

Documentation on Deploying a React Application on EC2 Instance with Terraform, Domain and Let's Encrypt.

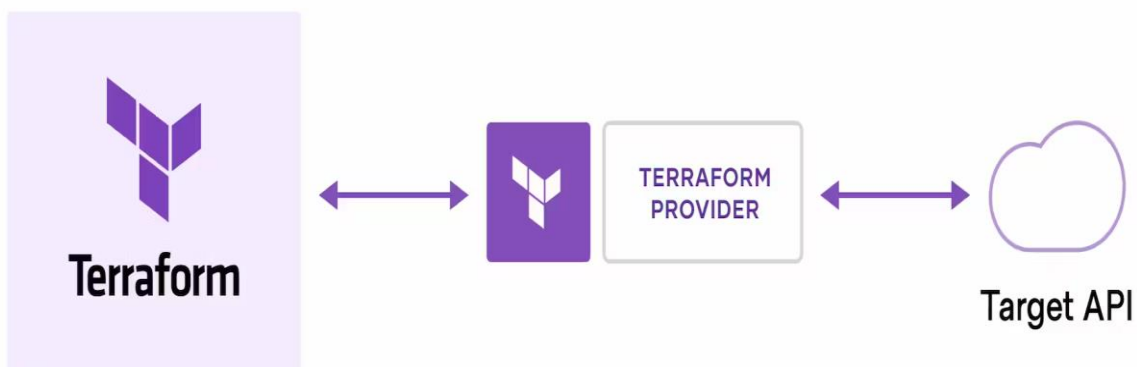
What is Terraform?

Terraform is an infrastructure as code tool that lets you build, change, and version cloud and on-prem resources safely and efficiently.

HashiCorp Terraform is an infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share. You can then use a consistent workflow to provision and manage all of your infrastructure throughout its lifecycle. Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.

How does Terraform work?

Terraform creates and manages resources on cloud platforms and other services through their application programming interfaces. Providers enable Terraform to work with virtually any platform or service with an accessible API.

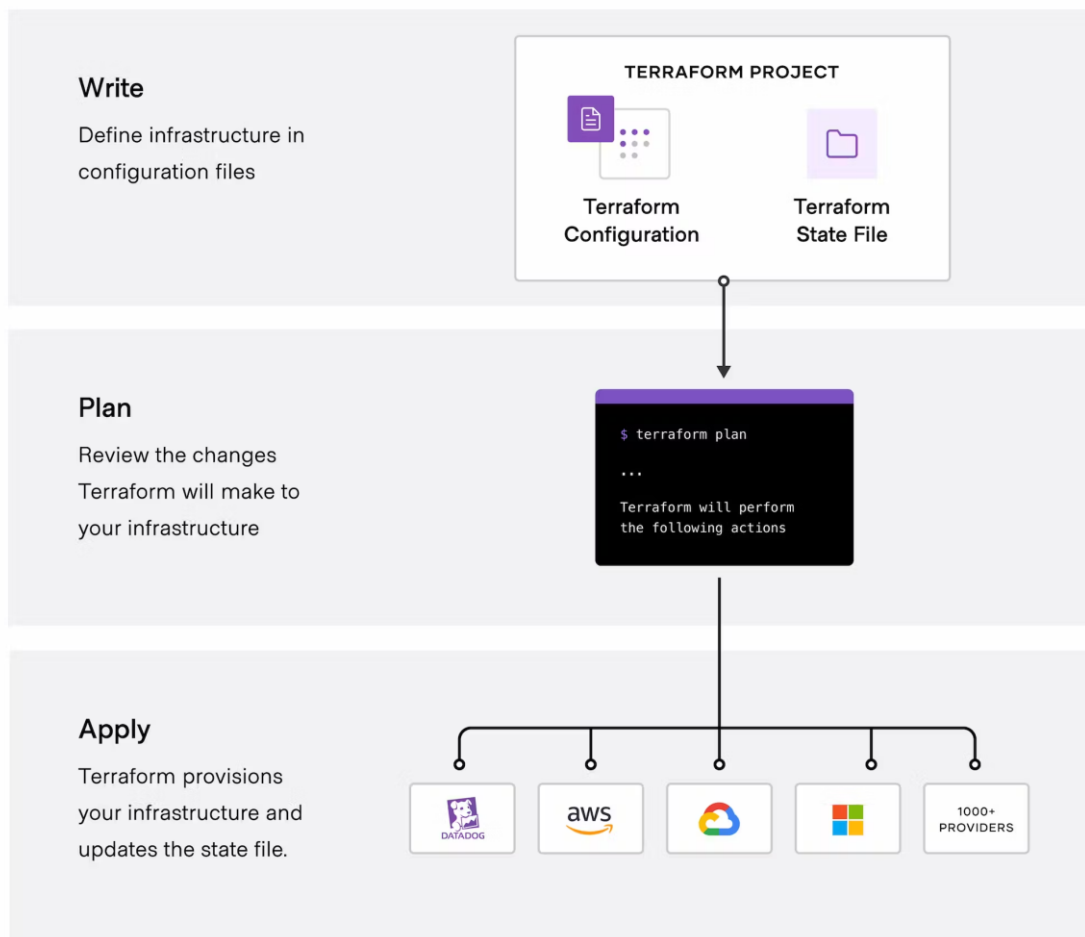


The core Terraform workflow consists of three stages:

Write: You define resources, which may be across multiple cloud providers and services. For example, you might create a configuration to deploy an application on virtual machines in a Virtual Private Cloud (VPC) network with security groups and a load balancer.

Plan: Terraform creates an execution plan describing the infrastructure it will create, update, or destroy based on the existing infrastructure and your configuration.

Apply: On approval, Terraform performs the proposed operations in the correct order, respecting any resource dependencies. For example, if you update the properties of a VPC and change the number of virtual machines in that VPC, Terraform will recreate the VPC before scaling the virtual machine.



In Terraform, modules and templates are both essential concepts that help you organize and reuse your infrastr

ucture code.

Modules in Terraform:

What are Modules?

Modules in Terraform are reusable, self-contained packages of Terraform configurations that represent a specific set of resources or a piece of infrastructure. They allow you to abstract and encapsulate complex parts of your infrastructure code, making it more maintainable and modular.

Why we use Modules:

Templates in Terraform usually refer to templates for generating configuration files dynamically. For example, you can use template files (often in HashiCorp Configuration Language - HCL) to generate variable values or configurations based on dynamic or external data.

In Terraform, you can create modules for different parts of your infrastructure, like a database, a web server, or a network setup. Then, you can use these modules to build your infrastructure repeatedly and consistently across projects. Modules are reusable, maintainable, and a great way to keep your code organized.

Here's a basic example of using a module in Terraform:

```
module "web_server" {  
  source = "example.com/modules/webserver"  
  instance_count = 2  
  instance_type = "t2.micro"  
}
```

In this example:

- module "web_server" is declaring a module named "web_server."
- source specifies where the module's code is located. It can be a local path or a remote Git repository.
- Other arguments can be passed to configure the module, like the number of instances or instance types.

To create Terraform template for deploying a React application on an EC2 instance, setting up a domain, and securing it with Let's Encrypt, it's essential to follow a step-by-step approach.

Step1: Setup

Before starting, ensure that you have the following requirements in order:

Terraform Installed: Ensure that Terraform is installed on your local machine.

AWS Account and AWS CLI Configuration: You should have an AWS account and the AWS Command Line Interface (CLI) installed and configured with the necessary credentials.

Domain Name and DNS Provider: Obtain a domain name that you want to use for your application. Ensure that you have access to a DNS provider like AWS Route 53 or another provider where you can manage DNS records.

Step2 Directory Structure:

Create a directory for your Terraform project and set up the initial directory structure:

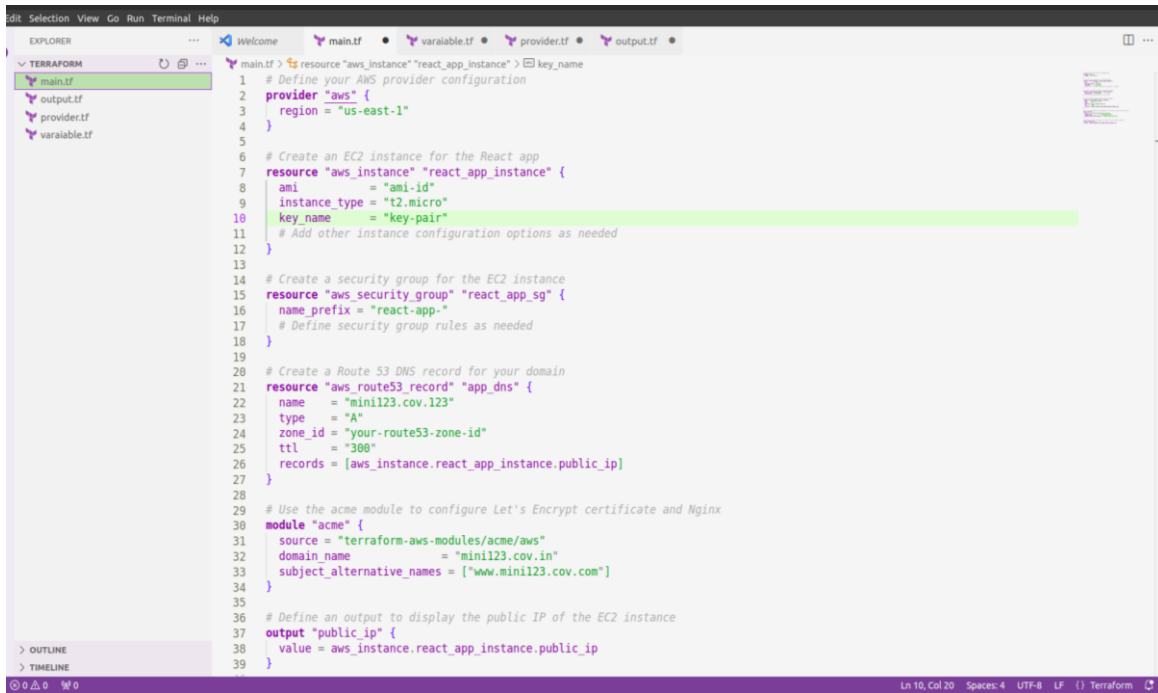
Step 3: Terraform Configuration Files:

Inside your project directory, create the following Terraform configuration files:

- **main.tf** : Define your infrastructure resources and configurations.
- **variable.tf** : Declare variables for customizing your deployment.
- **output.tf** : Define outputs to display essential information.
- **terraform.tfvars** : Store sensitive or variable-specific values like AWS access keys.

Step 4: Write Terraform Configuration

- In main.tf, define your AWS resources such as the EC2 instance, security groups, IAM roles, Route 53 DNS records, and let's Encrypt certificate configuration.

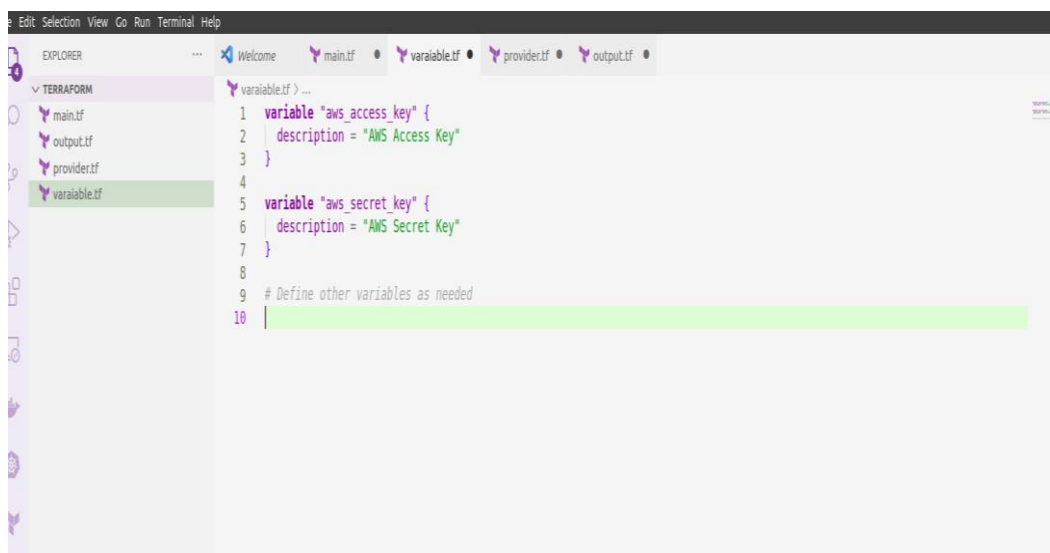


```
1 # Define your AWS provider configuration
2 provider "aws" {
3   region = "us-east-1"
4 }
5
6 # Create an EC2 instance for the React app
7 resource "aws_instance" "react_app_instance" {
8   ami           = "ami-id"
9   instance_type = "t2.micro"
10  key_name       = "key-pair"
11  # Add other instance configuration options as needed
12 }
13
14 # Create a security group for the EC2 instance
15 resource "aws_security_group" "react_app_sg" {
16   name_prefix = "react-app-"
17   # Define security group rules as needed
18 }
19
20 # Create a Route 53 DNS record for your domain
21 resource "aws_route53_record" "app_dns" {
22   name    = "mini123.cov.123"
23   type    = "A"
24   zone_id = "your-route53-zone-id"
25   ttl     = "300"
26   records = [aws_instance.react_app_instance.public_ip]
27 }
28
29 # Use the acme module to configure Let's Encrypt certificate and Nginx
30 module "acme" {
31   source = "terraform-aws-modules/acme/aws"
32   domain_name = "mini123.cov.in"
33   subject_alternative_names = ["www.mini123.cov.com"]
34 }
35
36 # Define an output to display the public IP of the EC2 instance
37 output "public_ip" {
38   value = aws_instance.react_app_instance.public_ip
39 }
```

Step 5:

Variable Declaration

In variable.tf declare variables that allow you to customize your infrastructure deployment. Variables make your code more flexible and reusable:



```
1 variable "aws_access_key" {
2   description = "AWS Access Key"
3 }
4
5 variable "aws_secret_key" {
6   description = "AWS Secret Key"
7 }
8
9 # Define other variables as needed
10
```

Step 6: Initialize Terraform

Run the following command to initialize your Terraform project and download the necessary providers and modules:

```
terraform init
```

Step 7: Apply the Configuration

Use the following command to apply your Terraform configuration and create the infrastructure:

```
terraform apply
```

Step 8: Configure Your React App

SSH into the EC2 instance and configure your React app. You'll need to:

- Install Node.js and npm.
- Clone your React app repository.
- Build and serve your app using a web server (e.g., Nginx).
- Configure Nginx to serve your React app.
-

Step 9: Secure Your EC2 Instance

Configure security groups to allow only necessary inbound and outbound traffic.

Step 10: Set Up Let's Encrypt

Once app is running on the EC2 instance, set up Let's Encrypt for SSL. This involves:

- Installing Certbot on your EC2 instance.
- Running Certbot to obtain and configure SSL certificates for your domain.

SSL (Secure Sockets Layer) : In Simple terms, SSL Certificate make sure that our online communication is private and Secure.

Step 11: Update Your Nginx Configuration

Update your Nginx configuration to use the Let's Encrypt SSL certificates.

Step 12: Test Your Deployment

Ensure your React app is accessible over HTTPS with your domain.