# Project Overview: Automating Static Website Hosting on AWS with Terraform

This project involves setting up a fully automated pipeline for hosting a static website on **AWS Cloud** using **Terraform**, an Infrastructure-as-Code (IaC) tool. The objective is to simplify and standardize the deployment of a static website while leveraging AWS's scalable and cost-effective services. By automating the infrastructure setup, you reduce manual intervention, enhance repeatability, and ensure version-controlled configuration.

## Key Components of the Project

### 1. Static Website Hosting on AWS

We will use **Amazon S3** for hosting the static content (HTML, CSS, JavaScript). S3 provides a highly available and scalable storage solution that supports static website hosting. The content will be accessible via an S3 bucket configured for public access or through a CloudFront distribution for enhanced performance and security.

### 2. Terraform for Automation

Terraform is used to define the infrastructure as code, enabling:

- Consistency across environments.
- Version-controlled infrastructure changes.
- Easy collaboration through modular configurations.

### 3. AWS Services

The project will involve the following AWS components:

- **Amazon S3**: Stores and serves the static website files.
- **AWS CloudFront** (Optional): Distributes the content globally via a Content Delivery Network (CDN) for faster access and added security.
- **Amazon Route 53** (Optional): Manages DNS records if you need a custom domain for the website.
- **IAM Roles and Policies**: Secures access to the resources.
- **ACM (AWS Certificate Manager)**: Manages SSL/TLS certificates for secure HTTPS traffic.

### 4. Terraform Configuration

The Terraform configuration files will:

- Define AWS resources (S3 bucket, CloudFront distribution, IAM roles).
- Set up outputs to expose key resource information (e.g., S3 bucket name, CloudFront URL).
- Use modules and variables for reusability and simplicity.

- Include backend configuration (e.g., using S3 and DynamoDB for Terraform state management).

## Project Workflow

### Step 1: Prerequisites

1. Set up a Terraform environment (install Terraform CLI).
2. Configure AWS CLI with credentials and permissions.
3. Prepare static website files to be uploaded.

### Step 2: Terraform Code Development

1. **Create Terraform Modules**:
   - Module for S3 bucket creation and configuration.
   - Module for CloudFront distribution (if required).
   - Module for Route 53 DNS setup (if needed).
2. **Define Variables**:
   - Define variables for environment-specific configurations (e.g., bucket name, region).
3. **Write Terraform Configuration Files**:
   - Use `.tf` files to define the resources and outputs.
   - Configure the provider (`aws` block) and backend storage for the Terraform state.

### Step 3: Deploy the Infrastructure

1. Initialize Terraform:

   ```
   bash
   Copy code
   terraform init
   ```

2. Validate the configuration:

   ```
   bash
   Copy code
   terraform validate
   ```

3. Plan the infrastructure:

   ```
   bash
   Copy code
   terraform plan
   ```

4. Apply the changes:

   ```
   bash
   Copy code
   terraform apply
   ```

**Step 4: Website Deployment**

1. Upload static files to the S3 bucket using the AWS CLI or Terraform's `aws_s3_bucket_object` resource.
2. Verify the website is accessible via the S3 endpoint or CloudFront URL.

**Step 5: Secure and Optimize**

1. Configure HTTPS using an SSL certificate.
2. Apply access restrictions using bucket policies or origin access identity (OAI) for CloudFront.