Terraform

What is Terraform

**Terraform** is a popular infrastructure-as-code tool that allows you to Automate the provisioning and management of infrastructure resources.

**1. Install Terraform on Windows**

**Step 1: Download Terraform**

Go to the Terraform download page.

Select the appropriate version for Windows. You’ll typically download a .zip file.

**Step 2: Extract the Zip File**

Locate the downloaded .zip file.

Right-click on the file and select Extract All....

Choose a directory to extract the files to. For example, you might extract it to C:\terraform.

**Step 3: Add Terraform to the System Path**

**Open Environment Variables:**

Right-click on This PC or Computer on the Desktop or in File Explorer.

Select Properties.

Click on Advanced system settings on the left.

Click on Environment Variables in the System Properties window.

**Edit the Path Variable:**

In the Environment Variables window, find and select the Path variable in the System variables section.

Click Edit.

Click New and add the path where you extracted Terraform, e.g., C:\terraform.

**Confirm the Changes:**

Click OK to close each window.

**Step 4: Verify Installation**

Open a new Command Prompt (cmd) or PowerShell window.

Type terraform --version and press Enter.

You should see the Terraform version information if the installation was successful.

to use AWS CLI credentials to authenticate Terraform instead of hardcoding AWS keys in your configuration files, you can do so by configuring your AWS CLI properly. Here’s how to set it up:

**1. Configure AWS CLI**

**Step 1: Install AWS CLI**

**Download and Install AWS CLI:**

Go to the [AWS CLI installation page](https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html) and download the installer for Windows.

Run the installer and follow the prompts to complete the installation.

**Verify Installation:**

Open Command Prompt or PowerShell or GitBash.

Run aws --version to verify that the AWS CLI is installed correctly.

**Step 2: Configure AWS CLI**

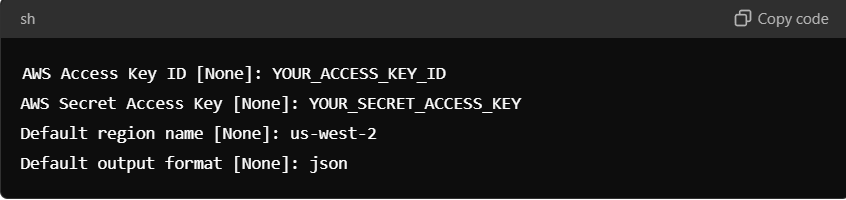
**Open Command Prompt or PowerShell.**

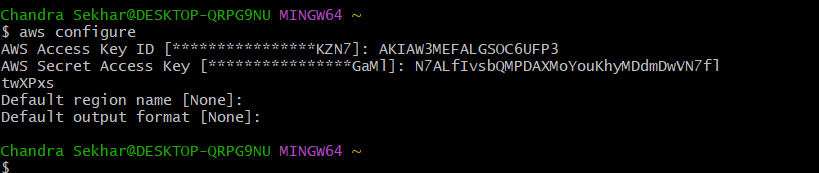
**Run the AWS configure command:**



**Provide Your AWS Credentials:**

You will be prompted to enter your AWS Access Key ID, AWS Secret Access Key, Default region name, and Default output format.





Replace YOUR\_ACCESS\_KEY\_ID and YOUR\_SECRET\_ACCESS\_KEY with your AWS credentials.

2. Configure Terraform to Use AWS CLI Credentials

Terraform will automatically use the AWS credentials configured by the AWS CLI without requiring explicit credentials in the Terraform configuration file.  
  
  
Key Terraform Components:

**Provider**

**Description:** Providers are plugins that Terraform uses to interact with cloud providers and other services. They handle the API calls and manage the lifecycle of resources.

**Example:** AWS, Azure, Google Cloud, etc.

**Configuration Example:**

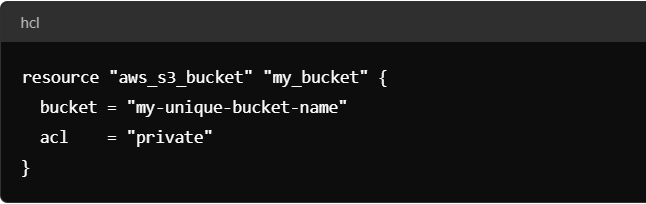


**Resource**

**Description:** Resources represent individual components of your infrastructure, such as virtual machines, databases, and networking components. They define the desired state of those components.

**Example:** An AWS S3 bucket, an Azure VM, or a Google Cloud SQL instance.

**Configuration Example:**



**Data Source**

**Description:** Data sources allow you to fetch and use data from outside of Terraform. They can be used to get information from existing resources that Terraform doesn’t manage.

**Example:** Fetching information about an existing AWS VPC.

**Configuration Example:**

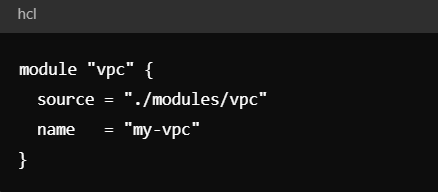
****

**Module**

**Description:** Modules are containers for multiple resources that are used together. They allow you to encapsulate and reuse configurations.

**Example:** A module to set up a complete VPC with subnets, routes, and security groups.

**Configuration Example:**

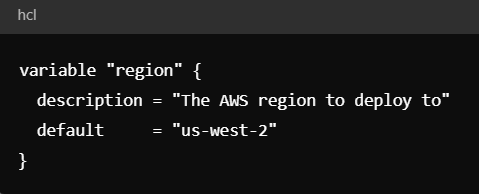


**Variable**

**Description:** Variables allow you to parameterize your Terraform configurations, making them more flexible and reusable.

**Example:** Defining region and instance type as variables.

**Configuration Example:**

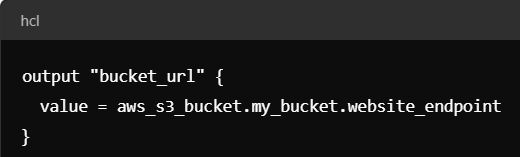


**Output**

**Description:** Outputs are used to display information about your resources after they are created. They can also be used to pass data between modules.

**Example:** Outputting the URL of an S3 bucket.

**Configuration Example:**



**State File**

**Description:** The state file (terraform.tfstate) keeps track of the resources that Terraform manages and their current state. It is essential for mapping your configurations to real-world resources.

**Location:** Typically stored in the working directory or remotely in a backend like AWS S3.

**Backend**

**1. Choose a Backend Type**

Terraform supports several backend types like S3 (AWS), GCS (Google Cloud), Azure, Consul, and many others. Choose a backend based on your infrastructure provider and needs.

**2. Prepare the Remote Storage**

Before configuring Terraform, you need to set up the remote storage where Terraform's state files will be kept.

**For AWS S3**

Create an S3 bucket.

Enable versioning on the bucket to keep a history of state files.

Optionally, set up an S3 bucket policy to restrict access.

**For Google Cloud Storage**

Create a GCS bucket.

Optionally, configure IAM permissions for access control.

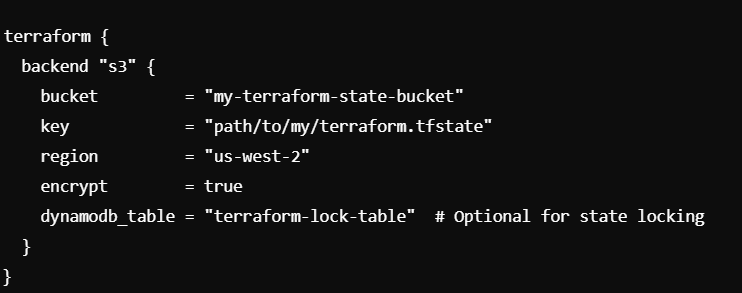
**For Azure Blob Storage**

Create a storage account and a blob container.

**3. Configure the Backend in Terraform**

In your Terraform configuration, you need to define the backend block in one of your .tf files. This block specifies the type of backend and any required configuration options.

**Example for AWS S3**

**Configuration Example:** 

**Create a DynamoDB Table:**

**Log in to the AWS Management Console**:

Navigate to the DynamoDB service.

**Create a New Table**:

Click on "Create Table".

**Table Name**: Choose a descriptive name, like terraform-lock-table.

**Partition Key**: Use a string key named LockID.

**Configure Settings**:

**Read/Write Capacity Mode**: Choose either "Provisioned" or "On-demand" based on your expected usage. "On-demand" is simpler and scales automatically, but might be more expensive.

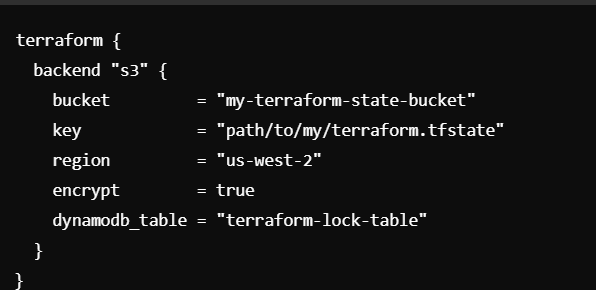
**Tags** (Optional): Add any tags you find necessary for identifying or managing the table.

**Create the Table**:

Click "Create" to finalize the setup.

**2. Configure the S3 Backend with DynamoDB Lock Table**

In your Terraform configuration, add the dynamodb\_table parameter in the backend "s3" block. This tells Terraform to use the specified DynamoDB table for state locking.



**Workspace**

**Description:** Workspaces allow you to manage different versions of your state. The default workspace is default, but additional workspaces can be created for environments like staging or production.

**Commands:** terraform workspace new, terraform workspace select, terraform workspace list.

**Workflow Overview**

**Write Configuration:**

Define your infrastructure using HCL (HashiCorp Configuration Language) in .tf files.

**Initialize:**

Run terraform init to initialize the directory containing Terraform configuration files. This downloads the provider plugins and prepares the working directory.

**Plan:**

Run terraform plan to create an execution plan. This step shows what actions Terraform will take to reach the desired state described in your configuration files.

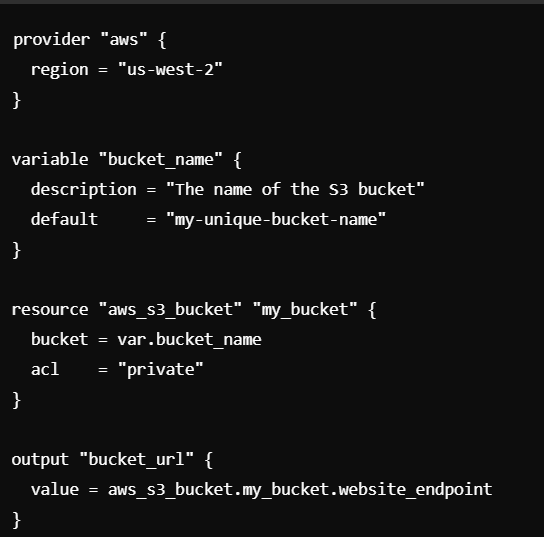
**Apply:**

Run terraform apply to execute the actions proposed in the plan. This creates, updates, or deletes resources as necessary to achieve the desired state.

**Destroy:**

Run terraform destroy to remove all resources defined in your configuration.

**Example Full Configuration:**

****