Experiment

**Aim :** To construct change and destroy AWS resources using terraform

**Theory:**

Infrastructure as Code (IaC) is a modern DevOps practice that involves managing and provisioning computing infrastructure through machine-readable configuration files, rather than through physical hardware configuration or interactive configuration tools. Terraform, developed by HashiCorp, is one of the most widely used IaC tools that enables users to define both low-level (e.g., EC2, networking) and high-level (e.g., DNS, load balancers) components of infrastructure.

Terraform allows users to write configuration files using HashiCorp Configuration Language (HCL) or JSON. These files describe the desired state of the infrastructure. When Terraform is run, it compares the current state of infrastructure with the configuration, and determines the actions needed to align the infrastructure with the declared configuration.

**Key Features of Terraform:**

* **Declarative Syntax**: Users declare *what* resources are needed, and Terraform figures out *how* to create them.
* **Execution Plan**: Before making changes, Terraform shows a preview (using terraform plan) of what it will do.
* **State Management**: Terraform keeps track of infrastructure using a state file, which records what resources it manages.
* **Provisioning and Tearing Down**: With terraform apply, resources are created or modified. With terraform destroy, they are safely deleted.
* **Idempotency**: Running the same configuration multiple times results in the same infrastructure state.
* **Cloud Provider Support**: Terraform supports multiple providers like AWS, Azure, GCP, Kubernetes, etc., enabling multi-cloud strategies.

**How Terraform Works with AWS:**

To use Terraform with AWS, the user must configure credentials—usually through access keys generated by creating an IAM user with programmatic access. Once configured, users write .tf files that specify AWS services like EC2 instances, VPCs, security groups, and more.

The typical workflow includes:

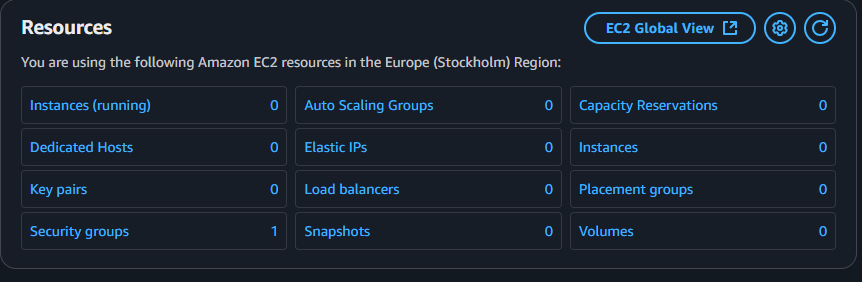
1. **Initialization (terraform init)**: Prepares the working directory and downloads necessary provider plugins.
2. **Planning (terraform plan)**: Shows the actions Terraform will take to achieve the desired state.
3. **Applying (terraform apply)**: Applies the changes and provisions the resources.
4. **Destroying (terraform destroy)**: Removes all infrastructure managed by Terraform when it's no longer needed.

**Advantages of Using Terraform:**

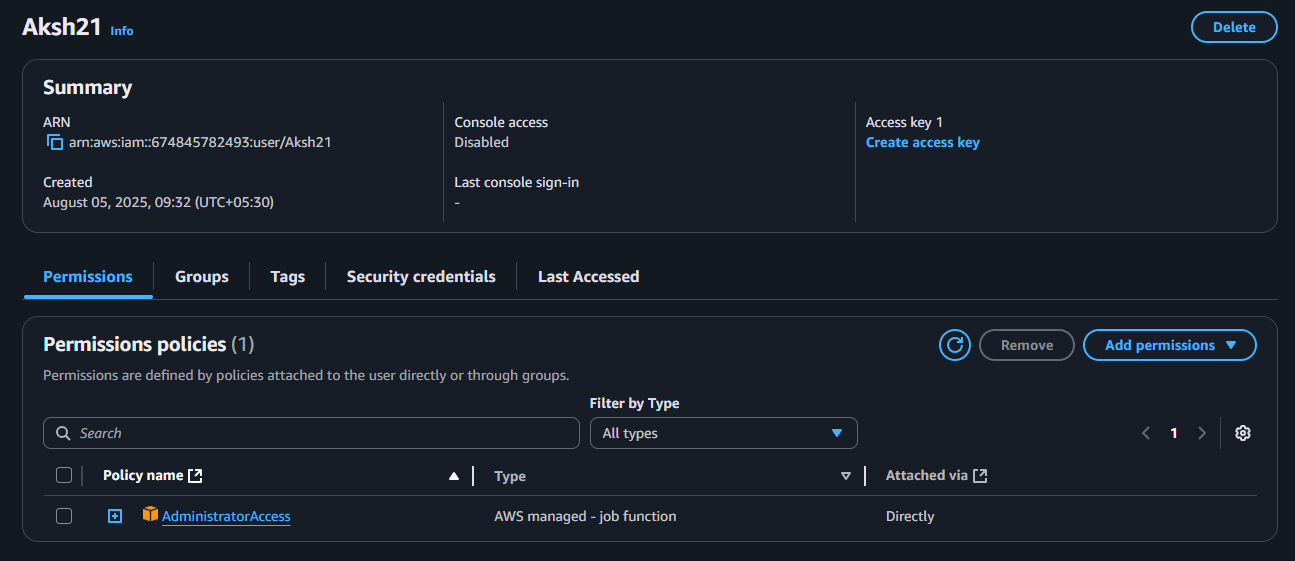
* Automation reduces the risk of human error.
* Reusability of configuration files improves productivity.
* Infrastructure changes are version-controlled like code.
* Enables consistent environments across development, staging, and production.
* Useful in CI/CD pipelines for automated infrastructure deployment.

In this experiment, Terraform is used to automate the deployment and removal of an EC2 instance in AWS. This showcases Terraform's ability to manage cloud infrastructure reliably and efficiently.

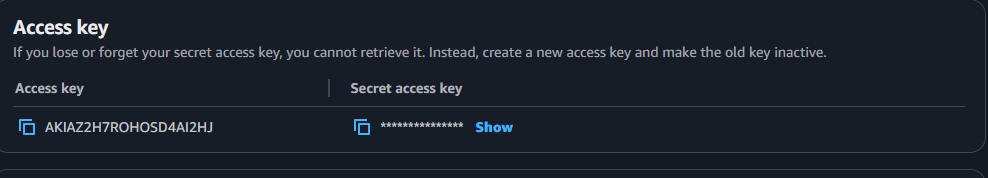
**Step 1 :** Check that no instance is running on EC2



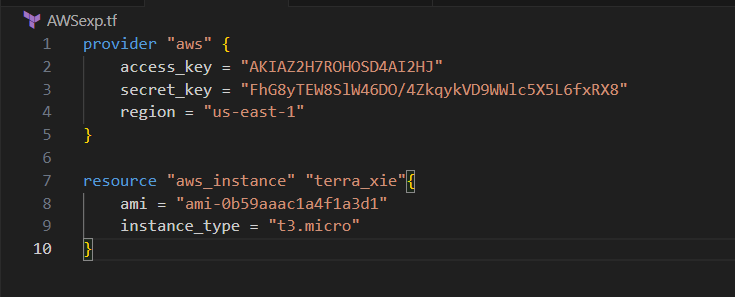
**Step 2 :** Create IAM user with programmatic password and administrator access



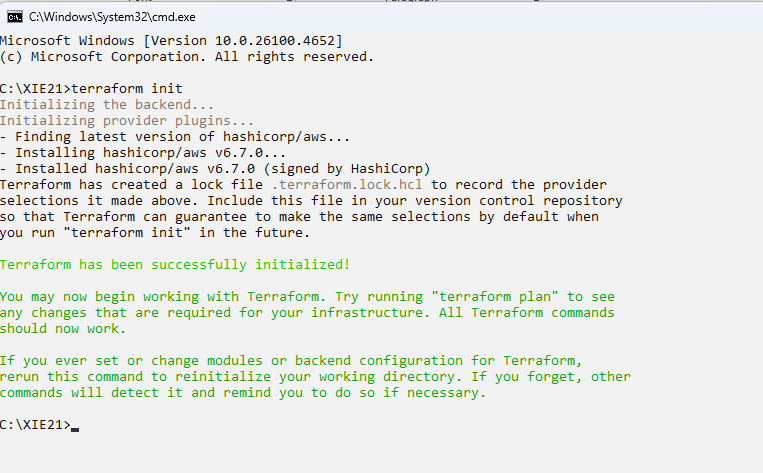
**Step 3 :** Create access key and secret key command line interface



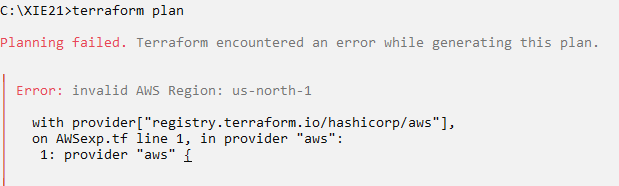
**Step 4 :** Write a terraform program



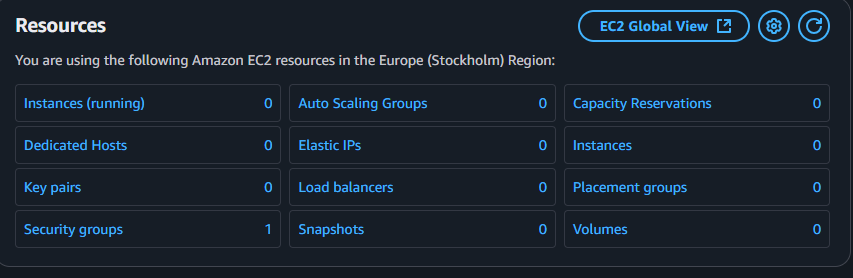
**Step 5 :** Initialize the terraform by command `terraform init`



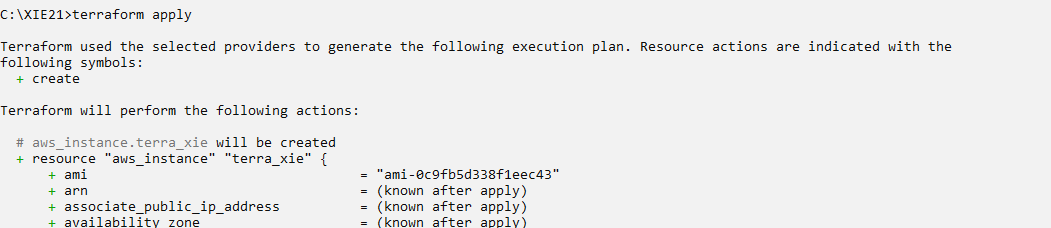
**Step 6 :** Run the command terraform plan

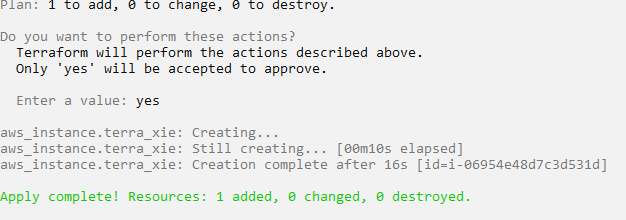


**Step 7 :** Check the instance on EC2 before the apply command

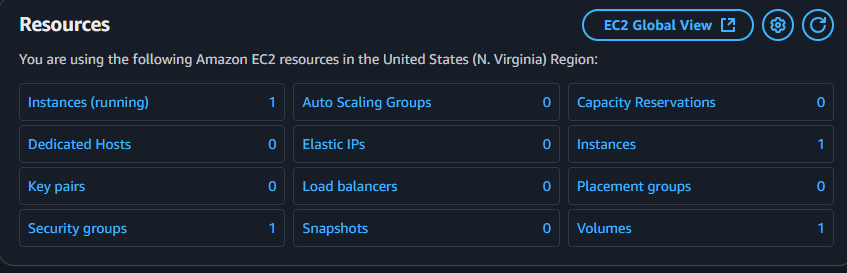


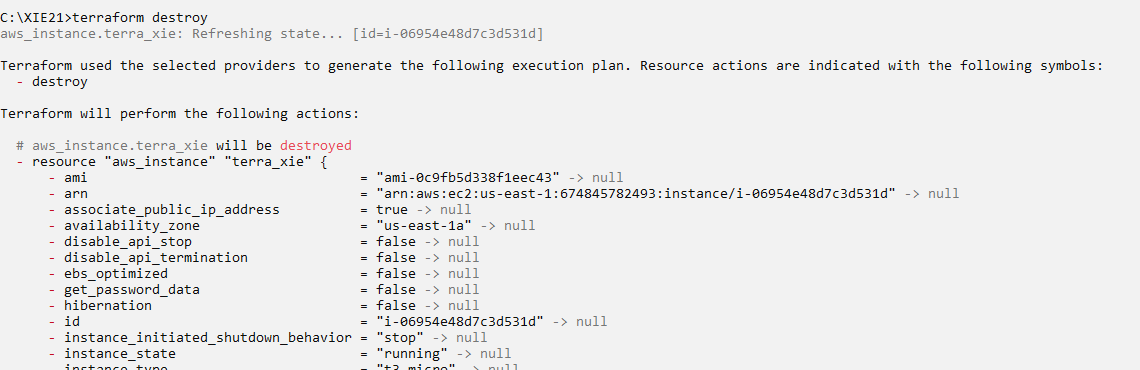
**Step 8 :** Run the command terraform apply

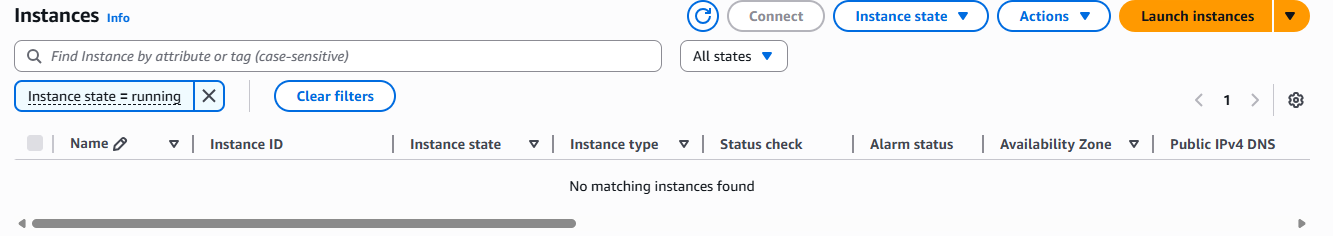




**Step 9:** Check terraform create instance on EC2



**Step 10:** Now Destroy the instance from command prompt by writing the command terraform destroy



**Conclusion:** In this experiment, we leveraged Terraform to create, update, and tear down AWS infrastructure. Using a declarative approach, we successfully deployed an EC2 instance and observed how Terraform applies changes in a controlled and incremental way. When the resources were no longer required, they were cleanly removed through automation. This exercise highlighted Terraform’s effectiveness in simplifying and managing cloud infrastructure workflows.