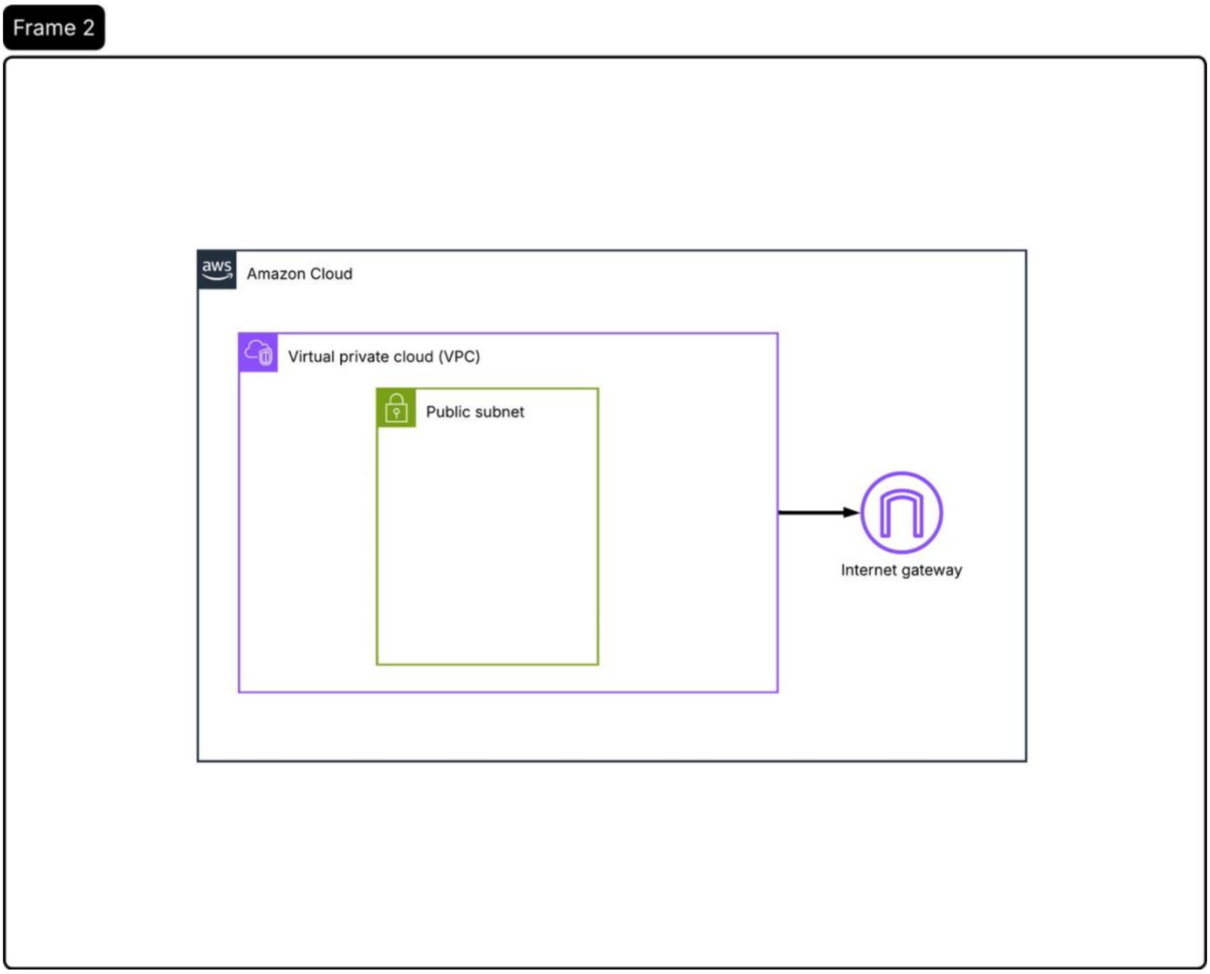
BUILD A VIRTUAL PRIVATE NETWORK



Introducing Today's Project

In this project, I focused on building a **Virtual Private Cloud (VPC)** on AWS to strengthen my understanding of cloud networking fundamentals.
The goal was to create a secure and isolated network, configure a **public subnet**, attach an **Internet Gateway**, and verify external connectivity.

To deepen my learning, I implemented the same architecture in **two ways**:

- Using the **AWS Management Console**
- Using **Terraform (Infrastructure as Code)**

This allowed me to compare manual configuration versus automated, repeatable infrastructure provisioning.

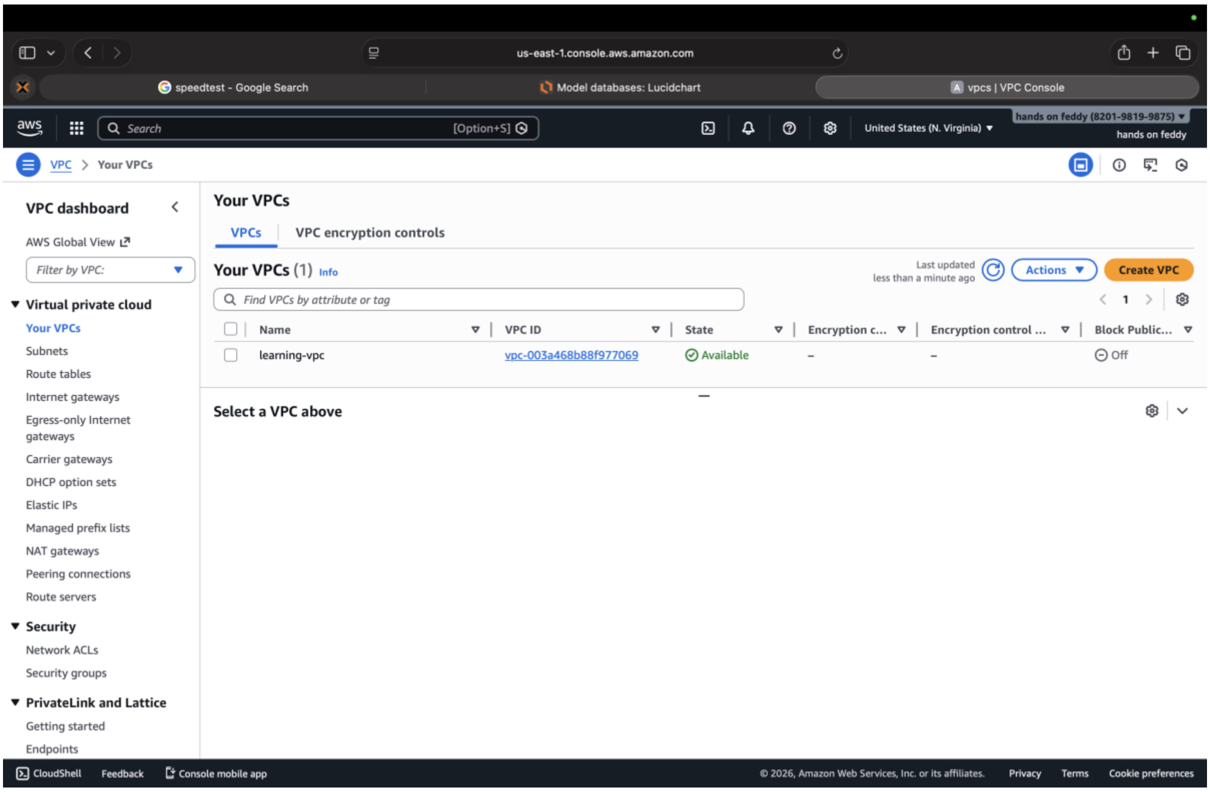
Virtual Private Clouds

What I Did in This Step

In this step, I created a custom VPC with the following configuration:

- Defined a private IPv4 CIDR block
- Enabled DNS support and DNS hostnames
- Attached an Internet Gateway to allow outbound internet access
- Created a public subnet within the VPC
- Configured routing to allow internet traffic

This setup forms the **foundation of almost every AWS architecture**, from simple web apps to complex microservices.



VPC details page (CIDR, DNS enabled)

How VPCs Work

A VPC works by isolating network traffic using:

- **CIDR blocks** to control IP address ranges
- **Subnets** to segment the network
- **Route tables** to define how traffic flows
- **Gateways** to control access to and from the internet

Resources inside a VPC cannot communicate with the internet unless routing and gateways are explicitly configured.
This design follows the **principle of least privilege**, which is critical for cloud security.

Defining IPv4 CIDR Blocks

For this project, I used the following CIDR structure:

- **VPC CIDR:** `10.0.0.0/16`
- **Public Subnet CIDR:** `10.0.1.0/24`

This approach:

- Provides enough IP space for future expansion
- Keeps subnet boundaries clean and predictable
- Avoids conflicts with common home or office networks

Understanding CIDR blocks is essential because **poor IP planning causes serious issues** when scaling, peering VPCs, or adding VPN connections.

CIDR block configuration during VPC creation

Subnets

Subnets are subdivisions of a VPC that allow you to group resources based on access requirements.

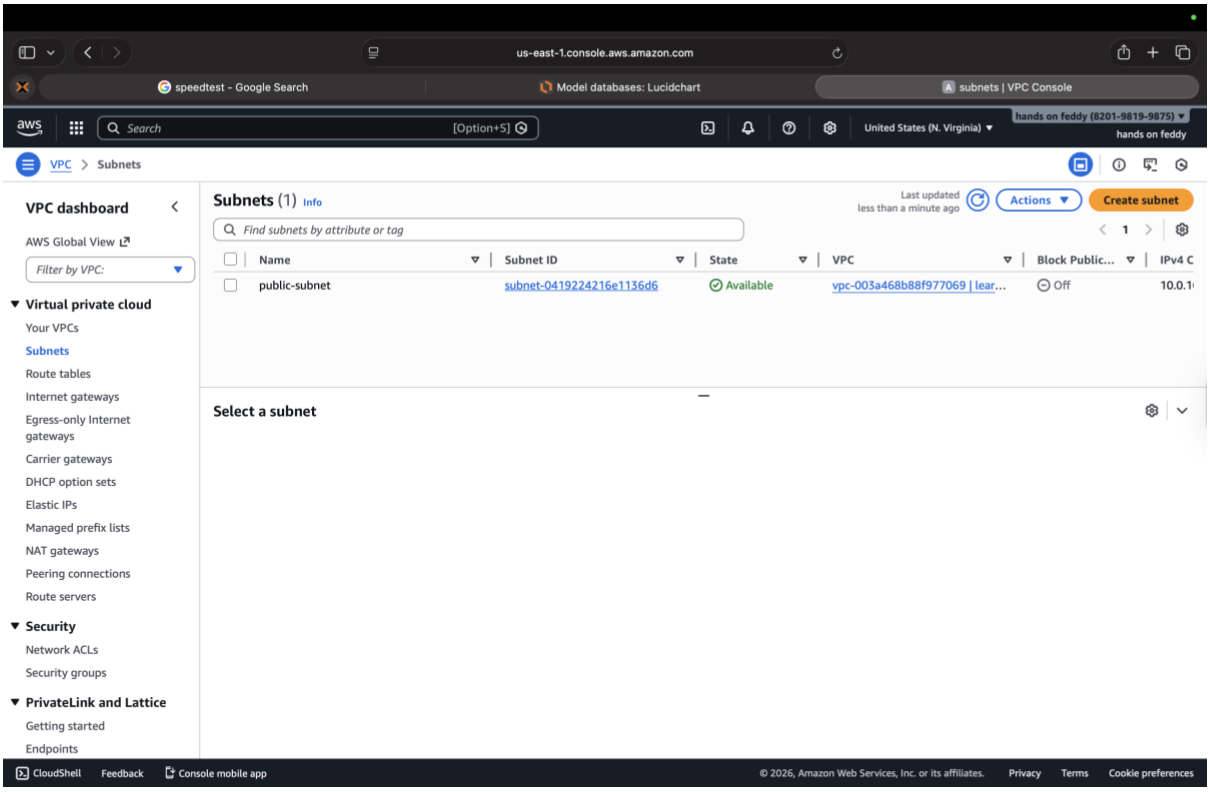
In this project, I created:

- **One public subnet** that allows resources to communicate with the internet

Key configurations included:

- Assigning the subnet to a specific Availability Zone
- Enabling automatic public IP assignment
- Associating the subnet with a route table that routes traffic to the Internet Gateway

Without proper subnet configuration, resources remain isolated even if an Internet Gateway exists.



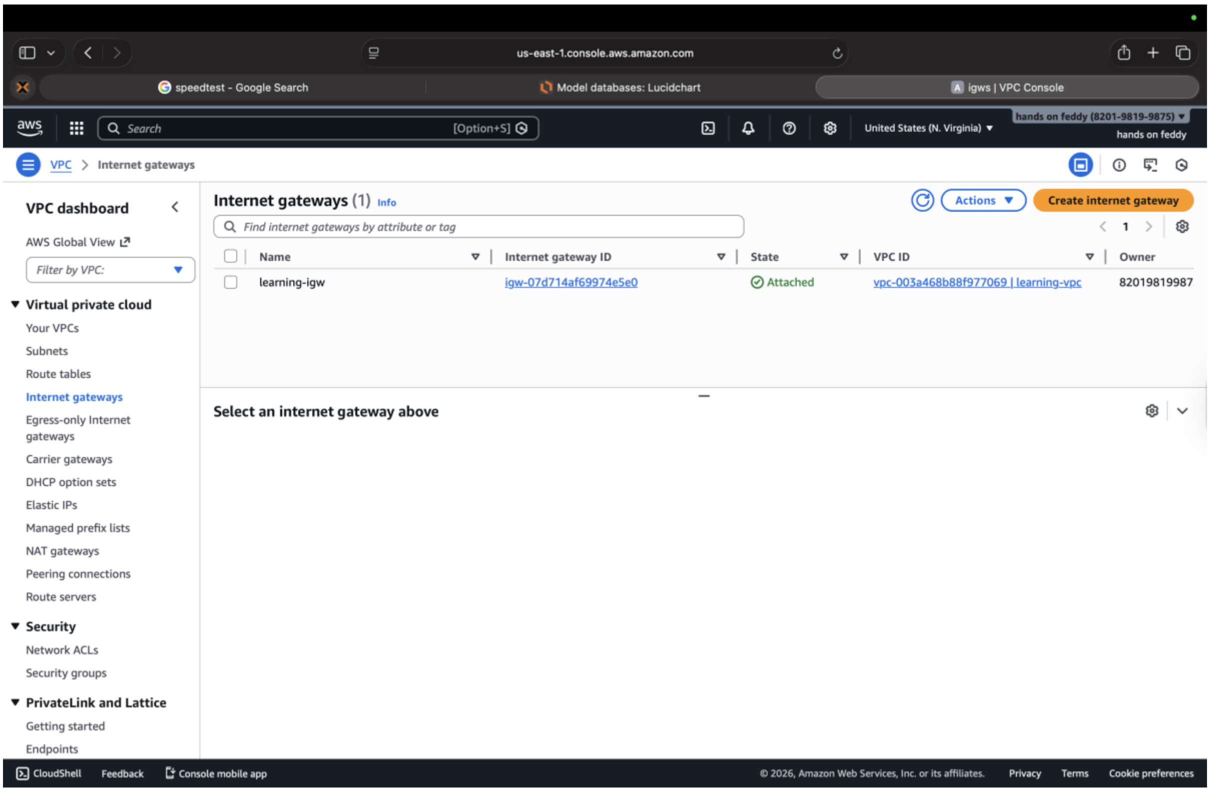
Subnet details page showing public IP auto-assign enabled

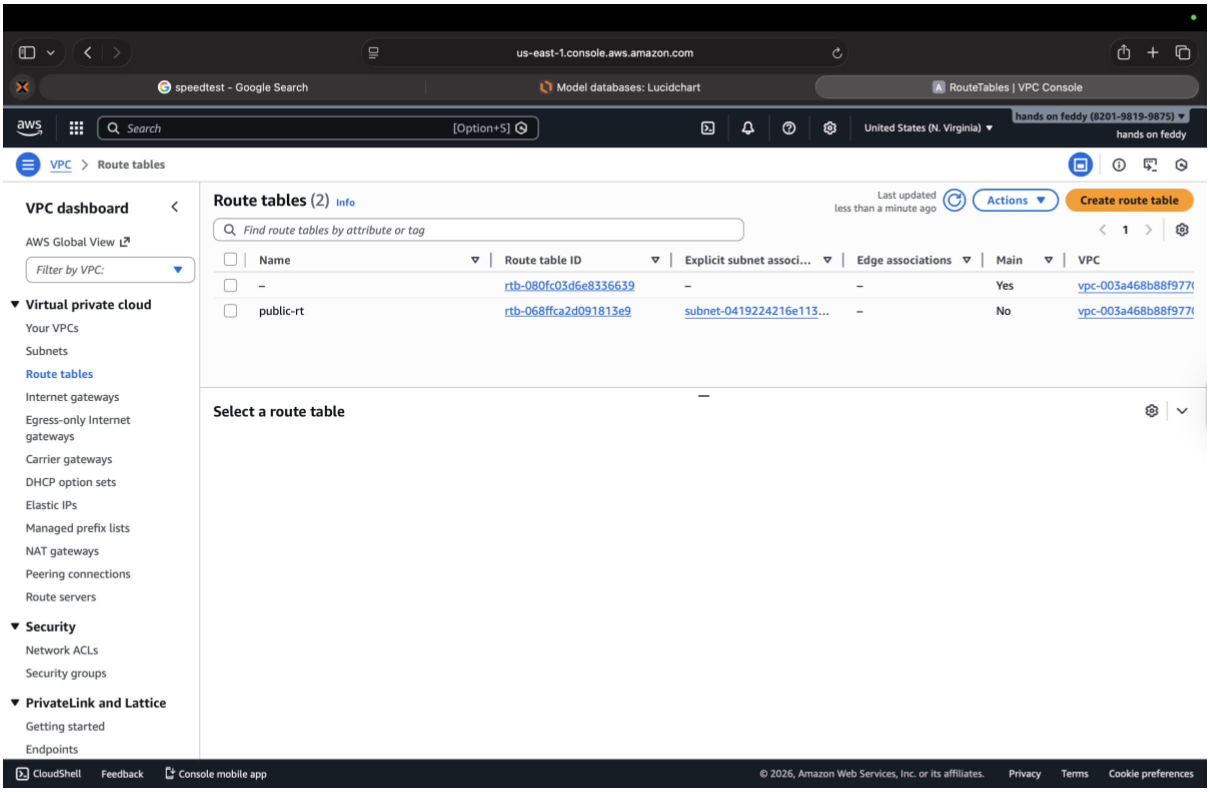
Internet Gateway and Routing

To allow external connectivity, I:

- Created an **Internet Gateway**
- Attached it to the VPC
- Updated the route table to route `0.0.0.0/0` traffic to the Internet Gateway
- Associate the route table with the public subnet

This step is critical because **subnets do not become public by default**.
They must explicitly route traffic to an Internet Gateway.





Route table showing 0.0.0.0/0 → IGW

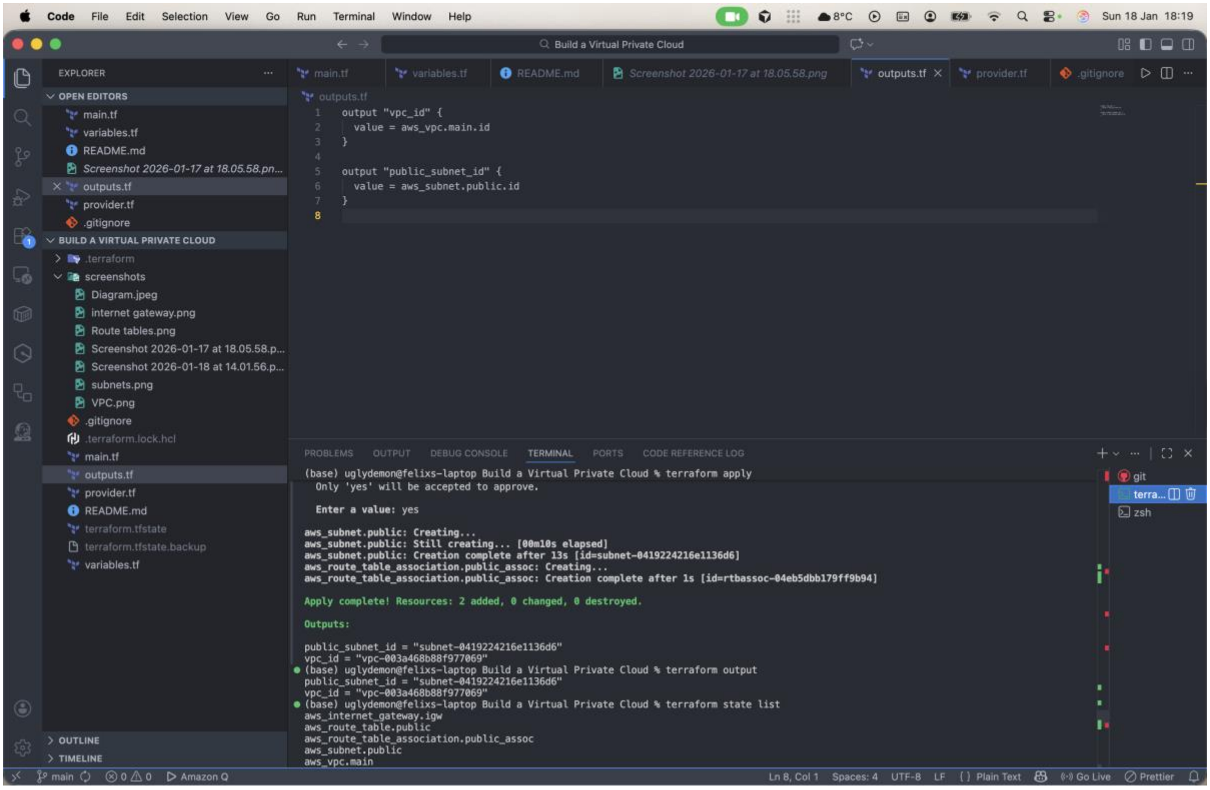
Implementing the Same Architecture with Terraform

After completing the setup via the AWS Console, I recreated the entire architecture using **Terraform**.

Using Terraform allowed me to:

- Define infrastructure declaratively
- Version-control my cloud setup
- Recreate the environment consistently
- Reduce human error

This reinforced the importance of **Infrastructure as Code (IaC)** in modern DevOps and cloud engineering workflows.



Terraform plan/apply output or repo structure

Personal Reflection

This project helped me clearly understand how AWS networking components work together rather than in isolation.
The biggest takeaway was realizing that **internet access is not automatic;** it is the result of correctly configured routing, gateways, and subnet settings.

Building the same setup using both the console and Terraform gave me confidence in:

- Reading AWS architecture diagrams
- Debugging networking issues
- Translating manual steps into code

This project is part of my ongoing journey to become a **Cloud Engineer**, and I will continue building on this foundation with EC2, security groups, and automation.