***What is Terraform Cloud?***

Terraform Cloud is a platform for managing Terraform configurations, state files, and collaborative workflows. It is designed for teams to collaborate on infrastructure provisioning and automation. Terraform Cloud integrates with version control systems (like GitHub, GitLab) and provides automation features like automatic runs, state management, and collaboration tools.

Key Features of Terraform Cloud:

1. Workspaces: A workspace is an environment where Terraform runs and manages infrastructure. It links to a Git repository and runs Terraform commands automatically.
2. Version Control Integration (VCS): Connect your GitHub, GitLab, or Bitbucket repositories to Terraform Cloud to trigger runs whenever changes are pushed.
3. State Management: Terraform Cloud securely stores the state file, which keeps track of the infrastructure being managed. It ensures the state is consistent across users and environments.
4. Variable Management: Manage environment and sensitive variables (like API keys) directly within the platform.

***Step-by-Step Guide to Setting Up Terraform Cloud using version control***

**1. Create a Terraform Cloud Account**

1. **Go to Terraform Cloud**:
   * Navigate to [Terraform Cloud](https://app.terraform.io/signup/account) and click on **Sign Up**.
   * You can sign up using your email or sign up with GitHub or any other OAuth option if you'd prefer.
2. **Create Your Organization**:
   * After signing up, Terraform will prompt you to create an **organization**. Your organization is a container for all your workspaces, users, and other configurations.
   * Choose an appropriate name for your organization.
3. **Verify Your Email**:
   * Terraform Cloud will send you a verification email. Click on the link to verify your email address and activate your account.

**2. Set Up Your Workspace in Terraform Cloud**

1. **Create a Workspace**:
   * In Terraform Cloud, click on **Create a Workspace**.
   * Choose **Version Control (VCS)** if you're using GitHub, GitLab, or Bitbucket for your Terraform code.
   * For now, choose **"Version control"** if you want to manually upload your code and manage the workspace yourself.
2. **Connect Your Git Repository**:
   * Select your version control provider (e.g., GitHub).
   * Authorize Terraform Cloud to access your Git account by signing in.
   * After connecting, you can select your repository that contains your Terraform code.

**3. Set Up Your GitHub Repository for Terraform Code**

1. **Create a GitHub Repository**:
   * Go to [GitHub](https://github.com/) and create a new repository.
   * If you already have a repository for your Terraform configuration, skip this step.
2. **Push Terraform Code to GitHub**:
   * If you have already written your Terraform files (like main.tf, variables.tf, etc.), push them to the GitHub repository you created.

git init

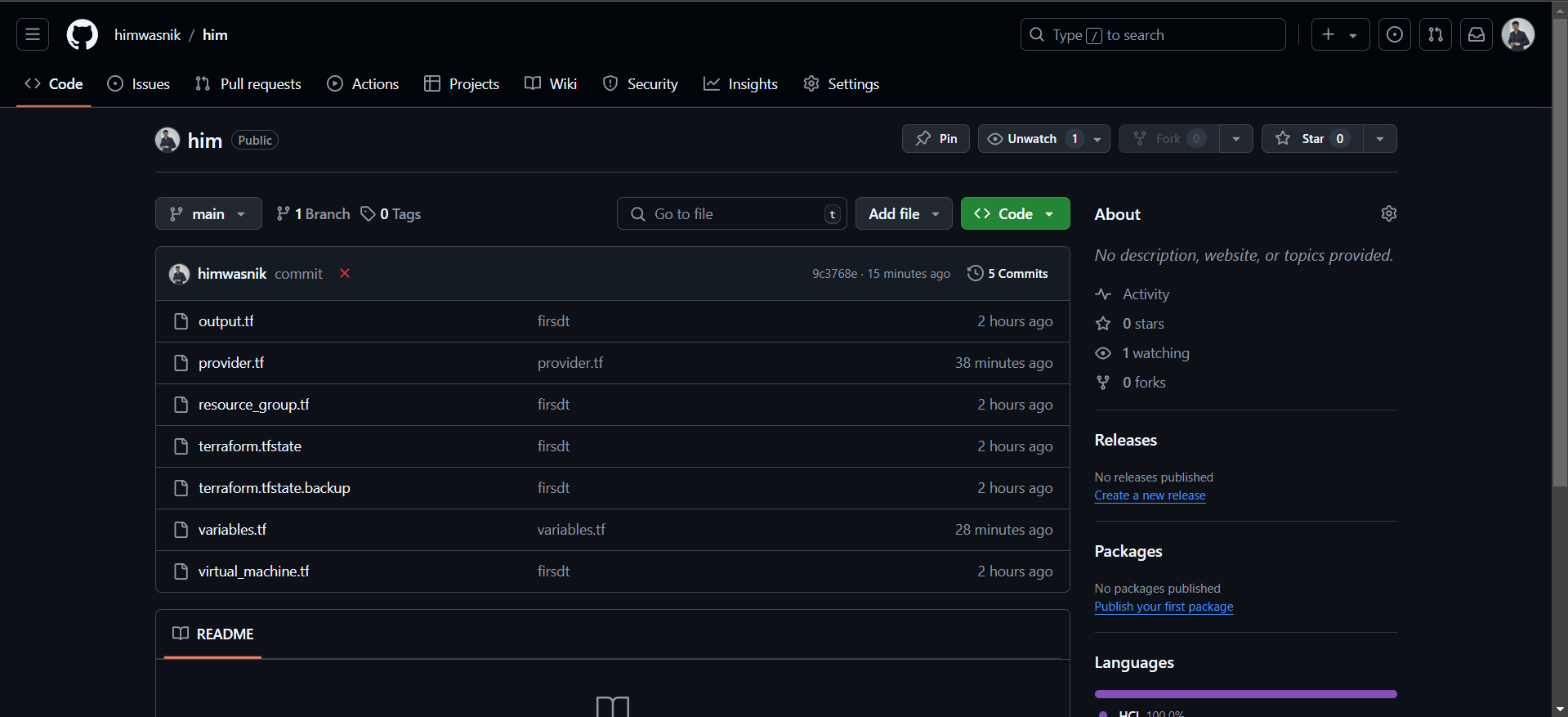
git remote add origin <your-repository-url>

git add .

git commit -m "Initial commit"

git push -u origin main

1. **Verify Your Terraform Code is on GitHub:**
   * Check that your code is visible on GitHub after pushing.



**4. Configure Terraform Cloud Workspace with Git**

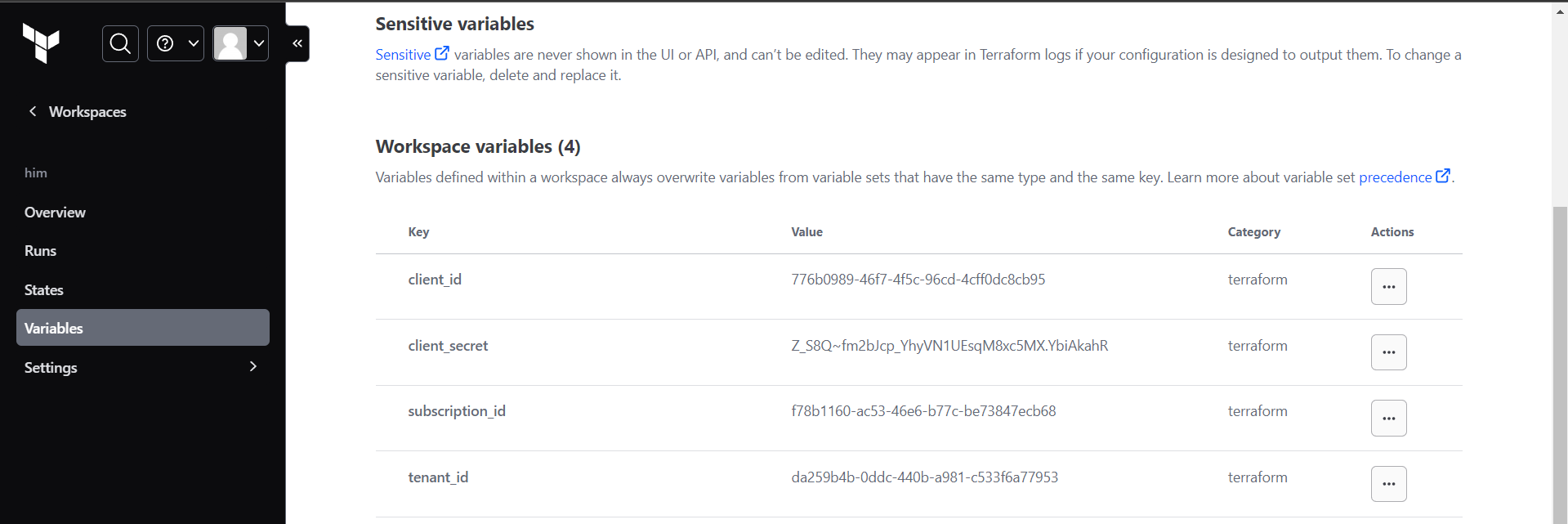
1. **Go Back to Terraform Cloud**:
   * In the Terraform Cloud dashboard, go to **Workspaces**, and click the workspace you created earlier.
2. **Set Up Version Control Integration**:
   * Under the **Version Control** tab in your workspace, select the GitHub repository you pushed your code to.
   * Terraform Cloud will now pull your configuration files from GitHub.

**5. Configure Variables in Terraform Cloud**

1. **Set up Credentials as Terraform Cloud Variables**:
   * Go to **Workspace Settings > Variables**.
   * Add the following variables to authenticate with Azure:
     + client\_id
     + client\_secret
     + tenant\_id
     + subscription\_id

These should match the credentials for your service principal in Azure (as set up earlier). Terraform Cloud will use these values to authenticate to Azure.

1. **Set up terraform Variables**
   * If you need environment variables such as the Azure region, you can define them here too.



**6. Configure and Run Your Terraform Code**

1. **Configure Terraform Cloud to Use Your Azure Provider**:
   * Ensure that your provider.tf file in your repository has the correct setup for Azure using the variables you created in Terraform Cloud.

provider "azurerm" {

  client\_id       = var.client\_id

  client\_secret   = var.client\_secret

  tenant\_id       = var.tenant\_id

  subscription\_id = var.subscription\_id

  features {}

}

**Create an Azure Service Principal (SP) for Authentication**

The first step is to create an Azure Service Principal that Terraform will use to authenticate with Azure.

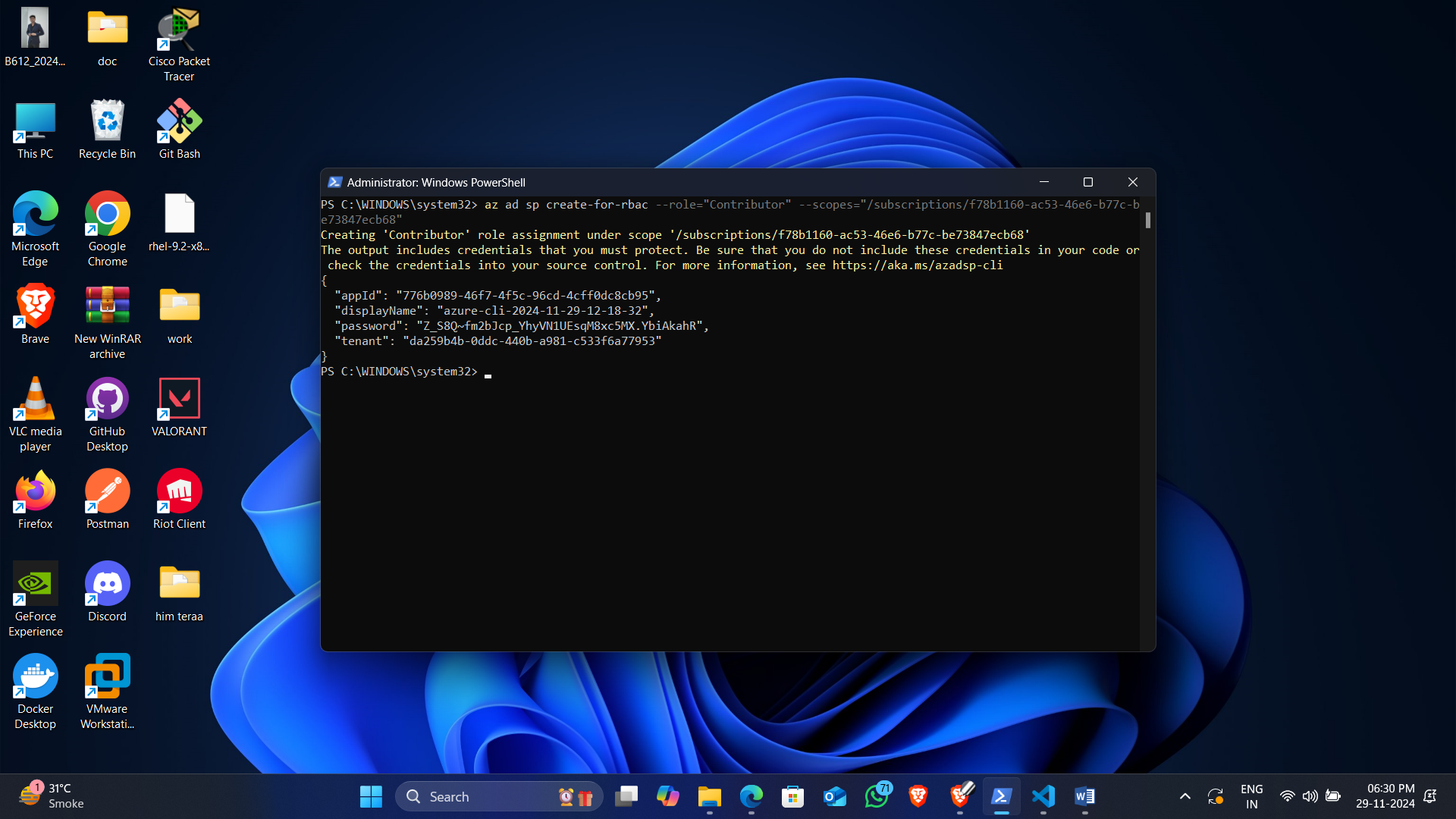
Command:

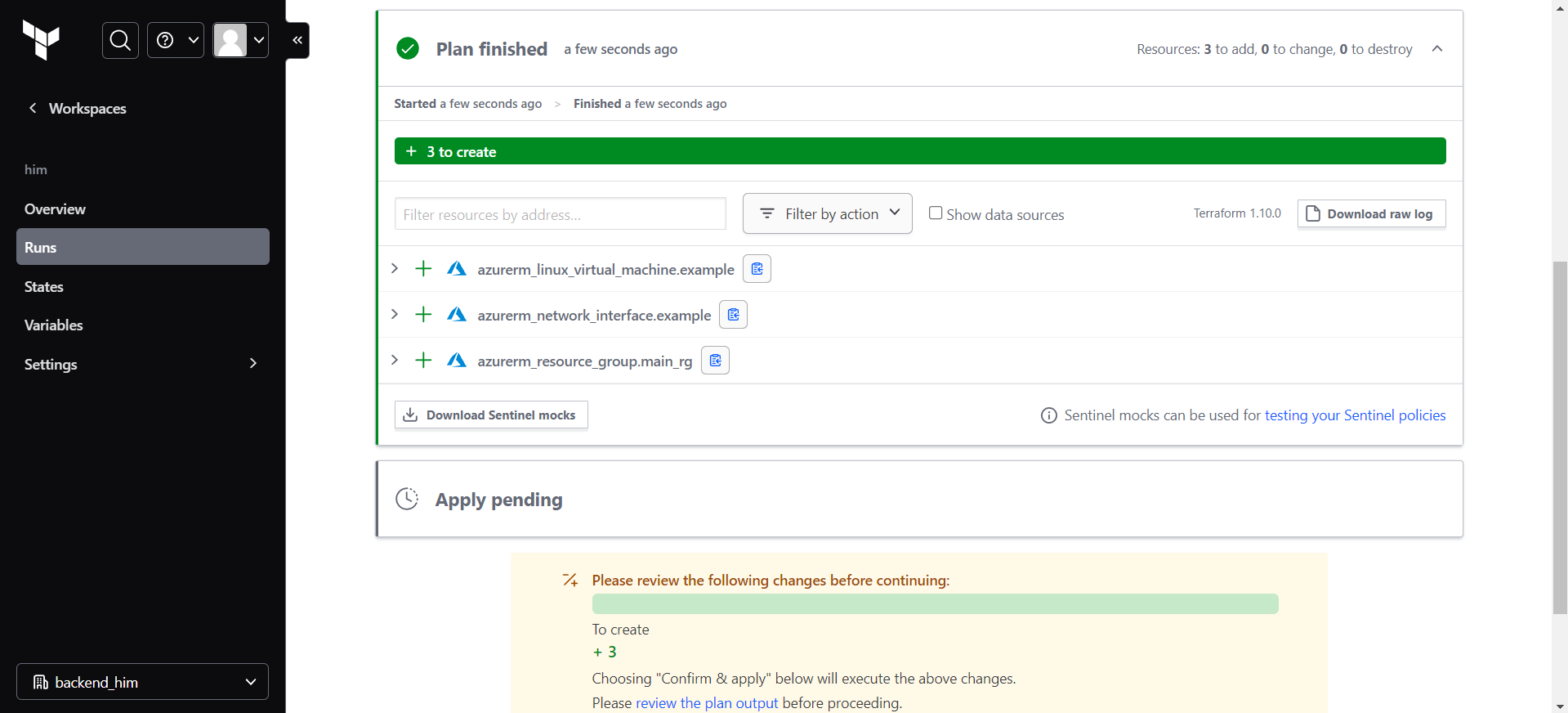
bash

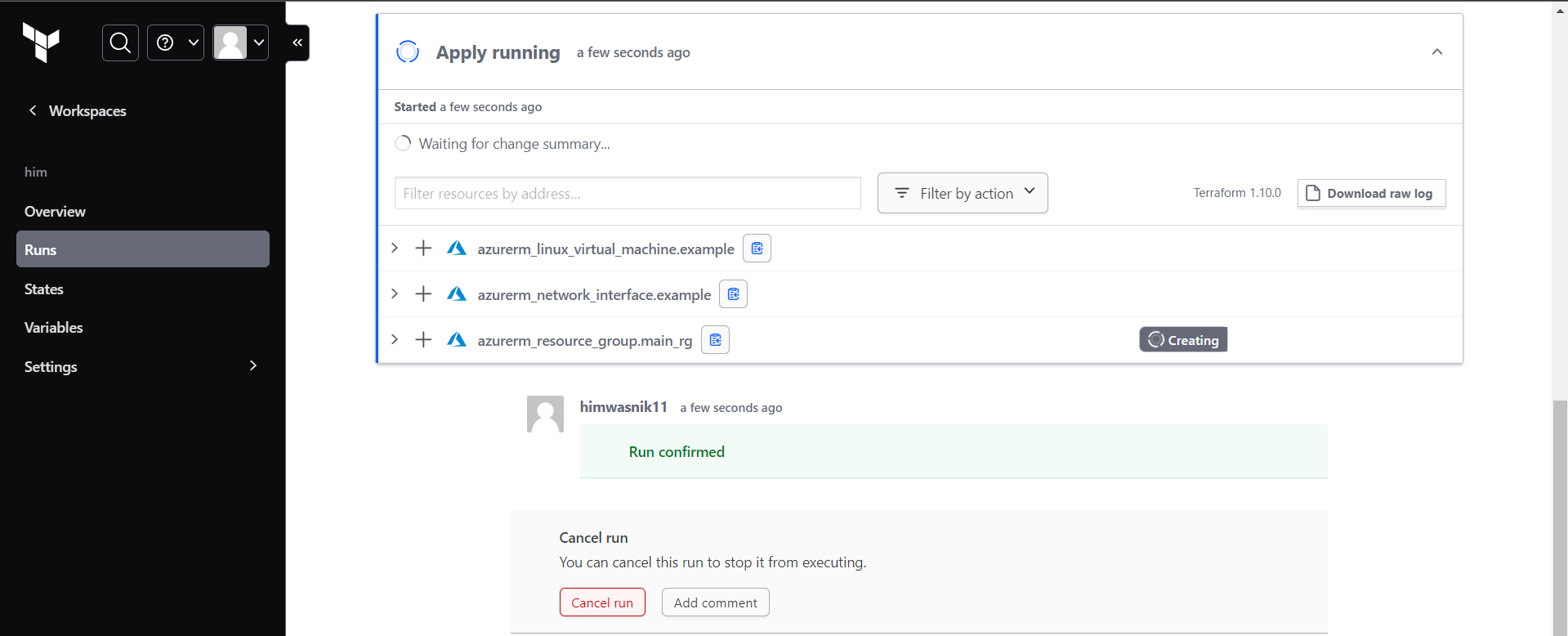
az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/f78b1160-ac53-46e6-b77c-be73847ecb68"

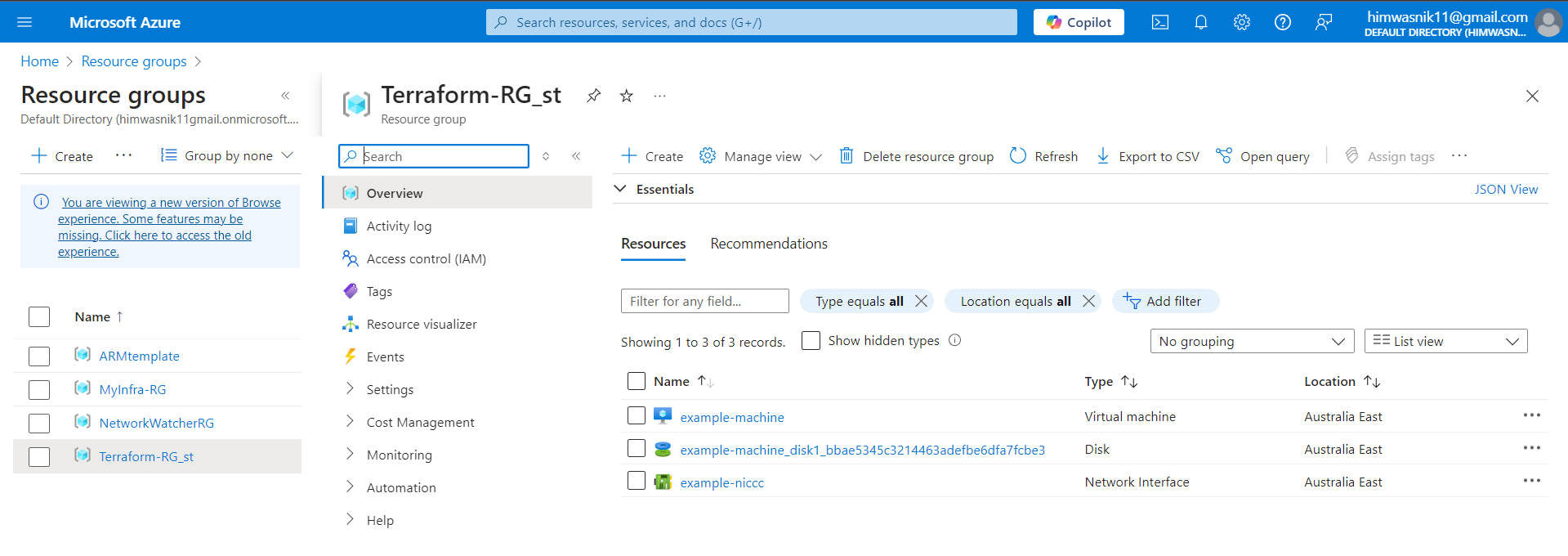
Replace {subscription-id} with your actual Azure subscription ID. This command creates a service principal and assigns the Contributor role to it within your subscription.

Output: This command will output several credentials, including:









Video link:-

<https://www.youtube.com/watch?v=NBC3mu75pEU>

***Terraform Cloud Using CLI***

Terraform Cloud is a platform that manages Terraform runs in a consistent and scalable way. When you use the CLI workflow with Terraform Cloud, the configuration files reside locally, but Terraform Cloud handles the backend operations, such as storing state, locking state files, and managing runs.

Here’s an overview of using Terraform Cloud with the CLI workflow:

**1. Set Up Terraform Cloud**

**Prerequisites:**

* An account on Terraform Cloud.
* A Terraform Cloud organization (e.g., backend\_him).
* An API token from Terraform Cloud.

**2. Configure Terraform Cloud Backend**

You need to specify the remote backend in your terraform block in your configuration file. The backend configuration allows Terraform to know where to store state files and manage operations.

Example :-

terraform {

  backend "remote" {

    hostname     = "app.terraform.io"

    organization = "backend\_him"

    workspaces {

      name = "terraformcli"

    }

  }

}

provider "azurerm" {

  client\_id       = var.client\_id

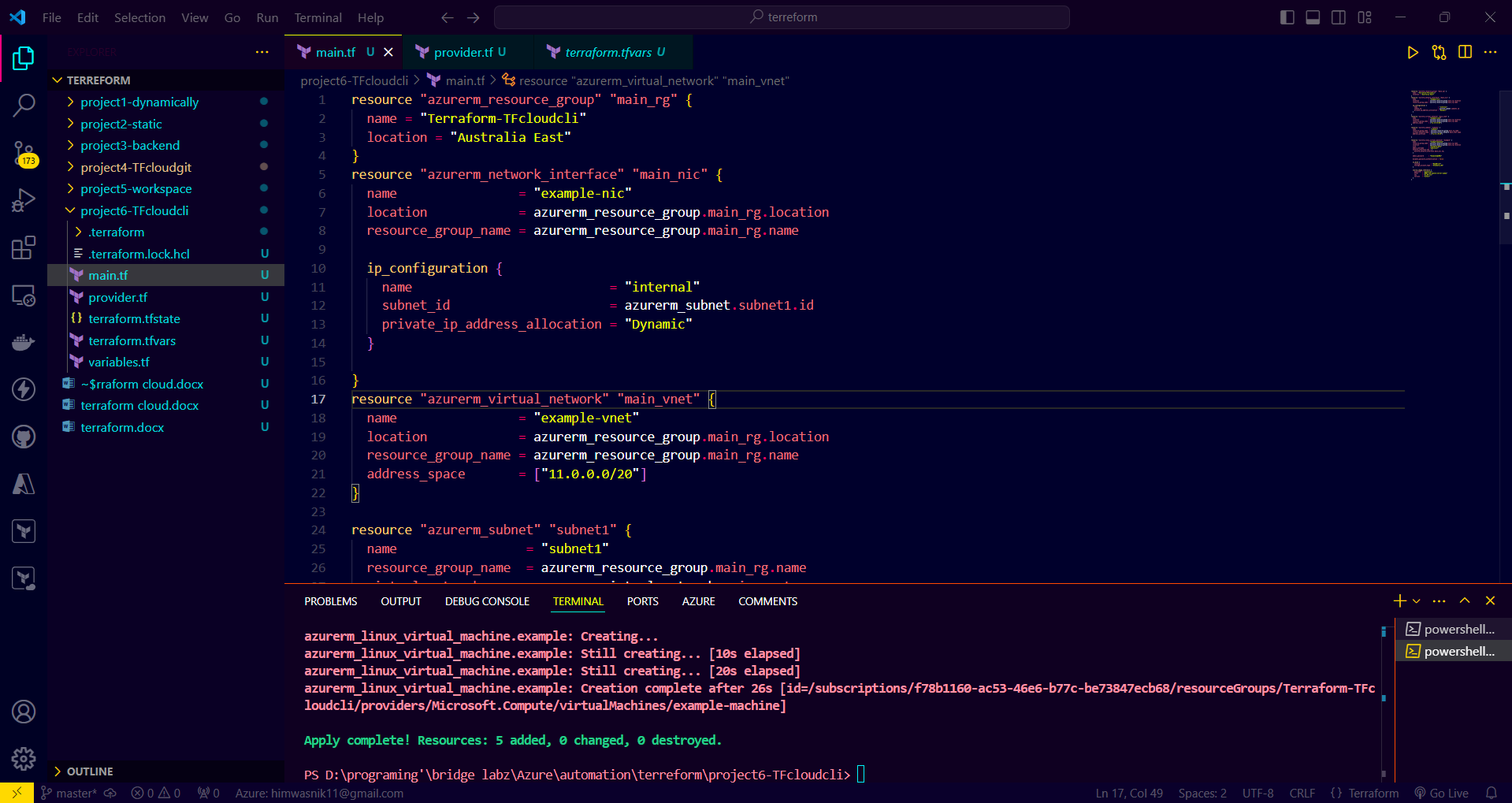
  client\_secret   = var.client\_secret

  tenant\_id       = var.tenant\_id

  subscription\_id = var.subscription\_id

  features {}

}



**3. Log In to Terraform Cloud(optinal)**

Use the terraform login command to authenticate to Terraform Cloud:

bash

terraform login

This command:

* Opens a browser to authenticate with your Terraform Cloud account.
* Generates a local API token that Terraform CLI uses to interact with Terraform Cloud.

Alternatively, you can manually create a token in the Terraform Cloud UI and save it to ~/.terraform.d/credentials.tfrc.json if needed.

**4. Initialize Terraform**

Run the terraform init command to initialize the backend configuration and download provider plugins.

bash

terraform init

* This connects your configuration to the specified Terraform Cloud workspace.
* If the workspace doesn’t exist, Terraform Cloud will create it automatically.

**5. Plan Changes**

Run terraform plan to see the proposed changes.

bash

terraform plan

This command:

* Generates an execution plan.
* Displays the actions Terraform will take to achieve the desired state.

Terraform Cloud will handle locking the state file during the operation.

**6. Apply Changes**

Run terraform apply to execute the changes in your configuration.

bash

terraform apply

This command:

* Uploads the plan to Terraform Cloud.
* Applies the changes through the Terraform Cloud backend.

**7. Benefits of Using CLI with Terraform Cloud**

1. **Centralized State Management**: Terraform Cloud securely stores state files remotely, avoiding the need to manage them locally or on shared storage.
2. **State Locking**: Automatic locking prevents simultaneous operations on the same state.
3. **Collaboration**: Multiple team members can access and manage the same Terraform configuration and workspace.
4. **Scalable Runs**: Terraform Cloud manages runs, providing consistent execution environments.
5. **Detailed Logs**: All plans and applies are recorded in Terraform Cloud for audit purposes.

