****This study guide is based on the video lesson available on TrainerTests.com****

Provisioning Infrastructure as Code with Terraform Study Guide

Introduction to Infrastructure as Code (IaC)

Infrastructure as Code (IaC) is a modern approach to managing and provisioning IT infrastructure through code. This allows infrastructure to be automatically created, modified, and destroyed using scripts or configuration files, rather than relying on manual processes. IaC ensures that infrastructure is repeatable, scalable, and easy to manage, making it a core component of DevOps practices.

Terraform, developed by HashiCorp, is one of the most popular IaC tools. It allows users to define infrastructure in a declarative configuration language (HCL - HashiCorp Configuration Language) and automate the creation of cloud resources across multiple cloud providers, including AWS, Azure, and Google Cloud.

In this chapter, we will focus on setting up Terraform, creating a basic configuration file, and using it to provision and manage infrastructure in AWS. By the end of this chapter, you will understand how to:

- Install Terraform.
- Create and configure an AWS environment for Terraform.
- Write Terraform configuration files.
- Apply Terraform to create infrastructure and manage state changes.

Installing Terraform

Terraform is available for multiple operating systems, including Windows, Linux, and macOS. Below are the steps to install Terraform for each platform.

Installing Terraform on Windows

1. **Download Terraform**:
   - Go to the official Terraform website and download the latest version for Windows.
   - You will get a `.exe` file, which is the Terraform executable.
2. **Move Terraform to a Directory**:
   - Move the `terraform.exe` file to a directory that is included in your system's `PATH` environment variable. For instance, you could move it to the `C:\Users\YourUsername\Terraform` directory.
3. **Add to PATH (Optional)**:

   - If the directory is not in your system's `PATH`, you will need to add it manually. This allows you to run Terraform commands from anywhere in the command prompt.
   - To add it to `PATH`:
     1. Open **Control Panel → System → Advanced system settings**.
     2. Under the **System Properties** window, click on **Environment Variables**.
     3. Under **System Variables**, select `Path` and click **Edit**.
     4. Add the path where `terraform.exe` is located.
4. **Verify Installation**:
   - Open the command prompt and run the following command to verify Terraform is installed correctly:

     ```
     bash
     Copy code
     terraform --version
     ```

   - This should display the installed version of Terraform, confirming it is ready to use.

Installing Terraform on Linux

1. **Download the Binary**:
   - Run the following command to download the latest version of Terraform:

     ```
     bash
     Copy code
     wget
     https://releases.hashicorp.com/terraform/X.X.X/terraform_X.X.X_linux_amd64.zi
     p
     ```

     Replace `X.X.X` with the current version number.

2. **Unzip the File**:
   - Unzip the file using:

     ```
     bash
     Copy code
     unzip terraform_X.X.X_linux_amd64.zip
     ```

3. **Move the Binary to a Directory**:
   - Move the unzipped Terraform binary to a directory in your `PATH`:

     ```
     bash
     Copy code
     sudo mv terraform /usr/local/bin/
     ```

4. **Verify Installation**:
   - Check if Terraform is installed correctly by running:

     ```
     bash
     Copy code
     terraform --version
     ```

Installing Terraform on macOS

1. **Install Using Homebrew**:

- The easiest way to install Terraform on macOS is using Homebrew:

  ```
  bash
  Copy code
  brew install terraform
  ```

2. **Verify Installation**:
   - Verify the installation by running:

     ```
     bash
     Copy code
     terraform --version
     ```

Setting Up AWS for Terraform

Before you can use Terraform to manage AWS resources, you need to create an IAM user with the appropriate permissions and configure your AWS credentials in your local environment.

Creating an IAM User in AWS

1. **Log in to the AWS Management Console**:
   - Navigate to the **IAM** (Identity and Access Management) service.
2. **Create a New User**:
   - Click on **Users** in the left-hand menu, then click **Add user**.
   - Name the user (e.g., `terraform-demo`) and enable **Programmatic access** (to generate access keys).
3. **Attach Permissions**:
   - In the **Permissions** section, attach the **AdministratorAccess** policy to the user, giving it full control over AWS services.
4. **Create an Access Key**:
   - After the user is created, go to the **Security Credentials** tab and click **Create access key**.
   - Save both the **Access Key ID** and **Secret Access Key**. These credentials will be used by Terraform to authenticate with AWS.

Configuring AWS CLI

To provide Terraform with the credentials to interact with AWS, you need to configure the AWS CLI.

1. **Install AWS CLI**:
   - If you haven't already installed the AWS CLI, you can follow the installation steps covered in earlier chapters.
2. **Run `aws configure`**:
   - Open your terminal or command prompt and configure AWS credentials using:

     ```
     bash
     Copy code
     aws configure
     ```

   - Enter the **Access Key ID** and **Secret Access Key** when prompted, along with your default region (e.g., `us-east-1`).

Writing Terraform Configuration Files

Terraform uses configuration files written in HCL (HashiCorp Configuration Language) to define the infrastructure that you want to create.

Basic Structure of a Terraform Configuration File

1. **Create a New Directory**:
   - First, create a directory to store your Terraform configuration files:

     ```
     bash
     Copy code
     mkdir terraform-lab
     cd terraform-lab
     ```

2. **Create a Configuration File**:
   - Use a text editor like `notepad` (on Windows) or `nano` (on Linux/macOS) to create a new file called `main.tf`.
3. **Define the Configuration**:
   - A basic Terraform configuration file might look like this:

     ```
     hcl
     Copy code
     provider "aws" {
       region = "us-east-1"
     }

     resource "aws_instance" "my_ec2" {
       ami           = "ami-0c55b159cbfafe1f0"
       instance_type = "t2.micro"
     }
     ```

   - **Provider**: Specifies which cloud provider you're working with (AWS in this case).
   - **Resource**: Defines what resource you're creating (`aws_instance` is used for EC2).
4. **Save the File**.

Initializing and Applying Terraform

Once the configuration file is written, you need to initialize Terraform and apply the configuration.

Initialize Terraform

1. **Run `terraform init`**:
   - Inside the directory where your `main.tf` file is located, run:

     ```
     bash
     Copy code
     terraform init
     ```

   - This command initializes the directory by downloading the necessary provider plugins (in this case, AWS) and sets up the working environment for Terraform.

Plan and Apply the Configuration

1. **Run `terraform plan`**:
   - Before applying the changes, use `terraform plan` to see what Terraform will create:

```
bash
Copy code
terraform plan
```

   - This command provides an execution plan, detailing the resources that will be created, modified, or destroyed.
2. **Run `terraform apply`**:
   - To create the infrastructure, run:

     ```
     bash
     Copy code
     terraform apply
     ```

   - Terraform will ask for confirmation. Type **yes** to proceed.
3. **Verify in AWS**:
   - Log in to the AWS Management Console and navigate to **EC2** to confirm that the EC2 instance has been created.

Managing State with Terraform

Terraform uses a **state file** (`terraform.tfstate`) to keep track of the current infrastructure managed by it. The state file is critical for tracking changes, making it possible to manage infrastructure over time.

Changing the Configuration

1. **Modify `main.tf`**:
   - Update the instance type in the configuration file:

     ```
     hcl
     Copy code
     instance_type = "t2.small"
     ```

2. **Run `terraform apply`**:
   - Apply the changes by running `terraform apply` again. Terraform will compare the state file with the new configuration and update the EC2 instance accordingly.

Destroying Infrastructure with Terraform

Once you are done with your infrastructure, you may want to delete it to avoid incurring costs.

1. **Run `terraform destroy`**:
   - To remove all resources created by Terraform, use:

     ```
     bash
     Copy code
     terraform destroy
     ```

   - Terraform will prompt for confirmation before deleting the resources.

Cleaning Up AWS Resources

After using Terraform, remember to clean up sensitive information like access keys and IAM users in AWS.

1. **Delete the Access Key**:
    - Navigate to IAM in the AWS Management Console, find the user, and delete the access key you created for Terraform.
2. **Remove the User**:
    - If the user is no longer needed, you can also delete the IAM user.

Conclusion

In this chapter, we explored the basics of using Terraform to provision infrastructure in AWS. Terraform simplifies the process of creating, managing, and destroying cloud resources through its declarative approach to infrastructure. By learning how to write configuration files, initialize Terraform, apply configurations, and manage infrastructure state, you can automate infrastructure tasks and improve the efficiency of your cloud operations. Terraform’s flexibility across cloud providers makes it a powerful tool for managing infrastructure as code.