Multi-Cloud VM Deployment with Terraform (AWS & GCP)

Mohammed Amir

PROJECT OVERVIEW

This project demonstrates automated provisioning of virtual machines in both AWS and GCP using Terraform. It showcases multi-cloud infrastructure management by deploying multiple EC2 instances in AWS and Compute Engine VMs in GCP with dynamic scaling using the count parameter.

The project emphasizes best practices in Terraform, including variable usage, provider configuration, and clean resource definitions. This setup is ideal for demonstrating cloud automation, infrastructure-as-code (IaC) skills, and multi-cloud deployment capabilities.

Key Skills: AWS, GCP, Terraform (IaC), Visual Studio, Linux(WSL)

PROJECT STEPS

Step 1: Install Terraform

Terraform was installed on WSL (Linux) to enable infrastructure provisioning. For installation, refer to theofficial documentation page

Figure 1: Terraform Installation on WSL

To check if Terraform is installed and verify its version

```
terraform - -version
```

Step 2: Define Variables

Before creating providers and resources, it is a best practice to define all **variables** that you will reuse throughout your Terraform project.

What are Variables in Terraform?

- **Variables** are placeholders that store values (e.g., `AWS region`, `GCP project ID`, `machine type`, etc.).
- Instead of hardcoding values inside `main.tf`, you reference variables.
- This makes your code **cleaner, reusable, and easier to update**.

Reason for Defining Variables First

- **Centralized Configuration** – All input values are defined in one place.
- **Reusability** – The same Terraform configuration can be deployed in different environments (Dev, Test, Prod) just by changing variable values.
- **Scalability** – If you need to increase instance count or change region, you only update the variable, not every resource block.

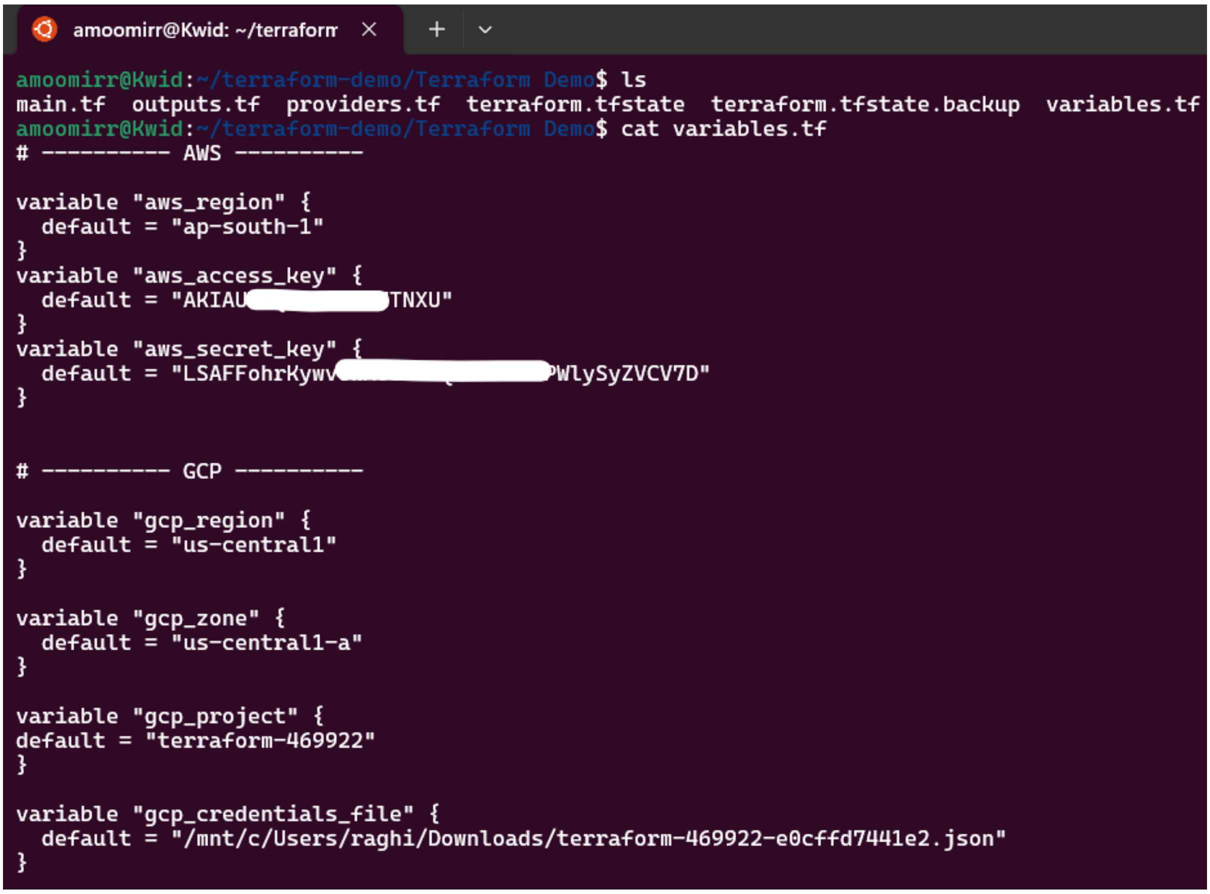

Figure 2: Variables.tf

Step 3: Configure Cloud Providers

Providers in Terraform allow it to interact with different cloud platforms like **AWS** , **Azure** and **GCP**. They act as plugins that translate Terraform configuration into API calls.

Action:

- **Define AWS Provider:** Specify the AWS region, access key, and secret key to allow Terraform to manage AWS resources.
- **Define Google Cloud Provider:** Specify the GCP project ID, region, zone, and credentials JSON file to authenticate Terraform with Google Cloud.

Reason for Configuring Providers:

- **Multi-Cloud Deployment:** Enables Terraform to manage resources across multiple platforms (e.g., AWS + GCP).
- **Authentication:** Providers supply the credentials required for Terraform to access the cloud.
- **Flexibility:** Different environments (Dev, Test, Prod) can be set up easily by switching provider configurations.
- **Scalability:** Providers allow Terraform to orchestrate infrastructure at scale across regions and accounts.

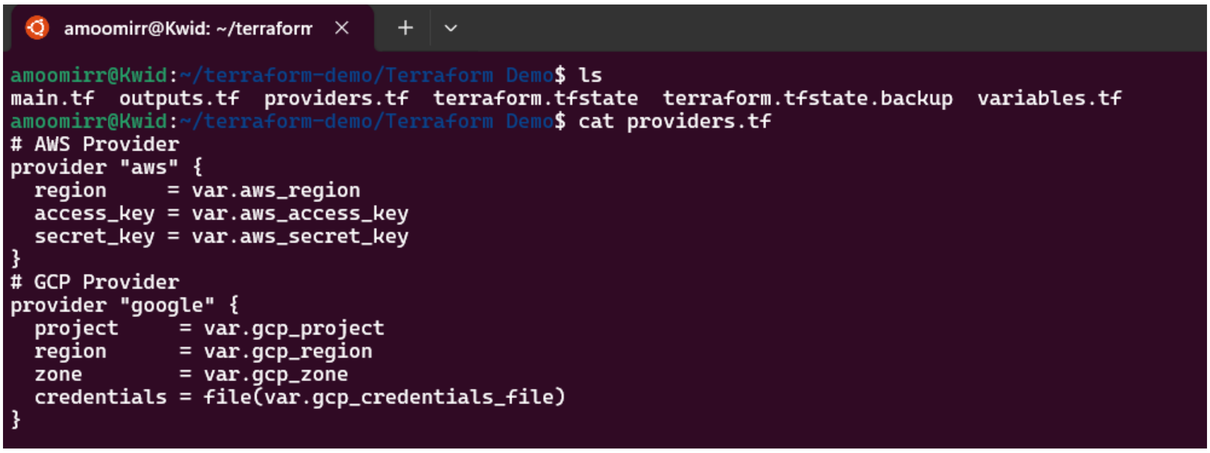

Figure 3: Provider.tf

Step 4: Resources

Resources in Terraform are the actual cloud components that will be created, managed, or destroyed. Examples include AWS EC2 instances, GCP Compute Engine instances.

Action:

- Define `aws_instance` to create EC2 instances in AWS.
- Define `google_compute_instance` to create VM instances in GCP.
- Reference variables (like machine type, project ID, AMI, region, etc.) instead of hardcoding.
- Use provider settings (already defined in Step 2) for authentication and region selection.

Reason:

- Resources are dependent on both **variables (Step 1)** and **providers (Step 2)**.
- Variables ensure reusability and flexibility, while providers allow Terraform to interact with AWS or GCP.
- Defining resources after variables and providers ensures proper execution order in the Terraform workflow.

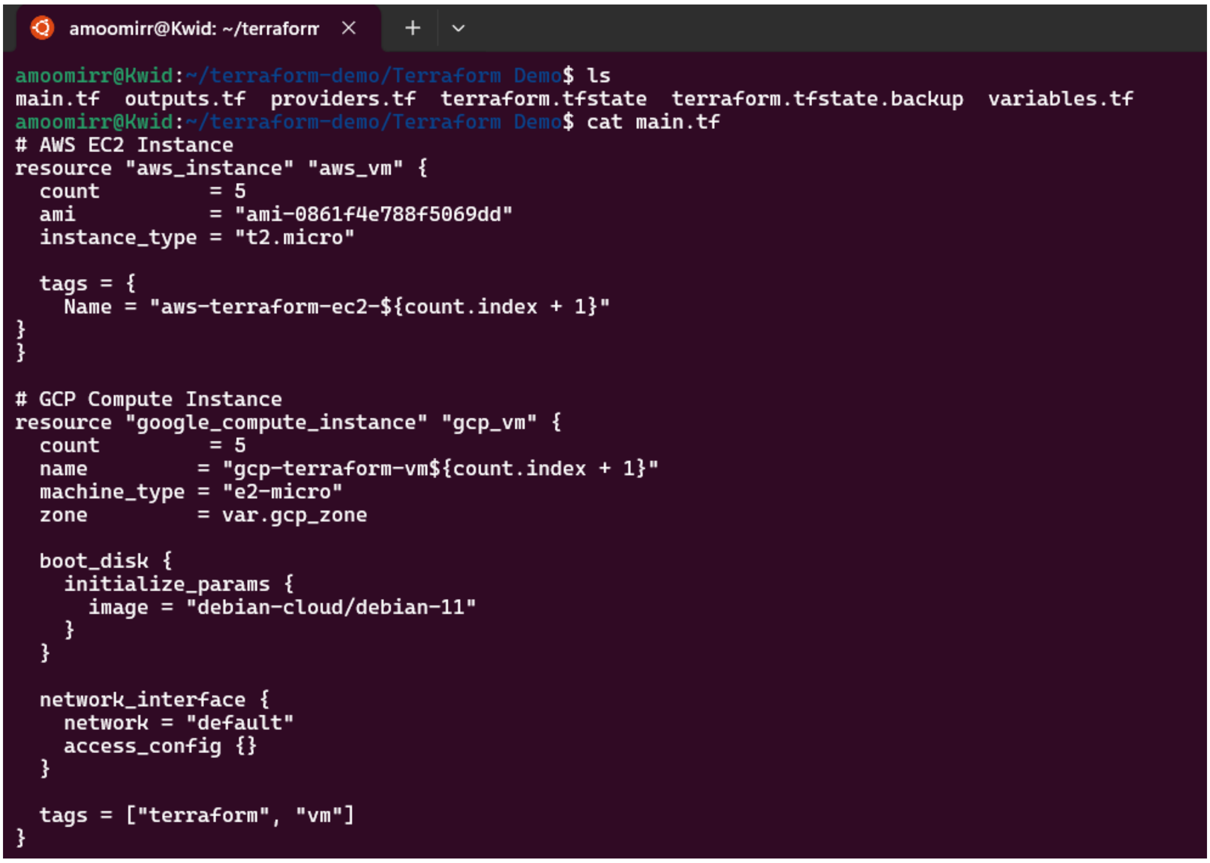

Figure 4: Main.tf

Step 5: Initialize Terraform and Apply Configuration

Terraform was initialized, plan was checked, and configuration applied.

- **terraform init**: Initializes the Terraform working directory and downloads required provider plugins.

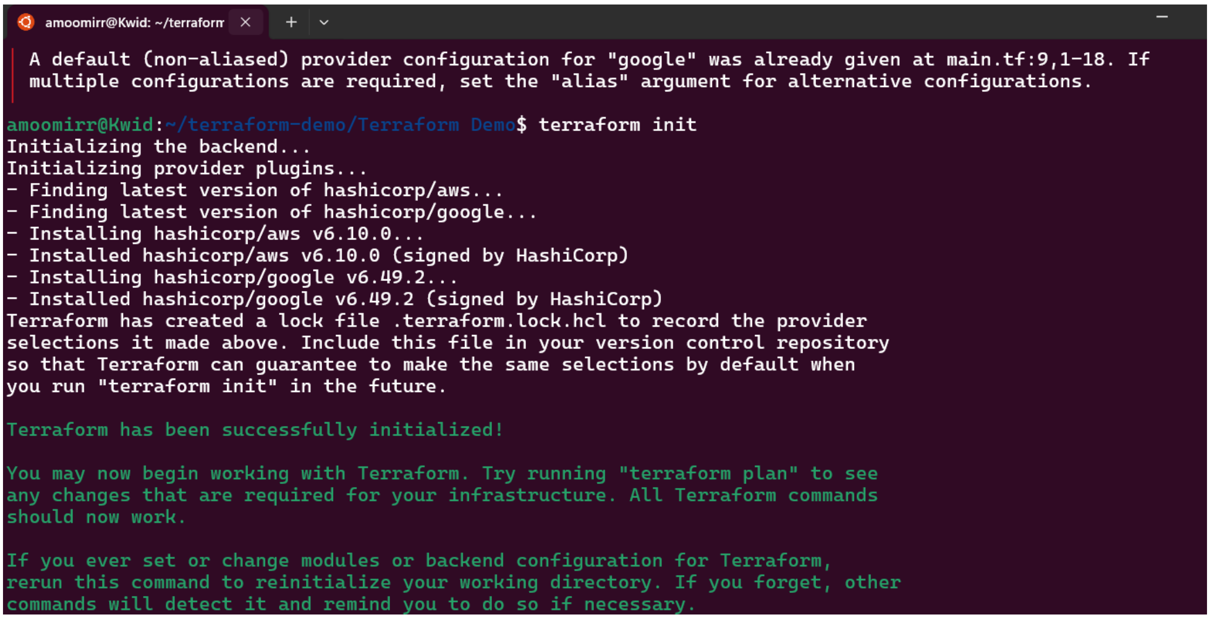

Figure 5: Terraform Init

- **terraform plan**: Generates an execution plan showing what actions Terraform will perform.

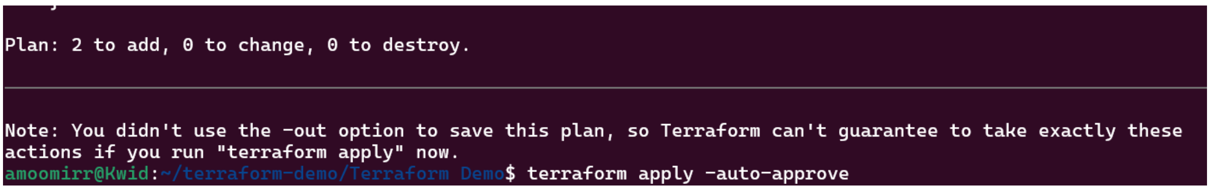

Figure 6: Terraform Plan

- **terraform apply**: Applies the changes required to reach the desired state of the configuration.

```
google_compute_instance.gcp_vm[4]: Creating...
google_compute_instance.gcp_vm[0]: Creating...
google_compute_instance.gcp_vm[3]: Creating...
google_compute_instance.gcp_vm[2]: Creating...
google_compute_instance.gcp_vm[1]: Creating...
aws_instance.aws_vm[1]: Still creating... [00m10s elapsed]
aws_instance.aws_vm[4]: Still creating... [00m10s elapsed]
aws_instance.aws_vm[2]: Still creating... [00m10s elapsed]
aws_instance.aws_vm[0]: Still creating... [00m10s elapsed]
aws_instance.aws_vm[3]: Still creating... [00m10s elapsed]
google_compute_instance.gcp_vm[0]: Still creating... [00m10s elapsed]
google_compute_instance.gcp_vm[4]: Still creating... [00m10s elapsed]
google_compute_instance.gcp_vm[3]: Still creating... [00m10s elapsed]
google_compute_instance.gcp_vm[2]: Still creating... [00m10s elapsed]
google_compute_instance.gcp_vm[1]: Still creating... [00m10s elapsed]
aws_instance.aws_vm[2]: Still creating... [00m20s elapsed]
aws_instance.aws_vm[4]: Still creating... [00m20s elapsed]
aws_instance.aws_vm[1]: Still creating... [00m20s elapsed]
aws_instance.aws_vm[3]: Still creating... [00m20s elapsed]
aws_instance.aws_vm[0]: Still creating... [00m20s elapsed]
google_compute_instance.gcp_vm[3]: Still creating... [00m20s elapsed]
google_compute_instance.gcp_vm[4]: Still creating... [00m20s elapsed]
google_compute_instance.gcp_vm[0]: Still creating... [00m20s elapsed]
google_compute_instance.gcp_vm[1]: Still creating... [00m20s elapsed]
google_compute_instance.gcp_vm[2]: Still creating... [00m20s elapsed]
aws_instance.aws_vm[4]: Creation complete after 21s [id=i-019b9ea1fb6845b86]
aws_instance.aws_vm[2]: Creation complete after 21s [id=i-07f24500ac44a3cb8]
google_compute_instance.gcp_vm[1]: Creation complete after 22s [id=projects/terraform-469922/zones/us-central1-a/instances/gcp-terraform-vm2]
google_compute_instance.gcp_vm[3]: Creation complete after 29s [id=projects/terraform-469922/zones/us-central1-a/instances/gcp-terraform-vm4]
google_compute_instance.gcp_vm[4]: Creation complete after 29s [id=projects/terraform-469922/zones/us-central1-a/instances/gcp-terraform-vm5]
google_compute_instance.gcp_vm[0]: Creation complete after 30s [id=projects/terraform-469922/zones/us-central1-a/instances/gcp-terraform-vm1]
aws_instance.aws_vm[1]: Still creating... [00m34s elapsed]
aws_instance.aws_vm[0]: Still creating... [00m34s elapsed]
aws_instance.aws_vm[3]: Still creating... [00m34s elapsed]
google_compute_instance.gcp_vm[2]: Still creating... [00m33s elapsed]
aws_instance.aws_vm[1]: Creation complete after 35s [id=i-08afd90c97422900c]
aws_instance.aws_vm[0]: Creation complete after 35s [id=i-07e0f2c31094bc508]
google_compute_instance.gcp_vm[2]: Creation complete after 41s [id=projects/terraform-469922/zones/us-central1-a/instances/gcp-terraform-vm3]
aws_instance.aws_vm[3]: Still creating... [00m44s elapsed]
aws_instance.aws_vm[3]: Creation complete after 45s [id=i-0f9daa9b4567cf07d]

Apply complete! Resources: 10 added, 0 changed, 0 destroyed.
```

Figure 7: Terraform Apply

Step 6: Outputs

Outputs in Terraform are return values from your infrastructure that are displayed after running `terraform apply`. They help you quickly view useful information without digging into state files.

Action:

- Define `output` blocks in Terraform to display key values.
- Example: Show `public_ip` of an AWS EC2 instance or `instance_id` of a GCP VM.
- Reference resource attributes (e.g., `aws_instance.myserver.public_ip`).

Reason for Outputs:

- **Quick Results** – Instantly see useful values like IP addresses or DNS names.
- **Reusability** – Outputs can be used as input for other Terraform projects (via remote state).
- **Clarity** – Avoids manually checking the cloud console for resource details.

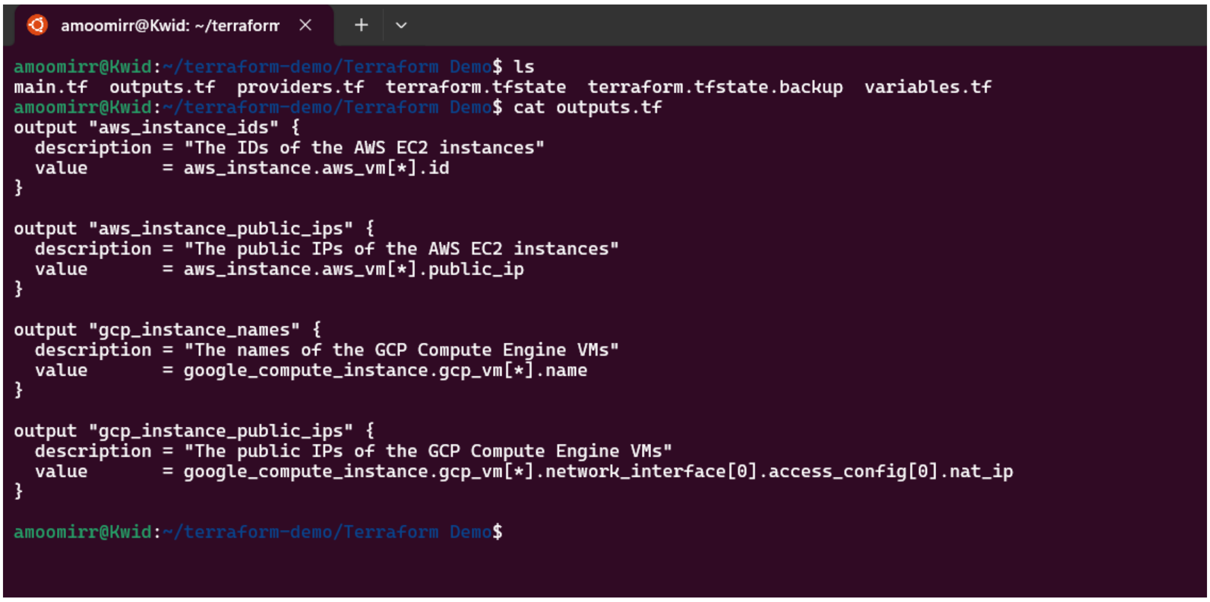

Figure 8: Output.tf

Step 7: Verify Through Cloud Console

Definition: After applying Terraform configuration, it is important to confirm that the resources have been created successfully in the respective cloud platforms.

Action:

- Log in to the AWS Management Console and check the `EC2 Dashboard` to confirm the instance(s) are running.
- Log in to the Google Cloud Console and check the `Compute Engine` section to verify VM instances.

Reason: Verifying through the console provides a visual confirmation that Terraform has correctly provisioned the infrastructure. It also ensures the configurations (like instance type, region, and networking) match what was defined in the Terraform files.

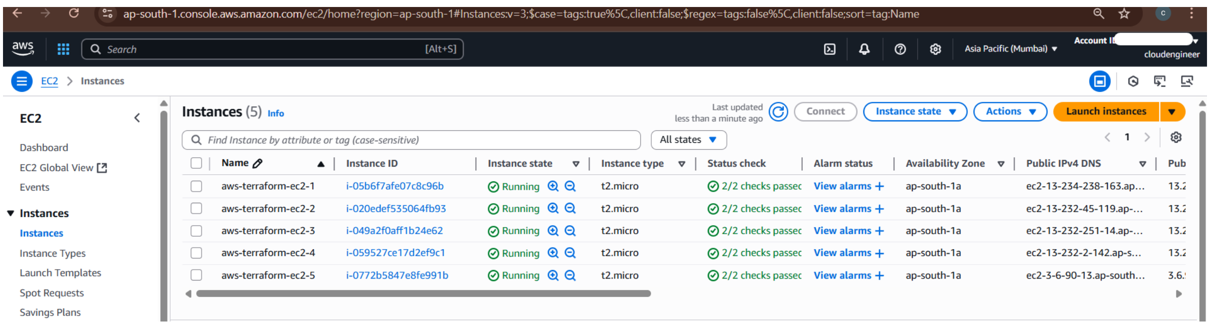

Figure 9: AWS Dashboard

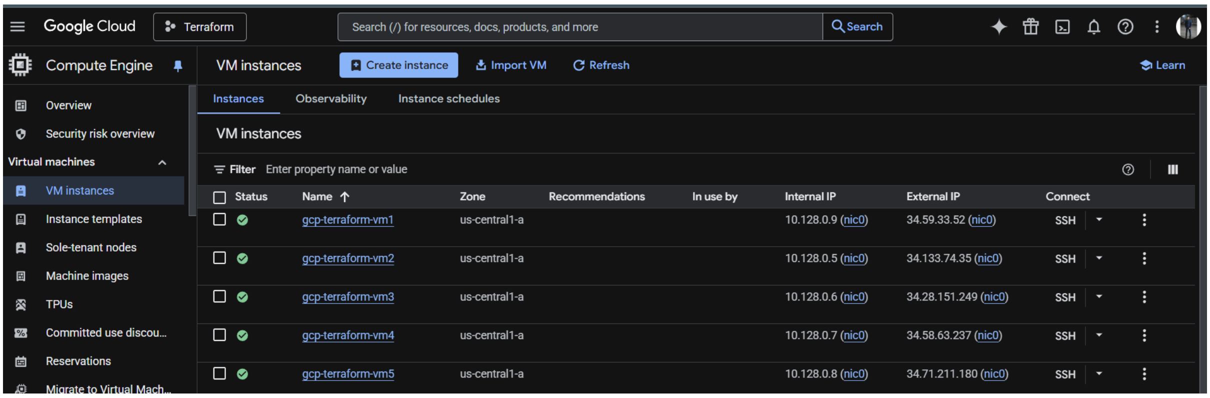

Figure 10: GCP Dashboard

Step 8: Destroying All Resources

Terraform ensures that all resources created by IaC are safely removed. This one-command clean-up prevents unnecessary costs and keeps your cloud environment tidy.

Command

```
terraform destroy -auto-approve
```

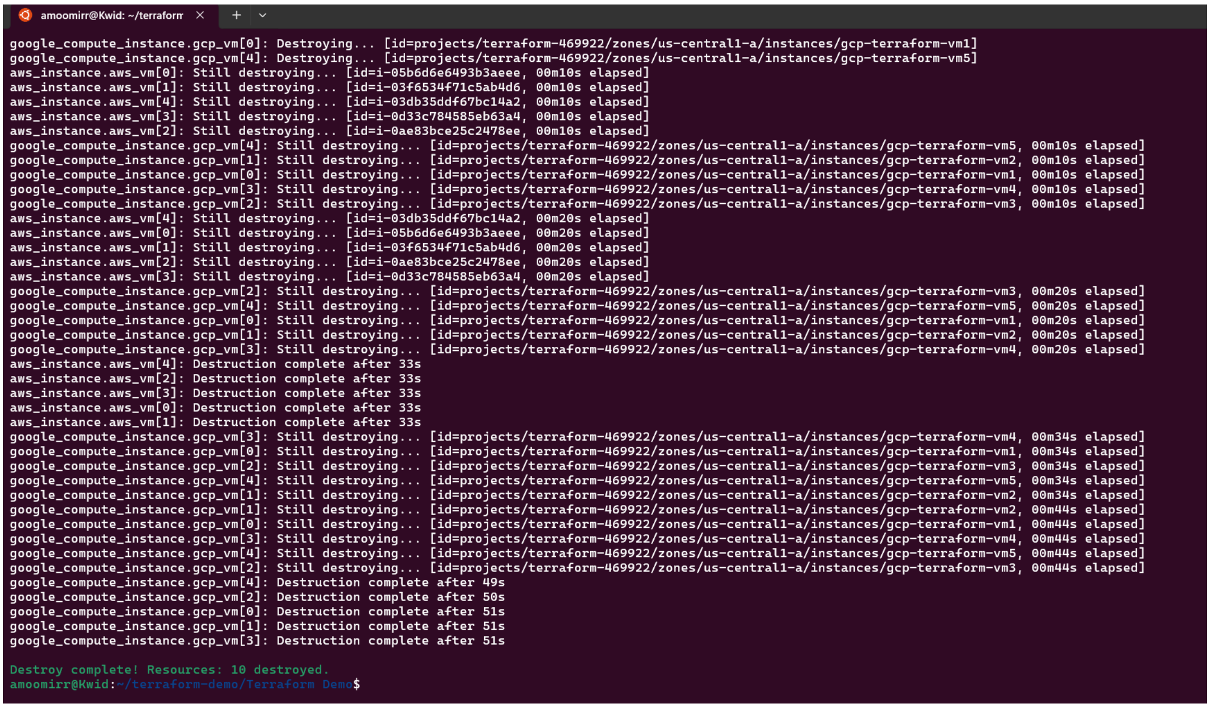

Figure 11: Terraform Destroy

CONCLUSION

This project demonstrated the deployment of virtual machines across two leading cloud platforms — AWS and GCP — using Terraform as the Infrastructure-as-Code (IaC) tool. By structuring the project into variables, providers, resources, outputs, and verification, we followed a clean and modular workflow that highlights industry best practices.

Key takeaways from this project:

- **Automation:** Terraform enabled automated provisioning of multi-cloud infrastructure with minimal manual effort.
- **Reusability:** Using variables and separate configuration files made the setup flexible, reusable, and easy to adapt for different environments.
- **Scalability:** Dynamic scaling with the `count` parameter allowed launching multiple instances in both AWS and GCP efficiently.
- **Transparency:** Outputs provided quick access to important information (like IP addresses), while verification in cloud consoles ensured correctness.
- **Clean Resource Management:** The `terraform destroy` command ensured that all resources were safely and completely removed, preventing unwanted costs.

In conclusion, this project successfully highlighted how Terraform can serve as a powerful tool for managing multi-cloud environments. It not only reduced manual configuration errors but also emphasized the importance of infrastructure automation, portability, and scalability in modern cloud computing.