100
## 192.168.1.110

# If you have multiple hosts following a pattern you can specify
# them like this:

## www[001:006].example.com

# Ex 3: A collection of database servers in the 'dbservers' group

[dbservers]
##
## db01.intranet.mydomain.net
## db02.intranet.mydomain.net
## 10.25.1.56
## 10.25.1.57

# Here's another example of host ranges, this time there are no
# leading 0s:

## db-[99:101]-node.example.com

~ 
~ 
~ 
~ 
:wq!

```

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

## Ansible Playbook script with outline variables

### vpc.yaml file

#### 1. Setting Up the Playbook:

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

#### 2. VPC Creation:

The ec2\_vpc\_net module creates a Virtual Private Cloud (VPC) with specific configurations:

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

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

### **3. Internet Gateway (IGW) Creation:**

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

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

## 1. Public Subnet Creation

The public subnet is created with the following properties:

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

## 2. Private Subnet Creation

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

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

```

Ansible_Playbook script with outline variables ×
C: > Users > Spandana Baindla > Downloads > ! Ansible_Playbook script with outline variables
25 ##### Internet gateway Creation #####
37 ## create an vpc pub subnet
38
39
40     - ec2_vpc_subnet:
41         aws_access_key: "{{ aws_access_key }}"
42         aws_secret_key: "{{ aws_secret_key }}"
43         vpc_id: "{{ vpc_result.vpc.id }}"
44         region: "{{ region }}"
45         az: "{{ pubzone }}"          # az is the availability zone
46         state: present
47         cidr: "{{ pubsubnet_cidr_block }}"
48         # enable public ip
49         map_public: yes
50         resource_tags:
51             Name: "{{ pubsubnet_name }}"
52         register: pubsubnet_result
53
54 ## create an vpc pvt subnet
55
56
57     - ec2_vpc_subnet:
58         aws_access_key: "{{ aws_access_key }}"
59         aws_secret_key: "{{ aws_secret_key }}"
60         vpc_id: "{{ vpc_result.vpc.id }}"
61         region: "{{ region }}"
62         az: "{{ pvtzone }}"          # az is the availability zone
63         state: present
64         cidr: "{{ pvtsubnet_cidr_block }}"
65         # enable public ip
66         map_public: no
67         resource_tags:
68             Name: "{{ pvtsubnet_name }}"
69         register: pvtsubnet_result
70
71 ## create an vpc pub route table
72
73     - ec2_vpc_route_table:

```

Ln 179, Col 28 Spaces:2 UTF-8 CRLF YAML

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

## 1. Public Route Table Creation

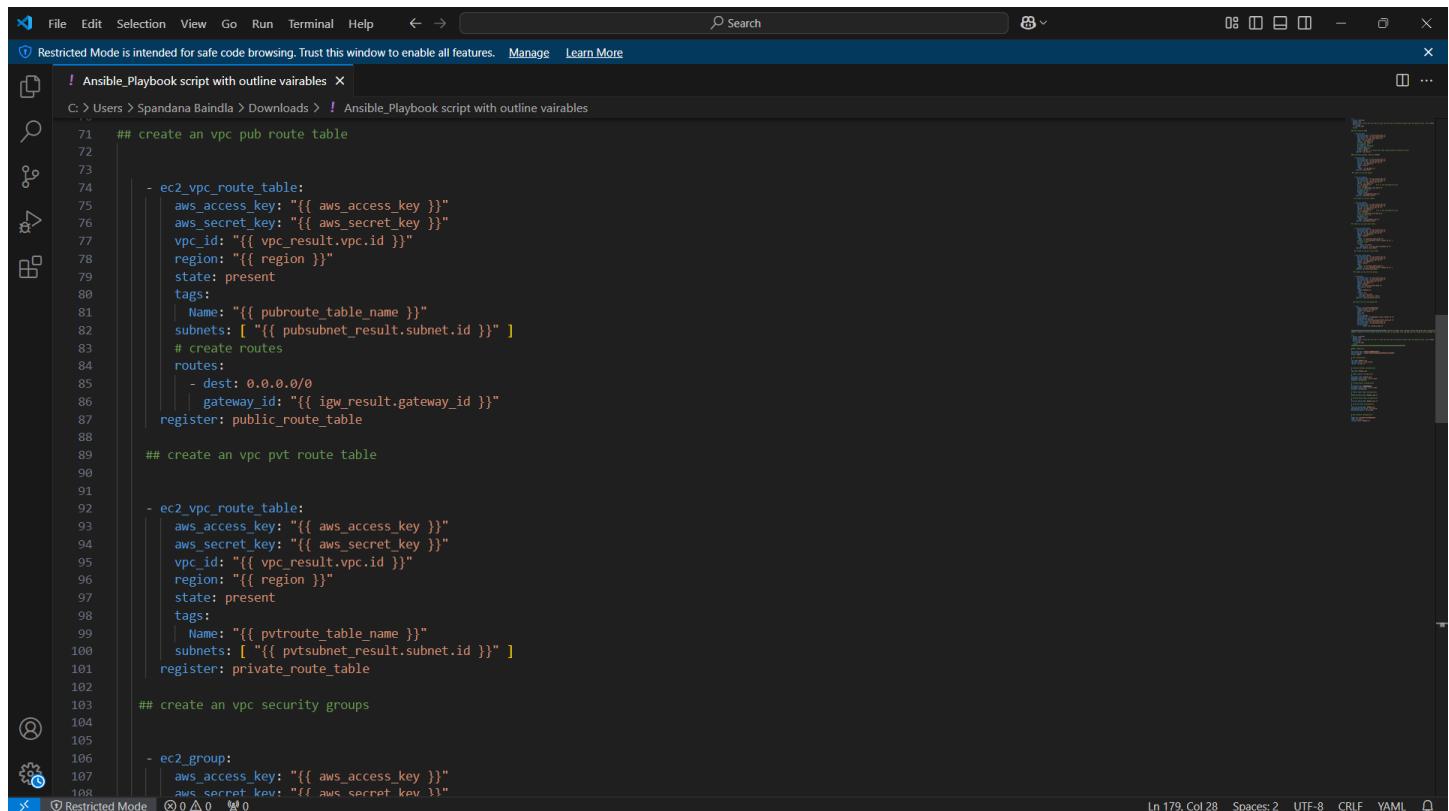
The public route table is created using the `ec2_vpc_route_table` module with the following details:

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

## 2. Private Route Table Creation

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

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



A screenshot of a code editor window titled "Ansible\_Playbook script with outline variables". The file path is "C:\Users\Spandana Baindla\Downloads\Ansible\_Playbook script with outline variables". The code is written in YAML and defines three route tables: a public route table, a private route table, and a security group. The public route table is associated with a specific VPC and its subnets. The private route table is also associated with the same VPC and its subnets. The security group is created within the same VPC. The code uses AWS credentials and specifies the AWS region.

```
## create an vpc pub route table
71 - ec2_vpc_route_table:
72   aws_access_key: "{{ aws_access_key }}"
73   aws_secret_key: "{{ aws_secret_key }}"
74   vpc_id: "{{ vpc_result.vpc.id }}"
75   region: "{{ region }}"
76   state: present
77   tags:
78     Name: "{{ pubroute_table_name }}"
79   subnets: [ "{{ pubsubnet_result.subnet.id }}" ]
80   # create routes
81   routes:
82     - dest: 0.0.0.0/0
83       gateway_id: "{{ igw_result.gateway_id }}"
84   register: public_route_table
85
86 ## create an vpc pvt route table
87
88 - ec2_vpc_route_table:
89   aws_access_key: "{{ aws_access_key }}"
90   aws_secret_key: "{{ aws_secret_key }}"
91   vpc_id: "{{ vpc_result.vpc.id }}"
92   region: "{{ region }}"
93   state: present
94   tags:
95     Name: "{{ pvtroute_table_name }}"
96   subnets: [ "{{ pvtsubnet_result.subnet.id }}" ]
97   register: private_route_table
98
99 ## create an vpc security groups
100
101 - ec2_group:
102   aws_access_key: "{{ aws_access_key }}"
103   aws_secret_key: "{{ aws_secret_key }}"
104
105
106
107
108
```

## 1. Create a VPC Security Group

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

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

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

## 2. Launch an EC2 Instance

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

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

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

**Below we have secrets.yaml file**

## General Information

- Access Keys:
    - aws\_access\_key: "ARTM7"
    - aws\_secret\_key: "Oz7 "
  - These keys are used to authenticate and authorize actions in the AWS account.
  - Title: "ARTH" (It might be a project name or an identifier for the setup.)

## VPC Configuration

- VPC Name: Spandana\_vpc  
A virtual private cloud named "Spandana\_vpc" will be created.

- CIDR Block: 15.0.0.0/16  
Specifies the IP range for the VPC.
- Region: us-east-1  
All resources will be provisioned in the "US East (N. Virginia)" region.

## Internet Gateway Configuration

- Name: Spandana\_igu  
An internet gateway will be created and attached to the VPC for external connectivity.

## Public Subnet Configuration

- Name: Spandana\_pub  
A public subnet will be created with the following:
  - CIDR Block: 11.0.1.0/24  
Allocates a specific range of IPs for the subnet.
  - Availability Zone: us-east-1a  
The subnet will reside in the specified availability zone.

## Private Subnet Configuration

- Name: Spandana\_pvt  
A private subnet will be created with the following:
  - CIDR Block: 11.0.2.0/24  
Allocates a specific range of IPs for the subnet.
  - Availability Zone: us-east-1b  
The subnet will reside in the specified availability zone.

## Public Route Table Configuration

- Name: Spandana\_pub\_rt  
A public route table will be created and associated with the public subnet. It will include a route to the internet gateway, enabling internet access for resources in the public subnet.

## **Private Route Table Configuration**

- Name: Spandana\_pvt\_rt

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

## **Security Group Configuration**

- Name: Spandana\_sg

A security group named "Spandana\_sg" will be created with the following rules:

- Allow All Traffic: CIDR block 10.0.0.0/0

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

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

This rule allows SSH access to the instances.

## **EC2 Instance Configuration**

- Image ID: ami-0caf660076a6e32

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

- Instance Name: Spandana\_ec2

The instance will be tagged with this name.

- Instance Type:

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

```

ec2-user@ip-172-31-84-59:~ - D X
-->
aws access_key: "AKIAZI2LHP2MWS2MDIM7"
aws secret_key: "mjrzOZhGK70dd2wr9dexTREmV4AD40Elaxiej6K"
title: "ARTH"

# VPC configuration
#
vpc_name: Spandana_vpc
vpc_cidr_block: '11.0.0.0/16'
region: "us-east-1"

# Internet Gateway configuration
#
igw_name: Spandana_igw

# Public Subnet configuration
#
pubsubnet_name: Spandana_pub
pubsubnet_cidr_block: '11.0.1.0/24'
pubzone: "us-east-1a"

# Private Subnet configuration
#
pvtsubnet_name: Spandana_pvt
pvtsubnet_cidr_block: '11.0.2.0/24'
pvtzone: "us-east-1b"

# Public Route Table configuration
#
pubroute_table_name: Spandana_pub_rt

# Private Route Table configuration
#
pvtroute_table_name: Spandana_pvt_rt

# Security Group configuration
#
security_group_name: Spandana_sg
destination_cidr_block: '0.0.0.0/0'
port22_cidr_block: '0.0.0.0/0'

# EC2 Instance configuration
#
image_id: "ami-0ca9fb66e076a6e32"
type: "t2.micro"
instance_name: Spandan[ec
~
~
~
```

48,23

All

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

### **Ansible-playbook:**

This is the primary command to run an Ansible playbook.

### **vpc.yaml:**

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

### **–syntax-check:**

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

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

```

ec2-user@ip-172-31-84-59:~ 
-rwxr-xr-x 1 root root      2953 Oct 10  2008 zipgrep
-rwxr-xr-x 2 root root    189472 Dec  1  2022 zipinfo
-rwxr-xr-x 1 root root     95896 Aug  1  2018 zipnote
-rwxr-xr-x 1 root root    99968 Aug  1  2018 zipsplit
-rwxr-xr-x 1 root root     2041 Apr 14  2022 zless
-rwxr-xr-x 1 root root     2859 Apr 14  2022 zmore
-rwxr-xr-x 1 root root    5343 Apr 14  2022 znew
lwxrwxrwx 1 root root          6 Dec 17 10:39 zsoelim -> soelim
[ec2-user@ip-172-31-84-59 bin]$ sudo pip3 install boto3 botocore
WARNING: Running pip install with root privileges is generally not a good idea. Try 'pip3 install --user' instead.
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)
Requirement already satisfied: s3transfer<0.9.0,>=0.8.2 in /usr/local/lib/python3.7/site-packages (from boto3) (0.8.2)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/local/lib/python3.7/site-packages (from boto3) (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; python_version < "3.10" 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)
[ec2-user@ip-172-31-84-59 bin]$ cd
[ec2-user@ip-172-31-84-59 ~]$ ll
total 4
-rw-rw-r-- 1 ec2-user ec2-user 1678 Jan 10 03:52 spandana.pem
[ec2-user@ip-172-31-84-59 ~]$ sudo chmod 600 < pem-file-name>.pem
-bash: pem-file-name: No such file or directory
[ec2-user@ip-172-31-84-59 ~]$ sudo chmod 600 spandana.pem
[ec2-user@ip-172-31-84-59 ~]$ sudo vi /etc/ansible/hosts
[ec2-user@ip-172-31-84-59 ~]$ sudo vi vpc.yaml
[ec2-user@ip-172-31-84-59 ~]$ sudo vi secret.yaml
[ec2-user@ip-172-31-84-59 ~]$ ansible-playbook vpc.yaml --syntax-check
ERROR! 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 '-' indicator

The error appears to be in '/home/ec2-user/secret.yaml': line 12, column 1, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

#
aws_access_key: "AKIAZT2LHP2MXCK05XR7"
^ here
[ec2-user@ip-172-31-84-59 ~]$ ansible-playbook secret.yaml --syntax-check
ERROR! 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

```

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

### ansible-playbook:

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

### vpc.yaml:

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

```

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

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

The offending line appears to be:

---
aws_access_key: "AKIAZI2LHP2MWS2MDIM7"
^ here
[ec2-user@ip-172-31-84-59 ~]$ ansible-playbook vpc.yaml --syntax-check

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

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

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

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

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

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

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

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

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

PLAY RECAP ****
127.0.0.1 : ok=8    changed=8      unreachable=0   failed=0    skipped=0   rescued=0   ignored=0
[ec2-user@ip-172-31-84-59 ~]$ 

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

PLAY RECAP ****
127.0.0.1 : ok=8    changed=8      unreachable=0   failed=0    skipped=0   rescued=0   ignored=0
[ec2-user@ip-172-31-84-59 ~]$ 

```

Ec2 instance has been created successfully as shown below.

The screenshot shows the AWS EC2 Instances page. The main table lists three instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
Spandana-ec	i-089399afaa5e36e67	Running	t2.micro	Initializing	View alarms +	us-east-1a	ec2-54-172-219-135.co...	54.172.219.135
Ansible	i-061985d6c0e830f0f	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1a	ec2-44-201-124-167.co...	44.201.124.167
Rabbani-ec	i-026272c087505dc58	Shutting-d...	t2.micro	-	View alarms +	us-east-1a	ec2-18-212-196-53.co...	18.212.196.53

Details for the Spandana-ec instance:

- Public IPv4 address: 54.172.219.135 [open address]
- Instance state: Running
- Private IP DNS name (IPv4 only): ip-11-0-1-218.ec2.internal
- Instance type: t2.micro
- VPC ID: vpc-07af0804bc99988d3 (Spandana\_vpc)
- Subnet ID: subnet-0a090eb8476c953a (Spandana\_pub)
- Instance ARN: arn:aws:ec2:us-east-1:123456789012:instance/i-089399afaa5e36e67

VPC has been created successfully as shown below.

The screenshot shows the AWS VPCs page. The main table lists three VPCs:

Name	VPC ID	State	Block Public...	IPv4 CIDR	IPv6 CIDR	DHCP option set	Main route table
Rabbani_vpc	vpc-0b7c72627232b5883	Available	Off	11.0.0.0/16	-	dopt-08010d2dc2ef4a5ce	rtb-0d3c4653cd9
Spandana_vpc	vpc-07af0804bc99988d3	Available	Off	11.0.0.0/16	-	dopt-08010d2dc2ef4a5ce	rtb-0081a7bbe12
-	ypc-014e267882d4c37b4	Available	Off	172.31.0.0/16	-	dopt-08010d2dc2ef4a5ce	rtb-02a08c64822

Details for the Spandana\_vpc VPC:

- VPC ID: vpc-07af0804bc99988d3
- State: Available
- Tenancy: default
- Default VPC: No
- IPv6 CIDR (Network border group): -
- Block Public Access: Off
- DHCP option set: dopt-08010d2dc2ef4a5ce
- IPv4 CIDR: 11.0.0.0/16
- Network Address Usage metrics: Disabled
- Route 53 Resolver DNS Firewall rule groups: -
- DNS hostnames: Enabled
- Main route table: rtb-0081a7bbe128e943f
- IPv6 pool: -
- Owner ID: 637423550105

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

This screenshot shows the AWS VPC Subnets console. On the left, there's a sidebar with navigation links like 'VPC dashboard', 'Virtual private cloud', 'Subnets', 'Security', etc. The main area displays a table of subnets. One row, 'Spandana\_pvt', is selected and highlighted in blue. The table columns include Name, Subnet ID, State, VPC, Block Public Access, IPv4 CIDR, IPv6 CIDR, and IPv6. Below the table, a detailed view for 'subnet-02da5f2376554ae6 / Spandana\_pvt' is shown. It includes sections for Details, Network ACL, and Tags. The 'Details' section provides specific subnet information such as Subnet ID, State, IPv4 CIDR, Availability Zone, Route table, Auto-assign IPv6 address, and IPv4 CIDR reservations. The 'Network ACL' section lists the associated Network ACL (acl-0b283e45295a84a00). The 'Tags' section shows no tags are assigned.

The below is the public subnet of the vpc.

This screenshot shows the AWS VPC Subnets console, similar to the previous one but for a different subnet. The selected subnet is 'Spandana\_pub'. The table and detailed view are identical to the private subnet, showing its configuration and network settings. The 'Details' section for 'subnet-0a090eb847d6c953a / Spandana\_pub' includes the same fields as the private subnet, such as Subnet ID, State, IPv4 CIDR, Availability Zone, Route table, Auto-assign IPv6 address, and IPv4 CIDR reservations. The 'Network ACL' section also lists the associated Network ACL (acl-0b283e45295a84a00).

Successfully public subnet has been created as shown below.

The screenshot shows the AWS VPC Route Tables console. The left sidebar navigation includes 'VPC dashboard', 'Virtual private cloud' (with 'Your VPCs' and 'Route tables' selected), 'Security', and 'PrivateLink and Lattice'. The main content area displays a table of route tables with one row selected: 'rtb-0a512f2a742b5b8fb / Spandana\_pub\_rt'. The 'Details' tab is active, showing the route table ID, VPC (vpc-07af0804bc99988d3 | Spandana\_vpc), and explicit subnet associations: 'subnet-0a90eb847d6c953a / Spandana\_pub'. The 'Main' column is set to 'No'.

Successfully private subnet has been created successfully as shown below.

The screenshot shows the AWS VPC Route Tables console. The left sidebar navigation includes 'VPC dashboard', 'Virtual private cloud' (with 'Your VPCs' and 'Route tables' selected), 'Security', and 'PrivateLink and Lattice'. The main content area displays a table of route tables with one row selected: 'rtb-04a1e9acc25485b64 / Spandana\_pvt\_rt'. The 'Details' tab is active, showing the route table ID, VPC (vpc-07af0804bc99988d3 | Spandana\_vpc), and explicit subnet associations: 'subnet-02da5f23765544ae6 / Spandana\_pvt'. The 'Main' column is set to 'No'.

Successfully internet gateway has been created as shown below.

The screenshot shows the AWS VPC dashboard with the 'Internet gateways' section selected. The table lists three internet gateways:

Name	Internet gateway ID	State	VPC ID	Owner
Spandana_igw	igw-06a7764279fb8a1a1	Attached	vpc-07af0804bc99988d3   Spandana_vpc	637423550105
-	igw-0917bbbd60434e117	Attached	vpc-014e267882d4c37b4	637423550105
Rabbani_igw	igw-09c93e7fce97a2c64	Attached	vpc-0b7c72627232b5883   Rabbani_vpc	637423550105

Details for the first gateway (igw-06a7764279fb8a1a1 / Spandana\_igw) are shown:

Internet gateway ID	State	VPC ID	Owner
igw-06a7764279fb8a1a1	Attached	vpc-07af0804bc99988d3   Spandana_vpc	637423550105

Security groups has been created successfully has shown below.

The screenshot shows the AWS Security Groups console with the 'Security groups' section selected. The table lists seven security groups:

Name	Security group ID	Security group name	VPC ID	Description	Owner
-	sg-0baa426a0a6fd1e32	default	vpc-0b7c72627232b5883	default VPC security group	637423550105
Spandana-sg	sg-08f749cf1c8b9c7f1	Spandana_sg	vpc-07af0804bc99988d3	allow	637423550105
-	sg-0eca3a3e229961b07	launch-wizard-1	vpc-014e267882d4c37b4	launch-wizard-1 created 2024-12-20T0...	637423550105
Rabbani-sg	sg-07d042aa4e5f1ff4	Rabbani_sg	vpc-0b7c72627232b5883	allow	637423550105
-	sg-064a73e5f0b14efdd	default	vpc-07af0804bc99988d3	default VPC security group	637423550105
-	sg-08642703e678efa0c	devops-sg	vpc-014e267882d4c37b4	allow	637423550105
-	sg-0534beff44a8aa	default	vpc-014e267882d4c37b4	default VPC security group	637423550105

Details for the first gateway (sg-08f749cf1c8b9c7f1 - Spandana\_sg) are shown:

Inbound rules	Outbound rules	Sharing - new	VPC associations - new	Tags
(1)				

Inbound rules table:

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
-	sgr-0ddd6b7edb30b3d54	IPv4	All traffic	All	All	0.0.0.0/0	allow all traffic

## Ansible\_Playbook script without variables

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

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

### Explanation of playbook:

#### VPC Creation script explanation :

ec2\_vpc\_net: This defines a task that interacts with AWS EC2 to create a VPC.

aws\_access\_key: The AWS Access Key is used to authenticate your connection with AWS services.

aws\_secret\_key: The AWS Secret Key, used along with the Access Key, is also part of the authentication for AWS.

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

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

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

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

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

#### Internet Gateway creation script explanation :

- ec2\_vpc\_igw: This task creates an Internet Gateway (IGW) for the VPC.
- aws\_access\_key: The Access Key for AWS authentication.
- aws\_secret\_key: The Secret Key for AWS authentication.
- vpc\_id: "{{ vpc\_result.vpc.id }}" This dynamically references the ID of the VPC created in the previous task (vpc\_result holds this ID).
- region: The region where the Internet Gateway will be created (us-east-1).
- state: present Ensures the Internet Gateway is created. If it already exists, nothing will change.

- tags: Assigns the tag Name: Puppyligw to the Internet Gateway.
- register: igw\_result Saves the details of the created Internet Gateway (like its ID) in the igw\_result variable

### Creation of Public subnet script :

- ec2\_vpc\_subnet: This task creates a Subnet inside the VPC.
- aws\_access\_key: The Access Key for AWS authentication.
- aws\_secret\_key: The Secret Key for AWS authentication.
- vpc\_id: "{{ vpc\_result.vpc.id }}" References the VPC ID created earlier.
- region: us-east-1 The region for the subnet (us-east-1).
- az: us-east-1a The Availability Zone in which to create the subnet (us-east-1a).
- state: present Ensures the subnet is created.
- cidr: 10.0.0.0/20 The CIDR block for the subnet (10.0.0.0/20 means 4096 IP addresses, with the first 4 bits fixed).
- map\_public: yes This indicates the subnet will automatically assign public IPs to instances (making it a public subnet).
- resource\_tags: Adds a tag with the name Puppy-pub to the subnet.

```
ec2-user@ip-172-31-84-59:~$ cat vpc.py
hosts: localhost
become: yes
tasks:
- ec2_vpc_net:
    aws_access_key: AKIAZI2LHP2MWS2MDLM7
    aws_secret_key: mjrOZHozGK7ddZwr9dexTREmV4AD40Elaxiej6K
    cidr_block: 10.0.0.0/16
    name: Puppy-vpc
    region: us-east-1
    state: present # to delete Vpc then replace absent instead of present
    register: vpc_result

### Internet gateway Creation #####
- ec2_vpc_igw:
    aws_access_key: AKIAZI2LHP2MWS2MDLM7
    aws_secret_key: mjrOZHozGK7ddZwr9dexTREmV4AD40Elaxiej6K
    vpc_id: "{{ vpc_result.vpc.id }}"
    region: us-east-1
    state: present
    tags:
        Name: Puppy-igw
    register: igw_result

## create an vpc pub subnet

- ec2_vpc_subnet:
    aws_access_key: AKIAZI2LHP2MWS2MDLM7
    aws_secret_key: mjrOZHozGK7ddZwr9dexTREmV4AD40Elaxiej6K
    vpc_id: "{{ vpc_result.vpc.id }}"
    region: us-east-1
    az: us-east-1a      # az is the availability zone
    state: present
    cidr: 10.0.0.0/20
    # enable public ip
    map_public: yes
    resource_tags:
        Name: Puppy-pub
    register: pubsubnet_result

## create an vpc pvt subnet

- ec2_vpc_subnet:
    aws_access_key: AKIAZI2LHP2MWS2MDLM7
    aws_secret_key: mjrOZHozGK7ddZwr9dexTREmV4AD40Elaxiej6K
    vpc_id: "{{ vpc_result.vpc.id }}"
    region: us-east-1
    az: us-east-1b      # az is the availability zone
-- INSERT --
```

23,22 Top

## **Creation of private subnet :**

- ec2\_vpc\_subnet: This task creates another subnet, but this one will be private.
- aws\_secret\_key: The Secret Key for AWS authentication.
- vpc\_id: "{{ vpc\_result.vpc.id }}" References the VPC ID created earlier.
- region: us-east-1 The region where the subnet will be created (us-east-1).
- az: us-east-1b The Availability Zone where this private subnet will be created (us-east-1b).
- state: present Ensures the subnet is created.
- cidr: 10.0.1.0/20 The CIDR block for this subnet (10.0.1.0/20), providing a range of IPs.
- map\_public: no This means no public IP addresses will be assigned to instances in this subnet (making it a private subnet).
- resource\_tags: Adds a tag with the name Puppy-pvt to the subnet.
- register: pvtsubnet\_result The details of the private subnet are saved in pvtsubnet\_result.

## **Creation of Public route table script:**

- ec2\_vpc\_route\_table: Creates a route table for a public subnet to route traffic, including to the internet through the internet gateway (IGW).
- aws\_access\_key: The AWS access key for authentication.
- aws\_secret\_key: The AWS secret key for authentication.
- vpc\_id: "{{ vpc\_result.vpc.id }}" References the VPC where the route table will be created.
- region: us-east-1 The region for the route table (us-east-1).
- state: present Ensures that the route table is created.
- tags: Tags the route table with the name Puppy-pub-rt.
- subnets: Specifies the subnet ID (pubsubnet\_result.subnet.id) where this route table will be used. It associates the public subnet with the route table.
- routes: Defines the routes for the table.
  - dest: 0.0.0.0/0: This route means all traffic (0.0.0.0/0) will be forwarded to the internet gateway (gateway\_id: "{{ igw\_result.gateway\_id }}").
- register: public\_route\_table Saves the result of this task (e.g., route table ID) to the public\_route\_table variable.

## Creation of private route table :

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

```
ec2-user@ip-172-31-84-59:~
- ec2_vpc_subnet:
  aws_access_key: AKIAZI2LHP2MWS2MDLM7
  aws_secret_key: mjr0ZHozGK70dd2wr9dexTREmV4AD40Elaxiej6K
  vpc_id: "({ vpc_result.vpc.id })"
  region: us-east-1
  az: us-east-1b    # az is the availability zone
  state: present
  cidr: 10.0.16.0/20
  # enable public ip
  map_public: no
  resource_tags:
    Name: Puppy-pvt
  register: pvtsubnet_result

## create an vpc pub route table

- ec2_vpc_route_table:
  aws_access_key: AKIAZI2LHP2MWS2MDLM7
  aws_secret_key: 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
  vpc_id: "({ vpc_result.vpc.id })"
  region: us-east-1
  state: present
  tags:
    Name: Puppy-pub-rt
  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: AKIAZI2LHP2MWS2MDLM7
  aws_secret_key: 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
  vpc_id: "({ vpc_result.vpc.id })"
  region: us-east-1
  state: present
  tags:
    Name: Puppy-pvt-rt
  subnets: [ "({ pvtsubnet_result.subnet.id })" ]
  register: private_route_table

## create an vpc security groups

- ec2_group:
  aws_access_key: AKIAZI2LHP2MWS2MDLM7
-- INSERT --
```

**ec2\_security\_group:** This task creates a security group within the VPC to control access to instances.

**aws\_access\_key:** The AWS access key for authentication.

**aws\_secret\_key:** The AWS secret key for authentication.

**vpc\_id: "{{ vpc\_result.vpc.id }}"** References the VPC where the security group will be created.

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

**group\_name: Puppy-sg** The name for the security group (Puppy-sg).

**description:** A brief description of the security group.

**ip\_permissions:** Specifies the rules for allowing traffic.

- **ip\_protocol:** tcp: Allows TCP traffic.
- **from\_port:** '22', **to\_port:** '22': Allows SSH access on port 22.
- **ip\_ranges:** cidr\_ip: 0.0.0.0/0: Allows access from any IP address.

**register: security\_group\_result** Saves the details of the created security group (like its ID) to security\_group\_result.

```
ec2-user@ip-172-31-84-59:~
```

```
## create an vpc pvt route table

- ec2_vpc_route_table:
  aws_access_key: AKIAZIZLHP2MWS2MDLM7
  aws_secret_key: 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
  vpc_id: "({ vpc_result.vpc.id })"
  region: us-east-1
  state: present
  tags:
    Name: Puppy-pvt-rt
  subnets: [ "({ vpcsubnet_result.subnet.id })" ]
  register: private_route_table

## create an vpc security groups

- ec2_group:
  aws_access_key: AKIAZIZLHP2MWS2MDLM7
  aws_secret_key: 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
  vpc_id: "({ vpc_result.vpc.id })"
  region: us-east-1
  state: present
  name: Puppy-sg
  description: allow
  tags:
    Name: tina-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: t2.micro
  region: us-east-1
  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: AKIAZIZLHP2MWS2MDLM7
  aws_secret_key: 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
  instance_tags:
    Name: Puppy-Ec
-- INSERT --
```

128,22 Bot

## EC2 Instance Creation:

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

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

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

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

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

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

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

```

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

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

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

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

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

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

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

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

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

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

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

PLAY RECAP ****
127.0.0.1 : ok=9    changed=8    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
[ec2-user@ip-172-31-84-59 ~]$ 

```

New Instance has been created successfully as shown below.

The screenshot shows the AWS CloudWatch terminal interface. The terminal window title is "login-logoff (Channel) - DevOp". The terminal content displays the execution of an Ansible playbook named "VPC.yaml". The output shows various Ansible tasks being run on a local host (127.0.0.1), such as gathering facts, creating a VPC network, internet gateway, subnet, route tables, and security groups. The final "PLAY RECAP" section indicates that 9 tasks were successful (ok=9), 8 tasks resulted in changes (changed=8), and there were no errors (failed=0). The terminal prompt at the bottom is "[ec2-user@ip-172-31-84-59 ~]\$".

## New security group has been created successfully.

The screenshot shows the AWS EC2 Security Groups page. On the left, there's a navigation sidebar with sections like Instances, Images, Elastic Block Store, Network & Security, Load Balancing, and CloudShell. The main content area displays a table titled "Security Groups (1/5) Info". A new security group, "Puppy-sg" (sg-0065e030c212fbde7), has been created and is highlighted. The table includes columns for Name, Security group ID, Security group name, VPC ID, Description, and Owner. The "Details" tab for the selected security group is open, showing its configuration: Security group name is "Puppy-sg", Security group ID is "sg-0065e030c212fbde7", Owner is "637423550105", Inbound rules count is "1 Permission entry", and Outbound rules count is "1 Permission entry".

New vpc has been created successfully as shown below.

The screenshot shows the AWS VPC console. The left sidebar includes sections for VPC dashboard, Virtual private cloud (Your VPCs), Security, PrivateLink and Lattice, and CloudShell. The main area displays a table titled "Your VPCs (1/2) Info". A new VPC, "Puppy-vpc" (vpc-0570b119898fc4de4), is listed with a status of "Available". The table columns include Name, VPC ID, State, Block Public..., IPv4 CIDR, IPv6 CIDR, DHCP option set, and Main route table. The "Details" tab for the selected VPC is open, showing its configuration: VPC ID is "vpc-0570b119898fc4de4", State is "Available", Tenancy is "default", Default VPC is "No", IPv4 CIDR is "10.0.0.0/16", Network Address Usage metrics is "Disabled", and Route 53 Resolver DNS Firewall rule groups is "-". Other details shown include DNS hostnames (Enabled), Main route table (rtb-0f126ce0cbba225cc), IPv6 pool (-), and Owner ID (637423550105).

New Private subnet has been created successfully as shown below.

**Subnets (1/8) Info**

Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR	IPv6 CIDR	IPv6 Cl
-	subnet-04313c5575f4c8c2b	Available	vpc-014e267882d4c37b4	Off	172.31.0.0/20	-	-
-	subnet-03d26f3ac0060abff	Available	vpc-014e267882d4c37b4	Off	172.31.80.0/20	-	-
-	subnet-0c85ac199c0740b9f	Available	vpc-014e267882d4c37b4	Off	172.31.16.0/20	-	-
-	subnet-05b7cf61a1fe49e8e	Available	vpc-014e267882d4c37b4	Off	172.31.32.0/20	-	-
-	subnet-0c0a60992779b0d64	Available	vpc-014e267882d4c37b4	Off	172.31.64.0/20	-	-
-	subnet-0038fa9df930e78d5	Available	vpc-014e267882d4c37b4	Off	172.31.48.0/20	-	-
Puppy-pvt	subnet-0ac0ab847b587a59b	Available	vpc-0570b119898fc4de4   Puppy-pvt	Off	10.0.16.0/20	-	-
Puppy-pub	subnet-003c27d926b06700f	Available	vpc-0570b119898fc4de4   Puppy-pub	Off	10.0.0.0/20	-	-

**subnet-0ac0ab847b587a59b / Puppy-pvt**

**Details**

Subnet ID subnet-0ac0ab847b587a59b	Subnet ARN arn:aws:ec2:us-east-1:637423550105:subnet/subnet-0ac0ab847b587a59b	State Available	Block Public Access Off
IPv4 CIDR 10.0.16.0/20	Available IPv4 addresses 4091	IPv6 CIDR -	IPv6 CIDR association ID -
Availability Zone us-east-1b	Availability Zone ID use1-a2z	Network border group us-east-1	VPC vpc-0570b119898fc4de4   Puppy-pvt
Route table rtb-00705de72b8769050   Puppy-pvt-rt	Network ACL acl-0f89379cdd47916d0	Default subnet No	Auto-assign public IPv4 address No
Auto-assign IPv6 address No	Auto-assign customer-owned IPv4 address No	Customer-owned IPv4 pool -	Outpost ID -
IPv4 CIDR reservations		IPv6-only	Hostname type

New public subnet has been created successfully has shown below.

**Subnets (1/8) Info**

Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR	IPv6 CIDR	IPv6 Cl
-	subnet-04313c5575f4c8c2b	Available	vpc-014e267882d4c37b4	Off	172.31.0.0/20	-	-
-	subnet-03d26f3ac0060abff	Available	vpc-014e267882d4c37b4	Off	172.31.80.0/20	-	-
-	subnet-0c85ac199c0740b9f	Available	vpc-014e267882d4c37b4	Off	172.31.16.0/20	-	-
-	subnet-05b7cf61a1fe49e8e	Available	vpc-014e267882d4c37b4	Off	172.31.32.0/20	-	-
-	subnet-0c0a60992779b0d64	Available	vpc-014e267882d4c37b4	Off	172.31.64.0/20	-	-
-	subnet-0038fa9df930e78d5	Available	vpc-014e267882d4c37b4	Off	172.31.48.0/20	-	-
Puppy-pvt	subnet-0ac0ab847b587a59b	Available	vpc-0570b119898fc4de4   Puppy-pvt	Off	10.0.16.0/20	-	-
Puppy-pub	subnet-003c27d926b06700f	Available	vpc-0570b119898fc4de4   Puppy-pub	Off	10.0.0.0/20	-	-

**subnet-003c27d926b06700f / Puppy-pub**

**Details**

Subnet ID subnet-003c27d926b06700f	Subnet ARN arn:aws:ec2:us-east-1:637423550105:subnet/subnet-003c27d926b06700f	State Available	Block Public Access Off
IPv4 CIDR 10.0.0.0/20	Available IPv4 addresses 4090	IPv6 CIDR -	IPv6 CIDR association ID -
Availability Zone us-east-1a	Availability Zone ID use1-a2z	Network border group us-east-1	VPC vpc-0570b119898fc4de4   Puppy-pvt
Route table rtb-054728c9255c391f0   Puppy-pub-rt	Network ACL acl-0f89379cdd47916d0	Default subnet No	Auto-assign public IPv4 address Yes
Auto-assign IPv6 address No	Auto-assign customer-owned IPv4 address No	Customer-owned IPv4 pool -	Outpost ID -
IPv4 CIDR reservations		IPv6-only	Hostname type

Public route table has been created successfully as shown below.

VPC dashboard

Route tables (1/4) Info

Name	Route table ID	Explicit subnet associ...	Edge associations	Main	VPC	Owner ID
Puppy-pub-rt	rtb-054728c9255c391f0	subnet-003c27d926b06700f	-	No	vpc-0570b119898fc4de4   Puppy-vpc	637423550105
-	rtb-0f126ce0cba225cc	-	-	Yes	vpc-0570b119898fc4de4   Puppy-vpc	637423550105
Puppy-pvt-rt	rtb-00705de72b8769050	subnet-0ac0ab847b587a59b	-	No	vpc-0570b119898fc4de4   Puppy-vpc	637423550105
-	rtb-02a08c64822c08da8	-	-	Yes	vpc-014e267882d4c37b4	637423550105

rtb-054728c9255c391f0 / Puppy-pub-rt

Details Routes Subnet associations Edge associations Route propagation Tags

**Details**

Route table ID rtb-054728c9255c391f0	Main <input type="checkbox"/> No	Explicit subnet associations subnet-003c27d926b06700f / Puppy-pub	Edge associations -
VPC vpc-0570b119898fc4de4   Puppy-vpc	Owner ID 637423550105		

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Private subnet has been created successfully as shown below.

VPC dashboard

Route tables (1/4) Info

Name	Route table ID	Explicit subnet associ...	Edge associations	Main	VPC	Owner ID
Puppy-pub-rt	rtb-054728c9255c391f0	subnet-003c27d926b06700f	-	No	vpc-0570b119898fc4de4   Puppy-vpc	637423550105
-	rtb-0f126ce0cba225cc	-	-	Yes	vpc-0570b119898fc4de4   Puppy-vpc	637423550105
Puppy-pvt-rt	rtb-00705de72b8769050	subnet-0ac0ab847b587a59b	-	No	vpc-0570b119898fc4de4   Puppy-vpc	637423550105
-	rtb-02a08c64822c08da8	-	-	Yes	vpc-014e267882d4c37b4	637423550105

rtb-00705de72b8769050 / Puppy-pvt-rt

Details Routes Subnet associations Edge associations Route propagation Tags

**Details**

Route table ID rtb-00705de72b8769050	Main <input type="checkbox"/> No	Explicit subnet associations subnet-0ac0ab847b587a59b / Puppy-pvt	Edge associations -
VPC vpc-0570b119898fc4de4   Puppy-vpc	Owner ID 637423550105		

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

New internet gateway has been created successfully has shown below.

The screenshot shows the AWS VPC dashboard with the 'Internet gateways' section selected. A table lists two internet gateways:

Name	Internet gateway ID	State	VPC ID	Owner
Puppy-igw	jgw-04526915066d08fba	Attached	vpc-0570b119898fc4de4   Puppy-vpc	637423550105
	jgw-0917bbbd60434e117	Attached	vpc-014e267882d4c37b4	637423550105

Details for the first gateway (jgw-04526915066d08fba) are shown in the details pane:

Internet gateway ID jgw-04526915066d08fba	State Attached	VPC ID vpc-0570b119898fc4de4   Puppy-vpc	Owner 637423550105
--	-------------------	---	-----------------------

New network ACLs has been created successfully has shown below.

The screenshot shows the AWS VPC dashboard with the 'Network ACLs' section selected. A table lists two network ACLs:

Name	Network ACL ID	Associated with	Default	VPC ID	Inbound rules count	Outbound rules count
ad-0cc99e51760a4e66	5 Subnets		Yes	vpc-014e267882d4c37b4	2 Inbound rules	2 Outbound rules
acl-0f89379cd47916d0	2 Subnets		Yes	vpc-0570b119898fc4de4   Puppy-vpc	2 Inbound rules	2 Outbound rules

Details for the second network ACL (acl-0f89379cd47916d0) are shown in the details pane:

Network ACL ID acl-0f89379cd47916d0	Associated with 2 Subnets	Default Yes	VPC ID vpc-0570b119898fc4de4   Puppy-vpc
--	------------------------------	----------------	---

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

The screenshot shows the AWS VPC console interface. On the left, there's a navigation sidebar with options like VPC dashboard, EC2 Global View, Virtual private cloud, Security, PrivateLink and Lattice, and CloudShell. The main area is titled "Your VPCs (1/2) Info" and shows two VPCs: "Puppy-vpc" and another unnamed one. Below this, the "Resource map" tab is selected, displaying a hierarchical diagram. At the top level is the "VPC Show details" node for "Puppy-vpc". This connects to two "Subnets (2)" nodes: "us-east-1a" and "us-east-1b". From each subnet, arrows point to two "Route tables (3)" nodes: "Puppy-pub-rt", "rtb-0f126ce0cbba225cc", and "Puppy-pvt-rt". Finally, arrows from the route tables point to a single "Network connections (1)" node labeled "Puppy-igw".

## Ansible Playbook script with inline variables

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

### Explanation of vpcinline.yaml

#### 1. hosts: localhost

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

#### 2. become: yes

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

#### 3. gather\_facts: false

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

#### 4. vars (Variables)

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

- VPC and Resources: Names and configurations of AWS resources like VPC (vpc\_name), subnets (pubsubnet\_name, pvtsubnet\_name), route tables (pubroute\_table\_name, pvtroute\_table\_name), security group (security\_group\_name), etc.
- CIDR Blocks: These define the IP address ranges for VPC and subnets (vpc\_cidr\_block, pubsubnet\_cidr\_block, pvtsubnet\_cidr\_block).
- Region and Availability Zones: Specifies the AWS region (us-east-1) and the availability zones (pubzone, pvtzone).
- Instance Configuration: image\_id, type, and instance\_name are the EC2 instance configurations such as the AMI ID, instance type (t2.micro), and instance name (Baindla-ec).

## 5. tasks

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

### **VPC Creation Task Explanation:**

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

#### **1. ec2\_vpc\_net:**

- This is an Ansible module used to create or manage an AWS VPC.

#### **2. aws\_access\_key and aws\_secret\_key:**

- These variables are used for authenticating with AWS to perform the operation. They are passed from the playbook's variables (aws\_access\_key, aws\_secret\_key).

#### **3. cidr\_block:**

- This specifies the IP address range for the VPC (in this case, vpc\_cidr\_block), which is 11.0.0.0/16 based on the variable. This will define the available range of private IP addresses for the resources inside this VPC.

#### **4. name:**

- The name of the VPC being created. The value of vpc\_name is dynamically provided from the variables (Baindla\_vpc).

## 5. region:

- This defines the AWS region where the VPC will be created (e.g., us-east-1).

## 6. dns\_support and dns\_hostnames:

- These options are set to yes to enable DNS support and DNS hostnames for instances within the VPC. Enabling this allows instances to resolve DNS names in the VPC.

## 7. tenancy:

- This defines the tenancy of the VPC. The value default means that instances in the VPC will use shared hardware. If you want dedicated hardware, you could set this to dedicated.

## 8. state: present:

- This indicates that the task should ensure the VPC is created. If the state were set to absent, it would delete the VPC instead of creating it.

## 9. register: vpc\_result:

- The result of this task (the VPC details) will be stored in the variable vpc\_result. This can be used later in the playbook for further actions, such as retrieving the VPC ID or using it in other tasks.

```
ec2-user@ip-172-31-84-59:~
```

```
- 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: "AKIAZI2LHP2MWS2MDLM7"
- aws_secret_key: "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"
- title: "ARTH"
- vpc_name: Baindla_vpc
- igw_name: Baindla_igw
- pubsubnet_name: Baindla_pub
- pvtsubnet_name: Baindla_pvt
- pubroute_table_name: Baindla_pub_rt
- pvtroute_table_name: Baindla_pvt_rt
- security_group_name: Baindla_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: Baindla-ec

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 present
  register: vpc_result
```

## Internet Gateway Creation Task:

1. **ec2\_vpc\_igw:**
  - This Ansible module is used to create an Internet Gateway (IGW) for the VPC.
2. **aws\_access\_key** and **aws\_secret\_key:**
  - These provide AWS credentials to authenticate the request.
3. **vpc\_id:**
  - The VPC ID is dynamically retrieved from the result of the previous task (i.e., vpc\_result.vpc.id), which holds the information about the created VPC.
4. **region:**
  - Specifies the AWS region where the IGW will be created, passed from the region variable.
5. **state: present:**
  - Indicates that the task should ensure the creation of the Internet Gateway. If state: absent, it would delete the IGW.
6. **tags:**
  - Assigns the name of the Internet Gateway using the variable igw\_name (e.g., Baindla\_igw).
7. **register: igw\_result:**
  - Stores the result of the task (details about the IGW) in the igw\_result variable, which can be used later.

## Public Subnet Creation Task:

1. **ec2\_vpc\_subnet:**
  - This module is used to create a subnet inside the VPC.
2. **aws\_access\_key** and **aws\_secret\_key:**
  - These credentials authenticate the request to AWS.
3. **vpc\_id:**
  - The VPC ID from the previous task (vpc\_result.vpc.id) is passed to create the subnet in the correct VPC.
4. **region:**
  - Specifies the region for the subnet creation, passed from the region variable.
5. **az:**
  - Specifies the Availability Zone for the subnet, passed from the pubzone variable (e.g., us-east-1a).
6. **state: present:**
  - Ensures that the subnet is created.

## 7. **cidr:**

- Defines the CIDR block for the subnet using the pubsubnet\_cidr\_block variable (e.g., 11.0.1.0/24).

## 8. **map\_public: yes:**

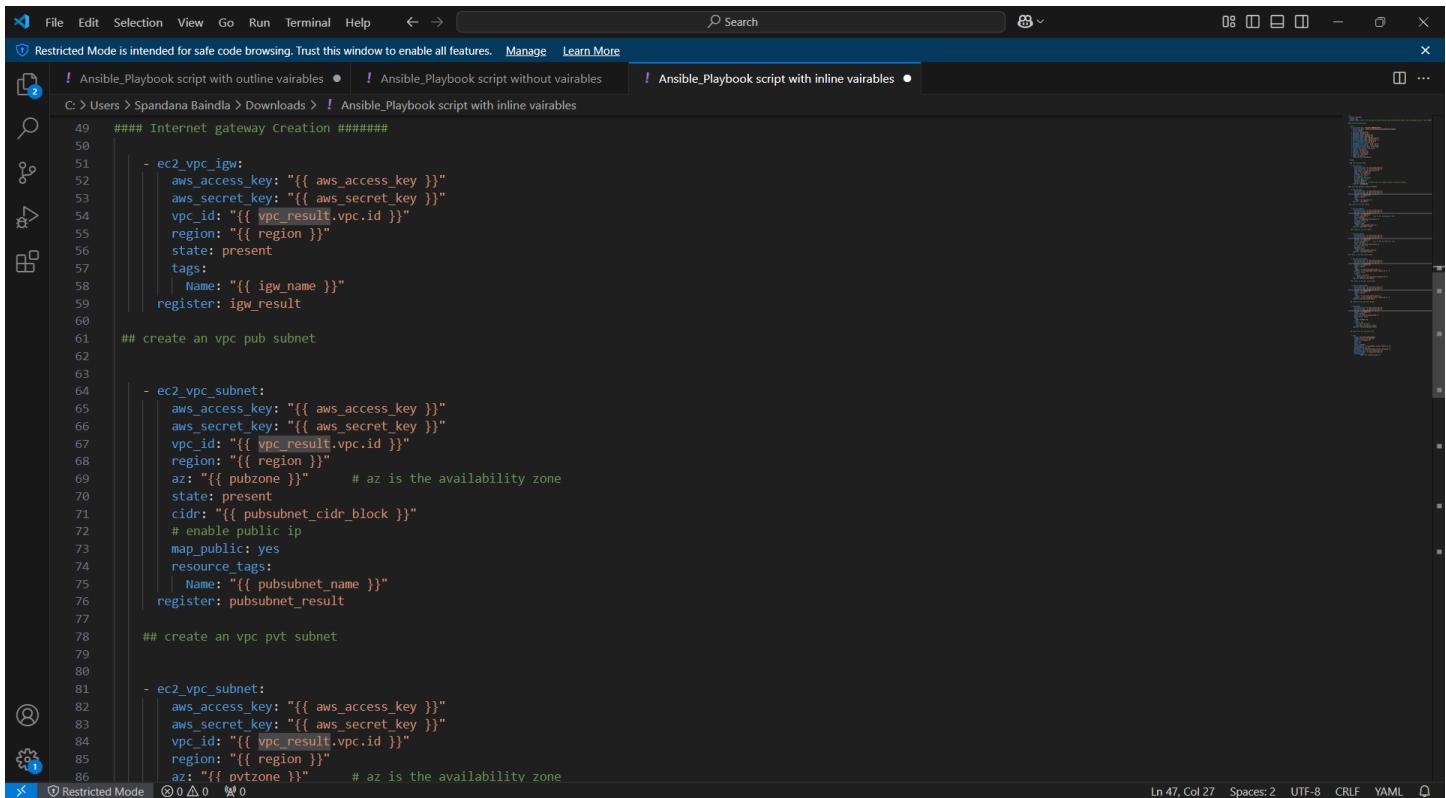
- Enables the automatic assignment of public IP addresses to instances launched in this subnet. This is useful for creating a public-facing subnet.

## 9. **resource\_tags:**

- Assigns a tag with the name of the subnet, using the pubsubnet\_name variable (e.g., Baindla\_pub).

## 10. **register: pubsubnet\_result:**

- Stores the result of the subnet creation in the pubsubnet\_result variable, which can be referenced later.



The screenshot shows the Sublime Text code editor with an open file named 'Ansible\_Playbook script with inline variables'. The code is as follows:

```
49     ##### Internet gateway Creation #####
50
51     - ec2_vpc_igw:
52         aws_access_key: "{{ aws_access_key }}"
53         aws_secret_key: "{{ aws_secret_key }}"
54         vpc_id: "{{ vpc_result.vpc.id }}"
55         region: "{{ region }}"
56         state: present
57         tags:
58             | Name: "{{ igw_name }}"
59         register: igw_result
60
61     ## create an vpc pub subnet
62
63
64     - ec2_vpc_subnet:
65         aws_access_key: "{{ aws_access_key }}"
66         aws_secret_key: "{{ aws_secret_key }}"
67         vpc_id: "{{ vpc_result.vpc.id }}"
68         region: "{{ region }}"
69         az: "{{ pubzone }}"          # az is the availability zone
70         state: present
71         cidr: "{{ pubsubnet_cidr_block }}"
72         # enable public ip
73         map_public: yes
74         resource_tags:
75             | Name: "{{ pubsubnet_name }}"
76         register: pubsubnet_result
77
78     ## create an vpc pvt subnet
79
80
81     - ec2_vpc_subnet:
82         aws_access_key: "{{ aws_access_key }}"
83         aws_secret_key: "{{ aws_secret_key }}"
84         vpc_id: "{{ vpc_result.vpc.id }}"
85         region: "{{ region }}"
86         az: "{{ pvtzone }}"        # az is the availability zone
```

The status bar at the bottom right indicates 'Ln 47, Col 27' and 'Spaces:2'.

## Create Private Subnet:

### 1. **ec2\_vpc\_subnet:**

- **aws\_access\_key** and **aws\_secret\_key**: These are the AWS credentials for authentication.
- **vpc\_id**: The VPC ID for the subnet, taken from vpc\_result.vpc.id (created in a previous task).
- **region**: The AWS region, passed from the region variable.
- **az**: Specifies the Availability Zone for the subnet, here pvtzone (e.g., us-east-1b).

- **state: present:** Ensures that the subnet is created.
- **cidr:** The CIDR block for the subnet, passed from pvtsubnet\_cidr\_block (e.g., 11.0.2.0/24).
- **map\_public: no:** This disables the automatic assignment of public IP addresses to instances in the private subnet.
- **resource\_tags:** Tags the subnet with a name, taken from the pvtsubnet\_name variable (e.g., Baindla\_pvt).
- **register: pvtsubnet\_result:** Registers the result of the task in the variable pvtsubnet\_result to be used later.

## Create Public Route Table:

### 2. ec2\_vpc\_route\_table:

- **vpc\_id:** The VPC ID, which is retrieved from vpc\_result.vpc.id.
- **region:** The region in which the route table will be created.
- **state: present:** Ensures the creation of the route table.
- **tags:** Assigns a name to the route table, passed from pubroute\_table\_name (e.g., Baindla\_pub\_rt).
- **subnets:** Associates the created public subnet (from pubsubnet\_result.subnet.id) with this route table.
- **routes:** Defines the routing rule, where 0.0.0.0/0 (all traffic) is routed through the IGW (Internet Gateway).
- **register: public\_route\_table:** Stores the result of the route table creation in public\_route\_table.

## Create Private Route Table:

### 3. ec2\_vpc\_route\_table (for private subnet):

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

```

File Edit Selection View Go Run Terminal Help ⏪ ⏫ Search ⓘ
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
Ansible_Playbook script with outline variables ● | Ansible_Playbook script with inline variables ●
C: > Users > Spandana Baindla > Downloads > ! Ansible_Playbook script with inline variables
49 ##### Internet gateway Creation #####
61 ## create an vpc pub subnet
62 -
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95 ## create an vpc pub route table
96
97
98 - ec2_vpc_route_table:
99   aws_access_key: "{{ aws_access_key }}"
100  aws_secret_key: "{{ aws_secret_key }}"
101  vpc_id: "{{ vpc_result.vpc.id }}"
102  region: "{{ region }}"
103  state: present
104  tags:
105    Name: "{{ pubroute_table_name }}"
106    subnets: [ "{{ pubsubnet_result.subnet.id }}" ]
107    # create routes
108    routes:
109      - dest: 0.0.0.0/0
110        gateway_id: "{{ igw_result.gateway_id }}"
111    register: public_route_table
112
113 ## create an vpc pvt route table
114
115
116 - ec2_vpc_route_table:
117   aws_access_key: "{{ aws_access_key }}"
118   aws_secret_key: "{{ aws_secret_key }}"
119   vpc_id: "{{ vpc_result.vpc.id }}"
120   region: "{{ region }}"
121   state: present
122   tags:
123     Name: "{{ pvtroute_table_name }}"
124     subnets: [ "{{ pvtsubnet_result.subnet.id }}" ]
125   register: private_route_table
126
127 ## create an vpc security groups
128
129

```

Ln 47, Col 27 Spaces:2 UTF-8 CRLF YAML

## Create Security Group:

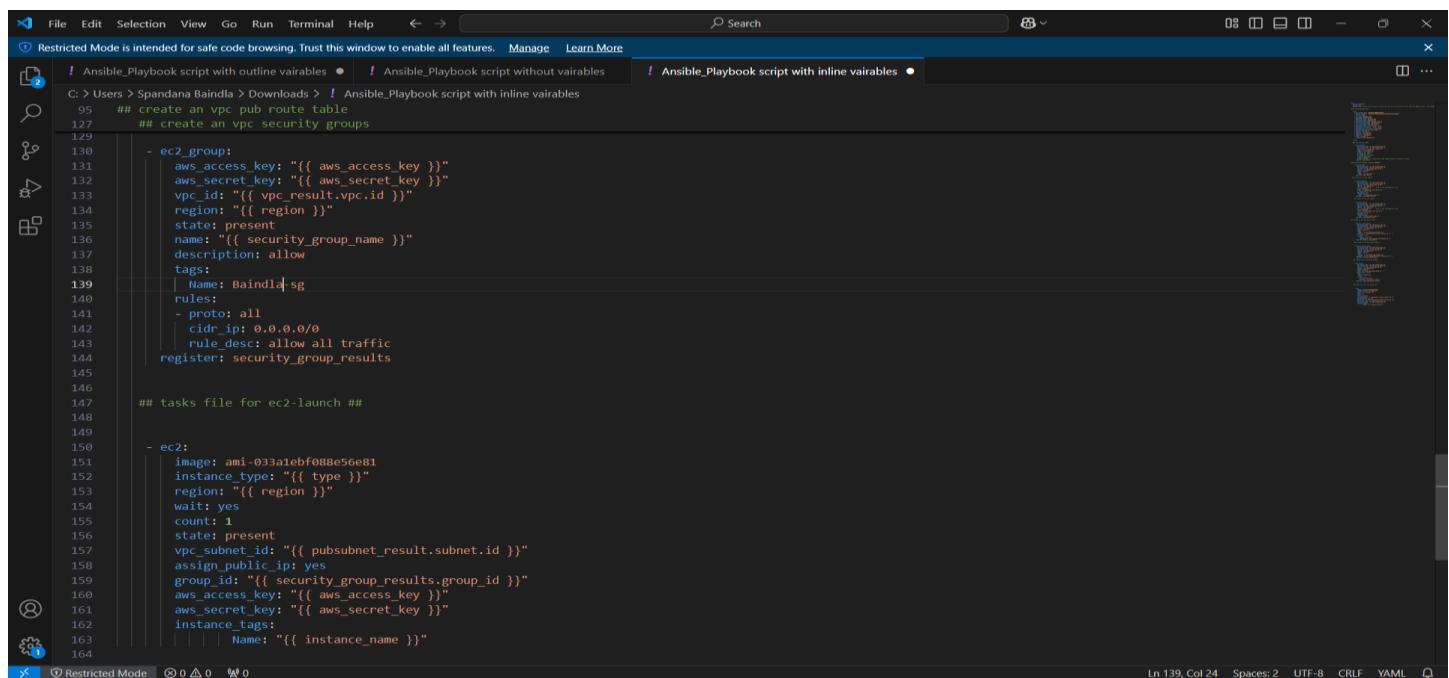
### 4. ec2\_group:

- **vpc\_id:** The ID of the VPC, fetched from vpc\_result.vpc.id.
- **region:** The AWS region where the security group is created.
- **state: present:** Ensures the security group is created.
- **name:** The name of the security group, passed from the security\_group\_name variable (e.g., Baindla\_sg).
- **description:** A description of the security group, set as allow.
- **rules:** Defines the security rules. The rule here allows all inbound traffic (proto: all and cidr\_ip: 0.0.0.0/0).
- **register: security\_group\_results:** Registers the result of security group creation in security\_group\_results.

## Launch EC2 Instance:

### 5. ec2:

- **image:** The AMI ID for the EC2 instance (e.g., ami-033a1ebf088e56e81).
- **instance\_type:** The instance type, passed from the type variable (e.g., t2.micro).
- **region:** Specifies the AWS region where the EC2 instance is created.
- **wait: yes:** Ensures the playbook waits for the instance to be in the "running" state before moving on.
- **count: 1:** Launches a single EC2 instance.
- **state: present:** Ensures the EC2 instance is created.
- **vpc\_subnet\_id:** Associates the EC2 instance with the public subnet (from pubsubnet\_result.subnet.id).
- **assign\_public\_ip: yes:** Assigns a public IP address to the EC2 instance, making it accessible from the internet.
- **group\_id:** Associates the instance with the security group created earlier (security\_group\_results.group\_id).
- **instance\_tags:** Tags the EC2 instance with the name from instance\_name (e.g., Baindla-ec).



```
C:\ Users > Spandana Baindla > Downloads > ! Ansible_Playbook script with inline variables
95 ## create an vpc pub route table
96
97 ## create an vpc security groups
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
```

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

```
[ec2-user@ip-172-31-84-59 ~]$ ls
c-ea25c75f62f8", "http_status_code": 400, "request_id": "c3f57d79-105c-4cb0-a0ec-ea25c75f62f8", "retry_attempts": 0})
PLAY RECAP ****
127.0.0.1 : ok=2    changed=0    unreachable=0   failed=1    skipped=0    rescued=0   ignored=0

[ec2-user@ip-172-31-84-59 ~]$ ll
total 8
-rw----- 1 ec2-user ec2-user 1678 Jan 10 03:52 spandana.pem
-rw-r--r-- 1 root   root   3554 Jan 10 05:10 VPC.yaml
[ec2-user@ip-172-31-84-59 ~]$ rm -rf VPC.yaml
[ec2-user@ip-172-31-84-59 ~]$ ll
total 4
-rw----- 1 ec2-user ec2-user 1678 Jan 10 03:52 spandana.pem
[ec2-user@ip-172-31-84-59 ~]$ sudo vim Vpcinline.yaml
[ec2-user@ip-172-31-84-59 ~]$ ll
-bash: ll: command not found
[ec2-user@ip-172-31-84-59 ~]$ ll
total 12
-rw----- 1 ec2-user ec2-user 1678 Jan 10 03:52 spandana.pem
-rw-r--r-- 1 root   root   4702 Jan 10 05:20 Vpcinline.yaml
[ec2-user@ip-172-31-84-59 ~]$ ansible-playbook Vpcinline.yaml

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

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

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

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

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

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

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

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

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

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

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

The screenshot shows the AWS CloudWatch Metrics Insights interface. A log stream for an EC2 instance is displayed, showing logs related to the creation of a new instance named 'Baindia-ec'. The logs indicate the instance is now in a 'Running' state with an 'Initializing' status check. The interface includes a sidebar with navigation links for EC2 Global View, Instances, Images, Elastic Block Store, Network & Security, and Load Balancing. The main pane displays detailed information about the instance, including its ID, subnet ID, and various configuration settings like IAM Role, IMDSv2, and Platform details.

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

The screenshot shows the AWS EC2 Security Groups console. On the left, there's a navigation sidebar with sections like Instances, Images, Elastic Block Store, Network & Security, Load Balancing, and CloudShell. The main area displays a table titled "Security Groups (1/5) Info" with columns for Name, Security group ID, Security group name, VPC ID, Description, and Owner. A row for "Bainsla\_sg" is selected, showing its details: sg-08da4ce43563f0409, Bainsla\_sg, vpc-0fec8c1b1941a1da7, allow, and owner 637423550105. Below this, a detailed view for "sg-08da4ce43563f0409 - Bainsla\_sg" is shown with tabs for Details, Inbound rules, Outbound rules, Sharing - new, VPC associations - new, and Tags. The "Details" tab shows fields like Security group name (Bainsla\_sg), Security group ID (sg-08da4ce43563f0409), Owner (637423550105), Inbound rules count (1 Permission entry), Description (allow), and VPC ID (vpc-0fec8c1b1941a1da7).

Successfully new VPC has been created.

The screenshot shows the AWS VPC console. The left sidebar includes sections for VPC dashboard, 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, and PrivateLink and Lattice). The main area displays a table titled "Your VPCs (1/2) Info" with columns for Name, VPC ID, State, Block Public..., IPv4 CIDR, IPv6 CIDR, DHCP option set, and Main route table. A row for "Bainsla\_vpc" is selected, showing its details: vpc-0fec8c1b1941a1da7, Available, Off, 11.0.0.0/16, -, dopt-08010d2dc2ef4a5ce, rtb-0b56c9e72ed. Below this, a detailed view for "vpc-0fec8c1b1941a1da7 / Bainsla\_vpc" is shown with tabs for Details, Resource map, CIDs, Flow logs, Tags, and Integrations. The "Details" tab shows fields like VPC ID (vpc-0fec8c1b1941a1da7), State (Available), DNS resolution (Enabled), Main network ACL (acl-0811f4a03bf9448bf), IPv6 CIDR (Network border group) (-), State (Available), Tenancy (default), Default VPC (No), Network Address Usage metrics (Disabled), Block Public Access (Off), DHCP option set (dopt-08010d2dc2ef4a5ce), IPv4 CIDR (11.0.0.0/16), and Route 53 Resolver DNS Firewall rule groups (-). Other sections include DNS hostnames (Enabled), Main route table (rtb-0b56c9e72ed), IPv6 pool (-), and Owner ID (637423550105).

## Successfully Public and Private subnets has been created.

The screenshot shows the AWS VPC Subnets console. On the left, there's a navigation sidebar with sections like VPC dashboard, Virtual private cloud, Security, and PrivateLink and Lattice. The main area displays a table of subnets with columns for Name, Subnet ID, State, VPC, Block Public, IPv4 CIDR, IPv6 CIDR, and IPv6 C. Two subnets are selected: 'Baindla\_pub' and 'Baindla\_pvt'. The table shows the following data:

Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR	IPv6 CIDR	IPv6 C
-	subnet-04313c5575f48c2b	Available	vpc-014e267882d4c37b4	Off	172.31.0.0/20	-	-
-	subnet-03d26f5ac0060abff	Available	vpc-014e267882d4c37b4	Off	172.31.80.0/20	-	-
-	subnet-0c85ac199c0740b9f	Available	vpc-014e267882d4c37b4	Off	172.31.16.0/20	-	-
-	subnet-05b7cf61a1fe49e8e	Available	vpc-014e267882d4c37b4	Off	172.31.32.0/20	-	-
-	subnet-0ca60992779b0d64	Available	vpc-014e267882d4c37b4	Off	172.31.64.0/20	-	-
<input checked="" type="checkbox"/> Baindla_pub	subnet-01ae9139ed31be5a8	Available	vpc-0fec8c1b1941a1da7   Baindla_vpc	Off	11.0.1.0/24	-	-
<input checked="" type="checkbox"/> Baindla_pvt	subnet-03b9ce7ca872a1574	Available	vpc-014e267882d4c37b4	Off	172.31.48.0/20	-	-

Below the table, it says "Subnets: subnet-01ae9139ed31be5a8, subnet-03b9ce7ca872a1574".

New public and private route table has been created successfully.

The screenshot shows the AWS RouteTables console. On the left, there's a navigation sidebar with sections like VPC dashboard, Virtual private cloud, Security, and PrivateLink and Lattice. The main area displays a table of route tables with columns for Name, Route table ID, Explicit subnet associa..., Edge associations, Main, VPC, and Owner ID. Two route tables are selected: 'Baindla\_pub\_rt' and 'Baindla\_pvt\_rt'. The table shows the following data:

Name	Route table ID	Explicit subnet associa...	Edge associations	Main	VPC	Owner ID
<input checked="" type="checkbox"/> Baindla_pub_rt	rtb-054267a02b9280d48	subnet-01ae9139ed31be...	-	No	vpc-0fec8c1b1941a1da7   Baindla_vpc	637423550105
<input checked="" type="checkbox"/> Baindla_pvt_rt	rtb-0e0c44400265d2343	subnet-03b9ce7ca872a1...	-	No	vpc-0fec8c1b1941a1da7   Baindla_vpc	637423550105
<input type="checkbox"/> -	rtb-0b56c9e72ed67543	-	-	Yes	vpc-0fec8c1b1941a1da7   Baindla_vpc	637423550105
<input type="checkbox"/> -	rtb-02a08c64822c08da8	-	-	Yes	vpc-014e267882d4c37b4	637423550105

Below the table, it says "Route tables: rtb-034267a02b9280d48, rtb-0e0c44400265d2343".

New internet gateway has been created successfully.

The screenshot shows the AWS VPC console interface. On the left, there's a navigation sidebar with sections like 'VPC dashboard', 'Virtual private cloud', 'Security', 'PrivateLink and Lattice', and 'CloudShell'. The main area displays a table titled 'Internet gateways (1/2) Info' with one item listed: 'Baindla\_igw' (ID: igw-0830cf3255e53c5cc), which is attached to two VPCs (vpc-0fec8c1b1941a1da7 and vpc-014e267882d4c37b4). Below the table, a detailed view for 'igw-0830cf3255e53c5cc / Baindla\_igw' is shown, including its details (Name: igw-0830cf3255e53c5cc, State: Attached), VPC ID (vpc-0fec8c1b1941a1da7 / Baindla\_vpc), and Owner (637423550105).

New network ACLs has been created successfully.

The screenshot shows the AWS VPC console interface. The left sidebar includes sections for 'VPC dashboard', 'Virtual private cloud', 'Security', 'Network ACLs', and 'PrivateLink and Lattice'. The main area shows a table titled 'Network ACLs (1/2) Info' with two entries: 'acl-0cc99e51760a4e66' (Associated with 6 Subnets) and 'acl-0811f4a03bf9448bf' (Associated with 2 Subnets). Both are marked as Default and belong to the VPC 'vpc-0fec8c1b1941a1da7 / Baindla\_vpc'. Below the table, a detailed view for 'acl-0811f4a03bf9448bf' is displayed, showing its details (Name: acl-0811f4a03bf9448bf, Associated with 2 Subnets, Owner: 637423550105), Default status, and VPC ID (vpc-0fec8c1b1941a1da7 / Baindla\_vpc).

The below image is the Route Map for the Vpc.

The screenshot shows the AWS VPC Resource Map for the 'Baindla\_vpc' VPC. The VPC has two subnets: 'us-east-1a' and 'us-east-1b'. It contains three route tables: 'Baindla\_pub\_rt', 'Baindla\_pvt\_rt', and 'rtb-0b56c9e72ed7543'. A single internet gateway, 'Baindla\_igw', is connected to all three route tables. The VPC has a CIDR range of 172.31.0.0/16.

Name	VPC ID	State	Block Public...	IPv4 CIDR	IPv6 CIDR	DHCP option set	Main route table
Baindla_vpc	vpc-0fec8c1b1941a1da7	Available	Off	11.0.0.0/16	-	dopt-08010d2dc2ef4a5ce	rtb-0b56c9e72ed7543
-	vpc-014e267882d4c37b4	Available	Off	172.31.0.0/16	-	dopt-08010d2dc2ef4a5ce	rtb-02a08c64822

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

### Important Notes:

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