EC2 Instance Deployment in a Custom Public Subnet Using Terraform

Introduction :-

This project demonstrates the automated deployment of an EC2 instance within a custom public subnet using Terraform. By leveraging Infrastructure as Code (IaC) principles, the configuration is modular, reusable, and secure.

Project Overview :-

VPC and Networking Configuration:-

* Custom VPC: Established a Virtual Private Cloud (VPC) with a specified CIDR block to ensure network isolation.

* Public Subnet: Configured a subnet within the VPC, designated as public by associating it with a route table that directs traffic to the internet.

*Route Table: Set up a route table with a route to the Internet Gateway, enabling internet access for resources within the public subnet.

* Internet Gateway: Deployed and attached an Internet Gateway to the VPC to facilitate internet connectivity.

EC2 Instance Deployment:-

Instance Launch: Deployed an EC2 instance within the configured public subnet, ensuring it receives a public IP address for accessibility.

Terraform Configuration Structure :-

The Terraform scripts are organized into distinct files for clarity and maintainability:

* providers.tf: Specifies the required providers, primarily AWS in this context.

* main.tf: Contains the core resource definitions, including VPC, subnets, route tables, and EC2 instances.

* variables.tf: Declares input variables to parameterize the configuration, enhancing reusability.

* terraform.tfvars: Assigns values to the declared variables, allowing for customization per deployment.

* output.tf: Defines output values to provide essential information post-deployment, such as the EC2 instance's public IP and key name.

* backend.tf: Configures the backend for storing the Terraform state file, ensuring secure and remote state management.

This modular approach enhances script organization, simplifies modifications, and improves readability for collaborators.

Security Best Practices :-

Remote State Management: The Terraform state file, containing sensitive infrastructure data, is stored in a secure remote backend (e.g., AWS S3) to prevent unauthorized access.

Reusability and Parameterization :-

* To accommodate varying requirements, input variables are utilized for:

* EC2 Instance Type: Allows selection of the desired instance type (e.g., t2.micro).

* Key Pair Name: Specifies the SSH key pair for secure access to the instance.

* AMI ID: Defines the Amazon Machine Image ID for the instance's operating system.

Values for these variables are provided in the terraform.tfvars file, facilitating easy adjustments for different deployment scenarios.

Outputs :-

Post-deployment, the following outputs are provided:

* Public IP Address: The EC2 instance's public IP, essential for SSH access.
* Key Pair Name: The name of the SSH key pair associated with the instance.

These outputs are crucial for users to connect to the EC2 instance, especially since direct access to the state file is restricted.

Challenges and Resolutions :-

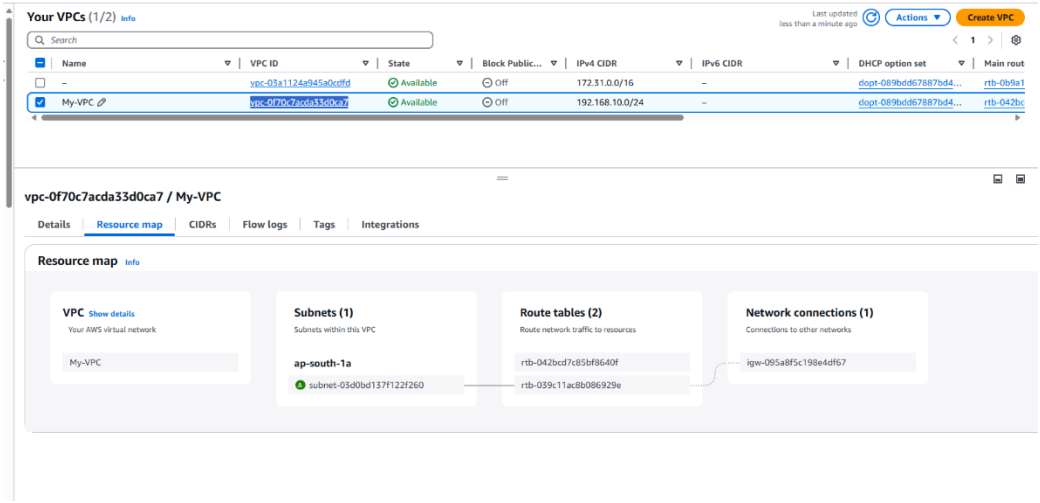
A notable challenge encountered was the EC2 instance not receiving a public IP address upon launch. This was resolved by modifying the subnet configuration to enable automatic public IP assignment, ensuring the instance is accessible as intended.

Conclusion :-

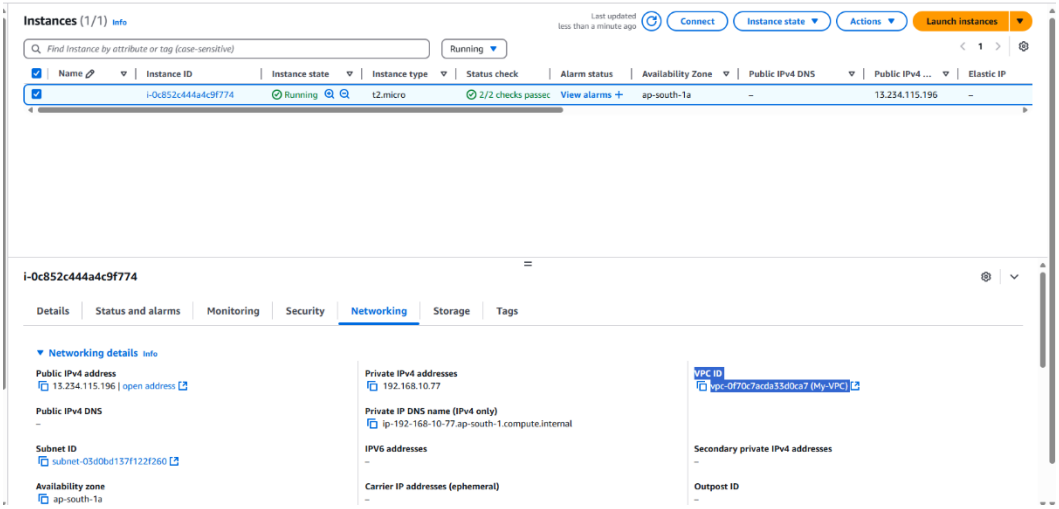
This project underscores the effectiveness of Terraform in automating AWS infrastructure deployment. By adhering to best practices in script organization, security, and parameterization, the configuration is both robust and adaptable to various use cases.

Outcomes of the Project :-

* Custom VPC created with Public Subnet



* EC2 instance launched in custom Subnet (Inside Custom VPC)



* State file stored in S3 bucket