CREATING AWS TRANSIST GATEWAY USING TERRAFORM DOCUMENT

Prerequisites:

- AWS Account
- Terraform Installed
- Visual Studio code Editor in local PC(VS)

STEPS:

Create Terraform files using VS:

- Create “Provider.tf” file

  It Specifies the AWS provider with the region set to "ap-southeast-2".

- Create “web-app-vpc.tf” file

1. Resource “AWS_vpc”

   Creates a new VPC with a CIDR block of `10.0.0.0/16`, which provides a range of private IP addresses for your resources (e.g., EC2 instances). This is the primary network container where all other resources are deployed.

2. Resource “AWS_subnet”

   Defines a subnet within the VPC, allowing resources to be placed in a smaller segment of the VPC's IP address range. The subnet is located in a specific availability zone (`ap-southeast-2a`), providing redundancy and isolation within the VPC.

3. Resource “AWS_internet_gateway”

   Creates an Internet Gateway, which allows resources in the VPC to connect to the internet. This is essential for public-facing applications or for downloading software updates and other internet-based services

4. Resource “AWS_route” "web_app_route"

   Adds a route to the VPC's default route table that directs all outbound traffic (`0.0.0.0/0`, which represents any IP address) to the Internet Gateway. This is necessary for instances in the subnet to access the internet.

5. Resource "AWS_route"

These routes enable communication between the VPC (where your web application is hosted) and other VPCs (presumably where backend services and shared databases are hosted). This is done via a Transit Gateway, which allows traffic to route between VPCs efficiently.

- Create backend-services-vpc.tf

1. Internet Gateway Route (backend_services_route):

In the route `aws_route "backend_services_route"`, you're using a placeholder comment (`# Replace with actual internet gateway ID`) but already referenced the `aws_internet_gateway.BACKEND_SERVICES_IGW.id`. So, the comment can be removed if it's intentional to use the newly created gateway.

2. Transit Gateway Routes:

You are referencing a transit_gateway_id in the routes backend_services_to_web_app and backend_services_to_shared_database, which depends on a VPC attachment resource (aws_ec2_transit_gateway_vpc_attachment.backend_services_attachment). However, the attachment itself is not defined in the provided code. You might want to add the resource for the VPC attachment if it isn't defined elsewhere.

3. Multiple Route Configurations for the Same Route Table:

You're modifying the same route table (aws_default_route_table.BACKEND_SERVICES_ROUTE) in multiple routes. Ensure that this is the intended design, especially if the backend VPC needs routes to both web application and shared database VPCs.

4. CIDR Block Adjustments:

Make sure the CIDR blocks for the different VPCs (backend, web app, shared database) are correctly configured and don't overlap to avoid network conflicts.

- Create "shared-database-vpc.tf" file

1. Type in Subnet Tag:

In the aws_subnet resource for SHARED_DATABASE_SUBNET, the tag key is set to Name = "SHARED_DAYABASE_SUBNET". It looks like there's a typo in SHARED_DAYABASE. It should be SHARED_DATABASE_SUBNET.

2. Transit Gateway Attachments:

   Similar to the backend services VPC, you reference a transit_gateway_id and a VPC attachment resource (aws_ec2_transit_gateway_vpc_attachment.shared_database_attachment), but that resource definition is missing. You'll need to add a resource block for the aws_ec2_transit_gateway_vpc_attachment that attaches the shared database VPC to the transit gateway.

3. Multiple Routes to the Same Route Table:

   As you're configuring the route table aws_default_route_table.SHARED_DATABASE_ROUTE to route traffic to both backend services and web app VPCs via a transit gateway, ensure that your transit gateway route tables are configured properly to support these routes.

4. Availability Zone:

   You specified availability_zone = "ap-southeast-2a" in your subnets. Ensure this matches your actual infrastructure's availability zones in the AWS region you're operating in.

5. Internet Gateway Route:

   Similar to the backend VPC, in the aws_route for shared_database_route, the comment for replacing the internet gateway ID can be removed, as the ID is already referenced with aws_internet_gateway.SHARED_DATABASE_IGW.id.

- Create "ec2-web-app.tf" file

  1. Security Group (aws_default_security_group):

     This is the firewall for your EC2 instance.
     Allows SSH (port 22) from anywhere (so you can log in).
     Allows HTTP (port 80) from anywhere (so the web server is accessible).
     Allows all outbound traffic (so the instance can make outgoing connections).

  2. EC2 Instance (aws_instance):

     Launches a virtual machine (EC2) using an **Ubuntu AMI**.
     The instance type is t2.micro, which is a small and free-tier eligible machine.
     The **key pair** (webapp_keypair) lets you securely log in via SSH.
     A **user data script** automatically runs when the instance starts, doing the following:

     1. Updates Ubuntu.
     2. Installs Nginx (a web server).
     3. Starts and enables Nginx on boot.
     4. Sets up a custom web page that displays the instance's hostname and IP address.

     Tags the instance as "Webapp-Linux-Instance".
     The instance is attached to a **security group** and a **subnet** (for network configuration).
     Automatically gets a **public IP** so it's accessible over the internet.

3. Output Block:

After the EC2 instance is created, this outputs the instance's public IP address so you can access it.

• Create "ec2-backend-services.tf" file

1. Security Group (aws_default_security_group):

This sets up a **security group** (a virtual firewall) for the **Backend Services VPC**.
SSH Access: Allows SSH traffic (port 22) from anywhere (0.0.0.0/0), though it's better to restrict this to your own IP.
HTTP Access: Allows HTTP traffic (port 80) from anywhere. This is useful for web server access.
Outbound Traffic: All outbound traffic is allowed, meaning the instance can connect to any IP address or port.
The security group is tagged as "BackendServices-sg" for easy identification.

2. EC2 Instance (aws_instance):

This creates an EC2 instance (a virtual machine) in the **Backend Services VPC**.

AMI: Uses an Ubuntu image (ami-007020fd9c84e18c7).

Instance Type: Uses a t2.micro instance, which is small and cost-effective.

Key Pair: The instance uses "webapp_keypair" for SSH access.

User Data Script: This script automatically runs when the instance starts:

a. Updates the package list.
b. Installs and starts the **Nginx** web server.
c. Configures an index.html page that displays the instance's hostname and IP address.

The instance is tagged as "Backend-Services-Linux-Instance" for easy identification.

Security Group: The instance is attached to the security group defined above.

Subnet: It's associated with the **Backend Services Subnet**.

Public IP: The instance is assigned a public IP so you can access it over the internet.

3. Output:

After the instance is launched, this outputs the **public IP address** of the EC2 instance, so you can access it directly via the internet.

- Create “ec2-shared-database.tf” file

    1. Security Group (aws_default_security_group):

        This defines a **security group** (a firewall) for the **Shared Database VPC**.
        SSH Access (port 22): Allows SSH traffic from anywhere (0.0.0.0/0), but it’s recommended to restrict this to your specific IP for better security.
        HTTP Access (port 80): Allows HTTP traffic from anywhere, so you can access the web server.
        Outbound Traffic: All outbound traffic is allowed, meaning the instance can connect to external resources.
        The security group is tagged as "Shareddatabase-sg" for identification.

    2. EC2 Instance (aws_instance):

        This creates an EC2 instance in the **Shared Database VPC**.
        AMI: Uses an Ubuntu image (ami-007020fd9c84e18c7).
        Instance Type: The instance type is t2.micro, which is small and cost-effective.
        Key Pair: The instance uses "webapp_keypair" for SSH access.
        User Data Script: A script runs when the instance is launched:

        1. Updates the package list.
        2. Installs **Nginx**, a web server.
        3. Starts and enables Nginx to run on startup.
        4. Creates a custom index.html file that displays the instance’s hostname and IP address.

        Tags the instance as "SharedDatabase-Linux-Instance".
        The instance is attached to the **Shared Database Subnet** and the security group created earlier.
        Public IP: The instance is assigned a public IP for internet access.

    3. Output Block:

        Outputs the **public IP address** of the EC2 instance after it's created, so you can access it directly.

    Note: Change the AMI ID’s (ubuntu) and Key’s

- Create “transist_gateway.tf” file

    1. Resource Block (aws_ec2_transit_gateway):

        Resource Type: Defines an AWS EC2 Transit Gateway.
        Description: Provides a description of the Transit Gateway. In this case, it's named "tg-web-backend-database" to indicate its purpose of connecting the **Web**, **Backend**, and **Database** VPCs.

Tags:
The Transit Gateway is tagged as "Web-Backend-Database Transit Gateway", making it easier to identify within the AWS environment.

2. Purpose of a Transit Gateway:

   A **Transit Gateway** acts as a centralized hub to route traffic between multiple VPCs, making it easier to manage and scale your network architecture in AWS.

- Create “transist_gateway_attachment.tf” file

1. aws_ec2_transit_gateway_vpc_attachment Resources:

   These resources define VPC attachments to the **Transit Gateway** (aws_ec2_transit_gateway.example), enabling traffic routing between VPCs

2. Attachments:

   Web App VPC Attachment:

   - **Transit Gateway:** Attaches the WEB_APP_VPC to the **Transit Gateway**.
   - **Subnet:** Associates the WEB_APP_SUBNET with the Transit Gateway.
   - **VPC ID:** Refers to the WEB_APP_VPC that needs to be attached.

   Backend Services VPC Attachment:

   - Same as above but attaches the BACKEND_SERVICES_VPC and its BACKEND_SERVICES_SUBNET to the **Transit Gateway**.

   Shared Database VPC Attachment:

   - Attaches the SHARED_DATABASE_VPC and its SHARED_DATABASE_SUBNET to the **Transit Gateway**.

3. Key Points:

   Transit Gateway ID: All VPCs are attached to the same Transit Gateway (aws_ec2_transit_gateway.example.id).
   Subnet Associations: Each VPC is associated with its respective subnet.
   Tags: Each attachment is tagged for easy identification.

Configure Terraform Server:

- ➢ Connect to the server
- ➢ Install the HashiCorp GPG Key

  - ♦ sudo yum install -y yum-utils

    - sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
- To Install Terraform
    - sudo yum install terraform –y
- Verify the installation
    - terraform –version

Pull all the files which we created in the VS:

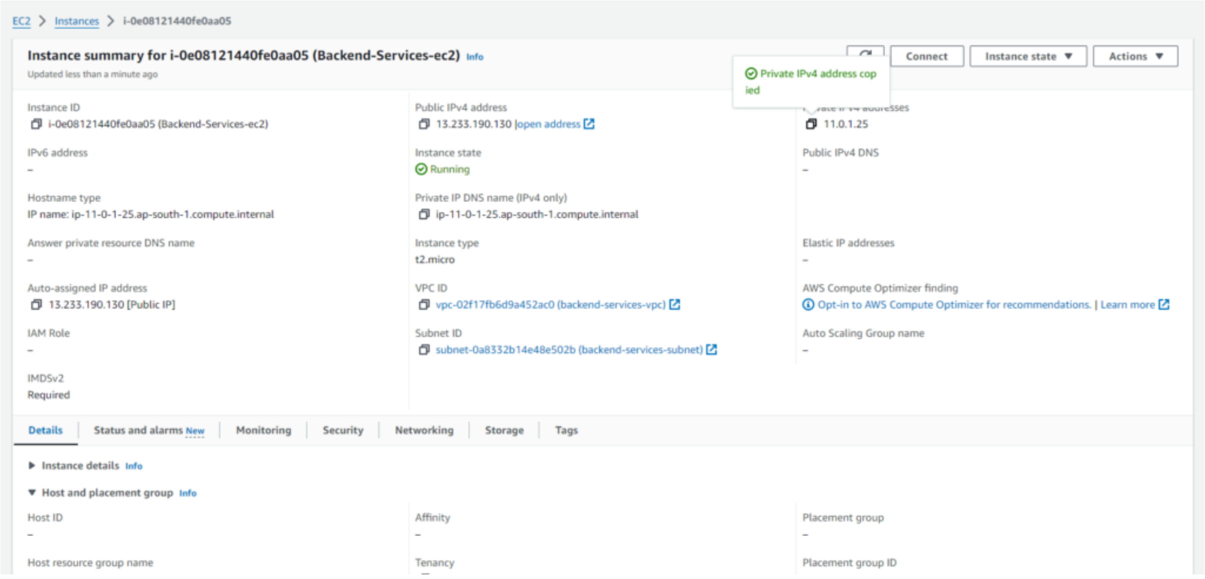
- Login to Git
- Create new repository
- Click on "upload an existing file"
- Select and upload all the files
- Give the commit message and "commit"
- Install git to the Terraform server
    - sudo yum install git –y
- In the Terraform server
    - git clone https://github.com/kakkepremchand/Createing-AWS-Transist-gateway-using-Terraform.git

Apply the configurations (in Terraform Server):

- terraform init
- terraform validate
- terraform plan
- terraform apply –auto-approve

Verification:

Check and confirm successful communication between your VPCs through the Transit Gateway



Instance summary for i-0e08121440fe0aa05 (Backend-Services-ec2) Info

Updated less than a minute ago

✓ Private IPv4 address copied

Instance ID
i-0e08121440fe0aa05 (Backend-Services-ec2)

IPv6 address
–

Hostname type
IP name: ip-11-0-1-25.ap-south-1.compute.internal

Answer private resource DNS name
–

Auto-assigned IP address
13.233.190.130 [Public IP]

IAM Role
–

IMDSv2
Required

Public IPv4 address
13.233.190.130 |open address

Instance state
✓ Running

Private IP DNS name (IPv4 only)
ip-11-0-1-25.ap-south-1.compute.internal

Instance type
t2.micro

VPC ID
vpc-02f17fb6d9a452ac0 (backend-services-vpc)

Subnet ID
subnet-0a8332b14e48e502b (backend-services-subnet)

Private IPv4 addresses
11.0.1.25

Public IPv4 DNS
–

Elastic IP addresses
–

AWS Compute Optimizer finding
ⓘ Opt-in to AWS Compute Optimizer for recommendations. | Learn more

Auto Scaling Group name
–

▶ Instance details Info

▼ Host and placement group Info

Host ID
–

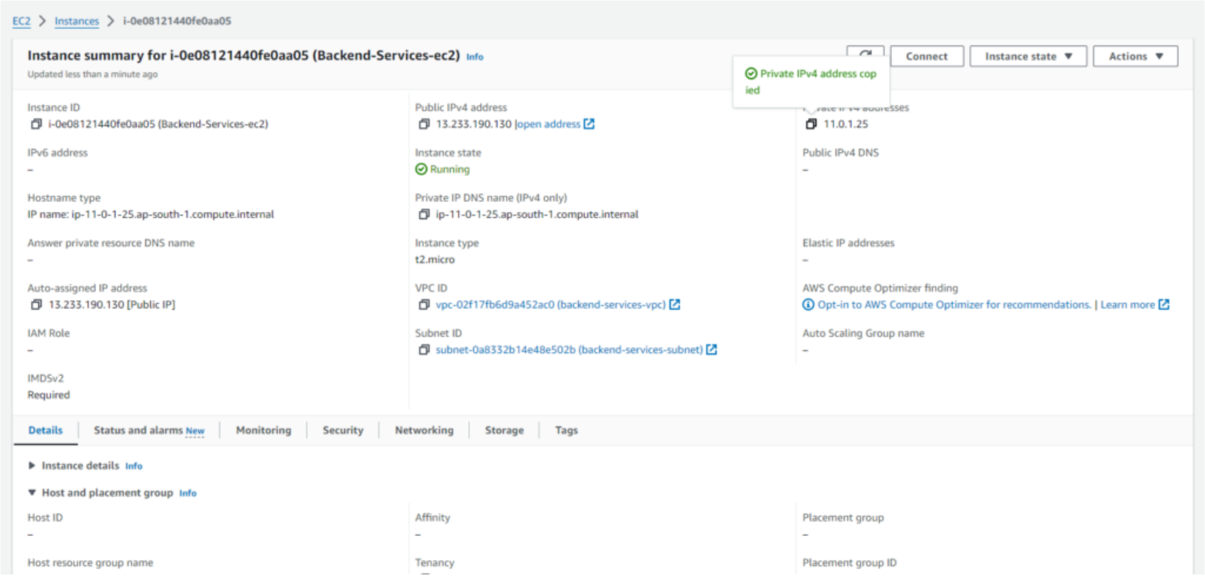
Host resource group name

Affinity
–

Tenancy

Placement group
–

Placement group ID



EC2 > Instances > i-0e08121440fe0aa05

Instance summary for i-0e08121440fe0aa05 (Backend-Services-ec2) Info

Updated less than a minute ago

✓ Private IPv4 address copied

Instance ID
i-0e08121440fe0aa05 (Backend-Services-ec2)

IPv6 address
–

Hostname type
IP name: ip-11-0-1-25.ap-south-1.compute.internal

Answer private resource DNS name
–

Auto-assigned IP address
13.233.190.130 [Public IP]

IAM Role
–

IMDSv2
Required

Public IPv4 address
13.233.190.130 |open address

Instance state
✓ Running

Private IP DNS name (IPv4 only)
ip-11-0-1-25.ap-south-1.compute.internal

Instance type
t2.micro

VPC ID
vpc-02f17fb6d9a452ac0 (backend-services-vpc)

Subnet ID
subnet-0a8332b14e48e502b (backend-services-subnet)

Private IPv4 addresses
11.0.1.25

Public IPv4 DNS
–

Elastic IP addresses
–

AWS Compute Optimizer finding
ⓘ Opt-in to AWS Compute Optimizer for recommendations. | Learn more

Auto Scaling Group name
–

▶ Instance details Info

▼ Host and placement group Info

Host ID
–

Host resource group name

Affinity
–

Tenancy

Placement group
–

Placement group ID